Dynamic Web Caching

Jiong Yang, Wei Wang, Richard Muntz
Department of Computer Science
University of California, Los Angeles
{jyang, weiwang, muntz}@cs.ucla.edu

Jia Wang
Department of Computer Science
Cornell University
jiawang@cs.cornell.edu

November 30, 1998

Abstract

With the exponential growth of hosts and traffic workload on the Internet, web caching has been recognized as the only real solution to alleviate web page server bottlenecks and reduce traffic over the Internet. In this paper, we propose a new web caching algorithm which employs a well known data mining application, association rule discovery, to guide the caching structure. Association discovery is used on trace data to find the access patterns of web pages, partition the web pages into clusters, and then to form multicast groups based on the page clusters. Furthermore, we develop a new cache replacement algorithm which takes into account not only the local reference pattern, but also whether a page is cached in some other host in the same multicast group. Based on the multicast group, a host can retrieve a web page from its closer peers rather than a distant server. Finally, we employ trace driven simulation to evaluate our web caching scheme using two web access pattern logs and show that our caching scheme provides a significant benefit.

1 Introduction

The number of users connected to the Internet continues to grow exponentially and popular web servers have to serve millions of requests each day. As the traffic on the Internet continues to grow, web page caching has become a popular method to alleviate server bottlenecks and reduce the overall traffic [Cha96] [Zha98] [Fan98] [Pit]. It has been widely recognized that the aggregate amount of data (web pages) is very large. Caching web pages seems to be the only solution to scale a popular web page server to millions of hosts/clients.

As part of the process of retrieving a cached copy of a particular web page, a host has to determine which cache(s) are most likely to contain a copy of the desired page. There are many different ways to locate a cache. One naive way is to ask the server each time. It means that a server has to keep track where a web page is cached. This may again result in the server becoming saturated, and hence the scalability of the server could be impacted. Moreover, retrieving data from a nearby host usually is much faster than fetching the data from a more distant server (less latency and, most likely, higher bandwidth). Thus web caching schemes generally do not require the host to contact the server on every web page request. In fact, only a fraction of web page requests go through the server. To achieve this, some kind of cache discovery procedure by the client is required. Many different approaches have been proposed for discovering a cached copy of a web page. In this paper, we propose a scheme using multicast groups which are based on web page access patterns. In our scheme, a multicast group links hosts which are collaboratively caching web pages that have high correlation of access. (Multicast groups could also be limited within a single administrative domain.)

In this approach a multicast group is formed for a set of pages that are often accessed together. When a host wants to access one of these web pages, it can multicast the request to that group. If some member (host) in the multicast group happens to have a cached copy of that web page, then the cached data will be fetched. However, if some web pages are "hot", then the hosts in the multicast group formed for that set of pages could number in the thousands. In this case if a request was multicast to all hosts in the multicast group, it would incur very high overhead. Thus, when the size of a multicast group would otherwise grow too large (i.e., too many hosts in the multicast group), it will be partitioned into several multicast groups. The size of a multicast group should not be too small either for the following two reasons: (1) If a multicast group is too small, then the cache hit rate will be low. (2) The number of multicast groups will be too large[1] if each multicast group only contains a small number of hosts.

We first describe the technique for grouping web pages according to observed access patterns. To achieve this, we borrow a technique invented in the data mining community. It is known that association rule discovery programs can find rules which specify the likelihood that a page will be accessed given that another page has been accessed. Based on these association rules, we cluster the page groups.

After the formation of these multicast groups, when a host has to find a copy of a particular web page, it joins at least one multicast group for that page if it has not already joined one. The clusters are allowed to overlap and so a page can belong to more than one cluster. The host multicasts a request for a page it does not cache locally to a multicast group associated with that page. If none of the members in the multicast group has that web page, then the request may be propagated to the neighboring multicast groups for the same page group. The process is terminated after a certain number of propagation steps (may be 0) or when the cached page is found. If a cached copy of the page is not found when the discovery process terminates, the host will ask the server directly for the web page. The main issue in this approach is how to form the multicast groups. We claim that they

[1]The overhead of multicast group management will be too large if the number of multicast group is large.

should be formed based on the bandwidth and latency of the communication links between the hosts as well as the recent web page access pattern of the hosts. Before addressing the algorithm for multicast group formation, we first describe the expected benefits of using this scheme.

- Multicasting a request can lead to reduced bandwidth consumption (compared to unicasting requests to the same number of hosts).
- If a host tries to locate a cached web page A, then it is more likely that it will find that page cached in the hosts which may have referenced A frequently in the recent past.
- Since on the Internet, the bandwidth and latency between hosts can vary significantly, it is beneficial to locate a page in a cache from a "closer" (in terms of bandwidth and latency) to the requesting host.

Our scheme takes into account the *distance*[2] between hosts. In other words, we may form several multicast groups for one page group. Different multicast groups for the same page group are formed based on the distance between hosts. We will explain the details of the multicast group formation, cache retrieval procedure, and cache replacement algorithm in the following sections.

The remainder of this paper is organized as follows. The problem statement is presented in Section 2. We discuss some related work in Section 3. Our page clustering algorithm and multicast group formation algorithm are presented in Section 4 and 5, respectively. In Section 6 and 7, we discuss the data retrieval process and cache replacement algorithms, respectively. Section 8 analyzes the results of trace driven simulations. Finally, conclusions are drawn in Section 9.

