An Expiration Age-Based Document Placement Scheme for Cooperative Web Caching

Lakshmish Ramaswamy, *Student Member*, *IEEE*, and Ling Liu, *Member*, *IEEE Computer Society*

Abstract—The sharing of caches among proxies is an important technique to reduce Web traffic, alleviate network bottlenecks, and improve response time of document requests. Most existing work on cooperative caching has been focused on serving misses collaboratively. Very few have studied the effect of cooperation on document placement schemes and its potential enhancements on cache hit ratio and latency reduction. In this paper, we propose a new document placement scheme which takes into account the contentions at individual caches in order to limit the replication of documents within a cache group and increase document hit ratio. The main idea of this new scheme is to view the aggregate disk space of the cache group as a global resource of the group and uses the concept of cache expiration age to measure the contention of individual caches. The decision of whether to cache a document at a proxy is made collectively among the caches that already have a copy of this document. We refer to this new document placement scheme as the Expiration Age-based scheme (EA scheme for short). The EA scheme effectively reduces the replication of documents across the cache group, while ensuring that a copy of the document always resides in a cache where it is likely to stay for the longest time. We report our study on the potentials and limits of the EA scheme using both analytic modeling and trace-based simulation. The analytical model compares and contrasts the existing (ad hoc) placement scheme of cooperative proxy caches with our new EA scheme and indicates that the EA scheme improves the effectiveness of aggregate disk usage, thereby increasing the average time duration for which documents stay in the cache. The trace-based simulations show that the EA scheme yields higher hit rates and better response times compared to the existing document placement schemes used in most of the caching proxies.

Index Terms—Cooperative Web caching, document placement, distributed caching.

✦

1 INTRODUCTION

THE popularity of the Internet and World Wide Web (Web) continues to grow. So does the number of users accessing information on the Web. This leads to continued increase of both network load and server load. One way to meet this challenge is to try to scale network and server bandwidth to keep up with the client demand, which is an expensive strategy. An alternative is caching, which reduces network bandwidth and server load by migrating server files closer to those clients that use the files. Caching can be done either at a client (by Web browser) or in the network between clients and content servers (by proxy servers).

Cooperative caching—the sharing and coordination of cache state among multiple caching proxies—has been recognized as one of the most important techniques to reduce Web traffic and alleviate network bottlenecks. Web cache sharing was first introduced in Harvest [4] to gain the full benefits of caching. The main idea is to allow a proxy cache to locate documents that are not available locally from other "nearby" caches before contacting the origin (content) server. Cooperative caching can be seen as a process of Web cache sharing where proxy caches on the same side of a common bottleneck link cooperate and serve each others misses. In this paper, we refer to those cooperating proxy caches collectively as a cache group.

The advantages of cooperative caching are apparent. First, it reduces the bandwidth contention on the backbone network. Second, it reduces the actual traffic on the origin servers. The third and most important implication of cooperation among caches is a reduction in the average latency experienced by the clients of these caches.

It is known that the benefits of cooperative caching are bounded by the ratio of intercache communication time to server fetch time. Researchers have studied cache-sharing protocols, which provide mechanisms to reduce the communication cost among cooperating caches. Internet Cache Protocol (ICP) [6] was designed specifically for communication among Web caches. ICP is a lightweight protocol and is implemented on top of UDP. The protocol consists of two types of messages that are exchanged between neighboring caches, namely, ICP queries and ICP replies. ICP query is a message sent by a cache that experienced a local miss to all its neighboring caches asking whether they have the particular document. ICP reply is a message from the caches receiving the ICP query to the query originator which communicates whether they have the particular document cached in them.

Although several cooperative caching protocols have been proposed [4], [5], [10], [16], [7], [12], [20], few studies have examined the cooperation of caches from document-placement point of view. To our knowledge, none has so far tried to answer the following questions: Can we devise a document placement scheme that utilizes the sharing and coordination of cache state among multiple communicating caches? Can such a document placement scheme improve hit ratios and reduce document-access latency? What are the potential advantages and drawbacks of such a scheme,

● *The authors are with the College of Computing, Georgia Institute of Technology, 801 Atlantic Drive, Atlanta, GA 30332.*
E-mail: {laks, lingliu}@cc.gatech.edu.

especially how does the document placement scheme relate to the ratio of the interproxy communication time to server fetch time?

Let us first review how documents are placed in the present cooperative caching protocols and what the potential problems of the existing schemes are. The document placement policies in either hierarchical or distributed caching architecture share a common scheme: When an ad hoc document request is a miss in the local cache, this document is either served by another "nearby" cache in the cache group or by the origin server. In either case, this document is added into the proxy cache where it was requested. Therefore, a document may be cached in several or all of the caches in the cache group if the document is requested at several or all of the proxy caches. We refer to this conventional scheme as the *ad hoc scheme*. An obvious drawback of the ad hoc document placement scheme is the fact that the caches within a group are treated as completely independent entities when making document placement decisions. This may result in "uncontrolled" replication of documents. Such "uncontrolled" replication of data reduces the efficiency of the aggregate disk space utilization of the group of caches. By efficiency, we mean the number of unique (nonreplicated) documents present in the cache. This reduction in the number of unique documents available in the cache group often leads to a reduction in the cumulative hit rate of the cache group.

In this paper, we propose a simple and yet effective scheme to limit the replication of documents within a group. We view the aggregate disk space of the cache group as a global resource of the cache group and introduce the concept of cache expiration age to measure the disk space contention of caches. The new scheme is based on the expiration ages of individual caches in the group, referred to as the Expiration Age-based scheme (EA scheme for short). The EA scheme effectively reduces the replication of documents across the cache group, while ensuring that a copy of the document always resides in a cache where it is likely to stay for the longest time. Further, the implementation does not involve any extra communication overheads when compared with the ad hoc scheme used in many existing cooperative-caching protocols. We use both analytic modeling and trace-based simulations to show the potential advantages and drawbacks of the proposed scheme. Both analytic modeling and trace-based analysis show that the EA scheme yields higher hit rates and better response times.

2   PROBLEM STATEMENT

As briefly mentioned in the Introduction, all existing cooperative caching protocols treat individual caches in a cooperation group as completely independent entities when making decisions on where to cache a document. More concretely, the decision of whether to place a particular document in a given cache is made without knowledge about the other caches. This blind caching can lead to "uncontrolled" replication of documents in the cache group. It may potentially reduce the hit rate of the group as a whole entity. To understand various factors that may influence the quality and efficiency of document placement in a group of caches, in this section, we walk through a simple scenario to illustrate the potential problems with the existing document placement scheme, which we call the *ad hoc* scheme.

Consider a cache group with three caches, say, C_1, C_2, and C_3, in it. Let us consider the distributed caching architecture as the cache cooperation structure in this example, although our arguments and our new document placement scheme are independent of specific cooperative cache architectures and work well with various document replacement algorithms.

Consider a scenario when the cache C_1 experiences a local miss when serving a client request for a document D. C_1 sends an ICP query to both C_2 and C_3. If the document is not available in both C_2 and C_3, C_1 fetches it from the origin server, stores a local copy, and serves the client. If after some time, C_2 gets a client request for the same document D, it sends an ICP query to C_1 and C_3, C_1 replies positively. C_2 now fetches the document from C_1, stores it locally, and serves the client request. Now, it is evident that the document D is present in both C_1 and C_2. Furthermore, as the document is fetched from C_1, it is considered to be a hit in the cache C_1. Therefore, the document in the cache C_1 is given a fresh lease of life. If C_3 gets a client request for the same document D, it can fetch it from either C_1 or C_2. So, the document D is now replicated at all the three caches. This example illustrates how the ad hoc schemes can lead to "uncontrolled" replication of data.

This uncontrolled replication of data affects the efficiency of the usage of aggregate disk space available in the group. The reduction in the effective disk space availability in the cache group increases the contention for disk space at the individual caches. Increased contention for disk space manifests itself in two ways: 1) The time for which individual documents live in the cache is reduced. 2) The number of unique documents available in the cache group decreases. It is well-known that the hit rate is proportional to the number of unique documents available.

The cumulative effect of the above two leads to a fall in the aggregate hit rate of the cache group when compared with a cache group having the same amount of disk space and with no replication or controlled replication of documents. Therefore, the usage of cumulative disk space in the cache group is not optimal in the existing (ad hoc) document placement scheme.

3   THE EXPIRATION-AGE-BASED DOCUMENT PLACEMENT SCHEME

In this section, we present our new document placement scheme for cooperative caching. In this scheme, each proxy makes intelligent decisions on whether to cache a particular document. The decision is based on a number of factors: whether the document is already available in the cache group; if so, how long is it likely to remain in the caches where it is currently stored; and the disk space contention of the proxies that contain the document. We use the Expiration Age of the caches to measure their cache contention levels. In the next section, we explain the concept of *Expiration Age* and how it can be used to make informed decisions.

3.1 Expiration Age—A Measure of Cache Contention

An important question related to document placement is to understand what type of measures can be used to compare disk space contention in a group of caches. One possible approach to measure the contention at a cache in a given time duration is to use the average document lifetime of a cache in that duration. This can be estimated in terms of the lifetime of documents that are evicted in the given duration. We argue that this approach is inaccurate.

