



ACADEMIC
PRESS

Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE @ DIRECT®

J. Parallel Distrib. Comput. 63 (2003) 945–962

Journal of
Parallel and
Distributed
Computing

<http://www.elsevier.com/locate/jpdc>

On scalable and locality-aware web document sharing

Li Xiao,^a Xin Chen,^b Xiaodong Zhang,^{b,*} and Yunhao Liu^a

^aDepartment of Computer Science and Engineering, Michigan State University, East Lansing, MI 48824-1226, USA

^bDepartment of Computer Science, College of William and Mary, P.O. Box 8795, Williamsburg, VA 23187-8795, USA

Received 14 May 2003

Abstract

We propose a scalable Web document sharing infrastructure model called Browsers-Aware Proxy Server. In this design, a proxy server connecting to a group of networked clients maintains an index file of data objects of all clients' browser caches. If a user request misses in its local browser cache and the proxy cache, the browsers-aware proxy server will search the index file attempting to find it in another client's browser cache before sending the request to an upper level proxy or the Web server. If such a request does hit in a remote client, this client will directly forward the data object to the requesting client; or the proxy server fetches the data object from this client and then forwards it to the requesting client. The contributions of this caching model are twofold. First, we show that the amount of sharable data in browser caches is significant and can be utilized for document sharing among clients to improve Web caching performance and scalability. Second, the browsers-aware model can effectively and further improve Web prefetching performance. The browsers-aware model and its supported prefetching technique build a strong locality-aware Internet environment to make Web accesses fast with low communication costs. Conducting trace-driven simulations, we show the effectiveness of the browsers-aware model, and its unique advantages to facilitate prefetching.

© 2003 Elsevier Inc. All rights reserved.

Keywords: Browsers; Prefetching; Proxy caching; Trace-driven simulations; Web servers

1. Introduction

Proxy caching is an effective solution to quickly access the cached data on the client side and to reduce Internet traffic to Web servers. A group of networked clients connects to a proxy cache server, where each client has a browser cache buffering popular and recently requested data objects. Upon a Web request of a client, the browser first checks if it exists in the local browser cache. If so, the request will be served by its own browser cache. Otherwise the request will be sent to the proxy cache. If the requested data object is not found in the proxy cache, the proxy server will immediately send the request to its cooperative caches, if any, or to an upper level proxy cache, or to the Web server *without considering if it exists in other browsers' caches*. We believe there are three practical reasons for a proxy server to exclude this consideration. First, the browser caches are not shared for software simplicity and user privacy reasons; and the dynamic status in each

cache is unknown to the proxy server. Second, the possibility of a proxy cache miss which is a browser cache hit may have been considered low although no such a study has been found in literature. Finally, a browser cache was initially developed as a small data buffer with a few simple data manipulation operations. Users may not effectively retain the cached data with a high quality of spatial and temporal locality. It is desirable to understand the potential performance gain by sharing data among browsers before making efforts to design and implement browsers-aware proxy servers with enforced or optional security considerations. We have following qualitative arguments for it.

First, since browser caches are not shared among themselves, the size of the proxy cache is limited, and no data consistency is maintained between browser caches and the proxy cache, it is certainly possible that a data object is stored in one or more browser caches, but has been replaced in the proxy cache.

Secondly, browsers provide a function for users to set the browser cache size. With the rapid increase of memory and disk capacity in workstations and PCs, and with the rapid growth of Web applications, user browser

*Corresponding author.

E-mail address: zhang@cs.wm.edu (X. Zhang).

cache size will tend to significantly increase as time passes. In addition, several new software techniques are introduced for users to effectively increase the browser cache size. For example, “browser cache switch” [12] allows users to set multiple browser caches in one machine, and to switch them from one to another during Web browsing. Thus, different caches can be used for different contents and for different time periods. This technique significantly increases the size of a browser cache for an effective management of multiple data types. However, the larger the browser cache size is set, the more spatial locality will be underutilized by the proxy cache server.

Thirdly, in order to help Web users to effectively use and manage large browser cache data, browser software has been upgraded to include several sophisticated database functions, such as file searching, folding, and grouping. With the aid of these functions, users will pay more attention to the organized browser cache data objects, and tend to keep them in the cache much longer than the unorganized data objects. However, the longer the cached data is retained, the more temporal browser cache locality will be underutilized by a proxy cache server.

Fourthly, in order to improve the browsing speed, a large memory drive can be configured to store the entire browser cache. This technique of “browser cache in memory”, has been implemented in several commercial browsers, such as Internet Explore and Netscape. This technique can be further extended to periodically save the cached data objects in a special directory in the disk. The data will be brought back from the disk to the special memory drive whenever the system is restarted or rebooted. Several studies (see e.g. [11,30]) have shown that in modern computer systems, transferring data through a moderate speed network will be significantly faster than obtaining the same amount of data from a local disk through page faults. The high speed memory access is not only beneficial to a local user, but also speeds up data accesses for remote users to share browser caches.

Finally, the number and types of Web servers have increased and will continue to increase dramatically, providing services to a wider and wider range of clients. Thus, the number of unique file objects cached in client browsers has increased and will continue to increase. It is impossible for proxy caches to cover all multi-requested file objects of tremendous types even with a perfect cache replacement strategy, increasing the possibility for browser caches to keep file objects that have been replaced in proxy caches.

In summary, the quality of spatial and temporal locality in browser caches has been and will continue to be improved, inevitably providing a rich and commonly sharable storage among trusted Internet clients or peers. We propose a Web document sharing infrastructure

model called Browsers-Aware Proxy Server. In this design, a proxy server connecting to a group of networked clients maintains an index file of data objects of all clients’ browser caches. If a user request misses in its local browser cache and the proxy cache, the browsers-aware proxy server will search the index file attempting to find it in another client’s browser cache before sending the request to an upper level proxy or the Web server. If such a request hits in a remote client, this client will directly forward the data object to the requesting client; or the proxy server fetches the data object from this client and then forwards it to the requesting client.

Besides utilizing client document resources to improve Web caching performance and scalability, the browsers-aware model has the following unique advantages to facilitate prefetching techniques in Web servers and proxy. First, in a browsers-aware environment, some cached files are stored in the proxy, and many of them are distributed in the browsers of clients. Thus, the proxy has additional space to store prefetched files, minimizing negative prefetching effects to the proxy caching. Second, the browser file index in the proxy enables the proxy to directly push the prefetched files to the browsers where they are not cached, reducing network traffics between proxy and browser. Finally, When prefetching is conducted between a Web server and a proxy, the index in the proxy can be used to ensure that the Web server only delivers the prefetched files that are neither in the proxy nor in its connected clients, eliminating unnecessary file duplications, and reducing the network traffic between Web servers and the proxy.

The paper is organized as follows. Section 2 presents the structure of the browsers-aware proxy server. Section 3 describes the trace-driven simulation environment. Performance evaluation is given in Section 4, where we also estimate overhead involved to implement the browsers-aware model. In Section 5, we show how prefetching techniques are beneficial from the browsers-aware model, and present additional prefetching performance improvement achieved by utilizing the browsers-aware model. We overview related work in Section 6, and conclude the work in Section 7.

2. Browsers-aware proxy server

In this design, the proxy server connecting to a group of networked clients maintains an index file of data objects of all clients’ browser caches. If a user request misses in its local browser cache and the proxy cache, the browsers-aware proxy server will search an index file attempting to find it in a client’s browser cache before sending the request to an upper level server. If such a request hits in a remote client, we propose two

alternative implementations to let the requesting client access the data object. First, the proxy server will inform this client to directly forward the data object to the requesting client. In order to retain user browsers' privacy, the message passing from the source client to the requesting client should be anonymous to each other. The second implementation alternative is to make the proxy server provide the data by loading the data object from the source client and then storing it to the requesting client.

In order to implement the browsers-aware concept in a proxy server, we create a *browser index file* in the proxy server. This index file records a directory of cached file objects in each client machine. Each item of the index file includes the ID number of a client machine, the URL including the full path name of the cached file object, and, if any, a time stamp of the file or the time to live (TTL) provided by the data source. Since the dynamic changes in browser caches are only partially visible to the proxy server (when a file object is sent from the proxy cache to the browser), the browser index file will be updated periodically by each browser cache. Here is another alternative. After a file object is sent from the proxy server to a client's browser cache, its index item is added to the browser index file. Whenever this file object is replaced or deleted from the browser cache, the client sends an invalidation message to the proxy server. After then, the proxy deletes the corresponding index item.

Fig. 1 presents the organization of the browsers-aware proxy server by an example. A group of client machines is connected by a local area network. For a given Web service request with a specific URL in client machine i , the browser cache is first searched attempting to satisfy the request. After the request misses in the browser

cache, client i sends the request to the proxy server, where the proxy cache is searched for the same purpose. After the request misses again in the proxy cache, the browser index file is searched, where the URL is matched in client machine j . The proxy server informs client machine j to forward the cached file object to client i , or fetches the cached object from machine j and then forwards it to client i . Client i hits the file object in the browser cache after the file object is delivered by client j or by the proxy server.

3. Simulation environment

The browsers-aware proxy server and related performance issues are evaluated by trace-driven simulations. The evaluation environment consists of different Web traces, a simulated clustered client machines, and a proxy server having aware or unaware browser caches. We will discuss the selected Web traces and our simulation model in this section.

3.1. Traces

Table 1 lists the Web traces we have used for performance evaluation, where *infinite* cache size is the total size storing all the unique requested documents.

1. NLANR traces: NLANR (National Lab of Applied Network Research) provides sanitized cache access logs for the past 7 days in the public domain [23]. NLANR takes special steps to protect the privacy of those participating in their cache mesh. Client IP addresses are randomized from day to day, but consistent within a single log file. Client IP addresses are very important in our study, so we use traces based on 1 day's log file. NLANR provides about 10 proxies' traces. We have used 1 day's trace of July 14, 2000 from the "uc" proxy, which is denoted as NLANR-uc. The name is uc.sanitized-access.20000714.
2. Boeing traces: The Boeing Company collected anonymized logs from Boeing's Puget Sound perimeter (firewall) proxies by using an anonymizer tool (log2anon) and made these logs available in [3]. For privacy reasons, client IP addresses are not identical between two different days, so we use traces based on 1 day's log file. We have used 1 day's trace on March 4, 1999, and 1 day's trace on March 5, 1999, which are the most recent traces in this site and denoted as Boeing-4 and Boeing-5.
3. BU traces: Boston University collected traces from the similar computing facility and user population in 1995 and 1998, which can be found in [4]. We select the traces in a period of 2 months of the 2 years, which are denoted as BU-95 and BU-98, respectively.

