CACHING AND PREFETCHING FOR WEB CONTENT DISTRIBUTION

Proxy caching effectively reduces the network resources that Web services consume, while minimizing user access latencies. Deploying Web caching proxies over the Internet is not easy, however, and presents several cache management challenges.

The World Wide Web is the Internet's most widely used tool for information access and dissemination, but today's users often experience long access latency due to network congestion—particularly during peak hours and big events, such as the Olympic Games. Caching frequently used data at proxies close to clients is an effective way to alleviate these problems. Specifically, caching can reduce load on both the network and servers (by localizing the traffic) and improve access latency (by satisfying user requests from local storage rather than remote servers).

Caching proxies have become vital components in most Web systems, and managing proxy cache is critical. Researchers have studied this management task extensively in other systems, such as memory hierarchies and distributed file-sharing systems. However, the Web and the Internet offer several unique challenges in this area, not the least of which are network size and the ever-evolving diversity of technologies and user behavior. Given this, we need novel solutions for deploying Web caching proxies on the Internet.

Here, we offer an overview of key management problems for Web proxy caching and prefetching and present state-of-the-art solutions to these problems. Our focus is on the distribution of conventional Web objects, such as HTML pages and images, but we also address issues arising from emerging applications and services, such as streaming media.

Overview: Proxy Caching Systems

A proxy is usually deployed at a network's edge, such as at an enterprise network's gateway or firewall. The proxy processes internal client requests either locally or by forwarding the requests to a remote server, intercepting the responses, and sending the replies back to the clients. Because this proxy is shared by internal clients who tend to have similar interests, it's natural to cache commonly requested objects on the proxy.

A client-side browser typically retrieves a Web object by initiating an HTTP `GET` command with the object's address. The browser first attempts to satisfy the request from its local cache; if it fails, it

1521-9615/04/$20.00 © 2004 IEEE
Copublished by the IEEE CS and the AIP

JIANLIANG XU
Hong Kong Baptist University
JIANGCHUAN LIU
Simon Fraser University, BC, Canada
BO LI
Hong Kong University of Science and Technology
XIAOHUA JIA
City University of Hong Kong

sends the unresolved request to its proxy. If the proxy finds the requested object in its cache, it returns the object to the client; otherwise, the request is forwarded to the object's origin server, which—as the authoritative source of the requested object—returns the object to the proxy. The proxy then relays the object to the client and, if needed, saves a copy in its cache. If a request is satisfied from the proxy cache, it is called a *cache hit*; otherwise, it's a *cache miss*. Figure 1 shows a stand-alone proxy's basic operations.

In addition to the basic `GET` command, HTTP also provides a conditional `GET` command. The proxy can use this modified `GET` with an `if-modified-since` date in the header to ensure that the remote server returns only copies modified since the specified date. Another important header is `expires`, which indicates when an object is no longer fresh. In HTTP 1.1 (the latest version), cache-control headers offer a richer set of proxy cache controls, including a list of directives to declare which objects can be cached, as well as expiration-mechanism modifications, and revalidation or reload controls.

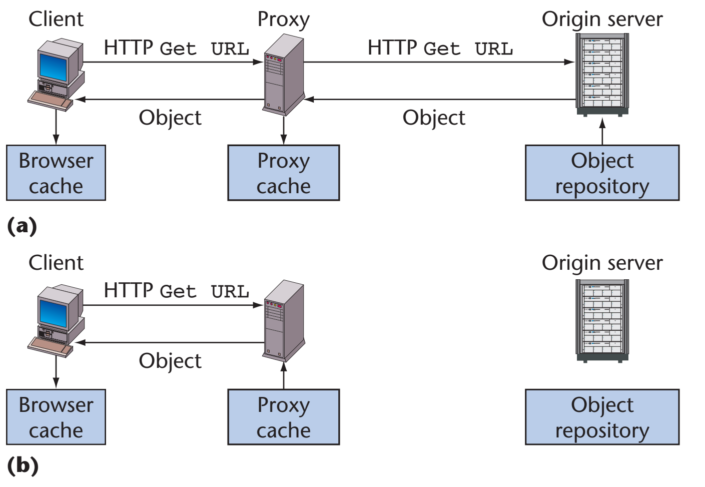

Figure 1. A stand-alone proxy. The browser initiates an HTTP GET command, and if it can't satisfy the request from local cache, it sends the request to the proxy. (a) When the proxy can't satisfy a request, a cache miss occurs. (b) A cache hit: the proxy finds the requested object and returns it to the client.

