Collaborative Web Caching Based on Proxy Affinities

Jiong Yang
T. J. Watson Research Center
IBM
jiyang@us.ibm.com

Wei Wang
T. J. Watson Research Center
IBM
ww1@us.ibm.com

Richard Muntz
Computer Science
Department
UCLA
muntz@cs.ucla.edu

ABSTRACT

With the exponential growth of hosts and traffic workloads on the Internet, collaborative web caching has been recognized as an efficient solution to alleviate web page server bottlenecks and reduce traffic. However, cache discovery, i.e., locating where a page is cached, is a challenging problem, especially in the fast growing World Wide Web environment, where the number of participating proxies can be very large. In this paper, we propose a new scheme which employs proxy affinities to maintain a dynamic distributed collaborative caching infrastructure. Web pages are partitioned into clusters according to proxy reference patterns. All proxies which frequently access some page(s) in the same web page cluster form an "information group". When web pages belonging to a web page cluster are deleted from or added into a proxy's cache, only proxies in the associated information group are notified. This scheme can be shown to greatly reduce the number of messages and other overhead on individual proxies while maintaining a high cache hit rate. Finally, we employ trace driven simulation to evaluate our web caching scheme using three web access trace logs to verify that our caching structure can provide significant benefits on real workloads.

1. INTRODUCTION

As traffic on the Internet continues to grow, how to improve Internet performance has become a challenging issue. Recent research to tackle this problem falls into three categories.

- *server load balancing.* It is well known that the accesses over Internet are not uniformly distributed. A common approach adopted by popular web sites to alleviate server bottleneck is to provide a virtual URL interface and use a distributed server architecture underneath. Some internal mechanism to dynamically assign client requests to the web servers is required to achieve scalability and transparency to the clients. This assignment decision can be taken at the IP level or at the domain name system level. In addition to their individual drawbacks [5], both schemes require the set up of additional server(s) at local or remote sites to handle request overloads. This approach can be taken to avoid request overload at web sites which are continuously popular (e.g., DBLP bibliography web site) or predictably popular during some period (e.g., Olympic game web site). However, due to its static nature, this mechanism can not handle unforeseeable bursty requests effectively.
- *intra-net collaborative caching.* Within a medium area network (MAN) (e.g., campus wide network), several proxies are usually connected via high speed links. Web pages fetched from the content server by one proxy can be used to potentially fulfill later requests (of the same web page) from other proxies within the same MAN. If multiple requests of the same web page are submitted from the same MAN within a short period of time, only the first one will go through the content server. Since the proxy is usually much "closer" to the client than the content server, the average latency of web page retrieval is reduced. In addition, only a fraction of all requests need to be handled by the server, thus the server workload and Internet traffic are also reduced. Summary cache [11] is an example of this type of technique. However, this approach still can not fully eliminate the server bottleneck caused by some unforeseeable Internet-wide bursty accesses since a cached copy of a web page is only shared by clients within the same MAN.
- *inter-net collaborative caching.* Based on the following observations, inter-net collaborative caching has been recognized as another feasible approach to improve Internet performance.
    1. Retrieving data from a nearby proxy usually is much faster than fetching the data from a more distant server (less latency and, most likely, higher bandwidth).
    2. A proxy that fetched a web page could potentially serve as the provider of this web page at a later time (as long as it still caches an up-to-date copy of the web page).

However, the number of proxies connected via the Internet is large. Without incurring large overhead, how to identify a potential cache provider and moreover, if multiple providers exist, how to choose an optimal one, are very crucial for the scalable performance of the caching scheme. Harvest [9] [6] uses a proxy hierarchy to conduct the search for a potential provider. This hierarchy is formed statically and independent of the proxy affinity for web pages. Thus, it is likely that the cache providers are far apart (in the hierarchy) from the requester. In such a case, large overhead will be incurred for cache discovery. [19] also tries to solve this problem by maintaining a dynamic routing table. In our proposed approach, instead of employing a fixed hierarchy, proxy affinities are used to group proxies dynamically. Intuitively, from the viewpoint of a proxy, each web page request falls into one of the following three categories.

1. The proxy locally caches an up-to-date version of the requested web page.
2. An up-to-date version of the requested web page is cached by some other nearby proxy.
3. The requested web page has to be obtained from the content server.

Web pages in the first and third categories are easy to handle since the operations are local to each individual proxy. However, collaborations among proxies is required to locate cached web pages in the second category. A carefully designed caching scheme is crucial in order to avoid large overhead. In our approach, proxies that have similar affinities will cooperate and share their cache content with each other in such a manner that little overhead is required to achieve a relatively high cache hit ratio. Whenever a proxy fetches a web page, it will serve as a cache provider until this copy is obsolete.

Our ultimate goal is to minimize the average response time of web page retrievals. Two aspects are involved. (1) What should be cached and where? (2) How does a proxy locate a remote cache containing a desired page? The first question can be addressed by algorithms for prefetching [14] [12]. Due to space limitations, we will not address this issue but rather focus on the second one.

The second issue is how to discover which proxy has a cached copy of a particular web page. Basically, there are two approaches to cache discovery: pull and push. The pull technique is that, when a proxy tries to locate a cache, it first finds which proxy caches the web page, and then retrieves the page. This usually increases the average response time. With the push technique, when the cache contents of a proxy changes, the proxy will tell other proxies about the change. However, the push technique can communicate a large amount of unnecessary information. This generates a significant overhead not only on the network, but also on the proxies due to message processing. In addition, to locate a cache for a particular web page, a proxy has to maintain the information about other proxy's cache contents. This can consume a significant amount of resources when the number of proxies is large. Therefore, a proxy should be selective concerning which proxies it should send the updates of its cache contents and how often. The trend of increasing diversity of web page popularity [2] also indicates that a static global caching structure regardless of individual proxy affinities would become less effective. In fact, an optimal cache scheme should have good adaptability to diverse proxy affinities and their changes. In our scheme, a dynamic distributed collaborative caching infrastructure is built according to the proxy affinities so that unnecessary communication between proxies of distinct affinities can be avoid and changes of proxy affinities can be adapted to easily and quickly.

Moreover, scalability is an important aspect of a web caching scheme since the number of proxies and servers could range up to hundreds of thousands in the Internet environment. It will be shown in a later section that our approach can scale well when the number of proxies is large.

In this paper, we propose a scheme using *proxy profiles* and *information groups* which is based on web page access patterns. The goal of this scheme is to reduce the average number of messages among proxies for updating the cache status while maintaining a high cache hit rate. In our scheme, a proxy profile contains a list of URLs of the web pages that are frequently accessed by the proxy, while an information group is essentially a multicast group that links proxies which are collaboratively caching web pages that are of common interest. We group web pages into a set of overlapping *web page clusters*. A proxy may choose to join an information group if the intersection of the associated page cluster and the proxy's profile is significant. Generally, a proxy joins enough information groups to cover its most frequently access pages. Since a proxy's profile may evolve over time, the web page cluster generation and information group formation are performed periodically (and incrementally) to adapt to changes in the proxy's access pattern.

