RC22173 (W0109-015) September 14, 2001
Computer Science

IBM Research Report

Understanding the Potential Benefits of Cooperation among Proxies: Taxonomy and Analysis

Kang-Won Lee, Sambit Sahu, Khalil S. Amiri, Chitra Venkatramani
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Delhi - Haifa - India - T. J. Watson - Tokyo - Zurich

LIMITED DISTRIBUTION NOTICE: This report has been submitted for publication outside of IBM and will probably be copyrighted if accepted for publication. It has been issued as a Research Report for early dissemination of its contents. In view of the transfer of copyright to the outside publisher, its distribution outside of IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g. , payment of royalties). Copies may be requested from IBM T. J. Watson Research Center , P. O. Box 218, Yorktown Heights, NY 10598 USA (email: reports@us.ibm.com). Some reports are available on the internet at http://domino.watson.ibm.com/library/CyberDig.nsf/home .

Understanding the Potential Benefits of Cooperation among Proxies: Taxonomy and Analysis

Kang-Won Lee   Sambit Sahu   Khalil Amiri   Chitra Venkatramani
IBM Thomas J. Watson Research Center
Email: {kangwon, sambits, amirik, chitrav}@us.ibm.com

September 13, 2001

Abstract

Cooperation among proxies is potentially beneficial in improving the Web performance for two reasons: First, it reduces *capacity misses* by effectively increasing total cache size. Second, it reduces *compulsory misses* by opportunistically finding the missed object in another proxy's cache. While several studies have investigated the benefit of cooperative caching, they vary significantly in their conclusions.

The goal of this study is to determine the potential benefits achievable through cooperation among nearby proxies under various network configurations and user access behaviors. We begin this study by providing a taxonomy, which categorizes existing cooperative caching schemes into three broad forms based on the level of cooperation involved. We then provide a simple mathematical model to quantify the *upperbound* of performance gain achievable through cooperation. Finally, we use a trace-driven simulation to validate the upperbounds derived from the analysis, and show that the *actual gain* of cooperation is quite sensitive to the user access behavior.

Our results corroborate the findings of previous studies and explain their results in a unified analytical framework. In practice, the findings in this paper can be used to suggest the proper level of cooperation that can reap the most benefit of cooperative caching given the network configuration and user access behavior.

1 Introduction

Caching has long been employed in the World Wide Web as well as in distributed filesystems as an effective technique to reduce client latencies and save network bandwidth. Web proxies at the edge of the network cache most frequently accessed documents thereby avoiding wide-area network transfers and reducing client download latencies. To further improve document retrieval performance for web clients, several researchers have investigated the potential for cooperation among closely-located web proxy caches [1, 2, 3, 4, 5, 6, 7, 8, 9]. However, a closer examination of these studies reveals that there is no clear consensus as to whether cooperation among proxy caches results in any additional benefit. Our goal in this work is to provide a closer examination at various forms of cooperation that have been studied in the literature and to analyze the potential benefit of these various forms of cooperation.

Cooperation amongst proxies is potentially beneficial in two ways: First, it reduces *capacity misses* by effectively increasing the total cache size used to store documents [4, 7]. Second, it reduces *compulsory misses* by opportunistically finding the missed object in another proxy's cache [5, 9]. While the former benefit is likely to be of little use if working set size is small, current trends in the Web suggest that workloads may potentially require large cache sizes. For example, the advent of content distribution networks that cache and distribute documents on behalf of heavily accessed sites may require large cache sizes because they have a large actively accessed working set. Moreover, the popularity of rich media has contributed to the increase of average document size. For example, `bmwfilms.com`, a site featuring short promotional movies each requiring hundreds of megabytes of storage, attracted more than 200,000 unique visitors last week of May. Access to such large rich media objects is expected to increase even further as broadband networks, currently reaching only 10% of the homes, extend their reach to the majority of residential areas.

A large body of literature exists on the various aspects of cooperative caching in the Internet [1, 2, 3, 4, 5, 6, 7, 8, 9]. Rabinovich et al. [1] have focussed on cost-aware algorithms for cooperative proxy caching, while others [2, 3] have looked at the optimal object placement algorithm in a cooperative cluster and a content distribution network. Other researchers have focussed on mechanisms to reduce the overhead placed by cooperation [4, 5, 6]. The effectiveness of cooperative caching has been studied in the following literature: Wolman et al. [7] have studied the traces collected from multiple Web proxies, during a synchronous time interval, in two large organizations and concluded cooperative caching yields limited benefit. Dykes et al. [8] have reported that cooperative caching is marginally viable in meshes but not viable in hierarchies using an analytical model. Krishnan et al. [9] studied the utility of cooperation among caching proxies in a corporate Intranet environment and reported varied improvement in hit ratio from 0.7 to 28.9%. Korupolu et al. [6] report that cooperative caching can result in significant benefits and studied efficient implementation of such algorithms. To sum up, although several studies [6, 7, 8, 9] have investigated the benefit from cooperation among proxy caches, they vary significantly in their conclusions, due to their differing definitions of cooperation and assumptions on network configurations.

The goal of this study is to determine the potential benefits achievable through cooperation among nearby proxies under various network configurations and user access behaviors. In order to achieve this goal, we begin this study by providing a taxonomy for various forms of cooperative caching. This taxonomy categorizes the existing cooperative caching schemes in the literature into three broad forms based on the level of cooperation involved among proxies. We then provide a simple mathematical model to quantify the *maximum possible* performance gain (or the *upperbound* on gain) achievable through these various forms of cooperation categorized above. This analytical model provides us insights as to what influences the achievable benefits and why there is disagreement in the reported conclusions across the several studies. We use a trace-driven simulation to validate the upperbounds derived from the analytical results. We also show that the *actual gain* of cooperation is quite sensitive to "user access density," as discussed in Section 4.3.

Our results corroborate the findings of previous studies and explain their results in a general unified analytical framework. From the analysis and simulation, we find that the proper level of cooperation depends on the particular environment. In practice, we make the following suggestions on the deployment of cooperative caching:

1. Cooperation among proxies can be beneficial when the working set doesn't fit in a single cache. In particular, the benefit of cooperative caching is most eminent when the client-to-proxy or the inter-proxy latencies are much smaller than the proxy-to-server latency.

2. The level of cooperation required to reap most of the benefit is dependent on the user access behavior. If the interests of users served by the different proxies are highly diverse, a simple cooperation scheme based only on the redirection of misses to other proxies that have the requested object is sufficient.

3. On the other hand, if the user requests submitted to the proxies are homogeneous, then a more sophisticated level of cooperation can yield substantial additional benefit over simple redirection. This level requires a proxy to consider the current state of other proxies and how often they access locally cached documents when making a replacement decision.

The remainder of the paper is organized as follows. We provide our taxonomy for cooperative caching in Section 2 and build our analytical model in Section 3. We then present the simulation results in comparison with the analytical results under various scenarios in Section 4. Finally, we conclude this paper by proposing a practical guideline for the deployment of cooperative caching in Section 6.

2 Taxonomy for cooperative caching

2.1 Cluster-based cooperative proxy caching architecture

Figure 1 illustrates a typical cluster-based cooperative proxy caching architecture. We define a set of cooperating proxies as a *cooperative cluster*. Each client has a designated proxy, called *local proxy*, through which

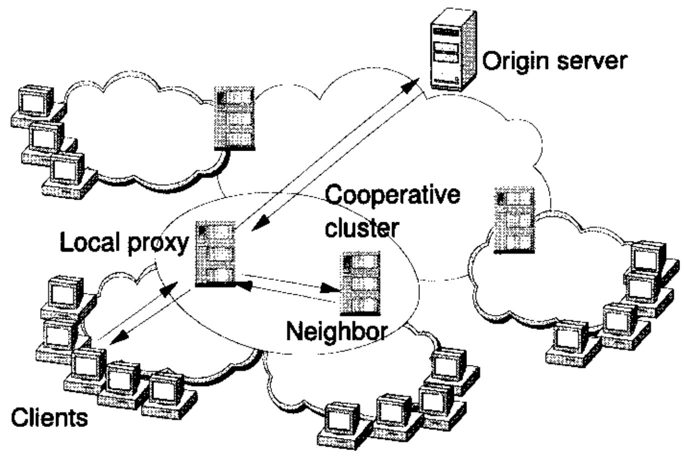


Figure 1: Cluster-based cooperative proxy caching architecture

it makes all of its HTTP requests. The other proxies in the same cluster are called *neighbors*. The HTTP requests from clients are first processed by the local proxy, which attempts to serve them from its local cache. In case of a miss, when there is no cooperation, the local proxy forwards the request to the origin server. If some form of cooperation is employed, an attempt is first made to serve the request from one of the neighbors in the cluster. If the object is not found in any neighbor, the request is redirected to the origin server. The local proxy can optionally cache the response and pass it to the client.

It must be noted that grouping of proxies into clusters is orthogonal to this study. Grouping can be determined by performance constraints, such as the latencies between the proxies, as well as administrative constraints. In this paper, we shall assume that a cooperative cluster has been already defined and given, then examine when and what level of cooperation would yield practical benefits. The results of this paper, however, can be used to guide a performance-based policy of clustering proxies into cooperative groups.

