Site-Based Approach to Web Cache Design

A site-based approach to Web caching tracks documents by site rather than individual document names or URLs, bringing different benefits to several different types of applications.

Web caching is widely viewed as an effective way to improve Internet performance. Web cache servers temporarily store Web documents. Because cache servers are typically located near the clients, using them reduces retrieval latencies and the number of requests reaching Web servers, thus reducing overall network activity.

Most Web cache servers make cache decisions based on document names or URLs. Although it seems natural to keep statistics on individual documents, it is not the only option. We propose a new basis for caching decisions: a site-based approach. With this approach, cache servers consider the Web site a document belongs to, rather than the document itself. Depending on the application, a site-based approach can offer many benefits, including proxy load reduction, higher prediction and hit ratios, and fewer memory requirements.

When we first proposed this idea in 1999, we were unaware of any previous related work. However, we have since found that some researchers implicitly applied a site-based approach, though they did not focus on it. Here, we describe the site-based approach and consolidate previous efforts, discussing both the approach’s benefits and its successful applications.

Caching Overview

There are three popular research areas in Web caching: cache replacement policy, prefetching, and multiple cache server design. Another less active but important area of research is proxy load reduction.

Proxy load reduction is crucial because as Web traffic increases, so too do the number of client requests reaching the proxy server. This in turn increases the proxy’s workload and can cause connection time-outs, incoming request refusals, and user delays. A simple way to reduce the proxy load is to upgrade the overloaded proxy. Of course, installing multiple cache servers also improves performance.

Cache replacement policy evaluates existing cached documents to determine which documents should be replaced when the cache is full and new documents arrive. The goal is to manage limited
cache space. A cache server (such as a proxy cache server) typically removes one or more cached documents that are unlikely to be accessed in the near future to make space for new documents. The policy objective is to optimize the cache space usage to obtain maximum cache hit ratio. Commonly used policies include Least Recently Used (LRU), which evicts documents according to oldest last-access time, and Least Frequently Used (LFU), which evicts the documents that have been accessed the least frequently.

Prefetching individually examines each hyperlink on a Web page to reduce the latency users experience. In the most basic client-server environment, a client-side application (such as a Web browser) prefetches Web documents from remote Web servers before the user requests them. When a user requests the prefetched documents, the local cache satisfies the request and thus decreases latency. To predict which documents the user will access, the client can simply check hyperlinks in the HTML document that the user is currently viewing. The client-side software then evaluates those links using some criteria — such as document popularity and size — to determine which links to prefetch. To do that, the software must maintain information from the user access history. Although prefetching can reduce latency, it also increases Internet congestion because it typically fetches more documents than users will actually access.

Multiple cache server design (multicache design) focuses on approaches for exchanging caching information and redirecting requests or documents between cache servers. The goal is to cooperatively form a large cache system that supports highly concurrent request streams. The caches are typically configured in a hierarchical way, first suggested in the Harvest Cache. This architecture consists of proxy servers hierarchically arranged in a tree-like structure. This lets downstream proxies (proxies closer to the clients) benefit from the cache of upstream proxies. When a cache server receives a request for an uncached document, it will ask if its siblings and parents in the hierarchy have the document in cache.

The Site-Based Approach
Table 1 summarizes the benefits of our site-based approach for addressing each of the main research areas.

Proxy Load Reduction
The high volume of Web traffic frequently overloads Internet proxy servers. To solve this problem, we propose the Site-Based Dispatching (SBD) technique, which reduces the unnecessary requests reaching proxy servers. When a proxy server receives unnecessary miss requests, it stores copies of requested documents that will never be accessed again, thus wasting computing power and system resources.

To implement the SBD technique, you must have a browser plug-in or a browser that is modified to communicate with the hot-site manager. The SBD technique’s basic principle is that if the browser knows the unnecessary requests in advance, it can forward these requests directly to the remote server via the Internet gateway, bypassing the proxy. (The Internet gateway can act as a router or a firewall that provides internal clients with Internet access.) To achieve this, we use site information obtained from client requests. Based on this information, the browsers forward to the proxy only the requests belonging to hot sites; it sends the rest to remote Web sites directly.

