Web server performance, tuning and caching
Course team
Developer: Jenny Lim, Consultant
Designer: Chris Baker, OUHK
Coordinator: Dr Li Tak Sing, OUHK
Member: Dr Andrew Lui Kwok Fai, OUHK

External Course Assessor
Prof. Mingshu Li, Institute of Software, Chinese Academy of Sciences

Production
ETPU Publishing Team

Copyright acknowledgements
The following copyright materials are reproduced under the terms of a licence issued by the Hong Kong Reprographic Rights Licensing Society. No further copying is permitted.

Figure 6.1.

Copyright © The Open University of Hong Kong, 2004.

All rights reserved.
No part of this material may be reproduced in any form by any means without permission in writing from the President, The Open University of Hong Kong.

The Open University of Hong Kong
30 Good Shepherd Street
Ho Man Tin, Kowloon
Hong Kong
Contents

Overview 1
Objectives 2
Introduction 3
Performance factors 4
  Client platform 5
  Physical network 5
  Web protocols 6
  Server platform 8
  Dynamic content 9
  User behaviour 10
Measuring performance 12
  Criteria 12
  Log file analysis 14
  Benchmarks 16
Improving performance 23
  Server hardware 24
  Network type and speed 26
  Web server software 26
  Improving the download time of Web content 28
Mirroring documents 34
  The pros and cons of mirroring 35
  Load balancing using round-robin DNS 36
Caching 39
  The benefits of caching 39
  How it works 40
  The role of HTTP 51
  Locating a cache 54
  Cooperative proxy caching 54
  Caching issues 55
Summary 60
Suggested answers to self-tests 61
References 66
Overview

Over the past decade, the World Wide Web has experienced phenomenal growth in terms of the volume of content and the numbers of users accessing this content. This has dramatically increased the need for reliability, scalability and high performance in online information servers.

For companies engaged in electronic business, the Web has become an indispensable tool for communicating and transacting with suppliers, partners and customers. Long waiting times and the unavailability of a company’s website can have a significant impact on its revenues, market share and ultimately, reputation.

Usability studies have shown that users can only keep their attention focused for at most ten seconds when waiting for a response (Nielsen 1997). Many e-business sites try to follow the ‘eight-second rule’, a widely held belief that webpages must be delivered to your users in eight seconds or less, or you risk losing your audience’s interest (Menasce and Almeida 2000). The questions now are: How can information providers achieve this level of performance? How can they even tell if their websites are performing well or poorly?

Due to the many components that are involved in satisfying a Web request, the answers to these questions are not so straightforward. The main components that must be monitored and analysed are: Web servers, Web clients, networks and the protocols used for communication. In this unit, we discuss how each of these components can contribute to website response time. We also look at performance factors which are beyond the information provider’s control, such as customer behaviour and user access patterns on a site.

In addition, we also look at ways to measure a Web server’s operational performance, both in the field and in controlled experiments. Log file analysis can help information providers understand how well their website is actually performing, while benchmark tests are especially useful for determining the performance limitations within a system and for planning future capacity. We also discuss the factors that must be considered when choosing the hardware and operating system requirements for a Web server.

After identifying the potential performance bottlenecks in a Web server system, we explore some common optimization and tuning techniques to remove the bottlenecks in these problem areas.

Caching proxy servers are frequently used to speed up performance by watching requests from clients and serving whatever requests the proxy servers can from their own local store of previously requested webpages. This unit ends with a discussion of how caching works and whether it is beneficial to Web server performance.

It should take you five weeks (or about 35 to 40 hours) to complete this unit. Please plan your time carefully.
Objectives

1. Describe how Web server components, process models and protocols can affect performance.

2. Explain the criteria that can be applied and the metrics that can be gathered when measuring Web server performance.

3. Outline the process and software tools that can be used to assess performance.


5. Discuss the concept of Web caching and explain how it can be implemented on a server, client, and proxy.

6. Analyse the benefits and limitations of Web caching: consistency issues, effectiveness and costs.
Introduction

Have you ever requested a webpage, waited a minute for it to download, and then cancelled your request because the website was taking too long to respond?

What could have been the cause of the problem?

Was a section of the network down? Had the disk drive on the Web server machine failed? Or was the Web server software simply overloaded with other connection requests that delayed your request? Or, maybe the computing platform’s input-output subsystem was clogged? Perhaps the computing platform was ill-suited to handling a large volume of network traffic, and it was really designed to act as a server for CPU-intensive applications?

This unit aims to guide you through the maze of possibilities and helps you to determine where to start looking for the underlying causes of a breakdown in Web server performance. We will begin the unit by looking at some general issues related to examining and measuring the performance of Web servers.
Performance factors

It is extremely difficult to get an exact measurement of a Web server’s performance because a server is a complex system built on top of existing services and technologies, all of which function more or less independently of each other. And yet, all these different components contribute to the site’s overall performance.

Latency (also called ‘round-trip time’) is the time that a user spends waiting for the page to begin showing up in the Web browser after the request is sent out. Each step on the network from client to server and from server back to client (figure 6.1) contributes to the latency of an HTTP operation.

How then do we begin to measure this complex and decentralized system? We’ll start by understanding how these different components can affect Web response times. Next, we’ll look at the processes, benchmarks and tools that can be used to measure Web server performance.
Client platform

You have already had a detailed look at the browser’s functionality back in Unit 1. You learned that webpages must still be parsed, interpreted and rendered by the browser after they are downloaded to the user’s local machine. If the user is on a low-end computer or using an older browser version, this process might take longer.

The user’s connection to the Internet also contributes to latency. Business users may be sharing an Internet connection with several other users on a Local Area Network (LAN). The majority of home users are still on dial-up connections (56Kbps and below). No matter how fast the intermediate network links are from the server, serving a 3 MB audio file to a client machine that is connected to the network at 56Kbps or to a congested LAN will be painfully slow.

Physical network

Essentially, the network is the line through which clients send requests to your server, and the path through which responses are sent from the server back to the client. In the case of the World Wide Web, the network used is the Internet.

The time it takes for those requests and responses to travel to and from your server is one of the largest limiting factors in user-perceived server performance. However, network latency is almost exclusively out of the control of Web server administrators. There is little you can do about a slow router on the Internet, the geographical distance between a client and your server, or network traffic congestion during peak usage hours and at cross-continent connections.

Most of the latency on the Internet is caused by the number of intermediate routers that data must go through. Each router contributes several milliseconds of latency because it has to read your data into a buffer before it can decide what to do with it. The fewer routers there are between a client and a server, the better the performance.

Different segments of the Internet also have varying traffic capacities, or network bandwidths. Bandwidth refers to the amount of data that can travel over a communications channel during a given period of time, usually measured in bytes per second (bps). Just as with any highway, data packets will encounter different network speeds as they hop from segment to segment.

The following online reading gives you a nice summary of the common data rates that are available in telephone and computer networks, including the Internet.
Web protocols

We’ve already encountered some of the important protocols used to implement the Web in Units 1 and 2. These protocols are Transmission Control Protocol (TCP), Hypertext Transfer Protocol (HTTP) and Domain Name Service (DNS). In this section, we will discuss the performance-related issues behind these protocols.

Domain name system

As described in Unit 1, DNS is the network name service which is used to convert a text-based host name (e.g., http://www.yahoo.com) into an Internet host address (e.g., 66.94.230.35). DNS is a highly essential service because most humans would rather remember domain names than machine IP addresses. 😊 DNS uses a hierarchical distributed database (figure 6.2). The root name servers know the IP addresses of all name servers that handle the top-level domains (e.g., .com, .net, .gov, .edu, etc.). If your local DNS server doesn’t have the IP address that you’re looking for, it knows the next root server in the hierarchy that it can ask. Delays may be caused when a DNS request needs to be sent to multiple name servers before it can be resolved. As of the time of writing (mid-2004), there are 13 root name servers in the world.

![Figure 6.2](http://www.anu.edu.au/people/Roger.Clarke/II/IPrimer.html)
With millions of Web requests per day, local DNS servers may get overloaded by address look-ups from Web clients. Even worse, a DNS server may go down or may be wrongly configured. If client machines are unable to retrieve a machine’s IP address because the DNS service is unbearably slow or unavailable, then they cannot send requests to the Web server at all, even if all the other pieces are in place and are otherwise functioning properly. Web administrators must be careful not to let the DNS service become a single point of failure.

Transmission Control Protocol

In previous units, we’ve seen how the Hypertext Transfer Protocol uses a separate TCP connection to transfer files for each request/response transaction. TCP was originally designed with the following assumptions:

1. connections would be relatively infrequent and last for a long period of time;
2. the amount of data transferred would be relatively large; and
3. correctness and completeness are more important than performance.

TCP is well-suited for providing reliable connections for session-oriented services such as Telnet and FTP. On the Web, however, the average file size retrieved by clients is on the order of 10 to 15 kilobytes (Rabinovitch and Spatscheck 2002) over connections which rarely last for more than 200 milliseconds (Wong 1997).

Clearly, HTTP and the Web places extraordinary demands on TCP and uses it in ways that it was never intended for. Here we look at the major performance problems caused by this.

1. Cost of setting up and closing a TCP connection

A client and server must exchange several rounds of packets in order to create and tear down a TCP connection: (a) three packets to initiate the connection; (b) a minimum of two packets to send a request and get a reply; (c) at least two more packets to close the TCP connection. This means that a minimum of seven IP packets must go across the network for even the smallest request (Killelea 1998).

All these exchanges constitute a considerable overhead for a typical HTTP data transfer of 10KB.

2. Acknowledgement windows and slow start

TCP sends an acknowledgement (ACK message) for every packet received. If an acknowledgement is not received within a required period of time, the sender will retransmit the packet. This is essentially how TCP ensures the reliable delivery of data.
To use the network more efficiently, TCP is able to send several packets without waiting for an acknowledgement of the first packet sent out. TCP uses a ‘slow-start’ or ‘sliding window’ protocol to determine how many packets to send prior to receiving an ACK message from the receiver. However, HTTP responses often have small file sizes and using the ‘slow-start’ protocol can actually increase response times. It is more efficient to send the whole request without waiting for the first acknowledgement to come back (Yeager and McGrath 1996).

As you can see, the poor interaction between TCP and HTTP may mean that HTTP spends more time waiting for data to be transferred rather than doing actual work.

**HTTP**

Under HTTP/1.0, neither the client nor the server begins processing the next request until the current one is processed completely. Separate TCP connections are also set up and torn down in order to request all the files that make up a single page.

Back in Unit 2, we discussed how HTTP/1.1 now allows persistent connections. With persistent connections, the TCP connection overhead only occurs once for each webpage download since the rest of the objects on the page are fetched over the same connection as the first object. HTTP/1.1 also allows pipelining over TCP, so a client can now send multiple requests over an existing TCP connection without waiting for previous requests to be processed.

This can greatly reduce the number of TCP/IP packets that are exchanged in the course of an HTTP operation. This leads to an overall reduction of Internet traffic.

Finally, most modern Web servers (released since 1999) can be set up to serve the compressed version of a static HTML page using GZIP format. HTTP compression has the effect of reducing the size of the HTML by 50% or more. There will be a more detailed discussion on HTTP compression in a later section called ‘Improving performance’.

**Server platform**

The Web server system consists of the hardware platform, operating system, networking software and an HTTP server.

The server hardware platform comes with limited resources, such as processor power, disk space and main memory. The Web server software must compete with the operating system and other application programs running on the system for these resources.

HTTP servers may also have varying performance efficiencies depending on their process model. For system sizing purposes, HTTP servers can be grouped into three categories: forking servers, threaded servers and keep-
alive servers (Wong 1997). We’ve already seen these types of servers before, but let’s discuss them a bit more here.

