Reduce network traffic with Web caching

Server Web caches speed access to Web pages and ease network traffic

Rawn Shah

Find out three ways to use Web caching to route Web traffic to your site more efficiently. Whether you run an extranet, intranet, or Internet site, Web caching can give you more control of your resources. Learn what hardware you need and what caching software to consider.

Like the mass transit systems that move groups of people between popular destinations, Web caching systems do the same for URL requests for popular Web sites. You can use Web caches to put users on the express track to their destinations.

Web caching stores local copies of popular Web pages so users can access them faster. A cache aggregates all the individual requests for a Web page and sends a single request as their proxy to the origin site, as the requested Web site is called. (But don't confuse a Web cache with a proxy server. The latter serves as an intermediary to place a firewall between network users and the outside world. A proxy server makes your outgoing network connection more secure, but it does little to reduce network traffic.) When the cache receives its copy of the contents, it then makes further copies and passes them on to the requesting users.

Caching in

Web caches can help lighten the load on a Web server by reducing the number of incoming requests; browsers retrieve portions of data from the cache rather than directly from the server. However, most Web content providers do not have access or control of which users or how many users arrive at their site. The cache server needs to sit nearer to the user end than to the Web server end. (Web load-balancing schemes distribute the incoming load across multiple servers at the Web content provider end, but that's a whole other story.) The most obvious beneficiary of Web caching is the user, who avoids some traffic snarls when browsing.

The network administrator and the remote Web site also benefit. According to the National Laboratory for Applied Network Research (NLANR), large caches with lots of clients may field as many as 50% of the hits that would otherwise travel through a network individually to the origin site. A typical cache could easily field about 30% of the intended hits, says the NLANR's 1996 research. Thus, statistically speaking, a Web cache could eliminate at least 30% of the Web traffic.
that would normally be going out over a WAN line. If you're paying dollars for megabytes, Web caching can then save you considerable sums in a relatively short time. Even if you have a flat-rate WAN connection, caching improves customer satisfaction levels, because it speeds access for all. For a larger perspective on how caching performs a public server for all users, read about the Global cache projects later in this article.

PCs have memory caches for code that's called often, and most browser programs have local caches that store recently surfed Web pages either in memory or on disk. A Web cache also stores frequently accessed Web pages, but it operates on a grander scale.

**Webrouting with cache servers**

In addition to reducing outgoing traffic by bundling duplicate requests from browsers, Web caches act like custom-dispatched express trains to solve the problem of Web routing: how to send Web traffic efficiently over a network. While Internet Protocol routing directs low-level traffic (individual IP packets) irrespective of the data contents, Web routing directs application-specific HTTP traffic across the network. Because Web traffic constitutes most of the Internet traffic, improving Web routing can improve the overall performance of the Internet.

Web routing depends upon IP routing because Web traffic flows only along the paths defined as IP routes. However, a single Web flow can change from server to server as it is redirected by different Web routers. A Web server can use the Redirect command of the HTTP protocol to send Web requests to other servers for processing. Web caches themselves redirect client and server traffic locally or to other caches to provide faster access to pages. Finally, load-balancing devices for Web servers can redirect incoming client requests to a group of servers in the same location or in other network locations to evenly distribute the incoming requests among the servers. You can think of all these devices as Web routers directing HTTP traffic.

The process of Web routing with cache servers begins after the Web request leaves the client browser workstation:

1. The cache server receives the request one of three ways: the request can be sent directly to the server; the server can actively monitor network traffic and pick out requests from the flow; other network devices can pick out the traffic and send it to the cache server.
2. Then the cache resolves the Web request. It must determine if the requested page is stored within its cache database. If not, it checks its partner cache servers, if any, for the requested data.
3. Finally, the cache server returns the data to the client, from its own database, from a partner's database, or from the original Web server.

Just as public transit systems use buses, trains, trollies, shuttles, taxis, and ferries, this three-step Receive-Process-Return process has been implemented in various forms.

**Receiving the Web request**

The most basic method for diverting requests to a cache is to configure the browser to point to the cache as its proxy server, an option on most popular browsers. The client browser then sends a request for a URL directly to the cache server to retrieve a document. This method ensures
that the cache does the greatest possible amount of request processing: every request goes through the cache server. One downside of this method is that you cannot always control whether the browser uses a proxy; thus, clever users who understand that this is a typical configuration option may try to bypass the proxy. Another downside: When you have hundreds or thousands of desktops and Web browsers to configure, this method can turn into a management headache.

Transparent proxy caching also diverts all traffic to the cache server. A cache server sits directly on the data path between the clients and the remote Web sites and intercepts all outgoing requests. Because the cache examines every packet of data to look for Web requests, it serves as an advanced packet filter.