2 Problem Statement

Before going to a detailed description of our caching scheme, we briefly discuss the motivation and the main differences between caching schemes on the Internet in general and caching schemes for a traditional hierarchical storage. In a typical traditional storage hierarchy, there are main memory, disk, tape, etc. To access data residing in a higher level of the hierarchy usually costs much less than to access data in a lower level of the hierarchy. But the capacity of a higher level is often much less than that of a lower storage level. Thus, the data search scheme starts from the highest level in the hierarchy; if the data does not exist there, then search on the next level in the hierarchy. Since the data is well indexed, it does not take too long to discover whether a given data resides on a given level of the

[2]The distance is measured in term of latency and bandwidth and it is not necessary the same as the physical distance.

storage hierarchy. This process continues until the data is located and loaded into the main memory.

Since the capacity of each level is limited, when some data in loaded into a top level of the hierarchy (e.g., main memory), some other data already resides in that level may have to be replaced. The replacement algorithm usually takes into account the reference stream pattern, and the page estimated to not be referenced for the longest time into the future will be replaced. The order of future references is hard to predict exactly and many approximate algorithms (e.g., least recent used, etc.) have been widely used.

However, these classical schemes can not be directly applied to the Internet for the following reasons. (1) There is no a clear hierarchical model for the storage. If the data is located in one host's main memory, then it is very inexpensive for that host to access the data, however, it will be more expensive for another host to access the data since the major cost can be just to determine which host may have the page cached. (2) It is not practical to exhaustively search all hosts for cached data. (3) The number of hosts connected to the Internet is huge, at least tens of millions currently and most of these hosts operate independently. It is very inefficient for a host to obtain information from all hosts (or even a relatively small subset of hosts) on which to base a page replacement decision. In other words, each individual host has to determine the page to replace with very little coordination with other hosts.

Currently, many web caching schemes have been proposed. (We will discuss some of them in the next section.) However, all of them take the approach that if a host accesses a web page recently, then the page is likely to be cached at that host. This information can be used as a hint by which to direct the future cache discovery. However, if two web pages are highly correlated (i.e., if one host access one of them, say A, then he/she is likely to access the other one, say B, within a short period of time), then we can use the information on whether a host references A (caches A) as an indicator of whether the same host has B in its cache. This is especially useful when individual hosts do not have up to date knowledge of caches on other hosts. However, no other web caching schemes we know of use this information.

Our approach takes advantage of this information by forming web page clusters based on host access patterns. Hosts choose to cache data in some page clusters based on their own interests, e.g., a host caches the data which it will use frequently. And the host implicitly advertises which page cluster(s) it caches (e.g., by joining a multicast group). Later, when some host wants to find data in one of these page clusters, it can contact those hosts which collaborate in caching that page cluster (e.g., via a multicast request) first to see whether the data is cached. The page replacement algorithm of our caching scheme also differs from the traditional ones. In our scheme, when a host has to replace some page, it does not only consider its own reference stream, but also whether that page resides on some other hosts in the same multicast group as well. If that page is the only copy in the multicast group, then the host will be reluctant to replace it. We will present the details of the cache replacement algorithm in a later section.

3 Related Work

In this section, we discuss some other web caching schemes proposed recently. We note that most of these cooperative caching schemes have only focused on how caches cooperate with each other. They do not take into account the users' access pattern. One major difference between our work and these schemes is that our work also focuses on how to organize caches into groups according to observed aggregate, i.e., over all hosts, access patterns to improve the cache hit rate.

3.1 Caching in Harvest

[Dan93] [Cha96] present the caching scheme used in Harvest. 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) and local URLs directly from the object's home. Otherwise, when a cache receives a request for a URL that misses, it performs a remote procedure call 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. Additionally, a cache option can be enabled that tricks the referenced URL's home site into implementing the resolution protocol to allow the cache to retrieve the object from the home site if it happens to be closer than any of the sibling or parent caches.

3.2 Summary Cache

Summary cache is a scalable wide area web cache sharing protocol proposed in [Fan98]. 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 local 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 [Blo70] 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 (Internet Caching Protocol).

3.3 Adaptive Web Caching

[Zha98] proposed an adaptive web caching scheme. Cache servers are self-organizing and form into 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 their entire 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 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.

3.4 Web Caching Based on Dynamic Access Patterns

A "local" caching algorithm based on dynamic access patterns was presented in [Pit]. 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.

4 Page Cluster

Web pages are identified by their URL. In this paper, we only take into account the first three levels of a URL and do not distinguish any difference at higher levels. For example, in this scheme web pages "http://www.cs.ucla.edu/csd/people/faculty" and "http://www.cs.ucla.edu/csd/people/graduates" are considered to have the same identity for the purpose of analysis of web page reference patterns. Page clusters can be formed according to the combined access patterns of all hosts (i.e., reference patterns are superposed). Each host keeps a log of the web pages accessed during each session. Grouping web pages into clusters based on these logs consists of two steps:

1. Mine association rules between web pages accessed during the same session.
2. Group web pages into clusters in such a way that the number of association rules "across" clusters and their associated *interests* are minimized.

Correlations are also called association rules and usually have the form $L \Longleftrightarrow R$ where L and R are sets of web pages respectively. This can be interpreted as "if a host accesses all web pages in L, then with relatively high probability it will access the web pages in R". Each association rule has associated *support* and *interest*. The support is the number of sessions which follow this rule (i.e., all web pages in L and R) whereas the interest is defined as $\frac{N(L \cap R)}{N(L)N(R)}$ where $N(L), N(R), N(L \cap R)$ are the number of sessions containing web pages in L, R and $L \cap R$ respectively. Usually, only association rules with large support and high interest are considered reliable as indications of events that occur often enough to be useful in improving performance [Bri97]. In this paper, we assume that L and R each contains only one URL for the sake of simplicity and clarity of exposition[3].