**Forking servers**

Forking servers got their name because they fork a server process for every incoming HTTP request. This simple process model can result in considerable overheads because most HTTP requests are short-lived and lightweight.

An intermediate solution is the pre-forking server. Instead of cloning a copy of the `httpd` program and running a separate server process for each HTTP request, a single parent process launches multiple child processes during system startup time. While this arrangement avoids the overhead of forking a new process, the helper processes are still relatively large and consume memory, so they are terminated after processing a certain number of requests or when no new requests are arriving.

**Threaded servers**

After their initial experience with forking servers, Web server implementers modified their code to use the threads model. Threaded servers are more efficient because the same copy of the Web server listens to multiple client connections. Different parts of the Web server program can run within different threads, with the operating system executing various threads simultaneously.

Switching between threads requires less overhead than switching between processes. Due to their increased efficiency, threaded servers consume less CPU time, permitting the server to process many more hits.

**Keep-alive servers**

Keep-alive servers are able to implement persistent connections, which is one of the key performance-related changes in the HTTP 1.1 specification. Persistent connections allow the transmission of multiple requests in a single connection. However, it is important to note that persistent connections can only be used to transport static content such as HTML pages and images.

**Dynamic content**

Serving dynamic Web content is a memory-intensive and CPU-intensive task. Interactive functions such as search engines, Web-based forms, online transactions and database access can consume a lot of processing resources. What’s more, server-side programs may need to interface with other services such as email, payment and transaction servers, further contributing to longer response times.
Even after a page has been downloaded to the client’s machine, any dynamic elements such as JAVA applets or FLASH animations will cause the webpage to be processed and rendered more slowly in the browser.

**User behaviour**

Website performance is also affected by the number of users who are on the site at a given time, and what pages or functions they are accessing. Users who are downloading a static HTML page, for example, may experience better response times than those who are waiting for an order confirmation. Response times may also suffer during peak visitor hours.

The following examples illustrate how user behaviour and access patterns may result in different experiences for your website audience:

1. There may be a sudden surge of visitors right after the Web address appears in a news story, when a new marketing campaign is launched or during the peak Christmas shopping season.

2. More customers may access websites between 7:00 and 10:00 p.m. after getting off from work.

3. Students who are taking a distance-learning course may need simultaneous access to an online chat room at 3:00 p.m. every Saturday.

Although an organization may have little control over its users, they need to understand the following aspects of customer behavior:

1. the pattern of services requested by customers

2. the demands that each service places on the site’s resources

3. the intensity at which customers arrive at the site.

Companies can monitor and analyse website usage in order to gather performance-related metrics such as number of hits, number of visitors, and most commonly accessed pages or functions. Understanding user behaviour and anticipating access patterns allows Web administrators to make informed decisions on website system requirements and capacity planning.
We will look into the ways of gathering website statistics in the next section. Meanwhile, complete the following self-test to assess your understanding of the factors which can affect Web performance.

**Self-test 6.1**

1. How can Web information providers ensure the robustness of their DNS service?

2. What features of TCP make it a poor choice for transmitting HTTP traffic?

3. Discuss the performance-related enhancements that were added to HTTP/1.1. Explain whether these enhancements can be used to speed up websites that serve primarily dynamic pages (e.g., pages built by server-side programs running in real-time).

4. Describe one or two examples of how understanding user behaviour and access patterns on the ABC Books online store can contribute to better server performance.
Measuring performance

How do you analyse the performance of a complex system like the Web? You could measure the user-perceived performance by analysing and interpreting the historical records gathered about the website’s actual performance. You could obtain live measurements of real website behaviour as it is taking place. Or you could measure the performance of individual components through controlled experiments to determine how efficiently each component is working and contributing to the whole system.

Let’s begin this tricky business of measuring Web server performance by understanding what to measure.

Criteria

There are three important variables that describe the performance of a Web server. The first two variables — connections per second and bytes per second, are related to throughput. The third variable — round-trip time, is related to latency. There is also a fourth set of variables pertaining to the hardware platform (e.g., CPU, memory and disk), which must be measured as well.

1 Connections per second

Also called requests per second or transactions per second, this can be calculated by getting the number of Web requests completed over a period of time, divided by the length of time (in seconds). It is more accurate to calculate the requests per second over short periods, such as every five minutes or even every minute of the day. This allows the Web administrator to accurately estimate peak traffic.

2 Bytes per second

This is also called the data rate. The more bits of data which can be pushed out by a Web server onto the network over a time interval, the more requests it can service.

This can be calculated by adding up the sizes of the files requested in the log files over a period of time and dividing this amount by the length of time (in seconds). Similar to connections per second, the data rate should be measured over several short intervals in order to determine the peak loads for the server.

3 Round-trip time (RTT)

This is the time that a user spends waiting for the page to begin showing up in the Web browser. The website’s response time from the user’s perspective can be computed as

\[
\text{client processing time} + \text{network processing time} + \text{website response time}
\]
As you can see, RTT is also equivalent to latency. Information providers may find it useful to set performance goals in terms of user response time, such as: ‘90% of HTTP requests for files under 10K must be served at a rate of five seconds or less for each file.’

4  CPU, disk and memory utilization of the server

For Web servers, the ideal utilization rate is 70%. If adequate system sizing and capacity planning was done beforehand, there’s no reason why utilization levels should ever exceed this level.

When the utilization of resources such as CPU, disk or RAM is almost 100%, it’s similar to a car that’s about to run out of gasoline! Ideally, Web administrators should configure their system or hire a monitoring service to send out an alert whenever the utilization levels exceed 70%, so that there’s time to react to the situation.

Most UNIX machines have programs such as `vmstat`, which is used to collect statistics about process, virtual memory, disk, and CPU activity. It may be a good idea to routinely log this data, so that it can be examined along with the Web log files.

```
example% vmstat 5
procs memory page disk faults cpu
  r  b  w  swap  free  re  mf  pi  po  de  sr  s0  s1  s2  s3  in  cs  us  sy  id
0 0 0 11456 4120 1 41 19 1 3 0 2 0 4 0 48 112 130 4 14 82
0 0 1 10132 4280 0 44 0 0 0 0 0 23 0 0 211 230 144 3 35 62
0 0 1 10132 4616 0 20 0 0 0 0 0 19 0 0 150 172 146 3 33 64
0 0 1 10132 5292 0 9 0 0 0 0 0 21 0 0 165 105 130 1 21 78
1 1 1 10132 5496 0 5 0 0 0 0 0 23 0 0 183 92 134 1 20 79
1 0 1 10132 5564 0 25 0 0 0 0 0 18 0 0 131 231 116 4 34 62
1 0 1 10124 5412 0 37 0 0 0 0 0 22 0 0 166 179 118 1 33 67
1 0 1 10124 5236 0 24 0 0 0 0 14 0 0 109 243 113 4 56 39
```

Figure 6.4  The following `vmstat` command displays a summary of what the system is doing every five seconds

In Activity 6.1, we will try a number of UNIX commands for monitoring the CPU, disk and memory usage on our Web server machine.

Online feedback to the activities in this unit is available on the OLE.

Activity 6.1

Run the following commands on your Web server to monitor CPU activity, memory usage and the available disk space.

```
1  vmstat — displays information on processes, memory usage, paging, block I/O and CPU activity:

shell% vmstat 5
```
2  top — displays all processes which are currently consuming the
most CPU resources (the display is updated every five seconds by
default):

    shell% top

3  ps — displays all the running processes with their memory and CPU
usage:

    shell% ps -aux

4  df — displays how much disk space is utilized:

    shell% df -ah

Consult the manpages on your Linux machine for more details on the
command-line options that can be used with these commands.

---

Log file analysis

Most Web servers keep logs of all requests, which include the time
requests were processed. This log information can be analysed to
determine two of the key measures: the connections per seconds and the
bytes per second. The third measure, round-trip time, is very difficult to
measure for an operational Web server. In the next activity, we will
perform some basic log file analysis by hand.

---

Activity 6.2

Download a log file sample from the course website.

Calculate connections per second and data rate (bytes per second) from
this sample. The correct answers can also be found on the course
website.

As you can see, a lot can be learned from the analysis of an operating
Web server. Log file analysis helps website administrators know what is
actually going on within the real system (figure 6.5). But this information
is essentially historical — if log files are analysed only once a month,
Web information providers can only spot problems at most one month
after they have occurred, which doesn’t give them an opportunity to
correct the problem in real-time.
Figure 6.5 Sample log file analysis output which compares the level of activity on weekdays and weekends

In the next activity, you will view sample outputs from WebTrends, a leading Web analytics company. As an optional exercise, you can also install a log file analyser on your machine and generate a comprehensive set of website metrics based on your log files.

Activity 6.3

Part 1 — Sample log file analysis report

1 View a sample report generated using Webtrends at:
   http://www.pon.net/service/webtrends/sample/index.htm

2 Click on the Table of Contents in the left-hand frame in order to go through the different categories of metrics that are included. Which categories include the measurements which are critical to Web performance?

Part 2 — Optional

1 Download and install Webalizer, a freeware log file analyser, on your local Web server at:
   http://www.mrunix.net/webalizer/

2 Run Webalizer against your Web server log file. If you do not have enough data, you can download a sample log file from the course website.

3 Go through the statistics produced in the report. Which of these measurements do you need for monitoring and evaluating Web server performance?

Many log file analysis software providers have repackaged their services so that they can now be offered over the Web. Web administrators can sign up for an account online, and then they are given a piece of code (e.g., Javascript) to insert in their pages. The client-side code is executed...
every time the page is loaded, sending back real-time statistics to the log file analysis company’s server. Web administrators can log in at anytime to view a wide range of reports and analysis for up-to-the-minute insights on server performance and visitor behaviour information. Websidestory and Webtrends are two of the leading companies who offer on-demand Web traffic analysis.

Even though log file analysis is the basic method for evaluating server performance, it is still unable to answer ‘what if’ questions such as ‘What if we modified the server to be multithreaded?’, ‘What if we added a second processor?’ or ‘What system enhancements do we need in order to handle twice as much traffic?’. Companies who perform log file analysis every month instead of on-demand are still relying on historical reports which may no longer reflect the current situation. There is also no way to measure RTT directly from log files alone.

To overcome some of these difficulties, it is necessary to conduct benchmark testing within a controlled laboratory experiment.

**Benchmarks**

Benchmarks are experiments that test the performance of computer systems using an artificially-generated workload. The workload must be designed to closely follow actual Web behavior as closely as possible.

During a Web benchmark test, a Web server is set up and requests are sent to it from several client computers over a TCP/IP-compatible network. When the server replies to a client request, the client records information such as how long the server took and how much data it returned and then sends a new request.

![Figure 6.6 Different components involved in a benchmark test](http://www.mindcraft.com)
The tester can also configure certain testing parameters such as the minimum number of clients, maximum number of clients, the number of testing iterations, and the frequencies at which certain files are to be retrieved from the Web server.

