A Survey of Web Caching Schemes for the Internet

Jia Wang*
Cornell Network Research Group (C/NRG)
Department of Computer Science, Cornell University
Ithaca, NY 14853-7501
jiawang@cs.cornell.edu

Abstract

The World Wide Web can be considered as a large distributed information system that provides access to shared data objects. As one of the most popular applications currently running on the Internet, the size of World Wide Web is of an exponential growth, which results in network congestion and server overloading. Web caching has been recognized as one of the effective schemes to alleviate the server bottleneck and reduce the network traffic, thereby minimize the user access latencies. In this paper, we first discribe the elements of a Web caching system and its desirable properties. Then, we survey the state-of-art techniques which have been used in Web caching systems. Finally, we discuss the research frontier in Web caching.

1 Introduction

The World Wide Web can be considered as a large distributed information system that provides access to shared data objects. The predicted size of the World Wide Web is shown in Figure 1 [5]. As the World Wide Web continues its exponential growth (the size of static Web pages increases approximately 15% per month), two of the major problems we face are the network congestion and server overloading that Web users are suffering today. The rapid growth of WWW could be attributed to the fact that at least till now, its usage is quite inexpensive, and accessing information is faster using Web than using any other resource. Also, the World Wide Web has documents that appeal to a wide range of interests, e.g. news, education, scientific research, sports, entertainment, stock market growth, travel, shopping, weather, maps, etc. Although the Internet backbone capacity increases as 60% per year, the demand for bandwidth is likely to outstrip supply for the foreseeable future as more and more information services are moved onto Web. If some kind of solution is not undertaken for its rapidly increasing growth, it would be too congested and its entire appeal would be lost.

Researchers have been working on how to improve Web performance since early 90's. Caching popular objects close to the clients has been recognized as the one of the effective solutions to alleviate Web server bottlenecks, reduce traffic over the Internet and improve the scalability of such a system. The idea of using proxy servers [50] to cache arose when they were first used to allow access to the Internet from within a firewall (see Figure 2). For security reasons, companies run a special HTTP server called a proxy on a firewall machine. It typically processes requests from within a firewall by forwarding them to the remote servers responsible, and sending the replies back to the client on receiving a response. Usually such a proxy server is used by clients on the same subnet. This leads to questions of the effectiveness of using these proxies to cache documents that are retrieved by the same group of clients. Since clients in the same group

*This material is based upon work supported under a National Science Foundation Graduate Fellowship.

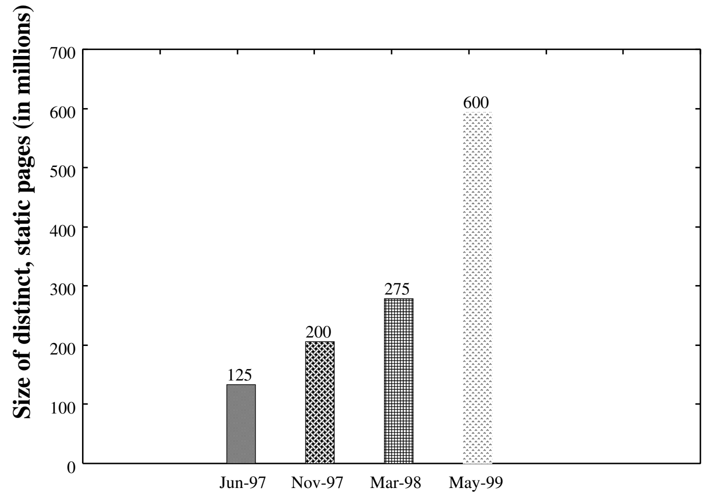

Figure 1: Size of distinct, static Web pages.

would probably be accessing the same documents and each client tends to browse back and forth in a short period of time, a document requested by a client that has been stored in the proxy cache would result in a hit, thereby not only saving network bandwidth but also lower access latency for the clients.

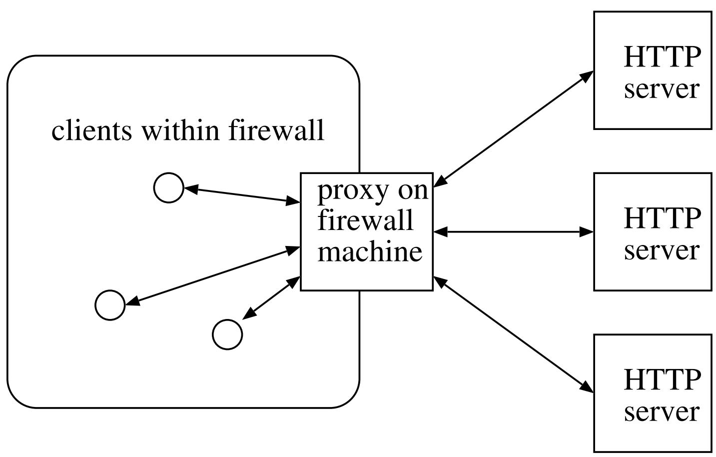

Figure 2: A proxy running on the firewall machine to service clients inside the subnet.

Documents can be cached on the client side, proxies, and the server side. The effects of Web caching are two-fold. It's shown that caching documents can improve Web performance significantly [13] [26] [42]. The advantages of Web caching are:

1. It reduces bandwidth consumption, thereby decreases network traffic and less congestion.

2. It reduces access latency due to two reasons:

   (a) By fetching frequently accessed documents from nearby proxy cache, the access latency are minimized.

   (b) Because of the reduction in network traffic, those documents not cached can also be retrieved relatively faster than without caching.

3. It reduces the workload of the remote Web server by disseminating data among the proxy caches over the wide area network.

4. If the remote server is not available due to the remote server's crash or network partitioning, it can access a cached copy at proxy. Thus, the robustness of the Web service is enhanced.

5. A side effect of Web caching is that it provides us a chance to analyze an organization's usage patterns.

Furthermore, it is shown that the cooperation of proxy caches is a powerful paradigm to improve cache effectiveness where caches cooperate both in serving each other's requests and in making storage decisions (e.g. objects placement and replacement) [40] [45]. However, it's worth to note that there are several disadvantages of using a caching system in Web services.

1. The main disadvantage is that a client might be looking at stale data due to the lack of proper updating.

2. The access latency may increase in the case of a cache miss. Hence, cache hit rate should be maximized and the cost of a cache miss should be minimized when designing a caching system.

3. A single proxy cache is always a bottleneck. A limit has to be set to the number of clients it can service, so as to be at least as efficient as when using direct contact with the remote servers.

4. A single proxy is a single point of failure.

5. Using a proxy cache will reduce the hits on the original remote server which might disappoint a lot of information providers, since they cannot maintain a true log of the hits to their pages. Hence, they might decide not to allow their documents to be cacheable.

A lot of research work have been done to study the effect of Web caching and try to maximize its benefit. There are several subtle issues on employing a caching system to facilitate Web services which need to be studied and solved (e.g. proxy location, cache routing, dynamic data caching, etc). A naive caching system can actually degrade Web performance drastically and introduce instabilities into network [30]. Intelligent and careful design is necessary to improve the quality of Web service.

