On Request Forwarding for Dynamic Web Caching Hierarchies

Cho-Yu Chiang  Yingjie Li  Ming T. Liu  Mervin E. Muller
Department of Computer and Information Science
The Ohio State University
Email: {chiang, yingjie, liu, muller-m}@cis.ohio-state.edu

Abstract

Based on Caching Neighborhood Protocol (CNP), we proposed a Web caching scheme featuring dynamic caching hierarchies as its underlying infrastructure [1]. Dynamic Web caching hierarchies consist of proxy servers building hierarchies on a per request basis, in contrast to static Web caching hierarchies that comprise proxy servers preconfigured into hierarchies. Concerns on overheads and efficiency in forwarding requests individually drove conventional Web caching schemes to use static Web caching hierarchies. Nevertheless, we showed that a Web caching scheme featuring dynamic caching hierarchies can be both efficient and effective in request forwarding.

1. Introduction

Organizing proxy servers [5][8][11] into Web caching servers in dynamic caching hierarchies resort to request forwarding tables (only P4's is shown and others' are similar) to decide to which proxy server an unresolved request should be forwarded. Notice that Figure 1 uses distinct arrowheads to mark request forwarding paths for different origin servers. By observation, a proxy server may be level one cache for one request and level two cache for another in dynamic caching hierarchies. Since each request causes
2. Fundamentals

2.1 Framework

CNP is a suite of protocols that govern operations of caching neighborhoods [1]. A caching neighborhood (CN) is a Web content partial replication group whose members include one origin server and a variable number of proxy servers. Basically, CNP defines data replication and dissemination mechanisms for CNs. Origin servers issue caching instructions to proxy members, maintain coherency of documents rendered by CN members, and control additions/removals of proxy members.

Our Web caching scheme begins with establishments of CNs. CNP-compliant origin servers invite proxy servers to join their CNs. A CN's participating proxy members are called Caching Representatives (C-Reps) for its origin server. For example, in Figure 1 P1 and P2 are A's C-Reps, P3 is B's C-Reps, and P4 and P5 are C's C-Reps, respectively. C-Reps in a CN can be viewed as partially replicate servers of the origin server and therefore, are legitimate candidates to handle requests on behalf of the origin server.

Let's use Figure 2 along with Figure 1 to describe the two request forwarding guidelines of CNP. (i) When a client-side proxy server (CSP) receives a request from a user, if it is not a C-Rep, it has to forward the request to the requested origin server or one of its C-Reps. (ii) When a C-Rep receives a request for one of its represented origin servers, if it has the requested document, the C-Rep returns the document; otherwise, it has to forward the request to the requested origin server. (i) reflects the motivation of proposing the CN concept and (ii) ensures that a request is never forwarded by more than two proxy servers. We elaborate the technical basis behind both principles in the following section.

![Figure 2 A flowchart that illustrates request forwarding process under CNP](image)

2.2 Technical basis

The technical basis for CNP relevant to request forwarding can be elaborated in two aspects.

First, the CN concept was proposed because of the observation that documents provided by more frequently accessed origin servers accounted for the majority of Web accesses [16]. Besides, analysis on proxy traces [9] indicated that Web accesses concentrated not only on a small percentage of origin servers, but also on a small percentage of documents provided by these servers [1]. Furthermore, documents provided by rarely-accessed origin servers were mostly accessed only once [1]. To exploit these Web accessing characteristics, CNP was designed such that frequently-accessed documents on frequently-accessed origin servers would be spread out to their respective C-Reps.

Second, we studied the performance in terms of request latencies for several Web caching schemes with different numbers of levels in caching hierarchies [2]. Our conclusion showed that in general, Web caching schemes performed the best with two-level hierarchies, i.e., hierarchies comprising two levels of proxy servers. Increasing height of caching hierarchies only deteriorated request latencies.

As a result, CNP's request forwarding approach is justified because under CNP, not only is a request always handled by the origin server or one of its C-Reps, but also a request would not be processed by more than two proxy servers.

3. Constructing request forwarding tables

3.1 Basics

Request forwarding at a CSP is a two-step task. First, as a CSP needs to forward a request, it searches its request forwarding table for the requested origin server to obtain CN information of the origin server, including server candidate set. This set includes an origin server's C-Reps that had shorter message turnaround times to this CSP than the origin server did. If there is no entry for the requested origin server in the table, the request is forwarded to the origin server. The CSP would receive CN information of the server along with the response if this server is CNP-compliant. Otherwise, the CSP would find the server candidate set and selects one C-Rep from the set to forward the request to.