<table>
<thead>
<tr>
<th>Testing parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>### BENCHMARK PARAMETERS – EDIT THESE AS REQUIRED</td>
<td></td>
</tr>
<tr>
<td>ITERATIONS=&quot;3&quot;</td>
<td>This set of parameters means that the WebStone benchmark will run from eight clients to 128 clients, in increments of eight. Each increment will run for 30 minutes, and the whole test will be repeated three times. This test suite would take roughly 24 hours to complete.</td>
</tr>
<tr>
<td>MINCLIENTS=&quot;8&quot;</td>
<td>The file list tells WebStone which files to retrieve on the server. The sample on the left consists of five different files. The number following the filename is the weight of this file in the distribution. For example, in this fileset the weights add up to 1,000. So the file500k.html page will occur 350 out of 1,000 times, and the file5m.html will occur once every 1,000 pages.</td>
</tr>
<tr>
<td>MAXCLIENTS=&quot;128&quot;</td>
<td></td>
</tr>
<tr>
<td>CLIENTINCR=&quot;8&quot;</td>
<td></td>
</tr>
<tr>
<td>TIMEPERRUN=&quot;30&quot;</td>
<td></td>
</tr>
<tr>
<td>### Sample filelist, abstracted from access logs</td>
<td></td>
</tr>
<tr>
<td>/file500.html 350 #500</td>
<td></td>
</tr>
<tr>
<td>/file5k.html 500 #5125</td>
<td></td>
</tr>
<tr>
<td>/file50k.html 140 #51250</td>
<td></td>
</tr>
<tr>
<td>/file500k.html 9 #512500</td>
<td></td>
</tr>
<tr>
<td>/file5m.html 1 #5248000</td>
<td></td>
</tr>
</tbody>
</table>

Figure 6.7  Sample testing parameters from WebStone benchmark

*Source: http://www.mindcraft.com.*

Key variables or computer configurations can be changed one at a time in order to detect any differences caused by the change. For example, performance can be compared for different Web server versions (e.g., Apache 2.0 and Apache 1.3) running on the same hardware, or different Web server hardware configurations (e.g., number of processors, amount of memory) serving the same workload. In this way, it may be possible to discover optimal configurations and to diagnose performance bottlenecks.

The next reading describes how benchmarks can be useful in analysing website performance, and how published benchmark results can be interpreted. This can guide information providers when they are planning a benchmark test or when they are going through benchmark results published by outside parties.
At the end of a test run, the performance results from all Web clients are combined into a single summary report. Most benchmark tests are able to measure latency (time to complete a request) and throughput (operations per second and bytes per second). The main difference between benchmark tests and log file analysis is that round-trip time can be measured fairly accurately, because the Web clients are also part of the experiment. RTT is shown in the column labeled ‘Response’ in the figure below.

### Test Run Details

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Conforming Connections</th>
<th>Percent Conform</th>
<th>Throughput ops/sec</th>
<th>Response ms</th>
<th>ops/sec/1000mt</th>
<th>khit/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>106402</td>
<td>100.0%</td>
<td>29962.7</td>
<td>353.7</td>
<td>2.81</td>
<td>338.9</td>
</tr>
<tr>
<td>2</td>
<td>106401</td>
<td>100.0%</td>
<td>29219.3</td>
<td>356.6</td>
<td>2.78</td>
<td>336.1</td>
</tr>
<tr>
<td>3</td>
<td>106401</td>
<td>100.0%</td>
<td>29428.7</td>
<td>357.7</td>
<td>2.78</td>
<td>335.2</td>
</tr>
</tbody>
</table>

**Figure 6.8** Sample benchmark output from SpecWeb99

Errors will begin to appear in the results if the server becomes too heavily loaded during the test. This means that some clients are timing out before their requests are served. In fact, the number of errors at a given load can indicate how your server will perform under extreme traffic conditions. This is also known as stress testing, because you are intentionally testing the limits of your server.

According to Reading 6.2, it is not always necessary to perform your own benchmark testing. This may be especially true for smaller businesses that cannot afford to set up their own testing environment. Information providers can still consult the results published by vendors online and then choose the particular configuration that suits their needs. In the next activity, you will browse the latest results posted on the SpecWeb99 website at http://www.spec.org.

The next activity also provides you with the option of running a benchmark test on the Apache HTTP server for your ABC Books website. We will use the ApacheBench (ab) tool to fetch the same document repeatedly from the Web server and then measure the results.
Running ApacheBench - please wait...
This is ApacheBench, Version 1.3c <$Revision: 1.37 $> apache-1.3
Copyright (c) 1996 Adam Twiss, Zeus Technology Ltd,
http://www.zeustech.net/
Copyright (c) 1998-1999 The Apache Group, http://www.apache.org/

Benchmarking www.ouhk.edu.hk (be patient)...

Server Software: Apache/1.3.22
Server Hostname: www.ouhk.edu.hk
Server Port: 80

Document Path: /
Document Length: 35640 bytes

Concurrency Level: 5
Time taken for tests: 5.654 seconds
Complete requests: 25
Failed requests: 0
Total transferred: 898325 bytes
HTML transferred: 891000 bytes
Requests per second: 4.42
Transfer rate: 158.88 kb/s received

Connection Times (ms):
    min  avg  max
Connect: 213 214 223
Processing: 875 910 928
Total: 1088 1124 1151

---

**Figure 6.9** Benchmarking results on OUHK’s homepage using ApacheBench. The page was requested 25 times, with five requests sent out at a time (e.g., concurrency level).

---

### Activity 6.4

**Part 1**

1. Visit [http://www.spec.org/web99/results/](http://www.spec.org/web99/results/) and view the latest results posted for the SpecWeb99 benchmark. Since the results are posted every three months, the latest quarter as of the time of writing is the first quarter of 2004.

2. Go to **Advanced Requests** on the SpecWeb99 Results Search form ([http://www.spec.org/cgi-bin/osgresults?conf=web99](http://www.spec.org/cgi-bin/osgresults?conf=web99)). Click on the **Advanced** search form to view the published benchmark results for the following criteria:

   Company matches “IBM”
# CPU matches “1”

Memory matches “64 GB”

**Part 2**

You will use the Apache HTTP server benchmarking tool (ab) to simulate a number of users concurrently hitting the homepage of your ABC Books website. You may choose to run the test locally (i.e., on your local Web server) or remotely (i.e., using a Web gateway). However, running it locally offers you more options.

1. The ab tool should already be installed in your system as part of the Apache Web server distribution. View the manpage for this command so that you can go through the command line options available for it. In particular, read the description for command line options `n` and `c`:

```
shell% man ab
```

2. Next, we will use ab to simulate ten users concurrently requesting the ABC Books homepage (`index.php`). Each simulated user makes ten requests, for a total of 100 requests over the entire test run. Ideally, you would want to run ab on a different machine from your Web server.

3. For example, here is the command for hitting the ABC Books homepage on the OUHK Web server:

```
shell% ab -n 100 -c 10 http://plbpc001.ouhk.edu.hk/abc/index.php
```

You can gradually increase the workload (i.e., number of concurrent users and number of total requests sent) to find out how your server behaves under different conditions. You may also want to determine the point at which your server becomes overloaded and starts dropping requests (but only if your site is on a non-production server). Make sure you do not conduct stress testing on any production server since you may end up crashing it.

4. If you do not have a separate machine for running the ab tool against your Web server, and your ABC Books is published online, you can also try the ApacheBench gateway at http://codeflux.com/ab/ (figure 6.10) to hit your server.

![ApacheBench Gateway]

**Figure 6.10** Online form for submitting requests to the ApacheBench gateway program
Now that we’ve covered all the factors that could affect performance, the criteria that must be measured and the methods for collecting and analysing these criteria, let’s go through some of the steps that a Web information provider will go through when determining the hardware requirements for a Web server.

**Reading 6.3**


*Note*: This reading includes the sections up to and including ‘Sizing the hard disk drive’.

To sum up, here is a five-step plan (Tittel 1997) that can be followed by any Web administrator for monitoring server performance and for anticipating future capacity needs:

1. **State the operational requirements**
   - What information or transactions will the Web server deliver?
   - Who is the intended audience?

2. **Monitor, collect and evaluate performance metrics**
   - What is the system doing?
   - How does the system behave under the loads it experiences?

3. **Analyse the data, identify causes, and isolate the bottlenecks**
   - A bottleneck is any factor that limits system performance.
   - Performance monitoring data can only suggest possible bottlenecks; detailed analysis and real detective work is still necessary to identify actual causes.

4. **Set measurable objectives and implement changes**
   - Once bottlenecks are identified, state explicitly what kinds of effects system changes should provoke or cause.
   - It’s even more important to state them in a way that can be measured objectively.

5. **Forecast the effects of changes**
   - State what results a change should produce.
   - Compare actual against anticipated results.

In the next section, we will look at specific areas that can be tuned and optimized for better server performance. But first, complete the following self-test to revise the performance factors and issues that were just covered.
Self-test 6.2

1. Identify some of the disadvantages of relying only on log file analysis for evaluating Web server performance.

2. Identify some of the benefits and downsides of benchmarking.

3. What is stress testing and why is it useful in benchmark tests?

4. Why is the average throughput not as useful a measurement as the peak throughput? Why can average performance measurements such as average connections per second and average throughput be misleading?

5. Calculate the network bandwidth required to support the following Web server requirements:
   - average size of a hit (KB) = 11 Kilobytes; and
   - hits per second = 250 hits per second.
Improving performance

If you’ll recall, the ultimate goal of website optimization and tuning is to reduce the round trip-time for users who are accessing the site. Out of the three components that add up to the user’s perceived response time, only Web server response time is within the company’s control. In this section, we will focus on specific concepts and techniques for improving the Web server’s response time.

Let’s start with three readings which summarize a lot of the tuning and optimization techniques which can be used to improve server performance. Although the first two readings are dated 1997 and 1998, their content is still very relevant to the prevailing situation on the Web.

**Reading 6.4**

*Note*: Scroll down the reading and read the section titled ‘What’s known about contributing factors?’ only.

**Reading 6.5**

**Reading 6.6**

Now let’s go through some of the recommendations from Readings 6.4–6.6 in more detail.
Server hardware

From the user’s point of view, a Web server is useful only as long as it is serving Web requests. Any other services and applications running on the same machine, competing with the Web server for CPU cycles, main memory, disk access and storage space, are regarded as overhead.

One such service is the windowing system, which occupies a great deal of RAM and CPU time. This should be turned off, especially since the currently active window is given a higher execution priority than other processes, including the Web server. This can be done on UNIX systems by simply turning off X Windows.

I have also seen clients who run database servers and mail servers on the same box as their Web server. This option may be cheaper than maintaining separate machines for each service, but it can severely affect performance if any of these services become overloaded. If a company can afford it, I would recommend that they use their Web server strictly for serving webpages only, and shift other services to different boxes.

Small businesses often start out with a shared hosting solution, where many websites are served from the same machine. They should be careful to find out just how many other customers they will be sharing the hardware with, and what kinds of activities these other sites will be engaged in. One website that I worked on had very poor response times, and upon further inspection, it turned out that the Web hosting company was running up to 200 websites on the same machine! Once again, if a company’s budget allows it, it’s better to get a dedicated server where the system resources are devoted entirely to their site alone.