Many strategies can be employed to mine association rules. In this paper, we use a simple method sketched as follows. Periodically, each host sends its page references log to one of the servers (which server is chosen to approximately balance server load and also on the basis of "distance" from the host). Each time a server receives the logs from all its associated hosts, it runs a data mining algorithm to mine the correlation among web pages accessed in the same session. The reason we employ a distributed algorithm rather than a centralized algorithm is that this process is computationally intensive. (Measured computation times are discussed later.) and it is preferable to balance the workload over all servers. Since a host only sends its log to one server, each server receives a disjoint subset of logs. To mine association rules from the union of all these logs, two steps are required. First, each server mines correlations from the subset of logs it received. Secondly, assume that all servers are ordered in some manner $S_1, S_2, \ldots, S_n$. Every server S_i (except the first server) sends its result R_i to the first server S_1. Then S_1 takes the union of these partial results as the candidate set of association rules, C, for the final result. This set of candidate will be passed through all servers in order to verify their global correctness. At each server S_i, the support and interest for each rule in $C - R_i$ is updated by utilizing the logs S_i has. For rules in R_i, the support and interest can be updated from the partial result S_i calculated previously. If S_i is not the last server, then S_i passes the candidate set with associated supports and interests to the next server S_{i+1}. Otherwise (i.e., at server S_n), the final result can be generated. Rules in the candidate set satisfying the thresholds for support and interest are in the final result R. Since there can be even millions of servers on the Internet, maintaining a total order of all servers is still not feasible. Moreover, logs generated by all hosts can be large and it will consume considerable network bandwidth to pass these logs to servers. We offer the following heuristics to deal with these problems[4].

1. Only a small subset of powerful servers are elected to participate in this process.

2. Each host takes samples from its log and sends this sample set to one server.

[3] A hypergraph will be generated if a rule contains more than two web pages. In such a case, a similar but more complicated algorithm will be employed to group web pages based on the set of association rules.

[4] We will not elaborate on this issue further due to space limitations.

3. A tree structure (i.e., partial order) of participating servers are maintained for the purpose of verifying the global correctness of association rules.

Looking ahead, we will group web pages into a set of page clusters with replication (i.e., a web page can be in more than one group). Given a page cluster g and a rule r, r is said to be *satisfied* by g if all web pages involved in r are in g. A rule r is said to be *violated* if no page cluster satisfies r.

We group web pages into a set of page clusters according to the rules in R. The size of each page cluster is required to be in the range s_{min} to s_{max} which are fixed during the grouping process, where $s_{max} \geq 2 \times s_{min}$[5]. Moreover, we want as many as possible of the rules in R to be satisfied by some page cluster. In addition, given the same number of violated rules, we want the aggregate interest V of those violated rules to be as low as possible. As we mentioned in Section 2, since a host does not have the most up-to-date knowledge about other hosts' cache contents[6], for a given web page A, two kinds of hosts could potentially cache a copy of A.

1. hosts who are known to have requested A.

2. hosts who are known to have requested some other page which is highly correlated with A.

Note that these two sets of hosts are not always the same since some hosts in the second set may request A after the knowledge was updated last time. These hosts are even more likely to still keep the copy of A in their cache since they requested it more recently. Based on this observation, it is preferable to have as many association rules as possible satisfied by some page cluster since web pages in the same rule are highly correlated with each other. V can be regarded as an indicator of cache miss rate in the sense that only the first set of hosts described above can be used to find caches of web pages in those violated rules. Looking ahead, it is shown by trace driven simulation in a later section that clustering web pages according to user access patterns in this manner achieves 20% to 30% more cache hits than the simple page grouping method[7].

Replications are allowed during grouping (i.e., two page clusters might contain some common web pages) to reduce rule violations. However, allowing a large amount of replication will increase the total number of page clusters and hence eventually add severe overhead to the page cluster management. Therefore, the percentage of replication is limited by a parameter r_{max}. This problem is equivalent to a graph partition problem with replication as follows. The rules in R can be mapped to a graph: each web page is mapped to a vertex; there is a edge between two vertices iff the corresponding web pages are involved in a rule in R; the associated interests are taken as the weight of the edge.

[5]How to set these two parameters will be discussed later.

[6]It is not feasible to update such knowledge for each request.

[7]The *simple page grouping method* will be explained in Section 8.

We employ a move-based algorithm [Kri91] [Alp97] to form the page clusters. Let N be the total number of distinct web pages we consider. We first randomly group all web pages into t clusters of equal size without any replication where $t = \lceil \frac{N \times (1+r_{max})}{s_{min}} \rceil$. Note that t is the maximum number of clusters we allow. 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 unreplicating pages 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** 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 A on a web page p is defined as the difference of the aggregate interest of rules satisfied by P' and the grouping after A is taken. In other words, the gain is the aggregate interest of rules becoming satisfied by taking action A on page p minus the aggregate interests of rules becoming violated due to this action. Note that the gain can be either positive or negative. After each pass, the best grouping observed during this pass[8] becomes the initial grouping for a new pass. This procedure terminates when a pass fails to improve the quality of its initial grouping. Finally, page clusters whose size are smaller than s_{min} are combined into larger clusters which satisfy the size requirement[9].

A data structure called *gainarray* is used to facilitate this process. A gainarray is an array of buckets, each of which contains a cell and its gain on some actions. The action on a page can be one of the following.

- move to another cluster.
- replicate in another cluster.
- unreplicate from another cluster.

Since each cluster can not exceed s_{max}, when a cluster PG_i grows to its maximum size, pages in other clusters are not allowed to move to PG_i or replicate in PG_i until a page A in PG_i is moved out or unreplicated from PG_i. Similarly, when the maximum allowed amount of replication is reached, a page is not allowed to be replicated until an "unreplication" is performed. Every time an action is taken on a page A, the gains of relevant pages (those pages participating in some rules with A) are updated.

Since the host access pattern may vary over time, some mechanism has to be employed to maintain the quality of the grouping. A simple approach is to periodically generate association rules from the logs and then apply the same procedure to generate a better grouping by taking the current grouping as the initial grouping. It is shown by experiments in a later section that this step can be done very efficiently.