Consider that a document D entered a cache at time T_0 and was removed at time T_R. The *Life Time* of document D is defined as $(T_R - T_0)$. The *Average Document Life Time* of the cache is defined as the mean of *Life Times* of documents that were removed in some finite time period. It can be seen that the *Average Document Life Time* of a cache depends on the disk space contention in the cache. If the disk space contention were higher, *Average Document Life Time* would be lower and vice versa.

While *Average Document Life Time* of a cache depends on the disk space contention, it does not accurately reflect the cache contention. A simple example illustrates this fact. Consider two caches C_1 and C_2 of equal size with identical initial configurations (containing the same set of documents cached at the same time). Let both these caches use the LRU replacement policy. Now, consider a set of client requests, say, $\{Q_0, Q_1, \ldots, Q_n\}$. Let these requests be for documents that are not present in the caches C_1 and C_2. Let us assume that no two requests are for the same document. Consider a situation where both C_1 and C_2 receive this set of client requests. In addition, let C_1 receive an additional request for a document D already present in the cache every second for the first N seconds. Therefore, there is a hit for the document D every second for the first N seconds. Now, consider that after N_R seconds, the document D was removed. Document D would have accumulated a large *Life Time* (because every hit in the first N seconds gave it a new lease on life). This contributes significantly to the *Average Document Life Time* of cache C_1 making it substantially higher than the *Average Document Life Time* of C_2. Hence, it is clearly seen that, although the contention for disk space is the same in both caches C_1 and C_2, C_1 has a significantly higher *Average Document Life Time*.

The above example shows that the *Average Document Life Time* does not truly reflect the disk space contention of individual caches. It also indicates that any measure for disk space contention should account for document hits and misses.

Our definition of the expiration age of a cache is influenced by the following two observations: When the cache contention is high, we observe two facts. First, the number of documents removed from the cache in a given period is higher. Second, and more importantly, even those documents that were hit recently or that were hit frequently are removed in a shorter period of time. We introduce the concept of *Expiration Age* of a document to indicate how long a document is expected to live in a cache since its last hit and how long on an average a document gets a hit at the cache.

3.2 Expiration Age of a Cached Document

Many proxy caches maintain either the last hit timestamp or the total number of hits for each document cached. For example, most of today's caching proxies employ the Least Recently Used (LRU) algorithm or the Least Frequently Used (LFU) algorithm or some variant of LRU or LFU as their cache replacement policy [18]. Proxy caches that employ LRU or one of its variants need to maintain the last hit timestamp for each document they cache. Similarly, proxy caches that choose to use LFU or one of its variants need to maintain the total number of hits experienced by every cached document.

In order to make the expiration age-based document placement scheme applicable to any existing proxy cache with almost no extra cost, we present two representative ways to define the *Expiration Age* of a cached document. The choice can be made based on whether a cache maintains the last hit timestamp or the total number of hits for each document it caches.

With these design ideas in mind, we define the *Expiration Age* of a document in a cache, denoted as $DocExpAge(\text{D}, \text{C})$, by a formula that combines the document expiration age computed based on the last hit timestamp of every cached document and the document expiration age computed when only the total number of hits experienced by every cached document is available in a cache. For presentation convenience, in the rest of the paper we use LRU expiration age, denoted as $DocExpAge_{LRU}(D, C)$, to represent the document expiration age computed based on the last hit timestamp of the cached document D in the cache C, since LRU requires proxy caches to maintain the last hit timestamp of every document they cache. Similarly, we use LFU expiration age, denoted as $DocExpAge_{LFU}(D, C)$, to represent the document expiration age computed when only the total number of hits experienced by the document D in the cache C is available since LFU requires proxy caches to maintain the total number of hits of every cached document:

$$DocExpAge(D, C) =$$
$$\begin{cases} DocExpAge_{LRU}(D, C) & \text{if cache C maintains the lasthit} \\ & \text{timestamp of D} \\ DocExPAge_{LFU}(D, C) & \text{if cache C maintains the total} \\ & \text{number of hits of D.} \end{cases}$$

(1)

$DocExpAge_{LRU}$ and $DocExpAge_{LFU}$ are defined as follows.

3.2.1 LRU Expiration Age

The LRU Expiration Age of a document is defined as the time duration from the time it was last hit in a cache to the time when it was evicted from the cache. Suppose a document D was removed at time T_R and the last hit on the document occurred at time T_l, then the LRU Expiration Age of document D, denoted by $DocExpAge_{LRU}(\text{D}, \text{C})$, is defined as

$$DocExpAge_{LRU}(D, C) = (T_R - T_l). \qquad (2)$$

The LRU expiration age of a document indicates how long a document can be expected to live in a cache after its last hit.

LRU is one of the most well-studied and well-documented document replacement policy for Web caching. In this policy, whenever a document has to be removed (in order to make room for incoming documents), the document that hasn't experienced a hit for the longest time is selected as the victim. Thus, caching proxies that use LRU or its variants will maintain the time of last hit for each document in the cache. Hence, it is straightforward for any caching proxy that uses LRU cache replacement to compute the expiration ages of the documents in the cache.

3.2.2 LFU Expiration Age

The Least Frequently Used (LFU) scheme is another well-studied cache replacement policy, although it is not supported in most of the proxy caches operational today. In the LFU scheme, whenever a document has to be evicted, the document with the least frequency of hits is selected as the victim. To implement this scheme, a HIT-COUNTER is maintained along with every document. This HIT-COUNTER is initialized to one when the document enters the cache. Every time a document is hit in the cache, its HIT-COUNTER is incremented by one. This HIT-COUNTER is used for calculating hit frequency.

Let us consider how to compute the Expiration Age for caches employing LFU replacement policy. Clearly, any caching proxy that employs LFU cache replacement policy will maintain at least two pieces of information for each cached document: the time when the document entered the cache and its hit counter (how many times the document was hit before it is evicted from the cache). We can then use the ratio of the total time a document lived in a cache to its HIT-COUNTER value to estimate the expiration age of a document. This ratio indicates how long on average the document D can get a hit in the duration of its life in the cache C. It can be used as a good indicator of the time that the document D is expected to live in C since its last hit.

Consider a document D that entered the cache C at time T_0. Suppose that this document was removed at time instant T_R. The Expiration Age of the document D under LFU replacement policy, denoted as $DocExpAge_{LFU}(D, C)$, can be estimated by the ratio of the total time it lived in the cache to its HIT-COUNTER value.

$$DocExpAge_{LFU}(D, C) = \frac{(T_R - T_0)}{HIT - COUNTER}. \quad (3)$$

The LFU expiration age of a document indicates the average time it takes for a document to get a hit. It is a good approximation of how long a document is expected to live after its last hit.

3.2.3 Discussion

We have discussed two representative definitions of document Expiration Ages. We end this section by describing how the Expiration Age can be used for a cache employing a two-level cache replacement algorithm, namely, an LRU variant or an LFU variant.

Consider the LRU-MIN cache replacement algorithm [11], [19]. It works as follows: Suppose the incoming document is of size S bytes, then the scheme prepares a list of all documents whose sizes are of at least size S and

applies LRU to this list and removes the least recently used document from this list. If the list is empty, then it reverts back to the original LRU policy. To understand the effects of size of documents in this removal policy, let us assume that the document sizes are uniformly distributed between 0 to $DOCSIZE_{MAX}$ without loss of generality. Now, suppose that documents A and B have sizes S_A and S_B and $S_A > S_B$. If these two documents experienced their most recent hit at almost the same time (i.e., $T_l(A)$ is almost the same as $T_l(B)$, where $T_l(A)$ and $T_l(B)$ denote the last hit time stamp of document A and document B, respectively), then the chances of removal of A or B is directly dependent on its size. This is because the size of the incoming document is also a uniform random variable and, hence, chances of A or B being kept in the LRU-MIN list is proportional to its size. Hence, if a document D, of size S_D, experienced a hit at time T_l and was removed at time T_R, then Document Expiration Age of D under the LRU-MIN policy is given by

$$DocExpAge_{LRU-MIN}(D, C) = (T_l(D) - T_H(D)) \times S_D. \quad (4)$$

Thanks to the fact that almost all operational proxy caches use LRU or one of its variants as their cache replacement policy, in this paper, all our experiments employ the LRU document replacement scheme. Hence, we use LRU Expiration Age as the disk space contention measure. In the rest of the paper, we use the terms LRU Expiration Age and Expiration Age interchangeably.

3.3 Expiration Age of a Cache

The expiration age of a cache is intended to be a measure of the disk space contention at the cache. Recalling the discussion in Section 3.1, we can easily conclude that the average of the expiration ages of those documents that were removed from a cache indicates the level of the disk-space contention at that cache. Thus, we define the expiration age of a cache in a finite time period by the *average of the expiration ages* of the documents that were removed from this cache in this period.

Formally, let $Victim(C, T_i, T_j)$ denote the set of documents that were chosen as victims for removal in a finite time duration (T_i, T_j). The cardinality of the set $Victim(C, T_i, T_j)$ indicates the total number of documents evicted from a cache C during the duration (T_i, T_j). The Expiration Age of the cache C in the duration (T_i, T_j), denoted by $CacheExpAge(C, T_i, T_j)$, is calculated as:

$$CacheExpAge(C, T_i, T_j) = \frac{\sum_{D \in Victim(C, T_i, T_j)} DocExpAge(D, C)}{|Victim(C, T_i, T_j)|}. \quad (5)$$

The Expiration Age of a cache indicates the average time a document can be expected to live in the cache after its last hit and can be considered as an accurate indicator of the disk space contention at the cache.