When a host wants to access a web page, it sends a request to its local proxy. If the requested web page is not cached in the local proxy, then the proxy checks whether other proxies in the information group(s) (for this page) currently have this page in their cache. The request will be sent to the "nearest" site which has this page. Once there are enough changes in the cache contents of a proxy, the proxy will notify other proxies in the same information group(s) of the changes. Therefore, in order to achieve optimal performance, each information group should be of moderate size because a large group size incurs significant overhead for bookkeeping while a small group suffers from low cache hit ratio.

The remainder of this paper is organized as follows. We discuss some related work in Section 2. Section 3 presents the objective model for our caching scheme. The simulation set up is presented in Section 4. Our page clustering algorithm and information group formation algorithm are presented in Section 5 and 6, respectively. In Section 7 and 8, we discuss the web page retrieval process and optimal information group size estimation, respectively. Finally, conclusions are drawn in Section 9.

2. RELATED WORK

In this section, we discuss some other web caching schemes proposed recently. We note that most previous cooperative caching schemes [6] [19] [11] focused on how caches cooperate with each other. They did not take into account the correlation among proxy affinities. How to best utilize reference pattern information to improve the Web performance has become a topic of increasing interest [8]. One contribution of our work is a scheme to efficiently analyze reference patterns to organize proxies into collaborative caching groups. High cache hit rate can still be maintained at the cost of much fewer inter-proxy messages as compared with previous schemes.

2.1 Caching in Harvest

The caching scheme in Harvest is presented in [9] [6]. Caches are organized in a hierarchy. Each cache in the hierarchy independently decides whether to fetch the reference from the object's home site or from its parent or sibling caches using a resolution protocol. If the URL contains any of a configurable list of substrings, then the object is fetched directly from the object's home, rather than through the cache hierarchy. This feature is used to force the cache to resolve non-cacheable URLs (e.g., cgi-bin) formed pages and local URLs directly from the object's home. Otherwise, when a cache receives a request for a URL that misses, it performs an ICP(Internet Caching Protocol) to all of its siblings and parents, checking if the URL hits any sibling or parent. Of all the sites which report having the object, the object is retrieved from the site with the lowest measured latency.

2.2 Adaptive Web Caching

An adaptive web caching scheme is proposed in [19]. Cache servers are self-organizing and form a tight mesh of overlapping multicast groups and adapt as necessary to changing conditions. This mesh of overlapping groups forms a scalable, implicit hierarchy that is used to efficiently diffuse popular web content towards the demand. Adaptive web caches exchange a description of their content state with other members of their cache groups to eliminate the delay and unnecessary use of resources of explicit cache probing. For each request, a cache first determines whether a copy of the requested web page is cached by one of its group members. If not, the request will be forwarded to another cache (in the web caching infrastructure) which is significantly more likely to have the data. This routing is accomplished using a URL routing table maintained at each web cache. The information in the URL routing table is learned from the source-based and content-based entries from other web caches.

2.3 Summary Cache

Summary cache is a scalable wide area web cache sharing protocol proposed in [11]. In this protocol, each proxy keeps a summary of the cache directory of each participating proxy. When a user request misses in the local cache, the proxy checks these summaries for potential hits. If a hit occurs, the proxy sends out requests to the relevant proxies to fetch the document. Otherwise, the proxy sends the request directly to the web server. In order to reduce the network overhead, a proxy does not update the copy of its summary stored with other proxies upon every modification of its directory: it rather waits until a certain percentage of its cached documents are not reflected in other proxies. In addition, bloom filters [3] are used to build summaries in order to keep each individual summary small. Experiments have shown that summary cache reduces the number of inter-cache protocol messages, the bandwidth consumption, the protocol CPU overhead significantly while maintaining almost the same cache hit ratio as the ICP. However, when the number of proxies is large, the network traffic for updating cache contents could also be large. Thus summary cache may have poor scalability.

2.4 Web Caching Based on Dynamic Access Patterns

A "local" caching algorithm based on dynamic access patterns was presented in [18]. This caching algorithm flexibly adapts its parameters (such as the number of documents, the size of cache, and the actual documents in the cache, etc.) based on the observation that there is a strong relationship between frequency and probability of access. The analysis is based upon a model from psychological research on human memory. The probability of future document access is estimated based on frequency of prior document accesses. However, this algorithm did not address the issue of cooperative caching.

2.5 Server Volumes and Proxy Filters

[8] proposed an end-to-end approach to improve Web performance using server volumes and proxy profiles. The server groups related resources into *volumes* based on access patterns (captured by conditional probability) and the file system's directory structure. This information is *piggybacked* in the servers response message to the requesting proxy. The proxy generated *filter*, which indicates the type of information of interest to the proxy, is used to tailor the piggyback information. This piggyback information can be used to improve the effectiveness of a variety of proxy policies such as cache coherency, prefetching, cache replacement, and so on. Experimental results show that probability-based volumes can achieve higher prediction rate with a lower piggyback size than the directory-based structure. Further reductions in processing and memory overhead are possible by limiting the calculation of probability implications to pairs of resources that have the same directory prefix, at the expense of missing associations between resources in different directories.

3. OBJECTIVE MODEL

The objective of our web caching scheme is to minimize the average latency (or response time) of web page retrieval. The following is a list of parameters for estimating the average response time.

- γ: local cache hit ratio. The ratio of the number of requests which are served by local proxy over the total number of requests.
- ω: remote cache hit ratio. The ratio of the number of requests which are served by remote proxies over the number of *outgoing requests*, i.e., the requests which can not be satisfied locally.
- $Local\_Cost$: the average response time of serving a request by the local proxy.
- $Remote\_Cost$: the average response time to serve a request through a remote proxy.

- *Server_Cost*: the average response time to serve a request from the content server.

- *Locating_Cost*: the average time to find whether (and where) a cached web page exists.

- *Push_Cost*: the average overhead (per outgoing request) incurred by multicasting the changes of cache content to remote proxies. Usually, the volume of valid web pages cached at each proxy remains the same roughly if the system is stable. In other words, the web page expiration rate is commensurate with the fetch-in rate. Therefore, the change of cache content can be measured by the amount of new web pages fetched in. Assume that whenever the cache content changes by $\alpha\%$, the proxy will multicast the description of the changes to other proxies, and assume that this $\alpha\%$ change is the result of q ($q > 1$) outgoing requests on average, which take a period of time of length t_q on average. Then, each proxy multicasts, on average, once for every t_q length of time. Assume that a proxy receives m multicast messages (from other proxies) on average between two consecutive multicasts. Then, a proxy needs to spend $t_s + m \times t_r$ time to process multicast messages, where t_s and t_r are the average time to generate and send out a multicast message to other proxies and the average time to receive a multicast message sent by other proxy and update its local information, respectively. Thus, the total *push cost* for all proxies over a period of time t_q is $N_p \times (t_s + m \times t_r) + N_p \times t_n$ where N_p, t_n, and $N_p \times t_n$ are the total number of proxies, the average additional delay to other network traffic due to the transmission of such a multicast message[1], and the overall overhead incurred due to the delivery all multicast messages sent out by all proxies during t_q, respectively. Since the information multicasted is used to facilitate the remote cache discovery process, we should factor in this cost when we estimate the *search cost* of each outgoing request. Since there is on average of $N_p \times q$ outgoing requests overall (during the period of t_q), the average overhead for each outgoing request is $Push\_Cost = \frac{t_s + m \times t_r + t_n}{q}$.