2.2 Taxonomy of cooperative caching

The proxies in a cluster can choose to coordinate their caching policies according to various degrees of cooperation. We distinguish two key aspects of cooperation that can be used to classify cooperative caching schemes. The first aspect is the level of coordination used in serving a requested object. The second is the level of cooperation employed in object replacement decisions. Based on these two aspects, we can classify cooperative caching algorithms into the following taxonomy:

- **No cooperation (NC)**: In this case, if the requested object is not available in the local proxy, it is fetched from the origin server. Each proxy acts independently without knowledge of other neighboring proxies. This case refers to no cooperation in either request redirection or object replacement and is taken to be the baseline case against which the other cooperation schemes are compared.

- **Cooperative lookup with selfish replacement (CSR)**: In this case, proxies cooperate in serving a request, but not in making object replacement decisions. More precisely, if a requested object is not found in the local proxy, it attempts to fetch the object from a neighbor within the cluster. Otherwise, the request is directed to the origin server. During object replacement, the local proxy relies only on local access information to choose an object, such that the average request response time of its clients is minimized [4, 5, 7, 8, 9].

- **Cooperative lookup with cooperative replacement (CCR)**: In this case, the same level of cooperation is followed in request redirection as in CSR. In addition, during object replacement, the local proxy makes its replacement decision taking into account requests to its objects from clients of its neighbors as well as the requests from its own clients. The decision is based on the goal of minimizing average

	Cooperative Lookup	*Cooperative Replacement*	*Replacement Scope*
NC	no	no	local cache
CSR	yes	no	local cache
CCR	yes	yes	local cache
CGR	yes	yes	entire cluster

Table 1: A taxonomy of cooperative caching algorithms.

access latency for requests received by all proxies in the cluster. A proxy can replace only its own cached objects, however, and cannot choose a victim object from the cache of a neighbor [1, 6].

- **Cooperative lookup with global replacement (CGR)**: The same level of cooperation as CSR or CCR is adopted in serving a request, and cache replacement is based on the goal of minimizing average access latency of all proxies in the cluster as in CCR. However, unlike CCR, object replacement is not restricted to choosing a victim from the local cache. Instead, a proxy can choose an object from any neighbor for eviction, if that reduces average request latency across accesses in the entire cluster [2, 3, 6].

Table 1 summarizes the key differences among these four schemes. In the above classification, NC employs zero cooperation among the proxies while CGR uses maximum possible cooperation.

Ignoring the associated overhead in its implementation, CGR is the ideal form of cooperative caching as it leverages global knowledge both in serving requests and in replacing objects. The policies considered in [2, 3, 6] can be classified as CGR based on the cooperation among the proxies. While in theory CGR achieves the best possible benefit of cooperative caching, it suffers from two major practical problems. First, it is computationally very expensive as its object replacement algorithm considers every object in the cluster as a potential candidate for replacement and it requires global frequency knowledge. Second, due to administrative restrictions, it may not be feasible to allow one proxy to evict objects from another. Together, the implementation overhead and the administrative restrictions make CGR unattractive in practice.

While NC, CSR and CCR are all feasible in practice, the information tracked and the computational complexity of the schemes increase as they involve more cooperation. We do not know apriori the additional benefit of each scheme over the other and whether it warrants the extra implementation complexity. While there have been several studies [6, 7, 8, 9] reporting the performance of cooperative proxy caching, there is no consensus on what level of cooperation is actually beneficial under what circumstances. In this study, we set out to *determine which level of cooperation provides the most practical benefit under a variety of workloads and network configurations using an analytical model and simulation.*

2.3 Algorithms for NC, CSR, and CCR

In order to answer the above question, we first present the abstract algorithms for implementing NC, CSR, and CCR.

We assume the following: The inter-proxy latencies, as well as the contents of neighbor proxies, are known to each proxy in the cluster. Also, in the CCR scheme, each proxy has perfect knowledge of the frequency of access for each cached object in its local cache from each client group of neighbors. Finally, the goal of all the caching schemes is to minimize the average access latency for their client populations. More precisely, for NC and CSR, the goal of each proxy is to minimize access latency for its immediate client group; whereas, for CCR, the goal is to reduce total access latency for the entire client population.

Given these assumptions, we can describe the implementation of each cooperative caching schemes. A proxy using NC or CSR simply uses an LFU-based replacement algorithm, which minimizes total access latency for its client group assuming the perfect frequency knowledge. On a cache miss, however, their behavior differs: NC forwards the request to the origin server, where as CSR first looks up its directory of the cluster cache content. If the object exists in the cluster, it forwards the request to the closest neighbor. Otherwise, it forwards the request to the origin server.

The CCR scheme behaves like CSR on a cache miss, fetching the object from the closest neighbor if possible. However, since its goal is to minimize total access latency for the *entire* client population, its cache replacement policy is more complex.

To describe the cache replacement policy for CCR, a few more formal notations are required. We define "aggregate average latency" for CCR as the average access latency over the entire client population. This is given by $\sum_{x=1}^{M}\sum_{i=1}^{n} f_x(i)\tau(i)$, where $f_x(i)$ denotes the access frequency of object i in proxy x, and $\tau(i)$ denotes the average latency to access object i in the cluster.

An implementation of CCR that minimizes the above objective function can be described as follows. Suppose there is a miss for object i at proxy x. If proxy x has to replace an object to bring in this object i, it evaluates the cost of replacing an object and the benefit of bringing in object i using the following cost-benefit comparison. We define the cost of removing an object j from a proxy x as:

$$C_x(j) = \sum_{y=1}^{M} \delta_y(x,j) f_y(j) \tag{1}$$

where $\delta_y(x,j)$ is given as

$$\delta_y(x,j) = \begin{cases} 0 & \text{if } j \in y \\ [\min_{w \neq x, j \in w}\{l_{y,w}\} - l_{y,x}]^+ & \text{otherwise} \end{cases} \tag{2}$$

where $[z]^+ = \max\{z, 0\}$, and $l_{x,y}$ denotes the latency between proxies x and y.

Note that the function $\delta_y(x,j)$ computes the increase in effective latency for proxy y to access object j, due to the removal of object j from proxy x. The product of $\delta_y(x,j)$ and frequency $f_y(j)$ thus describes the increase in cost, or the penalty, observed by proxy y due to x's eviction of object j. Therefore, the sum $C_x(j)$ corresponds to the total cost perceived by all the proxies in the cluster due to the removal of object j from proxy x.

Similarly, we can compute the benefit for bringing in a new object for all the proxies that will access this particular object from proxy x. Let us denote the benefit function as $B_x(i)$. Let $\tilde{j}$ denote the object such that $C_x(\tilde{j}) = \min_{1 \leq j \leq k} C_x(j)$. The proxy x replaces object $\tilde{j}$ with the new object i iff $B_x(i) - C_x(\tilde{j}) > 0$.

Based on the assumptions of the perfect frequency knowledge to each object and the fixed latencies among proxies, the above cost-benefit-based replacement minimizes the aggregate average latency as presented in Theorem 1.

Theorem 1 (Optimality) *Let $C_x(\tilde{j})$ denote the minimum penalty in the aggregate latency as a result of evicting object $\tilde{j}$ from proxy x. Let $B_x(i)$ denote the effective reduction in aggregate latency as a result of placing object i in proxy x. If $B_x(i) - C_x(\tilde{j}) > 0$, then the decision to replace object $\tilde{j}$ by object i minimizes the aggregate access latency.*

Proof We prove that our algorithm minimizes aggregate access latency by contradiction. We need to prove that (i) if a different object $\hat{j}$ is chosen instead of $\tilde{j}$ for replacement, or (ii) if no replacement is done, it will result in increased average access latency.

Let us now prove the fact in (i). Note that $C_x(\hat{j}) > C_x(\tilde{j})$. It is easy to verify that the objective function $\sum_{x=1}^{M}\sum_{j=1}^{n} f_x(j)\tau(j)$ is higher with removal of object $\hat{j}$ than with object $\tilde{j}$. Similarly, for (ii), when $B_x(i) - C_x(\tilde{j}) > 0$, it is easy to show that $\sum_{x=1}^{M}\sum_{j=1}^{n} f_x(j)\tau(j)$ is higher if $\tilde{j}$ remains in proxy x than the case when it is replaced by object i.
□

3 Analysis of cooperative caching: achievable benefits

In the previous section, we presented algorithms for NC, CSR and CCR schemes that minimize the average request latency according to their respective scopes. In this section, we provide a simple mathematical model to estimate the relative benefit of these different algorithms. For the simplicity of exposition, we first present the analysis to the case of two cooperating proxies. Then we generalize the result to the case of M cooperating proxies in a cluster. For both cases, our focus in this section is on the scenario when the object access patterns are homogeneous. Then in Section 3.3, we present a simple model that captures the heterogeneity in user access.

3.1 Analysis with homogeneous access: the case of two cooperating proxies

We denote by τ_l, τ_c, τ_s the average access latency between clients and their local proxies, between two neighboring proxies, and between proxies and the origin server, respectively. For simplicity, we assume that there is only one origin server and each proxy is equi-distant from the origin server. Using this notation, if an object was found in the local proxy, the client-perceived latency to the object is given by τ_l. If the object is found in a neighbor proxy, the latency can be expressed as $\tau_l + \tau_c$. Similarly, if the object is fetched from the origin server, the access latency is given by $\tau_l + \tau_s$.