[8] Note that an individual action during a pass might not always improve the grouping quality. But it may create a chance that a better grouping will be obtained later.

[9] This step is trivial since $s_{max} \geq 2 \times s_{min}$.

5 Multicast Group

As described in the previous section, all web pages managed by a set of servers are partitioned into a set of page clusters. It is likely that if one page is referenced in a page cluster by a host, then that host will access other pages in that cluster. Thus, this forms a basis for searching for a cached copy of the page. For example, if a host tries to reference a web page and it is not cached locally, it may ask other hosts which are likely to cache that page, if those hosts are closer to it than the server. Here the "distance" is measured by the delay and bandwidth of the communication link. This is the foundation for our multicast group caching scheme.

5.1 Multicast Group Formation

Each web page has a home site, which usually is a server site. If a host tries to reference a web page and it is not in any milticast group which is associated with that web page, it first asks the server which contains the web page for that page. (Assume the page is in the page cluster PG_i.) In turn, the server returns the requested page, the information for all pages (entries) in the same page cluster and the id(s) of the multicast group(s) for this page cluster. Initially, there could be no host in a multicast group for a page cluster because no host tried to access web pages in that page cluster.

Recall from the previous section, we only distinguish web pages by the top three levels of their URLs. However, since there could be a large number of entries for a page cluster, the number of bits required to describe all pages in a cluster could be large. Since this information is only used for deciding how well the profile of a user/host[10] matches with entries in a page cluster, this is really a membership problem. To solve this problem, we use Bloom Filter [Blo70] [Mar97] to describe the set of pages.

A Bloom Filter is a method for representing a set $A = \{a_1, a_2, \ldots, a_n\}$ of n keys to support 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. [Fan98] reports that when the number of hash functions is five and ten bits are used to encode an entry, the error (positive false) is about 0.9%.

If the host deems that it is profitable to join the multicast group for the page cluster, then it will do so. The decision is based on a **host profile**. There is a profile associated with each host. The host profile contains the set of pages that the host frequently accesses. If the page is one of the pages which the host accesses often, then the host should join the multicast group for that page. In the case that there exist multiple multicast groups for

[10]The profile of a user/host contains a list of pages that frequently referenced by that user or host.

a page cluster, then the host can choose a multicast(s) based on the "distance" between it and the members in that multicast group(s). Before the first request to a page, a host may join the multicast group for that page if the host profile indicates that this page will be accessed with high probability. After deciding which multicast group(s) it wants to join, the host sends its decision to the server.

5.2 Multicast Group Split

A member of a multicast group can track the change of latency and bandwidth of the communication channels to its neighbors. If it detects that the delay is too large or the bandwidth is too low, it can report this to the server. The server then decides whether the multicast group should be split based on the size, the delay, the bandwidth, cache hit rate etc. If the decision is to split a multicast group, then the server chooses two hosts with the maximum delay, least bandwidth, or a combination of two as two starters. We call the two starters s_1 and s_2. All hosts in the original multicast group (except s_1 and s_2) will try to join one of the two multicast groups which contain $s1$ and s_2, respectively. In the worst case, the size of the newly formed multicast group might be only one less than the original size. Then a new split may be in order. However, in general, the size of the multicast group will be reduced at a much faster pace.

5.3 Multicast Group Roll-over

Every time page clusters are revised, multicast groups are modified accordingly. Let PG be a page cluster and A be a multicast group formed for PG. The information describing pages in PG (after revising) is multicasted to hosts in A. Each host determines whether its profile still matches well with A. If so, that host will remain. Otherwise, it will leave A. If all hosts leave A, then A is destroyed. This case could occur when PG changes drastically.

6 Data Retrieval

In this section, we will discuss the detailed procedure for retrieval of web pages from the server or another host's cache given a page clustering and set of multicast groups. First, assume that client X has already joined multicast groups C and D as in Figure 1 (all multicast groups in Figure 1 are for the same page cluster), and it wants to access a web page. It first multicasts the request to all hosts/clients in Group C and D. After receiving a request for the data, a client replies with "yes" if it has a copy of the data. If the client does not have the requested data, it sends back a "no" message along with the group ids of the multicast groups it is in. When the requesting client receives a "yes" reply, it sends the data request to the host which first sends back the "yes" reply. Then the data delivery process is invoked.