It would be great if each CSP can store server candidate sets of all origin servers on the Web in a complete request forwarding table. However, the size of such a table is a major concern because the table has to fit into the main memory for efficient table look-up operations. Two factors decide the size of such a table -- the total number of origin servers on the Web and average number of bytes needed to store CN information of a server. Due to the concern that only studying one set of traces could result in drawing
incorrect conclusions, we examined two sets of proxy traces and attempted to make an estimate of the total number of origin servers on the Web. One set of proxy traces was collected in the period 01/12/99 - 02/07/99\(^1\) by a top-level cache ‘UC’ in NLANR [9] caching hierarchies, and the other was collected in the period 06/18/99 - 07/08/99 by a top-level cache in CA*net II [20] caching hierarchies in Canada, and hence labeled as ‘CA’.

Figure 3 shows relationships between accumulated number of accesses and numbers of distinct origin servers being accessed for the two traces. In the UC traces, the total number of origin servers that had ever been requested in the twelve-day observation period was 210,000. By observation, the curve continued to rise, but showed signs of flattening out. Because we were not able to extend the length of the observation period due to our system constraint, we used 400,000 as an estimate for the total number of origin servers. Assuming that storing CN information of an origin server needed 0.5Kbytes on average. The memory requirement for storing a complete request forwarding table would be 200Mbytes. If we do need to construct a complete request forwarding table, such a huge memory consumption was one major concern and the efficiency in performing table look-ups was the other.

In addition, note that if the CA curve were placed in the figure that contained the UC curve, although CA received much less requests than UC did in a day, its curve would almost overlap the initial segment of the UC curve. Such a fact hinted that both traces might share similar characteristics if we made observations based on numbers of received requests instead of lengths of observation periods measured in number of days. In Figure 3, numbers in parentheses are lengths of observation periods being used.

### 3.2 Pyramid sets

As mentioned earlier, a high percentage of requests congregate on a small fraction of origin servers. Hence, storing CN information of only a percent of all origin servers could be a promising solution. However, we needed to study the relationship between the percentage used to construct a *partial request forwarding table* and the effectiveness of such a table.

Suppose a CSP ranks all origin servers being observed within an observation period by their accumulated access counts. A pyramid set \( S = (CSP,d,l,p) \) is defined as a four-tuple. Here \( CSP \) is the client-side proxy server being considered, \( d \) is the starting day of the observation period, \( l \) is the length of the observation period in number of days, and \( p \) is the percentage used to obtain the elements of the set, which consists of only the top \( p\% \) of the origin servers being observed. A pyramid set, therefore, contains only the most frequently-accessed \( p\% \) of origin servers that were ever requested during the observation period. We attempted to investigate the effectiveness of storing CN information of only elements in pyramid sets in constructing request forwarding tables.

### 3.2.1 Assumptions

To study properties of pyramid sets, we made the following assumptions.

- Since Web accesses are closely related to daily life cycles of human beings, lengths of observation periods were set in multiples of days. We used one day, three days, five days, and seven days as lengths of observation periods. An observation period always started at 12am and ended at the same time of the next day. Because the average number of accesses received per day in UC traces was far less than that in UC traces, we also used 14 days and 28 days in studying CA traces so that we could see if the accumulated number of requests was a more important factor than the length of an observation period.
- To obtain pyramid sets, we tried 1%, 5%, 10%, and 20% in place of the percentage \( p \) in our studies. Percentages higher than 20% were not considered because we already learned from trace analyses [1] that if we ranked both origin servers and documents by their accumulated accesses, top 20% of origin servers provided more than 94% of top 20% of documents, and such top 20% of documents accounted for 93% of documents that generated cache hits.