The remainder of this paper is organized as follows. Section 2 outlines the elements of a World Wide Web caching system and Section 3 describes some desirable characteristics. Section 4 gives a brief survey of previous work on schemes to improve Web performance. Finally, we summarize our paper by identifying the research frontier in Web caching in Section 5.

2 Elements of a World Wide Web caching system

A generic model of Web caching system is shown in Figure 3. In such a caching system, documents can be cached at clients, proxies, and servers. A client always requests page from its local proxy if it doesn't have a valid copy of such page in its browser's cache. Upon receiving a request from client, the proxy first checks to see if it has the requested page. If so, it returns the page to the client. If it doesn't have the requested page in its cache, it sends request to its cooperative proxies. On receiving a request from another proxy, the proxy checks if it has the requested page. If so, it returns the page to the proxy issuing the request. If not, the proxy may further forward the request to other proxies or server. If none of the cooperative proxies has such page, the proxy will fetch the page from the server.

In order to make a WWW caching system working, the following problems need to be solved properly:

- How are the cache proxies organized, hierarchically, distributed, or hybrid? (caching system architecture)
- Where to place a cache proxy in order to achieve optimal performance? (proxy placement)

- What can be cached in the caching system, data, connection, and computation? (caching contents)
- How do proxies co-operate with each other? (proxy co-operation)
- What kind of data/information can be shared among co-operated proxies? (data sharing)
- How does a proxy decide where to fetch a page requested by client? (cache resolution/routing)
- How does a proxies decide which page and when to prefetch from Web server or other proxies to reduce its access latency in the future? (prefetching)
- How does a proxy manage which page to be stored in its cache and which page to be removed from its cache? (cache placement and replacement)
- How does the proxy maintain the data consistency? (cache coherency)
- How are the control information (e.g. URL routing information) distributed among proxies? (control information distribution)
- How to deal with data which is not cacheable? (dynamic data caching)

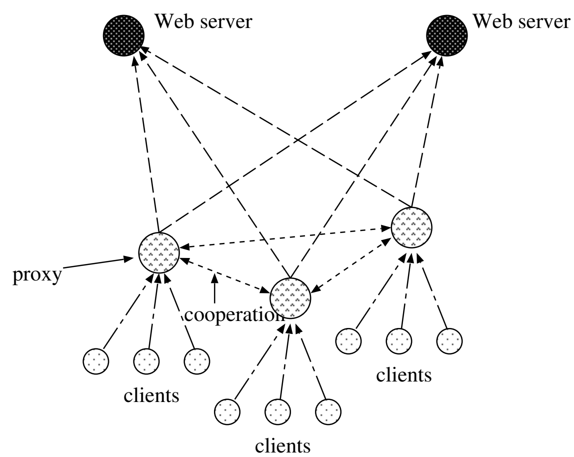

Figure 3: A generic WWW caching system.

These questions must be addressed in every reasonable caching system. Depending upon the choices made in answering each question, a variety of schemes have been proposed and these are discussed in Section 4.

3 Desirable properties of WWW caching system

Besides the obvious goal of Web caching system, we would like a Web caching system to have a number of properties. They are fast access, robustness, transparency, scalability, efficiency, adaptivity, stability, load balancing, ability to deal with heterogeneity, and simplicity. We discuss them in turn.

- *Fast access.* From users' point of view, access latency is an important measurement of the quality of Web service. A desirable caching system should aim at reducing Web access latency. In particular, it should provide user a lower latency on average than that without employing a caching system.

- *Robustness.* From users' prospect, the robustness means availability, which is another important measurement of quality of Web service. Users desire to have Web service available whenever they want. The robustness has three aspects. First, it's desirable that a few proxies crash wouldn't tear the entire system down. The caching system should eliminate the single point failure as much as possible. Second, the caching system should fall back gracefully in case of failures. Third, the caching system would be design in such a way that it's easy to recover from a failure.

- *Transparency.* A Web caching system should be transparent for the user, the only results user should notice are faster response and high availability.

- *Scalability.* We have seen an explosive growth in network size and density in last decades and is facing a more rapid increasing growth in near future. The key to success in such an environment is scalability. We would like a caching scheme to scale well along the increasing size and density of network. This requires all protocols employed in the caching system to be as lightweight as possible.

- *Efficiency.* There are two aspects to efficiency. First, how much overhead does the Web caching system impose on network? We would like a caching system to impose a minimal additional burden upon the network. This includes both control packets and extra data packets incurred by using a caching system. Second, the caching system shouldn't adopt any scheme which leads to under-utilization of critical resources in network.

- *Adaptivity.* It's desirable to make the caching system adapt to the dynamic changing of the user demand and the network environment. The adaptivity involves several aspects: cache management, cache routing, proxy placement, etc. This is essential to achieve optimal performance.

- *Stability.* The schemes used in Web caching system shouldn't introduce instabilities into the network. For example, naive cache routing based on dynamic network information will result in oscillation. Such an oscillation is not desirable since the network is under-utilization and the variance of the access latency to a proxy or server would be very high.

- *Load balancing.* It's desirable that the caching scheme distributes the load evenly through the entire network. A single proxy/server shouldn't be a bottleneck (or hot spot) and thereby degrades the performance of a portion of the network or even slow down the entire service system.

- *Ability to deal with heterogeneity.* As networks increase in scale and coverage, they span a range of hardware and software architectures. The Web caching scheme need adapt to a range of network architectures.

- *Simplicity.* Simplicity is always an asset. Simpler schemes are easier to implement and likely to be accepted as international standards. We would like an ideal Web caching mechanism to be simple to deploy.

4 Web caching schemes

Having described the attributes of an ideal Web caching system, we now survey some schemes described in the literature and point out their inadequacies.

4.1 HTTP

The HyperText Transfer Protocol (HTTP) is an application-level protocol for distributed collaborative, hypermedia information systems. HTTP has been in use by the World Wide Web global information interactive since 1990. Although HTTP can not be considered as Web caching schemes, understanding the features which HTTP provides us and making use of them in designing Web caching schemes are beneficial.

Currently two standard versions of HTTP are present: HTTP/1.0 [37] and HTTP/1.1 [38]. In spite of its popularity, it's widely understood that HTTP/1.0 has numerous flaws. HTTP/1.1 differentiates itself from HTTP/1.0 by improvement in several areas such as the extensibility, caching, bandwidth optimization, network connection management, message transmission, Internet address conservation, error notification, security, integrity, authentication, content negotiation, etc [43].

4.2 Recent Web caching systems

Several recent caching systems are briefly described here. They are:

1. Harvest cache
2. Cachemesh system
3. Summary cache
4. Adaptive Web caching
5. Access driven cache

4.2.1 Harvest cache

Harvest cache servers [14] are organized in a hierarchy with parents and siblings and they co-operate using a cache resolution protocol called Internet Cache Protocol (ICP) [80]. When a cache server receives a request for an object that misses, it sends a request to all its siblings and parents via remote procedure call. The object will be retrieved from the site with lowest latency.