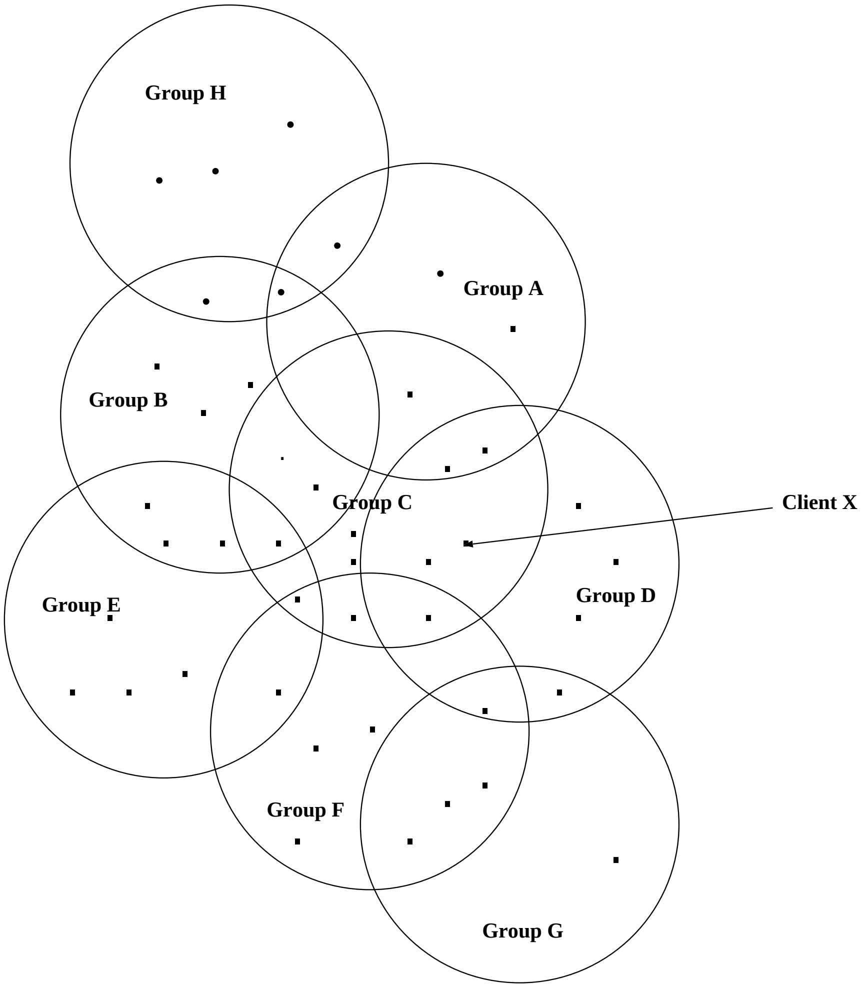

Figure 1: Multicast groups for cache searching

However, if all machines in group C send "no" reply, then X finds all its neighboring multicast groups (i.e., these multicast groups overlap with group C and D, i.e., group A, B, E, F and G). For each neighboring group, X chooses a host as the representative to search for the data on behalf of X[11]. The criteria of choosing the representatives can be based on the time of reply.

If there are a large number of multicast groups involved in retrieving one cache data, the number of messages can be large. For example, if all copies of some data is located many hops[12] away from the requesting client, then there will be many multicast requests, which could result in a significant workload. Therefore, we should put a limit on the number of hops a request can take, which is called **hop count threshold**. Looking ahead, our simulation results show that the optimal performance is achieved with hop count threshold

[11]This is doable since every client include other multicast groups it joins in the reply to X.

[12]Hop is a measurement of the distance between two multicast groups, i.e., group C has two hops away from group H and G while one hop away from group A, B, D, E, and F.

set to 1.

The multicast request plays a role as cache locator which finds which host has the cache. Once a cache is found, i.e., a "yes" reply is received, then the requesting host will send a unicast request to that host which has the cache. This process is described in Figure 2(a). However, if the search does not produce any "yes" reply, i.e., no cache is found. Then the requesting host will send a unicast request to the server for that data. In turn, the server will return the data. This case is shown in Figure 2(b).

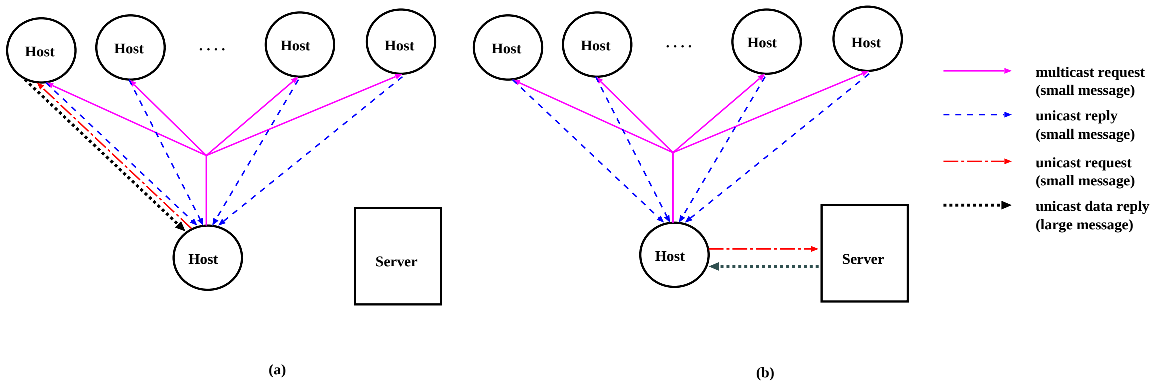

Figure 2: Cache Retrieving

The size of a multicast group is very important. If a group is too small, then the cache hit rate can be low, and the overhead of the cache discovery process can be in vain. On the other hand, if a multicast group grows very large, then the cost for multicast within a group can be large, and in turn the overhead may also be high. In other words, a small multicast group may yield fast multicast within a group, low cash hit rate on average, while a large multicast group would result in higher cache hit rate, but the multicast cost is large within a multicast group.

7 Cache Replacement

We present a cache replacement algorithm in this section. [Fan98] studied four kinds of caching algorithms for five sets of traced HTTP requests and reported that the group-wise LRU cache sharing (e.g., single-copy cache sharing scheme) has the best performance. Here our algorithm also can be categorized as a group-wise LRU caching sharing. For a host, there are two kinds of cached pages. The first kind consists of pages that it will reference soon in the near future. There are many algorithms that can estimate whether or not a page will be used soon, e.g., LRU. Secondly, it is also desirable to cache popular (i.e., frequently accessed by many hosts) pages by some host in the multicast group.

The first kind of cached page deals with the host's own reference pattern and it can be estimated whether a page will be referenced again soon. If a page is deemed replaceable by the local host, then the host checks whether or not another host has requested the page recently. This can be done since requests from the group members are multicast to all hosts in a multicast group. If some other host requested the page, that host has to load the page into its cache soon after it is requested (whether it gets the page from another host or the server is not important). If a page is considered replaceable by its local host and some other host in the multicast group made a recent request for the page, the local host will be more willing to replace it.