Caching Challenges and Solutions

Cache replacement and prefetching, consistency management, and cooperative management are key cache management issues. Although these issues date back to traditional memory hierarchies and file-sharing systems, several distinctive features of the Web and Internet necessitate different solutions.

First, there is the issue of size. The Internet is the world's largest interconnected network. Google alone receives more than 2,000 search queries a second. Given the Internet and Web's scale, any cache-management solution must be massively scalable—the proxy cache must be capable of handling numerous concurrent user requests.

Web application users also exhibit high heterogeneity in hardware and software configurations, connection bandwidth, and access behaviors. This diversity level continues to increase as new platforms and access technologies—such as mobile users with wireless access—proliferate. Hence, a simple one-size-fits-all solution for cache management might never be feasible.

In addition to this heterogeneity, proxy cache consumers (Web browsers) and suppliers (servers) are loosely coupled. Unlike in many distributed file-sharing systems, this loose coupling is key to Internet and Web success. However, it makes managing consistency and cooperation among proxy caches particularly difficult. Moreover, due to the lack of a centralized administration, security and privacy issues are deeply complicated.

Finally, the Web and the Internet change rapidly, both in traffic characteristics and network structures, which complicates analysis of the environment. The Web's dynamic nature easily makes existing products and even research findings obsolete in a few years. Thus, we need a flexible and extendable interface for any Web-oriented solution.

Cache Replacement and Prefetching

Faced with insufficient disk space, a proxy must decide which existing objects to purge when a new object arrives. Cache replacement policies address this issue. The classical cache replacement policy is least recently used (LRU), which purges the oldest among the cached objects. In the late '90s, researchers put significant effort into developing more intelligent cache replacement strategies. However, LRU offers limited room for improvement; in practice, the simple LRU policy dominates in cache products.

Cache prefetching is related to replacement, but unlike data caching, which waits on object requests, prefetching proactively preloads data from the server into the cache to facilitate near-future accesses. Studies have shown that, when combined with caching, prefetching can improve latency by up to 60 percent, while caching alone offers at best a 26-percent latency improvement.[1] However, a cache prefetching policy must be carefully designed: if it fails to predict a user's future accesses, it wastes network bandwidth and cache space. The prediction mechanism thus plays an important role

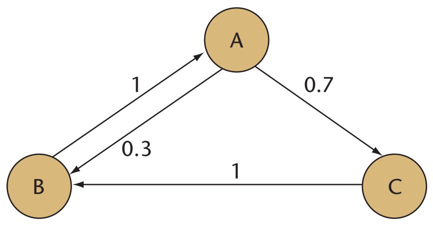

Figure 2. The Markov graph for a per-client access pattern. The graph enables prefetching predictions based on access history.

in cache prefetching policy design.

We can classify prefetching policies into three categories based on the type of information the prediction mechanism uses: mixed access pattern, per-client access pattern, and object structural information.

Mixed access pattern. This policy uses aggregate access patterns from different clients, but doesn't explore which client made the request. A typical example is the top-10 proposal,[2] which uses popularity-based predictions. Specifically, the scheme determines how many objects to prefetch from which servers using two parameters:

- M, the number of times the client has contacted a server before it can prefetch, and
- N, the maximum number of objects the client can prefetch from a server.

If the number of objects fetched in the previous measurement period L reaches the threshold M, the client will prefetch the K most popular objects from the server, where $K = \min\{N, L\}$.

