A New Document Placement Scheme for Cooperative Caching on the Internet

Lakshmish Ramaswamy and Ling Liu
College of Computing, Georgia Institute of Technology
801, Atlantic Drive, Atlanta, GA 30332, U S A
{laks, lingliu}@cc.gatech.edu

Abstract

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 called the EA 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. The experiments 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.

1. Introduction

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 [5] to gain the full benefits of caching.

Since then several cooperative caching protocols have been proposed [5, 6, 9, 13], aiming at improving hit ratio and reducing document-access latency. However, 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 the 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, especially how does the document placement scheme relate to the ratio of the inter-proxy communication time to server fetch time, and to the scale at which cooperation is undertaken? This paper presents a new document placement scheme and reports our initial experimental results with an in-depth analysis to answer these and other related questions.

Let us first review how documents are placed in present cooperative caching protocols and what are the potential problems of the existing schemes. 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. In other words the replication decision is made regardless of disk space contention experienced at individual caches. 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 (non-replicated) documents present in the cache.

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 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. Our trace-based simulation experiments show that the EA scheme yields higher hit rates and better response times. We report our simulations and experimental results in this paper.

2. Problem Statement

Cooperative caching is a mechanism where a group of web caches communicate among each other to achieve better performance. Whenever a cache experiences a miss for a document, it tries to locate whether the document is present in any of the nearby caches. If so it retrieves the document from that cache rather than contacting the origin server.

It is known that the benefits of cooperative caching are bound by the ratio of inter cache communication time to server fetch time. Researchers have been studying cache-sharing protocols, which provide mechanisms to reduce the communication cost among cooperating caches. Internet Cache Protocol (ICP) [1] was designed specifically for communication among web caches. ICP is a light-weight protocol and is implemented on top of UDP.

The protocol consists of two type of messages that are exchanged between neighboring caches viz. 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.

Most commercial and public domain proxy caches implement ICP or some variant of it for inter-proxy communication mechanism. However, as briefly mentioned in Section 1, all existing cooperative caching protocols treat individual caches in a cooperation group as completely independent entities when making decision on where to cache a document. More concretely, the decision of whether to place a particular document in a given cache is made without the knowledge about the other caches. This blind caching can lead to "uncontrolled" replication of documents in the cache group. It may potentially reduce the cumulative 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. They are related to each other through peer/sibling relationship. Without loss of generality, 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. All our arguments can be extended to Hierarchical cache architecture.

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 worst case of this limitation, though hypothetical, would be all the documents being replicated on all the caches. In this case, the effective disk space in the cache group is (1/N) times the aggregate disk space available, where N is the total number of caches 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.

- First, the time for which individual documents live in the cache is reduced.
- Second, the number of unique documents available in the cache group decreases.

The cumulative effect of the above two leads to a fall in the aggregate hit rate of 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 memory in the cache group is not optimal in the existing (ad-hoc) document placement scheme.

3 The Expiration-Age based Scheme

In this section, we present our new document placement scheme for cooperative caching. In this scheme, each proxy makes informed and 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 available how long it is likely to remain in the caches where they are 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 subsection we explain the concept of *Expiration Age* and how it can be used to make informed decisions.

3.1. Expiration Age - A Cache Contention Measure

In this subsection we discuss measures that can be used to compare disk space contention in various caches of a group. One obvious way 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 individual documents that are evicted in the given duration.

Consider a document D that entered a cache at time T_0 and was removed at time T_1. The *Life Time* of document D is defined as $(T_1 - T_0)$. The *Average Document Life Time* of the cache is defined as the average 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 doesn't accurately reflect the cache contention. In the technical report version of this paper [11] we provide a detailed example that illustrates the short coming of *Average Document Life Time* in accurately measuring the disk space contention in individual caches. The example also indicates that any measure for disk space contention should take into account the document hits and misses. This is simply because document replacement is an action to take for caches in reaction to the disk space contention problem and almost all document replacement (removal) policy depends on hits and misses, rather than the lifetime of a document in the cache. Typically LRU and its variants use the time when a document was last hit as a major factor for removal, whereas LFU and its variants use the number of times a document was hit since it entered the cache to determine which documents are victims for removal. This observation led us introduce the *Expiration Age* of a cache as the measure for the disk space contention of the cache.

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 average a document gets a hit at the cache. Obviously, the average of the expiration ages of the documents that were removed from a cache can be a useful measure of the disk-space contention at the cache. When the disk-space contention at the cache is higher, the average document expiration age of a cache will be lower and vice versa.