- *Search_Cost*: $Push\_Cost + Locating\_Cost$.

- *Cost*: average response time of retrieval of a web page.

$$\begin{aligned} Cost = \; & \gamma \times Local\_Cost + (1-\gamma) \times Search\_Cost \\ & +(1-\gamma) \times \omega \times Remote\_Cost \\ & +(1-\gamma) \times (1-\omega) \times Server\_Cost \end{aligned}$$

To simplify the problem, we assume that a web caching algorithm can only control the following parameters: *Cost*, *Locating_Cost*, *Push_Cost*, *Search_Cost*, ω, but does not have control over the rest of parameters; so they can be viewed as constant. To minimize *Cost*, we therefore need to minimize the following function:

$$\begin{aligned} & Search\_Cost + \omega \times Remote\_Cost \\ & +(1-\omega) \times Server\_Cost \\ = \; & \tfrac{t_s + m \times t_r + t_n}{q} + Locating\_Cost + \omega \times Remote\_Cost \\ & +(1-\omega) \times Server\_Cost \end{aligned} \qquad (1)$$

where $Server\_Cost > Remote\_Cost$.

Each of ω and *Search_Cost* can be viewed as a function of the number of proxies in collaboration (denoted by m). The more proxies that are involved, the higher the remote cache hit ratio and the search cost. Thus, a moderate value of m should be chosen to balance the tradeoff between the remote cache hit ratio and the search cost. Moreover, an optimal strategy will be a scheme which has a higher remote cache hit ratio for a given search cost, or a lower search cost for a given remote cache hit ratio. It follows that during the push process, the update information should only be sent to those proxies which have an affinity to the involved web pages. By similar reasoning, during the remote cache discovery process, only those proxies that have an affinity for the requested page need to be queried. In our proposed scheme, proxies that share some common affinities will form an *information group* which serves as the basis of collaboration. All other parameters except m in Function 1 can be either collected or estimated at run time. Then, the optimal size of an information group m (i.e., the number of proxies in collaboration) can be calculated accordingly. In addition, instead of enforcing a unique information group size, we employ two parameters I_{min} and I_{max} to indicate the range of optimal value of m and require that the size of each information group is within the range $[I_{min}, I_{max}]$. This not only accommodates the evaluated error during the estimation of m, but also provides some flexibility to the construction of information group (without sacrificing the quality of collaboration). Values of I_{min} and I_{max} will be discussed in Section 8.

4. TRACES AND SIMULATIONS

In this paper, we use three sets of traces of HTTP requests to verify our design:

- DEC traces [10]: Digital Equipment Corporation Web Proxy Server traces from Aug. 29 to Sep. 4, 1996.
- ClarkNet traces [7]: ClarkNet is a full Internet access provider for the Metro Baltimore-Washington DC area. The traces contain all requests to ClarkNet server from August 24 to September 10, 1995.
- L traces [13]: an 100 days (during Aug.31 to Oct.7, 1997) worth of logs of HTTP requests from about 3000 distinct clients. It contains a total of 10 million requests. There are a total 12,000 servers and 300,000 distinct web pages represented in the trace.

Since the information in the trace logs is insufficient for the purpose of simulation, e.g., the trace logs do not reveal the connectivities of these hosts, we had to approximate the needed information for the detailed simulation. To build the topology of these hosts, we first need to find the number

[1] Transmitting this multicast message would consume certain network bandwidth and hence might cause some delay in the delivery of other packets.

of servers. We assume that web pages served by the same server all have the same first level URL component. For example, the URLs of all web pages served by the ACM server starts with "www.acm.org". By examining the IP address of each user/host, we can discover the number of subnets. Furthermore, we assume that hosts in a subnet are connected by a 100 MBit/s fast Ethernet and there is a proxy for each subnet. Then we generate a random connected graph in which each node corresponds to a server or a subnet. The edges of the graph correspond to the communication links between the server and client subnets. The bandwidth on these links is 10MBit/s. We generate a background workload which consumes half of the bandwidth on each communication link on average. In this paper we assume that the average lifetime of a web page is 7 days, and after a proxy fetches a web page, this copy of the web page will reside in the proxy's cache until it becomes obsolete[2]. Last but not least, the page size follows a log-normal distribution with an average 8KB[2]. Table 1 shows some statistical information on the above three traces.

5. PAGE CLUSTER

Web pages are identified by their URLs. Intuitively, different sets of web pages may have different popularity, i.e., some web pages are hot spots whereas some others are rarely accessed. Experimental results have shown that majority of the references go to a small percentage of the web pages[4]. It is much more beneficial to improve the cache hit rate for these *hot* web pages. Therefore, we focus on the set of frequently accessed web pages for the following reasons: (1) If a page is not frequently accessed by any proxy, then it is unlikely that an up-to-date copy of this page is cached by some proxy. When this page is needed by a proxy, the proxy has to fetch it from the server in most cases. Therefore, there is no incentive to analyze the access patterns of this set of web pages because we can rarely benefit from it. (2) The analysis of access patterns has a certain overhead which is reduced quickly by eliminating from consideration all infrequently referenced pages. Each proxy keeps a log of the web page reference history of local clients, which is a sequence of web page references. Given a web page and a trace log, the ratio of the number of accesses to this web page over the total number of accesses within the trace log is referred to as the *frequency* of the web page within the trace log. For example, if there are 100 web page accesses overall and a web page is referenced 30 times, then the frequency of that web page is 30% in this trace.

We use a threshold β (specified as a percentage) to determine whether a page is considered to be frequently accessed or not. A page is considered frequently accessed if and only if the reference frequency of that web page is at least β. Intuitively, the frequency can be viewed as an indicator of popularity of that page. Table 2 shows the effects of varying the value of β. For example, when β is chosen as 0.01% (i.e., a page is considered frequently accessed only if it is accessed at least once per 10,000 references), then 4% of all pages are considered to be frequently accessed, and 40% of all requests are to these 4% of the pages referenced in the DEC trace. (These values are in the page coverage and request coverage columns, respectively.) We will discuss the effect of the value of β on the cache hit ratio, average number of messages, average size of a message, etc. in Section 7.

Page clusters (of frequently referenced web pages) can be formed according to the combined access patterns of all proxies. Grouping web pages into clusters based on these logs consists of two steps:

1. Each proxy sends its profile (i.e., local frequently accessed web pages) to a central site S.
2. An optimal or near optimal partition of frequently accessed web pages is generated.

Each proxy maintains a *profile* which consists of URLs of its locally frequently accessed web pages. This can be easily done by examining the local trace log. We will explain later that this profile is used not only by the central site to group the page clusters but also by the proxy to guide itself to join appropriate information groups.

At the site S all reported web pages are grouped into a set of page clusters (See Figure 1). If a proxy frequently accesses at least one page in a page cluster, then we say that it *has an affinity for* that page cluster. An information group will be formed for each cluster for collaborative caching. A proxy may choose to join an information group if it has an affinity for the corresponding cluster. However, due to the overlaps among page clusters, a proxy might not join an information group even though the proxy has an affinity for the corresponding page cluster since the proxy may choose to join another information group that also accounts for those pages. Let I_{max} be the maximum number of proxies allowed in an information group. The size of each page cluster is determined in such a way that the number of proxies that join the corresponding information group is at most I_{max} (e.g., $I_{max} = 3$ in Figure 1)[3]. As mentioned before, our goal is to reduce both the average number of messages processed by each proxy for updating other proxy's cache status and the size of these messages while maintaining the cache hit rate.