Aside from tuning the server by shutting down unnecessary services, shifting these services elsewhere, and making sure your website has sole control of system resources, you should also make sure that your current hardware configuration can handle your anticipated capacity needs. The following table summarizes the key hardware considerations for Web servers.
Table 6.1 Hardware considerations for Web servers

<table>
<thead>
<tr>
<th>Hardware component</th>
<th>Considerations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>For primarily static content, a one or two-CPU computer should be sufficient. However, sites that run search engines, shopping carts or online transactions with sufficiently high volumes of traffic may need a four-CPU setup. Note that multithreaded servers are needed in order to make full use of the processing power of multi-CPU architectures. In some cases, no matter how many CPUs it has, a single machine can no longer keep up with the volume and intensity of requests. Web administrators can consider spreading the site out across multiple servers. (Refer to the later section called ‘Mirroring documents’ for more details).</td>
</tr>
<tr>
<td>Memory</td>
<td>Accessing content within memory is faster than accessing it on disk by a factor of roughly 1 million (nanoseconds versus milliseconds). Performance-wise, the worst thing that can happen is when a memory shortage causes entire processes to be swapped out to disk. Servers must have a lot of RAM to reduce disk accesses. There must be enough memory to serve the needs of the operating system, Web server, any server-side programs, and cached content. Ideally, there should be enough RAM to hold all the Web content that you expect will be accessed within a five-minute interval (Killelea 1998). Multiple memory controllers can also allow multiple RAM accesses in parallel for greater overall throughput.</td>
</tr>
<tr>
<td>Disk</td>
<td>Servers should have separate high-speed SCSI disks for serving Web content and for logging. RAID (Redundant Arrays of Inexpensive Disks) technology can be used to set up a collection of disks where every bit of data is mirrored on two disks. RAID can also provide data striping, which involves spreading blocks of each file across multiple disks. This is highly recommended because it allows simultaneous or overlapping read operations on multiple disk drives.</td>
</tr>
<tr>
<td>Bus</td>
<td>The bus provides the communication channel between the different components in your machine (CPU, disk, memory and network cards). Choose a fast bus (e.g., 64-bit, 133 MHz PCI) so that internal data transfer rates on the server do not become the performance bottleneck.</td>
</tr>
<tr>
<td>Network Interface Card (NIC)</td>
<td>The NIC provides the connection between the network cable and the server’s bus. Data is copied between the server’s memory to the Network Interface Card memory. You will get the best performance from the most recent network cards. Consider Fast Ethernet (100 mbps), Gigabit Ethernet (1,000 mbps), or better. The most commonly used type of NIC card is the one that implements the Ethernet networking protocol. High-volume servers may require multiple network interfaces.</td>
</tr>
</tbody>
</table>
Network type and speed

When choosing the Internet Service Provider (ISP) that will host your website, the most important network factors to consider are:

1. topological and physical proximity to your customers
2. large amount of available bandwidth
3. redundant connections to multiple upstream providers.

Being topologically close to your customers means that there are relatively few routers on the path between your server and client machines. It’s ideal if your server is on the same ISP or shares a common upstream ISP as the customers, business partners or employees who frequently access your server.

The next reading describes the network, hardware, and website architecture requirements that you should look for in an ISP.

Reading 6.7


Network bandwidth is also important. Even if your machine has been tuned to provide high levels of throughput, your efforts will all go to waste if the pipe going into and out of your Web server isn’t big and fast enough to handle all the traffic generated. Look for ISPs with high-speed ATM or fiber-optic network backbones.

Activity 6.5

1. Review the two sample Unix server configurations offered by an ISP at:
   
   http://www.datapipe.com/unix_configs.aspx

2. Identify the differences between the basic and the optimized configuration. Explain how the additional features in the optimized configuration could lead to better performance.

Web server software

In our earlier discussion of server platforms in the first section of the unit, we reviewed the different Web server process models that were originally introduced in Unit 2. Among these process models, it is
recommended that you use multithreaded servers or pre-forking servers since they incur much less context-switching overhead. However, there is also some argument as to whether it is really beneficial to use multithreaded over pre-forking servers. By default, Apache 2.0 is distributed using a single-threaded, pre-forking, multiprocess model. This means that unless you have compiled your own copy of Apache 2.0 and configured it to be multithreaded, it is almost certain that it will not run in multithreaded mode.

You may wonder why Apache 2.0 is not configured as multithreaded by default. It's because Apache already has a proven and stable track record with the single-threaded, pre-forking model, while its multithreading features have not yet gone through the test of time. Also, if you plan on using your server primarily for PHP content, then running the Web server in multithreaded mode may slow down performance by as much as 15% because of the extra work done by PHP in order to be thread-safe (Zend Technologies 2003).

The rest of the section assumes that you are running Apache 2.0 or 1.x using the pre-forking model.

**Configuring the Web server**

Now let’s look at the directives within Apache’s configuration file (`httpd.conf`) which can be tuned for better server performance.

1 Number of processes — The important thing to keep in mind is to run only the number of server processes that your RAM can hold. If you run too many, you’ll start swapping processes out to disk and performance will suffer.

   MaxClients should not be too big, so it will not overload the machine's memory resources, and not too small, or it will force the users to wait for the child processes to become free to serve them.

   ```
   #Minimum number of idle child server processes
   MinSpareServers 10
   #Maximum number of idle child server processes
   MaxSpareServers 5
   #Number of child servers created at startup
   StartServers 5
   #Maximum number of child processes that will be created to #serve requests
   MaxClients 100
   #Limit on the number of requests that an individual child #server will handle during its life
   MaxRequestsPerChild 1000
   ```

2 Persistent connections — By default, persistent connections or keepalives are implemented on HTTP/1.1-compliant Web servers such as Apache 2.0 and 1.x.
Set MaxKeepAliveRequests high to save the overhead of setting up new connections. Set KeepAliveTimeout fairly low (e.g., 15 seconds) to avoid the impact of waiting for very slow or aborted connections.

```
# Enables HTTP persistent connections
KeepAlive On
# Number of requests allowed on a persistent connection
MaxKeepAliveRequests 200
# Amount of time the server will wait for subsequent requests on a persistent connection
KeepAliveTimeout 15
```

3 Logging — Reverse lookups allow the server to record domain names in the server’s log file, but log file analysis programs can look up the domain names offline anyway. Turn off runtime DNS reverse lookups to skip the extra latency it incurs.

```
# Enables DNS lookups on client IP addresses.
HostNameLookups Off
```

**Activity 6.6**

Use ApacheBench (refer to Activity 6.4) to compare the results of running the Web server with the following HTTP directives:

1 Test 1 — MaxClients 100, MaxRequestsPerClient 1500
2 Test 2 — MaxClients 50, MaxRequestsPerClient 1500

For both tests, simulate ten concurrent users accessing `index.php` a total of ten times each. Keep the other configuration directives constant.

You should find that the server has a higher throughput (e.g., requests per second) in Test 1, with MaxClients set to a larger number.

### Improving the download time of Web content

As a Web information provider, the client machine and Internet connection is beyond your control. However, there are still some ways to improve download times in the way you code and serve your Web content. Here are seven useful tips:

1. minimize HTML file size and complexity
2. use HTTP compression to reduce the size of files being transmitted over the network
3. validate your HTML
4 minimize the amount of dynamic content on the site

5 don’t always try to do everything in real-time

6 use server-side APIs, compiled programs or embedded interpreters instead of CGI

7 use secure sockets layer (SSL) only when necessary.

Let’s look at each tip in some more detail.

**Minimize HTML file size and complexity**

Yahoo and Google are two of the most visited websites on the planet. It’s also no coincidence that both of them have clean and simple user interfaces that are primarily text-based.

Web information providers should avoid fancy features such as animations, sound, video and huge graphics if they can communicate their message just as effectively using simple text and small images. Even when multimedia files are included, they must be optimized to make sure that their file sizes are as small as possible without sacrificing quality.

Many designers also use nested tables (e.g., HTML tables within tables) and tricks such as invisible images in order to control page layout. This makes HTML more complex and increases the amount of processing that the Web browser must do to display the page. It’s recommended that cascading style sheets (CSS) be used for page formatting and layout instead of HTML.

We will discuss CSS in more detail in *Unit 7*.

**Use HTTP compression to reduce the size of files being transmitted over the network**

All modern browsers released since early 1999 are capable of receiving compressed Internet content via standard IETF (Internet Engineering Task Force) Content Encoding. An HTTP/1.1-compliant browser indicates that it is able to accept compressed pages by sending the following header within the HTTP request:

```
Accept-Encoding: gzip, compress
```

If there is a static, compressed version of the requested document on the Web server’s hard drive which matches one of the formats in the `Accept-Encoding` header, the server can simply choose to send the pre-compressed version of the document instead of the much larger uncompressed original.

The next reading discusses in more detail how HTTP compression can work and notes its pros and cons.
Reading 6.8


*Note*: Start from the section called ‘What exactly is HTTP compression?’

Now let’s try out HTTP Compression for ourselves in the next activity. We will use the `telnet` command to request the homepage on Google’s server, just like in the previous reading. We will also verify the effect of the following header: `Accept-encoding: gzip` on the length of the HTTP response.

**Activity 6.7**

1. From the command line, open a connection to Google’s mirror site for Hong Kong:

   ```
   telnet www.google.com.hk 80
   ```

   Then enter your request line by line. Press the **Enter** key twice after the headers have been entered in order to get the blank line.

   ```
   GET / HTTP/1.1
   User-Agent: Mozilla/4.0 (compatible; MSIE 6.0)
   Host: www.google.com.hk
   Accept: */*
   [blank line here]
   ```

   Here are the HTTP response headers returned by Google. The size of the HTTP object returned is 3,322 bytes.

   ```
   HTTP/1.1 200 OK
   Cache-control: private
   Content-Type: text/html
   Server: GWS/2.1
   Content-length: 3322
   Date: Sat, 05 Jun 2004 15:42:01 GMT
   ```

2. Now request the homepage again, but this time, specify that the client can accept compressed content from the server.

   ```
   GET / HTTP/1.1
   User-Agent: Mozilla/4.0 (compatible; MSIE 6.0)
   Accept-Encoding: gzip, deflate
   Host: www.google.com.hk
   Accept: */*
   ```
This time around, the size of the HTTP object returned is 1,450 bytes, over 50% smaller than the regular version.

```
HTTP/1.1 200 OK
Cache-control: private
Content-Type: text/html
Content-Encoding: gzip
Server: GWS/2.1
Content-length: 1450
Date: Sat, 05 Jun 2004 15:37:43 GMT
```

Now let’s continue with our list of tips for improving the download times of Web content from our site.

**Validate your HTML**

Developers can help speed up the browser by ensuring that their HTML is error-free and compliant with existing HTML standards, such as HTML 3.2 or HTML 4.0. We will discuss HTML issues for website developers in more detail in Unit 7.

**Minimize the amount of dynamic content on the site**

In an earlier section, you saw how dynamic content can increase the CPU and memory requirements on the Web server. Whenever possible, generate the results of server-side scripts periodically instead of running the script for every request.

For example, ABC Books has a page for every book in its catalogue. Since the information about a book isn’t likely to change once it has been entered in the database, the page about each book can be generated once by pulling all the necessary content from the database. The static HTML page produced by the script can now be stored on disk and served to customers next time. Another example is the homepage, which may be updated with new featured books every month. The static version of the homepage can then be generated once a month by the server-side script which pulls in the book descriptions, images, etc. from the database.

Your server-side applications can also push whatever processing load they can onto the client. This saves on server-side resources and allows responses to appear almost instantaneously to the user. A common example is adding client-side code to validate user entries in online forms.

**Don’t always try to do everything in real-time**

On a particular online shopping website that I helped develop, performance would become unreliable whenever the user got to the Order Confirmation page. Instead of receiving a screen with their Order
Number and a message saying that their order had been successfully processed, the website would simply hang or time-out for some, but not all, customers.

It turned out that the Order Confirmation script had to wait a long time in order to send out the email confirmation message to customers. The actual work of sending out the email was done by the mail server, which was shared among other customers of the same Web hosting company. Whenever the mail server was heavily loaded, the Order Confirmation script would time out while waiting for it to finish the job.

The problem was solved by taking the email sending function out of the Order Confirmation script. Instead, a program was scheduled to run every 30 minutes which would generate and send out email confirmations to customers off-line. This dramatically improved the response times of the Order Confirmation script, and most customers probably did not mind receiving their email confirmation a little later in the day anyway.

The moral of the story is that any processing done within a server-side script can contribute to user response times. Look for any opportunities to offload tasks that do not need to be done in real-time.

Use server-side APIs, compiled programs or embedded interpreters instead of CGI

Back in Unit 3 Server-Side Programming, we looked at different mechanisms for providing Web server extensions. Among these mechanisms, the Common Gateway Interface (CGI) proved to be the simplest to implement but the most resource intensive. If you must use CGI, then use compiled rather than interpreted programs for better performance.