External packet filters and IP Layer-4 and Layer-7 switches can also handle and route client requests. These devices examine the packets that are going out of the network to identify Web requests and redirect them to the cache server. A packet filter can examine any or all of the contents of the packet and, based upon some predefined policy, redirect the traffic appropriately.

At the transport layer, a Layer-4 switch redirects TCP or UDP to an appropriate destination; because all HTTP traffic is TCP-based, most such traffic is passed on first.

At the application layer of the ISO stack, a Layer-7 switch looks only for application-specific protocols, such as HTTP, to direct to appropriate destinations.

Comparing methods for handling requests

Configuring every client Web browser can be a tedious task; transparent proxy caches are more practical for deployment on large networks or in organizations without strict control of the network. For example, an ISP can use transparent proxy caches for its dial-up modem clients without their knowledge. Such a cache server would have to sit closest to the outgoing WAN connection to provide the maximum benefit.

Transparent proxy caches work much more slowly, however, because the cache server has to process every single IP packet that goes through the network to look for Web packets. Thus transparent proxy caches require the fastest processors and fast dual network links.

Using external packet filters or layer-specific switches optimizes the function of the device. In fact, some implementations have their own protocols that monitor the activity of multiple caches for the purposes of load-balancing.

Processing the Web request

Once the cache server receives a Web request, it checks its database to see if it has the contents of the requested page stored somewhere.

Web caching originally began as a single-server system that contained all the data of the cache. Although that's effective, cache servers tend to grow large. A single server runs out of disk space to store the requested pages or cannot process the incoming requests fast enough. Eventually single-server schemes gave way to distributed cache servers working either hierarchically or in
parallel, or both. These servers balance among themselves the amount of cached information they contain, placing the most commonly requested data at the top of their hierarchy for the most people to see and the least commonly requested data at the bottom, closer to the specific users who need them.

Some cache server software actually works as extensions to existing Web server products. In such a case, there’s no point in logging Web access entries, so the administrator should either disable or limit logging to the server log file. A cache contains continuously changing information, and unless you know what each cached entry contains (which actual Web site it goes to), you won't know where the client was going. Cache logs may also get fairly large, because all your users will be contributing to it. It can consume disk space as quickly as third-graders do candy.

**Single-level caching**
A cache server is essentially a proxy Web client that stores a lot of pages locally. The server responds to requests by sending along the requested Web page if it’s available.

A successful retrieval from the local cache is called a cache hit, and an unsuccessful one is called a cache miss. In the case of a cache miss, the server begins its own access to the requested URL. Such a first-time access to a page forces the cache server to contact the origin Web server that hosts the page. The cache server checks to see if the page can be cached, retrieves the data to cache locally, and, at the same time, passes through the contents to the client. The user may never realize that the cache is between the client and server except in special circumstances.

A single cache server is the cheapest solution for improving Web routing, but its effectiveness is limited by the capacity of the server. By combining a firewall, an IP router, and a cache together, vendors have created a single-box solution that works well for small office intranets. To go even cheaper, you can build a device with similar capabilities using a PC, the Linux operating system, and open-source software available publicly.

**Parallel and load-balanced caching**
A single cache server can handle only so many requests at a time, and even pumping up the machine with memory, disk space, and processors takes its capacity only so far. A better way to handle high-volume requests is to keep several cache servers running in parallel, handling requests from the same clients or different groups of clients. These parallel cache servers usually contain identical data and communicate changes among themselves.

An enhancement to the parallel-server method involves creating a load-balancing system for the parallel servers. All the servers handle the same group of clients and balance the load of incoming requests among themselves.

**Multilevel caching**
A multilevel cache spreads the cached data contents across several servers across the network. The top level caching server holds the most commonly accessed pages, and the lowest-level caching server holds the least commonly accessed pages. The various levels combine in a network of cache servers called a Web caching mesh. The caches communicate among
themselves, using HTTP and special cache-coordination protocols, to divide the contents appropriately and maintain consistency among the servers.

Multilevel caching works almost the same as caching with single-cache servers. However, if there is a cache miss at one server level, the request is propagated up to the next higher level to see if that cache contains the data. Only when the request hits the top level and still encounters a cache miss will the cache server go directly to the origin Web site to retrieve the data. (You can customize this configuration of multilevel caching. Typically it looks at the nearest cache server before going up the chain to the top-level server, which might be several hops away.)

Multilevel cache systems work very well for a very large number of clients (in the 10,000s or 100,000s) accessing the system. Furthermore, if your many clients are spread widely across a WAN or the Internet, it's an even better solution.

**Returning the Web request**
Returning the results of a cache is currently still a simple process. Basically, the cache that contains the requested data examines the request packet, takes the source IP address, and sends the data to the client under the guise of the identity of the origin Web server.