3.4 Algorithms for Expiration Age-Based Document Placement

As we discussed in the previous section, the EA scheme uses the Cache Expiration Age as an indicator of the disk space contention at individual caches. In this section, we walk through the EA algorithms and answer the questions

such as how a cache shares its cache expiration age information with others in a group and how they reach decisions on whether to cache a document obtained from another cache in the group.

The EA document placement scheme involves sending the Cache Expiration Ages of the caches along with the HTTP request and reply messages among the caches in the group. A cache that experiences a local miss sends out an ICP query to all its siblings and parents (or peers). This cache is henceforth referred to as the *Requester*. The Requester, on receiving a positive reply from either a parent or a sibling, sends out an HTTP request to that particular cache. Along with the HTTP request message, it appends its Cache Expiration Age. The cache that receives the HTTP request, henceforth referred to as *Responder*, responds back by sending the document to the Requester. Along with the document, the Cache Expiration Age of the Responder is also communicated to the Requester. The Requester now compares its own Cache Expiration Age to that of the Responder. If the Cache Expiration Age of the Responder is greater than that of the Cache Expiration Age of the Requester, the Requester does not store the document locally, it just serves the user with the document. However, if the Cache Expiration Age of the Requester is greater than that of the Responder, it stores a copy of the document locally.

The Responder also compares its own Cache Expiration Age with the Cache Expiration Age of the Requester (that was obtained along with the HTTP request message). If its own Cache Expiration Age is greater than that of the Requester, the entry corresponding to the requested document is promoted to the HEAD of the LRU list. Otherwise, the document entry is left unaltered at its current position.

When the Requester receives negative ICP replies from all caches, we distinguish two situations. In cooperative caches using the distributed caching architecture, the requestor fetches the document from the origin server, caches the document, and serves it to its client. If the caches in the group communicate using hierarchical caching architecture, the requestor sends an HTTP request to one of its parents (assuming the cache has a parent). Along with that request, it attaches its own Cache Expiration Age. It is now the responsibility of the parent to resolve the miss. It retrieves the document from the origin server (possibly through its own parents). Then, it compares the Cache Expiration of the Requester with its own Cache Expiration Age. If the Cache Expiration Age of the parent cache is greater than that of the Requester, it stores a copy of the document before transferring it to the Requester. Otherwise, the document is just served to the Requester and the parent cache does not keep a copy of the document. In either case, the parent's Cache Expiration Age accompanies the document. The Requester acts in the same fashion as in the case where the document was obtained from a Responder (making a local copy if its own Expiration Age is greater than that of the parent).

Concretely, the EA scheme consists of two algorithms: the algorithm for the Requester cache and the algorithm for the Responder cache. We provide a sketch of these two algorithms in Algorithm 1 and 2 in Figs. 1 and 2, respectively.

3.5 Rationale and Features of the EA Scheme

The rationale behind the Expiration-Age-based placement scheme is based on two motivations. The first is to eliminate unnecessary replicas of a document in the cache group. The second is to ensure that the document be replicated only if the new copy has a reasonable chance to survive longer than the "original" copy. By this, we also ensure that the EA scheme never reports a miss for a document when it would have been a hit under the old scheme.

It is straightforward to see that a document is not replicated at a cache if its own Cache Expiration Age is less than that of the Responder's cache from which the document was obtained. This is because the copy of the document at the Responder is likely to survive longer than the new copy. Even if a copy was made locally at the Requestor's cache, it would be removed earlier than the copy at the Responder's cache. Therefore, under the EA placement scheme, the Requester will not store a copy of the requested document locally. This obviously reduces the number of replicas in the cache group.

Now, let us consider the situation when the Cache Expiration Age of the Requester is greater than or equal to the Cache Expiration Age of the Responder. In this case, under the EA scheme, the Requester stores a copy of the document locally. However, at the Responder's cache, the entry corresponding to this document is not promoted to the HEAD of the LRU list. Eventually, it gets removed if there are no local hits. Hence, in this situation, the copy of the document at the Responder's cache is not given an additional lease of life. Again, the EA scheme reduces the number of replicas of documents in the cache.

Furthermore, by making a local copy whenever the Cache Expiration Age of the Requester is greater than that of the Responder, we are ensuring that a copy is made if the new copy is likely to survive for a longer time. By doing so, the new scheme guarantees that it would not report a miss when the ad hoc scheme would have had a hit. Therefore, this scheme achieves both of our objectives.

In addition to the fact that the EA scheme reduces the number of replicas while ensuring a copy be made if the new copy is likely to survive longer than the already existing copy, there are other useful features of the EA scheme. First, the implementation of our scheme does not carry any extra overhead. Only extra information that is communicated among proxies is the Cache Expiration Age. Even this information is piggybacked on either an HTTP request message or an HTTP response message. Therefore, there is no extra connection setup between the communicating proxies. Hence, there are no hidden communication costs incurred to implement the EA scheme. Second, as is evident from the algorithm, the decision regarding caching a document is done locally. The caches don't rely upon any other process to decide whether to cache a particular document or not. This is in contrast to some of the distributed file system implementations where the decision to store a copy of a file (or a part of it) is taken by centralized or distributed manager processes [2].

Algorithm 1 Algorithm for the Requester
    On receiving a client request
    if requested document is locally available **then**
        Serve the client request
    else
        Send an ICP query request to all parents and siblings
        if At least one parent or sibling reply positively **then**
            Send a HTTP request to first positive Responder. Also send CACHE-EXP-AGE of this cache
            Obtain document along with document obtain CACHE-EXP-AGE of the Responder
            if CACHE-EXP-AGE of this cache $\geq$ CACHE-EXP-AGE of Responder **then**
                Store a copy of the document locally
                Serve the client
            else
                Serve the client
            end if
        else
            if Cache has any parents **then**
                Send HTTP request to a parent asking it to resolve the miss.
                Along with the request send CACHE-EXP-AGE of this cache
                Obtain the document from parent. Along with it obtain its CACHE-EXP-AGE
                if CACHE-EXP-AGE of this cache $\geq$ CACHE-EXP-AGE of Responder **then**
                    Store a copy of the document locally
                    Serve the client
                else
                    Serve the client
                end if
            else
                Obtain the document from origin server
                Store a local copy and serve the client
            end if
        end if
    end if

Fig. 1. Algorithm for the Requester.

4 ANALYSIS OF THE EA SCHEME

In this section, we mathematically analyze the EA scheme and the ad hoc scheme used in today's Web caches and compare them. Specifically, we analyze the disk usage efficiency of the ad hoc document placement scheme and the Expiration-Age-based scheme. We also analyze the percentage of improvement in the Expiration Age of the caches in the new scheme over that of the conventional ad hoc scheme.

4.1 Disk Usage Efficiency

First, we analyze the disk usage efficiency of the EA scheme and the ad hoc scheme. By disk usage efficiency, we mean the percentage of the total disk space used to store unique documents. If there are absolutely no replicas in the cache group, the scheme ensures 100 percent disk efficiency.

Let G be a cache group of N caches, say $C_1, C_2, \ldots, C_N$. Let $uniqueDocs(C_i)$ be the set of unique documents in the cache C_i ($i \in \{1, 2, \ldots, N\}$) and $totalDocs(C_i)$ be the complete set of documents in the cache C_i. The aggregate disk efficiency of the cache group G of N caches, denoted as $DiskEfficiency(G, N)$, can be calculated by the following equation

$$\text{DiskEfficiency(G, N)} = \frac{\sum_{i=1}^{N} \sum_{D \in \text{uniqueDocs}(C_i)} (\text{docSize(D)})}{\sum_{i=1}^{N} \sum_{E \in \text{totalDocs}(C_i)} (\text{docSize(E)})}. \quad (6)$$

If we assume that all documents are of the same size (e.g., the average size of the documents in the cache), then the disk usage efficiency can be simplified as the ratio of the number of unique documents in the cache group G to the total number of documents in G. Thus, the above equation is reduced to

$$\text{DiskEfficiency(G, N)} = \frac{\sum_{i=1}^{N} |\text{uniqueDocs}(C_i)|}{\sum_{i=1}^{N} |\text{totalDocs}(C_i)|}. \quad (7)$$

For simplicity in what follows, we will use the ratio of number of unique documents to total number of documents in the cache group as a measurement of the effectiveness of disk usage and compare the new EA placement scheme with the ad hoc document placement scheme.

4.1.1 Disk Usage Efficiency in the Ad Hoc Scheme

Let us first look at the disk usage efficiency of the ad hoc scheme and then compare it with the expiration-age-based scheme.

Consider a group G of N caches cooperating in a pure distributed fashion. For simplicity, let us assume that each cache has a disk space of X bytes and consider any one of

Algorithm 2 Algorithm for the Responder (Sibling or Parent)

On getting a ICP query request, check for availability of document
if Document is available in local cache **then**
    Send a positive ICP reply
else
    Send a negative ICP reply
end if

On getting a HTTP request, check for availability of document
if Document is available **then**
    if CACHE-EXP-AGE of Requester < CACHE-EXP-AGE of this cache **then**
        Move the document entry to the HEAD of LRU List
    end if
    Send the document and CACHE-EXP-AGE of this cache to Requester.
else
    if Requester is a child **then**
        Obtain the document from Origin Server
        if CACHE-EXP-AGE of Requester < CACHE-EXP-AGE of this cache **then**
            Cache the document locally
        end if
        Send the document to the Requester
    else
        Send a error message
    end if
end if

Fig. 2. Algorithm for the Responder (sibling or parent).

the caches in the group. Let the cache get p client requests per second. Let the cumulative hit rate of the cache group be k (this includes both local and remote hits for a document).