Moreover, we want to minimize the average number of information groups a proxy may join so that the bookkeeping overhead can be minimized. Replications are allowed during grouping (i.e., two page clusters might contain some common web pages) for such a purpose. However, allowing a large amount of replication would cause another problem: the number of collaborating proxies for a web page tends to be small[4]. As a result, the cache hit ratio could be impacted severely. In order to prevent such an occurrence, some mechanism has to be employed to restrict the number of replicas allowed for a page during the web page clustering procedure. Let I_{min} be the minimum number of collaborating proxies required to maintain an acceptable cache hit ratio for a web page ($I_{min} = 2$ in Figure 1). Then, the problem becomes how to guarantee that a web page A can be allowed to replicate in a page cluster PG_i only if it would not cause the size of any information group to be less than I_{min}.

[2]This is feasible since all cached web pages can be stored on disk.

[3]The optimal value of I_{max} is much larger in reality

[4]We will explain it later this section.

Table 1: Statistical Information of Trace Logs

	DEC Traces	ClarkNet Traces	L Traces
Number of Requests	3.54 million	3.3 million	10 million
Number of Proxies	16	2000	546
Number of Clients/Hosts	10,000	21,000	3000
Number of Servers	70,000	1	12,000

Table 2: Page/Request Coverage for Different Frequencies

β	DEC Traces		ClarkNet Traces		L Traces	
	page coverage	request coverage	page coverage	request coverage	page coverage	request coverage
0.005%	9%	60%	10%	58%	10%	62%
0.01%	4%	40%	4%	39%	4%	41%
0.05%	0.4%	17%	0.4%	16%	0.4%	17%

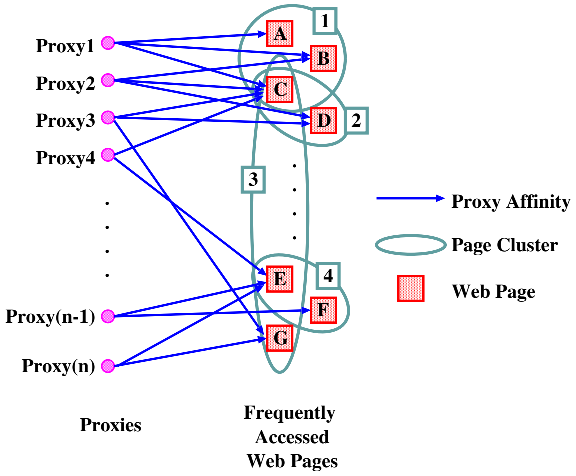

Figure 1: General Scenario

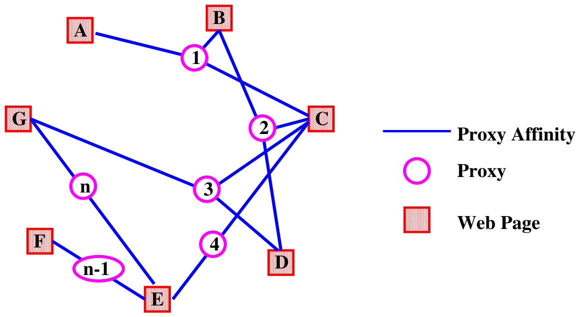

Figure 2: Map to a Hyper-graph

The information group sizes is crucial to the overall performance. We will analyze the effects of different information group sizes and how to determine the optimal size in later sections.

General Approach

This is a hyper-graph partition problem with replication as follows. Each frequently accessed web page is mapped to a vertex; each proxy's profile (i.e., affinity set) is mapped to a hyper-link among corresponding vertices (See Figure 2). The objective is to minimize the average number of information groups a proxy may join in order to cover all its frequently accessed web pages.

A move-based algorithm [1] can be employed to form the page clusters. Let N_p and n_p be the number of proxies and the average number of "frequently accessed" web pages per proxy, respectively. We first group all web pages into N_c clusters without any replication where $N_c = \lceil \frac{N_p \times n_p}{(I_{max}+I_{min})/2} \rceil$ if no page cluster exists previously. Otherwise, the previously page clusters will be taken as the initial grouping. Beginning from this initial grouping, a series of passes are made to improve the quality of grouping by either moving pages among clusters, replicating pages to some clusters, or removing replicas from some clusters. During each pass, pages are successively examined until each page has been examined exactly once. Given a current grouping P', the previously unexamined page with the highest **gain** (defined shortly) is examined and the corresponding action (moving, replicating, or unreplicating) to incur such gain is taken to modify the current grouping. Given the current grouping P', the gain of an action Ac is defined as the reduction of average number of information groups a proxy has to join if action Ac is taken. Note that the gain can be either positive or negative. After each pass, the best grouping observed during this pass[5] becomes the initial grouping for a new pass. This procedure terminates when a pass fails to improve the quality of its initial grouping.

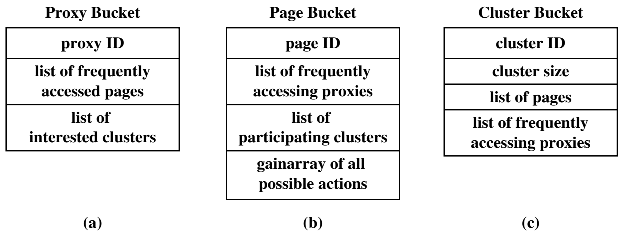

Figure 3: Data Structure

[5] Note that an individual action during a pass might not always improve the grouping quality. But it may create an opportunity for a better grouping to be obtained later.

Some necessary data structures are maintained to facilitate the partition process by a scan of all proxies' messages. A *proxy table*, a *page table*, and a *cluster table* are built for all proxies, for all web pages frequently accessed by some proxies, and for all web page clusters, respectively. Without loss of generality, we assume that each proxy (page / cluster) can be uniquely identified by its ID which is a number between 1 and N_p (N_w / N_c), where N_p (N_w / N_c) is the total number of proxies (frequently accessed web pages / web page clusters)[6]. In the proxy table, each slot, referred to as a *proxy bucket* (Figure 3(a)), consists of three components: proxy ID, a linked list of web pages this proxy frequently accesses, and a linked list of web page clusters to which this proxy has an affinity. In the page table, each slot, referred to as a *page bucket* (Figure 3(b)), maintains the following information:

1. page ID
2. a linked list of proxies which frequently access this page
3. a linked list of web page clusters which contain this page
4. *gainarray*: an array of buckets, each of which contains its gain on some action.

The action on a page can be one of the following.

- move to another cluster.
- replicate in another cluster.
- remove replica from this cluster.

Since the size of the information group corresponding to a cluster can not exceed I_{max}, when a cluster PG_i grows to such a level that its estimated information group size reaches I_{max}, pages in other clusters are not allowed to move to PG_i or to be replicated in PG_i until some page in PG_i is removed from PG_i. Every time an action is taken on a page A, the gains of *relevant* pages (those pages which are also frequently accessed by the proxies which frequently access A) are updated. For example, page B and C are relevant to A in Figure 1 since they are also frequently accessed by Proxy 1 that accesses A frequently.