Per-client access pattern. Here, the policy first analyzes access patterns on a per-client basis, then uses the aggregated access patterns for prediction. An example is the popular Markov modeling analysis tool,[3] in which the policy establishes a Markov graph based on access histories and uses the graph to make prefetching predictions. In the Markov graph, a set of Web objects (usually one or two objects) is represented as a node; if the same client accesses two nodes (A and B) in order within a certain period of time, the policy draws a direct link from A to B and assigns a weight with the transition probability from A to B.

Figure 2 shows an example in which the probability of accessing B after A is 0.3 and the probability of accessing C after A is 0.7. To make a prefetching prediction, a search algorithm traverses the graph starting from the current object set and computes the access likelihood for its successors; the prefetching algorithm can then decide how many successors to preload, depending on factors such as access likelihood and the bandwidth available for prefetching.

Object structural information. Unlike the previous categories, which are access-history based, object structural information schemes exploit the local information contained in objects themselves. Hyperlinks, for example, are good indictors of future accesses because users tend to access objects by clicking on links rather than typing new URLs.[4] The algorithm can also combine object information with access-pattern-based policies to further improve predication efficiency and accuracy.

Consistency Management

If the origin server updates an object after a proxy caches it, the cached copy becomes stale. A cache consistency algorithm's goal is to ensure consistency between the cached copy and the original object. Existing cache consistency algorithms can be classified as either weak or strong. If t is the delay between the proxy and server, a strong consistency algorithm returns the object outdated by t at most; algorithms that can't provide such a guarantee offer weak consistency.

Weak consistency. Weak cache consistency is generally supported by validation, in which proxies verify the validity of their cached objects with the origin server. There are two basic validation approaches: time-to-live (TTL)-based validation and proactive polling. With the TTL-based approach, the proxy assigns a TTL value to the object upon caching. When a request arrives, the proxy serves it with the cached copy if its lifetime has not expired; otherwise, the proxy sends the server a conditional request to download a newer version of the object, if it exists.[5]

Despite its simplicity and effectiveness, the TTL-based approach suffers from a major drawback: if an object expires, but the origin server has not yet updated it, the proxy must still verify with the server, which will return only a "not modified" message. This delays access, reducing proxy caching's effectiveness. To address this, the proxy can proactively poll the server to check cached copies' validity, either at fixed or adaptive intervals. Another option is to batch the validation requests and responses, or piggyback them over normal HTTP traffic to reduce polling bandwidth overhead.

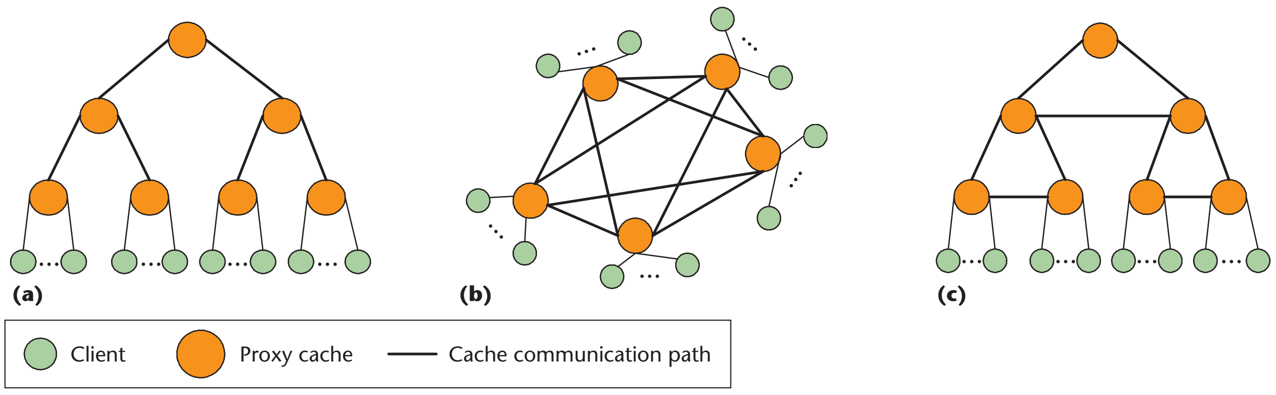

Figure 3. Different cooperative cache organizations. (a) A cache hierarchy, (b) a distributed cache architecture, and (c) a combination of the two.