We further assume that all the requested objects are of equal size, and each proxy has the same size and can cache up to k objects. We also assume that object popularity doesn't change over time and, without loss of generality, $f(i) > f(j)$ for any $i < j$, i.e., the objects are numbered in the order of their popularity rank (or access frequency) with smaller id's corresponding to more popular objects.

Finally, we assume that the access frequency to objects is the same at all the proxies. Intuitively, this means that clients accessing different proxies are statistically identical in their access pattern. Thus, the most popular object in one proxy is the same in another. We call such an access pattern, a "homogeneous access." More precisely, if we denote by $f_x(i)$ the access frequency of object i at proxy x, homogeneity implies that $f_x(i) = f_y(i)$ for any object $i = 1, \ldots, n$ and for any proxies x, y. We restrict our attention initially to a homogeneous access pattern because it is simpler to analyze. The later part of this section discusses the case of a heterogeneous access pattern.

3.1.1 Access latency with CSR and NC

Let us examine the caching behavior of a two-proxy cluster in the steady state. It is easy to show that in the steady state, NC would result in the k most popular objects being cached at each proxy. By caching the most popular objects to its local clients, it can minimize the average access latency in its scope. Furthermore, because of the homogeneity assumption, the two proxies would contain exactly the same set of objects. As a result, CSR, which is simply NC augmented with the ability to redirect misses to the other proxy, would observe no additional benefit from such redirections. Because the caches have the same contents, a miss in one proxy cache is a miss in the other. Therefore, there will be exactly the same k distinct objects in each proxy in the cluster of $2k$ slots. Thus, the average latency for NC and CSR is given by:

$$L_{CSR} = L_{NC} = \tau_l \sum_{i=1}^{n} f(i) + \tau_s \sum_{i=k+1}^{n} f(i) \tag{3}$$

The first term in Equation 3 represents accesses to local proxy, for both cached and uncached objects. The second term corresponds to the latency to access the origin server due to cache misses, for all the objects other than the k most popular ones.

3.1.2 Access latency with CCR

Computing the average access latency under CCR is much more involved. In this analysis, our aim is to construct the cache content of the ideal CCR algorithm in steady state. To construct this state, we use an *artificial* process that effects a transition of cache content from steady-state CSR (or NC) to the steady-state of an ideal CCR by removing duplicates based on CCR's cost-benefit equations. We call this the *duplicate removal process or DRP*. This process is based on the intuition that, in steady state, CCR would have brought in new objects, that are less popular than the top k objects, into the cluster replacing duplicate objects in order to reduce the aggregate average latency.

We stress that DRP is an artificial process that transforms CSR's cache content into CCR's cache content in steady state. During this transform, DRP bases its decision process on the perfect frequency knowledge of objects. Thus, the resulting cache content yields the optimal aggregate latency under CCR. In a natural replacement process, a proxy employing CCR must *estimate* the popularity of objects *on the fly* based on the requests it sees. As a result, the CCR's performance computed by DRP will provide an upperbound on the actual performance of a practical CCR implementation.

Now let us examine which duplicate objects are removed and which objects are fetched to replace them. DRP proceeds in a sequence of steps. During each step, it makes the replacement decision that yields the highest incremental benefit. The process terminates when average latency cannot be reduced through further replacements.

To estimate the total reduction in average latency after DRP terminates, we prove two key properties about DRP. The first property concerns the order of replacement performed by DRP. The second property concerns the optimal number of replacements performed by DRP under homogeneous access.

Let us first focus on the order of object replacement. We show that DRP will always replace duplicate objects in the order of increasing access frequency and will bring in the objects with the most popular object that are *not* already in the cluster.

Lemma 1 (Object Replacement) *DRP will only replace duplicate copy of an object from the cluster such that an object i is always replaced before an object j with $f(i) < f(j)$. In addition, it will first bring in a new object that has the highest frequency that is not already present in the cluster, i.e., an object s will never be brought into the cluster if there exists an object t such that $f(t) \geq f(s)$ and object t is not currently in the cluster.*

Proof To prove the above lemma, we prove the following three properties:

(i) DRP will only replace objects from the set of objects that are duplicated in both proxies.

(ii) DRP will always choose the duplicate copy of an object with the lowest access frequency first.

(iii) DRP will always bring in an object that has the highest access frequency first.

Case (i): It is easy to prove this as $f(s) > f(t)$ for any $s < t$. Let us examine the case what happens in case DRP replaces a cached object s that does not have any duplicate copy in the cluster with an object t that is not already in the cluster. Since $s < k$ and $t > k$ (where k is the size of each proxy), it follows that $f(s) > f(t)$. Let the average latency before the replacement is denoted by L' and after the replacement by L''. It is easy to show that $L'' - L' = (\tau_s - \frac{\tau_c}{2})(f(s) - f(t)) > 0$ as $\tau_s > \tau_c$ and $f(s) > f(t)$. It implies that replacement will lead to increase in average access latency. Thus DRP will not replace an object that does not have a duplicate copy in the cluster. But replacing one of the duplicate objects by a less frequently accessed uncached object can reduce aggregate average latency, for in this case $L'' - L' = \frac{\tau_c}{2}f(s) - (\tau_s - \frac{\tau_c}{2})f(t)$ which can be smaller than zero depending on the parameter values.

Using the similar argument as in the proof of (i), it is easy to prove (ii) and (iii).

□

Next, let us examine how many duplicate objects will be replaced by new objects with DRP. Since DRP follows CCR's cost-benefit equation, ideally it will keep replacing duplicate objects as long as bringing in the next uncached object reduces aggregate average latency.

Lemma 2 (Optimal Replacement) *Let $L(j)$ denote the average latency when j objects are replaced with new objects by DRP. Then DRP will replace exactly j number of duplicate objects from the most popular objects iff $L(j-1) - L(j) \geq 0$ and $L(j) - L(j+1) < 0$.*

Proof From Theorem 1, we know that CCR achieves minimum aggregate average latency by its optimal object replacement algorithm. Since DRP makes transitions to arrive at steady-state CCR's cache content, starting from steady-state CSR's cache content, it will keep replacing duplicate objects until there is no further decrease in aggregate latency. Thus in order to prove the fact in Lemma 2, we need to show that there is only one minimum in the function $L(i)$ and it occurs at $i = j$. In other words, we need to prove that $L(i)$ is a strictly convex function.

We prove the above fact using mathematical induction. Let the aggregate access latency increase for the first time after j replacements, i.e., $L(j+1) - L(j) \geq 0$ with $L(j) - L(j-1) < 0$. We show that for any $i > j$, $L(i) - L(i-1) > 0$. In other words, once the access latency increase with a replacement, it always increases if more objects are replaced. Let it be true with i replacements, i.e., $L(i+1) > L(i)$ for any arbitrary $i > j$. We show that $L(i+2) > L(i+1)$. Let the object that is replaced on i^{th} replacement be denoted as r_i and the object that is brought in as b_i. Now we write $L(i+2)$ in terms of $L(i+1)$, r_{i+2} and b_{i+2}. Note that $L(i+2) = L(i+1) + \frac{\tau_c}{2} f(r_{i+2}) - (\tau_s - \frac{\tau_c}{2}) f(b_{i+2})$. From the condition that $L(i+1) > L(i)$, we know that $\tau_c f(r_{i+1}) - 2\tau_s f(b_{i+1}) + \tau_c f(b_{i+1}) > 0$. Rewriting it, we have $\tau_c f(r_{i+1}) > (2\tau_s f(b_{i+1}) - \tau_c) f(b_{i+1})$. To prove the induction, we need to show that

$$\tau_c f(r_{i+2}) > (2\tau_s f(b_{i+2}) - \tau_c) f(b_{i+2}). \tag{4}$$

From Lemma 1, we know that $f(r_{i+2}) > f(r_{i+1})$ and $f(b_{i+2}) < f(b_{i+1})$. Thus $\tau_c f(r_{i+1}) > (2\tau_s - \tau_c) f(b_{i+2})$. As $f(r_{i+2}) > f(r_{i+1})$, it is then true that $\tau_c f(r_{i+2}) > (2\tau_s f(b_{i+2}) - \tau_c) f(b_{i+2})$.
□

From the results in Lemmas 1 and 2, the aggregate average access latency after j^{th} replacement can be written as follows:

$$L(j) = \tau_l \sum_{i=1}^{n} f(i) + \frac{\tau_c}{2} \sum_{i=k-j+1}^{k+j} f(i) + \tau_s \sum_{i=k+j+1}^{n} f(i). \tag{5}$$

Applying the results in Lemma 2, we obtain the following: The condition $L(j-1) - L(j) \geq 0$ results in $2\tau_s f(k+j) - \tau_c\{f(k-j+1) + f(k+j)\} \geq 0$. Similarly, $L(j) - L(j+1) < 0$ results in $2\tau_s f(k+j+1) - \tau_c\{f(k-j) + f(k+j+1)\} < 0$.

3.1.3 Results with Zipf popularity distribution

There are several empirical studies that have shown the popularity of HTTP objects [10, 11, 12] and streaming multimedia objects [13] follow Zipf-like distribution, i.e. the frequency of request to the i^{th} popular object is proportional to $\frac{1}{i^\alpha}$, where α is the parameter that determines the slope of the popularity distribution curve. When $\alpha = 1$, we have a closed form solution for j by solving the above inequalities.

$$k - \zeta(k + \frac{1}{2}) < j \leq k - \zeta(k + \frac{1}{2}) + 1. \tag{6}$$

where $\zeta = \tau_c/\tau_s$. Note that Equation 6 describes a tight bound on the value of j, the number of additional distinct objects that are brought into the cluster as a result of DRP. We illustrate the benefit of using CCR when the object access is described by a Zipf-like distribution in Section 4.