In the next subsection we formally introduce the concept of document expiration age and the methods to compute document expiration age. We introduce the concept of expiration age of a cache and the formula to compute the cache expiration age in the subsequent section.

3.2. Expiration Age of a Cached Document

Following the discussion in the previous section, there are two representative ways to define the *Expiration Age* of a cached document. It can be defined by the time between its removal and its last hit to indicate how long a document is expected to live in a cache since its last hit. This approach requires caching proxies to maintain the last hit time-stamp for every document. The alternative is to define the *Expiration Age* of a cached document in terms of the average time it takes for the document to get a hit. This requires caching proxies to maintain the number of hits for each document cached.

Most of today's caching proxies employs the Least Recently Used (LRU) algorithm or the Least Frequently Used (LFU) algorithm or some variants of LRU or LFU as their cache replacement policy [4, 14]. Any proxy that uses LRU or its variants will keep the last hit time-stamp for each document they cache. Similarly, any proxy that uses LFU or its variants maintains the number of hits every cached document experiences. Therefore, the expiration age based document placement scheme can be supported at almost any proxy with almost no extra cost.

With this observation in mind, We define the *Expiration Age* of a document in a cache, denoted as *DocExpAge*(D,C), by a formula that combines both LRU Document Expiration Age and LFU Document Expiration Age in the sense that *DocExpAge*(D,C) is equal to the LRU Document Expiration Age when the cache C uses LRU or its invariant and equal to the LFU Document Expiration Age when the cache C employs LFU for document replacement.

$$\text{DocExpAge }(D, C) = \begin{cases} DocExpAge_{LRU}(D, C) \\ \qquad \text{if cache } C \text{ employs LRU replacement policy} \\ \\ DocExPAge_{LFU}(D, C) \\ \qquad \text{if cache } C \text{ employs LFU replacement policy} \end{cases} \tag{1}$$

$DocExpAge_{LRU}$ and $DocExpAge_{LFU}$ are defined in section 3.2.1 and section 3.2.2 respectively.

3.2.1. LRU Expiration Age

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. It is straightforward for any caching proxy that uses LRU cache replacement to support the Expiration Age based document placement scheme.

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_1 and the last hit on the document occurred at time T_0, then the LRU Expiration Age of document D, denoted by *DocExpAge*$_{LRU}$(D,C), is defined as

$$DocExpAge_{LRU}(D,C) = (T_1 - T_0) \tag{2}$$

The LRU expiration age of a document indicates how long a document can be expected to live in a cache after its last hit. To compute the expiration age for each of the documents removed from a cache in a given period, the caching proxy simply needs to keep the time that a document was last hit and the time that this document was removed from the cache for the given period of time.

3.2.2. LFU Expiration Age

The Least Frequently Used (LFU) scheme is another popular cache replacement policy. In this scheme, whenever a document has to be evicted, the document with 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 1 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 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.

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

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 the time that a document is expected to live after its last hit.

We have discussed definitions of document Expiration Ages for two most popular and well-studied document replacement policies (LRU and LFU). We believe that it is possible to define the same for other replacement policies too. In this paper all our experiments employ 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. But it has to be remembered that Expiration Age definition depends on the replacement method being employed in the caches.

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. Recall 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 a 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).

$$\begin{aligned} Victim(C, T_I, T_j) &= \{D \mid \forall T < T_I, Live(D,C,T) \\ &\quad \wedge \forall T' > T_j, Evicted(D,C,T')\} \end{aligned} \tag{4}$$

Where the predicate $Live(D,C,T)$ denotes that the document D lives in the cache C at time T, and the predicate $Evicted(D,C,T')$ denotes that the document D was already evicted from the cache C at time T'.

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 by the following formula:

$$CacheExpAge(C, T_I, T_j) = \frac{\sum_{D \in Victim(C,T_I,T_j)} DocExpAge(D,C)}{|Victim(C,T_I,T_j)|} \tag{5}$$

The Expiration Age of a cache indicates the average time a document can be expected to live in the cache after its last hit. This measure reflects the disk space contention going on in the cache and thus can be considered as an indicator of the disk space contention at the cache. A high value for Cache Expiration Age indicates a low disk space contention and vice versa. We base our scheme on this crucial observation. We explain our scheme in the next subsection.