In general, as shown in Figure 4(a), when a web page A moves from cluster PG_i to PG_j, the information group for PG_j tends to grow since proxies which have an affinity for A may join this information group. On the other hand, the information group for PG_i tends to shrink since proxies which have an affinity for A but not for other web pages in PG_i will withdraw. Similarly, given that $A \in PG_{i1} \cap PG_{i2} \cap \cdots \cap PG_{ik}$ (Figure 4(b)), replicating A to PG_j ($j \neq i1, i2, \ldots, ik$) causes the information group for PG_j to grow and the information groups for $PG_{i1}, PG_{i2}, \ldots, PG_{ik}$ to shrink for the following two reasons.

[6]In a general scenario where this assumption does not hold, a hash table can always be employed to provide a fast mapping between a proxy ID (page ID / cluster ID) to its associated information.

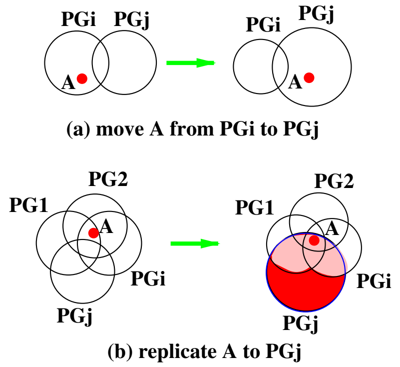

Figure 4: Moving and Replicating a Page

1. Before the replication, all proxies which have an affinity only for A and some other pages in the set $PG_j \cap (PG_{i1} \cup \cdots \cup PG_{ik})$ (light shaded area in Figure 4(b)) will join at least one of the information groups PG_{i1}, PG_{i2}, ..., PG_{ik}. After the replication, some of these proxies may join PG_j instead. Note that, in this case, the average number of proxies in each information group would not be changed since the replication only causes some proxies to move from information groups for PG_{i1}, PG_{i2}, ..., PG_{ik} to that for PG_j.
2. Those proxies which have an affinity for A and PG_j but not for any other web pages in $PG_{i1} \cup \cdots \cup PG_{ik}$ (dark shaded area in Figure 4(b)) will not join the information groups for PG_{i1}, PG_{i2}, ..., PG_{ik} any longer. They just need to join the information group for PG_j now. In other words, the information groups for PG_{i1}, PG_{i2}, ..., PG_{ik} will shrink while the size of information group for PG_j keeps the same in this case. As a consequence, the average number of proxies in an information group decreases.

Therefore, replication tends to reduce the average number of proxies in each information group. Besides the estimated information group size of PG_j, another criteria to determine if A should be replicated in a cluster PG_j is that the estimated information group size of $PG_{i1}, PG_{i2}, \ldots, PG_{ik}$ or PG_j must remain at least I_{min} if the replication is performed.

Performance measurements of our proposed web page partitioning algorithm is shown in Table 3. Here the centralized computing site S is a UltraSparc Workstation which has a 200MB main memory and one single 366MHz CPU. Note that the efficiency comes partially from the trick that new partition is always generated incrementally by taking the previous partition as the initial partition. This partition process does not significantly impact the performance of other workload on the central site and can be further optimized in the following ways.

- The entire partitioning process can be done offline or taken as a background process. If the central site is heavily loaded, it can postpone the computation of its frequently access web pages until it is idle or lightly loaded.

- The gainarray computation can be done parallel. Therefore, multiple machines can be used to share the workload of the partitioning process.

After the page clusters are formed, a server will be chosen to be the coordinator for each information group by S. Usually, the coordinator of an information group is the server which owns the most pages in the corresponding page cluster. Finally, the content of all page clusters and their coordinators are broadcast to all proxies[8]. Each proxy will match its profile with the page clusters and send message(s) to the coordinator(s) of page cluster(s), for which they have an affinity, to join the information group(s).

Since the content of all page clusters could be large, we use a Bloom Filter [3] [15] to encode the contents of page clusters. A Bloom Filter is a method for representing a set $A = \{a_1, a_2, \ldots, a_n\}$ of n keys to frequency membership queries. The idea is to allocate a vector v of m bits, initially set to 0, and then choose k independent hash functions, $h_1, h_2, \ldots, h_k$, with range $\{1, \ldots, m\}$. For each key $a \in A$, the bits $h_1(a), h_2(a), \ldots, h_k(a)$ of v are set to 1. Given a query key b, we check the bits $h_1(b), h_2(b), \ldots, h_k(b)$. If any of these bits are 0, then b is not in the set A. Otherwise, we conjecture that b is in the set although there is a small probability that we are wrong. [11] reports that when the number of hash functions is five and ten bits are used to encode an entry, the error (false positive) is about 0.9%.

6. INFORMATION GROUP MAINTENANCE

Information groups are formed based on page clusters. Each information group is associated with one page cluster. A proxy may join multiple information groups based on its profile and the contents of these page clusters. In the previous section, we described a periodically executed algorithm that globally forms page clusters. The manager broadcasts the page clustering information (i.e., the coordinator for each cluster and the content of each cluster) to all participating proxies. Then each proxy, based on its proxy profile, determines which page cluster(s) it has an affinity for.

A proxy may *join* the information groups for interesting page clusters. There may be several combinations of information groups a proxy can join. The choice of which information groups to join is based on the number of matches between a proxy's profile and the content of a page cluster. We employ a greedy algorithm to determine which information groups a proxy should join. First, a proxy finds a page cluster which contains the maximum number of web pages in the proxy's profile. The proxy joins the information group for that page cluster. Next, the proxy finds another page cluster which contains the maximum number of web pages in the remainder of the proxy's profile. The proxy joins the information group for the second page cluster. This process continues until the proxy joins the information groups for all web pages in its profile.

[8] An alternative would be that the central site S determines the information groups a proxy may join and sends to the proxy only related information. However, the proxy will lose the opportunity to join other information groups when its profile changes because of lack of information about other available web page clusters and their associated information groups.

Note that the proxy profile may change dynamically to reflect the changes of its client access affinity. Then every time the profile changes, the proxy may choose to withdraw from some information groups and/or join some other information groups to accommodate the change of the access affinity. It is possible that, at a given time, an information group may contain more than I_{max} or less than I_{min} proxies due to the changes of proxy affinities. In such a case, a *local reorganization* procedure can be performed on web pages in the web page clusters associated with the oversize and/or undersize information group(s). Due to space limitations, we will not elaborate on this in this paper.

If a proxy wants to join an information group, then the proxy sends a message to the coordinator of that information group. The coordinator keeps a record of which proxies are in the information group. It then sends to the new proxy a list of the members of the information group. And the new proxy sends the intersection of its cache content and the corresponding page cluster to all members in the information group. In turn, all old proxies in the information group also tell the new proxy their cache contents which intersect the corresponding cluster. (These messages can be sent via multicast.) If a proxy wants to withdraw from an information group, then the proxy sends a multicast message to all members (including the coordinator) of this information group to notify this change.

