Caching and Prefetching for Web Content Distribution[1]

Jianliang Xu

Hong Kong Baptist University
xujl@comp.hkbu.edu.hk

Jiangchuan Liu

The Chinese University of Hong Kong
ljc@cse.cuhk.edu.hk

Bo Li

Hong Kong University of Science and Technology
bli@cs.ust.hk

Xiaohua Jia

City University of Hong Kong
jia@cs.cityu.edu.hk

Abstract

Proxy caching is an effective technique to reduce the network resources consumed by web services as well as the access latencies perceived by web users. This article discusses the issues and challenges of deploying web caching proxies over the Internet. We focus on the cache management for conventional web objects such as HTML pages and images and present the state-of-the-art solutions to various cache management problems including prefetching, consistency maintenance, and cooperative caching. We also highlight several possible research directions with the emerging applications and services.

Keywords: World Wide Web, Caching, Prefetching, Proxy

This is an extended version of the article to appear in IEEE Computing in Science and Engineering Magazine: Special Issue on Web Engineering

[1] J. Xu's work was supported by a grant from Hong Kong Baptist University under contract FRG/02-03/II-34. J. Liu's work was supported in part by a grant from Hong Kong Research Grants Council under contract CUHK C001/2050312. B. Li's work was supported in part by grants from RGC under contracts AoE/E-01/99, HKUST6196/02E and HKUST6402/03E, a NSFC/RGC joint grant under the contract N_HKUST605/02, and a grant from Microsoft Research under the contract MCCL02/03.EG01. X. Jia's work was supported by Hong Kong Research Grant Council under grant No. CityU 1209/03E.

I. INTRODUCTION

The World Wide Web (WWW) has emerged as the most widely used tool for information access and dissemination throughout the Internet. Nevertheless, web users today often experience long access latency due to network congestions especially in peak hours and/or during big events such as the Olympic Games. An effective technique to alleviate these problems is to cache frequently used data at proxies close to clients. Specifically, they can reduce load on both the network and the servers (by localizing the traffic) and improve access latency (by satisfying the requests from local storage instead of remote servers).

Nowadays, caching proxy has become one of the vital components in most existing web systems. It is no doubt that the management of proxy cache is critical to such a system. Though the cache management has been extensively studied in other systems such as memory hierarchies and distributed file sharing systems, there are several unique challenges arising from the Web and its vehicle, the Internet [1, 2, 3].

Large-scale systems: The Internet is by far the largest inter-connected network, and the Web is now enjoyed by virtually all the Internet surfers. For example, Google (*http://www.google.com*) receives more than 2,000 search queries a second, more than half of which come from outside the United States.[2] The large scale of the Internet and Web implies that any solution for cache management must be massively scalable, *i.e.*, the proxy cache should be capable of handling a large number of concurrent requests from users that may be distributed worldwide.

Heterogeneous users: In web applications, users exhibit high heterogeneity in hardware and software configurations, connection bandwidth, as well as access behaviors. The level of heterogeneity is still increasing with the proliferation of new platforms and access technologies, *e.g.*, mobile users with wireless access. Hence, a simple one-fits-all solution for cache management might never be feasible.

Loosely-coupled components: Besides heterogeneity, the consumers (web browsers) and the suppliers (web servers) of a proxy cache are loosely coupled, which reflects one of the fundamental design principles of the Internet. This loosely coupling nature, unlike that in many distributed file sharing systems, leads to the success of the Internet and Web, but also makes the consistency management for proxy caches and cooperation among the caches particularly difficult. Moreover, due to the lack of centralized administration, achieving security and privacy is far more complicated.

[2] From Google Press Center: *http://www.google.com/press/funfacts.html*

Dynamic characteristics: Finally, it is worth noting that the Web and the Internet have been changing rapidly both in traffic characteristics and in network structures, which complicates the analysis of this computing environment. The dynamics of the Web also easily makes existing products and even research findings obsolete in a few years. Hence, a flexible yet extendible interface is necessary for a web-oriented product or solution.