3.4. The Algorithms for EA Scheme

In this section we explain the algorithmic details of the EA scheme. 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 *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 own 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 Requester, the Requester doesn't 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 the 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, 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 Cache Expiration Age is greater than that of the parent).

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. So even in the worst case our scheme is as good as the adhoc placement scheme. We explain how the EA scheme achieves these objectives.

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, Requester will not store a copy of the requested document locally, when its Cache Expiration Age is less than the Cache Expiration Age of the Responder. 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.

In addition, 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 wouldn't report a miss when the original 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 doesn't carry any extra overhead. Only extra information that is communicated among proxies is the Cache Expiration Age. Even this information is piggybacked on either a HTTP request message or a HTTP response message. Therefore there is no extra connection setup between the communicating proxies. Hence there is 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 a centralized or distributed manager process [2, 7].

In short, the EA placement scheme is independent of cooperative cache architectures and cache replacement policies and its implementation does not have any overheads.

4. Experiments and Performance Results

In this section we investigate whether a document placement strategy utilizing cache state information can improve document hit rates and latencies? To answer this question we performed various simulation based experiments and measured performance of both Ad-hoc scheme and 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. Cumulative Hit Rate is the ratio of the total hits in the group to total number of requests in all the caches in the group. Similarly cumulative byte hit ratio is the ratio of bytes that hit in the cache group to the total number of bytes requested in the cache group. Average Cache Expiration Age is the mean of the Cache Expiration Ages of all the caches in the group. If a document is requested by a client at time T_R and the document is served at time T_S, latency for document is defined as $(T_S - T_R)$. Average latency is the average of the latencies experienced for all the documents served.

In addition to the simulation experiments reported here, we have mathematically analyzed both the ad-hoc and EA schemes. Our mathematical analysis demonstrates that the EA scheme utilizes the aggregate memory available in the group more effectively and also improves cache expiration ages of individual caches in the group. Due to space constraints we haven't been able to include the analysis in this paper. Interested reader is referred to [11] for a detailed mathematical analysis.

4.1. Experimental Setup

This subsection explains our experimental setup. We implemented a trace driven simulator to evaluate the new EA scheme against the conventional Ad-hoc scheme. The simulator is written in Java and simulates both the EA placement and the Ad-hoc placement 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.

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 middle of November 1994 to end of February 1995. The logs contain records of requests from 591 users over 4700 sessions. The number of records in the log is 575,775, out of which 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 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 2, 4, and 8 caches. Cooperative caching architecture of these cache groups is distributed cooperative caching. So all the caches in the group are at the same level of hierarchy. For any misses in the cache group, it is assumed that the cache where the request originated retrieves the document from the origin server. In our simulations we also assumed that all the caches in the cache group have equal amounts of disk space. Hence if the aggregate disk space available in the cache group is X bytes and if there are N caches in the group, the disk space available at each cache is $\frac{X}{N}$ bytes.

We ran the simulation for cache groups of 2, 4 and 8 caches respectively. For each of the cache groups we measured the above metrics when the aggregate memory in the group was 100KB, 1MB, 10MB, 100MB and 1GB respectively. The next section discusses the results obtained from our simulations.

4.2. Performance Results

In this subsection we discuss the results obtained from our trace based simulations. Although we have performed experiments with cache groups of 2, 4 and 8 caches, due to page limitations we provide the graphs and tables corresponding to the 4 cache group. Complete set of graphs and detailed discussion on the results can be found in our technical report [11].

First we compare the hit rates of ad-hoc placement scheme and EA scheme. Figure 1 indicates the hit rate of both schemes for cache group of four caches. We measured the hit rates at aggregate cache sizes of 100KB, 1MB, 10MB, 100MB and 1GB. It can be seen that the difference