Figure 5 shows the average cache hit ratio as a function of the information group size m. In our simulations, when the contents of a proxy's cache for a page cluster changes by more than 10%, the proxy multicasts the changes to other proxies in the information group for that cluster. We choose 10% as the threshold because it is shown from experiments that the stale hit rate will be large otherwise [11]. For the summary cache, when the overall cache content in a proxy changes over 10%, the proxy then broadcasts the updates to all proxies. In our proposed scheme the updates are only multicasted to the proxies in the associated information group(s). When β decreases, the cache hit ratio increases because more pages are clustered and proxies exchange information on more web pages. It is more likely, when a proxy tries to retrieve a web page, it can fetch the page from another proxy instead of the server. Moreover, with $\beta = 0.005\%$ and a reasonable information group size (i.e., $m \geq 10$ for DEC trace and $m \geq 60$ for L trace and ClarkNet trace), the cache hit rate for our scheme is similar to that of summary cache because the web pages for about 60% of the requests are analyzed and clustered. The rest of the web pages are not "hot", and therefore, the requests for these non-clustered pages will likely result in a cache miss even in the summary cache. In addition, for those "hot" pages, collaborative caching among a relatively small number of proxies is sufficient to maintain a high cache hit rate. Coordinating a large number of proxies in such a process (i.e., the scenario in summary cache scheme) would incur large overhead but little improvement on cache hit rate. We explore this issue more in the next section.

7. WEB PAGE RETRIEVAL

In this section, we first discuss how the content of the cache at one proxy propagates to other proxies, then we will present the detailed procedure for actual retrieval of web pages from

Table 3: **Web Page Partition Cost**

	β	CPU Time on S (min)	Average CPU Time on Each Proxy[7] (min)	Average Message size (KB)
DEC	0.005%	30	10	122
	0.01%	20	10	47
	0.05%	6	1	3
Clark	0.005%	42	7.5	146
	0.01%	24	4	51
	0.05%	5	0.5	4
L	0.005%	38	5	101
	0.01%	20	3	33
	0.05%	4	0.35	2

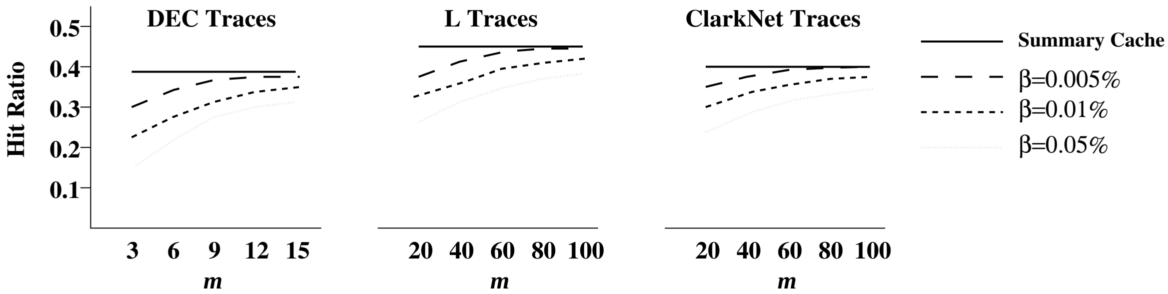

Figure 5: Cache Hit Rate versus the Information Group Size

the server or another proxy. A proxy may join several information groups. When the cache contents of one proxy for one page cluster has changed over a threshold (measured as a percentage), α, then the proxy will multicast the changes to all proxies in the information group (for that page cluster) using a Bloom Filter. This procedure is similar to that used in the summary cache by Fang, et. al. [11] except that in our approach the scope of consideration is each individual page cluster and its corresponding information group. When the cache contents for one page cluster changes by more than α at a proxy, the proxy will multicast the changes (related to that page cluster) to other proxies in the associated information group. In addition, this multicast message can be delivered at a lower priority than other packets to reduce its impact to the normal Internet traffic[9].

When $1\% \leq \alpha \leq 10\%$, the stale cache hit ratios of summary cache and our scheme (with $\beta = 0.005\%, 0.01\%, 0.05\%$) are very similar, which are between 9% to 12% for the three traces. A similar result is also reported in [11]. In the summary cache scheme, each proxy needs to maintain information of all other proxies' cache contents, whereas in our scheme, a proxy only needs to keep information of collaborating proxies' cache contents. Figure 6 shows the different size of cache status information maintained by the summary cache and our scheme. For both schemes, the pages are encoded with a Bloom Filter and each entry consumes 8 bits. It is clear that the summary cache would require much more maintenance effort than our caching scheme especially when the number of proxies is large. ClarkNet traces have the largest number of proxies and the size of cache status information of other proxies ranges up to several GB in summary cache and it could severely impact the proxy's ability to serve pages to its clients. In contrast, with the increase of the number of proxies or the cache size on each proxy, the size of cache status information in our proposed scheme does not increase as fast as the summary cache. This is due to the fact that each proxy only receives the cache status of other proxies in the same information group(s) and only cache status of web pages in the corresponding page cluster(s) is maintained.

[9]We find empirically that the incurred small delay in updating cache status information has little impact on the overall cache hit ratio.

Now we explain how a page is retrieved. Let's assume a client attached to proxy PR_X sends a request for a particular web page to proxy PR_X. If PR_X can not find the web page in its local cache, then it will decide from which remote proxy or the server it will fetch the web page. During this process, PR_X first finds which page cluster the web page belongs to by examining the Bloom Filters. If the page does not belong to any cluster, then PR_X will directly fetch the page from the server. Otherwise, PR_X finds the information group(s) for the cluster(s). (The page may belong to multiple clusters.) Then PR_X ranks all proxies in the information group(s) and the server of that particular web page based on distance. We use the round-trip latency to estimate the distance between two proxies [17] [16]. This does not require extra overhead because the round-trip time can be obtained when the updates of cache contents are sent out and the acknowledge of the updates are received.

PR_X forwards the request to the "nearest" site (proxy or the server) which has the page. If the remote proxy does not have the web page because of (1) cache invalidation (stale hit) or (2) error of the Bloom Filter (false hit), then PR_X will forward the request to the second "nearest" site (proxy or the server). This process continues until the web page is

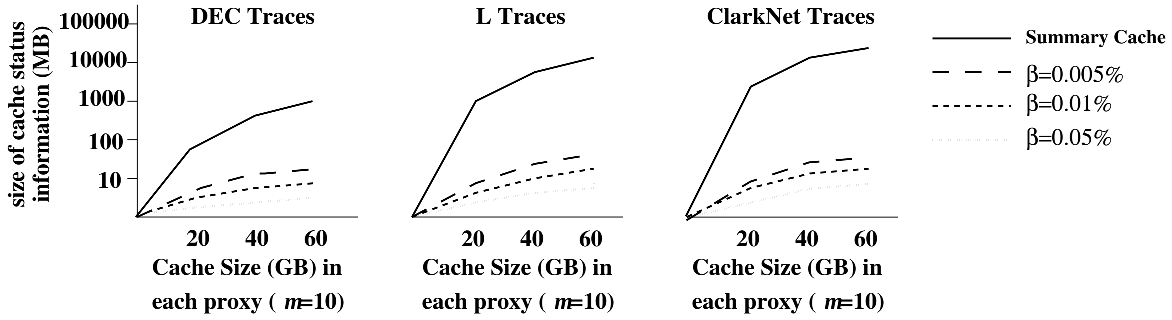

Figure 6: Size of cache status information (Note the y-axis is in the log scale)

retrieved. It is guaranteed that this process will terminate eventually because the server always has the web page.