Use secure sockets layer (SSL) only when necessary

The SSL protocol is widely used for encrypting data sent to and from Web servers. There is a high overhead associated with SSL because of the extra processing involved in encrypting and decrypting every byte of data sent out over the network. SSL can only encrypt entire webpages, not just the confidential bits of information within the page. So make sure that you use SSL only when you really need to secure a webpage, and keep the file sizes of encrypted pages as small as possible.

There will be a more detailed discussion on SSL in Unit 8.
Self-test 6.3

1 Assess and evaluate the performance tuning tips from Readings 6.4–6.6 in relation to your own Web server. Rank 5 of these tips in the order of their relevance to your local Web server and explain your ranking.

2 Explain the effect of the following server configurations on server performance and user response times:

- setting MaxSpareServers and MinSpareServers to a large number;
- setting MaxRequestsPerChild to 0 (e.g., unlimited requests per child, which means that child processes never expire);
- setting KeepAliveTimeOut to a larger number; and
- setting HostNameLookups to ON.

3 On a Web server computing platform, an overhead is any action that does not directly contribute to useful work in handling Web responses. Give a few examples of system overheads.
Mirroring documents

We’ve already discussed how documents may be divided among different servers in Unit 2. This is sometimes necessary to avoid having a single Web server become the performance bottleneck. The two main ways of doing this are to create a full or to create a partial replica of an origin server. In this section, we’ll take a closer look at the practice of Web mirroring or replication, where the entire document tree is fully copied to several mirror or replication servers.

Mirroring an entire server is not difficult to do. In practice, mirrored servers usually take a snapshot of the server periodically, perhaps once a day or as often as every hour. This means that the mirrored version may not be totally up-to-date if Web content changes rapidly on a particular site.

Figure 6.11 Here’s a short list of the mirror sites for Apache.org. The complete list can be found at http://www.apache.org/mirrors.

Aside from HTML documents, you may also find it necessary to replicate content stored on relational database servers. MySQL version 1.x and up allows master-slave relationships to be set up between database servers, where the slave server is configured to connect to the master server and take periodic snapshots of the database.

Websites that have primarily read-only content and do not wish to introduce the complexity of replication can keep HTML and image files in a Network File System (NFS) instead. This allows multiple Web servers to access and manipulate files concurrently on the NFS drive over a LAN. As far as the Web server is concerned, these files appear as if they were stored locally on their own system. In this way, Web administrators can avoid the extra administrative tasks involved in replicating and synchronizing the content on mirrored servers.
The pros and cons of mirroring

The following are the benefits of mirroring:

1. The website is easily available due to the mirroring of content on multiple servers. Even if one server goes down, other servers are still available to process requests.

2. Users can obtain documents from a mirror server nearer to them on the network, which reduces their waiting time.

However, mirroring also has drawbacks. Copying all the documents on a server can result in wasted disk space if many replicated documents are rarely used. The Web server document tree must be replicated for each mirrored copy. The multiple copies of the Web server documents must be kept consistent as new documents are added or existing documents are changed over time. Ideally, there should also be a mechanism for directing users to mirror sites in such a way that the workload is distributed evenly among the alternate servers.

A very basic method is to present the user with a webpage of alternate servers and require the user to manually select a server by clicking on a link. However, this approach does not attempt to balance the workload across servers. It’s even possible that the mirror server chosen by the user may not be able to accept any more incoming requests due to overloading.

A more sophisticated mechanism involves using a simple script that automatically redirects a user to an appropriate server. For example, the script could examine the user’s domain name and determine which mirror site to direct the user to. The mirror site could be sent to the user’s browser via the HTTP redirect header, like Location: http://www.someserver.com. The following HTTP exchange illustrates how Google automatically redirects me to the mirror site http://www.google.com.hk.
$ telnet www.google.com 80
Trying 66.102.7.147...
Connected to www.google.akadns.net.
Escape character is('^]'.

GET / HTTP/1.1
Host: www.google.com
Accept: */*

HTTP/1.1 302 Found
Location: http://www.google.com.hk/cxfer?c=PREF%3D:TM%3D1086449405:S%3D545LVx0OlIGWAY6U
Set-Cookie: PREF=ID=7a67cd0262ba7ade:CR=1:TM=1086449405:LM=1086449405:S=75C61Qyyvy2ZntLT; expires=Sun, 17-Jan-2038 19:14:07 GMT; path=/; domain=.google.com

Content-Type: text/html

Server: GWS/2.1
Content-length: 207
Date: Sat, 05 Jun 2004 15:30:05 GMT

<HTML><HEAD><TITLE>302 Moved</TITLE></HEAD><BODY>
<H1>302 Moved</H1>
The document has moved
</BODY></HTML>

Figure 6.12 Automatic redirection to a mirror site by Google

The difficulty with this approach is that domain names do not always correspond to the user’s physical location. Another problem is that it still does not verify availability or balance the load among mirror servers.

Load balancing using round-robin DNS

Round-robin DNS is a more sophisticated and elegant solution which returns a different IP address each time it is queried for the IP address corresponding to a Web server. This attempts to distribute the workload evenly across a set of servers (as shown in figure 6.13).

In this architecture, the DNS/load balancing support and Web servers are on a LAN in Hong Kong. The load balancer can also direct incoming HTTP traffic to a Web server on a different network in London. This configuration creates a virtual Web service that spans all three Web servers. If a Web client with a company business unit in France wanted to download large images from the virtual Web service, the load balancer
may be able to look at the Web client’s domain name, determine that she is in Europe, and redirect her request to the London Web server.

Figure 6.13 Load balancing for an international company with networks spanning several continents

The next reading tells you more about load balancing via round-robin DNS, along with its limitations.

Reading 6.9


Note: Read up to the section titled ‘Disadvantages of hardware load balancers’ on the second page only.

In this section you have reviewed mirroring and load balancing as ways of boosting a website’s performance by making available several copies of a Web document from different servers. Now do the following self-test to assess what you have just learned.
Self-test 6.4

1. Discuss three mechanisms for directing users to an appropriate mirror server.

2. Identify some of the disadvantages of mirroring a website.
Caching

Aside from upgrading to a faster CPU, newer motherboard, bigger memory and the latest O/S version, it is also a good strategy to make use of the server’s resources more effectively. Caching is one way to wring out more performance from your existing platform without having to upgrade your hardware or operating system.

In general terms, caching refers to the ‘storage of recently retrieved computer information for future reference’ (Wessels 2001). Caching is a commonly used technique for enhancing performance in computer and networking systems. Computer processors, operating systems, file systems and even Internet services such as the Domain Name System (DNS) all make use of caching to speed up response times.

Specifically, a Web cache sits between Web servers (or origin servers) and Web clients. It examines outgoing HTTP requests for objects such as HTML pages, images, audio files, ZIP files and so on. When an incoming HTTP response is returned for a specific request, the cache may save a copy of the object for itself. Then, if there is another request for the same object, it can serve the request itself instead of asking the origin server for it again.

A hit occurs whenever requested data are served from the cache, while a miss takes place when referenced data is found in the cache. Service times for cache hits are much less than service times for misses, so the performance improvement from caching relies on increasing the percentage of all requests that are hits (also called the hit ratio).

Caching takes advantage of a principle known as locality of reference. This means that requests for certain pieces of data are likely to occur together. A request for the OUHK homepage is usually followed by requests for all the page’s embedded graphics. Files that were accessed most recently are also likely to be accessed again soon. Caches use locality of reference to predict future accesses based on previous ones.

The benefits of caching

Here are some of the benefits of caching Web content:

1 Reduces latency

You’ve already encountered the concept of latency earlier in this unit. Latency refers to amount of time it takes to transmit data from one point to another. Cached documents are closer to the client on the network and are therefore faster to retrieve. Performance increases are especially evident for frequently downloaded documents (e.g., Yahoo’s homepage) or for very large documents.
2 Reduces the server load

Caching reduces the number of requests made to an originating Web server. This is because the cache acts on behalf of the Web server in delivering documents to the user.

3 Reduces the load on the network (bandwidth consumption)

Caching results in fewer HTTP requests and responses travelling over the network. It can lead to considerable savings for companies whose Internet service is billed on a per-usage basis or on bandwidth use rather than at flat rates.

For the remainder of this section, we will go into the details of how caching can be used to improve the performance and efficiency of the Web. You will learn how Web caches work, how they interact with clients and servers, and the role that HTTP plays in Web caching. Aside from the technical aspects of caching, we will also talk about some of the political issues behind it.

How it works

There are three different types of Web caches, depending on where they are located on the path between a client and an origin server.

Many browsers have built-in caches which can store responses in the client computer’s memory and disk space for later use. A browser cache is limited to just one user, and it gets hits only when the user revisits a page.

Proxy caches, unlike browser caches, can serve many different users with cached objects from many servers. They are typically found in organizations such as Internet Service Providers (ISPs), corporations and schools. Since the users within an organization are more likely to visit the same websites, caching proxies usually have higher hit ratios than browser caches. Hit rates of 50% efficiency or greater are not uncommon (Nottingham 1999). OUHK offers its own proxy service which is accessible over the Web, called the OUHK Library Proxy Service (http://proxy.lib.ouhk.edu.hk).

Surrogates comprise the third type of Web cache. These are used by Content Distribution Networks (CDNs) to replicate information at many different locations on behalf of a content provider. Web clients are automatically directed to the nearest surrogate which contains a given resource. Akamai and Digital Island are two of the leading Content Distribution network providers.

Caching can also be done on the server-side. The output of dynamic requests can be cached in memory so that the server-side script does not have to be executed every time. If a database server is becoming a performance bottleneck, recently executed queries and their results can be cached as well.
Figure 6.14  Cache usage on the Web


We will take a closer look at browser caches and proxy caches in the next sections.

**Browser caches**

Try this quick experiment: fetch a series of static webpages from one of your favourite Internet sites. Disconnect your computer from the Internet. (If you dial-up to an Internet Service Provider, hang-up the modem connection.) Select the **Back** button on your browser to revisit a URL you just visited and then select the **Forward** button. You should be able to see the webpages you just visited when you were connected to the Internet. How is it possible to view a webpage if your computer is not connected to the Internet? This is your browser’s cache at work.

The Web browser cache is an area on the computer’s hard disk (and in the memory) where the browser manages and stores files it retrieves from Web servers on the Internet. If the user requests the same document more than once, the browser may return to the user the document in the cache rather than fetching the document again from the Internet. Retrieving documents from a browser’s cache is much faster than fetching documents from the Internet because the cached documents are on the local computer’s hard disk. In the next activity you will learn how the browser’s cache works.

**Activity 6.8**

In this experiment you will learn how to adjust your Web browser cache settings, view the contents of the browser cache, turn off or ‘zero-out’ the browser cache, activate your browser cache, and witness how your browser cache stores documents as you make Web requests.
1 How do I adjust the cache?

Netscape: From the menu, select **Edit → Preferences → Advanced → Cache**. You should see a menu similar to the one in figure 6.15.

![Figure 6.15 Adjusting the browser cache in Netscape](image)

IE: From the menu, select **Tools → Internet Options → General → Temporary Internet Files → Settings**. You should see a menu similar to figure 6.16.

![Figure 6.16 Adjusting the browser cache in Internet Explorer](image)
You can specify how the browser will use its cache through the **Check for newer version of stored pages** (IE) and **Compare the page in the cache to the page on the network** (Netscape) settings. You will learn how this works later on in the unit.

At this time select one of the following settings: **Once per session**, **When the page is out of date**, **Automatically**, or **Every time you start Internet Explorer**.