---

\(^1\)Trace collected on 01/19/99 was missing due to file damage.
3.2.2 Properties of pyramid sets

- **URL covering percentages for pyramid sets**

The URL covering percentage of a pyramid set \( S = (CSP, d, p) \), \( URL_{cover}(S) \), is defined as the percentage of requests that are for elements in \( S \) among all requests received by CSP within the observation period \([d, d + 1 - 1])\), where the first number indicates the day that the observation starts and the second number indicates the day that the observation ends. \( URL_{cover}(S) \) can be regarded as an important indicator for the effectiveness of pyramid sets. URL covering percentages for both UC and CA traces in their respective observation periods are shown in Figure 4. As expected, URL covering percentages increased as lengths of observation periods were extended. In addition to the periods of 1, 3, 5 and 7 days used for studying the UC traces, periods of 14 and 28 days were included for studying the CA traces so that comparisons could be made based on identical number of requests. If we aligned the accumulated total number of requests for both graphs, URL covering percentages for both were very close. The differences in URL covering percentages could be attributed to the disparity of client groups UC and CA terms of using pyramid sets to locate server candidate sets for unresolved requests. Note that \( URL_{cover}(S) \) was derived from analyzing requests already been collected. Since neither is it possible to exactly predict the future, nor does the proxy server have capacity to regularly compute pyramid sets in the course of a day, it would be economical to obtain pyramid sets when the demand to the proxy server is low. Pyramid sets used to construct request forwarding tables should be derived from analyzing collected access traces in the near past.

The effectiveness of a pyramid set \( S = (CSP, d, p) \) can be measured by the function \( Eff(S, t) \), where \( t \) is the day in which the effectiveness of \( S \) is evaluated. The function \( Eff(S, t) \) returns the probability of elements in \( S \) being requested on the day \( t \). That is, \( Eff(S, t) \) represents the effectiveness of using past to predict future. Figure 5 shows measurements of effectiveness of pyramid sets for the UC trace collected on 01/27/99.

3.3 Usages of pyramid sets

Comparing Figure 4 and Figure 5 made us believe that pyramid sets used by CSPs could be obtained from the most recent traces in the past to achieve near-optimum performance. According to studies on UC traces, if a proxy server needs to reach 90% of "hit rate" in searching for an origin server in its request forwarding table, it has to store the CN information for about 20% of origin servers by examining the past traces from the most recent \( 10^7 \) requests. The conclusion is derived from checking Figure 3 that there could be about \( 10^7 \) requests received within seven days in the UC traces. Note that \( 10^7 \) was only an approximate number and the value might change for different proxy servers. Also from Figure 3, the total number of
Therefore, to summarize, we have seen that constructing request forwarding table based on the concept of pyramid sets is a feasible and promising approach. In the next section, we will discuss how to make an effective selection from the server candidate set to forward an unresolved request to.

4. Selecting a server from a candidate set

In this section, based on the assumption that a CSP needs to forward a request and a set of C-Reps are legitimate candidates to receive this request, we present our findings on how to select a C-Rep from the candidates to optimize the request latency. To the best knowledge of authors, latencies for processing requests on proxy servers were not collected in proxy traces available to the public nowadays [9][20]. As an alternative, we divided a day into fixed-length intervals and used total numbers of requests received in an interval to measure concurrent demands for proxy servers in the interval. Since high demands very likely correspond to long request latencies, therefore, we approximate average request latency for requests received in an interval with the total number of requests received in the interval. Based on such an assumption, we designed some server selection heuristics after identifying some characteristics exhibited in the examined proxy traces. We will, as a result, present our findings on the characteristics of proxy traces before engaging in the discussion of selecting among a number of server selection heuristics.

4.1 Daily proxy access distribution

Prompted by the need to look for good heuristics that could approximate demands for proxy servers at a particular instant, we scrutinized proxy traces for time depen-
Figure 7 Distributions of requests measured in fixed-length intervals

dency, day dependency, and weekday dependency. What we mean by time dependency is that if numbers of requests received in adjacent intervals are correlated, by day dependency is that if numbers of requests received in the same intervals on adjacent days are correlated, and by weekday dependency is that if total number of requests received in the same intervals on the same days in adjacent weeks are correlated. For not knowing the granularity of dependencies if they did exist, we used 15 minutes, 30 minutes, and 60 minutes to fix lengths of intervals so that we could count number of requests received in a certain interval. Thus, the course of a day was divided into 96, 48, and 24 intervals, respectively. Part of our analysis results are presented in Figure 6 and Figure 7. Rest of the figures can be obtained from [22]. Since figures for CA were similar to figures for UC, only the latter are shown and discussed.

Figure 6 depicts numbers of requests received in consecutive intervals. The similarity of adjacent vertical impulses indicated that prior latency measurements could be used in predicting request latencies. Using 15 minutes to divide a day into 96 intervals obviously showed the best time dependency among the three different lengths. However, from the viewpoint of reducing the amount of CN information of an origin server to reduce the size of a request forwarding table, the granularity used to divide a day into intervals should be as coarse as possible. That is, longer lengths are favored than shorter lengths because they resulted in less intervals per day.