4.2.2 Cachemesh system

Cachemesh is a distributed cache system for Web [75]. It uses co-operative cache placement and cache routing to achieve close and effective co-operation among a group of cache servers. In a Cachemesh system, cache servers establish a cache routing table among them, and each cache server becomes the designed cache server for a number of Web sites. User requests are then forwarded to the proper cache server according to the cache routing table. Cachemesh can significantly reduce latency, search overhead, and space utilization in a co-operating cache system.

4.2.3 Summary cache

Summary cache [29] is a scalable wide area cache sharing protocol. Each proxy improves cache sharing by keeping a compact summary of the cache directory of every participating proxy. When a client request misses in the local cache, the proxy checks these summaries for potential hits. If a hit occurs, the proxy sends request to the relevant proxies to fetch the Web page. Otherwise, the proxy sends request directly to the Web server. The proxy updates the copies of summary stored at other proxies until a certain percentage of its cached documents are not reflected in other proxies. Bloom filters [4] are used to keep each individual summary small. Experiments have shown that Summary cache reduces the number of inter-cache protocol messages, bandwidth consumption, and protocol CPU overhead significantly while maintaining almost the same cache hit ratio as ICP (Internet Caching Protocol) [80].

4.2.4 Adaptive Web caching

Adaptive Web caching [59] is an adaptive, scalable, and robust caching system. Cache servers are self-organized and form into a tight mesh of overlapping multicast groups and adapt as necessary to changing conditions. This mesh of overlapping groups forms a scalable, implicit hierarchy that is used to efficiently diffuse popular Web content towards the demand. Adaptive Web caches exchange their entire content state with other members of their cache groups to eliminate the delay and unnecessary use of resources of explicit cache probing. For each request, a cache first determines whether a copy of the requested Web page is cached by one of its group members. If not, the request will be forwarded to another cache in the Web caching infrastructure which is significantly more likely to have the data using a URL routing table maintained at each Web cache. The information in the URL routing table is learned from the source-based and content-based entries from other Web caches.

4.2.5 Access driven cache

Access driven Web caching [84] is a scheme using proxy profiles and information groups which are based on Web page access patterns to reduce the average number of messages among proxies for updating the cache status (comparing to Summary cache [29]) while maintaining a high cache hit ratio. Association rule discovery is used to find and summarize the most prevalent access patterns to Web pages from trace data. Such information is then used to partition the Web pages into clusters. All proxies which frequently access some page in the same Web page cluster form an information group. When a host wants to access a Web page, it sends a request to the local proxy. If the page is not cached in the local proxy, a request is sent to the "nearest" site (either a proxy in the same information group or the Web server) to fetch the page. When proxy cache content changes, only proxies in the same information group are notified. Furthermore, a new cache replacement algorithm is proposed, which takes into account not only the local reference pattern, but also whether a page is cached on some other proxy in the same information group. This scheme is shown to greatly reduce the number of messages and overhead on individual proxies while maintaining a high cache hit rate.

4.3 Caching architectures

The performance of Web caches depends on the size of client community connected to it; the more people use it, the higher the probability that a given document has already been requested and is presented in the cache. Caches cooperate to increase the probability to hit a document. A caching architecture should provide a paradigm that assists proxies cooperate efficiently with each other.

4.3.1 Hierarchical caching architecture

One approach to make cache cooperate is by setting up a caching hierarchy. With hierarchical caching, caches are placed at multiple levels of the network. For the sake of simplicity, we assume that there are four levels of caches: bottom, institutional, regional, and national levels. At the bottom level of the hierarchy there are the client caches. When a request is not satisfied by the client cache, the request is redirected to the institutional cache. If the document is not found at the institutional level, the request is then forwarded to the regional level cache which in turn forwards unsatisfied requests to the national level cache. If the document is not found at any cache level, the national level cache contacts directly the origin server. When the document is found, either at a cache or at the origin server, it travels down the hierarchy, leaving a copy at each of the intermediate caches. Further requests for the same document travel up the caching hierarchy until the document is hit at any cache level [70].

Hierarchical Web caching cooperation was first proposed in the Harvest project [14]. Other examples of hierarchical caching are Adaptive Web caching [59], Access Driven cache [84], etc. A hierarchical architecture is more bandwidth efficient, particularly when some co-operating cache servers do not have high speed connectivity. In such a structure, popular Web pages can be efficiently diffused towards the demand. However, there are several problems associated with a caching hierarchy [70] [72]:

1. To set up such a hierarchy, cache servers often need to be placed at the key access points in the network. This often requires significant coordination among participated cache servers.

2. Every hierarchy level may introduce additional delays.

3. Higher level caches may become bottlenecks and have long queueing delays.

4. Multiple document copies are stored at different cache levels.

4.3.2 Distributed caching architecture

Recently, a number of researchers have proposed the setup of a totally distributed caching scheme, where there are only caches at the bottom level of the network which cooperate. In distributed Web caching systems [62] [72], no intermediate caches are set up, and there are only institutional caches which serve each others' misses. In order to decide from which institutional cache to retrieve a miss document, institutional caches keep metadata information about the content of every other cooperating institutional cache. To make the distribution of the metadata information more efficient and scalable, a hierarchical distribution can be used. However, the hierarchy is only used to distribute information about the location of the documents and not to store document copies. With distributed caching most of the traffic flows through low network levels, which are less congested and no additional disk space is required at intermediate network levels. In addition, distributed caching allows better load sharing and more fault tolerant. Nevertheless, a large-scale deployment of distributed caching may encounter several problems (i.e. high connection times, higher bandwidth usage, administrative issues, etc.) [70].

There are several approaches to distributed caching. The Harvest group designed the Internet Cache Protocol (ICP) [80], which supports discovery and retrieval of documents from neighboring caches as well as parent caches. Another approach to distributed caching is the Cache Array Routing protocol (CARP) [74], which divides the URL-space among an array of loose coupled caches and lets each cache store only the documents whose URL are hashed to it. Provey and Harrison also proposed a distributed Internet cache [62]. In their scheme upper level caches are replaced by directory servers which contain location hints about the documents kept at every cache. A hierarchical metadata-hierarchy is used to make the distribution of these location hints more efficient and scalable. Tewari et al. proposed a similar approach to implement a fully distributed Internet caching system where location hints are replicated locally at the institutional caches [72]. In the central directory approach (CRISP) [33], a central mapping service ties together a certain number of caches. In Cachemesh system [75], cache servers establish a cache routing table among them, and each cache server becomes the designed cache server for a number of Web sites. User requests are then forwarded to the proper cache server according to the cache routing table. In Summary Cache [29], Cache Digest [69], and the Relais project [67], caches inter-exchange messages indicating their content and keep local directories to facilitate finding documents in other caches.