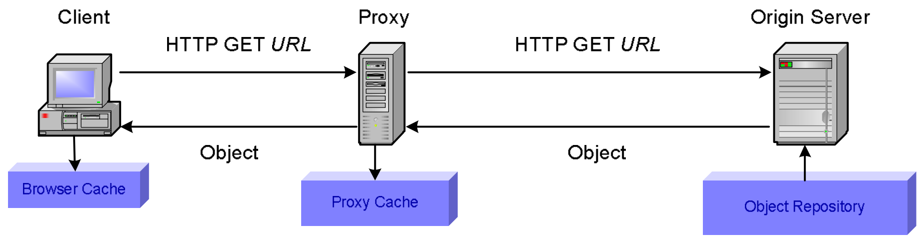
Given these unique characteristics, novel solutions are needed to deploy web caching proxies on the Internet. This article puts together important management problems for web proxy caching and presents the state-of-the-art solutions to these problems. We mainly focus on the cache management for conventional web objects such as HTML pages and images since they dominate the web traffic today; yet new issues arising from emerging applications and services such as streaming media will be briefly mentioned. We start by an overview of the proxy caching systems in Section II. We present the critical cache management issues in Section III. Section IV summarizes the article and outlines some future research directions.

II. Overview of Proxy Caching Systems

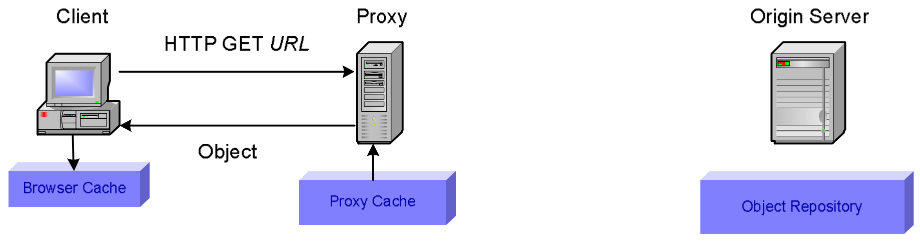
A. Basic Proxy Caching Architecture

A proxy is usually deployed at the edge of a network, *e.g.*, at the gateway or firewall for an enterprise or campus network, and processes requests from its internal clients either locally or by forwarding the requests to a remote server, intercepting the responses, and sending the replies back to the clients. Since this proxy is shared by all internal clients with similar interests, it is natural to cache some objects of common interest on the proxy, beyond the local caching mechanisms implemented by web browsers.

More explicitly, to retrieve a web object, a client-side browser can initiate an HTTP *GET* command with a Uniform Resource Locator (URL) for the object. The browser first attempts to satisfy the request from its local cache, and if it fails, sends unresolved requests 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, will return the object to the proxy. The proxy then relays the object to the client, and, if needed, saves a copy of the object in its cache. If a request is satisfied from the proxy cache, it is called a *cache hit*, and otherwise a *cache miss*. The operations for a standalone proxy are depicted in Fig. 1.



(a)



(b)

Figure 1. Operations for HTTP GET command with proxy caching. (a) Cache miss; (b) Cache hit.

Besides the basic GET command, HTTP also provides a *conditional GET* command, which combined with header *if-modified-since date* can be used by the proxy to request the remote server to return a copy only if it has been modified since the specified date (with the granularity of a second). Another important header is *expires*, which indicates the time after which a respective object is not fresh. In HTTP 1.1 [5], the latest version of HTTP, a richer set of controls over proxy caches is provided by means of *cache-control* headers. A cache-control header includes a list of directives to declare which objects can be cached, as well as modifications of the expiration mechanism and revalidation or reload controls. Specifically, header *no-cache* disables caching of a web object at either proxy or browser and, hence, every request for the object has to go to the origin server. The *must-revalidate* header informs a cache that it must validate the cached object before using it, possibly through a conditional request to the origin server. And the *proxy-revalidate* header is similar to *must-revalidate* yet only applies to proxy caches. In addition to these, some caching proxies also permit the administrator to specify a sets of URLs or object types for which to apply special rules, *e.g.*, never to cache URLs containing a suffix ".cgi", since this directs to a CGI (Common Gateway Interface) script which can create dynamic objects. All the above headers and policies are very useful in cache replacement and prefetching, consistency management, and proxy cooperation, which are discussed in the next section. Finally, HTTP 1.1 also provides a *via* header, which requires that each proxy processing the request must append an entry including the proxy name and the protocol used to receive the request. Hence, the information of all the proxies lying between the client and the server can be obtained.