Figure 7 displays proofs for existence of day dependencies in the traces we studied. The implication was that there was a tendency that could be used to estimate numbers of requests received in different intervals of a day since today’s request distribution with respect to the intervals would be similar to yesterday’s distribution.

during weekends dropped considerably and therefore, affect the day dependency. What we were not able to show in these figures was the weekday dependency. It is a factor that we also took into account when designing server selection heuristics.

4.2 Heuristics and their performance studies

Recall that our research aimed at looking for a C-Rep with least predicted request latency from a handful of candidates. Physical proximity and estimated demands for candidates altogether form a good prediction on request latencies. Assuming that C-Reps in the same server candidate set have about the same turnaround time away from a CSP. Thus, the server of which predicted request latency is the least among all candidates is selected as the C-Rep to forward the request to.

After identifying the dependency characteristics from the collected traces, we compiled the following heuristics to predict concurrent demands for a server at any instant in a day as follows. Note that some of the heuristics carried a few more experimental options in our studies.

Heuristics 1. Use information of request latencies acquired in recent intervals on the same day.

Heuristics 2 Use the concurrent demand measured at the same interval in the proceeding day.

• H2a: skip all weekends;
• H2b: for weekend days, use previous weekend days;
• H2c: for Saturdays, use the proceeding Saturdays'; for Sundays, use Saturdays one day ahead;
• H2d: the original Heuristics 2;

Heuristics 3. Use the average of concurrent demands measured at the same interval in previous few days.

• H3a: average of previous 3 days, excluding weekends;

ends;

• H3b: average of previous 5 days, excluding weekends
Figure 8 Analyses of various heuristics for busy level predictions of UC traces

Heuristics 4: Use the average of concurrent demands measured at the same intervals on the same days in the past few weeks;

• H4a: average of concurrent demands on the same days in the previous 3 weeks;
• H4b: concurrent demand on the same day only in the previous week

To evaluate the effectiveness of these heuristics, we introduced a performance metric, ratio of difference, which measured the degree of differences between predicted value and actual value. It is calculated as follows.

\[ \text{Ratio}_{\text{diff}} = \frac{\text{demand}_{\text{predicted}} - \text{demand}_{\text{actual}}}{\text{demand}_{\text{actual}}} \]

As discovered in our experiments, we learned that Heuristics 1 and Heuristics 3 more accurately predicted demands for server candidates. As a result, we proposed Heuristics 1 required very little memory space for storing CN information. Nevertheless, the major drawbacks of using Heuristics 1 was that in practice a CSP might not have request latencies measured in earlier intervals for all server candidates.

Our intuition that combined Heuristics 1 and Heuristics 3 was proven to be a promising approach. In fact, Heuristics 5 families matched Heuristics 1 in both performance and stability aspects, as shown in Figure 8. However, the tradeoff was that they needed to store the past request distributions for all servers in request forwarding tables. One possible remedy was to use Heuristics 1 whenever possible, and stored CN information needed by Heuristics 5 on the disk as a backup measure.
VI. SUMMARY

In this paper, we discussed issues regarding request forwarding in the dynamic Web caching hierarchies. We started by emphasizing the importance of the problem and providing the background information. Then we identified two major issues of this problem. The first issue is what servers should be stored in the request forwarding table. After having an answer to the question, the other question of our concern is to which server the request should be forwarded. We laid out our scheme of handling the request forwarding by studying the pyramid set first because prior research indicated that only the most frequently accessed servers deserved the attention of caching. From examining the proxy traces, we derived several important properties about the pyramid set, which helped determining the most appropriate pyramid set under the given system resource limitations to achieve the best performance in terms of guaranteeing high URL covering percentage.

For the second issue of handling the request forwarding problem, several heuristics were presented for finding the best forwarding destination server among a group of known candidates. The heuristics were proposed by analyzing the request distributions of the proxy traces. A simple heuristic which uses the prior experience in the previous interval on the same day was found to be among the best, however, it still has some limitations and may not work for all the elements in the pyramid set. By interpreting the outcome of the experiments, we proposed a hybrid scheme which took advantage of not only the prior experience on the same day, but also the experience in the recent past. This heuristics was found to be outstanding except the drawback in needing considerable amount of memory.


[12] Squid Internet Object Cache (http://squid.nlanr.net/Squid/)