4.3.3 Hybrid caching architecture

In a hybrid scheme, where a certain number of caches cooperate at every level of a caching hierarchy using distributed caching. ICP [80] allows for cache cooperation at every level of a caching hierarchy. The document is fetched from the parent/neighbor cache with a document copy that has the lowest RTT. Rabinovich et al. [66] proposed to limit the cooperation between neighbor caches to avoid obtaining documents from distant or slower caches, which could have been retrieved directly from the origin server at a lower cost.

4.3.4 Performance of caching architectures

The main performance measure is the expected latency to retrieve a Web document. It's debatable that which caching architecture can achieve optimal performance. A recent research work [70] shows that hierarchical caching has shorter connection times than distributed caching, and hence, placing additional copies at intermediate levels reduces the retrieval latency for small documents. It's also shown that distributed caching has shorter transmission times and higher bandwidth usage than hierarchical caching. A "well configured" hybrid scheme can combine the advantages of both hierarchical and distributed caching, reducing the connection time as well as the transmission time.

4.4 Cache resolution/routing

Scalability and deployment concerns have led most designers of Web caching infrastructures to design schemes based on deploying a large number of Web caches scattered around the Internet. The main challenge in such approaches is being able to quickly find a cache containing the desired document. While this problem is similar to the general problem of network routing, it cannot be solved in the same way. Conventional routing protocols scale because of the route aggregation made possible by hierarchical addressing. However, since documents with the same URL prefixes or server address prefixes will not necessarily be delivered to the same clients, there is no necessary location among their cache locations. With no way to aggregate routes, the cache routing tables would be unmanageably large. In addition, they have to be updated frequently. This lead to cache misses when proxies don't have updated cache routing information. In order to minimize the cost of a cache miss, an ideal cache routing algorithm should route requests to the next proxy which is believed to cache the desired document along (or close to) the path from the client to the Web server.

The common approach is to grow a caching distribution tree away from each popular server towards sources of high demand and do cache resolution either via *cache routing table* or via *hash functions*. This works well for requests for very popular documents, because these documents will propagate out to many caches, and so will be found quickly. For less popular documents, the search may follow a long and circuitous path of numerous failed checks. The impact of this is substantial since the hit rate on Web caches is typically less than 50%, indicating a large number of documents of only low to moderate popularity [2].

4.4.1 Cache routing table

Malpani et al. [58] work around this problem by making a group of caches function as one. A user's request for a page is directed to an arbitrary cache. If the page is stored there, it's returned to the user. Otherwise, the cache forwards the requests to all other caches via IP multicast. If the page is cached nowhere, the request is forwarded to the home site of the page.

Harvest cache system [14] organizes caches in a hierarchy and uses a cache resolution protocol called Internet Cache Protocol (ICP) [80]. Requests for Web documents are forwarded up the hierarchy in search of a cached copy. In attempt to keep from overloading caches at the root, caches query their siblings before passing requests upwards.

Adaptive Web Caching [59] uses a mesh of caches in which distribution trees for each server are built. The caches in the mesh are organized into overlapping multicast groups through which a request travels in search of a cached document. This scheme benefits from constructing different distribution trees for different servers (so no root node will be overloaded) and being robust and self-configuring. For less popular objects, queries travel through many caches, and each check requires a query to and responses from a group of machines. The authors suggest dealing with this problem by limiting the number of caches a request will access.

Provey and Harrison [62] construct a manually configured hierarchy that must be traversed by all requests. Their scheme is promising in that it reduces load on top-level caches by only keeping location pointers in the hierarchy.

Wang [75] describes a preliminary plan in Cachemesh system to put cache routing tables in caches to specify, for each page or server, where to look next if the local cache does not hold the document. A default route for some documents would help to keep table size reasonable.

To reduce the time needed to find relatively unpopular, but cached, documents and reduce the latency of searching for documents that are not cached, Legedza and Guttag [52] integrate the routing of requests to caches with the routing services already provided within the network.

4.4.2 Hashing function

The Cache Array Routing Protocol (CARP) [74] allows for "queryless" distributed caching by using a hash function based upon the "array membership list" and URL to provide the exact cache location of an object, or where it will be cached upon downloading from the Internet. When one proxy server is added or removed, $1/n$ URLs need to be reassigned and the new hash function need to be distributed among proxies, where n is the number of proxy servers.

In Summary cache [29], each proxy keeps a summary of the URLs of cached documents of each participating proxy and checks these summaries for potential hits before sending any queries. To reduce the overhead, the summaries are stored as a Bloom filter [4] and updated only periodically. Experiments have shown that Summary cache reduces the number of inter-cache protocol messages, bandwidth consumption, and protocol CPU overhead significantly while maintaining almost the same cache hit ratio as ICP (Internet Caching Protocol) [80].

Karger et al. [41] describe a theoretically based technique for constructing per-server distribution trees with good load balancing properties using consistent hashing.

4.5 Prefetching

Although Web performance is improved by caching documents at proxies, the benefit from this technique is limited [25] [42]. Previous research shown that the maximum cache hit rate that can be achieved by any caching algorithm is usually no more than 40% to 50%, that is, regardless of the caching scheme in use, one out of two documents can not be found in the cache [2]. One way to further increase the cache hit rate is to anticipate future document requests and preload or prefetch these documents in a local cache.

Prefetching can be applied in three ways in the Web contexts:

1. Between browser clients and Web servers.

2. Between proxies and Web servers.

3. Between browser clients and proxies.

4.5.1 Between browser clients and Web servers

Early studies focus on the prefetching schemes between browser clients and Web servers. Padmanabhan and Mogul [64] analyze the latency reduction and network traffic of prefetching using Web server traces and trace-driven simulation. The prediction algorithm they used is based on the Prediction-by-Partial-Matching (PPM) data compressor with prefix depth of 1. The study shows that prefetching from Web servers to individual clients can reduce client latency by 45% at the expense of doubling the network traffic. Bestavros and Cunha [7] presents a model for speculative dissemination of World Wide Web documents. The work shows that reference patterns observed at a Web server can be used as an effective source of information to drive prefetching, and reaches similar results as [63]. Cunha and Jaccoud use [17] a collection of Web client traces and study how effectively a user's future Web accesses can be predicted from his or her past Web accesses. They show that a number of models work well and can be used in prefetching. Crovella and Barford [12] analyzed the network effects of prefetching and shows that prefetching can reduce access latency at the cost of increasing network traffic and increasing network traffic burstiness (and thereby increasing network delays). They proposed a rate-controlled prefetching scheme to minimize the negative network effect by minimizing the transmission rate of prefetched documents. However, these early studies do not consider or model caching proxies and hence fail to answer the question about performance of prefetching completely.