You can specify the size of the browser’s disk cache with the **Disk space** and **Amount of disk space to use** settings. A large cache will store and manage a lot of files for a long period of Web browsing; a small cache can only store a few files over a short period of Web browsing. You should see that the size of your browser cache is greater than 0. If it is not, set the value to a number greater than 0, and browse around a bit on the Web before returning to this exercise.

### 2 Where is the browser cache?

The dialog boxes in figures 6.15 and 6.16 also show the folder where cached content is stored by the two browsers. You can look for these folders within the file system and verify that the cached files are there.

### 3 How do I view the contents of the browser cache?

The browser’s cache is a protected directory that is managed by the browser application. The browser determines the way the files in the cache are organized and named. The cache may be organized as a single directory or as a tree directory structure. You should not attempt to directly view files in the cache since the browser is actively managing and protecting this directory. However, it is safe to copy cache files into another directory and then view the copy with a Web browser.

To see the contents of the Internet Explorer browser cache use:

```
Tools -> Internet Options -> General -> Temporary Internet Files -> Settings -> View Files
```

The Netscape (version 6) browser cache is organized as a tree directory structure and named according to the browser’s own internal naming convention. View the cache with Explorer by opening the following directory:

```
C:\WINNT\Profiles\{username}\Application Data\Mozilla\Users50\default\{username}\Cache
```

The files in the cache are organized in a single directory and named according to their URLs. Select and copy a few of the cached files to a new directory and view the copies with a Web browser. You should recognize the images and documents as files you have already downloaded from the Web.
4 How do I turn off the browser cache?

For some experiments in this unit it is necessary to turn off the browser caching feature or clear out old entries from the cache. Netscape offers you better control in disabling the browser cache, so it is recommended that you use Netscape in these experiments. The Netscape browser cache can be turned off by setting the size of the disk cache to 0. Setting the disk cache to 0 and then resetting the disk cache to a size greater than 0 will ‘clean out’ all the current browser cache entries.

Follow these instructions to turn-off the Netscape cache:

- From the menu, select Edit → Preferences → Advanced → Cache.
- Set Memory Cache size to 0.
- Set Disk Cache size to 0.
- Select Clear Memory Cache.
- Select Clear Disk Cache.
- Fetch a static webpage from the Internet.
- Disconnect your network connection to the Internet. If you dial-up to an Internet Service Provider, hang-up the modem connection.
- Try to refetch the same webpage. You should get an error. If you can see the webpage then your Web browser cache is still turned on and you have not disabled your browser correctly.

5 How do I see the browser cache in action?

- Clear out the browser cache by setting the disk cache size to 0 and then resetting the value to a value greater than 0.
- Use the Explorer Find Tool to view the contents of the cache. It should show that the cache is empty.
- Fetch a static webpage from the Internet
- Select Find Now with the Explorer Find Tool to see an updated view of the cache contents. You should see a few new files that your cache has stored from your Web requests.

You have just witnessed your browser cache at work!

Proxy caches

As the name suggests, a proxy server is one that provides services on behalf of another server. Inside the firewall, clients send all requests to the proxy server. The proxy’s job is to retrieve the requested objects from their external location and then provide them to the requesting
client. This arrangement permits external traffic to be directed only to the proxy system, rather than exposing every internal client to the hazards of the Internet.

Proxy servers often improve their performance by using caching techniques. Such servers are also known as proxy caches or caching proxies. By caching the requests and their replies, caching proxy servers are able to satisfy many internal requests without having to issue requests on the external Internet.

![Diagram of a proxy cache](http://www.dlib.org/dlib/january96/ncsa/01mcgrath.html)

**Figure 6.17** Diagram of a proxy cache

*Source: http://www.dlib.org/dlib/january96/ncsa/01mcgrath.html.*

Here are the steps illustrated in the figure above:

1. The browser sends the request to the proxy server rather than to the Web server.

2. The proxy server checks to see if it has a copy of the requested document and, if so, returns it to the client, just as if it came from the Web server.

3. If the document is not in the cache, the proxy server acts as a client, sending the request to the correct Web server, and then relays the response to the browser.

4. The proxy will save the document so that it will not have to request it again from the server.

As you can see, proxy caches must handle a larger load than other servers, since they are both clients and servers. As a middleman or go-between, the proxy must split an HTTP request into two separate TCP connections, one to the client and the other to the server.
In the next activity, you will configure your Web server to also act as a LAN Web caching proxy, configure your Web browser to use the proxy, and then discover how the proxy works to cache Web files.

---

**Activity 6.9**

Some proxy caches do not cache domain name mappings to IP addresses. In this experiment you will use IP addresses within the URLs of pages that you access so that you can witness how the proxy cache works without any influences due to DNS.

1. Find the IP address of any Web server that serves static HTML pages. For example, you may choose `http://www.esdlife.com`.

2. Use `nslookup` to resolve the domain name `www.esdlife.com` to an IP address:

   ```
   > nslookup www.esdlife.com
   Non-authoritative answer:
   Name: www.esdlife.com
   Address: 203.184.172.159, 203.184.176.159
   ```

   Later on, you will access `http://203.184.172.159/` via the caching proxy.

3. Save a new version of the Web server configuration file (httpd.conf) and name it `httpd8080.conf`. Edit this new version by uncommenting the configuration directives needed to activate caching. Follow the instructions within the comment lines below.

   ```
   # Set the port number on which the proxy server is to listen.
   Port 8080

   # Uncomment these modules so they are dynamically loaded by
   # the Apache Web server
   LoadModule expires_module modules/mod_expires.so
   LoadModule headers_module modules/mod_headers.so
   LoadModule proxy_module modules/mod_proxy.so

   # Uncomment to allow the proxy to cache documents
   CacheNegotiatedDocs

   # Uncomment these to turn the Expire headers on#
   ```
ExpiresActive on
ExpiresDefault M60

#Insert this log file format so your proxy
will log the relevant HTTP header values
LogFormat "%h %l %u %t \"%r\" %>s %b
\"%{User-Agent}i" \"%{Pragma}i" \"%{Cache-Control}i" \"%{Expires}i" \"%{Via}i"
\"%{ETag}i" \"%{Refresh}i" " debug

#Turn on the custom logging capability
CustomLog logs/access.log debug

#Uncomment to enable the proxy server.
IMPORTANT!: Change #the “Order” and “Allow”
parameters as shown below so that #your local
Web browser can use the proxy.
<IfModule mod_proxy.c>
    ProxyRequests On
    <Directory proxy:*>
    Order allow,deny
    Allow from all
    </Directory>
    ProxyVia On
    <Directory proxy:*>
    Order allow,deny
    Allow from all
    </Directory>

    #Modify here to indicate your own Apache
    proxy root directory, ${PROXYROOT_DIR}
    CacheRoot "/usr/local/apache/proxy"

    #Uncomment these cache parameter defaults
    without modifying the default values
    CacheSize 5
    CacheGcInterval 4
    CacheMaxExpire 1
    CacheLastModifiedFactor 0.1
    CacheDefaultExpire 2

    #This cache parameter remains commented-out
    # NoCache a_domain.com another_domain.edu
</IfModule>

4 Stop and then start your Web server using httpd8080.conf as
the configuration file. This will activate the Web server as a caching
proxy.

5 Configure the Web browser to turn off browser caching and use the
proxy server instead. The instructions for doing this on Netscape are
given below. Instructions for Internet Explorer are not included since
there is no version of IE available for Linux.
For Netscape

1. From the menu, access **Edit → Preferences → Advanced → Proxies**.

2. Select **Manual proxy configuration**. This directs your browser to first look for a document in the proxy cache before fetching the document from a Web server on the Internet.

3. Set the HTTP proxy value to **127.0.0.1** and the port value to **8080**, which is the port that your local Web server is listening on. Select **OK**.

4. Turn off Web browser caching (refer to Activity 6.8).

5. Fetch a few webpages from the Internet using a URL that contains an IP address. For example, http://203.184.172.159, http://203.184.172.159/, etc.

6. Disconnect your computer from the Internet. Now select the **Back** button on your browser to revisit a URL you have just viewed. Then select the **Forward** button. You should still see the webpages you just visited since they are now served from the proxy cache. This is your proxy cache at work!

7. Open the proxy root directory `${PROXYROOT_DIR}` and you will see that the proxy cache is organized as a tree structure just like the Web browser cache. The proxy cache manages that structure and the internal naming of the cache files. For example, the cached HTML file from http://203.184.172.159 was found in

   $PROXYROOT/m/1/i/41f4brm2v5bi0ffrmxc4gzg

   You can open the cached file in a word processor. Notice that the HTML and GIF file in the cache can no longer be displayed as normal HTML and GIF files with a Web browser because the proxy cache has written some additional data at the top of each file to help it manage the files in the cache. The proxy cache is caching the HTTP header files sent from the original Web server along with the HTML and GIF files.

8. Re-enable your Web browser cache and refetch the page from http://203.184.172.159 (or whatever file you are using). Then disable your Internet connection and refetch the same page. Which cache is returning the page to you now? Is it the browser cache or the proxy cache? Identifying the correct copy of the Web document and the correct source of the document is a common problem when using multiple cache systems.
What cannot be cached

A cache decides if a particular response can be cached by looking at different components of the HTTP request and response. For typical HTTP request streams, about 75% of responses are cachable (Wessels 2001).

Here are the relevant portions of an HTTP message which are examined by the cache:

1 The response status code

The most common status code is 200 (OK), which means that a request was successfully processed. By default, the 200 code is cachable. For information about whether a document is cachable for a particular status code, go to http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html.

2 The request method

GET is the most popular request method, and responses to GET requests are cachable by default. A POST response is cachable only if the HTTP response includes an expiration time or if it includes certain Cache-control directives (see below).

3 Response Cache-control directives

HTTP/1.1 now includes the Cache-control header in the HTTP response. This header can be used to instruct caches on how to handle requests and responses.

For example, the cache can be instructed to store a response only in the browser cache (e.g., private), but not in a proxy cache (e.g., shared). It can also be used to specify the expiration time of an object by including its maximum age in seconds. Table 6.2 lists some examples of Cache-control directives and their descriptions.
Table 6.2  Examples of Cache-control directives

<table>
<thead>
<tr>
<th>HTTP Cache-control headers</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache-control: private</td>
<td>Permits browser caches to store a response but prevents shared caches from doing so.</td>
</tr>
<tr>
<td>Cache-control: public</td>
<td>Makes normally uncachable responses cachable by both shared and unshared caches.</td>
</tr>
<tr>
<td>Cache-control: max-age=3600</td>
<td>Specifies the expiration time of an object in seconds. In the example to the left, the object needs to be fetched from the origin server again after it has been in the cache for an hour or more.</td>
</tr>
<tr>
<td>Cache-control: no-cache</td>
<td>Allows a response to be cached, but it must be revalidated before it can be reused.</td>
</tr>
<tr>
<td>Cache-control: no-store</td>
<td>Makes a response uncachable. It is intended to prevent the temporary storage of information which the owner does not want to be cached.</td>
</tr>
</tbody>
</table>

Cache-control directives will be discussed in more detail in the section called ‘The role of HTTP’.

4  A response validator

Validation is used whenever caches have a local copy of a document, but they’re not sure if it’s still up-to-date.

The most common validator is the time that the document last changed, the Last-Modified time. When a cache has an object stored that includes a Last-Modified header, it can use it to ask the server if the object has changed since the last time it was seen, with an If-Modified-Since request. If the cached object is still valid, the server replies with a short HTTP 304 (Not Modified) message. This avoids having to download the entire object unnecessarily.

We will discuss the validation mechanisms available under HTTP/1.1 in more detail in the section called ‘The role of HTTP’.