3.2 Homogeneous object access: the case of M cooperating proxies

For the analysis in this section, we make the following assumptions. We first assume that the latencies between any two proxies in the same cluster are the same and are denoted by τ_c as in the two proxy case. Later in the paper, we examine various cluster configurations and find that the performance of cooperative caching doesn't get affected much by specific inter-proxy latency allocations, as long as the average inter-proxy latency τ_c remains the same.

As in the two-proxy case, we assume that all the requested objects are of same size, and each proxy has the same size and can cache up to k objects. We also assume that object popularity doesn't change over time and $f(i) > f(j)$ for any $i < j$. Finally, we assume the "homogeneity" in object access as in the previous case, i.e., $f_x(i) = f_y(i)$ for any object i and for any proxies x, y.

3.2.1 Access latency with CSR and NC

Following the same logic as in the two proxy case, we can easily show that the average latency for NC and CSR is given by:

$$L_{CSR} = L_{NC} = \tau_l \sum_{i=1}^{n} f(i) + \tau_s \sum_{i=k+1}^{n} f(i) \tag{7}$$

The first term in the equation represents accesses to local proxy, for both cached and uncached objects. The second term represents the latency to access the origin server due to cache misses, for all uncached objects.

3.2.2 Access latency with CCR

In order to calculate the average access latency of CCR among M cooperative proxies, we need to extend the *duplicate removal process (DRP)* presented in the previous section. In particular, we generalize the object replacement procedure as follows.

Lemma 3 (Object Replacement (M proxy case)) *DRP will first replace duplicate copies of object i until there is no further duplicate objects in the cluster, i.e., until there is only one copy of object i left in the cluster, before replacing object j if $f(i) < f(j)$. For $i_1 \ldots i_M$, copies of object i, the increase in the average access latency by replacing any copy of i_m from the cluster is the same.*

Sketch of Proof The lemma consists of two part: (i) DRP will replace duplicate copy of object i until there is only one copy left in the cluster before replacing other object j when $f(i) < f(j)$, and (ii) the order of replacement among duplicates of object i doesn't affect the increase in average access latency.

The first point can be shown following the same logic that was used to prove Lemma 1. The second point is trivial since $f(i_1) = \ldots = f(i_M)$ and we assume the inter-proxy latencies between any two proxies are equal. Therefore, removing any duplicate copy of i will incur the same amount of latency increase.
□

Lemma 2 still holds in the M proxy case, and its proof is a straightforward extension from the two proxy case.

Based on Lemma 3 and Lemma 2, we can express the average access latency with CCR after j object replacement as follows:

$$L(j) = \tau_l \sum_{i=1}^{n} f(i) + \tau_c \{ \frac{m-1}{M} f(k-l) + \frac{M-1}{M} \sum_{i=k-l+1}^{k+j} f(i) \} + \tau_s \sum_{i=k+j+1}^{n} f(i) \tag{8}$$

where k represents the size of each individual cache, $l = \lfloor \frac{j}{M-1} \rfloor$, and $m = j \bmod (M-1)$.

The first term represents latency to access all objects and the third term represents additional latency to access uncached objects. The second term represents latency to access objects in neighboring proxies. In particular, the first part of the second term represents the average latency to access object l when m copies of object l has been replaced with new objects, and the second part represents the average latency to access objects that are less popular then object l. Note that when $M = 2$ the above equation reduces to Equation 5.

3.3 Heterogeneous object access

In the previous section, we assumed that the object access is homogeneous across all the proxies. In this section, we examine how the benefit of cooperative caching is affected as a result of non-homogeneity in the object access across the proxies. Before we derive the model for computing aggregate latency with heterogeneous access, we first present an important result, stating that the maximum benefit under CCR is achieved when the access pattern is homogeneous:

Theorem 2 (Condition of Maximum Benefit) *Let the object access is defined to be homogeneous when $f_x(i) = f_y(i)$ for any object $i = 1, \ldots, n$ and for any proxies x, y. With CCR, maximum reduction in aggregate average latency over CSR and NC occurs when the object access is homogeneous across proxies.*

Proof We need to show that whenever the object access is heterogeneous, the average latency gain by CCR over CSR and NC is smaller than if the access pattern was homogeneous. We recall that DRP replaces a duplicate object and brings in a different one that is not already present in the cluster. In homogeneous case, with NC, all cached object are duplicates and have the same rank in both proxies. Let us now examine the case with heterogeneous object access. Unlike homogeneous case, there are two more cases that can happen in heterogeneous case: (i) the same objects are cached in both proxies, but they may differ in their access popularity (or in the rank within each proxy), (ii) the objects in both proxies may differ. We have to show that benefit of CCR over NC in both the cases is smaller compared to that with homogeneous access.

Case 1: Consider the object with i^{th} rank (i.e., i^{th} popular object) in one of the proxies. Denote this as object i. Let this object has rank j in the other proxy and assume that $j > i$. If the CCR scheme in the first proxy decides to remove the object i as it sees there is a duplicate copy in the second proxy, it will incur higher penalty due to object removal. This is because $f(j) < f(i)$. If it were the other way around, the result will be effectively the same as the case with homogeneous access. Thus the incremental benefit with heterogenous access will be at the best same as homogeneous access.

Case 2: In this case, the number of duplicate objects is strictly less than the homogeneous case. Thus the number of objects that can be removed in order to bring in new objects will be less compared to the homogeneous case. Also we note that the relative benefit of replacing the subsequent duplicate and bringing in the next uncached object will be smaller as well. Both these factors together will result in lower incremental gain as compared to homogeneous access.

Using the similar argument, we can show that latency reduction of CCR over CSR is maximum when the access pattern is homogeneous.

□

Note that Theorem 2 indicates that the incremental benefit of CCR will be less with heterogeneous access. We now need to quantify the aggregate access latencies of CCR, CSR, and NC under heterogeneous access. In order to incorporate heterogeneity in access into our model, we first need to quantitatively define heterogeneity. Note that, unlike the homogeneous case, there is no single answer what heterogeneity access is. While there are possibly more sophisticated models for capturing the heterogeneity in object access, we consider the case in which the top γ percentage of the most popular objects be homogeneous in both proxies, while the rest of the objects differ from each other. This model refers to the case when most popular objects are likely to be the same everywhere. Henceforth, we refer to this model as the one-step heterogeneity model.

It is easy to show that the average latency with NC, under this heterogeneity model, is given as follows:

$$L_{NC} = \tau_l \sum_{i=1}^{n} f(i) + \frac{\tau_s}{2} \sum_{i=k*\gamma+1}^{(2-\gamma)k} f(i) + \tau_s \sum_{i=(2-\gamma)k+1}^{n} f(i) \tag{9}$$

Similarly, the average latency with CSR, under the one-step heterogeneity model, is given by:

$$L_{CSR} = \tau_l \sum_{i=1}^{n} f(i) + \frac{\tau_c}{2} \sum_{i=k*\gamma+1}^{(2-\gamma)k} f(i) + \tau_s \sum_{i=(2-\gamma)k+1}^{n} f(i) \tag{10}$$

We now compute the average latency with CCR under this heterogeneity model. Note that there are $k * \gamma$ duplicate copies that can be replaced by DRP to bring in objects that are currently not in the cluster. It is easy to show that the order of object replacement and condition on optimal number of object removal is the same as the homogeneous case for the $k * \gamma$ duplicate objects. Thus the aggregate average latency when j duplicate objects are replaced with new objects is given by:

$$L(j) = \tau_l \sum_{i=1}^{n} f(i) + \frac{\tau_c}{2} \sum_{k*\gamma-j+1}^{(2-\gamma)k+j} f(i) + \tau_s \sum_{(2-\gamma)k+j+1}^{n} f(i) \tag{11}$$

Solving for simultaneous equations $L(j-1)-L(j) \geq 0$ and $L(j)-L(j+1) < 0$ gives the number of duplicate objects replaced. We derive the closed form solution for a Zipf-like object popularity with $\alpha = 1.0$ case. It can be shown that for this case, $k * \gamma - \zeta(k + \frac{1}{2}) < j \leq k * \gamma - \zeta(k + \frac{1}{2}) + 1$, where $\zeta = \frac{\tau_c}{\tau_s}$.

4 Performance Analysis

In this section, we study the performance gain due to CCR and CSR over NC using the analytical results derived from the latency estimation presented in the previous section, in comparison with the results from a trace-driven simulation. We consider a wide range of values for various parameters that model user access behavior and cluster configuration in the two-proxy cluster shown in Figure 1.

We recall that the performance gain computed by the analysis provides an *upperbound* on the actual gain achievable through cooperation, because DRP makes idealistic assumptions, such as perfect knowledge of object popularity and steady state operation, for the simplicity of analysis. Nonetheless, we find that the performance gain obtained by the simulations can closely reach the upperbound predicted by the analysis, under certain user access characteristics. These comparisons are presented in Section 4.2.