4.5.2 Between proxies and Web servers

After proxies have been used to assist Web access, research interest has shifted to investigate prefetching techniques between proxies and Web servers. Kroeger et al. [42] investigates the performance limits of prefetching between Web servers and proxies, and shows that combining perfect caching and perfect prefetching at the proxies can at least reduce the client latency by 60% for high bandwidth clients. Markatos and Chronaki [57] propose that Web servers regularly push their most popular documents to Web proxies, which then push those documents to the clients. They evaluate the performance of the strategy using several Web server traces and find that the technique can anticipate more than 40% of a client's request. The technique requires cooperation from the Web servers. The study did not evaluate client latency reduction from the technique. Cohen et al. [18] also investigates similar techniques. Wcol [22] is a proxy software that prefetches documents links and embedded images. The proxy, however, does not push the documents to the client. Gwertzman and Seltzer [34] discuss a technique called Geographical Push-Caching where a Web server selectively sends it documents to the caches that are closest to its clients. The focus of the study is on deriving reasonably accurate network topology information and using the information to select caches.

4.5.3 Between browser clients and proxies

Prefetching can also be done between browser clients and proxies. Loon and Bharghavan [51] proposed a design and an implementation of a proxy system that performs the prefetching, image filtering, and hoarding for mobile clients. Fan et al. [31] proposed an approach to reduce latency by prefetching between caching proxies and browsers. The approach relies on the proxy to predict which cached documents a user might reference next (based on PPM data compressor), and takes advantage of idle time between user requests to either push or pull the documents to the user. Simulation results show that prefetching combined with large browser cache and delta-compression can reduce client latency up to 23.4%.

4.5.4 Summary

The first two approaches run the risk of increased wide area network traffic, while the last one only affects the traffic over the modems or the LAN. All of these approaches attempt to prefetch either documents that are considered as popular pages at servers or documents that are predicted to be accessed by user in the near future based on the access pattern.

4.6 Cache placement/replacement

A key aspect of the effectiveness of proxy caches is a document placement/replacement algorithm that can yield high hit rate. While cache placement has not been well studied, a number of cache replacement algorithms have been proposed in recent studies, which attempt to minimize various cost metrics, such as hit rate, byte hit rate, average latency, and total cost. They can be classified into the following three categories.

1. Traditional replacement policies and its direct extensions:
   - *Least Recently Used* (LRU) evicts the object which was requested the least recently.
   - *Lease Frequently used* (LFU) evicts the object which is accessed least frequently.
   - *Pitkow/Recker* [79] evicts objects in LRU order, except if all objects are accessed today, in which case the largest one is removed.
2. Key-based replacement policies: (i.e. the replacement policies in this category evict objects based upon a primary key, break ties based on secondary key, tertiary key, etc.)
   - *Size* [79] evicts the largest object.
   - *LRU-MIN* [2] biased in favor of smaller objects. If there are any objects in the cache which have size at least S, LRU-MIN evicts the least recently used such object from the cache. If there are no objects with size at least S, then LRU-MIN starts evicting objects in LRU order of size at least $S/2$. That is, the object who has the largest log(size) and is the least recently used object among all objects with the same log(size) will be evicted first.
   - *LRU-Threshold* [2] is the same as LRU, but objects larger than a certain threshold size are never cached.
   - *Hyper-G* [79] is a refinement of LFU, break ties using the recency of last use and size.
   - *Lowest Latency First* [78] minimizes average latency by evicting the document with the lowest download latency first.
3. Cost-based replacement policies: (i.e. the replacement policies in this category employ a potentially cost function of the different factors such as time since last access, entry time of the object in the cache, transfer time cost, object expiration time and so on.)
   - *GreedyDual-Size* (GD-Size) associates a cost with each object and evicts object with the lowest cost/size.
   - *Hybrid* [78] associates a utility function with each object and evicts the one has the least utility to reduce the total latency.
   - *Lowest Relative Value* [55] evicts the object with the lowest utility value.
   - *Least Normalized Cost Replacement* (LCN-R) [71] employs a rational function of the access frequency, the transfer time cost and the size.
   - *Bolot/Hoschka* [10] employs a weighted rational function of transfer time cost, the size, and the time last access.
   - *Size-Adjusted LRU* (SLRU) [3] orders the object by ratio of cost to size and choose objects with the best cost-to-size ratio.

- *Server-assisted* scheme [19] models the value of caching an object in terms of its fetching cost, size, next request time, and cache prices during the time period between requests. It evicts the object of the least value.
- *Hierarchical GreedyDual* (Hierarchical GD) [40] does object placement and replacement cooperatively in a hierarchy.

4.7 Cache coherency

Caches provide decreased access latency at a cost: every cache will sometimes provide users with *stale* pages - pages which are out of date with respect to their master copies on the Web servers where the pages originated [27]. Every Web cache must somehow update pages in its cache so that it can give users pages which are as fresh as possible. Caching and the problem of cache coherency on the World Wide Web are similar to the problems of caching in distributed file systems. However, the Web is different than a distributed file system in its access patterns, its larger scale, and its single point of updates for Web objects [35].

4.7.1 HTTP commands that assist Web proxies in maintaining cache coherence

Before dealing with the cache coherence mechanisms, we first give a brief overview about the commands that HTTP [37] provides to assist Web proxies in maintaining cache coherence.

1. *HTTP GET*. Retrieves a document given its URL.
2. *Conditional GET*. HTTP *GET* combined with the header *IF-Modified-Since*. *date* can be used by proxy to tell a remote server to return a copy only if it has been modified.
3. *Pragma:no-cache*. This header appended to *GET* can indicate that the object is to be reloaded from the server irrespective of whether it has been modified or not. Most browsers like Netscape offer a *Reload* button which uses this header to retrieve the original copy.
4. *Last-Modified:date*. Returned with every *GET* message and indicates the last time the page was modified.
5. *Date:date*. The last time the object was considered to be fresh; this is different from the *Last-Modified* header. Netscape, the most popular browser, does not provide a mechanism for displaying the value of a page's *Date* header in the *Document Info* window.

The above has been used by the clients via Web proxies. If a remote server receives a *Conditional GET* request but does not support it, it just sends the entire document. However, Loutonen and Altis reported in [50] that at least all major HTTP servers already support the *Conditional GET* header.

4.7.2 Cache coherence mechanisms

Current cache coherency schemes providing two types of consistency, *strong cache consistency* and *weak cache consistency*, have been proposed and investigated for caches on the World Wide Web.

1. Strong cache consistency
   (a) *Client validation*. This approach is also called *polling-every-time*.The proxy treats cached resources as potentially out-of-date on each access and sends an *If-Modified-Since* header with each access of the resources. This approach can lead to many 304 responses (HTTP response code for "*Not Modified*") by server if the resource does not actually change.

(b) *Server invalidation*. Where upon detecting a resources change, the server sends invalidation messages to all clients that have recently accessed and potentially cached the resource [21]. This approach requires a server to keep track of lists of clients to use for invalidating cached copies of changed resources and can become unwieldy for a server when the number of clients is large. In addition, the lists can become out-of-date causing the server to send invalidation messages to clients who are no longer caching the resource.