B. Alternative Architectures

The above proxy caching architecture is by far the most widely employed. It however requires that all the client-side web browsers be manually configured to direct their requests to the dedicated proxy cache. Subsequently, if the proxy is no longer available, all clients have to reconfigure their browsers, which is clearly cumbersome. One solution to this problem is the browser auto-configuration using script files. In addition, the Internet Engineering Task Force (IETF) is actively developing the Web Proxy Auto-Discovery Protocol (WPAD) [6]. WPAD relies on resource discovery mechanisms, including domain name service (DNS) records and Dynamic Host Configuration Protocol (DHCP), to automatically locate a proxy configuration file.

Yet a more sophisticated solution is transparent caching [7, 8], which filters all HTTP requests from all outbound Internet traffic, and redirects them to appropriate proxies. It not only solves the proxy configuration problem, but also enables additional administrative controls, *e.g.*, adaptively distributing requests among multiple caches. Transparent caching can be implemented either at the switch level or at the router level, both being transparent to the application-level HTTP. A router-level implementation is more flexible, as various policy-based routing policies can be employed; a switch-level implementation, on the other hand, is less expensive and incurs lower processing overhead. Even so, HTTP filtering is still computation-intensive, resulting in higher latency for web browsing. More importantly, it violates the end-to-end principle as the clients' requests to the origin server are now unconsciously intercepted by the transparent proxies. This might be a problem when a client requires states to be maintained throughout successive requests or during a logical request involving multiple objects.

Other interesting proxy caching architectures include reverse proxy caching and push caching [9, 10]; in both mechanisms the web server is activity involved. In the reverse proxy caching scheme, caches are deployed near the origin server instead of near clients. This is an attractive solution for a server that receives a high number of concurrent requests and expects to ensure a high degree of quality-of-service. On the other hand, the push caching scheme keeps objects of interest close to its clients by dynamically mirroring objects to the proxy near the clients. Since the origin server has the knowledge of the global request pattern, it can easily identify where requests originate from and hence decide which proxy to push the objects. To date, the deployment of the above alternative architectures remains quite limited, however. Hence, in the rest of this paper, we mainly focus on the basic architecture for proxy caching, though many of the solutions for cache replacement, prefetching, and consistency management can be applied to other architectures as well.

III. CACHE MANAGEMENT ISSUES

We now proceed to discuss several important cache management issues including cache replacement and prefetching, consistency management, and cooperative management. The provenances of these issues dated back to the traditional memory hierarchies and file sharing systems. However, as mentioned in the Introduction, the distinctive features of the Web and the Internet often call for different solutions.

A. Cache Replacement and Prefetching

A natural question for a proxy is to decide, when a new object arrives, which object(s) is to be purged if the available disk space is insufficient to store the new object. This problem is addressed by a *cache replacement* policy. The classical cache replacement policy is the Least Recently Used (LRU) policy, which purges the oldest among the cached objects. There were significant research efforts towards more intelligent cache replacement strategies in the late 90's. However, recent studies have argued that, for several reasons, replacement policies are not the dominating factor in proxy performance [1]. First, intelligent replacement policies typically come at the cost of accurately estimating access patterns and network characteristics, which are usually difficult to obtain in the dynamically changing web environments; second, with the rapid growth of disk capacity, cache replacement becomes less important; finally and perhaps most importantly, a large fraction of web objects are not cachable and many others are dynamically updated, thus even an ideal cache with infinite space would achieve a hit ratio of 40-50% only [11]. The room for possible improvement over the LRU is quite limited. Therefore, in practice, the simple LRU policy dominates in the cache products. For new applications such as video object caching, cache replacement however remains as an important problem, which we will address in Section IV.

A related issue is *cache prefetching*. While data caching, as a side-product of object requests, is passive in nature, prefetching on the other hand proactively preloads the data from the server into the cache to facilitate near-future accesses. Existing studies have showed that prefetching combined with caching can potentially improve the latency by up to 60%, whereas caching alone can improve the latency at best by 26% [11]. However, a cache prefetching policy must be carefully designed. If it fails to predict the user's future accesses, it would waste network bandwidth and cache space in addition to imposing additional load to both the server and the network. Thus, the prediction mechanism plays an important role in the design of a cache prefetching policy. Based on the type of information used in the prediction mechanism, we can classify the prefetching policies into three categories: *mixed access pattern based*, *per-client access pattern based*, and *object structural information based*. Note that access patterns can be collected at the server, the proxy, or the client and exchanged among them for the purpose of prediction.