Figure 7 and 8 show the average number of network messages per user request and the average size of network messages per user request. Here each message processed by a proxy counts as one message. For example, if a multicast message is sent to 20 proxies, then the message is counted 20 times. Our proposed scheme has not only a much smaller number of network messages per user request, but also the message size is much smaller especially when the number of proxies is large (L and ClarkNet traces) due to the cache update information.

Figure 9 shows the average latency for a user request. In DEC traces, there are total 16 proxies, our proposed scheme does not have a significant advantage or disadvantage over summary cache. However, when the total number of proxies is large, our proposed scheme could have a significant benefit over summary cache due to the less network traffic seen by each proxy. The average latency of our proposed scheme varies significantly as a function of the information group size m. When m is too small, the cache hit rate will be low but the network traffic will also be low because each proxy only need to notify a small number of proxies. On the other hand, when m is large, the cache hit ratio is high but the network traffic is also high. From the experiment, we found that 50 to 70 would be the appropriate range for m for the trace data in our test. Moreover, from Figure 9, we can see that the scalability of our proposed scheme is better than that of summary cache. The response time of summary cache scheme increases from L trace to ClarkNet trace because of the increase of the proxies (about four folds). However, the response time of our scheme accurately decrease because of the utilization of the proxy affinity!

8. ESTIMATION OF INFORMATION GROUP SIZE

It is clear that the information group size m plays a decisive role in our scheme and is a prerequisite to partition page clusters. In this section, we propose a method to estimate the optimal information group size. As we explained in Section 3, besides m, the following parameters also participate in the cost function.

- α: Every time a proxy's cache content changes by $\alpha\%$, the proxy will multicast the description of the changes to other proxies in the information group. We set $\alpha = 10\%$.
- q: the average number of outgoing requests (from a proxy) which would cause this $\alpha\%$ change.
- t_s: the average time for a proxy to generate the description of its cache changes and send out as a multicast message.
- t_n: the average network overhead incurred by transmitting this multicast message. Because our scheme requires a much smaller number (and size) of multicast messages for updating cache status changes and these messages are delivered at lower priority, no significant delay for delivery of other packets will be thereby incurred. Thus we will omit it in the cost function (i.e., $t_n = 0$).
- t_r: the average time for a proxy to receive this multicast message and update its local information.
- $Locating\_Cost$: the average time to find whether (and where) an up-to-date copy of the requested web page exists in the information group.
- ω: remote cache hit ratio. It is the probability that an outgoing request from a proxy will be served by another proxy in the information group.
- $Remote\_Cost$: the average time to retrieve a cached web page from another proxy in the information group.
- $Server\_Cost$: the average time to serve a request from the content server.

Therefore, we need to obtain the value of these parameters in order to calculate the optimal m. Fortunately, all of them can be easily collected by each proxy and reported to the central site S together with the proxy profile except for the first time the web page clusters are constructed. At the first time, since no information group exists previously, some parameters have to be estimated from other available information. Each proxy can still collect the value of q, t_s, t_r, and $Server\_Cost$ because introducing information group will not have significant influence to these parameters.

However, the value of $Locating\_Cost$, ω, and $Remote\_Cost$ depends heavily on the formation of the information group. We employ some heuristics to estimate them. For each page

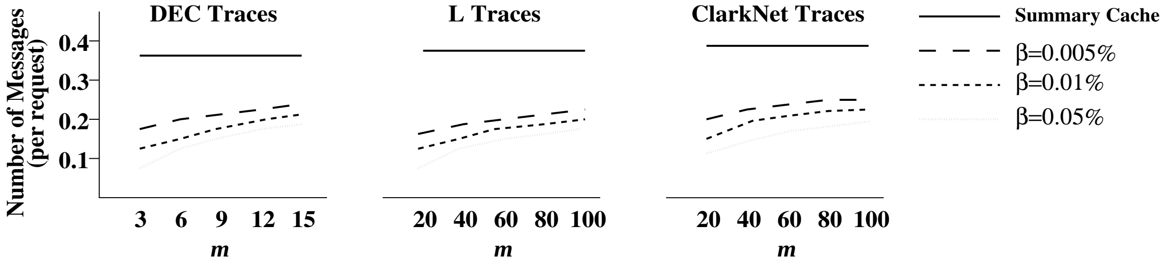

Figure 7: Number of network messages per user request

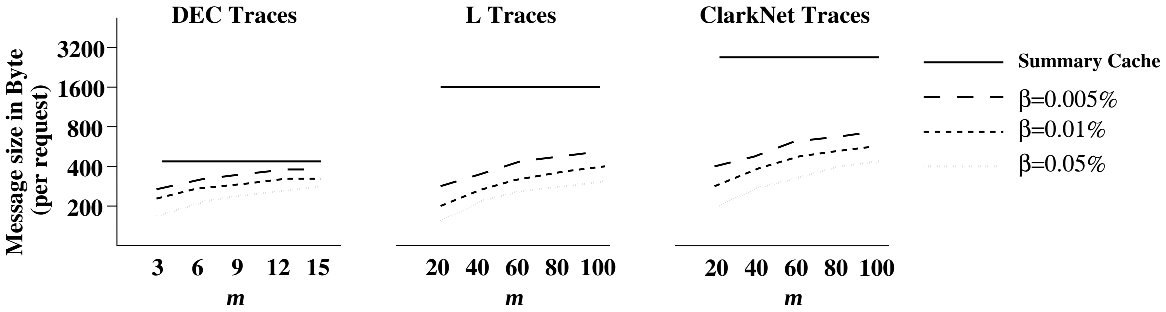

Figure 8: Average bytes of network messages per user request. Note the y-axis is in the log scale

A in the proxy profile, each proxy PR_i calculates the number of requests of A from its local clients and the percentage of time (referred to as $p_i(A)$) PR_i caches a valid copy of A and sends this information together with the profile to the central site S. Note that $p_i(A)$ is essentially the probability that PR_i is able to serve the request of A for other proxy. In addition, each proxy also reports the number of requests to infrequently accessed web pages. All these additional calculations can be easily done locally by a proxy via a sequential scan of its local trace log.

At the central site S, a small random sample of web page URLs are taken from the pool of proxy profiles for the purpose of parameter estimation. The estimation of *Locating_Cost* and *Remote_Cost* are similar. For each web page A in the sample set, the list of proxies which have an affinity for A is generated. This proxy list together with their associated $p_i(A)$ are sent to each individual proxy in the list. Every proxy then computes the expected *Locating_Cost* and *Remote_Cost* of A from its own prospect and returns back the S. Finally, the weighted average of values reported by all queried proxies on all web pages in the sample set is taken as the estimated value of *Locating_Cost* and *Remote_Cost* where the number of accesses of a web page by a proxy serves as the weight factor.