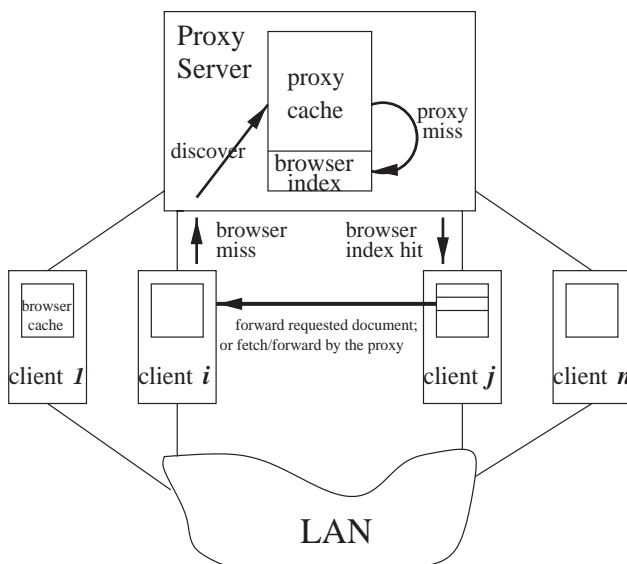


Fig. 1. Organization of the browsers-aware proxy server.

Table 1
Selected web traces

Traces	Period	# Requests	Total GB	<i>Infinite</i> Cache (GB)	# Clients	Max. hit ratio (%)	Max. byte hit ratio (%)
NLANR-uc	7/14/00	360806	4.36	3.72	95	19.11	14.80
Boeing-4	3/4/99	219951	7.54	6.21	3996	44.91	17.69
Boeing-5	3/5/99	184476	7.00	5.50	3659	45.07	21.63
BU-95	Jan.95–Feb.95	502424	1.31	0.90	591	64.14	31.37
BU-98	Apr.98–May 98	72626	0.45	0.29	306	40.62	35.94
CA*netII	9/19–9/20/99	745943	0.089	0.062	3	34.20	29.84

4. CA*netII traces: The CA*netII (Canada's coast to coast broadband research network) parent cache provides sanitized log files in [5]. The client IDs are consistent from day to day, so we concatenate 2 days' logs together as our trace denoted as CA*netII. The two logs we used are the traces collected on September 19, 1999 and September 20, 1999, which are the most recent traces in this site. Their names are access.1999-09-19.gz and access.1999-09-20.gz.

3.2. A browsers-proxy caching environment

We have built a simulator to construct a system with a group of clustered clients connecting to a proxy server. The cache replacement algorithm used in our simulator is LRU. All the traces have the size of a document for each request. If a user request hits on a document whose size has been changed, we count it as a cache miss. We have implemented and compared the following five Web caching organizations using the trace-driven simulations:

1. *Proxy-cache-only*: each client does not have a browser cache. Every client request is sent directly to the proxy cache server.
2. *Local-browser-cache-only*: each client has a private browser cache, but there is no proxy cache server for client machines.
3. *Global-browsers-cache-only*: each client has a browser cache which is globally shared among all the clients by maintaining an index file in each client machine. The index file records a directory of cache documents of all clients. A browser does not cache documents fetched from another browser cache. If a request is a miss in its local browser, the client will check the index file to see if it is stored in other browser caches before sending the request to a Web server. There is no proxy cache server for the group of client machines.
4. *Proxy-and-local-browser*: each client has a private browser cache, and there is a proxy cache server for the group of client machines. If a request misses in its local browser, it will be sent to the proxy to check if the requested document is cached there. If it misses

again, the proxy will send the request to an upper level server.

5. *Browsers-aware-proxy*: this is the enhanced proxy caching technique presented in Section 2.

We have validated our simulator motivated by the method in [8]. We input the NLANR trace to our simulation with an *infinite* proxy cache size. We compared the simulated hit ratios and the actual daily hit ratios reported in the public domain [23]. The reason we use an *infinite* cache size for comparisons is as follows. All the proxies of NLANR allocate about 16 GB of the disk for caching. But, for privacy and protection reasons, we are only able to use 1 day's log file, whose total requested document size is less than 16 GB. Our experiments show that the average hit ratio difference is about 6% for the NLANR trace. In the actual daily statistics of the NLANR trace, some of today's requests hit the documents cached "yesterday". The simulation does not reflect this small number of special hits. This is a major reason for the 6% error. We also validated our simulator for all the traces by comparing the hit ratios and byte hit ratios of above schemes 4 and 5 with infinite proxy cache and browser cache. They all join to the same point.

We use two performance metrics. *Hit ratio* is the ratio between the number of requests that hit in browser caches or in the proxy cache and the total number of requests. *Byte hit ratio* is the ratio between the number of bytes that hit in browser caches or in the proxy cache and the total number of bytes requested.

4. Performance evaluation

Before presenting performance results, we will first look into a browser and proxy cache size related issue to provide a basis and a rationale for us to configure our simulated Web caching system.

4.1. Sizes of browser and proxy caches

Rousskov and Soloviev [27] have studied seven Squid proxies covering several levels of the caching hierarchy from leaf university proxies, to top level proxies for

Table 2
Representative proxy cache configurations presented in [27]

Proxies	Hot memory (MB)	Disk cache (GB)	# clients	Memory cache/client (MB)	Disk cache/client (MB)
<i>ruu</i>	32	5.6	518	0.0618	10.8
<i>uit</i>	32	3.8	378	0.0847	10.1
<i>adfa</i>	32	5.8	798	0.0401	7.3

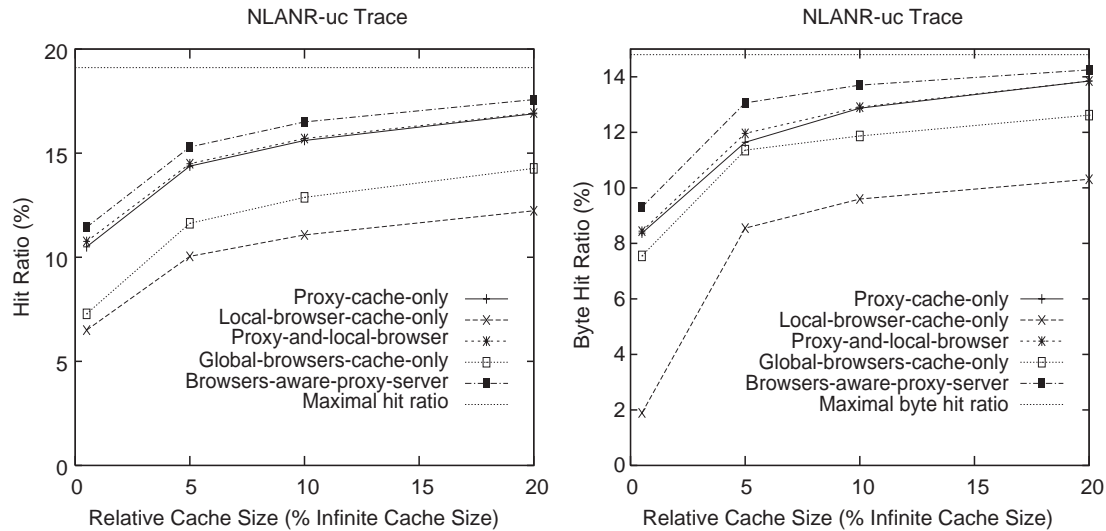


Fig. 2. The hit ratios and byte hit ratios of the five caching policies using NLANR-uc trace, where the browser cache size is set *minimum*.

large country-wide networks, and to the international root proxy located at NLANR. Three of them are leaf proxies which are related to our study: *ruu* from Netherlands, *uit* from Norway, and *adfa* from Australia. Their proxy cache related configurations are listed in the second to fourth columns of Table 2. Squid uses a two level cache. The first level is a small and hot memory in which very popular and recently requested documents are kept. The second level is a disk cache where the majority of documents reside. The second and third columns in Table 2 are the sizes of the hot memory and disk caches. The last two columns are the average proxy cache size in hot memory per client and the average proxy cache size in disk per client. We assume each client's browser has a cache. If we use the average proxy cache size per client in Table 2 as the browser cache size of each client, the memory space ranging from 0.04 to 0.08 MB is certainly too small, and the total cache size ranging from 7.34 to 10.86 MB is also not large enough in practice for today's computer systems. Therefore, in our study we define a *minimum* browser cache size as

$$\text{Min}(\text{Cache}_{\text{browser}}) = \frac{\text{Cache}_{\text{proxy}}}{m}, \quad (4.1)$$

where $\text{Cache}_{\text{browser}}$ is the size of a client browser cache, m is the number of clients, and $\text{Cache}_{\text{proxy}}$ is the size of the

proxy cache responsible for the m clients. We also conservatively define an *average* browser cache size as

$$\text{Average}(\text{Cache}_{\text{browser}}) = \frac{\beta \text{Cache}_{\text{proxy}}}{m}, \quad (4.2)$$

where β is in a range of 2–10. If the accumulated browser cache size increases faster than the increase of the proxy cache size, the value of β tends to increase if both clients and the proxy server are upgraded as time passes.

4.2. How much is browser cache data sharable?

To answer this question, we have operated the five caching policies with different traces on a simulated Web caching environment where the browser cache size of each client is set to *minimum*. Performance results of all the traces we have used are quite consistent. Due to the page limit, we only present the results of hit ratios and byte ratios from the NLANR-uc trace in Fig. 2, where the size of the proxy cache is scaled from 0.5%, 5%, 10%, and to 20% of the *infinite* proxy cache size, the browser cache size is also scaled up accordingly.

Fig. 2 shows that the hit ratios (left) and byte hit ratios (right) of the *browsers-aware-proxy-server* are the highest, particularly, the hit ratios are up to 5.94% higher and the byte hit ratios are 9.34% higher than

those of the *proxy-and-local-browser*, even when the browser cache size is set to *minimum*. This means that sharable data locality does exist, even for a small browser cache size. The sharable data locality proportionally increases as browser cache size increases and as the number of unique file objects cached in browsers increases, both of which are the trends in Web computing. In next subsection, We will show that significant proxy cache performance improvement can be achieved by the proposed browsers-aware proxy server to exploit sharable data locality.