Mixed access pattern based: Prediction makes use of aggregate access pattern from different clients but does not explore which client makes the request. A typical example is the *Top 10* proposal [12], which employs popularity-based predictions. Specifically, the scheme determines how many objects to prefetch from which servers by using two parameters: *M*, the number of times the client must have contacted a server before it can prefetch, and *N*, the maximum number of objects it would prefetch from a server. If to some server, the number of objects fetched in the previous measurement period, denoted by *L*, reaches the threshold *M*, the client will prefetch the *K* most popular objects from that server, where $K = \min\{M, L\}$.

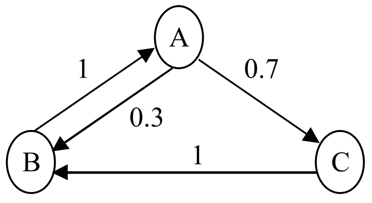


Figure 2. A Markov Graph.

Per-client access pattern based: This kind of policies first analyze access patterns on a per-client basis and use the aggregated access patterns for prediction. A popular analysis tool is called *Markov modeling* [13]. The basic idea is to establish a *Markov graph* based on access histories and make prefetching predictions based on the graph. 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 sets of objects *A* and *B* in order within a certain period of time, a direct link is drawn from *A* to *B* and its weight is assigned with the transition probability from *A* to *B*. Fig. 2 gives an example, where 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 set of objects and computes the access likelihood for its successors; the prefetching algorithm can then decide how many of the successors to preload, depending on various factors such as the access likelihood and the amount of bandwidth available for prefetching.

Object structural information based: While the previous two categories of policies rely on access histories, object structural information based prefetching schemes exploit the local information contained in objects themselves. Hyperlinks are good indictors of future accesses because the users tend to access objects by clicking on the links instead of typing new URLs [14]. Such information can also be combined with the access pattern based policies to further improve the predication efficiency and accuracy.

B. Consistency Management

After the proxy caches a copy of an object, the origin server may update this object. This makes the cached copy *stale*. A *cache consistency* algorithm is to ensure that the copy cached on the proxy is consistent with the original object. Different applications have diverse consistency requirements. For some applications like news and reports, users can tolerate a certain degree of staleness on the received data. However, for time-critical applications like traffic information and stock prices, users often expect to receive the most up-to-date information. To this end, existing cache consistency algorithms can be classified into two categories: *weak consistency* and *strong consistency*. Let t be the delay between the proxy and the server, a strong consistency algorithm returns the object outdated by t at most. If a cache algorithm fails to provide such guarantee, it falls into the category of 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: *TTL-based validation* and *proactive polling*. With the TTL-based approach, when the proxy caches an object, it assigns a value of time to live (TTL) to the object. The value of TTL can be explicitly supplied by the origin server or implicitly assigned by the proxy based on some heuristic. A popular heuristic to compute the TTL value is based on a predefined threshold and the weighted *object age*, which is the time interval from the object's last update to the current time [15]:

$$\text{TTL} = \min\{(k \times (current\_time - last\_update\_time), threshold)\},$$

where k is a constant typically set to 0.1 or 0.2. The TTL-based approach carries out validation in an on-demand manner. When a request comes to the proxy, if the lifetime of the cached object has not expired, the cached copy is used to serve the request; otherwise, the proxy sends to the server a conditional request to download a newer version of the object, if exists.

Despite its simplicity and effectiveness, the TTL-based approach suffers from a major drawback: if an object expires but has yet not updated on the origin server, the proxy still needs to verify with the server, though only a simple “Not Modified” message is returned. This introduces extra access delay, reducing the effectiveness of proxy caching. An improvement is for the proxy to proactively poll the server to check the validity of the cached copies, either at a fixed interval or an adaptive interval. With such asynchronous validations, the proxy can answer all cache-hit requests immediately from the proxy; thereby effectively reducing the latency overhead for all objects including those expired but still valid. Unfortunately, the proactive polling approach may waste the network bandwidth if the cached copy is rarely accessed after validation. To alleviate this problem, one may batch the validation requests and responses, or piggyback them over normal HTTP traffic [16].