Assume that every second there are p client requests. As the hit rate is k, we can assume that, out of the p client requests per second, $p(1 - k)$ are misses and the rest pk are hits. As there are N caches in the group, by the law of averages, we can assume that out of the pk hits, $\frac{pk}{N}$ are hits in the same cache (local hits). The rest $\frac{pk(N-1)}{N}$ are hits in other caches (remote hits). In the ad hoc scheme, given a cache C, each document that is either a miss or a remote hit will be added into the cache C. Hence, the total number of documents that are added every second into the cache group G as a whole, denoted as NumDocsAdded-AdHoc(G, N), can be estimated by the following formula:

$$\text{NumDocsAdded-AdHoc(G, N)} = p(1 - k) + \frac{pk(N - 1)}{N}. \quad (8)$$

Out of these documents that are added into the cache every second, $\frac{pk(N-1)}{N}$ are obtained from other caches. Hence, these are replicas of documents present in other caches. Therefore, the number of unique document added on to the cache is $p(1 - k)$. We know that the cache removal policy is unbiased and treats both replicas and unique documents equally. Therefore, the ratio of unique documents to total documents in the cache can be calculated as follows. This ratio is a measurement of the effectiveness of the disk usage under the ad hoc placement scheme, denoted by DiskEfficiency-AdHoc(G, N).

$$\text{DiskEfficiency-AdHoc(G, N)} = \frac{p(1 - k)}{\left(p(1 - k) + \frac{pk(N-1)}{N} \right)} \quad (9)$$

$$= \frac{N(1 - k)}{(N - k)}. \quad (10)$$

4.1.2 Disk Usage Efficiency in the EA Scheme

In the EA scheme, whenever there is a remote hit, the documents are added into the cache only if the Expiration Age of the Responder's cache is less than that of the Requester. Because we are considering a random cache in the group, it is reasonable to assume that half of these documents are from Responders that have lower Cache Expiration Ages than the Requester and the other half have Cache Expiration Ages that are greater than that of the Requester. Therefore, under the EA scheme, the number of documents entering the cache every second is:

$$\text{NumDocsAdded-EA(G, N)} = p(1 - k) + \frac{pk(N - 1)}{2N}. \quad (11)$$

Out of these documents that are added into the cache every second, the number of unique documents (nonreplicas) is $p(1 - k)$. Hence, the ratio of unique documents to the total number of documents under the new EA scheme can be calculated as follows. This ratio is seen as a measurement of the effectiveness of the disk usage under the new EA placement scheme, denoted as DiskEfficiency-EA(G, N).

$$\text{DiskEfficiency-EA(G, N)} = \frac{p(1 - k)}{\left(p(1 - k) + \frac{pk(N-1)}{2N} \right)} \quad (12)$$

$$= \frac{2N(1 - k)}{2N - k(N + 1)}. \quad (13)$$

In order to give insight into the actual numbers these equations yield, we have tabulated the values of both DiskEfficiency-AdHoc and DiskEfficiency-EA for cache groups having two, four, and eight caches and at cumulative hit rates of 0.25 and 0.5. Table 1 gives these values.

TABLE 1
A Comparison of Effectiveness of Disk Space Usage of Cache Groups

Number of Caches	Hit Rate	DiskEfficiency$_{Ad-hoc}$ (%)	DiskEfficiency$_{EA}$ (%)
2	0.25	85.71	92.30
2	0.5	66.67	80.00
4	0.25	80.00	88.89
4	0.5	57.14	72.73
8	0.25	77.41	87.37
8	0.5	53.33	69.57

4.2 Improvement in Cache Expiration Age

In Section 3.1, we defined the Expiration Age of caches and how it reflects the disk space contention in the cache. From the discussion in Section 3.5, it is clear that the EA scheme reduces the contention for disk space in the individual caches. In this section, we try to quantify this reduction in disk space contention by estimating the percentage improvement in the expiration age of the individual caches in the EA scheme as against the ad hoc scheme.

As before, let us consider a group of N caches, each cache being of size X bytes and receive p requests per second, the average size of documents being q and the cumulative byte hit ratio being v. Let us first analyze the ad hoc scheme. Consider a random cache in the group. This cache receives requests for p documents every second. The average size of each document is q. Therefore, the cache receives requests for $u = pq$ bytes of data every second. As the byte hit rate is v, the number of bytes missed in this cache per second is $u(1 - v)$. As there are N caches in the group, by the law of averages, it is reasonable to assume that, out of uv byte hits, $\frac{uv}{N}$ are available in the same cache (local hits bytes). The number of bytes hits in other caches (remote byte hits) can be approximated as $\frac{uv(N-1)}{N}$. In the ad hoc scheme, a document is added to the cache if it is a miss or a remote hit. Therefore, the number of bytes entering the cache per second can be calculated as the sum of size of missed documents and size of remote hit documents. Hence, the number of bytes added to the cache every second is given by:

$$\text{Num-Bytes-AdHoc} = u(1-v) + \frac{uv(N-1)}{N}. \quad (14)$$

When the caches are under a steady state of operation, if M bytes enter the cache every second, at least M bytes have to be removed from the cache. Therefore, in the ad hoc scheme, $u(1-v) + \frac{uv(N-1)}{N}$ bytes have to be removed from the cache every second. If the average size of each document is q bytes, the average number of documents being evicted per second is $\frac{u(1-v)+\frac{uv(N-1)}{N}}{q}$. The approximate number of documents that are present in the cache at any time is $\frac{X}{q}$.

Assuming that each cache is of capacity X bytes and employs LRU scheme for document replacement, the average time for which a document stays in the cache since its last hit is:

$$\text{AvgExpAge-AdHoc} = \frac{\frac{X}{q}}{\left(\frac{u(1-v)+\frac{uv(N-1)}{N}}{q} \right)} \quad (15)$$

$$= \frac{NX}{u(N(1-v) + v(N-1))}. \quad (16)$$

Now, let us consider our new scheme. In our scheme, documents obtained from other caches are added on only when the Cache Expiration Age of the Responder is less than the Cache Expiration Age of the Requestor (in this case, the cache under consideration). As before, we can assume that, on average, half of the documents are obtained from caches that have higher Cache Expiration Ages and the other half are obtained from caches that have lower Cache Expiration Age value than that of the cache under consideration. Hence, in the new scheme, the number of bytes entering cache per second would be:

$$\text{NumBytes-EA} = u(1-v) + \frac{uv(N-1)}{2N}. \quad (17)$$

As before, these many bytes have to be removed from the cache every second. Hence, the average age since the last hit of the documents would now be

$$\text{AvgExpAge-EA} = \frac{\frac{X}{q}}{\left(\frac{u(1-v)+\frac{uv(N-1)}{2N}}{q} \right)} \quad (18)$$

$$= \frac{2N}{u(2N(1-v) + v(N-1))}. \quad (19)$$

The percentage improvement of the LRU Expiration Age can be calculated as

$$\text{Improvement} = \frac{(\text{AvgExpAge-EA} - \text{AvgExpAge-AdHoc})}{\text{AvgExpAge-AdHoc} \times 100}. \quad (20)$$

Substituting for the values of AvgAge-AdHoc and AvgAge-EA from the equations above, we obtain

$$\text{Improvement} = \frac{v(N-1)}{(2N - v(N+1))} \times 100. \quad (21)$$

Similar to Table 1 in the previous section, Table 2 tabulates the values of percentage improvements in expiration ages for cache groups of two, four, and eight caches at byte hit rate of 0.25 and 0.5. It is widely acknowledged [11]

TABLE 2
Percentage Improvement in the Expiration Age of Cache Groups

Number of Caches	Byte Hit Rate	Expiration Age Improvement (%)
2	0.25	7.7
2	0.5	20.00
4	0.25	11.11
4	0.5	27.28
8	0.25	15.21
8	0.5	30.43

TABLE 3
Percentage Improvement in the Expiration Age of Cache Groups at Various Document Hit Rates

Number of Caches	Document Hit Rate	Byte Hit Rate	Expiration Age Improvement (%)
2	0.25	0.08	2.13
2	0.50	0.18	5.20
2	0.75	0.37	12.37
4	0.25	0.08	3.16
4	0.50	0.18	7.61
4	0.75	0.37	16.42
8	0.25	0.08	3.66
8	0.50	0.18	8.76
8	0.75	0.37	19.62

that, in any cache, byte hit rates are usually an order of magnitude less than the corresponding document hit rates. Therefore, to be more realistic in reporting the percentage improvement of expiration ages, we tabulate percentage improvements in expiration ages for cache groups of two, four, and eight caches at different document hit ratios and their corresponding byte hit ratios in Table 3. It should, however, be noted that the relation between document hit rate and byte hit rate is seldom the same for two different traces. Furthermore, the ratio $\frac{\text{Document Hit Rate}}{\text{Byte Hit Rate}}$ also varies across the document hit rate spectrum. At lower document hit rates, the ratio is higher and progressively falls as the document hit rate improves. The values we have used for byte hit ratios in Table 3 are based on the characteristics of the particular trace we have used in our experiments (Boston University traces). In Section 5.3, we report our experimental results on the expiration ages of the ad hoc scheme and the EA scheme, which closely follow the trends indicated in Table 3.