For a page to be replaced, it has to meet two conditions. First the local host should regard the page having a low priority for its own use. Secondly some other host in the multicast group made a recent request for that page. However, it is conceivable that no page in a host meets both conditions. Then a host should keep the pages it considers as useful because, after all, the cache resides in its own storage. To achieve this, a host finds a set of pages which are considered least useful by that host. And once a page has to be replaced, the most recently requested by other host in the same multicast group, which is in the low priority set is chosen. This scheme approximates the conditions mentioned above. Note that in this scheme, the popular pages are always cached by some host which either accesses those pages frequently enough or just made the request recently. In the single-copy cache sharing scheme, the initial host which has the page will serve as the server for that page even though it may not access the page any longer. This can increase the host's CPU workload and reduce the local cache hit rate. Our cache replacement algorithm has better local cache hit rate because it reduces the chance that a host caches some pages which are frequently requested by some other hosts than itself.

8 Trace Driven Simulation

We apply our web caching scheme to the following two Internet trace workloads.

- DEC traces [DEC]: Digital Equipment Corporation Web Proxy Server traces, serving about 17,000 workstations. The total number of servers is about 140,000. The trace is for a period of 26 days (Aug. 28 to Sep. 22, 1996) and consists of 24.48 million requests overall and about 3.9 million distinct web pages.

- L traces [L]: a 100 days worth of logs of HTTP requests from about 3000 distinct clients. It consists of total 10 millions requests. There are a total 12,000 servers and 200,000 distinct web pages represented in the trace.

Since the information in the log is not complete, e.g., the log does 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 of servers. We look through all URLs and assume that web pages served by the same server all have the same first level URL components. For example, the URLs of all web pages served by the computer science department server in UCLA starts with "www.cs.ucla.edu". By examining the IP address of each user/host, we discovered that all users belong to 504 subnets in the L trace while there are several thousand subnets in the DEC trace. Furthermore, we assume that hosts in a subnet are connected by a 100 MBit/s fast Ethernet. Then we generate a random connected graph in which each node corresponds to a server or a subnet. The edges corresponding the communication links between the server and client subnets. The bandwidth on these links is 10 MBit/s. We also assume each host has 1MB buffer space for storing the web pages. There are two reasons why we set the buffer space so small. First there may be many programs currently executing on a host, e.g., operating system, compiler, word processor, and so on. The space allocated for the web browser application may be limited to a small amount. Secondly, many web pages are text based, so we set the average page size to be 8KB. However, in the future, many web pages will have multimedia features, and the average web page size may be much larger.

In this paper we assume the average lifetime of a web page is 4 days, i.e., the expiration time of a web page is set to be 4 days. After four days, a cached page will become obsolete and that page will be discarded.

The association rule discovery program is invoked at the beginning of the 11st, 21st, 31st, etc. day for the L-trace data. For DEC trace, the data mining program will be invoked every five days, i.e., the 6th, 11th, etc. day. It is obvious that each time the process discovers partial knowledge based on the trace data of the past 5 or 10 days since not all web pages have been accessed during the past 5 or 10 days. Based on this observation, a rule discovered previously is always kept in the current rule set unless there is enough evidence to show it is no longer valid (i.e., the support of the rule is met but the interest is below the required threshold). We employ a data mining algorithm which is similar to that described in [Agr94]. For the trace we used, the association rule discovery takes between 1 and 1.5 hours to mine all rules on a UltraSparc workstation with 160MB main memory. If a distributed rule discovery program is used , then response time should improve. In all simulations, we set the maximum amount of replication to be 20%. When the graph partition program is executed the first time, the response time is longer than for subsequent executions because in the subsequent execution, the program just revises the previous results. It can take several hours to find the initial page clusters and it takes about one half hour to one hour to find subsequent page clusters. For the first 5 or 10 days, we group all pages in one server to be in one cluster. We use Bloom Filter to describe the web pages in multicast groups. Five hash functions are employed and ten bits are used to encode each entry. We found that it requires approximately 5KB to represent entries in a page cluster on average. This parameter is used for all simulations in this section.

8.1 Cache Hit Rate

We first examine average cache hit rates in our proposed scheme. There are two kinds of cache hit rate. First is the local cache hit rate, i.e., a web page is cached in the local host and the life time has not yet expired. With respect to the maximum number of hosts in a multicast group and the hop count threshold, the average local cache hit rate is the same; approximately 40%. The second cache hit rate is the group cache hit rate[13], i.e., when a host requests a web page in some multicast group(s), the fraction of the times it can obtain the page from another host in the multicast group. To analyze the benefit of applying our page clustering scheme. We compared our clustering scheme against a simple clustering scheme which put all pages served by the same server into the same cluster. Figure 3(a) and (b) show the change of group local cache hit rate with respect to the variation of the maximum number of hosts in a multicast group and the hop count threshold respectively. In Figure 3(a), we fix the hop count to be 1 (i.e., no propagation of requests to neighboring groups) while we fix the maximum number of hosts in a multicast group to be 50 in Figure 3(b). The result is that our page clustering scheme has 20% to 30% improvement over the simple page set scheme.

It is clear that for each page cluster size, the average group cache hit rate increases with the max number of hosts in a group and the hop count. This is due to the fact that when the hop count and group size increases, the number of hosts searched also increases. As a result, it is more likely that the cache will be found by multicast requests. However, it does not mean that the larger the multicast group is, the better performance gains. In fact, when the group size is large, a large number of hosts have to process the request and a large number of communication links have to carry the request messages.