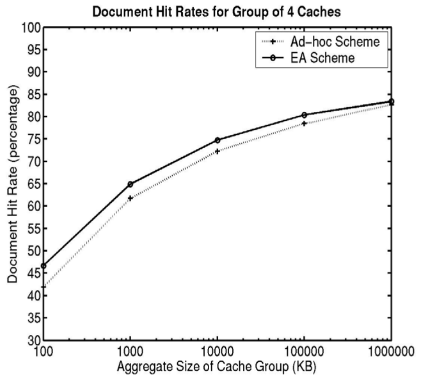

Figure 1. Document Hit Rates for 4 cache group

between the two schemes is higher when the cache sizes are smaller. There is about 6.5% increase in the document hit rate for a group of 8 caches when the aggregate memory is 100KB. Difference between the hit rates drops to 2.5% 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 memory yields substantial improvements in cache hit rates. This phenomenon has been discussed by many researchers. It can also be observed in our graphs. When the cache size is increased 10 times from 100KB to 1MB, the cache hit rate increases by almost 20%, whereas hit rate increases only by 3% when the cache size increases from 100MB to 1GB. Therefore when cache sizes are small, a better utilization of the memory yields substantial gains in terms of document hit rate. However at larger cache sizes, a better utilization of memory doesn't translate correspondingly into hit rates. However it performs slightly better than the ad-hoc scheme for large caches. In summary, the EA scheme yields significantly better performance when the cache sizes are limited.

Next we studied the effect of the EA scheme on cumulative byte hit rates. Figure 2 represents the byte hit rates for

Aggregate Memory	Conventional Scheme	EA Scheme
100K	3.21	4.03
1M	38.17	44.80
10M	446.91	507.40
100M	4570.96	5369.32

Table 1. Average Cache Expiration Age (in Secs) for 4 cache group

cache group of four caches. It can be seen that byte hit rate patterns are similar to those of document hit rates. For a group of 8 caches, improvement in byte hit ratio is approximately 4% when the aggregate cache size is 100KB and is about 1.5% when the aggregate cache size is 100MB.

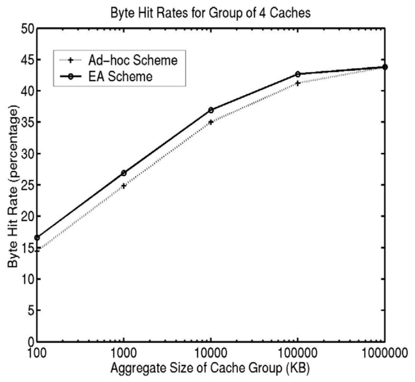

Figure 2. Byte Hit Rates for 4 cache group

To illustrate the reduced disk space in the cache group, we tabulate the Average Cache Expiration Age for both the Ad-hoc and for the EA scheme. Table 1 shows these values for cache group of 4 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 EA scheme the documents stay for much longer as compared with the Ad-hoc scheme. This demonstrates that EA scheme reduces disk space contention in the cache group.

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 [1]. We ran the experiments five

[1]If we closely observe cooperative caching, the hits can be of two types. The first type is what is referred to as a local hit. If the document is available in the same cache where the client request first came in it is a local hit. If the document is retrieved from other cache in the group it is a remote hit.

thousand 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 latency of a miss (ML) was 2784 milliseconds. To obtain latency values for misses, we measured latency values for various web sites and took their mean value.

Let LHR, RHR, MR represent the local hit rate, remote hit rate and miss rate and LHL, RHL, ML represent the local hit latency, remote hit latency and miss rate latency respectively. The average latency of requesting a document can be estimated by the following formula:

$$\text{Average Latency} = \frac{\text{LHR} \times \text{LHL} + \text{RHR} \times \text{RHL} + \text{MR} \times \text{ML}}{\text{LHR} + \text{RHR} + \text{MR}} \quad (6)$$

The graph in Figure 3 indicates the estimated average latency for group of four caches. 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 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, it is obvious that the remote hit rates under the EA scheme will increase. Table 2 shows the local and remote hit rates for a group of 4 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.

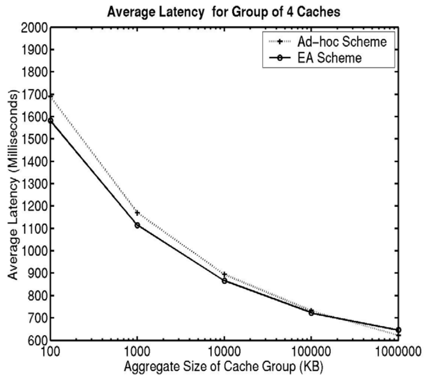

Figure 3. Estimated latency for 4 cache group

Second, when the cache sizes are small, the miss rates under the ad-hoc scheme is higher than that 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. This is simply because the latency for serving misses dominates the average document latency. 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. For example, in our experiments when the aggregate cache size was 1GB, the remote hit rate under the EA scheme was 32.02% whereas the remote hit rate of the ad-hoc scheme is only 11.06%. The difference between the miss rates of the two schemes is 0.6%. We observe that in general when the miss rates of the ad-hoc scheme is higher than the miss rates of the EA scheme, the EA scheme performs substantially better than the ad-hoc scheme.