To identify unnecessary requests, we use a regularly updated hot-site list of the most popular Web sites. Each browser has a copy of the hot-site list, which it checks each time a user enters a URL. If the requested Web site appears in the list, the browser forwards the request to the proxy. Otherwise, it bypasses the proxy and retrieves the document from the corresponding remote Web server directly (see Figure 1, on the next page). Our rationale for this approach is that if a document belongs to a hot site, there is higher chance that it is a hot document. And, because it is a hot document, there is a higher chance that it has been cached in the proxy. This rationale is supported by the fact that most requests to hot sites target only their hot documents.

The hot-site list requires regular updates to both add new sites and omit those no longer popular with the clients. To maintain this global picture,
we use a hot-site manager that can run on any local network host. Updating the hot-site list requires cooperation between the browsers and hot-site manager. When a browser is launched, it retrieves the latest list from the hot-site manager and stores it locally. The browser also records the names of user-requested Web sites in a log file. When the browser is terminated, it sends the log file to the hot-site manager. Based on these client machine histories, the hot-site manager periodically constructs a new hot-site list that reflects current Web site popularity.

Our study of the SBD technique’s performance shows that it redirects 23 percent of the total client requests to bypass the proxy and reduces the cache hit ratio by less than 1 percent. More details on these findings and the SBD technique are available elsewhere.3

Cache Replacement Policy

There are at least two major components to managing space in a cache server: the replacement policy and the storage architecture, which organizes cached documents in the local disk. As the “Storage Architectures” sidebar explains, existing architectures require a sacrifice of either resources or administrative control. Our approach resolves this trade-off, offering the control of the original CERN httpd design, without requiring the additional, resource-intensive database characteristic of more recent storage architectures.

To design our cache replacement policies, we use the Least Recently Used (LRU) site-based approach. Figure 2 shows how document- and site-based LRU algorithms manage the link list. In document-based LRU policy, a document list links cached documents (Figure 2a). When a new document is requested, the system inserts it at the top of the document list. Also, each time a user accesses a cached document, it moves to the top of the list. As time goes by, the least recently accessed documents migrate to the bottom of the list. When a replacement must be made (such as when request 9 arrives in Figure 2), the system removes the last documents in the list. In this example, the system removes documents A/1.html, B/1.html, and C/1.html, freeing up 3 Kbytes of cache space for the new 2.5-Kbyte document, E/1.html.

In site-based LRU, the system links site names to form a site list (Figure 2b). When a user requests a document belonging to a new site, the system inserts the site name (rather than document name) at the top of the site list. It then stores the request-

Reference

ed document under the newly inserted site. Note that in the site-based approach, the system stores in cache only a limited number of a site’s documents, specified by a parameter Max_Docs. When an incoming document arrives and its site already has Max_Docs documents cached, the least recently used site document is replaced with the new arrival. This prevents caching the cool documents in a hot site. Each time a user requests a document, its corresponding site moves to the top of the site list. As time goes by, the least recently accessed sites migrate to the bottom of the list. When a replacement is required (such as when request 9 arrives in this example), the system purges the last sites in the list and all related cached documents to accommodate the new document. In our example, the system removes all the documents in sites C and D.

Because each node in the list has a dedicated sublink list, site-based LRU allows the system to perform actions (such as delete all cache files for a given site) and queries (such as asking which documents are cached for a given site). While the benefit of using site-based LRU is clear, a question remains: Will using site information alone degrade the cache performance?

To answer this question, we ran a simulation based on a real cache access log. The goal of our simulation was to compare the cache hit performance of document-based LRU to that of site-based LRU. We downloaded the latest publicly available access log from the San Jose (SJ) proxy cache at the U.S. National Lab for Applied Network Research (ftp://ircache.nlanr.net/Traces). We extracted only successful Get requests and URLs that did not include query strings. Table 2 shows the characteristics of the trace. We obtained the Maximum Hit Ratio by averaging the hit ratio of each day that the system used an infinite-size cache. For simplicity, we did not consider cache consistency. All simulations were initiated with an empty cache, and the simulation program reported the hit ratio for each day of the log.