We simulate not only our cache replacement algorithm, but also the single-copy cache sharing algorithm in [Fan98]. In the single-copy cache sharing scheme, a host does not cache the web pages fetched from other hosts. Rather, the other host marks the web page as most-recently-accessed and increases its caching priority. It is reported in [Fan98] that this cache replacement algorithm can achieve the same or better cache hit rate than other cache replacement algorithms in most cases. We found that the single-copy cache sharing algorithm and our caching algorithms have a similar group cache hit rate because whether a page should be replaced out of a group's cooperative cache is determined similar. Therefore, we do not show the group cache hit rate of the single-copy cache sharing scheme. (The group hit rate of our scheme is shown in Figure 3.) However, our scheme has a significantly better local cache hit rate. This can be explained as follows. For example, if host H is the first host in a multicast group to have a cache of web page A, then all subsequent requests of page A from the same multicast group will be served by H and these hosts which uses A will discard A because it has been cached on H. This brings two down sides. (1) All the subsequent cache request from other hosts will result in a local cache miss (and a group cache hit if A is kept by H). (2) It reduces H's own cache size because it has to keep A. Our simulation results also show that the local cache hit rate with single-copy cache sharing is 32%, which is significantly less than our caching scheme. Our caching scheme has a 40%

[13]Since the page hit rate and byte hit rate exhibit the same pattern. Thus we omit the byte hit rate and only present the page hit rate.

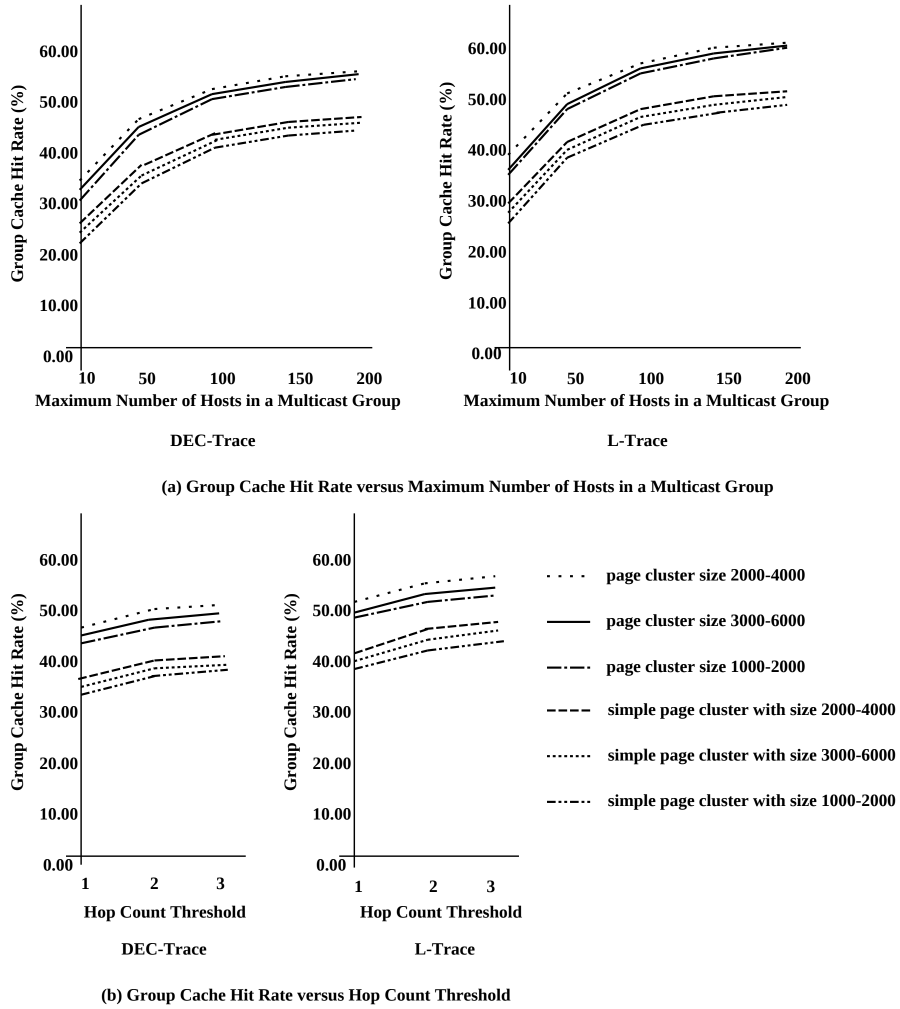

Figure 3: Group Cache Hit Rate

local cache hit rate.

8.2 Number of Messages

In this section, we measure the number of messages for each request. There are several types of messages: multicast small, unicast small, and unicast large as shown in Figure 2 (a) and (b). Table 2 shows the average number of messages for each request. We fix the page cluster size to be 2000 to 4000 and the maximum number of hosts in a multicast group to be 50. From this table, we can see that the number of unicast large messages stays relatively constant because if the page is not cached locally, it has to be sent from some remote site. On the other hand, the number of unicast small and multicast small messages grow exponentially respect to the hop count threshold due to the topology of the multicast groups.

The number of messages only tells one side of the story. The disadvantage of a large

Hop Count Threshold	1 (L)	2 (L)	3 (L)	1 (DEC)	2 (DEC)	3 (DEC)
Number of Unicast Small	21	85	221	23	88	255
Number of Unicast Large	0.61	0.61	0.61	0.58	0.58	0.58
Number of Multicast Small	0.62	2.51	7.33	0.58	2.67	8.01

Table 1: Average number of Messages

number of messages is the overhead of processing these messages (CPU load) and the consumption of the Internet bandwidth. Table 2 shows the average CPU load on each client and server machine and bandwidth consumption on the communication links. (Here we only analyze the "Internet Links", i.e., 10 MBit/s communication links.) From this table, it is clear that the server's CPU load decreases with larger hop count threshold because less request will be sent to the server. However, with the larger hop count the client's CPU increases exponentially because the more message needs to be processed by each client. For the same reason, the bandwidth utilization also increases.

