Building a Large and Efficient Hybrid Peer-to-Peer Internet Caching System

Li Xiao, *Member*, *IEEE*, Xiaodong Zhang, *Senior Member*, *IEEE*, Artur Andrzejak, and Songqing Chen, *Student Member*, *IEEE*

Abstract—Proxy hit ratios tend to decrease as the demand and supply of Web contents are becoming more diverse. By case studies, we quantitatively confirm this trend and observe significant document duplications among a proxy and its client browsers' caches. One reason behind this trend is that the client/server Web caching model does not support direct resource sharing among clients, causing the Web contents and the network bandwidths among clients to be relatively underutilized. To address these limits and improve Web caching performance, we have extensively enhanced and deployed our browsers-aware framework, a peer-to-peer Web caching management scheme. We make the browsers and their proxy share the contents to exploit the neglected but rich data locality in browsers and reduce document duplications among the proxy and browsers' caches to effectively utilize the Web contents and network bandwidth among clients. The objective of our scheme is to improve the scalability of proxy-based caching both in the number of connected clients and in the diversity of Web documents. In this paper, we show that building such a caching system with considerations of sharing contents among clients, minimizing document duplications, and achieving data integrity and communication anonymity is not only feasible but also highly effective.

Index Terms—Internet systems, peer-to-peer systems, proxy caching, browser caching, data integrity, communication anonymity.

1 Introduction

A *proxy-browser system* is a commonly used client/server infrastructure for Web caching, where a group of networked clients connects to a proxy cache server and each client has a browser cache. A standard Web caching model built on a proxy-browser system has the following data flows: Upon a Web request of a client, the browser first checks if the requested document 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 document 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 the document exists in other browsers' caches.

This model has two features that prevent it from effectively utilizing the rapid improvement in Internet technologies and from adapting, in a timely manner, the changes of the supply and demand of Web contents. First, with a significant increase of memory and disk capacity in workstations and PCs and with the improvement of Web browser caching capability, users are able to enlarge browser cache size for faster access to more cached documents and to retain the documents in an organized manner for a longer period of time. Furthermore, studies have shown that one reason for proxy cache hit ratio decline is that more requests are absorbed by local browsers (e.g., [1]), so there exist some documents that are already replaced in the proxy cache but still retained in one or more browser caches. This is due to the fact that the request rates to the proxy and to browsers are different, causing the replacement in the proxy and browsers at a different pace. However, the browser caches are not shared among the clients and the available locality and bandwidth among browsers are underutilized in Web caching. When a requested document misses in a local browser cache and the proxy cache, it may still have been cached in other browser caches.

Second, with the rapid increase of Web servers and the huge growth of Web client populations in both numbers and types, the requested Web contents have become, and will continue to become, more diverse, causing a decrease of proxy hit ratios. Meanwhile, existing proxy-browser systems cause a large amount of document duplications. The amount of document duplications between the proxy and browser caches is generally very large because the requested document is always cached in both the proxy and a requesting client browser. It is also highly possible to generate a large amount of document duplication among browsers for the following reason: When multiple clients request some popular documents cached in the proxy, each requesting client will duplicate these documents in its local browser cache. Envisioning the rapid advancement of networking technology, we argue that the duplication issue can seriously limit potential benefits to be gained from the current structure of Web caching systems. Here are the reasons:

1. High-speed networking technology will soon close the speed gap between local and remote accesses. Therefore, file sharing and transferring among clients will become easy and a common practice.

- *L. Xiao is with the Department of Computer Science and Engineering, Michigan State University, East Lansing, MI 48824. E-mail: lxiao@cse.msu.edu.*
- *X. Zhang and S. Chen are with the Department of Computer Science, College of William and Mary, Williamsburg, VA 23187. E-mail: {zhang, sqchen}@cs.wm.edu.*
- *A. Andrzejak is with the Division of Computer Science, Zuse-Institute Berlin, Takustr. 7, D-14195 Berlin, Germany. E-mail: andrzejak@zib.de.*

Manuscript received 29 June 2003; revised 22 Nov. 2003; accepted 25 Nov. 2003.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0105-0603.

TABLE 1
Selected Web Traces

Traces	Period	# Requests	Total GB	Infinite Cache (GB)	# Clients	Max. Hit Ratio	Max. Byte Hit Ratio
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 %
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%

2. Data duplications will cause additional overhead, such as global data invalidations and broadcasting. Minimizing the number of owners for a data document also strengthens security and privacy protections.
3. Unnecessary data duplications over the Internet can widely waste storage space. Both the additional operation and space overheads will certainly limit the scalability of Internet performance.

The capability of the proxy cache will be limited as the number of clients and document types increase. Adding more and more additional space to a proxy cache might temporarily increase the proxy hit ratio, but is not a remedy against decreasing efficiency.

Peer-to-Peer (P2P) computing is an emerging distributed computing technology that enables direct resource sharing of both computing services and data files among a group of mutually trusted clients over the Internet. There are several reasons why peer-to-peer computing model is demanding [18]. One of the reasons is that the increasing amounts of Web content and bandwidth among clients will get underutilized if client/server models continue to be dominant in the Internet, such as the centralized Web search engines and Web servers. Similarly, if every client has to be served by a proxy on a miss, the number of clients connected to the proxy will be limited (nonscalable) and the available bandwidths among the clients will be underutilized. P2P systems can be classified into two classes: a pure P2P, where peers share data without a centralized coordination, and a hybrid P2P, where some operations are intentionally centralized, such as indexing of peers' files.

We have proposed a hybrid peer-to-peer Web caching management framework, called *browsers-aware caching* (see [37] and [39]). In this paper, we present its enhancement and deployment by practically making the browsers and their proxy share the contents to exploit the neglected but rich data locality in browsers and by reducing document duplications among the proxy and browsers' caches to effectively utilize the Web contents and network bandwidth among clients. Our objective is to apply a hybrid P2P technique in Web caching, aiming at reducing latency, exploiting idle resources, and facilitating the exchange of distributed files. This system can be very effective in an environment where the bottleneck of the communication path is between the proxy and the origin server. However, the P2P design domain also opens other challenging issues. In our study, data integrity and communication privacy are the two issues that must be addressed.

In this paper, we make three contributions: First, using trace-driven simulations with Web traces, we compare the proposed Web caching management scheme with the traditional approach. As a result, we show that both the hit ratio and byte hit ratio of this scheme are significantly higher and the Web server access latency is substantially reduced. We also empirically show that the performance of our scheme compares very favorably with the performance of offline Web caching algorithms. Second, we ensure data integrity using checksums and enforce communication anonymity by effectively hiding the identity of each client as it provides or requests documents. Finally, we have implemented a browsers-aware prototype by interfacing two user daemons with the Squid proxy and the Netscape browser. We show that building such a peer-to-peer caching system with considerations of minimizing document duplications, client anonymity, and data reliability is not only feasible but also highly effective.

2 Motivation and Rationale

2.1 Trends of Hit Ratio Decrease and Access Diversity

The hit ratios to proxy caches have been observed to be in a decreasing trend for a few years. There are two major reasons for the decrease. First, e-commence and personalized services have increased the percentage of dynamic documents. Dynamic documents are usually noncachable. Many recent studies (e.g., [42]) have shown that requests for dynamic Web content also contain substantial locality for identical requests and have provided several methods to cache dynamic Web contents. This is not our focus in this paper. Second, the increase of proxy cache size has not been sufficient to keep up with the rapid increase in the numbers and types of Web servers and clients' diverse interests. For example, Barford et al. [3] give a comprehensive study of the changes in Web client access patterns based on the traces collected from the same computing facility with a similar nature of the user population separated by three years. Their experiments show that, compared with the data three years ago, the hit ratios are reduced and the most popular documents are less popular in the transfer data set. This implies that accesses to different types of Web servers have become more evenly distributed. One reason for this, we believe, is that the access variations have increased as more and more Web servers are emerging.

In order to further understand the increasing diversity in Web accesses and its effects to proxy hit ratios, we have analyzed the access pattern of browser traces used in [3]. We select the traces in a period of two months of the two years, which are denoted as BU-95 and BU-98 (see Table 1), respectively. The difference between the total number of requests of BU-95 trace and that of BU-98 trace is very large. In order to make fair comparisons, we compare request ratios

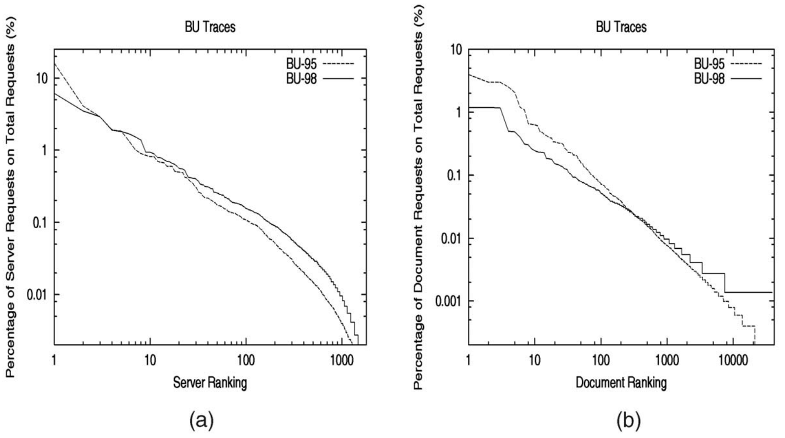


Fig. 1. The percentage of the requests to each server or document over the total requests versus server ranking or document ranking.

instead of the numbers of requests between these two traces. We compare two statistical results: 1) the percentages of requests to different servers over the total requests, and 2) the percentages of requests to different documents over the total requests, both reflecting access distributions.

In Fig. 1a, we plot the percentage of the requests to each server over the total requests versus server ranks. The ranks are obtained by sorting the percentages of accesses to servers in decreasing order. In Fig. 1b, we plot the percentage of the requests to each document over the total requests versus document ranks by sorting the percentages of accesses to documents in decreasing order. We have two observations. First, the access distributions presented in the two figures are consistent with an observation reported in [5]—the distributions of requesting accesses and server accesses follow the Zipf-like distribution Ω/i^{α}. Second, the access patterns had been changed toward more even distributions during the three year period. Specifically, the request percentages of BU-95 trace are higher than those of BU-98 for very high rank servers and documents. But, for lower rank servers and documents, the request percentages of BU-98 exceed those of BU-95. So, for the same cache size, BU-95 could get a higher hit ratio than BU-98. However, the hit ratios of BU-98 can be increased at a faster pace than that of BU-95 if the cache is larger than a certain size.