5  Request authentication

Requests that require user authentication are not normally cachable. When a user tries to access a protected resource, the HTTP server returns a 401 (unauthorized) status code. It also returns a WWW-Authenticate header, which normally will result in the browser prompting the user for a username and password.
When a proxy cache finds the \texttt{WWW-Authenticate} header in a request, it knows that the corresponding response is uncachable unless the origin server explicitly allows it based on the \texttt{Cache-control} directives that are also returned.

### The role of HTTP

HTTP/1.1 provides a number of headers for the exchange of information between origin servers and caches, as shown in the following table.

<table>
<thead>
<tr>
<th>HTTP header</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>The current time when an origin server generates the HTTP response. This is always presented as Greenwich Mean Time (GMT). RFC 2616 requires origin servers to generate a Date header for every response.</td>
</tr>
<tr>
<td>Last-Modified</td>
<td>The time when the resource was last modified on the server, also presented in GMT format. RFC 2616 says that an origin server ‘should send a Last-Modified value if it is feasible to send one …’ For objects stored on disk, this corresponds to the modification time recorded in the file system. However, dynamic responses generated from server-side scripts often do not have a Last-Modified time stamp.</td>
</tr>
<tr>
<td>Expires</td>
<td>A response that includes the Expires header may be reused without validation (i.e., a cache hit) until the expiration time is reached. The Expires header can also turn an otherwise uncachable response into a cachable one. For example, responses to \texttt{POST} requests are uncachable by default, but they can be cached if there is an Expires line within the HTTP response.</td>
</tr>
<tr>
<td>Cache-control</td>
<td>Contains a number of directives which are basically instructions to the cache. Some of these directives are Private, Public, No-Cache, No-Store and Max-age.</td>
</tr>
</tbody>
</table>

The next online reading describes the \texttt{Last-Modified}, \texttt{Expires}, and \texttt{Cache-control} headers in more detail, along with some examples.
Reading 6.10


This reading is composed of three parts:

1 http://www.wdvl.com/Internet/Cache/caches.html
   Note: Read the section titled ‘Controlling freshness with the expires HTTP header’.

2 http://www.wdvl.com/Internet/Cache/cache2.html
   Note: Read the section titled ‘Cache-control HTTP headers’.

3 http://www.wdvl.com/Internet/Cache/cache2.html
   Note: Read the section titled ‘Validators and validation’.

Now that we know the relevant HTTP headers for Web caching, how can we generate these headers as part of our server responses?

One way would be to configure the Web server to send them out automatically. Apache comes with a module called mod_expires which allows you to configure your Web server to set the Expires header for outgoing responses. Another alternative is to use the mod_headers module to include a Cache-control header.

Another way would be to set these headers within your server-side scripts. Since these response headers are simply lines of text within an HTTP message, they can also be easily written out as part of your script’s output.

We will try both ways of generating these headers in the next activity.

Activity 6.10

Before you proceed with this activity, make sure that your browser cache is enabled and that its contents have been completely cleared out. Refer to Activity 6.8 for instructions on how to do this.

Part 1

1 Read the section titled ‘Apache cache control’ at http://www.sitepoint.com/article/effective-website-acceleration/4 in order to understand how cache control policies can be set using expiration dates for specific object types on the server.

2 Now apply the following directives to your local Apache Web server:

   ExpiresActive On
Run the Java Spy program so that it will display the HTTP request and response messages being exchanged on Port 80. Refer to Unit 2 for more information on using the Java Spy program.

Now access HTML pages with embedded images and dynamic pages from ABC Books, such as about.html. View the Java Spy display window and check whether expiration headers are now included in the HTTP response. Alternatively, you can also verify that the files you just accessed are now stored in your browser’s cache.

Aside from the Expires header, you can also use the Cache-control directive. Try adding this to your httpd.conf file and see its effect on the HTTP headers generated by the server:

```
Header append Cache-control "max-age=3600"
```

The only downside is that you cannot set different maximum ages for different types of objects.

**Part 2**

You can execute the sample script below to see how the Expires HTTP header can be generated using PHP code.

```
<?php
function setExpires($expires) {
    header ('Expires: ' . gmdate('D, d M Y H:i:s', time() + 
    expires) . 'GMT');
}
#Set the page to expire in 10 seconds.
setExpires(10);
# Show the current time in Greenwich Mean Time and allow #
# the user to view the page again using the browser #
# Refresh button. You should notice that the time is #
# updated only every 10 seconds.
echo 'This page will self-destruct in 10 seconds<br/>
';
echo 'The GMT is now ' . gmdate('H:i:s') . '<br/>
';
echo '<a href="" . testExpires.php . '"">View Again</a><br/>
';
?>
```


Now access this page, then reload it before ten seconds go by. Verify that the page is loaded from the cache by checking that the date
displayed has not changed. Now reload the page after ten seconds have elapsed. Verify that the page now displays the updated time, which means that a new version is retrieved from the Web server.

---

### Locating a cache

Most Internet caches are placed at sites where there is a serious network bandwidth load, like the link between the local area network and the Internet. Most likely, the cache will be located near the firewall or external router — in other words, as close to the network perimeter as possible. This will maximize the number of people who can use the cache.

Companies with large internal networks (e.g., several departments, each having its own LAN) can consider building an internal cache hierarchy. For example, child caches can be placed on departmental-level networks and they can be configured to forward cache requests to a company-wide parent cache. This concept of several proxy caches sharing their cached objects is known as *cooperative proxy caching*, and it will be discussed in more detail in the next section.

### Cooperative proxy caching

It’s highly possible for proxy caches to duplicate each other’s efforts, with several independent caches fetching and storing the same documents. Cooperative proxy caching consists of different methods whereby individual proxy caches share their cached objects with each other’s clients with the aim of achieving greater performance.

There are two alternatives which we will look at:

- broadcast queries; and
- hierarchical caching.

#### Broadcast queries

The basic idea of a broadcast query is that when a proxy receives a client request for a cachable object that is not in its local cache, it can be configured to request this object from all other proxies it cooperates with (Rabinovitch and Spatschek 2002). This method was introduced by the Harvest system and uses the Internet Cache Protocol (ICP) for exchanging queries and responses between cooperating caches.

#### Hierarchical caching

Another way for proxy caches to cooperate with each other is to become part of cache hierarchies. The next reading explains how hierarchical caching may work.
Reading 6.11


*Note:* Read the section on ‘Hierarchical caching’. (You need to scroll down a bit.)

Reading 6.11 shows how the broadcast query configuration can also be extended to a hierarchy. If no local cache has the document, the Harvest cache can be configured to check with one or more parent caches. This configuration would allow regional networks and service providers to have a cache, used by all the networks connected to it. Retrieving from the regional cache will be slower than from a local cache, but it should still be much faster than retrieving it from the origin server.

Caching issues

Surprisingly, not everyone thinks Web caching is such a wonderful solution to the Web performance problem. Caching provides many benefits, but at the same time, it comes with some unintended side issues that content providers and users alike must be aware of.

Maintaining cache consistency

It is possible for a Web cache to return out-of-date information to a user. Many Web servers do not even include headers such as `Last-Modified` or `Expires`, which would at least indicate how long a document can be kept in the cache.

On-demand validation is the only way to guarantee a cached response is up-to-date. However, validating each and every response will result in longer response times since requests still need to be sent out over the network to the origin server.

Processing overhead

An efficient, well-managed cache can greatly improve performance as seen by Web users. The performance gain is especially great for accessing large files over vast network distances. The use of caching, though, requires extra actions by the proxy that can be thought of as overheads. For example, the proxy cache must locate the document in the cache, copy the Web document into the cache, as well as carry out the initial document fetch from the originating Web server.

If the caching proxy spends too much time completing overhead tasks, it is possible for the addition of a network cache to degrade performance. The cache overheads can be high if there is no document locality, that is, no single document is more popular than any other. In this case, the
cache will spend a lot of time fetching documents from the originating Web server and the cache may be of little benefit. The key to an efficient cache is maximizing the number of cache hits and minimizing the cache overheads.

Some aspects of a cache’s performance can be improved by tuning, or by making adjustments to the cache’s decision-making process. The most important decision a cache makes is how to handle stale documents. However, many Web content providers do not include the necessary headers to indicate how long their pages can be kept and served from a cache via the HTTP Expires and Cache-control headers.

The Web server is then forced to guess when the document should expire. Web servers typically have configuration parameters that can be adjusted to optimize the guessing process. The Apache Web server has the following tunable cache configuration parameters:

1. CacheMaxExpire
2. CacheLastModifiedFactor
3. CacheDefaultExpire.

Many Web servers guess based on the time the HTML file was last modified, and that is what the CacheLastModifiedFactor parameter does. CacheMaxExpire and CacheDefaultExpire specify default document expiration values. You can briefly review in the ‘Apache Web server reference manual’ at http://localhost/manual/mod/mod_proxy.html to see how these configuration parameters affect the cache’s ability to guess which documents are stale.

**Cache size**

Another major tunable cache parameter for the browser cache is the size of the cache, which determines how many files are kept in the cache and for how long. This parameter affects the cache replacement policy that decides which file should be removed when the cache is full and a new, incoming file needs to be added to the cache.

Compared to the Web caching proxy, the Web browser cache supplies very crude controls to the user for handling document consistency. Both Netscape and IE browsers ask the user to specify how the browser will use the cache, through the Check for newer version of stored pages and Compare the page in the cache to the page on the network settings. The Never setting indicates the browser cache will always return a document to the user if it is in the browser cache. The Every time I view the page and Every visit to the page settings indicate the Web browser cache will always check with the origin server to see if there is a newer version of the document before it returns the cached copy. Other settings instruct the browser to periodically check with the original Web server, such as once a session, before using the document in the Web cache.
In the next activity you will witness how cache replacement decisions are made in the Web browser cache.

**Activity 6.11**

The main parameter that a user can adjust with the browser cache is the size of the cache. In this experiment you will see how the browser cache implements policies, based on cache size, that decide which files are kept in the cache and for how long.

1. Clear your browser cache by setting the size to 0 and then to a small size such as 200 Kilobytes.

2. Use the Explorer Find Tool to view the browser cache entries and select **Find Now** to update your view of cache entries as you download files from the Web.

3. Select a website that contains a few large files, such as an online art gallery. For example, access the homepage of the Museum of Modern Art in New York at http://www.moma.org.

4. Visit some of the graphics-heavy pages in their online store. Watch the contents of the cache change. At what point does one image file disappear from the cache? Which image file disappears? Did the first image loaded into the cache disappear? Or did the largest image, regardless of age, disappear from the cache?

---

**Information providers’ desire for control**

Web caches make things complicated for Web information providers because cache hits are not recorded in the origin server’s log file. Proxy caches also hide the identity of users and make them all appear to come from the same IP address. Server administrators may configure all the objects on their server as uncachable if they wish to have better control and accurate visitor counts. However, this approach is bad for user response times. Another less restrictive approach is to carefully identify which of your pages can and cannot be cached.

Some websites such as online news or stock trading may offer premium content on a subscription basis. In this case, the information provider will not want the information paid for once and then cached for many subsequent uses! For this reason, most caching proxy servers will not cache any information that requires user authentication (e.g., **WWW-Authenticate** header).

**Copyright**

Copyright laws give authors or creators the right to control how their original works are copied and distributed. However, copying is a
fundamental activity on the Internet. People like the Web because it lets them share information with each other, often at little or no cost. Web caches keep copies of objects returned by Web servers for future reuse. Information providers may therefore feel that caches violate their right to distribute and keep copies of their work. However, HTTP protocol does allow content providers to specify if, and how, their information should be distributed and handled by different types of caches.