2. Weak cache consistency

   (a) *Adaptive TTL*. The adaptive TTL (also called Alex protocol [11]) handles the problem by adjusting a document's time-to-live based on observations of its lifetime. Adaptive TTL takes advantage of the fact that file lifetime distributions tend to be bimodal; if a file has not been modified for a long time, it tends to stay unchanged. Thus, the time-to-live attribute to a document is assigned to be a percentage of the document's current "age", which is the current time minus the last modified time of the document. Studies [11] [34] have shown that adaptive TTL can keep the probability of stale documents within reasonable bounds ($< 5\%$). Most proxy servers (e.g. CERN *httpd* [50] [76]) use this mechanism. The Harvest cache [14] mainly uses this approach to maintain cache consistency, with the percentage set to 50%. However, there are several problems with this expiration-based coherence [27]. First, user must wait for expiration checks to occur even though they are tolerant to the staleness of the requested page. Second, if a user is not satisfied with the staleness of a returned document, they have no choice but to use a *Pragma:No-Cache* request to load the entire document from its home site. Third, the mechanism provides no strong guarantee as to document staleness. Forth, users cannot specify the degree of staleness they are willing to tolerate. Finally, when the user aborts a document load, caches often abort a document load as well.

   (b) *Piggyback Invalidation*. Krishnamurthy et al. proposed piggyback invalidation mechanisms to improve the effectiveness of the cache coherency [18] [46] [47] [48]. Three invalidation mechanisms are proposed. The Piggyback Cache Validation (PCV) [46] capitalizes on requests sent from the proxy cache to the server to improve coherency. In the simplest case, whenever a proxy cache has a reason to communicate with a server it piggybacks a list of cached, but potentially stale, resources from that server for validation. The basic idea of the Piggyback Server Invalidation (PSI) mechanism [47] is for servers to piggy back on a reply to a proxy, the list of resources that have changed since the last access by the client. The proxy invalidates cached entries on the list and can extend the lifetime of entries not on the list. They also proposed a hybrid approach which combined the PSI and PCV techniques to achieve the best overall performance [48]. The choice of the mechanism depends on the time since the proxy last requested invalidation for the volume [18]. If the time is small, then the PSI mechanism is used, while for longer gaps the PCV mechanism is used to explicitly validate cache contents. The rationale is that for short gaps, the number of invalidation sent with PSI is relatively small, but as the time grows larger the overhead for sending invalidation will be larger than the overhead for requesting validations.

4.8 Caching contents

Cache proxy has been recognized as an effective mechanism to improve Web performance. A proxy may serve in three roles: *data cache*, *connection cache*, and *computation cache*. A recent study shown that caching Web page at proxy reduces the user access latency 3% - 5% relative to the no-proxy scheme. In presence of P-HTTP, a proxy can be used as a connection cache. By using persistent connections between clients and proxy and between proxy and Web server, the total latency improvements grow substantially (i.e. 20% - 40%) [13] [30].

Computation caching can be viewed as the Web server replicates and migrates some of its services to the proxies to alleviate the server bottleneck. One application of such computation caching is dynamic data caching. The motivation is that, in presence of current caching schemes, the hit ratio at proxy is at most 50%, which is limited by the fact that a significant percentage of Web pages is dynamically generated and thereby is not cacheable. Computation caching can be used to improve the performance to retrieve dynamic data by caching dynamic data at proxies and migrating a small piece of computation to proxies [16] [23] [54] to generate or maintain the cached data.

4.9 User access pattern prediction

Proxy's policies for managing cached resources (i.e. prefetching, coherence, placement and replacement) and TCP connections rely on the assumptions about client access pattern. To improve the information exchanges between Web servers and proxies, a variety of techniques have been proposed to predict the future requests. One set of approaches are to group resources that likely to be accessed together based on the likelihood that pairs of resources are accessed together, server file system structure, etc [20] [84]. Others are using Prediction by Partial Match (PPM) model to predict which page is likely to be accessed in near future [31] [63] [64].

4.10 Load balancing

Many of us have experienced the hot spot phenomenon in the context of Web. Hot spots occur any time a large number of clients wish to simultaneously access data or get some services from a single server. If the site is not provisioned to deal with all of these clients simultaneously, service may be degraded or lost. Several approaches to overcoming the hot spots have been proposed. Most use some kind of replication strategy to store copies of hot pages/services throughout the Internet; this spreads the work of serving a hot page/service across several servers (i.e. proxies) [14] [36] [58] [65].

4.11 Proxy placement

The placement of proxies is also important to achieve optimal Web performance. The desirable properties of such proxy placement policies are self-organizing, efficient routing, efficient placement, load balancing, stable behavior, etc. However, little study has been done to address this issue. Li et al. [53] attempted to solve it based on the assumptions that the underlying network topologies are minimum spanning tree and modeled it as a dynamic programming problem.

4.12 Dynamic data caching

As we mentioned before, the benefit of current Web caching schemes are limited by the fraction of total data accessed by users which are cacheable. Non-cacheable data (i.e. personalized data, authenticated data, server dynamically generated data, etc.) is of a significant percentage of the total data. For example, measurement results show that 30% of user requests contain cookies [13] [30]. How to make more data cacheable and how to reduce the latency to access non-cacheable data have become crucial problems in order to improve Web performance. Current approaches can be classified into two categories: *active cache* and *server accelerator*.

Active cache [23] supports caching of dynamic documents at Web proxies by allowing servers to supply cache applets to be attached with documents and requiring proxies to invoke cache applets upon cache hits to finish the necessary processing without contacting the server. It's shown that Active cache scheme can result in significant network bandwidth savings at the expense of CPU costs. However, due to the significant CPU overhead, the user access latencies are much larger than that without caching dynamic objects.

Web server accelerator [54] resides in front of one or more Web servers to speed up user accesses. It provides an API which allows application programs to explicitly add, delete, and update cached data. The API allows the accelerator to cache dynamic as well as static data. Invalidating and updating cached data is facilitated by the Data Update Propagation (DUP) algorithm which maintains data dependence information between cached data ad underlying data which affect their values in a graph [16].

4.13 Web traffic characteristics

Understanding the nature of the workloads and system demands created by users of the World Wide Web is crucial to properly designing and provisioning Web services. The effectiveness of caching schemes relies on the presence of temporal locality in Web reference streams and on the use of appropriate cache management policies that appropriate for Web workloads. A number of measurements have been done to exploit the access properties at clients, proxies, and servers [1] [6] [8] [25] [26].

5 Conclusion

As Web service becomes more and more popular, users are suffering network congestion and server overloading. Great efforts have been made to improve Web performance. Web caching is recognized to be one of the effective techniques to alleviate server bottleneck and reduce network traffic, thereby minimize the user access latecny. In this paper, we give an overview of recent Web caching schemes. By surveying previous work on Web caching, we notice that there are still some open problems in Web caching such as proxy placement, cache routing, dynamic data caching, fault tolerant, security, etc. The research frontier in Web performance improvement lies in developing efficient, scalable, robust, adaptive, stable Web caching scheme that can be easily deployed in current and future network.

References

[1] G. Abdulla, E. A. Fox, M. Abrams, and S. Williams, WWW proxy traffic characterization with application to caching (http://csgrad.cs.vt.edu/ abdulla/proxy/proxy-char.ps).