Next, we demonstrate that the achievable gain of CCR is sensitive to the *density* of user access. In particular, we show that CCR gain in simulation deviates from the analytical upperbound as the number of distinct objects requested in the same amount of accesses increases. A measure to characterize the user access density and the its effects on CCR gain are elaborated in Section 4.3.

This paper quantifies the performance improvement due to cooperative caching by the relative gain in latency. More precisely, we denote the performance improvement due to CCR over NC as:

$$\Gamma_{\frac{CCR}{NC}} = \frac{L_{NC} - L_{CCR}}{L_{NC}}.$$

Similarly, we denote the performance improvement due to CSR over NC as:

$$\Gamma_{\frac{CSR}{NC}} = \frac{L_{NC} - L_{CSR}}{L_{NC}}.$$

Note that $0 \leq \Gamma \leq 1$, where $\Gamma = 0$ when there is no benefit and $\Gamma = 1$ when there is maximum benefit, or the resulting latency is zero. In the rest of the paper, we present all the results in terms of Γ or the *latency gain.* The actual latency numbers can be found in [14]. In all the performance plots, the *x-axis* represents the relative cache size with respect to the number of distinct objects requested, and the *y-axis* represents the latency gain Γ.

We start this section with a brief description of our frequency estimation algorithm, followed by a comparison of the analytical results with those from simulations under various parameters.

4.1 Simulator design and frequency estimation

As discussed in Section 2, NC and CSR use a LFU-based cache replacement policy. Similarly, CCR employs a cost-based cache replacement as described in Section 2 that utilizes the estimated frequency and latency information. Thus, a good frequency estimation technique is important for all three schemes.

In our implementation, we track object frequency as follows. We associate a last access time with each object that was ever accessed. Upon a new access to the same object, the inter-access time is computed by taking the difference between the current time and the last access time. If inter-access time was constant, then the inverse of this inter-access time would represent the exact "frequency" for that object. In practice, since inter-access time varies, we keep a time-decaying average of the inverse of the inter-access time and use it as an estimate of the frequency. That is, the running estimate of an object's frequency, $f_{history}$ is updated upon an access occurring at time t_{now} to be: $f_{history} = (1 - \eta) * \frac{1}{(t_{now} - t_{prev})} + \eta * f_{history}$.

where t_{prev} is the previous access time, and η is the decay factor. The default value of η in our simulation was set to 0.75 for graceful decay of history. We found that our results are not very sensitive to the choice of decay factor value across a wide range of η, from 0.5 to 0.95.

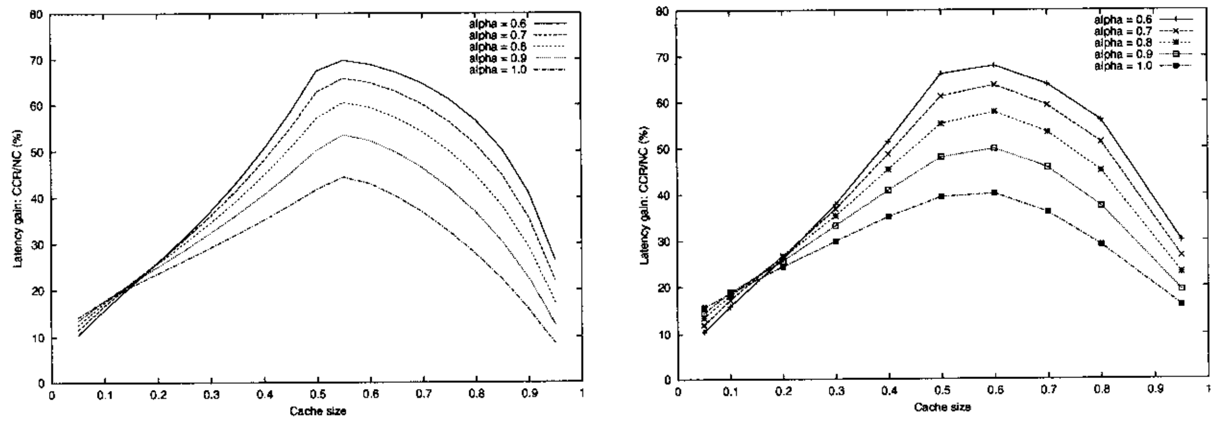
Lastly, we note that the user accesses were synthetically generated using a method similar to [11]. We have deliberately excluded such parameters as the percentage of uncacheable objects, because we are only concerned with the relative gain due to cooperation over non-cooperation for cacheable objects. Ongoing research will validate our results using real proxy traces.

4.2 Baseline results

Now we present the results from our analysis and simulation under a wide range of values for the parameters that characterize the user access behavior and network configuration. In particular, we model user access behavior using the Zipf parameter (α), and model network configuration using the ratio of the proxy-to-server latency to proxy-to-proxy latency (τ_s/τ_c) and the ratio of the proxy-to-server latency to the client-to-proxy latency (τ_s/τ_l). When varying one of the parameters, we fix the other parameters to their default values, which are $\alpha = 0.7, \tau_s/\tau_c = 10$, and $\tau_s/\tau_l = 20$. For simulations in this section, a total of 1 million requests were generated over 10,000 distinct objects. More diverse user access patterns are considered in the next section, where we discuss the impact of user access density on the performance cooperative caching.

4.2.1 The Zipf-like distribution parameter α

One of the key parameters that characterize user access pattern is the Zipf parameter α, which determines the slope of the object popularity distribution. When α is high, user accesses are concentrated to a relatively small number of popular objects; when α is low, user accesses are distributed among a relatively large number of objects. The empirical values of α reported in most studies lie in a relatively tight region between 0.65 and 0.9 [10, 11, 13, 15, 12].



(a) Analysis (b) Simulation

Figure 2: The latency gain $\Gamma_{\frac{CCR}{NC}}$ with respect to α. The cooperative caching gain increases as α decreases., i.e., as the working size increases.

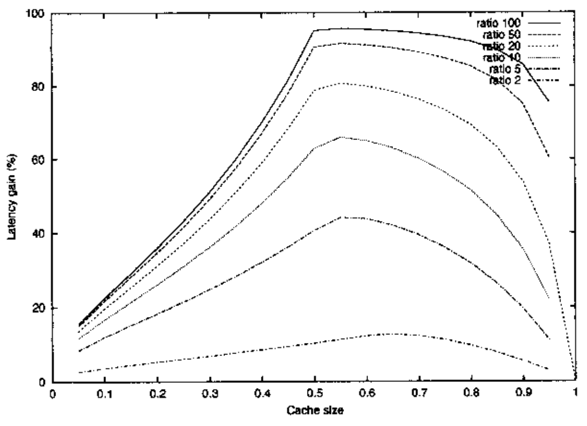
Figure 2 plots the latency gain $\Gamma_{\frac{CCR}{NC}}$ versus the Zipf parameter α obtained from the analytical model and simulations. Overall, the figures show a close agreement of the analytical results (the upperbound of gain) and the simulation results (the actual gain). Thus, we validate that our analysis makes reasonably good predictions on the impact of α in this scenario. From the figures, we find that the potential gain of cooperative caching increases with the decrease of α. This must be due to the following reason: when α is low, as we have mentioned above, user requests are distributed among a relatively larger number of objects, i.e. the working set is larger, than the case when α is high. Since cooperative caching is most effective when the working set is large, its gain is greatest when α is lowest. The reason why we still see a relatively high gain when the cache size is large enough to capture most of the working set is due to the compulsory misses. We recall that cooperative caching reduces compulsory misses by fetching the missed objects from anoother proxy's cache. In all the subsequent results, we observe similar trends.

In sum, for the entire range of empirical α values (0.65 – 0.9), we observe that the cooperative caching gain lies between 40% and 70%.

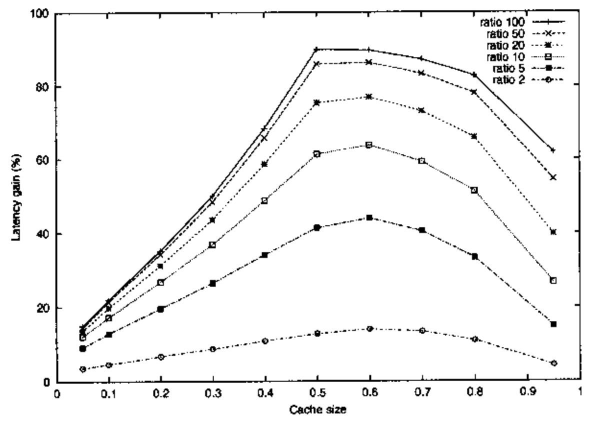
4.2.2 Cluster configuration

There are mainly two factors that determines the cluster configuration on a given network: (i) how tight the connections between the proxies in the cluster are? and (ii) how close the local proxy is, compared to the origin server? We quantify the first parameter by the τ_s/τ_c ratio and the second parameter by the τ_s/τ_l ratio. We discuss the impact of changing these ratios below.

We recall that the latencies in these ratios are not just network delays but total latencies incurred to download objects. As a result, even when network delays are comparable, τ_s may be an order of magnitude higher than τ_c and τ_l. More precisely, these ratios are determined not only by network delay but also by object size and network bandwidth as well, as shown in the following example. Consider two links: L_A with 200msec delay and 100Kbps bandwidth, and L_B with 50msec delay and 10Mbps bandwidth. When downloading very small objects, the download latency will be dominated by the network latency, thus the latency ratio of L_A to L_B will be close to 4. However, when downloading large objects, the download latency will be dominated by the network bandwidth, thus the latency ratio of L_A to L_B will be around 100! In this paper, we consider a range of τ_s/τ_c and τ_s/τ_l values from 2 to 100.