This type of access pattern change demands progressive increase of the cache size in order to retain a fixed hit ratio during a period of time. To estimate the cache size requirement difference between BU-95 and BU-98 for a given hit ratio, we fit the curves in Fig. 1 into Zipf-like distributions. We assume that the file size with the same rank in BU-95 is the same as that in BU-98 and the priority of caching a document is based on the document popularity. From the fit Zipf-like distribution curves, we estimate that a 12.7 times larger cache is needed for BU-98 to achieve a given hit ratio in BU-95. (In fact, we obtained a number of 10 that is smaller than 12.7 from simulation results to be presented in Section 4. This is because the average document size in BU-98 is smaller than that in BU-95.) These assumptions and numbers may not be directly used to guide the proxy cache design, but we attempt to show the trends of decreasing hit ratios in proxies and the diversity of the Web contents. In order to retain the proxy cache hit ratios, we have to enlarge the cache size as time passes. However, the proxy cache size enlargement will no longer be sufficient. Therefore, we should consider alternative methods to effectively utilize the limited caching space.

We have also analyzed proxy access pattern statistics of NLANR (National Lab of Applied Network Research) available in the public domain [29] between 1998 and 2001 and observed the same trend. We do not show and discuss the analytic figures in this paper because the data comes from proxy traces.

2.2 Case Studies of Duplications in Web Caching

We have analyzed the two BU browser traces described in the previous section and two Boeing traces also listed in Table 1. The Boeing Company collected anonymous logs from Boeing's Puget Sound perimeter (firewall) proxies by using an anonymizer tool (log2anon) and made these logs available in [6]. For privacy reason, client IP addresses are not identical between two different days, so we use traces based on one day's log file. We have used one day's trace on 4 March 1999 and one day's trace on 5 March 1999, which are the most recent traces in this site.

These traces have operated in a simulated system with an infinite proxy cache and infinite browser caches, where *infinite proxy/browser cache* is the total size storing all the unique requested documents in the proxy/browser cache. There are two types of data sharing in Web surfing: individually requested documents by a single client and commonly requested documents by multiple clients. We define the "intrasharing" ratio as the percentage of the requests only hit in local browsers for individual usage of clients out of the total **hit requests** in the proxy-browser system. We further define the "intersharing" ratio as the percentage of the requests coming from multiple clients but hitting the same documents out of the total **hit requests** in the proxy-browser system.

We have three observations based on the trace analysis results reported in Table 2. First, the average hit ratio of the two traces is 48.69 percent, which means that 51.31 percent of requested documents are only accessed once and they remain in both proxy and browser caches. Second, among the total **hit requests** in the proxy-browser system, the average intrasharing ratio is 36.20 percent. Since this large

TABLE 2
Trace Analysis on Document Duplications and Sharing Based on the Proxy-Browser System Hit Ratios, Intrasharing Ratios, and Intersharing Ratios

Traces	BU-95	BU-98	Boeing-4	Boeing-5	**average**
hit ratio (%)	64.14	40.62	44.91	45.07	**48.69**
intra-sharing (%)	27.64	35.18	39.55	42.42	**36.20**
inter-sharing (%)	72.36	64.82	60.45	57.58	**63.80**

portion of documents is only for individual usage, the documents do not need to be cached in the proxy, but only need to be cached in the requesting local browser caches. Unfortunately, the standard Web caching model stores this high percentage of documents in the proxy. Finally, the hits for intersharing by multiple clients that need to be cached in the proxy is 63.80 percent. However, documents of this type are duplicated in the proxy cache and multiple browser caches.

Our analysis and case studies show that a significant amount of document duplication exists in commonly used Web caching models. If supply and demand of diverse Web contents are continually increased, this duplication will soon limit the effective utilization of caching space. In addition, current Web caching models lack a data sharing mechanism between the proxy and browsers with which to further exploit data locality and utilize caching space. This preliminary trace analysis motivates us to propose new caching management schemes to improve performance by exploiting data locality in browsers and reducing the document duplications among a proxy and its browsers to utilize more caching space.

3 The Enhancement of Browsers-Aware Caching and Its Data Structures

3.1 The Basic Framework

Basically, 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, which is called *browser index file*. If a user request misses in its local browser cache and the proxy cache, the browsers-aware proxy server will search the browser index file attempting to find it in a client's browser cache before sending the request to an upper level server. If such a hit is found in a client, there are two alternative implementations to let the requesting client access the data object. First, the proxy server will inform this client to 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. The anonymity issues of these two alternatives will be discussed in Section 5.

Upon a client request, the browsers-aware caching scheme provides the following data flows for document service and storage management:

1. When the request is missed in the entire proxy-browser system, the requested document will be provided by an upper level proxy or a Web server. The initial document coming externally will be cached only in the requesting browser. However, if the proxy cache has enough free cache space for the document, it can be cached in the proxy at the same time.
2. If the request is a hit in the local browser, the document will be read from the browser cache.
3. If the request misses in its local browser cache but hits in the proxy, then, in addition to providing the document, the proxy will increment the counter of the number of remote accesses to this document from this requesting client. The proxy will inform the requesting client to cache this document only if the value of this counter is larger than a predetermined threshold, TH_BROWSER.
4. If the request is a miss in the local browser and the proxy, the browser index file in the proxy will be searched to see if the document is cached in another browser cache. If the request is a hit in another client's browser cache, then the hit browser will do two bookkeeping operations besides providing the document: a) increment the counter of the total number of distinguished remote requesting clients to this document if the requesting client accesses this document for the first time and b) increment the counter of the number of remote accesses to this document from this client. If the first counter is larger than a predetermined threshold, TH_PROXY, it means this document is shared by a sufficient number of clients so that the hit browser will transfer and cache the document to the proxy. The requesting browser is informed to cache this document only if the value of the second counter is larger than TH_BROWSER.

In the above items 1, 3, and 4, the document may be cached in either a browser cache or the proxy. When the browser cache or the proxy cache does not have sufficient space to store the document, one or more currently cached documents have to be replaced. LRU_Threshold (which does not cache a document larger than a threshold size) is used as the basic replacement policy for our scheme. (Most practical systems use algorithms similar to LRU_Threshold [32].) For a document larger than the threshold, our scheme also caches it as long as the cache has enough free space, but it is marked as an LRU document. The cache size threshold used in LRU_Threshold in the proxy and a browser cache is different due to significant difference of their cache sizes.

Major contributions of this paper are browser-aware caching data structures for a practical implementation and its enhancement of reducing duplications. The rest of the sections will present these issues.

3.2 Data Structures and Operations

Besides the browser index file, two other structures are maintained to facilitate this scheme. One structure allocated in each browser is used to manage cached documents in it. Another structure allocated in the proxy is used to manage all documents cached there.

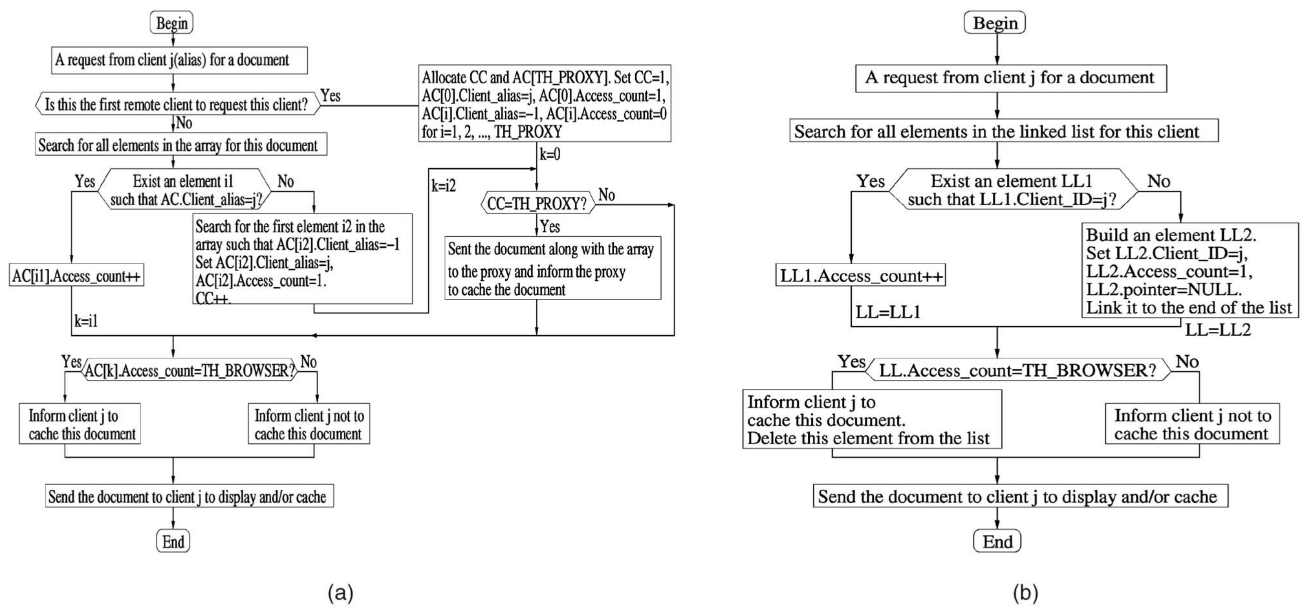

Begin
A request from client j(alias) for a document
Is this the first remote client to request this client?
Yes
No
Allocate CC and AC[TH_PROXY]. Set CC=1, AC[0].Client_alias=j, AC[0].Access_count=1, AC[i].Client_alias=−1, AC[i].Access_count=0 for i=1, 2, ..., TH_PROXY
k=0
Search for all elements in the array for this document
Yes
Exist an element i1 such that AC.Client_alias=j?
No
k=i2
CC=TH_PROXY?
No
Yes
Search for the first element i2 in the array such that AC[i2].Client_alias=−1 Set AC[i2].Client_alias=j, AC[i2].Access_count=1. CC++.
Sent the document along with the array to the proxy and inform the proxy to cache the document
AC[i1].Access_count++
k=i1
Yes
AC[k].Access_count=TH_BROWSER?
No
Inform client j to cache this document
Inform client j not to cache this document
Send the document to client j to display and/or cache
End
(a)
Begin
A request from client j for a document
Search for all elements in the linked list for this client
Yes
Exist an element LL1 such that LL1.Client_ID=j?
No
LL1.Access_count++
LL=LL1
Build an element LL2. Set LL2.Client_ID=j, LL2.Access_count=1, LL2.pointer=NULL. Link it to the end of the list
LL=LL2
Yes
LL.Access_count=TH_BROWSER?
No
Inform client j to cache this document. Delete this element from the list
Inform client j not to cache this document
Send the document to client j to display and/or cache
End
(b)

Fig. 2. (a) The management operations in each browser when a remote client request hits in it. (b) Management operations in the proxy when a client request hits in the proxy.