Strong consistency. Strong cache consistency can be enforced by either server-driven invalidation or client-driven validation.[6,7] In the server-driven invalidation approach, the server must invalidate a proxy's copies before it can update the objects. To notify the proxies of object updates, the server must maintain for each object a state record of the list of proxies that cache the object. For a popular Web server, the extra space required to maintain all objects' states can be significant. In contrast, client-driven validation doesn't require the server to maintain any state. Instead, the proxy validates the cached copies' freshness with the server for every cache-hit access. However, as with the TTL-based approach, this introduces unnecessary access delay for valid cache-hit objects. In addition, if the object is accessed more often than it's updated, validity checking with this approach can generate numerous unnecessary messages.

To balance the space required to maintain states with the message volume that validations require, researchers developed a hybrid approach called *leases*.[6,7] In this approach, the server and proxy agree that the server will notify the proxy if the leased object is updated during the lease period. The server can grant a lease to an object for every proxy request, or for cache-hit requests only (because invalidations are useful only for frequently accessed objects). If the lease has not expired, the proxy can immediately serve client requests from the cache. Otherwise, the proxy must validate the cached object and renew the lease on the first access after expiration.

Cache Cooperation

So far, we've considered only stand-alone proxy design. One disadvantage of this design is that the proxy represents a single point of failure and performance bottleneck. In other words, it lacks robustness and scalability. *Cooperative caching*, in which caching proxies collaborate with one another in serving requests, overcomes these limitations.[8–11]

Because the Internet topology is generally organized hierarchically, setting up a cache hierarchy is a popular technique for making caches cooperate. Such a hierarchy defines a parent-child relationship between proxy caches (see Figure 3a). Each cache in the hierarchy is shared by a group of clients or children caches. The cooperative caches process a Web object request as follows. If the client can't locate the object in its local cache, it sends the request to its leaf proxy. If this leaf cache contains the object, it returns the object to the client. Otherwise, the cache forwards the request to its parent. This process recursively proceeds up the hierarchy until the object is located on some proxy or on the origin server. The caching hierarchy often requires manual configuration, which makes it vulnerable to topology changes. This typically limits hierarchy depth—most operational hierarchies have only three levels: institutional, regional, and national.

An alternative is to set up a distributed cache architecture, where the participating proxy caches are peers (see Figure 3b). A simple yet popular distributed cooperation model is based on broadcast queries: If a proxy can't satisfy a client's request from its own cache, it will broadcast the request to all proxies it cooperates with, trying to resolve the request without contacting the origin server. However, it's well known that large-scale broadcast overhead is prohibitively high, even if multicast is used. A more intelligent and efficient way is to forward an object request only to proxies that might contain the object. Many methods have been devised to achieve this objective, such as to distribute the cached objects' digests to peer caches, or to hash partition the object namespace among proxies.

Researchers have also proposed hybrid schemes that combine the advantages of hierarchical and distributed caching. Figure 3c shows an example, in which proxies are organized into local groups. Using multicast, clients first query requested objects within a local group. If this produces a cache miss, the request is recursively forwarded to the proxy group in the next higher layer. The Internet Cache Protocol, a generic protocol for intercache communications,[11] contains additional hybrid configurations.

In a cooperative caching environment, managing cache replacement and consistency is clearly more complex than with stand-alone proxies. However, studies show significant benefits when proxies cooperate on cache replacement decisions.[12] To manage consistency, cooperative caches (such as Harvest and Squid) still widely use the simple TTL-based schemes. Recently, researchers have also developed multicast-based invalidation for cache hierarchies and cooperative leases for distributed caches.[13]

Web proxies play a vital role in efficiently distributing Web contents over the Internet. Many of the proposed solutions that we've presented here have matured and are deployed in commercial products and systems. With emerging applications and service models, however, cache management for Web proxies remains a fertile research area. There are several possible directions for such research, including caching dynamic content and streaming objects, and security and integrity issues.