(a) Analysis

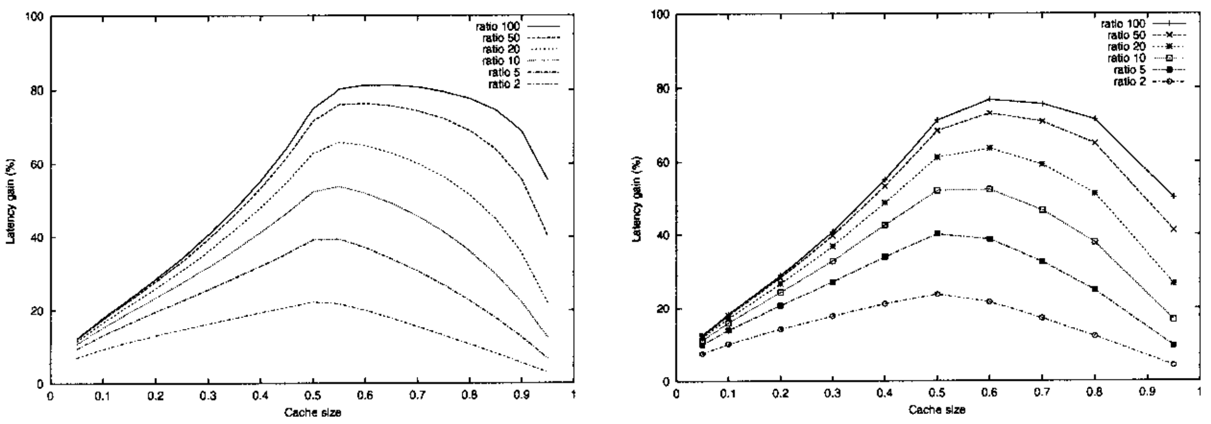


(b) Simulation

Figure 3: The latency gain $\Gamma_{\frac{CCR}{NC}}$ with respect to various τ_s/τ_c ratio values. The cooperative caching gain increases as the τ_s/τ_c ratio increases, i.e., as the inter-proxy latencies decrease.

Figure 3 plots the latency gain $\Gamma_{\frac{CCR}{NC}}$ with respect to various τ_s/τ_c values. Again, from the figures, we find close conformance between the potential gain predicted by the analytical model and the actual gain obtained from simulations. From the figure, we observe that the cooperative caching gain increases as the τ_s/τ_c increases. In particular, the gain can become as large as 90% when τ_s/τ_c is 100. However, when τ_s/τ_c is 2, the potential benefit of cooperation is marginal. Qualitatively, this implies that cooperative caching will become more effective when downloading objects from a neighboring proxy takes much less time than getting them from the origin server. In practice, the result means that cooperative caching gain is most prominent when objects are large and collaborating proxies are connected via high bandwidth connections compared to their connections to the origin server.

These results corroborate the findings reported in [8] that cooperative caching gain is sensitive to the ratio of server latency and cluster latency. In particular, we confirm that the gain is negligible when the ratio is relatively small as assumed in [8]. When τ_s/τ_c is large as assumed in [16], however, we predict that cooperative caching can be significantly beneficial.



(a) Analysis (b) Simulation

Figure 4: The latency gain $\Gamma_{\frac{CCR}{NC}}$ with respect to τ_s/τ_l ratio. The cooperative caching gain increases as the τ_s/τ_l ratio increases, i.e. as local proxies are closer to clients

Now we study the impact of τ_s/τ_l ratio. Figure 4 plots the results from analysis and simulation. In this figure also, we find that analysis and simulation closely agree with each other. From the figures, we observe a similar trend that was found in τ_s/τ_c case, i.e., cooperative caching gain increases as the τ_s/τ_l ratio increases. In particular, when $\tau_s/\tau_l = 100$, the benefit of cooperation can be as high as 80%; whereas when $\tau_s/\tau_l = 2$, the benefit of cooperation is moderate, under 20%.

This result shows that not only *caching gain* (performance with cache vs. without cache) but also *cooperative caching gain* (performance with cooperation vs. without cooperation) is sensitive to the latency to access the local proxy. In particular, we find the proximity of local proxies are important factors to achieve large cooperative caching gain.

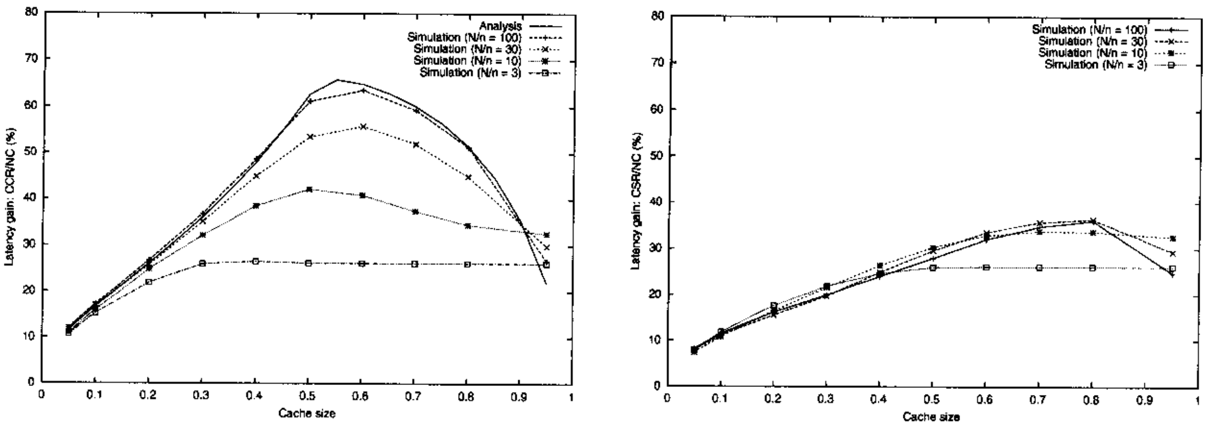
To sum up, we find both inter-proxy latency and client-to-proxy latency play important roles in determining the achievable cooperative caching benefit. Since their values may vary within a large range depending on network environments and the achievable benefit is quite sensitive to their values, we feel that they are determinant factors when answering whether cooperative caching must be deployed in a particular environment or not. In addition, overall, our analytical model seem to be effective in predicting the potential gain of cooperation across a wide range of latency ratios.

Now we discuss the cases when the actual gains do not meet the upperbound predicted by the analysis.

4.3 User access density

In the previous section, we examined the effect of the Zipf parameter α, latencies between clients, proxies, and the server on the upperbound of the benefits of cooperative caching. In this section, we show that the *actual gain* of cooperation, even when the parameters that determine the analytical upperbound are fixed, is quite sensitive to "user access density," which we define as the ratio of the number of total request N to the number of distinct objects n during a measurement period t. Intuitively, when the N/n ratio is high, we refer the user access is dense and user interests are focussed, since the user requests are destined to a relatively small number of objects. On the other hand, when the N/n ratio is low we call the user access is sparse and interests are diverse. The N/n ratio in the previous section was 100.

Now let us examine how the access density affects the achievable gain of CCR. We decrease the access density (or increase the diversity in user interest) by maintaining the same number of total number of requests but increasing the number of distinct objects in the requests. For simulation, we varied the number of distinct objects from 10,000 to 330,000, which corresponds to case where the N/n ratio changes from 100 to 3.



(a) $\Gamma_{\frac{CCR}{NC}}$ (b) $\Gamma_{\frac{CSR}{NC}}$

Figure 5: Latency gains $\Gamma_{\frac{CCR}{NC}}$ and $\Gamma_{\frac{CSR}{NC}}$ with respect to N/n values. The benefit from CCR decreases as N/n decreases, whereas the benefit from CSR is relatively insensitive to the changes of N/n.

Figure 5 presents latency gains $\Gamma_{\frac{CCR}{NC}}$ and $\Gamma_{\frac{CSR}{NC}}$ with respect to various N/n values. From the figure, we observe that actual gain of CCR measured from simulation deviates from the upperbound of the gain computed from analysis by a large degree, as the N/n ratio decreases. In particular, when N/n ratio are under 10, the actual gain was more than 20% less than the upperbound. On the other hand, the gain from CSR is more or less insensitive to the changes in N/n ratio.

We provide the following reasoning to explain why CCR was able to achieve its potential benefit when the user access was highly focussed and dense. We know that cooperative caching provides the most benefit when the working set does not fit in a single cache. Cooperative caching schemes promise improved performance by effectively increasing the total usable cache size, thereby capturing more of the working set than a single cache can. In particular, the performance of CCR depends on accurate estimation of the access frequency for these "extra" objects (the most popular objects that are not already cached by CSR and NC, but can be captured by the aggregate cache in the cluster), so that they can capture the *right* set of objects to minimize aggregate average latency, and CCR can yield benefits when these extra objects contribute to increase in hit ratio. However, estimating access frequency accurately for these less frequent objects requires a sufficient number of accesses to these objects. Furthermore, the benefit of accommodating these "extra" objects in the cluster will only pay off if these objects are accessed many times in the future. In this way, the initial miss to these objects will be amortized by many later hits.