5 EXPERIMENTS AND PERFORMANCE RESULTS

We have listed some questions to be investigated in this research in the Introduction. One of the questions was whether a document placement strategy utilizing cache state information could improve document hit rates and latencies? To answer this and other related questions, we performed various simulation-based experiments and measured performance of both the ad hoc scheme and the EA-scheme on key metrics like cumulative hit rate, cumulative byte hit rate, and latency. Other than these three traditional metrics, we evaluated the two schemes with respect to average cache expiration age. In this section, we discuss the experiments we conducted and the results obtained.

5.1 Performance Metrics

We consider the following metrics to compare the ad hoc scheme with our new scheme: Cumulative Hit Rate, Cumulative Bit Hit Rate, Average latency, and Average Cache Expiration Age of the caches in the group. Cumulative Hit Rate and Cumulative Byte Hit Rate indicate the reduction of the load on the backbone network connecting the cache group to the outside world. Average Cache Expiration Age is a measure of disk space contention in the caches of the group.

5.2 Experimental Setup

We have implemented a trace-driven simulator to evaluate the new EA scheme against the conventional ad hoc scheme. We simulated the cache group by executing the simulator on different machines in our department. The simulators running on different machines communicate via UDP and TCP for ICP and HTTP connections, respectively. The machines were Solaris machines with a sparcv9 processor operating at 440mhz. These machines have 256 MB RAM and 512 MB swap space.

We used the Boston University proxy cache traces for our simulation. The traces were recorded from proxies and logs from proxies and HTTP servers from Boston University. The traces were collected from the middle of November 1994 to end of February 1995. The logs contain records of requests from 591 users over 4,700 sessions. The number of records in the log is 575,775, out of which the total number of unique records were 46,830. Additional information about the traces can be obtained from [3]. In the traces, there were log records with a size field equal to zero bytes. We made the size of each such record equal to average document size of 4K bytes.

In our experiments, we simulated a cache group containing two, four, and eight caches. Cooperative caching architecture of these cache groups is distributed cooperative
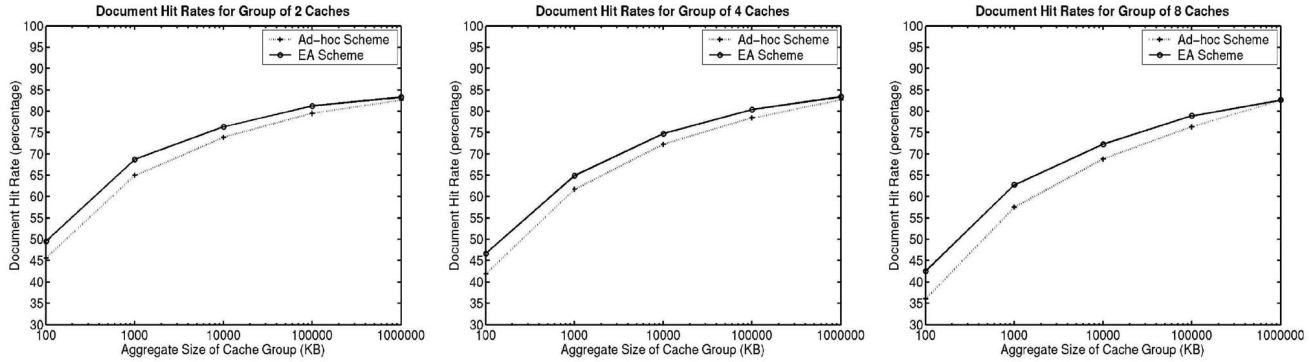
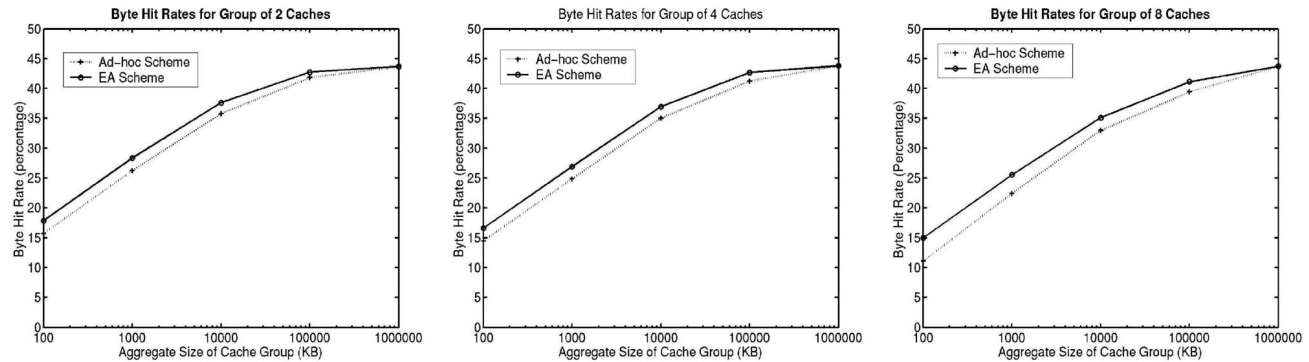Fig. 3. Document Hit Rates for cache groups of two, four, and eight caches.

Fig. 4. Byte hit rates for cache groups of two, four, and eight caches.

caching. In our simulations, we also assume that all the caches in the cache group have equal amounts of disk space. For each of the cache groups, we measured the above metrics when the aggregate cache size in the group varied from 100KB, 1MB, 10MB, 100MB to 1GB. All the reported results have been measured when all the caches have reached steady state.

5.3 Performance Results

In this section, we discuss the results obtained from five sets of trace-based simulations. The comparisons of performance between the EA scheme and the ad hoc scheme used in existing Web caches are conducted in terms of the document hit rate, byte hit rate, expiration age, and the latency experienced by clients.

5.3.1 Hit Rate Comparison

Fig. 3 indicates the hit rate of both ad hoc and EA document placement schemes for cache groups of two, four, and eight caches. We measured the hit rates at aggregate cache sizes of 100KB, 1MB, 10MB, 100MB, and 1GB. It can be seen that the difference between the two schemes is higher when the cache sizes are smaller. There is about a 6.5 percent increase in the document hit rate for a group of eight caches when the aggregate cache size is 100KB. The difference between the hit rates drops to 2.5 percent for the same cache group when the aggregate cache size is 100MB.

This observation can be explained as follows: When the cache sizes are small, even small amounts of increase in disk space yields substantial improvements in cache hit rates. This phenomenon has been discussed by many

researchers. Therefore, when cache sizes are small, a better utilization of the disk space yields substantial gains in terms of document hit rate. However, at larger cache sizes, a better utilization of disk does not translate correspondingly into hit rates. However, the EA scheme performs slightly better than the ad hoc scheme even for large caches. In summary, the EA scheme yields significantly better performance when the cache sizes are limited.

5.3.2 Cumulative Byte Hit Rate Comparison

Fig. 4 represents the byte hit rates for cache group of two, four, and eight caches. The byte hit rate patterns are similar to those of document hit rates. For a group of eight caches, improvement in byte-hit ratio is approximately 4 percent when the aggregate cache size is as small as 100KB and is about 1.5 percent when the aggregate cache size is at 100MB.

Improvement on the byte hit rates is substantially lesser than the improvement on the corresponding document hit rates. It is widely known that it is the smaller documents that are accessed repeatedly rather than documents of huge sizes. When smaller documents are accessed repeatedly, it contributes to an increase in the document hit rate, but it does not translate directly into an increase in byte hit ratio. When we compare the EA scheme and the ad hoc schemes with respect to their document and byte hit rates, the advantages of the EA scheme might seem limited. Indeed, the increase in document hit rates range from 2.5 to 6.5 percent for an eight cache group.

However, this fact should be considered in the light of the general hit rates of the trace at various cache sizes. For

TABLE 4
Average Cache Expiration Age (in Secs) for Group of Eight Caches

Aggregate Disk	Ad-hoc Scheme	EA Scheme	Percentage Improvement
100K	1.88	2.27	20.74
1M	21.49	26.79	18.51
10M	234.78	286.70	19.57
100M	2207.21	2664.09	20.70

example, the aggregate disk space of the cache group has to be increased by 10 times (from 100 KB to 1 MB) to achieve a hit rate improvement of around 20 percent for a group of eight caches. Similarly, increasing the aggregate cache size from 10 MB to 100 MB (again a 10 fold increase) for the same cache group results in the document hit rate improvement of just around 8 percent. These facts illustrate that the cache sizes have to be increased by very large amounts to achieve reasonable improvements in document and byte hit rates. Therefore, it is apparent that the document and byte hit improvements provided by the EA scheme, although not very momentous, is the result of significantly better utilization of the aggregate disk space available in the cache group. The effectiveness of the disk space usage is also reflected by the improvements in the average document expiration age, which we discuss next.