3.2.1 Browser Index File in the Proxy

This browser 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 a timestamp of the file, or the TTL (Time To Live). 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 that, the proxy deletes the corresponding index item.

3.2.2 The Structure in Each Browser

A counter and an array are allocated for each cached document that has been requested by other clients. The counter CC keeps the number of other clients that have accessed the document. Its value will be used to check if this document should be cached in the proxy. Each element of the array AC has two fields: $Client\_Alias$ and $Access\_Count$. An $AC.Client\_Alias$ records a client who has accessed the document. $AC.Client\_Alias$ is produced by the proxy to hide the true identity of the requesting client. The aliases are consistent and untraceable, as in LPWA [15]. $AC.Access\_Count$ records the number of accesses from the corresponding client. The array is of size TH_PROXY, of which CC elements are in use. It is allocated for a document only if there is a remote client requesting this document. When a document is replaced, the counter and array for this document are also replaced.

Fig. 2a presents the management operations when a remote client requests a document cached in this browser. When a browser is informed to cache a document sent by another client, it will cache this document.

3.2.3 The Structure in the Proxy

Each cached document in the proxy needs to count the number of accesses to this document from different requesting clients. This is used to check if this document should be duplicated in a requesting client browser. We use a linked list for each document. Each element of the list LL includes three fields:

1. $Client\_ID$: the ID number of a requesting client,
2. $Access\_Count$: the number of requests from this client, and
3. $Pointer$: a pointer to link to the next element.

A new element is allocated to the linked list of a document only if this document is requested by a client for the first time. When a document is replaced, its linked list is also replaced.

Fig. 2b presents the management operations when a client request hits in the proxy. When the proxy is informed to store a document sent by a client, the proxy first caches the document and then copies the necessary elements of the array sent by the client to the corresponding fields in the newly created linked list. (The necessary elements are those with $AC.Client\_ID \neq -1$ and $AC.Access\_Count < TH\_BROWSER$.) When the proxy has to fetch a document outside the proxy-browser system, it will pass the document to the requesting client and inform the client to cache it. The proxy will not cache the new arrival document.

3.3 Offline Algorithms for Performance Comparisons

The goal of obtaining optimal hit ratio and byte hit ratio in a proxy-browser system is equivalent to finding optimal replacement algorithms for objects with different sizes in a single cache whose size is the accumulated size of the proxy and all browsers. Studies in [21] provide two offline algorithms that are close to the optimal replacement algorithms. These offline algorithms are not viable in practice due to their requirement of knowing future requests. However, in order to evaluate the effectiveness of the proposed schemes, we compare their performance with that of the offline algorithms. This section gives an overview of two models of cost measurement for offline Web caching algorithms and discusses the respective approaches. They both follow [21] and are also discussed in [2]:

1. the **Fault Model**, where the cost of an algorithm for a request sequence σ equals the number of cache misses and
2. the **Bit Model**, where we sum up the sizes of the documents each time they are brought into cache.

Both models do not discriminate whether a document is stored in a proxy or in a browser cache.

If all documents have the same size and same costs of bringing them into cache, *Belady's Rule* [4] is known to be an optimal strategy for evicting pages from the cache: On a fault evict, the *most distant page* is the page whose next request is farthest in the future. However, for caching of Web documents with different sizes, these assumptions are not appropriate. Therefore, we consider, for each of the above models, a separate offline algorithm.

For the Fault Model, we use the Offline Fault Model Algorithm (OFMA) [21], which is shown in Algorithm 1. It guarantees that, for any request sequence σ, the number of cache misses is within the factor $2\log k$ of the number of cache misses for an optimal offline algorithm. Here, k is the ratio between the largest and the smallest document in σ.

Algorithm 1 Offline Fault Model Algorithm [21]

```
Divide the documents into at most $\lfloor \log k \rfloor + 1$ $l$-classes $C_l$,
where $C_l$ holds the documents of sizes $[2^l, \dots, 2^{l+1} - 1]$.
for each request to a document $d$ in a $l$-class:
  if $d$ is not in the cache:
    bring it in.
    if size of the cache is exceeded:
      for all $j$, do twice:
        if $C_j$ is not empty:
          evict the most distant document in $C_j$.
```

As for the BIT Model, we use the Offline Bit Model Algorithm (OBMA) from [21]. The cost of this algorithm is essentially within the factor $5(\log k + 4)$ of the optimal offline algorithm. As in OFMA, the documents are first divided into l-classes. If the cache capacity is exceeded by h when a new object is cached, OBMA evicts, from every class, the most distant objects until there is enough room or there are no objects in that class. In order to avoid evicting a large page when the cache is only exceeded by a small amount, OBMA maintains a counter for each class. If h is smaller than the most distant object in a class, it is added to the counter of this class. When the counter is larger than the size of the object, OBMA evicts the object and subtracts the size of the object from the counter.

The goals of the Fault Model and the Bit Model are to maximize hit ratios and byte hit ratios, respectively. In order to show how close the performance of the proposed caching scheme is to the optimal one, we compare the performance of our scheme with that of the two offline algorithms.

4 Performance Evaluation

In performance evaluation, we have conducted the following tasks in order to evaluate the performance of the browser-aware scheme comparatively and fairly:

- We first examine browser and proxy cache sizes in practical systems to provide a basis and a rationale for us to configure our simulated Web caching system.
- We compare the performance of the browser-aware scheme, including both browser sharing and duplication reduction with the conventional proxy-and-local-browser model and with the basic browser-aware model (browser-sharing only). In addition, we use an offline-algorithm to compare how close the browsers-aware model is to optimal.
- Specifically, we compare the hit ratios and byte hit ratios among the selected schemes by

  1. changing the proxy cache size,
  2. changing browser cache size,
  3. changing replacement threshold,
  4. comparing the memory hit ratios, and
  5. scaling the number of clients.

  These comparisons allow us to look into the performance insights of different schemes and to show the effectiveness of the browser-aware scheme.
- We estimate the achieved latency reduction by sharing browsers and by reducing the duplications.

4.1 Sizes of Browser and Proxy Caches

The study in [32] evaluates seven Squid proxies covering several levels of the caching hierarchy from leaf university proxies, to top level proxies for large country-wide networks, and to the international root proxy located at NLANR. Three proxies are leaf proxies which are related to our study: *ruu* from The Netherlands, *uit* from Norway, and *adfa* from Australia. Their proxy cache related configurations are listed in the second to fourth columns of Table 3. 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 3 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 that each client's browser has a cache. If we use the average proxy cache size per client in Table 3 as the browser cache size of each client, the memory space ranging from 0.04 MB to 0.08 MB is certainly too small. The total cache size ranging from 7.34 MB to 10.86 MB is also not large enough in practice for today's computer systems [23]. Therefore, in our study, we define β as

TABLE 3
Representative Proxy Cache Configurations Reported in [32]

Proxies	hot memory	disk cache	# clients	**memory cache/client**	**disk cache/client**
ruu	32MB	5.6GB	518	0.0618MB	10.8MB
uit	32MB	3.8GB	378	0.0847MB	10.1MB
adfa	32MB	5.8GB	798	0.0401MB	7.3MB

$$\beta = \frac{\sum_{i=1}^{P} Cache_{browser_i}}{Cache_{proxy}}, \qquad (1)$$

where $Cache_{browser_i}$ is the size of a client browser cache, P is the number of clients, and $Cache_{proxy}$ is the size of the proxy cache responsible for the P clients. The β is the ratio of the accumulated browser cache size to the proxy cache size, which is in a range of 0.1 to 50 in our experiments. If the accumulated browser cache size increases faster than the increase of the proxy cache size, the value of β tends to increase as both clients and the proxy server are upgraded.

4.2 Performance of the Browser-Aware System

Using trace-driven simulations, we have evaluated and compared the performance of four caching schemes with the two BU browser traces and two Boeing traces:

1. *proxy-and-local-browser*: If a request misses in its local browser and the proxy cache, the proxy will send the request to an upper level server without considering if the document exists in other browsers' caches.
2. *basic-browsers-aware*: If a user request misses in its local browser cache and the proxy cache, the browsers-aware proxy server will attempt to find the requested document from other browsers before sending the request to an upper level server, without considering unnecessary caching duplications.
3. *browsers-aware*: This is the comprehensive scheme discussed in Sections 3.1 and 3.2.
4. *offline-algorithm*: These are the offline algorithms close to optimal performance for comparisons with our proposed schemes which are discussed in Section 3.3.

We have validated our simulator using the similar method in [12]. We use three 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. *Latency* is the average access latency time per request.

We will discuss performance sensitivity to five important parameters: proxy cache size, browser cache size, cache size threshold for replacement, memory hit ratio, and the number of clients. We use ps to denote proxy cache size, which is based on the percentage of infinite proxy cache size. Here is the rationale for introducing this parameter. Because of the inconsistency of client IP addresses in the log files, we have to use one day's Boeing trace, whose size is so small that we cannot use an actual proxy cache size in our study. In our study, we scaled down the cache sizes accordingly by using a small percentage of the infinite proxy cache sizes. We use bs to denote browser cache size, which is based on the value of β. We assume that all browsers have the same size. We use th to denote cache size threshold used in LRU_Threshold cache replacement policy, which is a ratio of a given cacheable document threshold size over the proxy (or browser) cache size.

4.2.1 Evaluation of Sensitivity to the Proxy Cache Size

We have examined how sensitive the hit ratios and byte hit ratios are to the changes of the proxy cache size. For the experiments of each input trace, we set ps to 1, 2, 3, 5, and 10 percent of the infinite proxy cache size. We set $\beta = 10$. We also chose $th = 0.5$, which means the proxy size threshold is half of the proxy cache size and the browser size threshold is also half of browser cache size. Our trace-driven simulations show that our *browsers-aware* consistently outperforms *basic-browsers-aware* and *proxy-and-local-browser* for all the traces measured by hit ratios and byte hit ratios, in Figs. 3 and 4, respectively.