We also show that methods of *proxy-cache-only*, *local-browser-cache-only*, and *global-browsers-cache-only* are not as effective as the method of *proxy-and-local-browser*. *Local-browser-cache-only* had the lowest hit and byte hit ratios due to the minimum caching space. *proxy-and-local-browser* only slightly outperforms *proxy-cache-only*, which implies that performance gain from a local browser cache is limited. Another observation worth mentioning is that *proxy-and-local-browser* and *global-browsers-cache-only* had lower hit and byte hit ratios than *browsers-aware-proxy-server*. This observation confirms the existence two types of misses. First, there exist some documents which are already replaced in the proxy cache but still retained in one or more browser caches, because the request rates to the proxy and to browsers are different, causing the replacement in the proxy and browsers at a different pace. Second, there are some documents which are already replaced in browser caches but still retained in the proxy cache, because a browser cache is much smaller than the proxy cache. The *browsers-aware-proxy-server* effectively addresses these two types of misses.

Fig. 3 presents the breakdowns of the hit ratios and the byte hit ratios of the *browsers-aware-proxy-server* using NLANR-uc trace. There are three types of hits: hits in the local browser cache, hits in the proxy cache,

and hits in remote browser caches. We show that the hit ratio and byte hit ratio in remote browser caches should not be neglected even when the browser cache size is very small.

The *browsers-aware-proxy-server* has another advantage over the *proxy-and-local-browser* policy in terms of “memory” byte hit ratios. In other words, for the same byte hit ratio, a higher percentage of requests will hit in the main memory of browser caches and the proxy cache provided by the *browsers-aware-proxy-server*. (Accesses to main memory have much lower latency than accesses to disks.) To quantitatively justify this claim, we have compared the memory byte hit ratios of the two policies for an equivalent byte hit ratio.

In our simulation, we set the memory cache size in the proxy as 1/150 of the proxy cache size based on the memory ratio reported in [27]. We also set the memory size of a browser cache as 1/150 of the browser cache size, which is not in favor of the *browsers-aware-proxy-server* because the memory cache portion in a browser can be much larger than that for the proxy cache in practice. We also conservatively assume that one memory access of one cache block of 16 Bytes spends 200 ns (the memory access time is lower than this in many advanced workstations), and one disk access of one page of 4 kB is 10 ms.

Fig. 2 shows that the hit and byte hit ratios of the *browsers-aware-proxy-server* at 5% of the *infinite* cache size are very close to those of the *proxy-and-local-browser* policy at 10% of the *infinite* cache size (the hit ratio comparison is 15.3 vs. 15.7, and byte hit ratio comparison is 13.06 vs. 12.91). However, the memory byte hit ratios of the two schemes are quite different under the same condition, which are 3.5% for the *browsers-aware-proxy-server*, and 1.9% for the *proxy-and-local-browser* policy, respectively. The larger memory byte hit ratio of the *browsers-aware-proxy-server* in

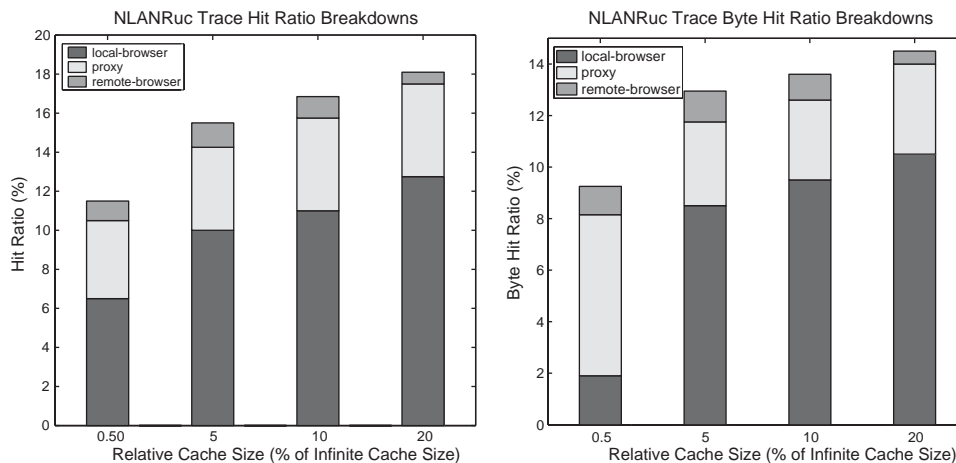


Fig. 3. The breakdowns of the hit ratios and byte hit ratios of the browsers-aware proxy using NLANR-uc trace, where the browser cache size is set *minimum*.

this case would reduce 15.2% of the total hit latency compared with the *proxy-and-local-browser*. The latency reduction due to the higher percentage memory accesses will be larger in practice because the memory cache size of each browser is much larger than the assumed size.

4.3. Performance of browsers-aware proxy

We have evaluated and compared the performance of the *browsers-aware-proxy-server* and *proxy-and-local-browser* schemes using the two Boeing traces and two BU traces. For experiments of each trace, the proxy cache size is set to 0.5%, 5%, 10%, and 20% of the *infinite* proxy cache size. Accordingly, each browser cache is also set to 0.5%, 5%, 10%, and 20% of the

average *infinite* browser cache size calculated from all the browsers. The *infinite* cache size of a browser is the total size of all uniquely requested documents in this client. For example, if the proxy cache is set to 0.5% of the *infinite* proxy cache, all browsers' caches will also be set to 0.5% of the average size of the *infinite* browser size of all browsers. The value of β calculated from each trace falls into the *average* range of 2–10. The value of β calculated from each Boeing trace is slightly larger than the *average* range. This is reasonable because one proxy serves a huge number of clients for the Boeing traces.

Figs. 4–7 present the hit ratios (left) and byte hit ratios (right) of the two policies on Boeing-4 trace, Boeing-5 trace, BU-95 trace, and the BU-98 trace, respectively. Compared with the *proxy-and-local-browser*

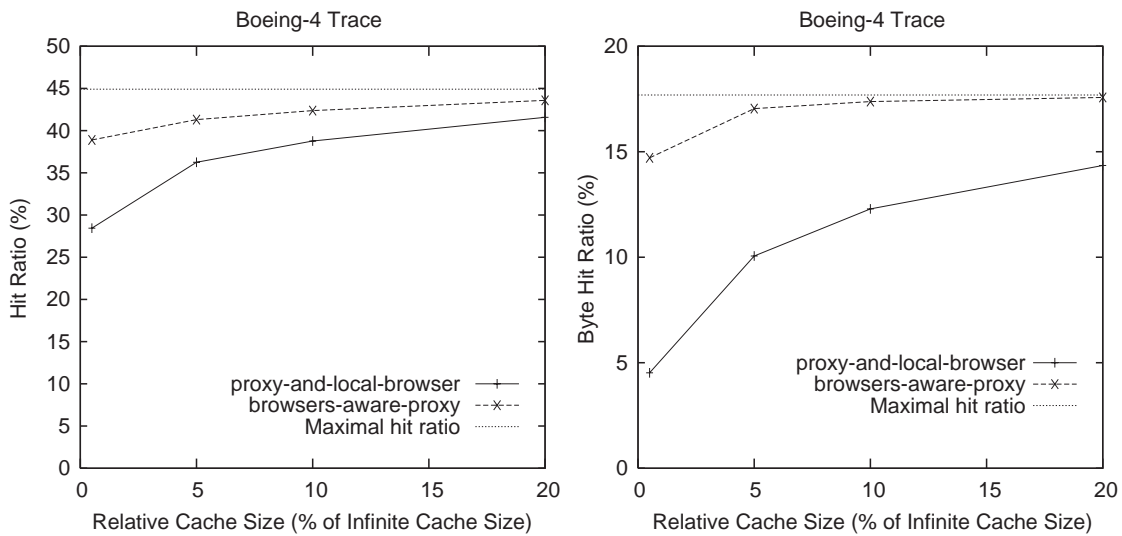


Fig. 4. The hit rates and byte hit rates of the *browsers-aware-proxy-server* and *proxy-and-local-browser* scheme using Boeing-4 trace, where the browser cache size is set *average*.

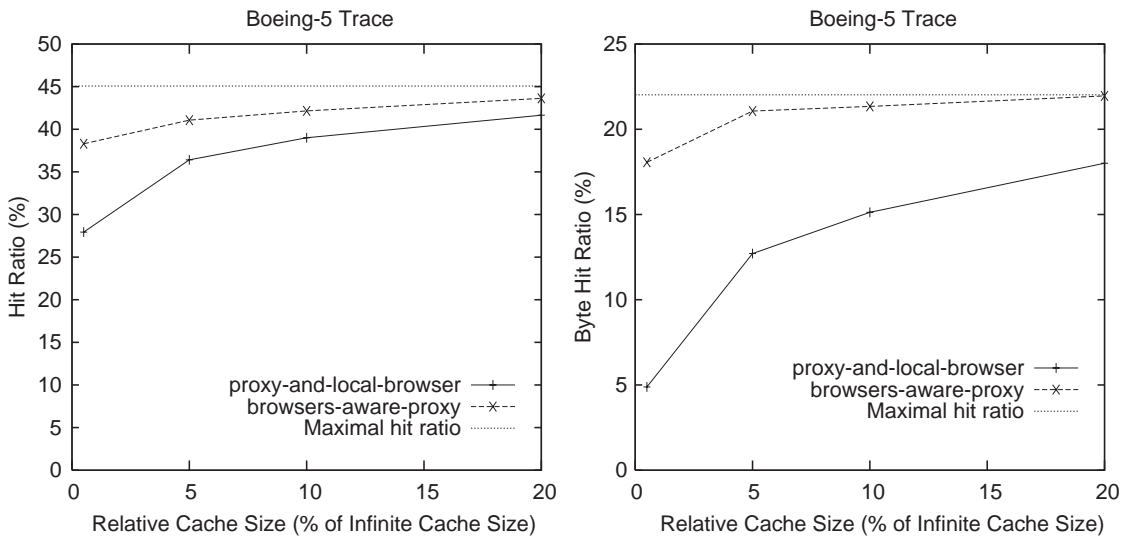


Fig. 5. The hit rates and byte hit rates of the *browsers-aware-proxy-server* and *proxy-and-local-browser* scheme using Boeing-5 trace, where the browser cache size is set *average*.