Strong Consistency: Strong cache consistency can be enforced either by *server-driven invalidation* or *client-driven validation* [17]. The server-driven invalidation approach requires the server to invalidate the copies on the proxies before the objects can be updated. Since the server sends out invalidation messages as needed, the message exchange between the server and the proxies is minimal. However, to notify the proxies of object updates, the server needs to maintain for each object a state record of the list of the proxies that cache the object. The extra space required for maintaining the states of all objects can be significant for a popular web server. Moreover, this approach does not work if the proxy is not reachable from the server when invalidation messages are delivered. The client-driven validation, on the other hand, does not require any state to be maintained on the server. Instead, the proxy validates the freshness of the cached copies with the server for every cache-hit access. However, similar to the TTL-based approach, it introduces unnecessary access delay for valid cache-hit objects. In addition, if the object is accessed more frequently than updated, this approach can generate tremendous amount of unnecessary messages for validity checking.

To strike a balance between the space usage for maintaining the states and the message volume required for validations, a hybrid approach is developed, called *leases* [18, 19]. The server and the proxy agree on a *lease* such that the server will notify the proxy if the leased object is updated during the lease. The server can grant a lease to an object for every proxy request, or cache-hit requests only based on the observation that invalidations are useful only for frequently accessed objects. Upon clients' requests, the proxy can immediately serve them from the cache if the lease has not expired. Otherwise, the proxy must validate the cached object and renew the lease on the first access after lease expiration. Clearly, the duration of a lease is crucial to the system performance. The shorter the lease duration, the less the states maintained on the server, but potentially the larger the network messages for validity checking. Interested reader is referred to [17] for the optimization analysis of lease duration.

C. Cooperation among Proxy Caches

So far we consider the design of a standalone proxy only. Notice that 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, where caching proxies collaborate with one another in serving requests, has emerged as an approach to overcome this limitation [20]. Conceptually, cooperation among proxies is based on the premise that it would be faster and cheaper to fetch an object from another proxy nearby rather than the origin server. In terms of scalability and availability, one can expect that multiple cooperated proxies can accommodate a large volume of concurrent client requests, survive failures of some caches, and perhaps can evenly distribute their workloads.

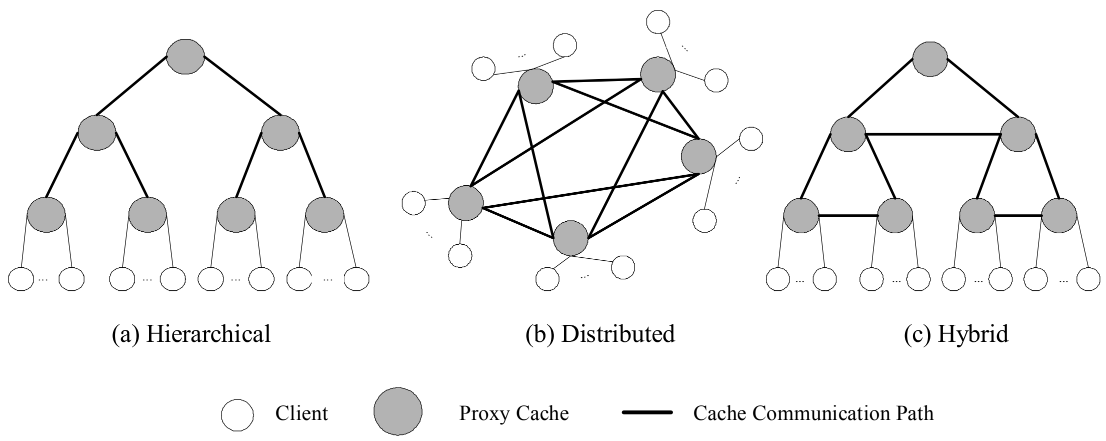


Figure 3. Examples of different cooperative cache organizations.

Since the Internet topology is in general hierarchically organized, a popular technique to make caches cooperate is by setting up a caching hierarchy, *i.e.*, defining a parent-child relationship between proxy caches; see Figure 3(a) for an example. Each cache in the hierarchy is shared by a group of clients or children caches. A request for a web object is processed as follows: if the client cannot locate the object in its local cache, it sends the request to the leaf proxy to which it directly connects. 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 the origin server, which then returns the object down the hierarchy and each cache along the path may cache the object.

Such hierarchical cache cooperation, originated from the famous Harvest project [21], is already adopted in plenty of Internet ISPs or enterprise networks. However, a simple hierarchy as illustrated above has several problems. First, top-level caches have to handle all the cache-miss requests from downstream caches and thus easily become the performance bottlenecks. Second, the recursive request-response process introduces additional delay as well as data redundancy, *i.e.*, the copies of some objects are stored at every hierarchy level. Finally and most importantly, organization of the caches in a hierarchy is often static and manually configured that not only requires significant coordination among participating proxies but also is vulnerable to *inappropriate* configurations and topology changes. Due to the above reasons, the depth of a hierarchy is often limited — most operational hierarchies have only three levels, namely, institutional, regional, and national levels.