We first compare the performance of browser traces BU-95 and BU-98. For *proxy-and-local-browser*, BU-98's hit ratio is much lower than BU-95's hit ratio, but is also much lower than BU-98's hit ratio of *offline-algorithm*, which means that the hit ratio of the BU-98 trace has much more potential for improvement, while the hit ratio of the BU-95 trace has almost no room for improvement because it is so close to *offline-algorithm*. Both traces' byte hit ratios of *proxy-and-local-browser* have similar performance gaps as *offline-algorithm*. Our scheme of *browsers-aware* improves hit ratios and byte hit ratios of both traces, which are very close to *offline-algorithm*. As an example of $ps = 5\%$, the *offline-algorithm* outperforms *browsers-aware* by only 3.03 and 4.26 percent, measured by hit ratio and byte hit ratio for BU-98. So, *browsers-aware* is more promising to improve year 1998's trace than the trace three years before because requests in the year 1998's trace are more evenly distributed.

Boeing-4 and Boeing-5 are proxies traces, but we still see a big performance gain from *browsers-aware*. The intranetwork overhead simulation for these two traces shows that the increase of intranetwork overhead of *browsers-aware* is trivial, which does not offset the (byte) hit ratio gain from this scheme. So, reducing document duplications among cooperative proxies in the same organization is still promising for performance. But, it is not desirable for higher-level proxies, which are closer to servers and farther from clients because long distances among these proxies and potential networking congestion may offset (byte) hit ratio gains so that response time cannot be improved [13].

The percentage (ps) reflects the ratio between the actual proxy cache size and the accumulated size of unique documents. If the increase of the numbers of servers and of the diverse client populations is faster than the increase of the proxy cache size, the relative proxy cache size (ps) will continue to decrease. In other words, our *browsers-aware*

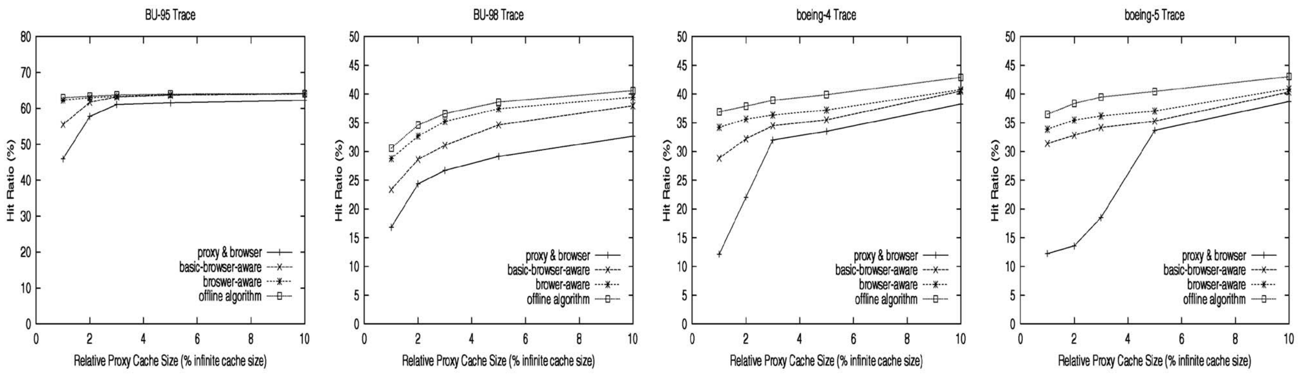


Fig. 3. Hit ratios of different caching schemes versus relative proxy cache sizes ($\beta = 10$, $th = 0.5$).

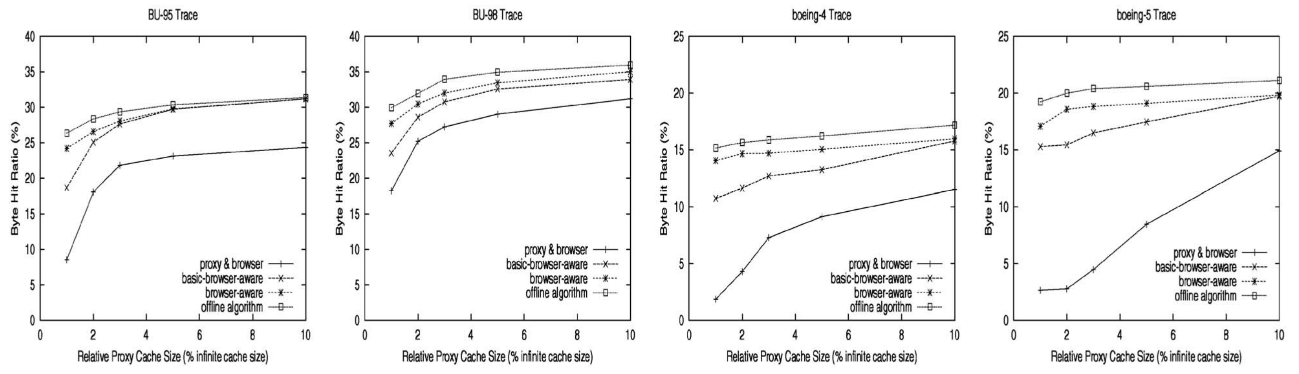


Fig. 4. Byte hit ratios of different caching schemes versus relative proxy cache sizes ($\beta = 10$, $th = 0.5$).

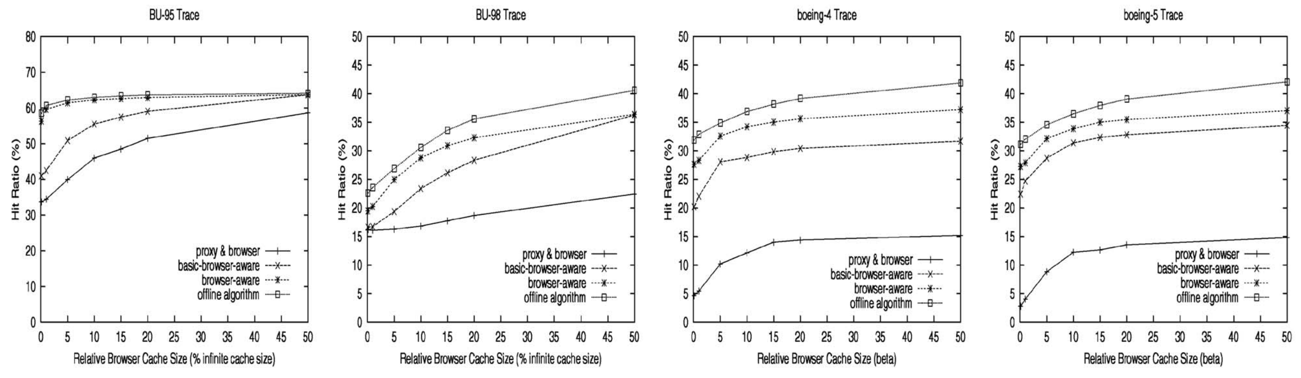


Fig. 5. Hit ratios of different caching schemes versus relative browser cache sizes with the four Web traces ($ps = 1\%$, $th = 0.5$).

scheme will be more performance beneficial as Web servers and Web client populations continue to increase in both numbers and types.

4.2.2 Evaluation of Sensitivity to a Browser Cache Size

We have examined how sensitive the hit ratios and byte hit ratios are to changes of a browser cache size. For the experiments of each input trace, we set β to 0.1, 1, 5, 10, 15, 20, and 50, respectively. We set ps to 1 percent of the infinite proxy cache size and chose $th = 0.5$.

Our trace-driven simulations show that our *browsers-aware* consistently outperforms *basic-browsers-aware* and *proxy-and-local-browser* for all the traces with all the given β values measured by hit ratios and byte hit ratios in Figs. 5 and 6, respectively. The performance gain of all the schemes is improved slowly after β reaches a certain value. The best performance gain was achieved for β in the range of 1 to 15. If β is too small, such as less than 0.1, the accumulated browser cache is not large enough to be effective for both *browsers-aware* and *basic-browsers-aware*. It is also not desirable to increase β to a very large value. (The paper in [34] also points this out). For the examples in Section 4.1, the range of 1 to 15 of β corresponds to a browser cache size in the range of 10 MBytes to 150 MBytes, which is a reasonable range of browser cache size in the current storage capacity of workstations.

4.2.3 Evaluation of the Sensitivity to the Replacement Threshold

We use the basic LRU_threshold cache replacement policy in both *proxy-and-local-browser* and *basic-browsers-aware*. We have revised the LRU_Threshold policy for *browsers-aware*,

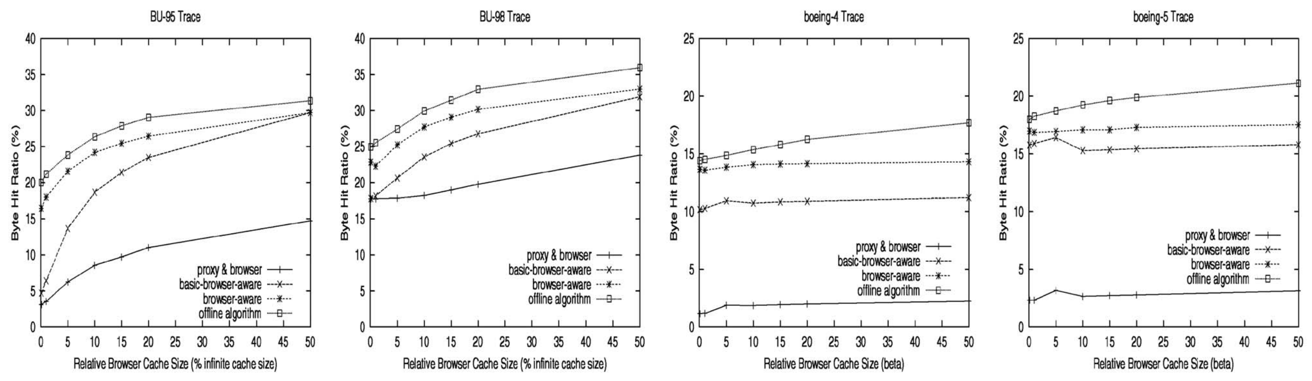


Fig. 6. Byte hit ratios of different caching schemes versus relative browser cache sizes with the four Web traces ($ps = 1\%$, $th = 0.5$).