5.3.3 Expiration Age Comparison

To illustrate the reduced disk space in the cache group, we tabulate the Average Cache expiration age for both the ad hoc and the EA scheme. Though we have performed experiments on cache groups of two, four, and eight caches, we limit our discussion to the 8-cache group due to space limitations. Table 4 show these values for cache groups of eight caches at various aggregate cache sizes. We have not come across any cache related paper that uses cache expiration age as a metric. However, we regard it as an important metric that indicates disk space contention in the cache group. We can observe that, with the EA scheme, the documents stay for much longer as compared with the ad hoc scheme. This demonstrates that the EA scheme reduces disk space contention in the cache group. It can also be seen from these tables that the improvement on average cache expiration age closely follows the analytic model described in Section 4.2. For example, for a cache group of

eight caches at an aggregate disk space of 10MB, the byte hit rate is 35 percent and the percentage improvement on the average expiration age is 20 percent, which is close to the value computed through the analytic model. Therefore, the experimental results validate our analytic models.

In order to compare hit rates and byte hit rates and cache expiration ages of cache groups having same aggregate cache size but having different number of caches, we plotted hit rates, byte hit rates, and average cache expiration ages of cache groups of two, four, and eight caches but having the same amount of aggregate disk space. Graphs in Fig. 5 show the hit rates, byte hit rates, and average cache expiration ages of both the schemes when the aggregate cache size is set to 1MB.

It can be observed that, when the aggregate cache size is constant, the hit rates decrease as the number of caches increase. This is because the available disk space in individual caches decrease as the number of caches increase. The graphs in Fig. 5 show that there is more than a 5 percent drop in document hit rate when the number of caches in the group increases from two to eight. However, our scheme reduces this effect. The drop is below 4 percent in our scheme.

5.3.4 Performance Comparison on Latency Experienced by Client

To study the impact of the EA scheme on the latency experienced by clients, we estimated average document latency for both the ad hoc scheme and the EA Scheme. In order to estimate the average document latency, we measured the latency for local hits, remote hits, and also misses for retrieving a 4KB document. We ran the experiments 5,000 times and averaged out the values. The latency of a local hit (LHL) was 146 milliseconds. The latency of a remote hit (RHL) was 342 milliseconds and the
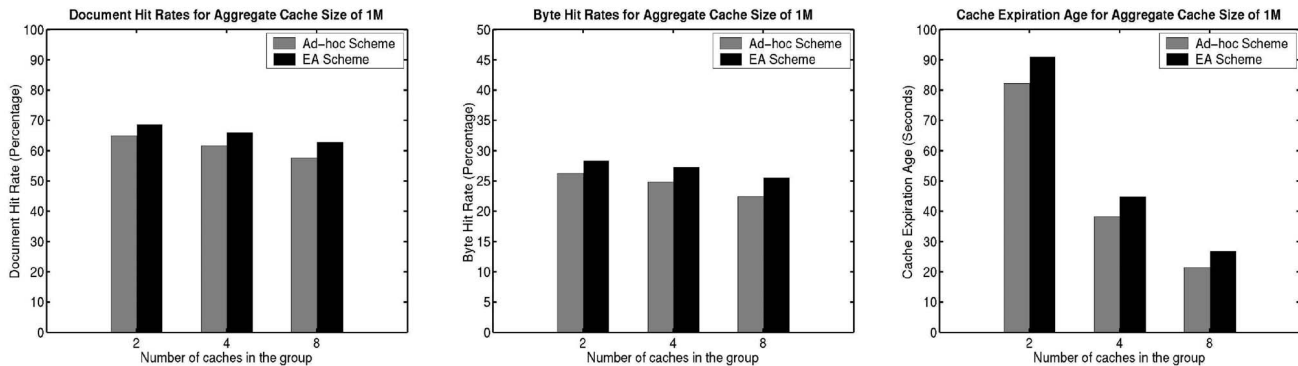

Fig. 5. Hit Rate, Byte Hit Rate, and Cache Expiration Age for groups of two, four, and eight caches with constant aggregate cache size.
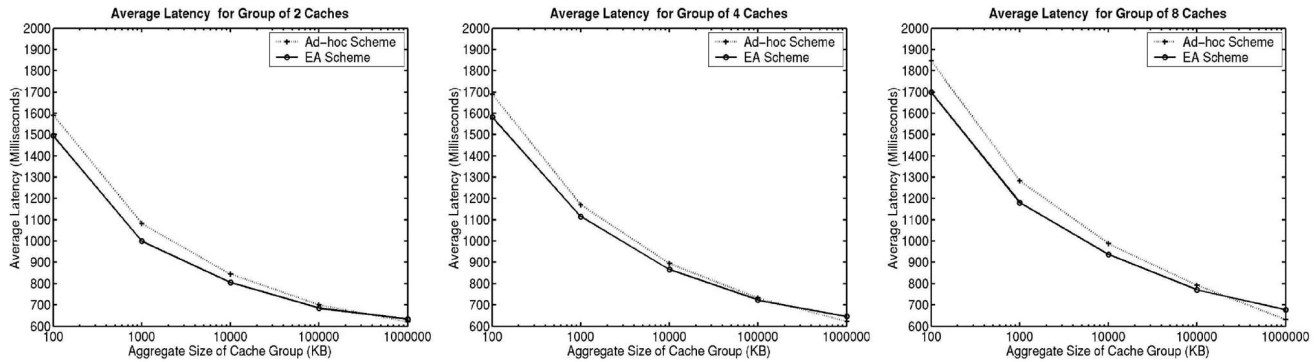
Fig. 6. Estimated latency for cache groups of two, four, and eight caches.

latency of a miss (ML) was 2,784 milliseconds. To obtain latency values for misses, we measured latency values for various Web sites and took their mean value.

In order to measure the average latency, we simulated a group of caches in which the local hit latency, remote hit latency, and miss latency were made random variables. The experimental setup was similar to the previous scenario. Average user latency would be the individual request latencies averaged over all requests at all caches. In our experiments, we modeled all random variables as having Gaussian distribution with variance of each Gaussian variable being set to half of its mean.

The first set of experiment results are plotted in Fig. 6. The graphs in Fig. 6 indicate the average latency for group of two, four, and eight caches, respectively. It is clear that the EA scheme performs significantly better when the cumulative cache size is 100KB, 1MB, and 10MB. When the cache size is 100MB, the latencies of both schemes are approximately the same, whereas, at 1GB, the average latency of the EA scheme is slightly higher than that of the ad hoc scheme for a number of reasons.

First, the local and remote hit rates under the EA scheme are different from the local and remote hit rates of the ad hoc placement scheme. As we reduce the number of replicas in individual caches, the remote hit rates under the EA scheme will increase. Table 5 shows the local and remote hit rates for a group of four caches along with the estimated latency values. As we can see, the remote hit rates in the EA scheme are higher than that of the ad hoc scheme.

Second, when the cache sizes are small, the miss rates under the ad hoc scheme are higher than those of the EA scheme. As the average latency for serving misses is relatively large compared to the latency for serving local

and remote hits, the average document latency under the EA scheme is much lower than that of the ad hoc scheme. However, as the cache size reaches 1GB, the difference between the miss rates of the two schemes becomes very little. Now, the remote hit latency becomes the dominating factor in determining the average document latency. Thus, the ad hoc scheme performs slightly better than the EA scheme at 1GB. We observe that, in general, when the miss rates of the ad hoc scheme are higher than the miss rates of the EA scheme, the EA scheme performs substantially better than the ad hoc scheme.

In the second set of experiments, we wanted to study the impact of the relative values of miss latency and remote hit latency. As already indicated, the measured values for these parameters were 2,784 millliseconds and 342 milliseconds. In other words, the ratio $\frac{\text{Miss Latency}}{\text{Remote Hit Latency}}$ is 8.14. This set of experiments studied the impact of varying this ratio on the average latency of the two schemes. In these experiments, we kept the Miss Latency values and the Local Hit Latency values to be constant at 2,784 milliseconds and 146 milliseconds, respectively, and varied the ratio $\frac{\text{Miss Latency}}{\text{Remote Hit Latency}}$ from 2.5 to 15 and obtained the corresponding values for Remote Hit Latency. The graphs in the Fig. 7 (from left to right) indicate the latency values for groups of two and eight caches, respectively. Each graph indicates the average latency when the aggregate cache size of the cache group (aggregate memory for short) was set to 100KB, 10MB, and 1GB for both ad hoc and EA schemes, respectively. The vertical line at 8.14 indicates the measured value of the ratio $\frac{\text{Miss Latency}}{\text{Remote Hit Latency}}$. For a