If the document is not available in the cache group, it is a miss.

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 available memory in the group.

5. Related Work and Conclusion

The field of cooperative caching has been researched extensively. As mentioned earlier, ICP was introduced for efficient and fast communication. Summary cache [6] and adaptive web caching [9] are mechanisms to reduce or optimize the number of ICP messages among caches. Few other techniques such as hints, directories, hashing etc. have been proposed as alternatives to ICP in order to reduce the overall cost of document location [8, 16].

In addition to cache sharing protocols, basic research in cooperative proxy caching ranges from cooperative caching architectures [5] , cache coherence mechanisms, to cache placement and replacement schemes. Cooperative caching architecture provides a paradigm that assists proxies cooperate efficiently with each other. Current cooperative cache architectures are roughly classified as Hierarchical [5] or Distributed cache architectures [13]. Document placement schemes determine when and where to place a document in the cooperative cache. There are two models for document placement viz. *Lazy mode* - in which documents are retrieved and cached only when some clients request them and *eager mode* - in which documents are pre-fetched and cached based on access log predictions [10]. Document replacement algorithm decides which documents should be evicted from caches in order to make room for new documents entering the cache. While document placement for cooperative caching has not been well studied, a number of document replacement algorithms have been proposed [4, 17], which attempt to optimize various cost metrics.

Although lot of research efforts have gone into this field

	Ad-hoc Scheme			EA Scheme		
Aggregate Memory	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

Table 2. A Comparison of Ad-hoc scheme and EA scheme for a 4 cache group

of cooperative proxy caching, very few have considered document placement schemes and the potential improvement of such schemes on hit rates and document latencies.

In this paper we have presented an Expiration-Age based document placement scheme (the EA scheme). This new scheme 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 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 caching proxy is made collectively among the caches that already have a copy of this document. 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 trace-based simulations. The experiments 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.

Acknowledgment

This research was partially supported by a NSF grant under IIS and a DARPA ITO grant under the Information Expedition program. The second author is also partially supported by the Yamacraw Mission, State of Georgia.

References

[1] Internet cache protocol: Protocol specification, version 2, September 1997. http://icp.ircache.net/rfc2186.txt.

[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 Transaction on Computer Systems*, February 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. In *Proceedings of SDNE '95*, 1995.

[4] P. Cao and S. Irani. Cost-aware www proxy caching algorithms. In *Proceedings of USENIX Symposium on Internet Technologies and Systems*, 1997.

[5] A. Chankhunthod, P. Danzig, C. Neerdaels, M. Schwartz, and K. Worell. A hierarchical internet object cache. In *Proceedings of the 1996 USENIX Technical Conference*, January 1996.

[6] L. Fan, P. Cao, J. Almeida, and A. Broder. Summary cache: A scalable wide-area web cache sharing protocol. In *Proceedings of ACM SIGCOMM 98*, September 1998.

[7] A. R. Kalrin, H. M. Levy, and C. Thekkath. Implementing global memory management in workstation cluster. In *Proceedings of SOSP '95*, December 1995.

[8] D. Karger, T. Leighton, D. Lewin, and A. Sherman. Web caching with consistent hashing. In *Proceedings of Eighth International WWW Conference*, May 1999.

[9] 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*, November 1998.

[10] V. N. Padmanabhan and J. C. Mogul. Using predictive prefetching to improve world wide web latency. *Computer Communication Review*, July 1996.

[11] L. Ramaswamy and L. Liu A New Document Placement Scheme for Cooperative Caching on the Internet. Technical report, College of Computing, Georgia Tech. *GIT-CC-02-12*, February 2002.

[12] Squid internet object cache. http://squid.nlanr.net.

[13] R. Tewari, M. Dahlin, H. Vin, and J. Kay. Beyond hierarchies: Design considerations for distributed caching on the internet. In *Proceedings of ICDCS '99*, May 1999.

[14] J. Wang. A survey of web caching schemes for the internet. *ACM Computer Communication Review*, October 1999.

[15] 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. In *Proceedings of SOSP '99*, December 1999.

[16] K.-L. Wu and P. S. Yu. Latency sensitive hashing for collaborative web caching. *Computer Networks*, June 2000.

[17] N. Young. The k-server dual and competitiveness for paging. *Algorithmica*, June 1994.