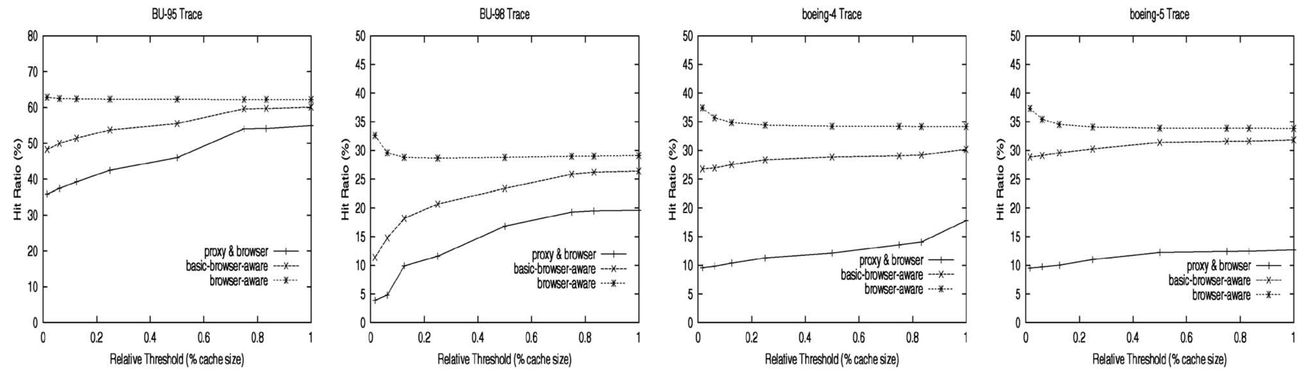


Fig. 7. Hit ratios of different caching schemes versus the replacement threshold with the four Web traces ($ps = 1\%$, $\beta = 10$).

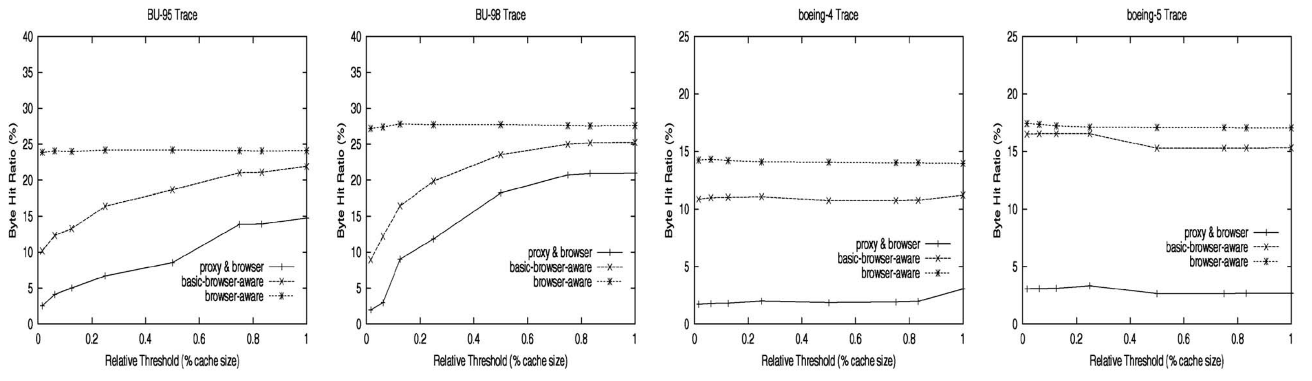


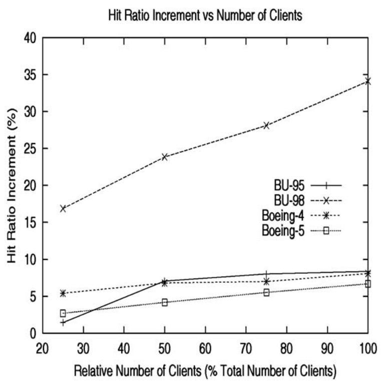
Fig. 8. Byte ratios of different caching schemes versus the replacement threshold with the four Web traces ($ps = 1\%$, $\beta = 10$).

where a document larger than the threshold could be cached as long as enough free caching space is available but is marked as an LRU document. We have examined how sensitive the hit ratios and byte hit ratios are to the changes of the replacement threshold. For experiments of each trace, the th variable is set to $\frac{1}{64}, \frac{1}{16}, \frac{1}{8}, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, \frac{5}{6}$, and 1, respectively. We set ps to 1 percent of the infinite proxy cache size and chose $\beta = 10$.

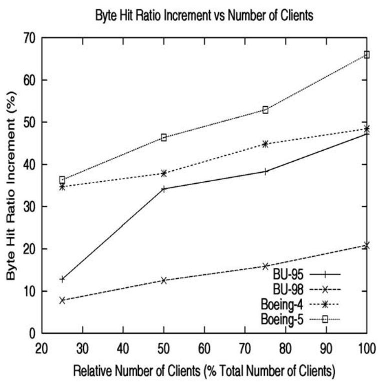
Our trace-driven simulations show that the *browsers-aware* consistently outperforms *basic-browsers-aware* and *proxy-and-local-browser* for all the traces with all the given relative thresholds measured by hit ratios and byte hit ratios in Figs. 7 and 8. Our experiments show that, in general, small cache threshold values are more effective for *browsers-aware* than large threshold values measured by the hit ratios. This is because file size distribution is heavy-tailed [3]. The average size of popular documents is smaller than that of unpopular documents. But, a very small threshold is not beneficial to performance measured by byte hit ratios. Comparing hit ratios of *browsers-aware* for trace BU-95 and BU-98, we show that small cache threshold values are more effective for BU-98 trace. This can be explained by the findings in [3]: BU-98 trace shows a shift toward smaller sizes overall than BU-95 trace. The threshold impact to (byte) hit ratios of *browsers-aware* is much less sensitive than those of *basic-browsers-aware* and *proxy-and-local-browser* for browser traces.

4.2.4 Performance Impact of Memory Hit Ratios

The *browsers-aware* 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



(a)



(b)

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

percentage of requests will hit in the main memory of browser caches and the proxy cache provided by the *browsers-aware*. (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 Table 3. 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* 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 KBytes is 10 ms.

Figs. 3 and 4 show that the hit and byte hit ratios of the *browsers-aware* at 2 percent of the *infinite* cache size are very close to those of the *proxy-and-local-browser* policy at 10 percent of the *infinite* cache size (the hit ratio comparison is 32.70 versus 32.66 and byte hit ratio comparison is 30.49 versus 31.21) for the BU-98 trace. However, the memory byte hit ratios of the two schemes are quite different under the same condition, which are 24.81 percent for the *browsers-aware* and 16.51 percent for the *proxy-and-local-browser* policy, respectively. The larger memory byte hit ratio of the *browsers-aware* in this case would reduce 26.93 percent of the total hit latency compared with the *proxy-and-local-browser*. We also compare *browsers-aware* at 1 percent of the *infinite* cache size and *proxy-and-local-browser* at 10 percent of the *infinite* cache size, where their byte/hit ratios are almost the same. Their memory byte hit ratios are 15.51 and 6.28 percent, respectively. The larger memory byte hit ratio of the *browsers-aware* in this case would reduce 28.88 percent 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.

4.2.5 Performance Impact of Scaling the Number of Clients

We have also evaluated the effects of scaling the number of clients to *browsers-aware*. For each trace, we observe its hit ratio (or byte hit ratio) increment changes by increasing the number of clients from 25 to 50 to 75 percent and to 100 percent of the total number of clients. We also regard 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 percent of the *infinite* proxy cache size when the relative number of clients is 100 percent. The byte hit ratio increment or the hit ratio increment of *browsers-aware* for a given trace is defined as

$$\frac{(byte)\ hit\ ratio\ of\ browsers\_aware - (byte)\ hit\ ratio\ of\ proxy\_and\_local\_browser}{(byte)\ hit\ ratio\ of\ proxy\_and\_local\_browser}.$$

Fig. 9a presents the hit ratio increment curves and Fig. 9b the byte hit ratio increment curves of the five traces as the relative number of clients changes from 25 to 100 percent. Our trace-driven simulation results show that both hit ratio increment and byte hit ratio increment of the *browsers-aware* proportionally increases as the number of clients increases. For some traces, the increments are significant. For example, the hit ratio increment of the BU-98 trace increases from 16.89 to 23.85 percent to 28.13 percent and to 34.13 percent as the relative number of clients increases from 25 to 50 percent to 75 percent and to 100 percent, respectively. The byte hit ratio increment of the Boeing-5 trace increases from 36.35 to 46.34 percent to 52.92 percent and to 66.02 percent.

The performance results indicate that *browsers-aware* is performance beneficial to client cluster scalability because it will exploit more browser locality and utilize more memory space as the number of clients increases in the cluster.

4.3 Latency Reduction

The access delay for fetching a missed document in the proxy cache from a remote server can be estimated by summing the network connection time and the data transferring time in the Internet. We estimated connection times and data transferring times by using the method presented in [20], where the connection time and the data transferring time are obtained by applying a least squares fit to measured latency in traces versus the size variations of documents fetched from different remote servers. The access latency to remote servers reduced by the *browsers-aware* can be further estimated by accumulating the latency times used to access remote servers for those requests missed in *basic-browsers-aware* or *proxy-and-local-browser*, but hit in *browsers-aware*. Our experiments show that the *browsers-aware* achieves average latency reduction of 21.25 percent, compared with the *basic-browsers-aware* scheme, and about 56.61 percent compared with the *proxy-and-local-browser* scheme.

4.4 Performance Summary

We have shown that the browser-aware scheme outperforms the other compared schemes under the conditions of changing the proxy cache size, the browser cache size, the replacement threshold, and scaling the number of clients. Furthermore, we show that the browser-aware scheme will be increasingly performance beneficial as the the number of Web servers increases and as the number of clients connected to a proxy scales. In addition to the high hit and byte hit ratios of the browser-aware scheme, the memory hit ratios on browsers and proxy are also high, which contributes to significant latency reductions.

5 Reliability and Privacy

In order to make the browsers-aware caching scheme feasible in practice, the reliability and privacy of the browser data must be seriously considered. These are also related to security. The first issue is reliability of shared files among browser caches. Since each client has the right to modify his/her browser cache, the browser data files that have been modified by an owner client are not reliable and secure for sharing among clients. We need to ensure the data integrity of the shared files among browser caches. The second issue is the privacy of clients who request and provide a file. A request from one browser (requesting browser) may be satisfied by another browser (hit browser). The requesting browser and the hit browser may not want to reveal their identities to each other and to other browsers and the hit document should not be visible by other clients to preserve the privacy of each client. This concern can be addressed by making anonymous communications between clients.