Figure 3 (next page) shows the simulation results: Under all cache sizes shown, site-based LRU performed slightly better than document-based LRU. Figure 4 reports the hit ratio for document-based LRU and site-based LRU with Max_Docs set at 3,000 per day and a 1-Gbyte cache. As the figure shows, site-based LRU performed slightly better than document-based LRU on all recorded days. Our results further verify the feasibility of using a site-based approach in practical proxy servers.

Figure 2. Link list management using (a) document-based LRU and (b) site-based LRU. The cache was initially empty, and fills up after serving requests 1 to 8. Solid lines show the current link list: One or more cached documents must be removed for the new request.

Table 2. Summary of SJ access log characteristics over a 14-day period in January 2001.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total number of requests</td>
<td>996,775</td>
</tr>
<tr>
<td>Total size of requested documents</td>
<td>14 Gbytes</td>
</tr>
<tr>
<td>Total number of unique documents</td>
<td>623,465</td>
</tr>
<tr>
<td>Total size of unique documents</td>
<td>6.8 Gbytes</td>
</tr>
<tr>
<td>Total number of unique Web sites</td>
<td>25,261</td>
</tr>
<tr>
<td>Maximum hit ratio</td>
<td>54 percent</td>
</tr>
</tbody>
</table>
Prefetching

We use the Top 10 approach to site-based prefetching. Top 10 requires cooperation between server and client-side entities to complete prefetching operations successfully. The server side must regularly prepare a list that records its most popular documents. To indicate how many hot documents to include, it uses a $\text{Top}_N$ parameter. The client side keeps its own access history. After a predefined time interval, a client checks to see if it has made a sufficient number of Web server requests (larger than the value specified in the $\text{Site~Select~Threshold}$ parameter). If so, it prefetches on the server, asking it to send its $\text{Top}_N$ documents. As Figure 5 shows, in the Top 10 approach, only the client prefetches documents from the hot Web sites, which are determined by the $\text{Site~Select~Threshold}$ parameter. The approach is site-based because prefetching decisions are made site by site.

Although a previous study focused on prefetching performance in relation to the $\text{Top}_N$ parameter, it did not explore the $\text{Site~Select~Threshold}$'s impact. Because the $\text{Site~Select~Threshold}$ parameter is significant in the site-based approach, we studied its effect on prefetching performance by re-running a trace-driven simulation of the Top 10 approach.

For our simulation, we downloaded the latest publicly available access log from the HTTP server of UC Berkeley's Computer Science Division (http://www.cs.berkeley.edu/logs/http). We used a 14-day section of the log starting from 4 December 2000 and containing 2,943,605 client accesses. We postprocessed the original log to remove the query requests and the requests from local clients. We set a time interval to 500,000 client accesses and assumed that the prefetched documents stayed in the client cache throughout. When the time interval expired, the Web server updated its $\text{Top}_N$ documents and the client reset the total number of requests targeting the servers and removed its cache content.

Figure 6 shows how the $\text{Site~Select~Threshold}$ parameter affects the prefetching performance. We define the prediction ratio as the total number of user requests that the prefetched documents can satisfy compared to the total number of prefetched documents. As the figure shows, in most cases, higher $\text{Site~Select~Threshold}$ gives a higher prediction ratio. We expected this because the $\text{Site~Select~Threshold}$ determines how hot the selected sites are. If hotter sites are selected, the chance that the prefetched documents will be accessed later is also higher.
Multicache Design

The site-based approach can also help in multicache system design. Researchers have proposed a Web caching scheme featuring dynamic caching hierarchies, which is different from the Harvest’s static caching hierarchies. In static caching hierarchies, proxy servers send all the miss requests to fixed higher stream proxies for service. The proxy servers in dynamic caching hierarchies use request-forwarding tables to decide which proxy server to forward the miss request to. Thus, a proxy will forward different requests to different higher stream proxies based on the table’s contents.