[2] M. Abrams, C. R. Standridge, G. Abdulla, S. Williams, and E. A. Fox, Caching proxies: limitations and potentials, Proceedings of the 4th International WWW Conference, Boston, MA, Dec. 1995.

[3] C. Aggarwal, J. L. Wolf, and P. S. Yu, Caching on the World Wide Web, IEEE Transactions on Knowledge and data Engineering, Vol. 11, No. 1, January/February 1999.

[4] B. Bloom, Space/time trade-offs in hash coding with allowable errors, Communications of ACM, 13(7), pp. 422-426, July 1970.

[5] K. Bharat and A. Broder, Measuring the Web (http//www.research.digital.com/SRC/whatsnew/sem.html).

[6] P. Barford, A. Bestavros, A. Bradley, and M. E. Crovella, Changes in Web client access patterns: characteristics and caching implications, World Wide Web (special issue on Characterization and Performance Evaluation), 1999.

[7] A. Bestavros and C. Cunha, Server-initiated document dissemination for the WWW, IEEE Data Engineering Bulletin, Sept. 1996.

[8] L. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker, Web caching and Zipf-like distributions: evidence and implications, Proceedings of Infocom'99.

[9] S. Bhattacharjee, K. Calvert, and E. W. Zegura, Self-organizing wide-area network caches, IEEE Infocom'98, April 1998.

[10] J. C. Bolot and P. Hoschka, Performance engineering of the World-Wide Web: Application to dimensioning and cache design, Proceedings of the 5th International WWW Conference, Paris, France, May 1996.

[11] V. Cate, Alex - a global file system, Proceedings of the 1992 USENIX File System Workshop, pp. 1-12, May 1992.

[12] M. Crovella and P. Batford, The network effects of prefetching, Proceedings of Infocom'98.

[13] R. Caceres, F. Douglis, A. Feldmann, G. Glass, and M. Rabinovich, Web proxy caching: the devil is in the details, ACM Performance Evaluation Review, 26(3): pp. 11-15, December 1998.

[14] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrel, A hierarchical Internet object cache, Usenix'96, January 1996.

[15] P. Cao and S. Irani, Cost-aware WWW proxy caching algorithms, Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems (USITS-97), Monterey, CA, Dec. 1997.

[16] J. Challenger, A. Iyengar, and P. Dantzig, A scalable system for consistently caching dynamic Web data, Proceedings of Infocom'99.

[17] C. R. Cunha, and C. F. B. Jaccoud, Determining WWW user's next access and its application to pre-fetching, Proceedings of ISCC'97: The second IEEE Symposium on Computers and Communications, July 1997.

[18] E. Cohen, B. Krishnamurthy, and J. Rexford, Improving end-to-end performance of the Web using server volumes and proxy filters, Proceedings of Sigcomm'98.

[19] E. Cohen, B. Krishnamurthy, and J. Rexford, Evaluating server-assisted cache replacement in the Web, Proceedings of the European Symposium on Algorithms-98, 1998.

[20] E. Cohen, B. Krishnamurthy, and J. Rexford, Efficient algorithms for predicting requests to Web servers, Proceedings of Infocom'99.

[21] P. Cao and C. Liu, Maintaining strong cache consistency in the World Wide Web, Proceedings of the 17th IEEE International Conference on Distributed Computing Systems, May 1997.

[22] K. Chinen and S. Yamaguchi, An interactive prefetching proxy server for improvement of WWW latency, Proceedings of INET'97, June 1997.

[23] P. Cao, J. Zhang, and K. Beach, Active cache: caching dynamic contents on the Web, Proceedings of IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'98), pp. 373-388.

[24] G. V. Dias, G. Cope, and R. Wijayaratne, A smart Internet caching system (http://www.isoc.org.ar/inet96/proc/a4/a4_3.htm).

[25] F. Douglis, A. Feldmann, B. Krishnamurthy, and J. Mogul, Rate of change and other metrics: a live study of the World-Wide Web, Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems (USITS-97), Dec. 1997.

[26] B. M. Duska, D. Marwood, and M. J. Feelay, The measured access characteristics of World Wide Web client proxy caches, Proceedings of USENIX Symposium on Internet Technologies and Systems (http://cs.ubc.ca/spider/feeley/wwwap/wwwap.html).

[27] A. Dingle and T. Partl, Web cache coherence, Fifth International World Wide Web Conference, Paris, France, 1996.

[28] D. Ewing, R. Hall, and M. Schwartx, A measurement study of Internet file transfer traffic, Technical Report CU-CS-571-92, University of Colorado, Dept. of Computer Science, Boulder, Colorado, January 1992.

[29] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, Summary cache: a scalable wide-area Web cache sharing protocol, Proceedings of Sigcomm'98.

[30] A. Feldmann, R. Caceres, F. Douglis, G. Glass, and M. Rabinovich, Performance of Web proxy caching in heterogeneous bandwidth environments, Proceedings of Infocom'99.

[31] L. Fan, P. Cao, W. Lin, and Q. Jacobson, Web prefetching between low-bandwidth clients and proxies: potential and performance, Proceedings of the Sigmetrics'99.

[32] S. Galssman, A cache relay for the WWW, Proceedings of the 1st International WWW Conference, Geneva, Switzerland, May 1994 (http://www.research.digital.com/SRC/personal/Steve_Glassman/CachingTheWeb.ps).

[33] S. Gadde, M. Rabinovich, and J. Chase, Reduce, reuse, recycle: an approach to building large Internet caches, Proceedings of the HotOS'97 Workshop, May 1997 (http://www.cs.duke.edu/ari/cisi/crisp-recycle/crisp-recycle.htm).

[34] J. Gwetzman and M. Seltzer, The case for geographical pushing-caching, HotOS Conference, 1994 (ftp://das-ftp.harvadr.edn/techreports/tr-34-94.ps.gz).

[35] J. Gwetzman and M. Seltzer, World Wide Web cache consistency, Proceedings of the USENIX Technical Conference, pp. 141-152, January 1996.

[36] A. Heddaya, S. Mirdrad, and D. Yates, Diffusion based caching along routing paths (http://cs-www.bu.edu/faculty/heddaya/Pepers-NonTR/webcache-wkp.ps.Z).

[37] Hypertext Transfer Protocol – HTTP/1.0, RFC 1945.

[38] Hypertext Transfer Protocol – HTTP/1.1, RFC 2068.

[39] J. Jung and K. Chon, Nation-wide caching project in Korea - design and experimentation, Proceedings of the 2nd Web Cache Workshop (http://ircache.nlanr.net/Cache/Workshop97/Papers/Jaeyeon/jaeyeon.html).

[40] M. R. Korupolu and M. Dahlin, Coordinated placement and replacement for large-scale distributed caches, Proceedings of the IEEE Workshop on Internet Applications, July 1999 (Technical Report TR-98-30, Department of Computer Science, University of Texas at Austin, December 1998).

[41] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy, Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web, STOC 1997.