We have proposed protocols to enforce data integrity and communication anonymity. Our study shows that the associated overheads are trivial. These protocols are based on symmetric and public key encryptions [28]. In a symmetric key system, two communicating parties share an identical secret, the symmetric key, used for encryption and decryption. DES (Data Encryption Standard) is such an example. In a public key system (e.g., RSA), such a party has a public/private key pair. A public key can be accessed by everyone. A sender encrypts an outgoing message using a receiver's public key and the receiver uses its private key to decrypt this ciphertext.

5.1 Data Integrity

To ensure that a document received by a client is tamper-proof, we need to find a way for a requesting browser to check whether the content it receives is intact. For this purpose, we use the proxy server to produce a digital water mark in the following manner: For a document f, the digital watermark is produced by first generating a message digest using MD5 [31] and then encrypt the message digest with the proxy server's private key. We assume that the private key of the proxy is x, the corresponding public key is y, and the public keys of the browser caches are known to all peer clients. We use $K(M)$ to represent either 1) the message M being encrypted with the key K or 2) the ciphered message M being decrypted with decryption key K.

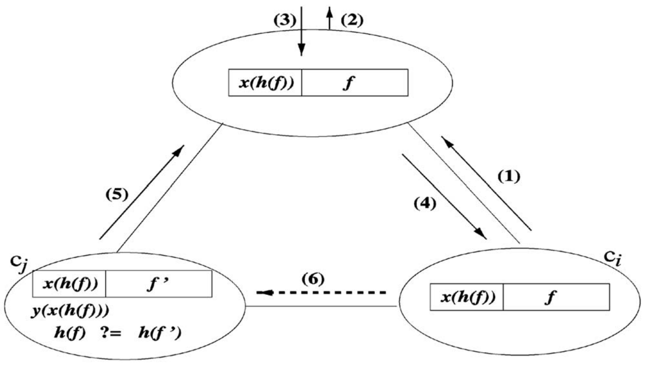


Fig. 10. Integrity protocol.

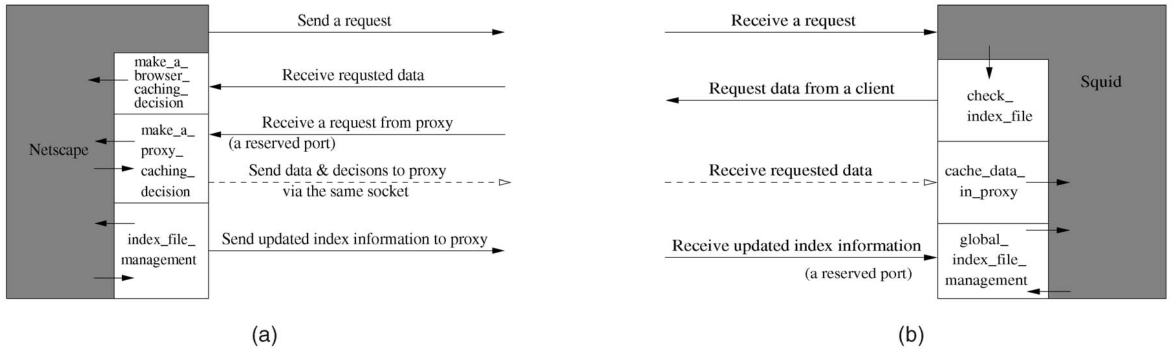
Fig. 10 shows the integrity protocol. Initially, when a client c_i sends a request to the proxy for a document, the proxy obtains the requested document, denoted as f, either from the server or an upper-level proxy. The proxy generates an MD5 message digest, $h(f)$, of the document. It then encrypts $h(f)$ with its private key x to produce a digital signature, $x(h(f))$. The message $\{f, x(h(f))\}$ is sent to the client c_i and stored in its local cache. If another client c_j requests the same document and this document has been replaced in the proxy cache and is found to be in c_i's cache, the proxy will instruct c_i to send the message $\{(h(f)), f\}$ to c_j. On receiving the message, c_j will produce a message digest for the received document using MD5 and compare the message digest with $y(x(h(f)))$. No client can tamper with the document f and produce a matching digital watermark because no client but the proxy server knows the private key of the proxy server.

5.2 Communication Anonymity

Our *browsers-aware* system hides the identities of both browser senders and receivers. This communication anonymity is ensured by having the proxy act as an anonymizing proxy. A client always sends a request to the proxy. The proxy contacts a targeted client and receives the content on behalf of the requesting client. The targeted client does not know which client requests the document and a requesting client does not know which client delivers the content. We have also developed several effective anonymity protocols that hide identities among peer browsers without or with limited centralized controls of the proxy. For detailed descriptions of mutual anonymity protocols, interested readers may refer to [38].

6 Implementations and Measurement Results

In order to implement the additional communication and computing functions in each client, the security and integrity protocols between clients, and the data management schemes for browsers-aware caching, we have built a system infrastructure based on existing client and proxy servers. The infrastructure consists of two parts: a client



Send a request
make_a_ browser_ caching_ decision
Receive requsted data
Netscape
make_a_ proxy_ caching_ decision
Receive a request from proxy
(a reserved port)
Send data & decisons to proxy via the same socket
index_file_ management
Send updated index information to proxy
(a)
Receive a request
Squid
Request data from a client
check_ index_file
Receive requested data
cache_data_ in_proxy
Receive updated index information
(a reserved port)
global_ index_file_ management
(b)

Fig. 11. The organization of (a) client daemon and (b) proxy daemon to interface with a client browser and the proxy.

daemon to interface its browser and to communicate with the proxy and a browsers-aware proxy server. Coordinating the operations between the two sites, we build a secured browsers-aware caching system.

6.1 A Client Daemon to Interface the Browser and Communicate with the Proxy

We have selected the mozilla (http://www.mozilla.org or netscape) software as the working browser system since it is widely used in applications. Instead of revising the browser source code, we have built a client daemon interfacing the browser and communicating with the proxy. This approach makes the commercial browser software still portable and keeps its independent functions. The client daemon consists of a pair of parent-child processes at the user level. The child process serves as a receptionist that "listens" at a reserved port to incoming messages of requesting data files from the proxy. If such a message is received, the receptionist searches and fetches the file from the client browser and sends it back to the proxy or sends it to a target client. The parent process serves as the browser file index manager. This manager periodically checks the status of file changes in the browser and timely sends the index updates to the proxy. Three major data management functions are implemented to coordinate caching activities between browsers and the proxy.

- *make_a_browser_caching_decision.* This function decides whether the arriving document should be cached in the local disk. The decision is made based on a threshold value of the local requesting counter (see the caching algorithm described in Section 3).
- *make_a_proxy_caching_decision*. This function decides whether a document requested by another client should be cached as a shared document in the proxy. The decision is made based on a threshold value of the global requesting counter (see the caching algorithm described in Section 3). For this purpose, a port is reserved for dedicated communication between a browser and the proxy. When a client sends back a document that proxy requests, it will use the same reserved port.
- *index_file_management*. This user function dynamically monitors the status of the local browser document index files. Whenever a sufficient amount of local files are replaced (for example, a 10 percent change) and the network is not busy, it will send the related items based on replaced files to the proxy for updating its browser index file.

With these three functions, the client daemon adds simple and sufficient functions to a client browser so that it is able to actively communicate with other clients directly or through the proxy. The client daemon is activated at the time when the system is booted. Fig. 11a illustrates the organization of the client daemon and its interface with the netscape browser.

6.2 A Browsers-Aware Proxy Server

We have selected the Squid proxy server (http://www.squid-cache.org) as the working system. Besides creating a global browser index file in the proxy, three additional functions are added to the proxy server:

- *check_index_file*. This function checks the global browser index file after a miss occurs in the proxy. If the index file search is successful, it sends a data request to the target client.
- *cache_data_in_proxy*. This function caches the data after receiving a positive decision from a client.
- *global_index_file_management*. This user function maintains and updates the global browser index file upon receiving a new file from an upper-level server or updated browser file status from a client. For this purpose, a port is reserved for a dedicated communication between a browser and the proxy.

The data integrity and communication anonymity issues are simply addressed by requiring the proxy server to do checksum, and handle the data searching and transferring among the browsers, as described in Section 5. Fig. 11b illustrates the organization of the proxy daemon and its interface with the Squid proxy.

6.3 Overhead Measurement and Analysis

6.3.1 Additional Operations by CPUs and Networks

There are three major items of additional operations involved in browsers and proxy if an object can be provided by another browser instead of going to a Web server:

- *Browser-index file searching*: The searching is done in the proxy after a request miss in the proxy. The browser index file consists of all the active URL's MD5 digests of browsers. We have used the searching facility for managing the cached documents in the Squid proxy. A hash function is used for the search, thus the search time is index file size independent. Specifically, function *storeGETPublic* is used, where function *hash_lookup* is called.

The searching time is denoted as T_{index}. Running the Squid proxy on a Pentium III 1000 MHz machine, we obtained the average searching time, $T_{index} = 0.0076\, ms$.

- *Requesting service from a client*: If the requested document is found in a browser cache after the index file searching, the proxy sends a request to the identified client. A requesting message is set to 256 bytes. The communication time is denoted as T_{req} and is dependent on a local area network speed.
- *Data delivering between a client and the proxy*: The browser fetches the requested document and sends it back to the proxy that delivers it to the requesting client. The data transferring time is denoted as T_{data} and is dependent on the local area speed and size of the document.

The additional browsers-aware service time is

$$T_{overhead} = T_{index} + T_{req} + T_{data}.$$

We measured this service time by varying the size of the requested document from browsers on a 100 Mb Ethernet and obtained $T_{overhead} = \alpha + \gamma D$, where $\alpha = 2.05\, ms$ is the startup time including both T_{index} and T_{req} and $\gamma = 1.10$ is the data transferring rate (ms/Kbytes) and D is the size in Kbytes of the document transferred between a browser and the proxy. Considering 8 Kbytes as the average size of a Web document, we obtain $T_{overhead} = 10.85\, ms$ from the model, which is very close to the measurement result.

There are also other types of unique operations in the browsers-aware proxy. For example, the user daemon in each browser periodically sends the updated browser content information to the proxy and the proxy updates its index file accordingly. However, these operations are not in the critical path of the browsers-aware caching system and can be done when the browser, proxy, and networks are not in a heavy demand.