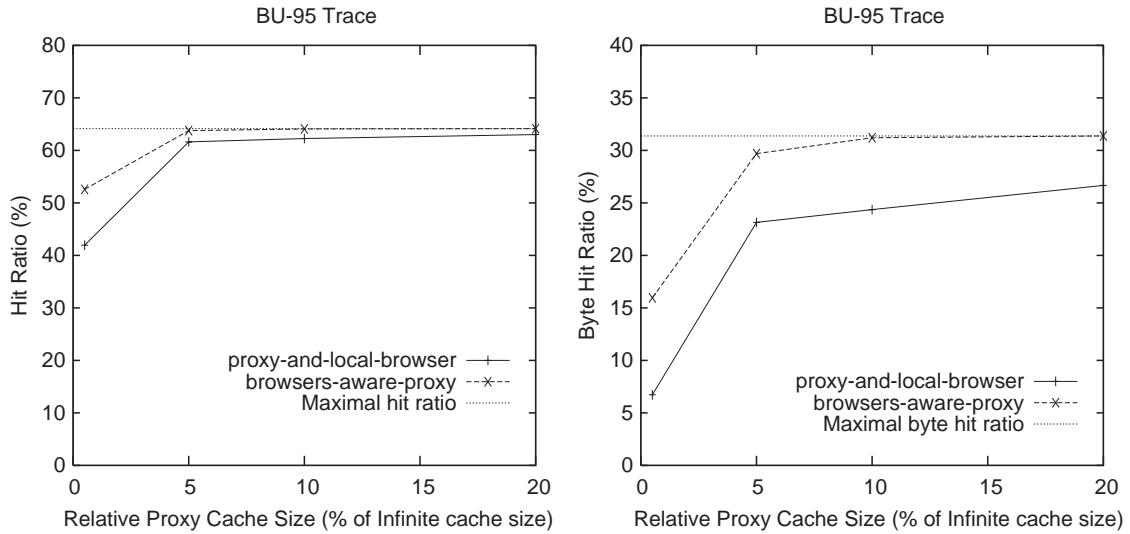


Fig. 6. The hit rates and byte hit rates of the *browsers-aware-proxy-server* and the *proxy-and-local-browser* scheme using the BU-95 trace, where the browser cache size is set *average*.

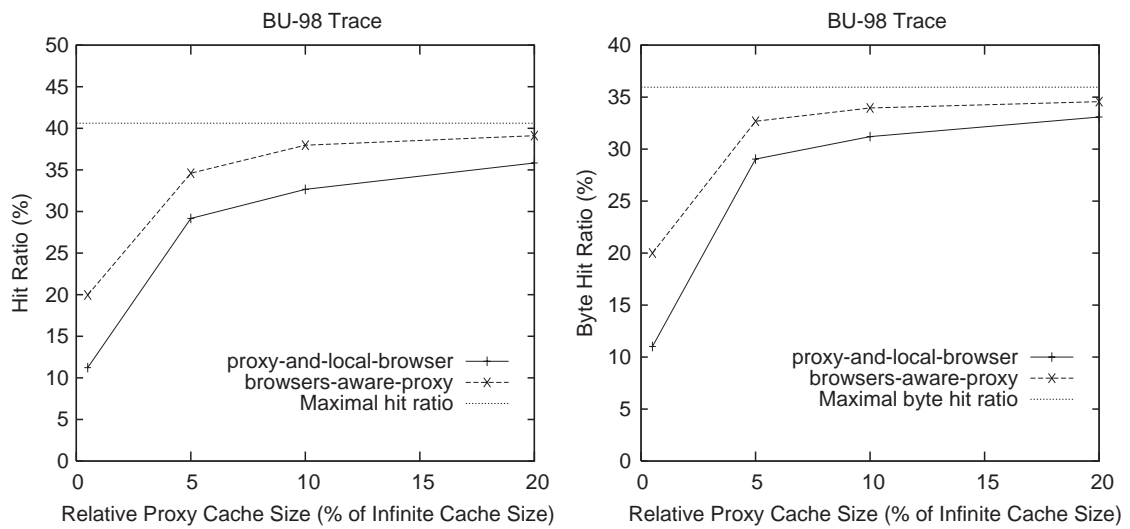


Fig. 7. The hit rates and byte hit rates of the *browsers-aware-proxy-server* and the *proxy-and-local-browser* scheme using the BU-98 trace, where the browser cache size is set *average*.

scheme, *browsers-aware-proxy-server* consistently and significantly increases both hit ratios and byte hit ratios on all the traces.

When the number of clients is small, and their accumulated size of the browser caches is much smaller or not comparable to the proxy cache size, the cache locality inherent in browsers is low, so the performance gain from the browsers-aware proxy cache will also be insignificant. Fig. 8 presents such an example, where the total number of clients of the CA*netII trace is only 3, the accumulated size of three browser caches is small. The increases of both average hit ratio and byte hit ratio of this trace by the *browsers-aware-proxy-cache* are below 1%, compared with the *proxy-and-local-browser* scheme.

4.4. Performance impact of scaling the number of clients

We have also evaluated the effects of scaling the number of clients to browsers-aware proxy servers. For each trace, we observe its hit ratio (or byte hit ratio) increment changes by increasing the number of clients from 25% to 50%, 75%, and 100% of the total number of clients. We also call each percentage as a relative number of clients. For all relative numbers of clients of each trace, the proxy cache size is fixed to 10% of the *infinite* proxy cache size when the relative number of clients is 100%. The byte hit ratio increment or the hit ratio increment of the browsers-aware proxy server for a

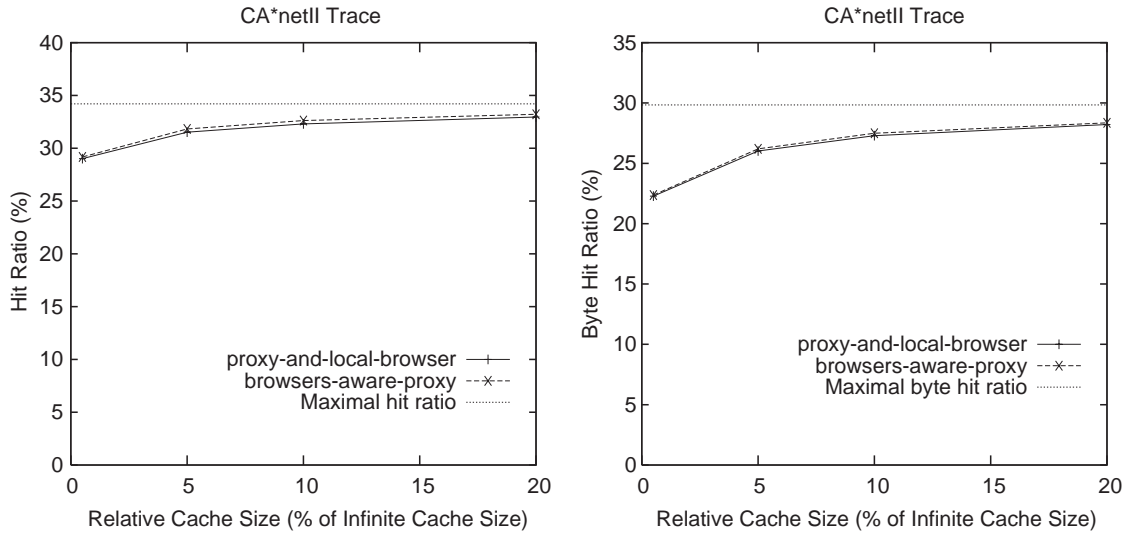


Fig. 8. The hit ratios and byte hit ratios of the *browsers-aware-proxy-server* and *proxy-and-local-browser* scheme using the CA*netII trace.

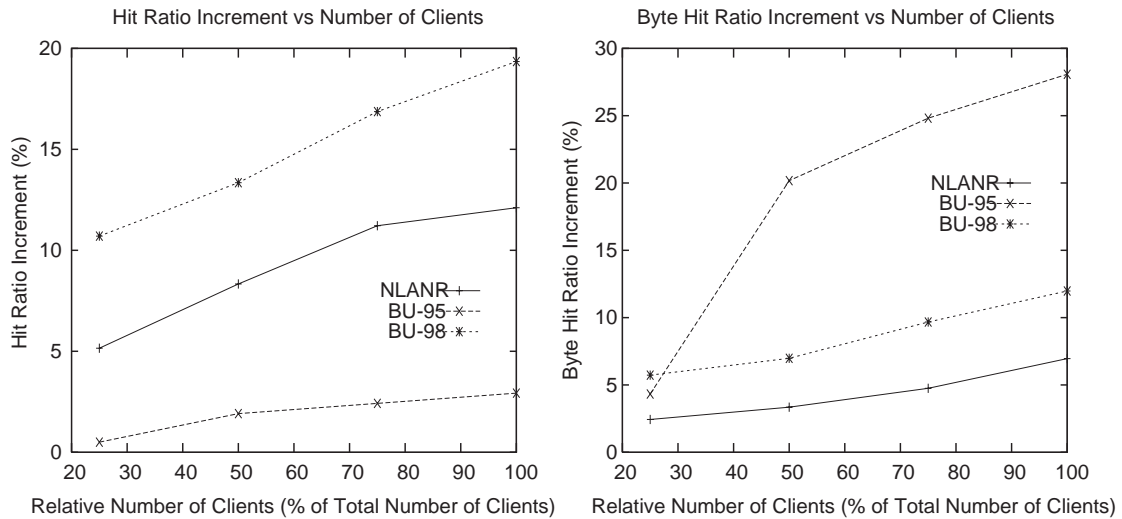


Fig. 9. The hit ratio and byte hit ratio increments of the *browsers-aware-proxy-server* over the *proxy-and-local-browser*.

given trace is defined as

$$\frac{(\text{byte}) \text{ hit ratio of browse_aware} - (\text{byte}) \text{ hit ratio of proxy_and_local_browser}}{(\text{byte}) \text{ hit ratio of proxy_and_local_browser}}$$

Fig. 9 presents the hit ratio increment curves (left figure) and the byte hit ratio increment curves (right figure) of the three traces as the relative number of clients changes from 25% to 100%. Our trace-driven simulation results show that both hit ratio increment and byte hit ratio increment of the browsers-aware proxy server proportionally increases as the number of clients increases. For some traces, the increments are significant. For example,

the hit ratio increment of BU-98 trace increases from 10.70% to 13.35%, 16.87%, and 19.35%, as the relative number of clients increases from 25% to 50%, 75%, and 100%, respectively. The byte hit ratio increment of BU-95 trace increases from 4.33% to 20.17%, 24.82%, and 28.08%.