TABLE 5
A Comparison of Ad Hoc Scheme and EA Scheme for Group of Four Caches

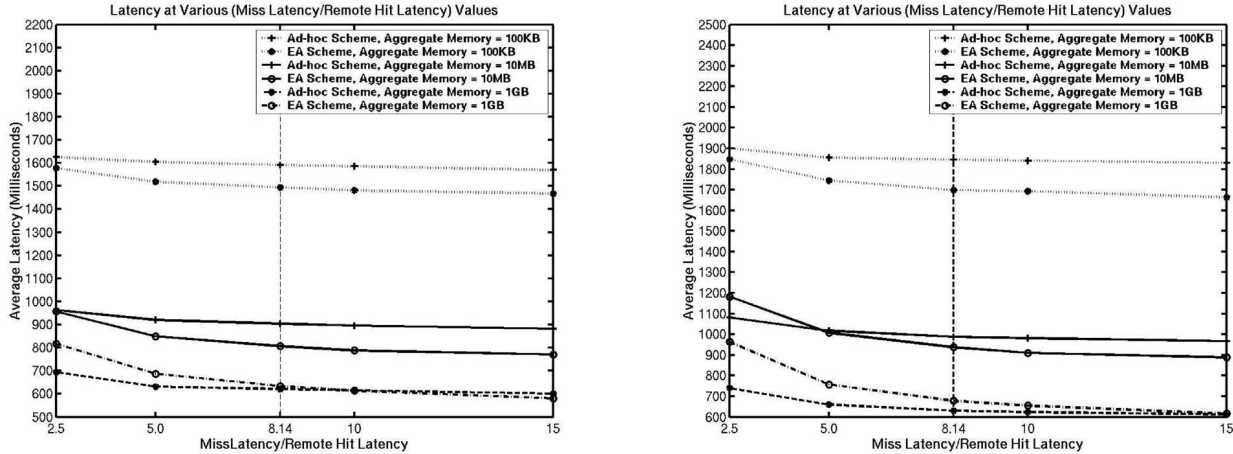| Aggregate Cache Size	Ad-hoc Scheme			EA Scheme		
	Local Hits	Remote Hits	Latency	Local Hits	Remote Hits	Latency
100 KB	36.18	5.70	1689.15	30.45	16.26	1582.44
1 MB	53.28	8.43	1170.48	42.32	22.58	1014.84
10 MB	62.55	9.71	894.69	48.38	27.37	866.08
100 MB	68.11	10.35	732.20	49.87	30.51	722.60
1 GB	71.62	11.06	622.15	51.38	32.02	645.89

Fig. 7. Comparing the ratio of miss latency and remote latency for cache groups of two and eight caches.

group of two caches, when the ratio $\frac{\text{Miss Latency}}{\text{Remote Hit Latency}}$ is 2.5, it is seen that the average latency of the EA-Scheme is slightly better than that of the ad hoc scheme (by about 50 milliseconds). This difference increases progressively as the value of $\frac{\text{Miss Latency}}{\text{Remote Hit Latency}}$ increases. When the ratio betweeen Miss Latency and Remote Hit Latency reaches 15, the average latency of EA scheme is better than that of the ad hoc scheme by almost 110 milliseconds. When the aggregate cache size of the cache group is 1GB, the average latency for the ad hoc scheme performs better than EA scheme by almost 100 milliseconds when $\frac{\text{Miss Latency}}{\text{Remote Hit Latency}} = 2.5$. However, when the ratio $\frac{\text{Miss Latency}}{\text{Remote Hit Latency}}$ reaches 15, the EA scheme performs better than the ad hoc scheme by a small amount (couple of milliseconds). Similar patterns of behavior are observed in the groups of four and eight caches.

From the graphs in Fig. 7, it can be concluded that the EA scheme is advantageous if the communication cost among the proxies is low when compared with the cost of fetching the document from the origin server. In fact, this is to be anticipated. As Table 5 indicates, in the EA scheme, the remote hit rate is significantly higher when compared with the ad hoc scheme. Hence, the remote hit latency plays a significant role in determining the average latency.

In our experiments, we have concentrated upon the distributed cooperative caching architecture. However, as explained in Section 3.4, the EA scheme is applicable both to hierarchical and to distributed caching architectures. We now discuss the likely impact of the EA scheme on cooperative caches that are organized in a hierarchical fashion.

First, it should be noted that the amount of document replication in hierarchical caches is, in general, higher than that of the distributed caches. This is because, in distributed caching, the request for a file would increase the number of replicas of that file in the cache group by at most by one. In contrast, a single request in hierarchical caches can create D replicas at the worst, where D is the depth of the cache hierarchy. While the EA scheme reduces the amount of replication both in the distributed and the hierarchical

architectures, we expect that the average number of file replicas existing in the hierarchical caches with the EA scheme to be higher than the average number of file replicas existing in their distributed caching counterparts.

The effectiveness of the EA scheme for hierarchically organized caches depends upon the relative amounts of disk space available at caches at different levels of hierarchy. For example, if the same amount of disk space is available at all caches irrespective of its position in the hierarchy, then the disk space contentions at caches located in higher levels of hierarchy are much higher than that of the caches in lower levels of hierarchy. In the ad hoc scheme, as the files are replicated in every cache they flow through, the files would be removed at a much faster rate in the caches at higher levels of hierarchy than the files in caches at lower levels of hierarchy. However, consider the file replication mechanism in the EA scheme. As the disk space contention is higher in the caches at higher levels of hierarchy, these caches rarely make a local copy of the files (as their Expiration Ages would be low). Hence, the EA scheme reduces the disk space contentions in caches that are at higher level of hierarchy, and hence, increases their Expiration Ages. For caches that are situated at lower levels, the performance of the EA Scheme and the ad hoc scheme are similar. Therefore, on the whole, the EA scheme improves the Average Cache Expiration Age of the entire cache group.

The effect of the EA scheme on the average latency experienced by the client in hierarchical caches is dependent upon the depth of the hierarchy. In fact, the depth of the hierarchy affects the average client latency in both the ad hoc and the EA schemes. The remote hit latency and the miss latency values for hierarchical caches are, in general, higher than the corresponding values for distributed caches. Further, in hierarchical caches, these values increase with the increasing depth of the hierarchy. This is due to the flow of the requests and the responses in the hierarchy. Hence, the effect of cooperative caches itself is fairly limited in hierarchical caches if we exclusively consider the average latency experienced by the client. We believe that the EA scheme would still improve the average client latency for hierarchically organized caches,

particularly when caches have limited disk space, although the improvement may be on a diminished scale when compared with distributed cooperative caches.

In summary, the EA document placement scheme yields higher hit rates, byte hit rates, and reduces the average document latencies in those caches where document hit rates are sensitive to the disk space availability in the group.

6 RELATED WORK

6.1 Comparison with Hash Routing Schemes

Of the vast literature available on Web caching, the research results that are most relevant to our work come from a family of cooperative caching schemes that extend the work of Name-based mapping and Hash routing [13], [17], [22], [21], [24], [8]. The central idea in these schemes is to use name-based mapping or hash routing algorithms to map URLs to caches that possibly contain the corresponding documents. The clients, on performing the local computations, contact the caches directly. If the cache contains the documents, then it is supplied to the client. Otherwise, the cache retrieves the document from the origin server, caches it, and supplies it to the client. These schemes aim at reducing the total latency experienced by the client through eliminating interproxy communication cost. In this section, we compare and contrast the EA scheme with some representative schemes from this family of cooperative caching schemes and also with a very recent work on coordinated placement on replacement for distributed caches.

The first and foremost difference between the EA scheme and the hash routing schemes is that all the hash routing schemes utilize hash-based routing protocols to locate a document in the cache group, whereas the EA scheme is primarily based on the ICP protocol for document discovery.

The second key difference lies in the amount and the nature of document replication occurring in these schemes. The original hash routing scheme [13] partitions the entire URL space among the participating proxies. Hence, there is absolutely no duplication of documents in this scheme. However, it was soon realized that this can lead to load imbalance among the participating proxies and can thus create hot spots. To alleviate this problem, Wu and Yu [21] proposed an *adaptable controlled replication* scheme (ACR scheme) which permits a document to be present in a cache to which it is not assigned to by the hashing algorithm, provided that the document is frequently accessed in that cache. In consistent hashing [8], the authors prove that no URL is stored in more than $O(logM)$ caches. However, in consistent hashing, the document-to-cache mapping is done by random hashing, and hence, the documents selected for replication is also at random. The EA scheme differs from both consistent hashing and hash routing schemes. The nature of document replication in the EA scheme is in a way comparable to that of the replication occurring in the ACR scheme. In both schemes, the number of replicas that exist in the cache group is directly dependent upon the *popularity* of the document within the community represented by the cooperative caching group. It is clearly advantageous to have multiple copies of popular documents in a cache group as it mitigates the hot spots problem. However, the approaches of the EA and ACR schemes are quite different.

First, the ACR scheme assumes that the caches cooperate through the hash routing mechanism, whereas the EA scheme is designed for caches cooperating via the ICP protocol. Second, the mechanisms employed in the ACR and the EA schemes for controlled replication are different. ACR is a hash-routing-based scheme and, therefore, it provides mechanisms to add replication to the cooperative caches that have no replication in a pure hash-routing-based design. The EA scheme is an ICP-based scheme and, therefore, it provides mechanisms to control and reduce the amount of replication that exist in a group of cooperative caches due to ad hoc placement of the existing ICP-based schemes.

6.2 Comparison with Other Document Placement Schemes

Recently, Korupolu and Dahlin [9] reported their work on coordinated placement and replacement schemes for distributed caches. Though their scheme is related to the EA scheme, it considerably differs from our work in various aspects. First and foremost, the problem formulations considered in the two papers, though related, are quite different from each other. They define a cost function for document placement. This function measures the cost incurred by a cache to obtain the document from its nearest available copy. The document placement schemes studied by them aim at minimizing the sum of the cost functions of all documents over all caches in the group. In contrast, our scheme is concerned with controlling document replication in the cache group.