A number of recent studies have proposed to set up a fully distributed cache architecture, where the participating proxy caches are peers, as illustrated in Figure 3(b) [22]. A simple yet popular distributed cooperation model is based on *broadcast queries*: If a proxy cannot satisfy a client's request for an object from its own cache, it will broadcast the request to all other proxies it cooperates with, trying to resolve the request without contacting the origin server. It is however well known that the overhead of large-scale broadcast is prohibitively high, even if multicast is used. A more intelligent and efficient way is to forward an object request to only the proxies that may contain the object. Many methods have been devised to achieve this objective, *e.g.*, maintaining a centralized directory for the objects [23], distributing the digests of the cached objects to peer caches [24, 25], and hash-partitioning of the object namespace among proxies [26].

Distributed caching is particularly suitable for cooperation within regional networks, where the nodes are well connected with abundant bandwidth. With the proxy caches at the same level, it can achieve shorter transmission delay as well as higher disk space utility. It also enables better load sharing and higher degree of fault tolerance. Nonetheless, a large-scale deployment of distributed caching may encounter several problems, such as the high bandwidth demands and the complex administrative policies to ensure that all the proxies fully cooperate. Thus, a hybrid scheme is often used to combine the advantages of both hierarchical and distributed caching, and to mitigate their respective disadvantages. For example, Harvest [21] and its descendants, such as Squid [27], also allow a proxy to query its neighbors at the same level (called *siblings*) using multicast before sending a cache-miss request to a parent, thereby not only reducing the workload of upper level proxies but also improving the access latency; see Figure 3(c) for an example. In the Adaptive Web Caching [28], all the origin servers and proxies are self-organized into multiple overlapping local groups, which altogether form a tight mesh. Requested objects are first queried within a local group using multicast. In case of a cache-miss within the group, the request is then forwarded toward the origin server along some implicit hierarchy of groups defined on the global mesh. More configurations for hybridization can be found in the Internet Cache Protocol (ICP) [27], a generic protocol for intercache communications. ICP was originally designed by the Harvest group and its variations were later implemented in many other cooperative caching systems.

In a cooperative caching environment, cache replacement and consistency management is clearly more complex as compared to the standalone proxy case. In fact, many of the proxies cooperate only in serving a request, but not in cache replacement. However, existing studies revealed that if the proxies are cooperative in making cache replacement decisions, more benefits can be gained when the client accesses are intensive and the requests are highly clustered to a relatively small set of hot objects, as in the Content Distribution Networks (CDNs) [28]. For consistency management, the simple TTL-based schemes are still widely used for cooperative caches such as Harvest and Squid to ensure weak consistency [29, 30]. Recently, multicast-based invalidation for cache hierarchies [31] and cooperative leases for distributed caches [32] have also been developed.

IV. SUMMARY AND FUTURE DIRECTIONS

Nowadays, web proxies play a vital role in efficient distribution of web contents over the Internet. This article presents an overview of the proxy cache design and management and examines the proposed solutions, many of which are mature enough and have been deployed in commercial products and systems. This article by no means covers all aspects of the proxy cache management. With emerging new applications (*e.g.*, streaming media) and service models (*e.g.*, QoS-oriented services), cache management for web proxies remains a fertile research area, and many newly arising issues have not been well addressed. To conclude this article, we list several possible research directions:

Caching dynamic content: A large portion of web traffic today is dynamic content, *e.g.*, dynamically generated data and personalized data, which contributes up to 30-40% of the total traffic. These types of data are normally marked as "uncachable." To further improve web performance, reverse caching has been suggested to make more dynamic content cachable and manageable [9]. However, challenge remains for how to efficiently maintain consistency between the cached content and the dynamically changing source data. Another important issue is the analysis of query semantics to evaluate a complex query over the cached content.