A different strategy is used to estimate the remote cache hit ratio ω. For each web page A in the sample set, we calculate the probability $p(A)$ that, at any given time, at least one proxy caches a valid copy. Without loss of generality, assume that A appears in the profiles of proxies $PR_1, \ldots, PR_j$ and each proxy's web page request behavior is independent of each other. We have $p(A) = 1 - \Pi_{1 \leq i \leq j}(1 - p_i(A))$. The value of $p(A)$ is used to approximate the cache hit ratio of A and the average cache hit ratio ω' of web page in the sample set can be computed by taking a weighted average[10] of all individual cache hit ratios. Note that ω' only captures the cache hit ratio of frequently accessed web pages (i.e., web pages reported in proxy profiles). Since proxies do not collaborate on infrequently accessed web pages, all such requests will go directly to the content server and hence no remote cache hit will happen. Then the overall remote cache hit ratio is $\omega = \frac{\omega' \times N_f}{N_f + N_u}$ where N_f and N_u are the total number of requests for frequently accessed web pages and the total number of requests for infrequently accessed web pages for all proxies[11].

Although the above estimation process is complicated and consumes a certain amount of resources, it only needs to be performed once — the first time the cache collaboration structure is employed. At each consequent page cluster reorganization, each proxy can collect the local value of these parameters and send to the central site together with the profile. The overall value of a parameter can be obtained by taking the (weighted) average of all local values of this parameter. It is clear that little overhead is incurred by such a process. After plugging in these estimated parameters, Function 1 in Section 3 becomes a a function with single variable m. The optimal value of m which minimizes the function can be obtained by examining the derivative of Function 1 (with respect to m). In order to accommodate the evaluated error during parameter estimation and provide more flexibility to the formation of an information group, I_{min} and I_{max} are used to indicate the range of valid information group size. These two parameters are set in such a manner that the value of Function 1 is within an acceptable distance (e.g., 10%) from the optimal one.

[10]The number of accesses to a web page is the weight factor.

[11]Again, these two parameters can be easily computed from the information provided by proxies.

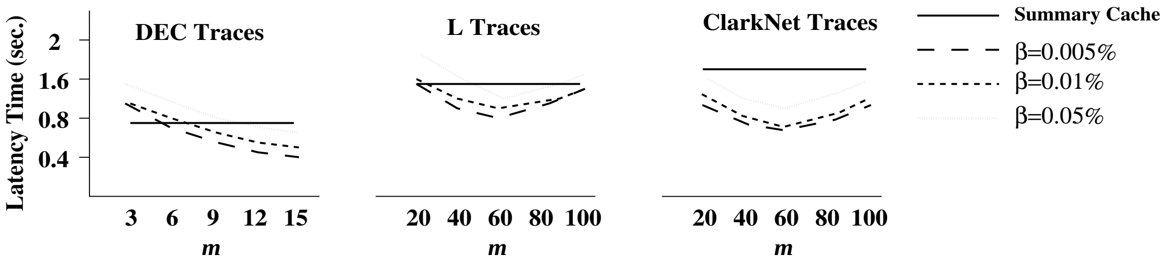

Figure 9: Average latency time a client request

9. CONCLUSION

In this paper, we propose a new scheme which employs proxy affinities to guide the caching structure. Web pages are partitioned into clusters dynamically according to proxy reference patterns. All proxies which frequently access some page in the same web page cluster form an "information group" which serves as the infrastructure for caching collaboration. The dynamic nature of the caching structure provides the opportunity to adapt the changes of proxy affinity efficiently. Trace driven simulation using three web access trace logs shows that this approach can greatly reduce the number of messages and other overhead on individual proxies while maintaining a high cache hit rate. Last but not least, our scheme also provides good scalability with respect to number of proxies.

10. ACKNOWLEDGMENTS

The authors are thankful to Jia Wang, Scott Michel, and Andreas Terzis for their invaluable discussion.

11. REFERENCES

[1] C. Alpert, J. Huang, and A. Kahng. Multilevel circuit partitioning. *Proc. of ACM Conf. on Design Automation Conference*, 530-533, 1997.

[2] P. Barford, A. Bestavros, A. Bradley, and M.E. Crovella. Changes in web client access patterns: characteristics and caching implications. *WWW Special Issue on Characterization and performance Evaluation*, (2):15-28, 1999.

[3] B. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of ACM*, 13(7):422-426, 1970.

[4] L. Breslau, P. Cao, L. Fan, G. Phillips and S. Shenker. Web caching and zipf-like distributions: evidence and implications. *Proc. of 18th Annual Joint Conf. of the IEEE Computer and Communications Societies. (INFOCOM)*, 126-134, 1999.

[5] V. Cardellini, M. Colajanni, and P. Yu. Redirection algorithms for load sharing in distributed web server systems. *Proc. of 19th IEEE Intern. Conf. on Distributed Computing Systems (ICDCS)*, 528-535, 1999.

[6] A. Chankhunthod, P. Danzig, C. Neerdaels, M. Schwartz, and K. Worrell. A hierarchical Internet object cache, available http://catarina.usc.edu/danzig/cache.ps

[7] ClarkNet traces, available at "http://www.acm.org/sigcomm/ITA/index.html"

[8] E. Cohen, B. Krishnamurthy, and Jennifer Rexford. Improving end-to-end performance of the web using server volumes and proxy filters. *Proc. of ACM Conf. on Communications Architectures, Protocols and Applications (SIGCOMM)*, 241-253, 1998.

[9] P. Danzig, R. Hall, and M. Schwartz. A case for caching file objects indide Internetworks. *Proc. of ACM Conf. on Communications Architectures, Protocols and Applications (SIGCOMM)*, 239-248, 1993.

[10] Digital's web traces, available at "ftp://ftp.digital.com/pub/DEC/traces/proxy/webtraces.html"

[11] L. Fan, P. Cao, J. Almeida, and A. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *Proc. of ACM Conf. on Communications Architectures, Protocols and Applications (SIGCOMM)*, 254-265, 1998.

[12] L. Fan, P. Cao, W. Lin, Q. Jacobson. Web prefetching between low-bandwidth clients and proxies: potential and performance. *Proc. of ACM SIGMETRICS Conference*, 178-187, 1999.

[13] S. Ramaswamy. Personal communication, 1998.

[14] T. S. Loon and V. Bharghavan. Alleviating the latency and bandwidth problem in WWW browsing. *Proc. of USENIX Symp. on Internet Technologies and Systems (USITS)*, 1997.

[15] J. Marais and K. Bharat. Supporting cooperative and personal surfing with a desktop assistant. *Proc. of 10th ACM Symp. on User Interface Software and Technology (UIST)*, 129-138, 1997.

[16] S.B. Moon, P. Skelly, and D. Towsley. Estimation and removal of clock skew from network delay measurements. *Proc. of 18th Annual Joint Conf. of the IEEE Computer and Communications Societies. (INFOCOM)*, 227-234, 1999.

[17] V. Paxson. On calibrating measurements of packet transit times. *Proc. of ACM SIGMETRICS*, 11-21, 1998.

[18] J. Pitkow and M. Recker. A simple yet robust caching algorithm based on dynamic access patterns, available http://www.vuw.ac.nz/ mimi/www/www-caching/caching.html

[19] L. Zhang, S. Michel, K. Nguyen, A. Rosenstern, S. Floyd, and V. Jocobson. Adaptive web caching: towards a new global caching architecture. *Proc. of 3rd International Caching Workshop*, 1998.