[42] T. M. Kroeger, D. D. E. Long, and J. C. Mogul, Exploring the bounds of Web latency reduction from caching and prefetching, Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems, Monterey, CA, Dec. 1997.

[43] B. Krishnamurthy, J. C. Mogul, and D. M. Kristol, Key differences between HTTP/1.0 and HTTP/1.1, Proceedings of WWW-8, Toronto, May 1999.

[44] M. R. Korupolu, C. G. Plaxton and R. Rajaraman, Placement algorithms for hierarchical cooperative caching, Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, January 1999.

[45] P. Krishnan and B. Sugla, Utility of co-operating Web proxy caches, Computer Networks and ISDN Systems, pp. 195-203, April 1998.

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

[47] B. Krishnamurthy and C. E. Wills, Piggyback server invalidation for proxy cache coherency, Proceedings of the WWW-7 Conference, pp. 185-194, 1998.

[48] B. Krishnamurthy and C. E. Wills, Proxy cache coherency and replacement - towards a more complete picture, ICDC99, June 1999.

[49] I. Lovric, Internet cache protocol extension, Internet Draft <draft-lovric-icp-ext-01.txt>.

[50] A. Luotonen and K. Altis, World Wide Web proxies, Computer Networks and ISDN Systems, First International Conference on WWW, April 1994.

[51] T. S. Loon and V. Bharghavan, Alleviating the latency and bandwidth problems in WWW browsing, Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems (USITS-97), Dec. 1997.

[52] U. Legedza and J. Guttag, Using network-level support to improve cache routing, Computer Networks and ISDN Systems 30, 22-23, pp. 2193-2201, Nov. 1998.

[53] B. Li, M. J. Golin, G. F. Italiano, X. Deng, and K. Sohraby, On the optimal placement of Web proxies in the Internet, Proceedings of Infocom'99.

[54] E. Levy-Abegnoli, A. Iyengar, J. Song, and D. Dias, Design and performance of Web server accelerator, Proceedings of Infocom'99.

[55] P. Lorenzetti, L. Rizzo, and L. Vicisano, Replacement policies for a proxy cache (http://www.iet.unipi.it/luigi/research.html).

[56] I. Melve, Client-cache communication, Internet Draft <draft-melve-clientcache-com-00.txt>.

[57] E. P. Markatos and C. E. Chronaki, A TOP-10 approach to prefetching on Web, Proceedings of INET'98.

[58] R. Malpani, J. Lorch, and D. Berger, Making World Wide Web caching servers cooperate, Proceedings of the 4th International WWW Conference, Boston, MA, Dec. 1995 (http://www.w3j.com/1/lorch.059/paper/059.html).

[59] S. Michel, K. Nguyen, A. Rosenstein, L. Zhang, S. Floyd and V. Jacobson, Adaptive Web caching: towards a new caching architecture, Computer Network and ISDN Systems, November 1998.

[60] I. Melve, L. Slettjord, T. Verschuren, and H. Bekker, Building a Web caching system - architectural considerations, Proceedings of the 8th Joint European Networking Conference, Edinburgh, Scotland, May 1997.

[61] M. Nabeshima, The Japan cache project: an experiment on domain cache, Computer Networks and ISDN System, September 1997.

[62] D. Povey and J. Harrison, A distributed Internet cache, Proceedings of the 20th Australian Computer Science Conference, Sydney, Australia, Feb. 1997.

[63] T. Palpanas and A. Mendelzon, Web prefetching using partial match prediction, Proceedings of WCW'99.

[64] V. N. Padmanabhan and J. C. Mogul, Using predictive prefetching to improve World Wide Web latency, proceedings of Sigcomm'96.

[65] M. Rabinovich, Issues in Web content replication.

[66] M. Rabinovich, J. Chase, and S. Gadde, Not all hits are created equal: cooperative proxy caching over a wide-area network, Computer Networks And ISDN Systems 30, 22-23, pp. 2253-2259, Nov. 1998.

[67] Relais: cooperative caches for the World Wide Web, 1998 (http://www-sor.inria.fr/projects/relais/).

[68] C. Roadknight and I. Marshall, Variations in cache behavior, Computer Networks and ISDN Systems, pp. 733-735, April 1998.

[69] A. Rousskov and D. Wessels, Cache Digest, Proceedings of 3rd International WWW Caching Workshop, June 1998.

[70] P. Rodriguez, C. Spanner, and E. W. Biersack, Web caching architectures: hierarchical and distributed caching, Proceedings of WCW'99.

[71] P. Scheuermann, J. Shim, and R. Vingralek, A case for delay-conscious caching of Web documents, Proceedings of the 6th International WWW Conference, Santa Clara, Apr. 1997.

[72] R. Tewari, M. Dahlin, H. Vin, and J. Kay, Beyond hierarchies: design considerations for distributed caching on the Internet, Technical Report TR98-04, Department of Computer Science, University od Texas at Austin, February 1998.

[73] R. Tewari, H. Vin, A. Dan, and D. Sitaram, Resource based caching for Web servers, Proceedings of SPIE/ACM Conference on Multimedia Computing and Networking (MMCN), January 1998.

[74] V. Valloppillil and K. W. Ross, Cache array routing protocol v1.0, Internet Draft <draft-vinod-carp-v1-03.txt>.

[75] Z. Wang, Cachemesh: a distributed cache system for World Wide Web, Web Cache Workshop, 1997.

[76] D. Wessels, Intelligent caching for World-Wide Web objects, Proceedings of INET'95, Honolulu, Hawaii, June 1995 (http://info.isoc.org/HMP/PAPER/139/archive/papers.ps.9505216).

[77] K. J. Worrell, Invalidation in large scale network object caches, M.S. Thesis, Department of Computer Science, University of Colorado, Boulder, Colorado, December 1994 (ftp://ftp.cs.colorado.edu/pub/cs/techreports/schwartz/WorrellThesis.ps.Z).

[78] R. P. Wooster and M. Abrams, Proxy caching that estimates page load delays, Proceedings of the 6th International WWW Conference, April 1997 (http://www.cs.vt.edu/ chitra/docs/www6r/).

[79] S. Williams, M. Abrams, C. R. Standridge, G. Abdulla, and E. A. Fox, Removal policies in network caches for World-Wide Web documents, Proceedings of Sigcomm'96.

[80] D. Wessels and K. Claffy, Internet cache protocol (IPC), version 2, RFC 2186.

[81] D. Wessels and K. Claffy, Application of Internet cache protocol (IPC), version 2, RFC 2187.

[82] J. Yin, L. Alvisi, M. Dahlin, and C. Lin, Using leases to support server-driven consistency in large-scale systems, Proceedings of the 18th International Conference on Distributed Computing System (ICDCS'98), May 1998.

[83] P. S. Yu and E. A. MacNair, Performance study of a collaborative method for hierarchical caching in proxy servers, Computer Networks and ISDN Systems, pp. 215-224, April 1998.

[84] J. Yang, W. Wang, R. Muntz, and J. Wang, Access driven Web caching, UCLA Technical Report #990007.