Now we are left with two important questions: (i) What is the N/n ratio found in the real world trace? (ii) Is N/n ratio somehow related to the Zipf parameter α? In what follows, we address these questions.

Table 2 shows a variety of N, n, and α values reported in several different studies. The N/n values are simply calculated from N and n. From the table, we find interesting trends: the N/n ratios in Web proxy traces are relatively low, ranging from 2 to 7; whereas the N/n ratios in Web server traces are relatively high and many of them are greater than 100. These numbers match our intuition on access density and user interest diversity: We expect user interest to be very diverse in the requests seen by Web proxies. However, accesses will be focussed on a confined set in the requests seen by Web servers. Based on the same logic, we conjecture that certain proxies, e.g., reverse proxies in front of servers, or caches in content distribution networks (CDN), will observe high access density. Validating this conjecture is an ongoing research.

To answer the second question, we draw our attention to DEC trace and IBM trace in the table. Both these traces have very similar α value, but their N/n ratios differ drastically. We note that, while the N/n ratio depends on α which describes the normalized access frequencies among objects, it is not fully determined by

Table 2: User access characteristics from Web proxy traces and Web server logs.

site	*duration (year)*	N	n	N/n	α	*notes*
DEC	7 days (1996)	3.5×10^6	1.0×10^6	3.5	0.77	Web proxy [10]
UCB	18 days (1996)	9.2×10^6	3.5×10^6	2.6	0.69	Web proxy [10]
UW	7 days (1999)	82.8×10^6	18.4×10^6	4.5	N/A	University proxy [7]
Microsoft	6 days (1999)	107.7×10^6	15.3×10^6	7.0	N/A	Corporate proxy [7]
CANARIE	45 days (1999)	7.3×10^6	4.6×10^6	1.6	0.77	level 2 proxy [17]
NLANR	30 days (1999)	24×10^6	8.4×10^6	2.9	0.74	level 3 proxy [17]
USask	N/A	5.0×10^6	1.7×10^6	2.9	0.80	University proxy [11]
N/A	75 days	25,000	$\sim$ 10,000	$\sim$ 2.5	0.73	RealMedia proxy[13]
NASA	2 months (1995)	3.0×10^6	9,300	322	N/A	Web server [15]
Clarknet	2 weeks (1995)	2.9×10^6	32,000	91	N/A	Web server [15]
NCSA	1 week (1995)	2.3×10^6	23,000	100	N/A	Web server [15]
IBM	3.5 days (1998)	15.5×10^6	39,000	497	0.76	Corporate Web server [12]
IBM	6 days (2001)	43×10^6	53,000	811	N/A	Corporate Web server [12]

α since in addition, N/n ratio captures the absolute order of magnitude of requests to a given object.

The effect of N/n ratio on the achievable gain of CCR has interesting implications: when N/n is small—as in the Web proxy environments, where user interests are diverse and access density is low—the benefit from CCR is insignificant; instead most of the benefit of cooperative caching can be achieved by CSR. Thus a simple cooperation of CSR is sufficient in such environment. On the other hand, when N/n is large—as in CDN environments, where we conjecture user requests to be focussed and dense—the potential benefit of CCR over CSR can be significant. Thus, a more sophisticated level of cooperation of CCR is worth considering.

4.4 The case of M proxy cluster

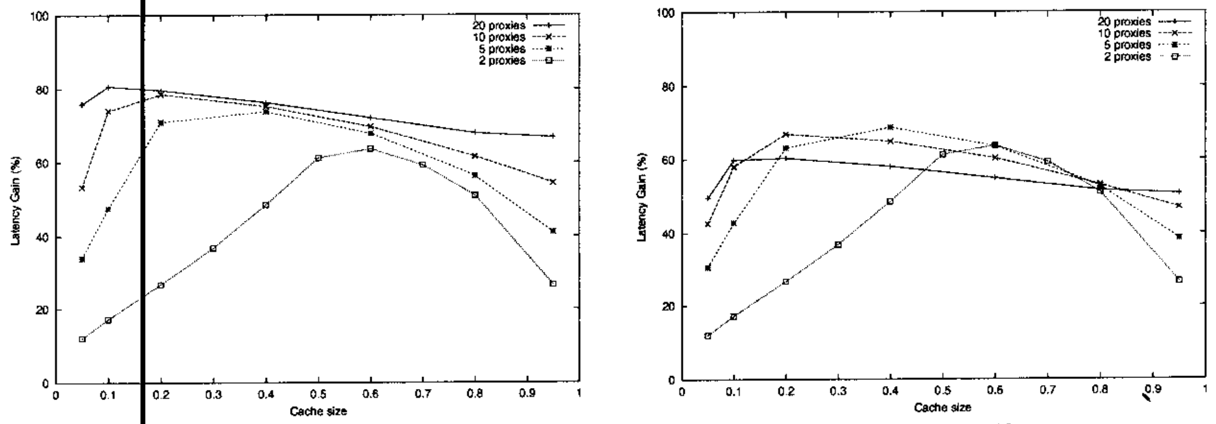
So far we have studied the case of a simple two-proxy cluster varying user access patterns and cluster configurations. In this section, we present the results with varying number of proxies in a cooperative cluster.

Before we present the performance results, we first need to discuss parameters that determine cluster configuration. We consider (i) the allocation of inter-proxy latency, (ii) relationship between the average inter-proxy latency and the cluster size, and (iii) the number of proxies cooperating in a cluster. The first parameter inter-proxy latency allocation is of our interest, since we represent network link characteristics and object sizes using latency between proxies. We have considered the following latency allocation schemes in this paper: (a) *equi-distant*: the latencies between any two proxies are the same; (b) *random*: the latencies between proxies are determined randomly; (c) *Cartesian-distant*: we assume proxies are placed on a plane; first, the locations of proxies on a plane are randomly determined, then the latencies between two proxies are set to the Cartesian distance between the;, and (d) *spanning tree*: first the locations of proxies on a plane are determined randomly, then a spanning tree is overlayed on top of them; the latencies between proxies are set to the traversal distance along the tree.

The above schemes resulted in different latency allocations among proxies. However, there was little difference in the relative performance of cooperative caching across different allocation schemes, as long as the average cluster latency $\tau_c (= \frac{1}{M}\sum l_{i,j})$ remains the same. Therefore, we present the results with random allocation in the remainder of the section.

The second parameter we consider in cluster configuration is the relationship between the average inter-proxy latency τ_c and the size of cluster. We considered two scenarios: (a) *constant*: the inter-proxy latency doesn't increase with the cluster size, and (b) *square-root*: the inter-proxy latency increases proportional to the square root of the cluster size, i.e. $\tau_c \propto \sqrt{M}$. The former represents the case of tight cluster, where caches are collocated or at least interconnected via local area connections; whereas the latter represents the case of loosely connected cluster, where caches are interconnected via wide area connections.

The third parameter to consider in our evaluation was the number of proxies cooperating in a cluster. We do not consider a very large scale cooperation in this paper because of the heterogeneity of user interests over wide area [7]. Besides, computation complexity and communication overhead makes a sophisticated cooperation in a very large scale cluster impractical. Therefore, in this paper, we consider cooperation among 2, 5, 10, 20 proxies in a cluster.



(a) τ_c = constant (b) $\tau_c \propto \sqrt{M}$

Figure 6: Latency gains $\Gamma_{\frac{CCR}{NC}}$ with respect to cluster size.

Figure 6 presents the latency gains of cooperative caching with respect to different cluster size in two different average latency increase scenarios. The x-axis represents the normailzed cache size of each proxy and the y-axis represents the latency gain $\Gamma_{\frac{CCR}{NC}}$. The τ_s/τ_c ratio was 10 in two proxy case, which remained the same for all M in constant τ_c case. In the square-root case, τ_c got increased according to $\sqrt{M}$, e.g., when $M = 5$, $\tau_s/\tau_c = 10 \times \sqrt{\frac{2}{5}}$. We used a homogeneous dense access pattern with $\alpha = 0.7$ and $N/n = 100$.

From the figures, we make two observations. First, the achievable gain from cooperative caching increases significantly, especially in the small cache size region, with the number of proxies when τ_c remains constant; however, when τ_c increases with $\sqrt{M}$, the maximum achievable gain doesn't increase much with the number of proxies. Second, the cache size that achieves the maximum cooperative caching gain decreases, as the number of proxies increases, in both cases. In other words, the optimal cache size to achieve the maximum cooperative caching gain reduces as the cluster size increases. Therefore, we can conclude that multiple proxy cluster will be most effective when the interconnections between cooperating proxies are tight, the working set size is relatively large, and user access pattern across all proxies are homogeneous.

To summarize, in this section, we validated our analytical results using simulations varying parameters that determine user access pattern and cluster configurations. We have also learned that the achievable benefit using cooperative caching is quite sensitive to the user access density. Finally, we have shown the effectiveness of multiple proxy cooperation under various configurations.

5 Related work

With the exponential growth in the use of the world wide web, caching in proxies both to improve client access latencies and to reduce network congestion has become a very actively researched area. More recently, researchers have investigated the potential for cooperation among web proxy caches to further accelerate web access and conserve Internet bandwidth. Several studies relevant to this topic have been reported in the literature. They have focused on characterizing web request traffic [10, 18], on devising efficient techniques for cooperation [4, 1], and on quantifying the benefit from cooperative caching under certain assumptions about the workload and the topology [9, 7, 8]. Recent work has also focussed on the placement and replacement algorithms that should be employed in a cooperative architecture [6].