One important question we want to ask is how much latency time we can reduce with the support of the browsers-aware service. Without such a service, a proxy miss will consequently cause a request to a Web server and a data delivering from the server to the proxy. The average static HTML service time from a Web server is over 50 ms without considering the network congestion [42]. In contrast, our measurements show that the browsers-aware service can reduce this time to 10.85 ms, a reduction of more than 78 percent, if the document exists in one of the clients.

6.3.2 Additional Space Allocation

The additional space of *browsers-aware* is allocated for the two data structures keeping track of reference counts to manage data placement, and the browser index file.

First, linked lists are used in the proxy to count the number of accesses to the same documents from different requesting browsers. The size of this space requirement depends on TH_BROWSER and the number of clients to access this document. The value of TH_BROWSER reflects the trade off between the amount of document duplications and intranetwork communication overhead. Our simulation results show that an optimal range of TH_BROWSER is 3 to 5. We use 5 to estimate the space requirement. Our simulation results also show that the average number of clients to access one document is less than 6. For each element in the list, we use 2 bytes for $LL.Client\_ID$, 1 byte for $LL.Access\_Count$, and 5 bytes for $LL.Pointer$ (see Section 3.2 for these three variables in the data structure). The 2 bytes can record up to 65,536 different clients. The 1 byte can represent up to 256 accesses which is much larger than the optimal TH_BROWSER we used. The 5 bytes could represent up to 1,024 G address space. We assume that the proxy has a 32 GByte cache, and an average document size is 8 KByte. The proxy has about 4 M Web pages. The proxy needs to allocate $(32GB/8KB) \times (2 + 1 + 5) \times 6 = 192$ MBytes for the linked lists, which only occupies 0.59 percent of the proxy cache and can be easily placed in the main memory of a proxy server.

Second, a counter and a structure array are allocated for each cached document that has been requested by other clients. The array size is TH_PROXY. The value of TH_PROXY also reflects the trade off between the amount of document duplications and intranetwork communication overhead. Our simulation results show that an optimal range of TH_PROXY is 3 to 7. We use 7 in our calculation, which will overestimate the space requirement. For each element of the array, we allocate 2 bytes for $AC.Client\_ID$, and 1 byte for $AC.Access\_Count$. (See Section 3.2 for the two variables in the data structure.) One byte is also enough for the counter because we use TH_PROXY = 7 here. We assume that each client has a large browser cache with a 80 MByte cache, and an average document size is 8 KByte. Each browser has about 10 K Web pages. The browser needs to allocate about $(80MB/8KB) \times (7 \times (2 + 1) + 1) = 220$ KBytes, which only occupies 0.27 percent of a browser cache. This requirement is overestimated because the array and the counter are allocated to a document only if this document is accessed by other clients.

Regarding the browser index file, we use the MD5 digest for each URL item, which is 16 bytes per URL. Considering an average browser cache of 80 MBytes containing 10,000 documents (8 KBytes for each), we need 160 KBytes space in the index file for each browser. If a browsers-aware proxy system manages 1,000 clients, we need 160 Mbytes space for the index file, which represents a very small portion of the space in a standard proxy. For example, for a 40 GByte proxy, the index file only occupies about 0.4 percent of the proxy space. We can also take advantage of a Bloom filter, which is used to keep URL indices of cooperative caches in [14], where each proxy builds a Bloom filter based on the list of URLs of the cached files. A bit array in each proxy is used to record the changes of cached files. The proxies timely exchange the bit arrays among themselves. Assume that there are still 1,000 clients connected to one proxy. Each client has a browser which has a 8MB cache. Similar to [14], we also assume that an average document size is 8KB. Each browser has about 1K Web pages. The Bloom filter can use 2KB to represent 1K pages of each browser with a certain probability of false positive. In this case, the proxy needs about 2000KB = 2MB to store the whole browser index file. The structure of the bloom filter is indeed space-efficient for index information sharing among many parties, but may cause large computation overhead and high probability of false positive. There is a trade off between the probability of a false positive and the number of bits used for each entry. Less probability of false positive means more storage requirement. We think that, for the current scale of proxy-browser system, a simple index structure presented in the paper without using the bloom filter is both performance and space-efficient for the browsers-aware scheme without causing false positive. The bloom filter can be used in a proxy-browser system with one proxy connecting with tens of thousands clients. In this case, the space requirement for the browser index file really becomes an issue.

6.3.3 CPU Overhead

In a browser, the additional CPU overhead comes from searching structure arrays. The size of each array is TH_PROXY. As we mentioned previously, an optimal range of TH_PROXY is 3 to 7. So, handling such a search for each request from a remote client requires $O(1)$ time.

In the proxy, the additional CPU overhead comes from searching a linked list for a hit request. The CPU time requirement for handling such a search for a document from a remote client depends on the number of clients that have requested the document. As we mentioned previously, the average number of clients to access one document is less than 6. Thus, handling such a search requires $O(1)$ time in average. But, it is possible that there is a long list for one document. The following strategies have been applied to alleviate this possible delay. First, an element for a client will be deleted from the list when the document has been requested as many times as TH_BROWSER because the client has been informed to cache this document. Second, the list search can be overlapped with passing the document to a client. In detail, when a client request hits in the proxy, the proxy first sends the requested document to the client. The client will spend some time viewing the document. At the same time, the proxy searches for the list of this document to check how many times this client has requested this document. Afterward, the proxy will inform the client whether to cache this document or not, depending on the searching result. The searching process will not delay response times to the clients.

7 Related Work

Before Web caching research was active, authors in [11] attempted to improve network file system performance by coordinating the contents client caches and allowing requests not satistifed by a client's local in-memory file cache to be satisfied by the cache of another client.

Client access patterns are characterized by several research groups (see e.g., [3], [12], [19], and [34]). Kelly [23] presents a method called cache-busting technique for tracing browser requests, which does not require client modifications, but only minor proxy modifications.

Numerous studies focus on local caching replacement policies. For example, [2] and [21] provide theoretical bases for approximate optimal performance and designing effective online algorithms. Papers [8], [10], and [20] propose practical caching replacement strategies and showed promising experimental performance results. However, cooperative caching can significantly improve performance compared to local replacement [24] and has been studied in both the horizontal and vertical directions.

In the horizontal direction, cooperative proxy caches are studied in many papers (e.g., [14], [17], [27], [35], [40]), which focus on the proxies at the same level. These papers provided different ways of attempting to effectively sharing files among same level proxies, such as how to locate a file cached in another cache precisely and quickly, and how to place a file as close as possible to a proxy requesting the file with highest probability. Cooperating caches of browsers and their proxy have not been studied in previous work. We believe there are two reasons: 1) A browser cache was initially developed as a small data buffer with a few simple data manipulation operations. Users were not supposed to be able to retain the cached data with a high quality of spatial and temporal locality. With a significant increase of memory and disk capacity in workstations and PCs and with the improvement of Web browser caching capability, users are able to enlarge the browser cache size in order to access more cached documents and to retain the documents in an organized manner for a longer period of time. 2) Data integrity and user privacy are concerns in browser sharing. Existing security technologies can be further improved to ensure the integrity and anonymity among browsers. Thus, cooperating caches of browsers and their proxy are another practical way to improve caching performance. None of the previous studies consider file duplications among same level proxies. A practical reason for allowing file duplications among proxies is because proxies are normally far from each other in locations. Emphasizing eliminating file duplications too highly could cause too many requests to remote proxies so that the overall response time might be hurt. However, browsers connecting to the same proxy are usually located nearby, thus, reducing file duplications among browsers enables more files to be shared to improve overall performance.

In the vertical direction, Web proxy workloads from different levels of a caching hierarchy are studied in [26]. Korupolu et al. [25] develop an optimal algorithm for hierarchical placement problem. Korupolu and Dahlin [24] and Tewari et al. [33] propose practical schemes to cooperate hierarchical proxies by hierarchical GreedyDual replacement algorithm and placement algorithm that cache files close to clients. They conclude that hierarchical cooperative caching can significantly improve performance. The study in [13] is not so optimistic about hierarchical cooperative caching and concludes that the performance in terms of response time cannot be improved without paying careful attention to details of cooperation design to eliminate overhead, such as better distributing network traffic and avoiding congested routes. Two previous studies attempt to reduce file duplications in hierarchical cooperative caching. Che et al. [9] propose a hierarchical cooperative caching architecture to avoid a requested file cached in each intermediate cache. A cache is viewed as a filter with its cutoff frequency equal to the inverse of the characteristic time. Files with access frequencies lower than this cutoff frequency have a good chance of passing through the cache without cache hits. A collaborative method is proposed in [41] for hierarchical caching in proxy servers to reduce duplicate caching between a proxy and its parent or higher-level proxies in the hierarchy. In particular, a collaboration protocol passes caching decision information along with the document to the next lower-level proxy to facilitate its caching decision. Our work focuses on a proxy-browser system, which is a different issue of reducing duplication in the different-level proxies. Our proposed scheme not only reduces the duplications between different-level caches (between proxy and browsers), but also reduces the duplications at the same-level caches (among browsers). Recently, a pure P2P Internet caching framework, Squirrel, has been proposed [22]. In this decentralized environment, proxy does not exist and document sharing is solely conducted among client browsers. We believe the client/server model will always coexist with pure P2P model and a hybrid P2P model, such as our enhanced browsers-aware caching system, can combine the merits of both centralized and decentralized systems.

8 Conclusion

We have demonstrated trends of decreasing proxy hit ratios and increasing access diversity and significant duplications in existing Web caching systems. In order to effectively utilize additional caching space and available bandwidths among browsers, we propose a peer-to-peer Web caching management scheme: *browsers-aware caching*. We show that the performance of our scheme compares very favorably with the performance of near-optimal offline Web caching algorithms. Our prototype implementation further shows the effectiveness of the proposed scheme and addresses the issues of data integrity and communication anonymity for browsers-aware caching systems. This work can also be extended for a more complex relationship between multiple proxies and clients, such as the structure presented in [36].

Acknowledgment

The authors would like to thank the anonymous referees for their critical and constructive comments on this paper. They would also like to thank Bill Bynum for reading the paper and his suggestions. This work is supported in part by the US National Science Foundation under grants CCR-9812187, EIA-9977030, and CCR-0098055, and by a USENIX Research Scholarship.

References

[1] M. Abrams, C.R. Standridge, G. Abdulla, S. Williams, and E.A. Fox, "Caching Proxies: Limitations and Potentials," *Proc. Fourth Int'l World Wide Web Conf.*, Dec. 1995.