Hop Count Threshold	1 (L)	2 (L)	3 (L)	1 (DEC)	2 (DEC)	3 (DEC)
Server CPU utilization	0.26	0.23	0.21	0.3	0.28	0.26
Client CPU utilization	0.41	0.53	0.74	0.43	0.54	0.76
Internet Bandwidth Utilization	0.22	0.26	0.38	0.2	0.25	0.39

Table 2: Average number of Messages

8.3 Response Time

One of the most important metrics for a system is the average response time, i.e., the average time between the initiation of a request and the receipt of the requested web page. In this section, we analyze the average response time with respect to the hop count threshold and the page clustering technique. We set the CPU time for a host or a server to process a small message is 1 ms and it takes 10 ms to process a large message. The average size of a small message and large message is set to 100 bytes and 8.05 KB, respectively. Figure 4 shows the average response time with respect to the hop count threshold. With larger hop count, the client CPU utilization increases, thus the average response time also increases significantly. For this simulation, we set the size of a page cluster to be between 2000 to 4000 and the maximum number of hosts in a multicast group to be 50. From the figure, it is also clear that our page clustering scheme poses a significant improvement of response time over simple page clustering scheme.

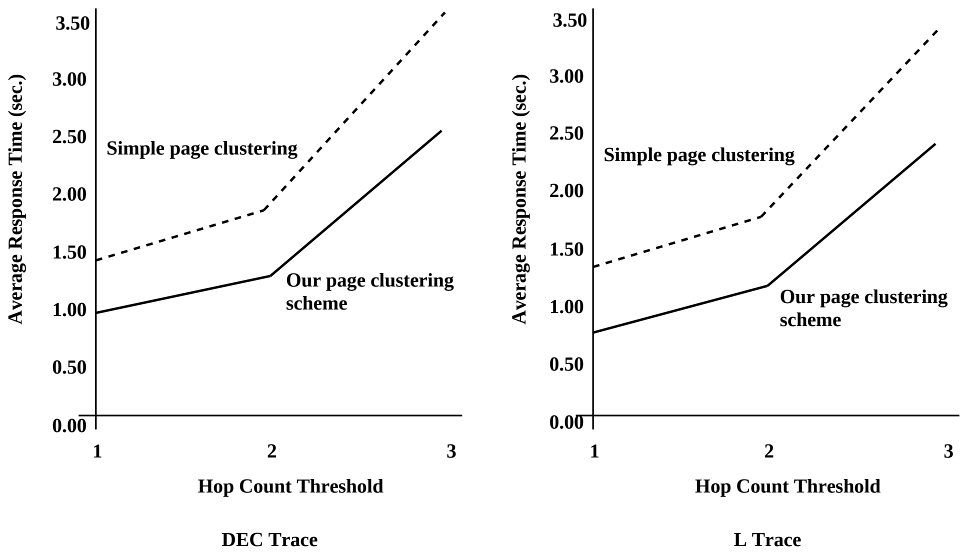

Figure 4: Average Response Time

9 Conclusion

We presented a new web caching scheme in this paper. Our scheme differs from other existing web caching schemes in two aspects. First our scheme takes into account the user references pattern. If a page is not interesting to a host, then the host does not want to know the information about where the cached copy of that page exists. Thus multicast groups are formed based on the web page access patterns. Secondly, we employ a cache replacement algorithm which takes account both the local reference patterns and whether the cache could exist on some other node in the same multicast group. Thus the local cache hit rate and group cache hit rate can be improved. The trace driven simulation also shows the benefit of our web caching scheme. Currently we are implementing a version of the web caching scheme presented in this paper and plan to install the prototype and monitor the performance of our caching scheme. In addition, we will investigate more on web page clustering and exploit the benefit of allowing replication.

References

[Alp97] C. Alpert, J. Huang, and A. Kahng. Multilevel circuit partitioning. *Proc. of ACM Design Automation Conference*, 1997.

[Agr94] R. Agrawal and R. Srikant. Fast algorithms for mining association rules, *Proc. 20th International Conference on Very Large Databases, Santiago*, 1994.

[Blo70] B. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of ACM*, 13(7):422-426, June 1970.

[Bri97] S. Brin, R. Motwani, C. Silverstein. Beyond market baskets: generalizing association rules to correlations. *Proc. ACM SIGMOD Conf. on Management of Data*, 1997.

[Cha96] A. Chankhunthod, P. Danzig, C. Neerdaels, M. Schwartz, and K. Worrell. A hierarchical Internet object cache. submitted to *USENIX 96*, available http://catarina.usc.edu/danzig/cache.ps

[Dan93] P. Danzig, R. Hall, and M. Schwartz. A case for caching file objects indide Internetworks. *Proc. of ACM SIGCOMM*, 1993.

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

[Din96] Adam Dingle. Web cache coherence. *Proc. of the Fifth Intl. World Wide Web Conference*, 1996.

[Fan98] L. Fan, P. Cao, J. Almeida, and A. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *Proc. of ACM SIGCOMM Conference*, 1998.

[Kri91] C. Kring and A. Newton. A cell-replicating approach to minicut-based circuit partitioning. *1991 IEEE International Conference on Computer-Aided Design*, pp. 2-5.

[L] Sridhar Ramaswamy. Personal communication, August 1998.

[Mar97] J. Marais and K. Bharat. Supporting cooperative and personal surfing with a desktop assistant. *Proc. of ACM UIST'97*, October 1997.

[Mot95] R. Motwani and P. Raghavan. *Randomized Algorithms*, mbridge University Press, 1995.

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

[Zha98] L. Zhang, S. Michel, K. Nguyen, A. Rosenstern, S. Floyd, and V. Jocobson. Adaptive web caching: towards a new global caching achitecture. *Proc. of Third International Caching Workshop*, 1998.