**Choosing protocols and options for multiple servers**
Coordinating the contents of a cache among multiple servers is a challenge. As soon as you add a second cache server to the system, you encounter this problem: how do you maintain the consistency among the multiple servers that should contain identical data? If you add multiple levels of cache servers, you have to ask two other questions: how do you know what the other caches contain, and how do you redirect the request to the appropriate cache?

This is where cache protocols come in. There are three main types:

- Query protocols send messages to other caches in a multilevel system to discover if they contain the needed data.
- Redirect protocols forward the client request to the cache server in the multilevel system that contains the needed data.
- Multicast protocols combine Query and Redirect protocols using multicast network communications.

Multicast cache protocols work in concert with all cache servers at the same time. Multicasting is the ability to create a virtual network of computers that can communicate directly with every other member at the same time. Multicasting is a function of the IP network protocol, with the help of special multicast routers and protocol stacks. With such cache protocols, a cache server can query all the other servers at the same time to find out if they contain the needed data. In addition, a client request sent to such a multicast group is automatically sent to all members, obviating any redirection. Within the group, one of the cache servers recognizes the requested URL as within its domain of responsibility and sends the data appropriately.

The problem with multicast protocols is that they are still not very popular. What's more, multicasting over the current Internet Protocol isn't really efficient because all of the Internet
is connected by a mass of single point-to-point, or unicast, links, which defeats the purpose of multicasting. Still the software methods exist, and within intranets it is possible to set them up. The future generation of the Internet Protocol, called IPv6, allows real multicasting to take place, but it will be some time before it's widely implemented.

Setting protocol options for cache servers

There are four options for caching protocols:

- The Internet Cache Protocol (ICP) is the first cache query documented as an informational standard by the Internet Engineering Task Force. It was developed during the research conducted in 1996 by the Harvest project, one of the early Web-caching projects. In a multilevel cache, ICP sends queries between the cache servers to check for specific URLs in other caches in the mesh. Unfortunately, ICP becomes inefficient beyond a certain number of distributed cache servers. If you're setting up one or two caches, this limitation of ICP does not pose a problem. On the other hand, if you're setting up a large multilevel cache with more than ten servers, ICP caches will spend too much of their time propagating changes and thus reduce efficiency. ICP also contains no real security to protect the communications between the cache servers.

- The HyperText Caching Protocol (HTCP) is a better query protocol that is used to discover cache servers on local networks and to inquire if URLs are contained on the servers. It includes the HTTP headers from the original client request so that the cache server may process them, if necessary, as part of the request.

- The Cache Array Routing Protocol (CARP) is a redirect protocol for a multilevel cache system. Each cache is programmed with a list of all the other cache servers in the system. The cache server uses a hash function that maps the URL to a given cache server. It then sends a CARP message to the other cache server containing the original HTTP request to fulfill. Microsoft's Proxy Server implements CARP.

- Cisco's proprietary Web Cache Control Protocol (WCCP) handles request redirection to a cache mesh from a router. One of the cache servers can send a WCCP message to the router to define the mapping between URLs and cache servers. The router processes outgoing packets and looks for HTTP traffic; it then uses a hash function to determine which cache server should process the URL in each request and redirects the traffic to the server with WCCP.

Selecting hardware for cache servers

Essentially, a cache server is a heavy-duty network file server. Unlike a proxy or firewall server, which can run on fairly low powered machines (even 486 machines can work well as firewalls), a cache server needs processing power and speed.

To be most effective, a cache server needs fast network connections to the internal LAN network and the external WAN. Typically, plan for a cache storage capacity of several gigabytes on disk, as well as at least 128 MB of RAM, preferably gigabytes of RAM. By increasing the RAM storage, you directly increase the performance of the system, because direct accesses to physical memory work much faster than accesses to disk-stored caches.
Also, a fast processor can help, but a multiprocessor system, even with slower CPUs, can perform better by handling more requests simultaneously. Cache server administrators recognize that RAM and disk storage are the most important performance factors.

A Linux-based cache server running on a dual-processor 350-MHz Pentium II system with 512 MB of RAM, 25 GB of SCSI disk space, and dual 100-Mbps Ethernet connection -- an estimated price between $2,500 and $5,000 -- should be able to handle one to two million requests a day, serving between 1,000 and 10,000 users.

**Selecting software for cache servers**

Typically, the cache server does not need any intervention from a server administrator, so the best choice is a fairly stable, reliable platform that can run unattended. Both commercial and freeware cache software is available.