Privacy

Most Web caches make a log entry for each and every request received from their users. A typical log entry includes the time of access, URL requested, and the requesting client’s network address. Cache administrators therefore have access to a large amount of personal data regarding their users, and they are accountable for the way this data is used. For example, there should be no reason for an administrator to examine specific requests by a certain user unless he/she is suspected of criminal or fraudulent activity.

Cache administrators can consider logging only partial information so that users can retain some privacy. For example, they may log only the first three octets of a client’s network address or they may drop the query string from a URL — information that appears behind the question mark (?) which may contain sensitive data such as usernames, passwords and account numbers.

Dynamic and personalized responses

Caching can be done more efficiently for static rather than for dynamic content. However, more and more websites are capable of returning dynamic, personalized pages that are constructed in real-time according to specific user requests. Some authors argue that as the percentage of dynamic content on the Web increases, Web caching may become less and less relevant. Problems may also arise when dynamic pages are cached and returned to users, providing them with stale and inaccurate information in some cases.

Self-test 6.5

1 Write down the HTTP request or response header for issuing the following instructions:

- Do not cache this object on a shared cache, only in a private cache.
- Cache the object, but revalidate it every time it is accessed.
- Specify that the expiration time for an object is one hour from the time it was accessed.
2 What are some of the advantages and disadvantages of a proxy cache using the **Expires** header versus the **GET** and **If-Modified-Since** header to maintain cache consistency?

3 Some websites send pre-expired responses. This means that the expiration timestamp of the object is equal to or less than the current timestamp. Explain why pre-expiration may be useful for origin servers.

4 What are some of the advantages and disadvantages of using hierarchical cache architecture versus a stand-alone cache on a LAN?

5 Explain why caching webpages can improve performance only if user access follows the principle of locality of reference.
Summary

This unit has covered a wide range of Web performance concepts. It has also provided you with the tools and techniques for identifying potential bottlenecks, investigating the causes of slow response times and optimizing website performance.

When measuring Web service performance, the most important performance criteria that must be gathered are latency and throughput. You looked at the actual Web metrics that reflect these two important variables, such as round-trip time, bytes per second and connections per second. Next, you learned the use of log file analysis, benchmarking strategies and system usage monitoring tools for evaluating server performance under actual and experimental workloads.

In order to improve Web server performance, the five areas that you investigated were: Web server hardware, Web server software, Web content, Web server architecture and network type and speed. You learned the basic requirements for the hardware and operating system on your Web server platform. You tried out some of the necessary httpd directives that can be configured to tune Web server performance. You also went through some important considerations when choosing an Internet Service Provider. And even if the client machine is beyond the control of information providers, you saw how adopting certain measures on Web content such as HTTP compression and pre-processing of dynamic pages can still reduce response times.

Web server architecture involves the organization of the documents and even the individual Web servers which may make up a website. This unit covered two structural configurations for boosting a website’s performance: scaling a Web service via load-balancing and replication and Web caching.

There is much variety in caching systems. A cache can cooperate with a system of other caches in delivering Web documents, it can be a stand-alone computer system, or it can be a network appliance on a LAN. Once a cache has been installed there are important management issues to understand such as the cache placement on the network, tuning, and cache consistency. Dynamically generated Web documents are difficult to manage well with a cache and secure documents shouldn’t be cached at all without additional security enhancements to the cache. All these techniques and network architectures have the potential to boost the performance of your website if they are understood and applied correctly.

Web server performance monitoring and optimization is a highly critical activity which must be uppermost in every Web administrator’s mind. To reinforce this idea, I leave you with this quote: ‘Users associate slow-loading pages with inferior quality products and services, compromised security, and low credibility’ (King 2003).
Suggested answers to self-tests

Self-test 6.1

1 Companies should use multiple, redundant name servers connected to different ISP networks and located in separate geographic regions. This ensures that their domain name can still be resolved even if there is network failure in a particular location or ISP.

2 TCP interacts inefficiently with HTTP because: (a) the use of a slow-start protocol to determine the optimum acknowledgement window can actually increase the round-trip time for HTTP transactions; (b) high overheads for setting up and tearing down a TCP connection are not justified by the relatively small file sizes of HTTP responses.

3 The performance-related enhancements in HTTP/1.1 are: (a) persistent connections — which allow multiple HTTP requests and responses to be transmitted over a single connection; (b) pipelining — which allows HTTP requests to be sent out together (e.g., in a batch) instead of one by one; (c) HTTP compression — compressed HTML files can be sent out by the server if a browser indicates that it can decompress them. Due to the nature of these improvements, they only speed up static pages, not dynamically generated content.

4 ABC Books can determine the peak traffic periods on the site and ensure that their Web server hardware is capable of handling the maximum workload. One possibility is to distribute the work to multiple Web servers only during peak seasons (e.g., Christmas) and then scale back to a single server afterwards. They can also analyse the content and functions that users are most likely to access, and the sequence in which they visit these pages during a particular session. For example, if they know that there is a 50% probability that users will search the book catalogue within a single visit, they can size the requirements for the database server more accurately.

Self-test 6.2

1 Log files are essentially historical information. They cannot be used to pinpoint performance problems in real-time. By the time reports are generated, they may no longer reflect the current situation (e.g., for performance and traffic). They can only be used to measure actual system performance. They cannot predict how the system will behave under different conditions or workloads.

Log files also do not measure round-trip time since the statistics are only gathered from the Web server's viewpoint. They may also contain misleading information. For example, the same visitor may be counted more than once if they access the site from different IP addresses. Caching proxy servers make it impossible for Web information providers to collect accurate statistics on their visitors and the files they are accessing.
2 Laboratory benchmarks are useful for assessing the limiting performance criteria in a system. Benchmarks also helpful in predicting results given changes to the system configuration such as server and network scaling, and the addition of interactive technologies. Perhaps the biggest limitation is related to the nature of the Web server workload. Some websites may spend a lot of time handling CGI or streaming multimedia requests, and this may not be reflected in the benchmark test used. Another limitation is that benchmark testing can only be done in a controlled environment (e.g., a LAN between your test clients and server system) and not over the Internet.

3 Stress tests use an extreme workload — one that creates an unusually large number or size of requests. Stress tests are useful in establishing the upper bounds of system performance and the maximum sustainable performance achievable by the Web server. They also help define the limiting performance factors in a system. Stress testing should only be done within a controlled experiment such as benchmark testing and should never be done in a production environment.

4 Average measurements over a long period of time do not adequately reflect performance at peak load. A finer granularity of time measurement (e.g., every five minutes or hourly instead of daily) is needed to reflect the Web server’s response during peak load.

5 Network bandwidth equals average size of a hit in bits * hits per second:

$$11\ \text{KB} \times 1024\ \text{bits per KB} \times 250\ \text{hits per second} = 2,816,000\ \text{bits per second}$$

Self-test 6.3

1 Here are the five most relevant tuning tips for my local Web server. I chose them because they involve making use of existing resources more smartly, instead of just blindly throwing more money and hardware at the problem:

- Remove unnecessary services and applications running on the server. Some services such as Remote Procedure Call (RPC) may be started up by default but are totally unnecessary.

- Use the latest versions of your Web server and operating system software. These versions often contain performance enhancements over their predecessors.

- Make fewer database queries. If you must query the database, make sure that tables are indexed according to the most commonly searched fields. For example, you can configure mySQL to record all queries which take a long time to execute (e.g., five seconds or more) using log-slow-queries.
• Preprocess content off-line whenever possible rather than generating dynamic content. If you must generate dynamic content, then use compiled scripts, server APIs or embedded interpreters instead of CGI.

• Get more memory (RAM) and more bandwidth. Note: This is the only tip here that requires money, assuming your server runs on open-source software. ☺

2 Here are the effects of various Web server configurations.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setting MaxSpareServers and MinSpareServers to a large number</td>
<td>Negative; resources may be consumed by the launch of unnecessary servers.</td>
</tr>
<tr>
<td>Setting MaxRequestsPerChild to 0 (e.g., unlimited requests per child, which means that child processes never expire)</td>
<td>Negative; servers are still kept running even when the workload has been reduced. Servers also tend to occupy more and more RAM since they grow to accommodate whatever request they are serving, and they do not release any memory until the server process dies.</td>
</tr>
<tr>
<td>Setting KeepAliveTimeOut to a larger number</td>
<td>Negative; server processes may be kept occupied waiting on connections with idle clients.</td>
</tr>
<tr>
<td>Setting HostNameLookups to ON</td>
<td>Negative; DNS lookups have high latencies.</td>
</tr>
</tbody>
</table>

3 A few examples of overheads on a Web server system are:

• virtual memory management — swapping pages in and out of memory;
• process creation and thread creation;
• creating and destroying TCP/IP sockets (connections); and
• running the windowing system.

Self-test 6.4

1 Users can be directed to an appropriate mirror server by:

• letting them manually choose from a list of alternate servers on a webpage;
• running a script which automatically redirects them to a mirror server based on their domain name; and
• load balancing.
2 Disadvantages with mirroring are that:

- some replicated documents may be rarely accessed, leading to wasted disk space;
- resources (e.g., time, money and personnel) are required to keep mirror sites updated, and aside from webpages, databases and streaming media may also need to be mirrored; and
- the mechanism used for directing users to a mirror server may not balance the workload appropriately among alternate servers, even when load balancing is used.

**Self-test 6.5**

1 Here are the HTTP headers which will issue the following instructions:

- Do not cache this object on a shared cache, only in a private cache:
  
  ```
  Cache-control: private
  ```

- Cache the object, but revalidate it every time it is accessed:
  
  ```
  Cache-control: max-age=0
  ```

- Specify that the expiration time for an object is one hour from the time it is accessed:
  
  ```
  Cache-control: max-age=3600
  ```

2 The advantage of the `Expires` header is that it provides precise accurate information to the cache, as supplied by the document’s author, about the date when the webpage is no longer valid. Therefore, cache users will not get a stale document if the author updates the document accordingly. The disadvantage of the `Expires` header is that the author has to explicitly specify this. The advantage of the conditional `GET` and `If-Modified-Since` headers is that an expiration date can be applied to a document even if the author did not supply one. The expiration date is calculated by an heuristic rule that says if the document has not changed recently, it is not likely to change anytime soon. The disadvantage of the conditional `GET` and `If-Modified-Since` headers is that the expiration date may not be accurate and proxy users can still get stale documents if the proxy caching server does not check frequently enough to determine if the document has been recently modified.

3 Pre-expiration allows origin servers to closely track accesses to their site but still make their content cachable. Here is an example of how the `Date` and `Expires` header can be set in order to pre-expire an HTTP response:

```
Date: Sun, 01 Apr 2004 18:32:48 GMT
Expires: Sun, 01 Apr 2004 18:32:48 GMT
```
The cache will then validate the response next time someone requests the page. There is still a performance gain when the server returns HTTP 304 Page Not Modified instead of the entire object. This allows the cache to serve the page using its local copy.

4 It is possible for a well-configured hierarchical cache system to improve document retrieval performance over stand-alone caches. A document that is a ‘cache miss’ in a local cache and a ‘cache hit’ in the regional cache, will typically be faster to retrieve than a ‘cache miss’ in a stand-alone cache. However, for a request that is both a cache miss in the local and regional cache, it will take a lot longer to retrieve the document from the originating Web server and return it to the user. If a hierarchical cache is poorly configured and both the regional and local cache experience a lot of ‘cache misses’, the performance for a hierarchical cache can be worse than the performance of a stand-alone cache. Another disadvantage of a hierarchical cache system is its complexity.

5 If webpage access does not exhibit locality of reference, the cache will constantly be overwritten with newly accessed files. Very few of the files stored in the cache will ever be served locally. Since the cache serves as both a client and a server, the overhead of a proxy can approach 200% if there are very few cache hits.
References


http://www.oreilly.net.