Second, their work assumes that the caches in the cache group are organized into *clusters* and these clusters themselves are arranged in a tree structure. Our scheme assumes no such cache organization and functions with both hierarchical and distributed cache architectures. Third, a key assumption in their scheme is that reasonably accurate access patterns are available for the caches in the cache group. The EA scheme needs no such predictions about access patterns. Fourth, the decision whether to cache a document at a particular node is made on the fly in the EA scheme, whereas the scheme proposed by Korupolu and Dahlin involves a bottom-up pass of the cluster tree structure to determine the cache where the document has to be stored. Hence, the EA scheme can be considered to be more general and practical than the scheme proposed in [9].

6.3 Discussion on Other Caching Related Issues

The basic research in cooperative proxy caching has been focused on cache sharing protocols and cooperative cache hierarchies aimed at improving hit rates and document access latencies. Summary cache [5] and adaptive Web caching [10] are mechanisms that are proposed by different researchers to optimize the number of ICP messages among caches in resolving misses. Other techniques such as hints, directories, hashing [8], [15], [23] have been proposed to reduce the overall cost of document location process. A cooperative caching architecture provides a paradigm that assists proxies in cooperating efficiently with each other. Current cooperative cache architectures are roughly classified as Hierarchical and Distributed cache architectures. Hierarchical architecture [4], [14] sets up a cache hierarchy by specifying a parent-child relationship among the caches. In contrast, distributed caching architecture does not have a

hierarchical structure. The only relationship that exists among the caches is the sibling or the peer relationship. Each of these caching architectures has its advantages and disadvantages. A detailed discussion on the cache architectures and their relative merits and demerits along with experimental evaluations is available in [16]. In addition to the above, research in cooperative proxy caching protocols ranges from cache coherence, document fetching modes to cache replacement schemes [1], [18], [19], [11].

7 CONCLUSION AND FUTURE WORK

Although the field of cooperative caching has been extensively researched, very few have studied the effect of cooperation on document placement schemes and its potential enhancements on cache hit ratio and latency reduction.

In this paper, we have presented an Expiration-Age-based document placement scheme (the EA scheme). The main idea is to view the aggregate disk space of the cache group as a global resource of the group and uses the concept of cache expiration age to measure the contention of individual caches. The EA scheme effectively reduces the replication of documents across the cache group, while ensuring that a copy of the document always resides in a cache where it is likely to stay for the longest time. We have reported our initial study on the potentials and limits of the EA scheme using both analytic modeling and trace-based simulation.

Our research on data placement and caching continues along several dimensions. We want to investigate the interactions of our data placement scheme with other collaborative caching schemes such as Latency sensitive hashing [23], Adaptive Web caching [10], and Self-organizing caching [7]. Further, we intend to develop application specific variants of our data placement scheme to target new environments such as peer-to-peer computing systems, mobile, and wireless computing systems. We believe that better data placement schemes not only have the potential to improve the performance of these systems, but are also attractive solutions for optimizing power consumption, which is crucial in mobile and wireless environments.

ACKNOWLEDGMENTS

REFERENCES

[1] C. Aggrawal, J.L. Wolf, and P.S. Yu, "Caching on the World Wide Web," *IEEE Trans. Knowledge and Data Eng.,* vol. 11, no. 1, Jan./Feb. 1999.

[2] T.E. Anderson, M.D. Dahlin, J.M. Neefe, D.A. Patterson, D.S. Roselli, and R.Y. Wang, "Serverless Network File Systems," *ACM Trans. Computer Systems,* Feb. 1996.

[3] A. Bestavros, R.L. Carter, M.E. Crovella, C.R. Cunha, A. Heddaya, and S. Mirdad, "Application-Level Document Caching in the Internet," *Proc. Second Int'l Workshop Services in Distributed and Networked Environments,* 1995.

[4] A. Chankhunthod, P. Danzig, C. Neerdaels, M. Schwartz, and K. Worell, "A Hierarchical Internet Object Cache," *Proc. 1996 USENIX Technical Conf.,* Jan. 1996.

[5] L. Fan, P. Cao, J. Almeida, and A. Broder, "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol," *Proc. ACM SIGCOMM '98,* Sept. 1998.

[6] Internet Cache Protocol: Protocol Specification, version 2, Sept. 1997, http://icp.ircache.net/rfc2186.txt.

[7] S. Inohara, Y. Masuoka, J. Min, and F. Noda, "Self-Organizing Cooperative WWW Caching," *Proc. 18th Int'l Conf. Distributed Computing Systems,* 1998.

[8] D. Karger, A. Sherman, A. Berkheimer, B. Bogstad, R. Dhanidina, K. Iwamoto, B. Kim, L. Matkins, and Y. Yerushalmi, "Web Caching with Consistent Hashing," *Proc. WWW-8,* May 1997.

[9] M.R. Korupolu and M. Dahlin, "Coordinated Placement and Replacement for Large-Scale Distributed Caches," *IEEE Trans. Knowledge and Data Eng.,* vol. 14, no. 6, Nov./Dec. 2002.

[10] S. Michel, K. Nguyen, A. Rosenstein, L. Zhang, S. Floyd, and V. Jacobson, "Adaptive Web Caching: Towards a New Global Caching Architecture," *Computer Networks and ISDN Systems,* Nov. 1998.

[11] M. Rabinovich and O. Spatscheck, *Web Caching and Replication.* Addison Wesley Professional, Dec. 2001.

[12] P. Rodriguez, C. Spanner, and E. Biersack, "Analysis of Web Caching Architectures: Hierarchical and Distributed Caching," *IEEE/ACM Trans. Networking,* Aug. 2001.

[13] K.W. Ross, "Hash-Routing for Collections of Shared Web Caches," *IEEE Network Magazine,* 1997.

[14] A. Rousskov and V. Soloviev, "A Performance Study of the Squid Proxy on HTTP/1.0," *World Wide Web,* vol. 2, nos. 1-2, Jan. 1999,

[15] P. Sarkar and J. Hartman, "Efficient Cooperative Caching Using Hints," *Proc. USENIX Conf. Operating Systems Design and Implementation,* Oct. 1996.

[16] R. Tewari, M. Dahlin, H. Vin, and J. Kay, "Beyond Hierarchies: Design Considerations for Distributed Caching on the Internet," *Proc. Int'l Conf. Distributed Computing Systems '99,* May 1999.

[17] D. Thaler and C. Ravihsankar, "Using Name-Based Mappings to Increase Hit Rates," *IEEE/ACM Trans. Networking,* Feb. 1998.

[18] J. Wang, "A Survey of Web Caching Schemes for the Internet," *ACM Computer Comm. Rev.,* Oct. 1999.

[19] D. Wessels, *Web Caching.* O'Reilly and Assoc., June 2001.

[20] A. Wolman, G.M. Voelkar, N. Sharma, N. Cardwell, A. Karlin, and H.M. Levy, "On the Scale and Performance of Cooperative Web Proxy Caching," *Proc. ACM Symp. Operating Systems Principles '99,* Dec. 1999.

[21] K.-L. Wu and P.S. Yu, "Load Balancing and Hot Spot Relief for Hash Routing among a Collection of Proxy Caches," *Proc. Int'l Conf. Distributed Computing Systems,* 1999.

[22] K.-L. Wu and P.S. Yu, "Local Replication for Proxy Web Caches with Hash Routing," *Proc. ACM Eighth Int'l Conf. Information and Knowledge Management,* 1999.

[23] K.-L. Wu and P.S. Yu, "Latency Sensitive Hashing for Collaborative Web Caching," *Computer Networks,* June 2000.

[24] K.-L. Wu and P.S. Yu, "Replication for Load Balancing and Hot-Spot Relief on Proxy Web Cache with Hash Routing," *Distributed and Parallel Databases,* vol. 13, no. 2, 2003.

Lakshmish Ramaswamy received the master's degree from the Indian Institute of Science, Bangalore, India, in 1999. He is currently a PhD student in the College of Computing at the Georgia Institute of Technology. His research interests are in distributed computing systems and databases, including Web caching and performance, efficient delivery of dynamic contents, peer-to-peer systems, and efficient processing of large data sets in distributed systems. He is a student member of the IEEE.

Ling Liu received the PhD degree in computer science in 1993 from Tilburg University, The Netherlands. She is currently an associate professor in the College of Computing at the Georgia Institute of Technology. Her research involves both experimental and theoretical study of distributed systems in general and distributed data intensive systems in particular, including distributed middleware systems, advanced Internet systems, and Internet data management. Her current research interests include performance, scalability, reliability, and security of Internet services, pervasive computing applications, as well as data management issues in mobile and wireless systems. Her research group has produced a number of software systems that are either operational online or available as open source software, including WebCQ, XWRAPElite, Omini, and PeerCQ. She is currently on the editorial board of the *International Journal of Very Large Database Systems* (*VLDBJ*), editor-in-chief of *ACM SIGMOD Record*, and a vice program committee chair of the IEEE International Conference on Data Engineering (ICDE 2004). She was the program committee cochair of the 2001 International Conference on Knowledge and Information Management (CIKM 2001) held in November 2001 in Atlanta and the program committee cochair of 2002 International Conference on Ontology's, DataBases, and Applications of Semantics for Large Scale Information Systems (ODBASE), held in October, Irvine, California. She is a member of the IEEE Computer Society.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.