According to a recent study,[5] dynamic content—such as dynamically generated data and personalized data—now contributes up to 40 percent of the total Web traffic. Dynamic data is typically marked as noncachable, and thus the origin server must serve each user request. To improve performance for data-intensive Web applications, developers have deployed reverse caches near the origin server to support dynamic content caching. The main challenge is to ensure freshness of cached content; otherwise, the user might receive stale data. Another important issue is the analysis of query semantics, which decomposes a given query into a *probing query* to be satisfied by the cache and a *remainder query* to be sent to the server for evaluation.

Analysts predict that streaming media, such as music or video clips, will soon represent a significant portion of Web traffic. Caching streaming media is particularly attractive given its content's static nature. However, due to streaming objects' distinctive features—such as huge size, intensive bandwidth use, and high interactivity—conventional proxy caching techniques don't perform efficiently.[14] One solution is partial caching. Many recent studies have demonstrated that even if a small portion of a streaming object is stored on the proxy, the client playback quality can be significantly improved. How to optimally choose the portions to cache and how to synchronize proxy and origin server delivery remain difficult tasks, as does cache management on cooperative proxies. The problem is further complicated in that streaming objects often have variable bit rate and stringent transmission delay or delay jitter demands.

Finally, using proxies creates many security problems. It's difficult, for example, to protect stand-alone caches from various attacks, including invasion and denial-of-service. For cooperative caches, establishing a trust model among participants is challenging. In addition, Web applications typically use the Secure Sockets Layer protocol to provide the end-to-end security for data transmissions between the client and the server, but the existence of an intermediate proxy largely violates SSL's functionality. To alter data, an attacker can now target both a proxy and the origin server, so it's crucial to ensure the integrity of a proxy's cached content.

Acknowledgments

Jianliang Xu's work was supported by a grant from Hong Kong Baptist University under contract FRG/03-04/II-19. Jiangchuan Liu's work was supported in part by a grant from Hong Kong Research Grants Council (RGC) under contract CUHK C001/2050312. Bo Li's work was supported in part by RGC grants under contracts HKUST6196/02E and HKUST6402/03E, a National Natural Science Foundation of China (NSFC)/RGC joint grant under contract N_HKUST605/02, and a grant from Microsoft Research under contract MCCL02/03.EG01. Xiaohua Jia's work was supported by RGC under grant CityU 1209/03E.

References

1. T.M. Kroeger, D.D.E. Long, and J.C. Mogul, "Exploring the Bounds of Web Latency Reduction from Caching and Prefetching," *Proc. Usenix Symp. Internet Technology and Systems*, Usenix Assoc., 1997, pp. 13–22.
2. E.P. Markatos and C.E. Chronaki, "A Top 10 Approach for Prefetching the Web," *Proc. Ann. Internet Society Conf.* (INET), Internet Soc., 1998; www.isoc.org/inet98/proceedings/1i/1i_2.htm.
3. A. Bestavros, "Using Speculation to Reduce Server Load and Service Time on the WWW," *Proc. 4th ACM Int'l Conf. Information*

and Knowledge Management (CIKM'95), ACM Press, 1995, pp. 403–410.