### Cache server software products

<table>
<thead>
<tr>
<th>Software</th>
<th>Vendor/Developer</th>
<th>Caching Type</th>
<th>Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Web Server caching module*</td>
<td>Apache Information Services</td>
<td>Single</td>
<td>AIX, BSD/OS, Digital UNIX, FreeBSD, HP-UX, IRIX, Linux, NetBSD, NextStep, SunOS, Solaris, SCO UNIX, Windows NT</td>
</tr>
<tr>
<td>BorderManager FastCache</td>
<td>Novell</td>
<td>Single, multilevel</td>
<td>NetWare</td>
</tr>
<tr>
<td>Cache Engine</td>
<td>Cisco</td>
<td>Single, multilevel, load-balancing</td>
<td>Custom hardware</td>
</tr>
<tr>
<td>CacheFlow Series</td>
<td>CacheFlow</td>
<td>Single, multilevel, load-balancing</td>
<td>Custom hardware</td>
</tr>
<tr>
<td>CacheRaq 2</td>
<td>Cobalt Networks</td>
<td>Single</td>
<td>Custom hardware appliance</td>
</tr>
<tr>
<td>DeleGate*</td>
<td>MITI ETL</td>
<td>Single, multilevel, load-balancing</td>
<td>AIX, EWS4800, HP-UX, HI-UX, IRIX, NextStep, NEWS-OS, Digital UNIX, Solaris, SunOS, BSD/OS, FreeBSD, Linux, NetBSD, OpenBSD, Windows 95/NT, OS/2</td>
</tr>
<tr>
<td>Internet Caching System</td>
<td>Novell</td>
<td>Single, multilevel, load-balancing</td>
<td>Custom hardware</td>
</tr>
<tr>
<td>Jigsaw caching proxy module*</td>
<td>World Wide Web Consortium</td>
<td>Single</td>
<td>Java</td>
</tr>
<tr>
<td>NetCache</td>
<td>Network Appliance</td>
<td>Single, multilevel, load-balancing</td>
<td>Custom hardware appliance</td>
</tr>
<tr>
<td>Proxy Server</td>
<td>AOL/Netscape</td>
<td>Single, multilevel, load-balancing</td>
<td>AIX, HP-UX, IRIX, Solaris, Windows NT</td>
</tr>
<tr>
<td>Proxy Server</td>
<td>Microsoft</td>
<td>Single, multilevel, load-balancing</td>
<td>Windows NT</td>
</tr>
<tr>
<td>Squid*</td>
<td>NLANR</td>
<td>Single, multilevel, load-balancing</td>
<td>AIX, Digital UNIX, FreeBSD, HP-UX, IRIX, Linux, NetBSD, NextStep, SunOS, Solaris, SCO UNIX, OS/2</td>
</tr>
<tr>
<td>Traffic Server</td>
<td>Inktomi</td>
<td>Single, multilevel, load-balancing</td>
<td>Digital UNIX, FreeBSD, IRIX, Solaris, Windows NT</td>
</tr>
</tbody>
</table>

*Freeware or open-source software
Global cache projects

On the global scale of the Internet, Web caches can reduce the overall burden of traffic through the numerous high-speed links between top-tier Internet service providers. By providing a cache hierarchy that maps to the provider's network topology, ISPs can create regional cache areas supported by groups of cache servers. Each region's cache contains data from Web sites in other regions. Thus, a programmer in California who wants to look up a page located in France would automatically be directed toward the cache server in California that holds a copy of the needed page, rather than pulling it straight down from halfway across the world.

Because of the public benefit of reducing Internet traffic, the National Science Foundation (NSF) has supported research projects that enable large-scale caching systems. One such project from the NLANR is currently investigating a multilevel cache system based on the national supercomputing centers located in public institutions and universities around the United States. The project began when a study in 1993 suggested that several strategically placed FTP servers could reduce the overall traffic on the then-NSFNet backbone by 44 percent.

Internet topology has changed significantly since 1993, but the basic precepts still hold true. In fact, the IRCACHE project from the NLANR has cache servers located at or near the recommended central exchange points.

In the United Kingdom, a similar project is part of the Joint Academic Network (JANET), a national academic network service, and the next generation SuperJANET system. This national cache service is also available for public use or cache peering arrangements.

Both the IRCACHE and JANET Cache offer open participation from the public to combine their cache servers with the projects' distributed system. This gives the benefit of a global cache system that empowers that of your own, which in turn, speeds the access of your users. For more details on participating, visit the sites listed in Related topics.
Related topics

- Find more network statistics at The National Laboratory for Advanced Network Research
- For info on the legal ramifications of using caches to access sites, read this synopsis of the Digital Millennium Copyright Act
- Explore the Squid FAQ
- Find out more about how to install Linux programs for turning Squid implementations into transparent proxy caches:
  - Transparent Proxy Caching with Squid
  - The IP Filter package
- For info on packet filtering on Linux, read HowTo for ipchains
- If you're interested in participating in Global caching systems, take a look at:
  - The NLANR IRCACHE Project
  - The National JANET Web Caching Service
- For details about the cache server products described in this article, visit the links in the commercial and freeware cache software table

© Copyright IBM Corporation 1999
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)