Breslau *et al.* [10] reported that the popularity of web objects follows a Zipf-like distribution. They reported that the cache hit rate of a web proxy grows in a log-like fashion as a function of the cache size, increasing rapidly initially then flattening out.

Cooperation among caches to serve user requests was introduced with hierarchical caching in Squid [19] and Harvest [20]. When a user request misses in a cache, it queries all its siblings using the ICP protocol. If the object is not found, it forwards the request to its parent. The hierarchies are manually configured and ICP is not a ver scalable protocol.

Fan *et al.* [4] proposed using a Bloom filter to reduce the amount of directory information that must be exchanged between cooperating proxies and implemented it in Squid. Rabinovich *et al.* [1] outlined a distance-sensitive model for cache cooperation to limit cooperation overhead only to the case where cooperation is beneficial. Directory information describing the contents of peer caches is kept only for those caches that are close enough.

Krishnan *et al* [9] studied the utility of cooperation among caching proxies in a corporate intranet environment. The study found out that cooperation improved hit rates, but that the improvement in the daily hit rates varied widely, from .7 to 28.9%. They suggested specific metrics to monitor cooperation, like useful extra network traffic and increased server load and concluded that smart algorithms which adaptively turn on and turn off coordination based on the actual benefit derived cooperation are crucial.

Wolman *et al.* [7] performed a trace-driven simulation and an analytical study of the effectiveness of cooperative web proxy caching. Their study concluded that cooperative caching has performance benefits only when the cooperating proxy caches serve client populations of limited size. For large client populations, however, cooperative caching was found to provide little benefit. The study further reported that the proxy caches sampled in their study were often large enough to capture the working set, and that capacity misses rarely were the bottleneck with only 3% of the requests to Microsoft Corporation's proxies missing due to the finite capacity.

The conclusions of Wolman *et al.* follow from the observation that web cache hit rates grow according to a log-like fashion, also reported in [10]. The effectiveness of cooperation thus depends on the relative size of the working set and the proxy cache size. During the time when the study was conducted, video and rich media objects were very rare, but trends suggest that is likely to change in the near future. Our study is complementary to theirs in that it investigates the potential of the different levels of cooperation even in the case when the working set is not small compared to the cache size, and it draws conclusions about the desirable levels of cooperation that are relatively independent of such parameters.

Dykes and Robbins [8] studied the performance of two different types of cooperative caching schemes, mesh and hierarchy, using a simple analytical model. The performance improvement in terms of average access latency were found to have a relatively low upper bound, even when optimistic assumptions are made about the hit rate increase expected from cooperation. For instance, the model predicts that a cooperation hit ratio of 0.40 yields a speedup of only 1.7. Our study shows that in the case when the cache size is small with respect to the working set size, high cluster hit rates can occur, in which case the benefit from cooperative service would be significant. We also find conditions under which cooperative replacement can yield performance gains over just cooperative lookup.

The closest cost-based algorithm to the optimal algorithm presented in this paper was proposed by Korupolu *et al.* [6]. Korupolu *et al.* proposed a cost-benefit model based on inter-node latencies to determine cache placement problem assuming hiearchy of cooperative caches. They proved that the optimal placement has the same complexity as the minimum-cost flow problem, and proposed an implementation that has close to optimal performance. The study concluded that coordinated placement and replacement can significantly improve performance.

In this study we classify caching schemes based on their degree of cooperation in two key aspects – request redirection and object replacement. We evaluate the performance of the different classes of cooperation under various parameter ranges. Our findings unify the results of earlier studies and highlight the circumstances under which each cooperation scheme yields the most benefit.

6 Summary and discussion

In this paper, we examined the potential benefits achievable through cooperation among proxies under various network configurations and user access behaviors, by classifying cooperation schemes and analyzing them under a unified mathematical framework. In addition, we validated the analytical results and quantified the achievable benefits of cooperation using a trace-driven simulation. From the understanding that we gained from the mathematical analysis and simulation results, we reached at the following conclusions:

- Cooperation among proxies is beneficial when the average object size is large and the working set does not fit in a single proxy. The benefit is also sensitive to the cluster configuration: it is most visible when the proxies are well-connected and when the local proxy is close to the clients.
- Simple form of cooperation (CSR)—in which proxies cooperate in serving missed requests from other proxies in the cluster—is mostly sufficient when the user interests are highly diverse, as in the Web proxy environments.
- However, additional level of cooperation (CCR)—in which the proxies cooperate in object replacement decisions—will yield further benefits when the user accesses are dense and the requests are highly focussed to a relatively small number of objects, as in CDN environments.

As the conclusions indicate, the benefit of cooperation among proxies is very much dependent on user access, user interests, and network configurations. Our work is the first to unify the different conclusions observed in the literature through a systematic examination on the benefit of cooperation among proxies. As part of future work, we intend to extend our simulation results using web access traces collected among a set of cooperating proxies. In addition, we intend to further enhance our mathematical model to incorporate various aspects of user access dynamics, such as changing user interests and lifetime of objects.

References

[1] M. Rabinovich, J. Chase, and S. Gadde, "Not all hits are created equal: cooperative proxy caching over a wide-area network," *Computer Networks and ISDN Systems*, vol. 30, no. 22–23, pp. 2253–2259, 1998.

[2] Madhukar R. Korupolu, C. Greg Plaxton, and Michael Dahlin, "Placement Algorithms for Hierarchical Cooperative Caching," in *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 586–595, January 1999.

[3] Jussi Kangasharju, James Roberts, and Keith W. Ross, "Object Replication Strategies in Content Distribution Networks," in *Proceedings of the Sixth Workshop on Web Caching and Content Distribution (WCW)*, June 2001.

[4] Li Fan, Pei Cao, Jussara Almeida, and Andrei Broder, "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol," in *Proceedings of the ACM SIGCOMM'98 conference*, pp. 254–265, September 1998.

[5] Renu Tewari, Michael Dahlin, Harrick Vin, and John Kay, "Beyond Hierarchies: Design Considerations for Distributed Caching on the Internet," Technical Report TR98-0, University of Texas, February 1998.

[6] Madhukar R. Korupolu and Michael Dahlin, "Coordinated Placement and Replacement for Large-Scale Distributed Caches," in *Proceedings of the IEEE Workshop on Internet Applications*, pp. 62–71, July 1999.

[7] Alec Wolman, Geoff Voelker, Nitin Sharma, Neal Cardwell, Anna Karlin, and Henry Levy, "On the scale and performance of cooperative Web proxy caching," in *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP'99)*, pp. 16–31, December 1999.

[8] Sandra G. Dykes and Kay A. Robbins, "A Viability Analysis of Cooperative Proxy Caching," in *Proceedings of IEEE INFOCOM 2001*, April 2001.

[9] P. Krishnan and Binay Sugla, "Utility of co-operating Web proxy caches," *Computer Networks and ISDN Systems*, vol. 30, no. 1–7, pp. 195–203, 1998.

[10] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker, "Web Caching and Zipf-like Distributions: Evidence and Implications," in *Proceedings of INFOCOM '99*, pp. 126–134, March 1999.

[11] Carey Williamson, and Madashiru Busari, "On the Sensitivity of Web Proxy Cache Performance to Workload Characteristics," in *Proceedings of INFOCOM 2001*, April 2001.

[12] Ronald P. Doyle, Jeffrey S. Chase, Syam Gadde, and Amin M. Vahdat, "The Trickle-Down Effect: Web Caching and Server Request Distribution," in *Proceedings of Web Caching Workshop 2001*, June 2001.

[13] Dario Luparello, Sarit Mukherjee, and Sanjoy Paul, "Streaming media traffic: an empirical study," in *Proceedings of the Sixth International Workshop on Web Caching and Content Distribution*, June 2001.

[14] Kang-Won Lee, Sambit Sahu, Khalil Amiri, and Chitra Venkatramani, "Understanding the Potential Benefit of Cooperative Caching: Taxonomy and Analysis," IBM Research Technical Report, IBM Thomas J. Watson Research Center, available at http://www.research.ibm.com/people/k/kangwon/techreports/coop-caching.ps, July 2001.

[15] Martin F. Arlitt and Carey L. Williamson, "Web Server Workload Characterization: The Search for Invariants," in *Proceedings of ACM SIGMETRICS '96*, May 1996.

[16] Peter Christy, "The Network Providers Business Case Internet Content Delivery," in *On-line document by Internet Research Group, available at http://www.akamai.com*, 1999.

[17] Anirban Mahanti, Carey Williamson, and Derek Eager, "Traffic Analysis of a Web Proxy Caching Hierarchy," in *IEEE Network Magazine*, May/June 2000.

[18] P. Cao and S. Irani, "Cost-aware WWW proxy caching algorithms," in *USENIX Symposium on Internet Technologies and Systems*, 1997.

[19] –, "Squid Web Proxy Cache," in *On-line document, available at http://www.squid-cache.org*, 2001.

[20] Anawat Chankhunthod, "The Harvest Cache and Httpd-Accelerator," in *On-line document, available at http://excalibur.usc.edu*, 1995.