In dynamic caching hierarchy design, Web servers first invite proxy servers to be their caching representatives (C-Reps). If a user issues a request to a client-side proxy server that is not a C-Rep, the proxy forwards the request to a server C-Rep. When a C-Rep receives a request (from other proxies or from users), it handles it in the same way as a standard proxy server. Figure 7 shows how a C-Rep (P4) passes requests to other C-Reps.

Figure 7 also shows P4’s request-forwarding table, which contains information for all original Web servers. Although such tables are easy to implement and can serve all client requests, Chiang and his colleagues point out that they typically have high memory requirements, which makes them impractical and inefficient (at 0.5 Kbytes per entry, a table with 400,000 entries requires 200 Mbytes of memory). To solve this problem, they introduced a partial request-forwarding table, which stores only the entries of the most popular $p$ percent of the original servers. This partial table thus has a much smaller memory requirement. The drawback of doing this, however, is that in the long run, the partial table limits the system to serving only $c$ percent of the total client requests. The proxies send the remaining $(100 - c)$ percent client requests directly to the remote Web server, which can lengthen response time and thus impact system performance.

Parameter $p$ is thus a design parameter you must choose carefully. Chiang’s multicache design does use a site-based approach to dispatch requests. Parameter $p$ is actually a site-selection parameter that selects only $p$ percent of the hottest sites to include in the forwarding table. To investigate how to choose the $p$ parameter, Chiang and his colleagues ran a trace-driven simulation to obtain the $c$ values under different $p$ values. We further calculated the corresponding memory requirements for different $p$ values. Table 3 shows the results for 10,000,000 total requests of 160,000 sites.
We constructed the table based on Chiang’s assumption that each entry needs 0.5 Kbytes on average. The full request-forwarding table size is thus 80 Mbytes for 160,000 entries. As Table 2 shows, reducing $p$ substantially reduces memory requirements. In the extreme case, when $p = 1$ percent, the partial table size is reduced 100 times — from 80 Mbytes to 800 Kbytes — and the table still serves more than half of all user requests. Better yet, when $p = 20$ percent, the table satisfies 90 percent of users’ requests and requires only 16 Mbytes of memory.

Based on our work, we’ve reached two major conclusions about multicache design. First, it is feasible to use site-based request forwarding in multicache design. Second, because of uneven Web server access rates, the site-based approach can greatly simplify the design complexity of dynamic caching hierarchies without a substantial drain on performance.

Conclusion
Our proposed site-based approach brings many benefits to different areas of Web caching. One problem, however, is that while maintaining only site information is sufficient in many cases, it is sometimes necessary to track individual documents. For example, although the site-based LRU purges a whole site when a cache replacement is required, the system might still need information on individual cached documents.

Currently, we are attempting to modify a real proxy server to use the site-based approach.

Acknowledgments
This research is supported by City University Research Grant 7001052. We also thank the anonymous referees for their valuable comments.

References

Kin Yeung Wong is a doctoral candidate in the Department of Electronic Engineering, City University of Hong Kong. His research interests include Web performance and Internet technologies. He received a BS in information technology from City University of Hong Kong.

Kai Hau Yeung is an associate professor in the Department of Electronic Engineering, City University of Hong Kong. His research interests include Internet caching systems, proxy server design, mobile communication systems, high performance I/O systems, and high-speed data distribution systems. He is also an active industry consultant in the areas of computer networking and communication systems. He is a member of the IEEE, ACM, and BCS. He is also a Cisco Certified Academy Instructor (CCAI).

Readers can contact Wong at kywong@ee.cityu.edu.hk or Yeung at eeayeung@cityu.edu.hk.

Coming in 2002 ...
• Peer-to-Peer Networking
• Usability and the World Wide Web
• Internet Telephony

Calls for papers are available at http://computer.org/call4ppr.htm.