The performance results indicate that a browsers-aware proxy server is performance beneficial to client

cluster scalability because it exploits more browser locality and utilizes more memory space as the number of clients increases.

4.5. Overhead estimation

The additional overhead of the browsers-aware proxy cache comes from the data transferring time for the hits in remote browsers. The simulator estimates the data transferring time based on the number of remote browser hits and their data sizes on a 10 Mbps Ethernet. The browser access contention is handled as follows. If multiple requests hit documents in a remote browser simultaneously, the bus will transfer the hit documents one by one in the FIFO order distinguished by each request's arrival time. Our experiments using the *ping* facility show that the startup time of data communications among the clients in our local area university network is less than 0.01 second. Setting 0.01 second as the network connection time, we show that the amounts of data transferring time and the bus contention time spent for communication among browser caches of the browsers-aware proxy server on all the traces is very low. For example, the largest accumulated communication and network contention portion out of the total workload service time for all the traces is less than 1.25%. In addition, the contention time only contributes up to 0.12% of the total communication time, which implies that browsers-aware proxy server does not cause bursty hits to remote browser caches.

Another potential overhead is the update of the browser index file if the update is not conducted at a suitable time or conducted too frequently. There have been some solutions to address this concern. For example, the browser could send its update information when the path between the browser and the proxy is free to avoid contention. The study in [9] shows that the update of URL indices among cooperative caches can be delayed until a fixed percentage of cached documents are new. The delay threshold of 1–10% (which corresponds to an update frequency of roughly every 5 min to an hour in their experiments) results in a tolerable degradation of the cache hit ratios. In their experiments, the degradation is between 0.02% and 1.7% for the 1% choice. Our concerns should be less serious because the updates are only conducted between browsers and the proxy without broadcasting. But in their case, each proxy need broadcast its updates to all the others. Thus, the index file update overhead between browsers and proxy is very low.

The last potential concern is the space requirement of the proxy cache to store the browser index. We address this concern by an example. Each URL is represented by a 16-byte MD5 signature [14,21]. Assume there are 1000 clients connected to one proxy. Each client has a browser with a 8 MB cache. We assume that an average

document size is 8 kB. Each browser has about 1000 Web pages. The proxy server only needs about $1000 \times (8 \text{ MB}/8 \text{ kB}) \times 16 = 16 \text{ MB}$ to store the whole browser index file for the 1000 browsers. If we apply the compression methods presented in [9,22], the browsers-aware proxy server requires even less space to store the index file (e.g. a storage of 2 MB is sufficient for the 1000 browsers with a tolerant inaccuracy).

We can also implement a Bloom filter that is used to keep URL indices of cooperative caches in [9]. Assume that there are 1000 clients connected to one proxy. Each client has a browser with an 8 MB cache. Similar to [9], we also assume that an average document size is 8 kB. Each browser has about 1000 Web pages. The Bloom filter need 2 kB to represent 1000 pages of each browser. The Proxy needs about 2 MB to store the whole browser index file.

5. Browsers-aware model supported prefetching

5.1. Prefetching methods in a standard proxy caching environment

Web prefetching has been proposed to further reduce the client-perceived latencies, which preloads the Web data a client may request in the near future based on the client's past surfing activity. The performance improvement of Web prefetching combined with caching can be doubled compared with caching alone (see e.g. [18]). Based on the source of the access history used for predictions, we can divide prefetching methods into three categories.

- *Client-initiated prefetching*: The simplest way to do prefetching is based on the client's own access history. Due to its potential performance benefits without a requirement of changing Web servers, client-based prefetching has gained a lot of attention [17]. Due to the limited information observed by an individual client, the prediction accuracy is low and overhead is relatively high, which prevents it from being widely deployed.
- *Proxy-initiated prefetching*: Compared with the browser, a proxy can observe the accesses from multiple users, which may make predictions more efficiently [10]. The prefetching is conducted between the proxy and the clients. The most attractive property of proxy-initiated prefetching is that it does not increase the external network traffic between the proxy and the server. However, the interests of the clients connected with one proxy are diverse. Although the proxy can effectively make correct predictions for the popular servers to all clients, a large number of requests are to unpopular servers, thus the proxy's average prediction abilities can be low.

- *Server-initiated prefetching*: In a sever-initiated prefetching, a Web server always pushes their popular documents to the clients. Compared with the client and the proxy, the Web server can observe much more accesses even though advanced Web caching technology has been well developed, which makes server-initiated prefetching scheme be a perspective solution.

Here is a typical procedure of a server-initiated prefetching in a standard proxy caching environment. Fig. 10 depicts the interactions between a client and a server starting with a request issued by the client.

1. A client sends a message to the proxy to indicate its current request status after either a browser hit or miss.
2. The proxy forwards the client message to a server. If the incoming message is a request (a miss in the browser cache), the proxy will make a local search. If the requested object is found, the proxy sends it to the client.
3. The server processes the incoming message and makes prefetching predictions.
4. The server sends back a set of prediction results to the proxy (piggybacked with the requested object if it is a miss in the proxy cache).
5. The proxy forwards server’s prediction results and/or the requested object to the client.
6. The client checks if the predicted objects have been cached or not after then.

7. The client sends its requests to the proxy based on its local checking for predicted objects.
8. The proxy forwards the requests to the server if the requested objects are not in its cache.
9. The server sends back the requested objects to the proxy.
10. The proxy forwards the objects to the client and keep a copy for each cacheable object.

In this scheme, the prefetching activities are transparent to the proxy, which only passively forwards the messages between the client and the server and caches every objects received from the server.

5.2. Prefetching overheads

The major concern on the deployment of Web prefetching is the related overheads, which may offset the potential benefits if they are not sufficiently reduced. The overheads come from the following sources.

- *Additional storage*: Compared with Web caching, Web prefetching needs more storage in client/proxy to keep the preloaded Web objects for future use. It may frequently trigger cache replacement to accommodate those objects, which can reduce the cache hit rate. Another storage overhead is the memory requirement for prediction data structure. It can be largely reduced by effectively restructuring the data structures of storing history information [7,25].

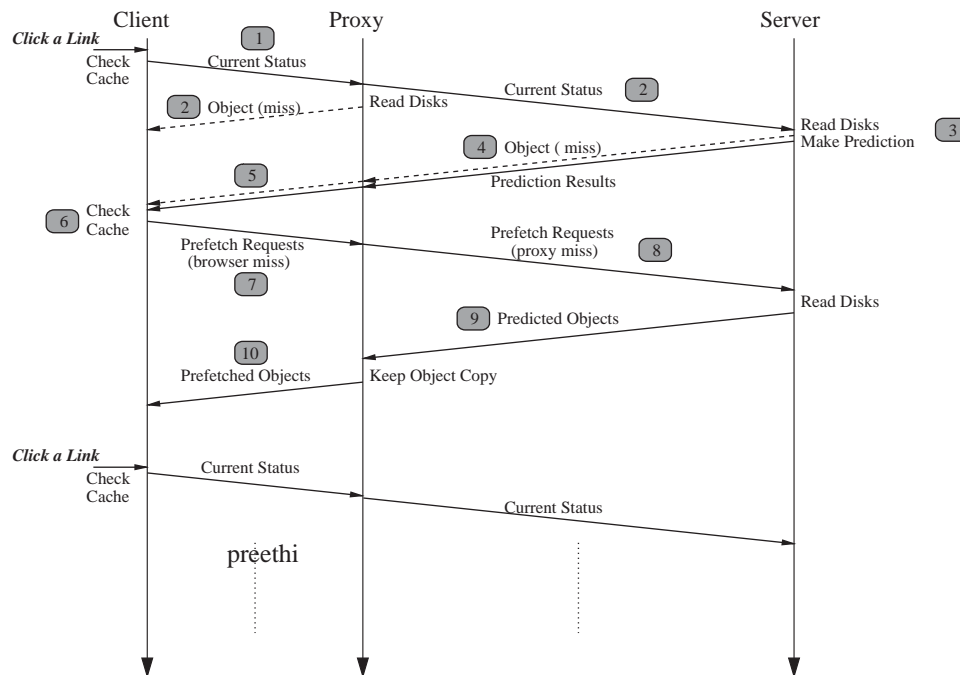


Fig. 10. The procedure of server-initiated prefetching scheme with a proxy between clients and servers.

- *Additional network traffic:* In most prefetching schemes, Web prefetching has higher network bandwidth demands because predictions are made based on history statistics and they are not always correct. In order to avoid wasting network resources when wrong predictions are made, in most prefetching schemes, a maximal size of the prefetched files as the prefetching threshold is defined. However, the network overhead is still significant when prefetching is applied aggressively.
- *Additional CPU cycles:* The main requirement of additional CPU cycles is on Web servers. More requests are received by Web servers which need more CPU cycles. Although the current CPU is powerful to quickly handle HTTP requests, the CPU overhead is still a major problems for two reasons: (1) The request rate is bursty. It is not uncommon that the request arrival rate during peak period can be 5–10 times higher than the average. (2) As dynamic content generation techniques are widely used, increasingly more CPU cycles will be consumed. Prefetching is also proposed to pre-compute the dynamical contents, which may have significant effects on the quality of existing Web services.

5.3. Reducing overheads by using the browsers-aware proxy model

Browsers-aware model can effectively reduce the prefetching overheads described in the previous subsection for the following reasons:

- The browsers-aware model provides more storage space.
The benefits come from two parts: (1) The index avoids unnecessary duplications between the proxy and clients. (2) Proxy cache is only used to store the commonly shared objects while a browser cache is used to store both shared and non-shared objects.
- The browsers-aware model avoids unnecessary data transmissions.
Traditionally, Web prefetching relies on the clients to decide which objects to be prefetched on the basis on their local browser caches. An individual client does not know the contents of other clients connected to the same proxy, which may result in transferring redundant data cached by other clients. Browsers-aware model can effectively avoid the unnecessary network traffic since the proxy records the information of objects in browser caches. The model also simplifies the prefetching procedure by using proxy to make decisions on behalf of clients.
- The prefetching aggressiveness can be adjusted by proxies.
Prefetching aggressiveness is measured by the amount of objects to be preloaded per client request. After each

prediction, the server sends a list of predicted objects to the proxy for consideration. In a normal proxy caching environment, the proxy knows little about the activeness status of each object, and accepts all the preloaded objects if space is available. Since the proxy space is limited, the replacement activity can be frequent to store preloaded objects. With the support of the browsers-aware model, the proxy is fully aware of the current locations of each cached object and its activeness status based on its popularity. The proxy can accept more predicted objects related to popular files, but less ones related to the files with low popularity. With this selection ability, the prefetching aggressiveness can be effectively adjusted by proxies. Since the prefetching accuracy is improved, the prefetching server load will be reduced accordingly.