4. D. Duchamp, "Prefetching Hyperlinks," *Proc. Usenix Symp. Internet Technology and Systems*, Usenix Assoc., 1999, pp. 127–138.
5. A. Feldmann et al., "Performance of Web Proxy Caching in Heterogeneous Bandwidth Environments," *Proc. Joint Conf. IEEE Computer and Comm. Societies* (Infocom), IEEE CS Press, 1999, pp. 107–116.
6. V. Duvvuri, P. Shenoy, and R. Tewari, "Adaptive Leases: A Strong Consistency Mechanism for the World Wide Web," *IEEE Trans. Knowledge and Data Eng.*, vol. 15, no. 4, 2003, pp. 1266–1276.
7. P. Cao and C. Liu, "Maintaining Strong Cache Consistency in the World-Wide Web," *Proc. Int'l Conf. Distributed Computing Systems*, IEEE CS Press, 1997, pp. 12–21.
8. A. Chankhunthod et al., "A Hierarchical Internet Object Cache," *Proc. Usenix Technical Conf.*, Usenix Assoc., 1996, pp. 153–163.
9. L. Fan et al., "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol," *IEEE/ACM Trans. Networking*, vol. 8, no. 3, 2000, pp. 281–293.
10. B. Li et al., "On the Optimal Placement of Web Proxies in the Internet," *Proc. Joint Conf. IEEE Computer and Comm. Societies* (Infocom), IEEE CS Press, 1999, pp. 1282–1290.
11. D. Wessels and K.C. Claffy, "ICP and the Squid Web Cache," *IEEE J. Selected Areas in Comm.*, vol. 16, no. 3, 1998, pp. 345–357.
12. X. Tang and S.T. Chanson, "Coordinated En-Route Web Caching," *IEEE Trans. Computers,* vol. 51, no. 6, 2002, pp. 595–607.
13. A. Ninan et al., "Cooperative Leases: Scalable Consistency Maintenance in Content Distribution Networks," *Proc. Int'l World Wide Web Conf.* (WWW10), ACM Press, 2002, pp. 1–12.
14. J. Liu and J. Xu, "Proxy Caching for Media Streaming over the Internet," *IEEE Comm.,* special issue on proxy support for streaming Internet, 2004, to appear.

Jianliang Xu is an assistant professor in the Department of Computer Science at Hong Kong Baptist University. His research interests include mobile and pervasive computing, Web content delivery, and wireless networks. He has served as a program committee member and an executive committee member for several international conferences, including IEEE Infocom. He received the BEng in computer science and engineering from Zhejiang University, Hangzhou, China, and a PhD in computer science from Hong Kong University of Science and Technology. He is a member of the IEEE and an executive committee member of the ACM Hong Kong chapter. Contact him at xujl@comp.hkbu.edu.hk.

Jiangchuan Liu is an assistant professor in the School of Computing Science at Simon Fraser University, BC, Canada. From 2003 to 2004, he was an assistant professor in the Department of Computer Science and Engineering at the Chinese University of Hong Kong. His research interests include multicast protocols, streaming media, wireless ad hoc networks, and service overlay networks. He is the recipient of a Microsoft Research Fellowship and a Hong Kong Young Scientist Award, and serves on the technical program committees of various networking conferences, including IEEE Infocom 2004 and 2005. He received a BEng (cum laude) in computer science from Tsinghua University, Beijing, and a PhD in computer science from the Hong Kong University of Science and Technology. Contact him at csljc@ieee.org.

Bo Li is an associate professor of computer science and codirector of the ATM/IP Research Center at the Hong Kong University of Science and Technology. He is also an adjunct researcher at Microsoft Research Asia in Beijing. His current research interests include wireless mobile networking supporting multimedia, video multicast, and all-optical networks using wavelength division multiplexing. He previously worked at IBM Networking System Division in North Carolina. He received a BS (summa cum laude) and MS in the computer science from Tsinghua University, Beijing, and a PhD in electrical and computer engineering from University of Massachusetts, Amherst. Contact him at bli@cs.ust.hk.

Xiaohua Jia is a professor in the Department of Computer Science at City University of Hong Kong and an adjunct professor in the School of Computing, Wuhan University, China. His research interests include distributed systems, computer networks, wavelength division multiplexing optical networks, Internet technologies, and mobile computing. He received a BSc and an MEng in computer science from the University of Science and Technology of China and a DSc in information science from the University of Tokyo. Contact him at csjia@cityu.edu.hk.