Caching streaming objects: It is predicted that streaming media like music or video clips will represent a significant portion of web traffic over the Internet. Due to the distinct features of streaming objects, such as large size, long duration, intensive use of bandwidth, and interactivity, conventional proxy caching techniques do not perform efficiently in this case, if not entirely inapplicable. To address these problems, many *partial caching* algorithms have been proposed in recent years [33, 34]. These algorithms demonstrated that, even if a small portion of the video is stored on the proxy, the network resources consumed could be significantly reduced. Yet how to optimally choose the portions to cache and how for the proxy and the origin server to jointly deliver streaming objects to clients remains a difficult task. The problem is further complicated by the fact that streaming objects often have variable bit rate and have stringent demands on transmission delay or delay jitter.

Security and integrity: The initial deployment of proxies is indeed for security reasons, *i.e.*, as firewalls. On the other hand, the use of proxies also complicates the task of maintaining the security. For example, for a standalone cache, how to protect against various attacks including invasion and denial-of-service? For cooperative caches, how to establish a trust model among the participants? In addition, the Secure Socket Layer (SSL) protocol is generally used 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 the functionality of the SSL and thus a remedy is needed. Finally, to alter data, an attacker can now target a proxy in addition to the origin server; it is thus crucial to ensure the integrity of the cache content in a proxy.

Other interesting issues include non-data caching (*e.g.*, TCP contentions), cache management with mobile Internet accesses, and optimal proxy placement in the wide-area networks [7, 8, 35]. In summary, the aforementioned issues offer a number of interesting research topics, and their solutions become increasingly imperative with the exponential growth of web applications and users.

REFERENCES

[1] M. Rabinovich and O. Spatscheck, *Web caching and replication*, Addison Wesley, 2002.

[2] I. Cooper, I. Melve, and G. Tomlinson, "Internet web replication and caching taxonomy," *IETF RFC 3040*, January 2001.

[3] C. Aggarwal, J. Wolf, and P. Yu, "Caching on the World Wide Web," *IEEE Transactions on Knowledge and Data Engineering*, Vol.11, No.1, pp.94-107, Jan./Feb. 1999.

[4] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: Evidence and implications," in *Proceedings of IEEE INFOCOM*'99, pp. 126-134, New York, Mar. 1999.

[5] R. Fielding *et al.*, "Hypertext Transfer Protocol - HTTP/1.1," *IETF RFC 2616*, June 1999.

[6] I. Cooper, P. Gauthier, J. Cohen, M. Dunsmuir, and C. Perkins, "The web proxy auto-discovery protocol," *IETF Internet Draft*, Nov. 2000. [Online]. Available at: http://www.wrec.org/Drafts/draft-cooper-webiwpad-00.txt

[7] P. Krishnan, D. Raz, and Y. Shavitt, "The cache location problem," *IEEE/ACM Trans. Networking*, vol. 8, pp. 568–582, Oct. 2000.

[8] J. Xu, B. Li, and D. Lee, "Placement problems for transparent data replication proxy services," *IEEE Journal on Selected Areas in Communication, Special Issue on Internet Proxy Services*, vol. 20, no. 7, pp. 1383-1398, Sept. 2002.

[9] K. S. Candan, W.-S. Li, Q. Luo, W.-P. Hsiung, and D. Agrawal, "Enabling dynamic content caching for database-driven web sites," in *Proceedings of ACM SIGMOD'01*, May 2001.

[10] J. Gwertzman and M. Seltzer, "The case for geographical push caching," in *Proceedings of the Fifth Annual Workshop on Hot Operating Systems*, pp. 51-55, Orcas Island, AW, May 1995.

[11] T. M. Kroeger, D. D. E. Long, and J. C. Mogul, "Exploring the bounds of web latency reduction from caching and prefetching," in *Proceedings of the USENIX Symposium on Internet Technology and Systems*, pp. 13-22, Dec. 1997.

[12] E. P. Markatos and C. E. Chronaki, "A top 10 approach for prefetching the Web," in *Proceedings of the INET Conference*, 1998.

[13] A. Bestavros, "Using speculation to reduce server load and service time on the WWW," in *Proceedings of ACM CIKM'95*, pp. 403-410, 1995.

[14] D. Duchamp, "Prefetching hyperlinks," in *Proceedings of the USENIX Symposium on Internet Technology and Systems*, pp. 127-138, 1999.

[15] A. Feldmann, R. Cáceres, F. Douglis, G. Glass, and M. Rabinovich, "Performance of web proxy caching in heterogeneous bandwidth environments," in *Proceedings of IEEE INFOCOM '99*, pp. 107-116, New York, NY, Apr. 1999.