5.4. Browsers-aware proxy supported prefetching scheme

In order to reduce the Web prefetching related overheads, we propose a browsers-aware proxy supported prefetching scheme, which utilizes the index structure in the proxy. We also place an additional field, called popularity, for each index entry to record the access frequency of correspondent objects stored in browser/proxy cache.

Fig. 11 shows the interactions among the client, the proxy and the server in the new prefetching scheme supported by browsers-aware proxy. In this scheme, the proxy actively participates the prefetching procedure.

1. A client sends a message to the proxy to indicate its current request status after either a browser hit or miss.
2. The proxy forwards the client message to a server. If the incoming message is a request (a miss in the browser cache), the browsers-aware proxy will handle this in its own way (described in Section 2). If the requested object is found either locally or in another browser, it will be delivered to the client. The field popularity of the object is incremented by one.
3. The server processes the incoming message and makes prefetching predictions.
4. The server sends back a set of prediction results to the proxy (piggybacked with the requested object if it is a miss in the proxy cache).
5. The proxy checks if the predicted objects have been cached or not in both the proxy and its connected clients using the index. Then it sends a list of demanded prefetched objects to the server, which are not cached in neither proxy nor in clients. In addition, the selection can also be influenced by the value of popularity.
6. The proxy sends the request for predicted objects to the server.

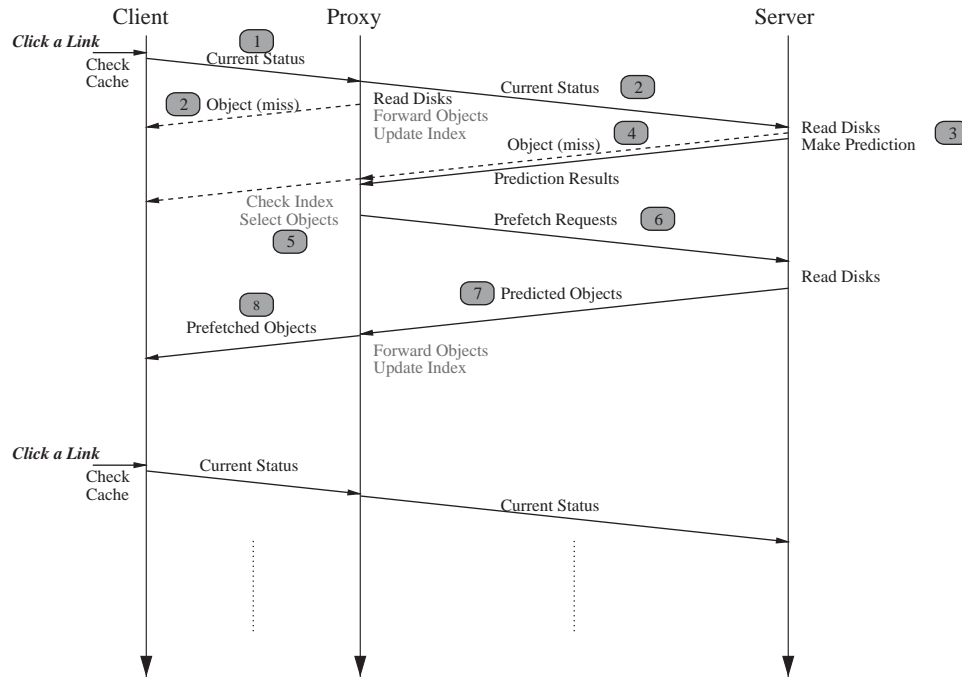


Fig. 11. The procedure of browsers-aware proxy supported prefetching scheme.

- The server sends back the requested objects to the proxy.
- The proxy forwards the objects to the client and updates the index.

There are three major differences compared with the scheme without a support from the browsers-aware proxy model:

- The proxy is mainly used to store the objects shared by multiple clients. It does not cache those objects with cookies or queries which are delivered to a specific client.
- With the support of the browsers-aware model, the proxy can select objects to prefetch on behalf of clients. Specifically, steps 5 and 7 in the original procedure are eliminated. It has two advantages: (1) network traffic is reduced; and (2) a practical deployment is straightforward. Since every object in the proxy and all clients' caches are known by the proxy, it effectively avoids the unnecessary communications for sending a list of prediction results (from the proxy to a browser) and for sending a selection decision (from the browser to the proxy). This reduces the traffic between proxy and the server because the proxy only requests the prefetched objects that are not cached in the proxy and the clients. In addition, the interactions between clients and the proxy are largely reduced.
- The proxy can be used to control aggressiveness of Web prefetching with the new field popularity entry in the index.

5.5. Performance evaluation

We also use the same set of proxy traces presented in Section 3 to evaluate the performance of our browsers-aware proxy supported prefetching scheme in our trace-driven simulations. Limited by the availability of correspondent Web server traces, we cannot measure the prefetching abilities for each individual server in the proxy traces, which is measured by prefetching accuracy and hit ratio. Based on the analysis of prefetching in real-world server traces, we set the following parameters in our experiments:

- A —*prefetching accuracy*, defined as the possibility that the predicted objects are used. Upon each client request, a number of objects are predicted, which may be accessed by the next request of the client. Specifically, the prefetching accuracy is equal to the ratio between the number of predicted objects being accessed by the next request and the total number of predicted objects for the next request.
- P —*prefetching hit ratio*, defined as the possibility that a request can be predicted. Not all requests can be predicted due to the limit of the number of predicted objects for usage and shortage of the access history. The prefetching hit ratio is equal to the number of requests accessing the objects predicted by their previous request and the total number of requests.

The number of prefetched objects for every request is decided by the accuracy and hit ratio. For example, if we

have $P = 0.8$ and $A = 0.2$, on average, 4 objects will be predicted for every request. However, the predicted objects may not be transferred from Web servers. If the requested URL has been cached, we assume the related predicted objects are also cached. If not, the predicted objects have certain probabilities being cached, which are equal to the cache hit ratio without prefetching for the same size cache.

We define *byte effectiveness* as the ratio between the prefetching hit ratio and the prefetching traffic increment. Three performance metrics are used in our performance evaluation:

- *Relative hit ratio*: the ratio of hit ratios between the browsers-aware proxy supported prefetching and the normal prefetching.
- *Relative traffic increment*: the ratio of traffic increment between the browsers-aware proxy supported prefetching and the normal prefetching.
- *Relative byte effectiveness*: the ratio of byte effectiveness between the browsers-aware proxy supported prefetching and the normal prefetching.

In all experiments, we set the browser cache size as 5 times of $\frac{Cache_{proxy}}{m}$, where $Cache_{proxy}$ is the size of the proxy

cache, and m is the number of clients. The replacement policy is LRU.

5.5.1. Space effects on hit ratios

We divide the hit ratios into two categories: cache hits which are defined as the hit ratios achieved by passively caching and prefetch hits which are defined as the hit ratios achieved by prefetching. Since more objects are downloaded to browser/proxy caches, the replacement operations can be more frequent. A limited cache size has effects on both cache hits and prefetch hits. In this part, we study the performance effects on cache hits of browsers-aware cache supported prefetching when the storage space is limited.

The results for different traces under different parameters are shown in Fig. 12. Four sets of parameters of P and A are used, which are set as $P = 0.8$ or 0.2 and $A = 0.8$ or 0.2 . The four experiments evaluate the performance for the prefetching scheme with different aggressiveness controls. In the first experiment, by setting $P = 0.2$ and $A = 0.8$, on average, there is one prefetched object every 4 requests. In the second and third experiments, by setting P and A equally, one object is predicted for every request. In the fourth experiment, by setting $P = 0.8$ and $A = 0.2$, every

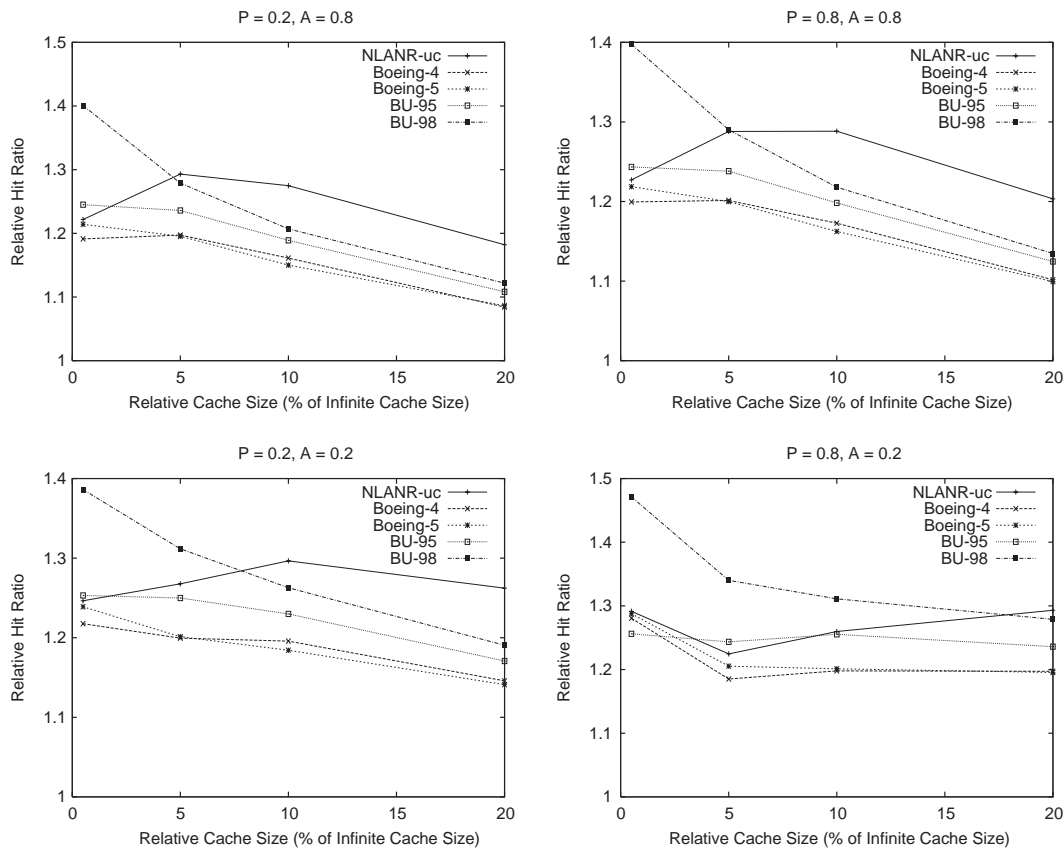


Fig. 12. Comparisons of hit ratios between browsers-aware proxy supported prefetching and normal proxy cache prefetching for different parameters.