[2] S. Albers, S. Arora, and S. Khanna, "Page Replacement for General Caching Problems," *Proc. 10th Ann. ACM-SIAM Symp. Discrete Algorithms (SODA '99)*, pp. 31-40, 1999.

[3] P. Barford, A. Bestavros, A. Bradley, and M. Crovella, "Changes in Web Client Access Patterns: Characteristics and Caching Implications," *World Wide Web J.*, vol. 2, no. 1, pp. 15-28, Jan. 1999.

[4] L.A. Belady, "A Study of Replacement Algorithms for Virtual Storage Computers," *IBM Systems J.*, vol. 5, pp. 78-101, 1966.

[5] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web Caching and Zipf-Like Distributions: Evidence and Implications," *Proc. IEEE INFOCOM*, 1999.

[6] Boeing log files, ftp://researchsmp2.cc.vt.edu/pub/boeing/, 2003.

[7] BU traces, ftp://cs-ftp.bu.edu/techreports/1995-010-www.client-traces.tar.gz and ftp://cs-ftp.bu.edu/techreports/1999-011-user trace-98.gz, 2003.

[8] P. Cao and S. Irani, "Cost-Aware WWW Proxy Caching Algorithms," *Proc. USENIX Symp. Internet Technologies and Systems*, Dec. 1997.

[9] H. Che, Z. Wang, and Y. Tung, "Analysis and Design of Hierarchical Web Caching Systems," *Proc. IEEE INFOCOM 2001*, Apr. 2001.

[10] E. Cohen and H. Kaplan, "LP-Based Analysis of Greedy-Dual-Size," *Proc. 10th Ann. ACM-SIAM Symp. Discrete Algorithms*, pp. 879-880, Jan. 1999.

[11] M.D. Dahlin, R.Y. Wang, T.E. Anderson, and D.A. Patterson, "Cooperative Caching: Using Remote Client Memory to Improve File System Performance," *Proc. First Symp. Operating Systems Design and Implementation*, Nov. 1994.

[12] B.M. Duska, D. Marwood, and M.J. Feeley, "The Measured Access Characteristics of World-Wide-Web Client Proxy Caches," *Proc. USENIX Symp. Internet Technologies and Systems*, Dec. 1997.

[13] S.G. Dykes and K.A. Robbins, "A Viability Analysis of Cooperative Proxy Caching," *Proc. IEEE INFOCOM 2001*, Apr. 2001.

[14] L. Fan, P. Cao, J. Almeida, and A.Z. Broder, "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol," *Proc. 1998 SIGCOMM Conf.*, pp. 254-265, 1998.

[15] E. Gabber, P. Gibbons, D. Kristol, Y. Matias, and A. Mayer, "Consistent, Yet Anonymous, Web Access with LPWA," *Comm. ACM*, vol. 42, no. 2, pp. 42-47, Feb. 1999.

[16] E. Gabber, P. Gibbons, Y. Matias, and A. Mayer, "How to Make Personalized Web Browsing Simple, Secure, and Anonymous," *Proc. Conf. Financial Cryptography*, 1997.

[17] S. Gadde, M. Rabinovich, and J. Chase, "Reduce, Reuse, Recycle: An Approach to Building Large Internet Caches," *Proc. Sixth Workshop Hot Topics in Operating Systems*, May 1997.

[18] L. Gong, "JXTA: A Network Programming Environment," *IEEE Internet Computing*, vol. 5, no. 3, May/June 2001.

[19] S.D. Gribble and E.A. Brewer, "System Design Issues for Internet Middleware Services: Deductions from a Large Client Trace," *Proc. 1997 Usenix Symp. Internet Technologies and Systems*, Dec. 1997.

[20] S. Jin and A. Bestavros, "Popularity-Aware GreedyDual-Size Web Proxy Caching Algorithms," *Proc. 20th Int'l Conf. Distributed Computing Systems (ICDCS '00)*, Apr. 2000.

[21] S. Irani, "Page Replacement with Multi-Size Pages and Applications to Web Caching," *Proc. 29th Ann. ACM Symp. Theory of Computing (STOC '97)*, pp. 701-710, 1997.

[22] S. Iyer, A. Rowstron, and P. Druschel, "Squirrel: A Decentralized Peer-to-Peer Web Caching," *Proc. 21st ACM Symp. Principles of Distributed Computing*, 2002.

[23] T. Kelly, "Thin-Client Web Access Patterns: Measurements from a Cache-Busting Proxy," *Computer Comm.* vol. 25, pp. 357-366, 2002.

[24] M.R. Korupolu and M. Dahlin, "Coordinated Placement and Replacement for Large-Scale Distributed Cached," *IEEE Trans. Knowledge and Data Eng.*, vol. 13, 2001.

[25] M.R. Korupolu, C.G. Plaxton, and R. Rajaraman, "Placement Algorithms for Hierarchical Cooperative Caching," *Proc. 10th Ann. ACM-SIAM Symp. Discrete Algorithms (SODA '99)*, pp. 586-595, Jan. 1999.

[26] A. Mahanti, C. Williamson, and D. Eager, "Traffic Analysis of a Web Proxy Caching Hierarchy," *IEEE Network*, special issue on Web performance, vol. 14, no. 3, pp. 16-23, May/June 2000.

[27] R. Malpani, J. Lorch, and D. Berger, "Making World Wide Web Caching Servers Cooperate," *Proc. Fourth Int'l World Wide Web Conf.*, Dec. 1995.

[28] A.J. Menezes, P.C. Van Oorschot, and S.A. Vanstone, *Handbook of Applied Cryptography.* CRC Press, 1996.

[29] Nat'l Lab Applied Network Research, http://www.ircache.net/, Sanitized access logs: ftp://ircache.nlanr.net/Traces/, and Statistics: http://www.ircache.net/Cache/Statistics/, 2003.

[30] K. Psounis and B. Prabhakar, "A Randomized Web-Cache Replacement Scheme," *Proc. IEEE INFOCOM 2001*, Apr. 2001.

[31] R. Rivest, "The MD5 Message-Digest Algorithm," *Internet RFC/STD/FYI/BCP Archives,* request for comments: 1321, (http://www.faqs.org/rfcs/rfc1321.html), Apr. 1992.

[32] A. Rousskov and V. Soloviev, "A Performance Study of the Squid Proxy on HTTP/1.0," *World Wide Web,* vol. 2, nos. 1-2, pp. 47-67, Jan. 1999. Also available at "On Performance of Caching Proxies," *Proc. SIGMETRICS '98,* pp. 272-273, 1998.

[33] R. Tewari, M. Dahlin, H.M. Vin, and J.S. Kay, "Design Considerations for Distributed Caching on the Internet," *Proc. 19th IEEE Int'l Conf. Distributed Computing Systems (ICDCS),* May 1999.

[34] A. Wolman, G. Voelker, N. Sharma, N. Cardwell, M. Brown, T. Landray, D. Pinnel, A. Karlin, and H. Levy, "Organization-Based Analysis of Web-Object Sharing and Caching," *Proc. Second USENIX Symp. Internet Technologies and Systems,* Oct. 1999.

[35] A. Wolman, G. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. Levy, "On the Scale and Performance of Cooperative Web Proxy Caching," *Proc. 17th ACM Symp. Operating System Principles (SOSP),* pp. 16-31, Dec. 1999.

[36] Z. Xiao and K.P. Birman, "Providing Efficient, Robust Error Recovery through Randomization," *Proc. Int'l Workshop Applied Reliable Group Comm.,* (jointly held with the *21st Int'l Conf. Distributed Computing Systems*), Apr. 2001.

[37] L. Xiao and X. Zhang, "Exploiting Neglected Data Locality in Browsers," *Proc. 10th Int'l World Wide Web Conf. (WWW10),* May 2001. (an extended abstract)

[38] L. Xiao, Z. Xu, and X. Zhang, "Low Cost and Reliable Mutual Anonymity Protocols in Peer-to-Peer Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 14, no. 9, pp. 829-840, Sept. 2003.

[39] L. Xiao, X. Zhang, and Z. Xu, "A Reliable and Scalable Peer-to-Peer Web Document Sharing System," *Proc. 2002 Int'l Parallel and Distributed Processing Symp., (IPDPS '2002),* 2002.

[40] J. Yang, W. Wang, and R. Muntz, "Collaborative Web Caching Based on Proxy Affinities," *Proc. ACM SIGMETRICS 2000,* pp. 78-89, June 2000.

[41] P.S. Yu and E.A. MacNair, "Performance Study of a Collaborative Method for Hierarchical Caching in Proxy Servers," *Proc. Seventh Int'l World Wide Web Conf.,* Apr. 1998.

[42] H. Zhu and T. Yang, "Class-Based Cache Management for Dynamic Web Content," *Proc. IEEE INFOCOM 2001,* Apr. 2001.

Li Xiao received the BS and MS degrees in computer science from Northwestern Polytechnic University, China, and the PhD degree in computer science from the College of William and Mary in 2002. She is an assistant professor of computer science and engineering at Michigan State University. She is a recipient of the USENIX Fellowship for her PhD 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 the ACM and the IEEE.

Xiaodong Zhang received the BS degree in electrical engineering from Beijing Polytechnic University in 1982 and the MS and PhD degrees in computer science from the University of Colorado at Boulder in 1985 and 1989, respectively. He is the Lettie Pate Evans Professor of Computer Science and the department chair at the College of William and Mary. He was the program director of advanced computational research at the US National Science Foundation from 2001 to 2003. He is a past member of the editorial board of the *IEEE Transactions on Parallel and Distributed Systems* and currently serves as an associate editor of *IEEE Micro*. His research interests are in the areas of parallel and distributed computing and systems and computer architecture. He is a senior member of the IEEE.

Artur Andrzejak received the PhD degree in computer science from the Swiss Federal Institute of Technology (ETH Zurich) in 2000. He is currently a researcher at Zuse-Institute Berlin, Germany. He was a postdoctoral researcher at the Hewlett-Packard Labs in Palo Alto, California, from 2001 to 2002. His research interests include utility computing, systems management, and P2P computing.

Songqing Chen received the BS and MS degrees in computer science from Huazhong University of Science and Technology, China, in 1997 and 1999, respectively. He is a PhD candidate in computer science at the College of William and Mary. His research interests are distributed and Internet systems and kernel systems programming. He is a student member of the IEEE and the ACM.