[16] B. Krishnamurthy and C. E. Wills, "Study of piggyback cache validation for proxy caches in the World Wide Web," in *Proceedings of the USENIX Symposium on Internet Technology and Systems*, pp. 1-12, Dec. 1997.

[17] V. Duvvuri, P. Shenoy, and R. Tewari, "Adaptive leases: A strong consistency mechanism for the World Wide Web," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 4, Jul./Aug. 2003.

[18] P. Cao and C. Liu, "Maintaining strong cache consistency in the World-Wide Web," in *Proceedings of The 17th International Conference On Distributed Computing Systems (ICDCS'97)*, May 1997.

[19] J. Yin, L. Alvisi, M. Dahlin, and C. Lin, "Volume leases for consistency in large-scaled systems," *IEEE Transactions on Knowledge and Data Engineering,* Jan. 1999.

[20] K.-W. Lee, K. Amiri, S. Sahu, and C. Venkatramani, “Understanding the potential benefits of cooperation among proxies: Taxonomy and analysis,” *IBM Research Report, RC22173*, Sept. 2001.

[21] A. Chankhunthod, P. Danzig, C. Neerdaels, M. Schwartz, and K. Worrell, “A hierarchical Internet object cache,” in *Proceedings of the USENIX 1996 annual technical conference,* pp. 153-163, San Diego, CA, USA, Jan. 1996.

[22] R. Tewari, M. Dahlin, H M. Vin, and J. Kay, “Beyond hierarchies: Design considerations for distributed caching on the Internet,” in *Proceedings of the 19th International Conference on Distributed Computing Systems (ICDCS)*, Jun. 1999.

[23] S. Gadde, M. Rabinovich, and J. Chase, “Reduce, reuse, recycle: An approach to building large Internet caches,” in *Proceedings of IEEE HotOS-VI*, pp. 93-98, Cape Cod, MA, May 1997.

[24] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, “Summary cache: A scalable wide-area Web cache sharing protocol,” *IEEE/ACM Transactions on Networking*, vol. 8, no. 3, pp. 281-293, 2000.

[25] A. Rousskov and D. Wessels, “Cache digests,” *Computer Networks and ISDN Systems*, vol. 30, no. 22-23, pp. 2155-2168, Nov. 1998.

[26] V. Valloppillil and K. Ross, “Cache Array Routing Protocol, *v*1.1,” *IETF Internet Draft*, Feb. 1998. [Online]. Available at: http://ds1.internic.net/internet-drafts/draftvinod -carp-v1-03.txt.

[27] D. Wessels and K. C. Claffy, “ICP and the Squid Web cache,” *IEEE Journal on Selected Areas in Communications*, vo.16, no. 3, pp. 345-357, April 1998.

[28] L. Zhang, S. Floyd, and V. Jacobson, “Adaptive web caching,” in *Proceedings of the 1997 NLANR Web Cache Workshop*, Jun. 1997.

[29] Y.-T. Hou, J.-P. Pan, B. Li, and S. Panwar, “On expiration-based hierarchical caching system,” to appear in *IEEE Journal on Selected Areas in Communications, Special Issue on: Recent Advances in Service Overlay Networks*, January 2004.

[30] E. Cohen and H. Kaplan, “Aging through cascaded caches: Performance issues in the distribution of Web content,” in *Proceedings of ACM SIGCOMM’*01, 2001.

[31] H. Yu, L. Breslau, and S. Shenker, “A scalable Web cache consistency architecture,” in *Proceedings of ACM SIGCOMM ’99*, Sept. 1999.

[32] A. Ninan, P. Kulkarni, P. Shenoy, K. Ramamritham, and R. Tewari, “Cooperative leases: Scalable consistency maintenance in content distribution networks,” in *Proceedings of WWW10*, Honolulu, Hawaii, USA, May 2002.

[33] A. Dan and D. Sitaram, “Multimedia caching strategies for heterogeneous application and server environments,” *Multimedia Tools and Applications*, vol. 4, pp. 279-312, May 1997.

[34] S. Sen, J. Rexford, and D. Towsley, “Proxy prefix caching for multimedia streams,” in *Proceedings of IEEE INFOCOM’99*, New York, NY, March 1999.

[35] B. Li, M. Golin, G. Italiano, X. Deng, and K. Sohraby, “On the optimal placement of Web proxies in the Internet,” in *Proceedings of IEEE INFOCOM’99*, pp. 1282-1290, New York, USA, Mar. 1999.