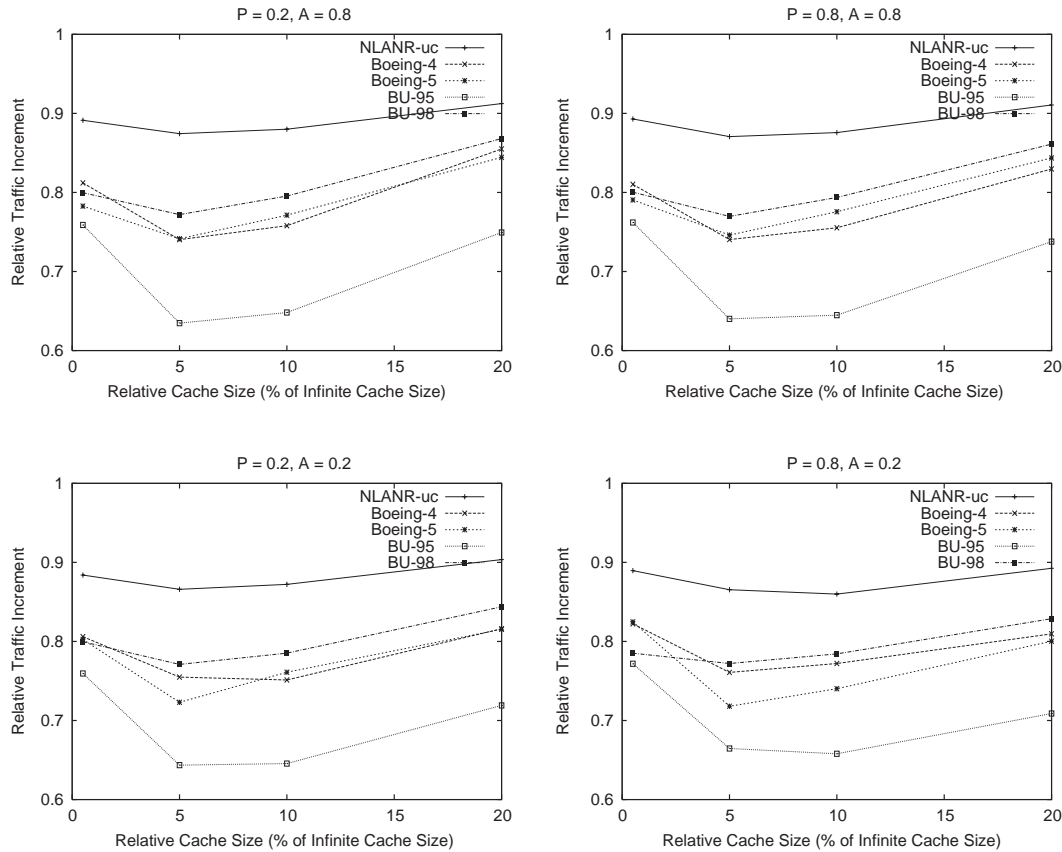


Fig. 13. Comparisons of traffic increment between browsers-aware proxy supported prefetching and normal proxy cache prefetching for different parameters.

request will have 4 candidate objects for prefetching. The browsers-aware proxy supported prefetching can improve the cache hit ratio up to 40%. Compared with the performance improvement when no prefetching is used, it is more effective because of higher space requirement of Web prefetching. Similar to the situation without prefetching, with the increase of proxy cache size, the differences of Web prefetching between a normal proxy and browsers-aware proxy cache are decreased. For all parameters, which represent different aggressive prefetching schemes, the browsers-aware proxy supported prefetching has higher cache hit ratios than normal proxy prefetching due to more effective cache space utilization. With an increase of prefetching aggressiveness, the relative hit ratio of browsers-aware proxy supported prefetching is also increased.

5.5.2. Traffic savings

With the deployment of Web prefetching, more network traffic is required to transfer two kinds of Web objects when the cache size is limited: (1) predicted Web objects, and (2) cached Web objects but replaced by prefetched objects. In order to account for two kinds of traffic increment, we compute the traffic increment in the following way:

- For the objects visited before by the same client, we first check the local browser cache and proxy cache. If it has been replaced, it should be transferred again.
- For the predicted objects, it is possible they have been cached and the possibility is equal to cache hit ratio without prefetching for the same size of browser/proxy cache.

The results for different traces are shown in Fig. 13. We use the same parameters in the previous set of experiments to evaluate the traffic increment when different aggressive prefetching schemes are used. For the traces in our test, browsers-aware proxy supported prefetching can reduce up to 35% network traffic compared with normal prefetching. We observe for most traces, the largest traffic saving is achieved when the proxy cache size is 5% or 10% of infinity cache size. The reason is that when the proxy cache size is small, the normal proxy cache hit ratio is increased in a faster pace than that in the browsers-aware proxy.

5.5.3. Effects on prefetching aggressiveness control

In this part, we evaluate the effectiveness of prefetching aggressiveness controlled by a browsers-aware proxy cache. We assume that the prefetching hit ratio is proportionally increased with the number of objects

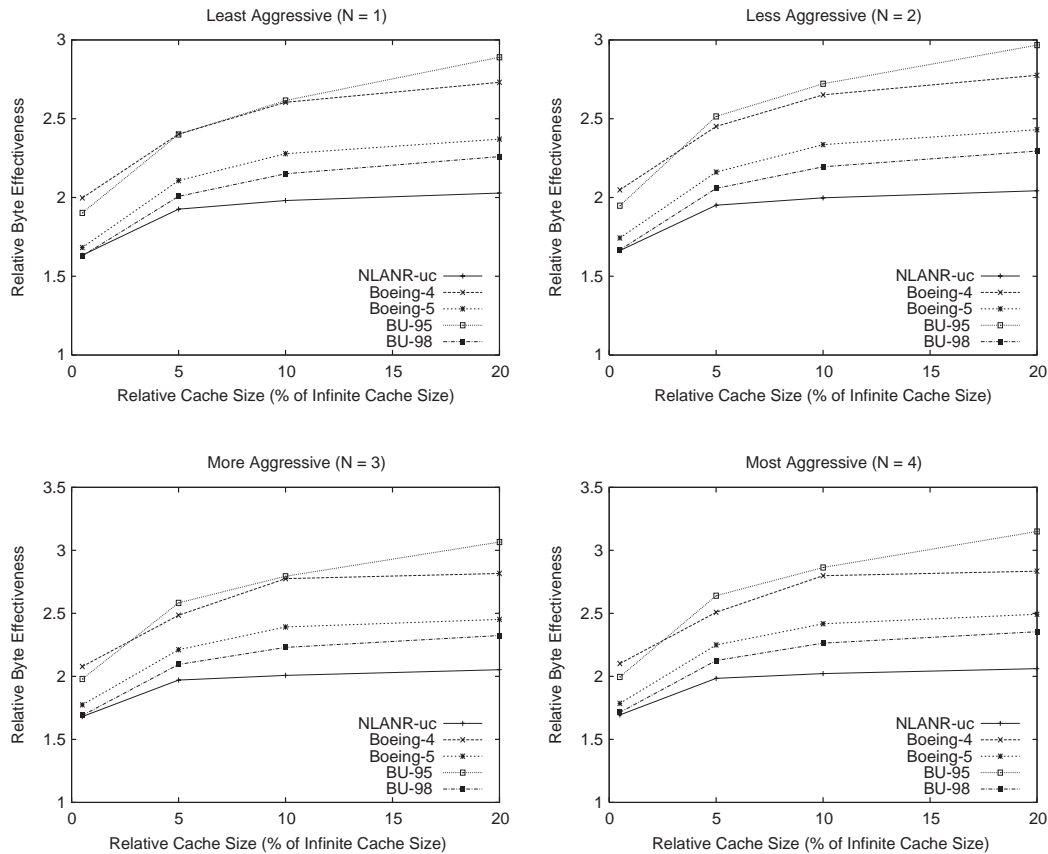


Fig. 14. Comparisons of prefetch effectiveness between fixed aggressiveness prefetching and variable aggressiveness prefetching of browsers-aware proxy supported prefetching.

prefetched. In our experiments, for every request, the number of prefetched objects is from 1 to 4, and the corresponding prefetching hit ratio is from 20% to 80%. The aggressiveness of prefetching can be represented by N , the number of prefetched objects for each request. In order to evaluate the effects of prefetching aggressiveness control, a variable aggressiveness prefetching scheme is designed to adjust the prefetching aggressiveness by the popularities of the objects. In this scheme, we select 10% most popular URLs for aggressive prefetching, in which 4 objects will be prefetched, and the rest 90% URLs for less aggressive prefetching which only prefetches 1 object for each request. If a URL is cached in the browser/proxy cache, we assume that the related prefetched objects are still in the cache. We compare the prefetching scheme with other fixed aggressiveness schemes with a constant number of prefetched objects from 1 to 4. The results are shown in Fig. 14.

For all the traces in our tests, the variable aggressiveness prefetching scheme has much higher byte effectiveness compared with that with the fixed aggressiveness. For example, in NLANR-uc trace, compared with the less aggressive prefetching scheme with $N = 1$, the variable aggressiveness prefetching scheme achieves 2 times higher in byte effectiveness. Similar results are presented in

Fig. 14 compared with different aggressive prefetching schemes. With the increase of proxy cache size, the performance improvement is also increased due to more popular objects and related predicted objects to be cached.

6. Related work

Much work has been done on Web caching at different levels. The caching issues on the server side are representatively discussed in [2,24]. Ref. [13] studies Web content sharing in a multi-level Internet storage hierarchy without considering client browsers. Designs and implementations of browsers have been studied in many papers (e.g. [19,26]). The work in [31] attempts to transfer server's functions to clients. The results of above cited work are supportive to our work based on increasingly powerful browsers.

Client access patterns are characterized by several research groups (see e.g. [8,15,29]). Ref. [1] gives a comprehensive study on the changes in Web client access patterns based on the traces collected from the same computing facility and the same nature of the user population separated by 3 years. Their experiments show that the hit ratios are significantly reduced

compared with the data 3 years ago. One reason for this, we believe, is that the access variations have increased as more Web servers are emerging. Thus, it will be more difficult to retain an optimal hit ratio by increasing proxy cache size and by improving cache replacement strategy. The browsers-aware proxy server has the potential to harvest the sharable data to adapt the trend of the increasing access variations.

Cooperative proxy caches are discussed in Refs. [9,13,20,28], which focus on the proxies at the same level. We have indirectly shown that overall performance will be further improved if each proxy in a cooperative system utilizes sharable data from all its clients.

Advanced proxy caching techniques, such as advanced replacement policies (e.g. [6,16]), can be beneficial to the browsers-aware proxy when both browsers and the proxy use the same policy. We have confirmed this by testing different replacement policies, including the GreedyDual-Size algorithm [6].

7. Conclusion

We have proposed and evaluated a browsers-aware proxy server to provide Web document sharing service. We have also quantitatively answered two questions: (1) how much browser data is sharable? and (2) how much proxy caching and prefetching performance improvement can we gain by the browsers-aware model? Our study shows that the amount of sharable data is significant, and can be utilized to improve the caching performance by the proposed browsers-aware structure. In addition, we show that the browsers-aware model has several advantages to effectively support Web prefetching activities. We are currently implementing a browsers-aware proxy server along with the prefetching schemes.

Acknowledgments

We are grateful to the anonymous referees for their constructive comments and critiques that help us to improve the quality of the paper. We thank Bill Bynum for reading the paper and for his constructive comments. This work is also a part of an independent research project sponsored by the National Science Foundation for program directors and visiting scientists.

References

- [1] P. Barford, A. Bestavros, A. Bradley, M. Crovella, Changes in Web client access patterns: characteristics and caching implications, *World Wide Web J.* 2 (1) (January 1999) 15–28.
- [2] R. Bianchini, E.V. Carrera, Analytical and experimental evaluation of cluster-based network servers, *World Wide Web J.* 3 (4) (December 2000).
- [3] Boeing log files, <ftp://researchsmp2.cc.vt.edu/pub/boeing/>
- [4] BU traces, <ftp://cs-ftp.bu.edu/techreports/1995-010-www.client-traces.tar.gz>; <ftp://cs-ftp.bu.edu/techreports/1999-011-usertrace-98.gz>.
- [5] Canada's coast to coast broadband research network: <http://ardnoc41.canet2.net/>; Sanitized log files: <http://ardnoc41.canet2.net/cache/squid/rawlogs/>
- [6] P. Cao, S. Irani, Cost-aware WWW proxy caching algorithms, *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, Monterey, CA, December 1997, pp. 193–206.
- [7] X. Chen, X. Zhang, A popularity-based prediction model for Web prefetching, *IEEE Comput.* (March 2003) 63–70.
- [8] B.M. Duska, D. Marwood, M.J. Feeley, The measured access characteristics of World-Wide-Web client proxy caches, *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, Monterey, CA, December, 1997, pp. 23–36.
- [9] L. Fan, P. Cao, J. Almeida, A.Z. Broder, Summary cache: a scalable wide-area Web cache sharing protocol, *Proceedings of 1998 SIGCOMM Conference*, Vancouver, Canada, pp. 254–265.
- [10] L. Fan, P. Cao, W. Lin, Q. Jacobson, Web prefetching between low-bandwidth clients and proxies: potential and performance, *Proceedings of ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, Atlanta, CA, May 1999, pp. 178–187.
- [11] M.J. Feeley, W.E. Morgan, F.H. Pighin, A.R. Karlin, H.M. Levy, Implementing global memory management systems, *Proceedings of the 15th ACM Symposium on Operating System Principles*, Copper Mountain, CO, December 1995, pp. 201–212.
- [12] J. Fox, Browser cache switch for Internet explorer, *WebDeveloper Conference 2000*, San Francisco, CA, September 2000.
- [13] S. Gadde, M. Rabinovich, J. Chase, Reduce, reuse, recycle: an approach to building large Internet caches, *Proceedings of the sixth Workshop on Hot Topics in Operating Systems*, Capecod, MA, May, 1997, pp. 93–98.
- [14] G. Gonet, R. Baeza-Yates, *Handbook of Algorithms and Data Structures in Pascal and C*, Addison-Wesley, Reading, MA, 1991.
- [15] S.D. Gribble, E.A. Brewer, System design issues for Internet middleware services: deductions from a large client trace, *Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems*, Monterey, CA, December 1997, pp. 207–218.
- [16] S. Jin, A. Bestavros, Popularity-aware GreedyDual-size Web proxy caching algorithms, *Proceedings of 20th International Conference on Distributed Computing Systems*, (ICDCS'2000), Taiwan, China, April 2000, pp. 254–261.
- [17] R.P. Klemm, WebCompanion: a friendly client-side Web prefetching agent, *IEEE Trans. Knowledge Data Eng.* 11(4) (July/August 1999) 577–594.
- [18] T.M. Kroeger, D.D.E. Long, J.C. Mogul, Exploiting the bounds of Web latency reduction from caching and prefetching, *Proceedings of Usenix Symposium Internet Technologies and Systems*, Monterey, CA, December 1997, pp. 13–22.
- [19] T.S. Loon, V. Bharghavan, Alleviating the latency and bandwidth problems in WWW browsing, *Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems*, Monterey, CA, December 1997.
- [20] R. Malpani, J. Lorch, D. Berger, Making World Wide Web caching servers cooperate, *Proceedings of the 4th International World Wide Web Conference*, Boston, MA, December 1995.
- [21] A.J. Menezes, P.C. van Oorschot, S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Boca Raton, FL, October 1996.

- [22] B.S. Michel, K. Nikoloudakis, P. Reiher, L. Zhang, URL forwarding and compression in adaptive Web caching, Proceedings of IEEE INFOCOM 2000, March, 2000, pp. 670–678.
- [23] National Lab of Applied Network Research: <http://www.ircache.net/> Sanitized access logs: <ftp://ircache.nlanr.net/Traces/Statistics>: <http://www.ircache.net/Cache/Statistics/>
- [24] V.S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, E. Nahum, Locality-aware request distribution in cluster-based network servers, Proceedings of the Eighth Symposium on Architectural Support for Programming Languages and Operating Systems (ASPLOS-8), October 1998, pp. 205–216.
- [25] J. Pitkow, P. Pirulli, Mining longest repeating subsequences to predict world wide Web surfing, Proceedings of the 1999 Usenix Symposium on Internet Technologies and Systems, Boulder, CO, April 1999, pp. 139–150.
- [26] M. Reddy, G.P. Fletcher, An adaptive mechanism for Web browser cache management, IEEE Internet Comput. 2 (1) (January 1998) 78–81.
- [27] A. Rousskov, V. Soloviev, A performance study of the squid proxy on HTTP/1.0, World Wide Web 2 (1–2) (January 1999) 47–67.
- [28] R. Tewari, M. Dahlin, H.M. Vin, J.S. Kay, Design considerations for distributed caching on the Internet, Proceedings of the 19th IEEE International Conference on Distributed Computing Systems (ICDCS), Austin, TX, May 1999, pp. 273–284.
- [29] A. Wolman, G. Voelker, N. Sharma, N. Cardwell, M. Brown, T. Landray, D. Pinnel, A. Karlin, H. Levy, Organization-based analysis of Web-object sharing and caching, Proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems, Boulder, CO, October, 1999, pp. 25–36.
- [30] L. Xiao, X. Zhang, S.A. Kubricht, Incorporating job migration and network RAM to share cluster memory resources, Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing (HPDC-9), Pittsburgh, PA, August 1–4, 2000, pp. 71–78.
- [31] C. Yoshikawa, B. Chun, P. Eastham, A. Vahdat, T. Anderson, D. Culler, Using smart clients to build scalable services, Proceedings of the USENIX 1997 Annual Technical Conference, Anaheim, CA, January 1997, pp. 105–117.

Li Xiao is an assistant professor of Computer Science and Engineering at Michigan State University. She received the BS and MS degrees in Computer Science from Northwestern Polytechnic University, China, and the Ph.D. degree in Computer Science from the College of William and Mary

in 2002. She is a recipient of USENIX Fellowship for her Ph.D. dissertation research from 2001 to 2002. Her research interests are in the areas of distributed and Internet systems, system resource management, and design and implementation of experimental algorithms. She is a member of ACM and IEEE.

Xin Chen is a Ph.D. candidate of Computer Science at the College of William and Mary. He received his B.S. degree from Xi'an Jiaotong University, China, in 1996, and M.S. degree from the University of Science and Technology of China in 1999, both in Computer Science. His research interests are in the areas of distributed, Internet and networking systems.

Xiaodong Zhang is Lettie Pate Evans Professor and Chairman of the Computer Science Department at the College of William and Mary. He was the Program Director of Advanced Computational Research at the U.S. National Science Foundation from 2001 to 2003. He is a past editor of *IEEE Transactions on Parallel and Distributed Systems*, and currently serves as an associate editor of *IEEE Micro*. He received his B.S. degree in Electrical Engineering from Beijing Polytechnic University in 1982, M.S. and Ph.D. degrees in Computer Science from University of Colorado at Boulder in 1985 and 1989, respectively. His research interests are in the areas of parallel and distributed computing and systems, and computer architecture.

Yunhao Liu received his BS degree in Automation Department from Tsinghua University, China, in 1995, and a MA degree in Beijing Foreign Studies University, China, in 1997, and a MS degree in Computer Science at Michigan State University in 2003. He was a Regional Manager in China Telecom from 1997 to 1998, and a Deputy Director in China Post from 1998 to 2001. He is now a Ph.D. student of Computer Science and Engineering at Michigan State University. His research interests are in the areas of distributed systems, Internet and e-commerce technologies, peer-to-peer systems, location-aware computing, and high-speed networking. He is a student member of the IEEE.