Measurement Based Intelligent Prefetch and Cache Technique in Web

Zheng Zhao, Jie Yang, Song Wang, Gang Zhang, Yantai Shu
Department of Computer Science, Tianjin University, Tianjin 300072, P.R.China
Tel.: +86-22-27404394, Fax: +86-22-27404544
E-mail: zhengzh@tju.edu.cn

Abstract

In Web, users in the same workgroup might have similar interests and habits. In this paper we study intelligent proxy techniques for people who use a proxy to access Web. Our intelligent proxy has two parts: cache and prefetch.

We researched three replacement policies for proxy cache: LRU and two variations of LRU. Our simulation showed that basic LRU produces worst hit rate, while the other two policies achieve roughly the same results. Therefore, we choose a mixture of them as our proposed scheme.

By introducing the prediction algorithm and threshold algorithm, The proxy can predict which Web files will be needed in the near future and download some of them before they are really requested by the user group. We can get a more accurate probability by running it on the proxy server than on the client site, when client user has not visited a file in a Web server often enough.

By implementing the techniques of prefetch and cache on the proxy server, we can reduce the Web latency perceived by users and also the total number of access requests to Web server, thus propose a reasonable way of decreasing their cost to access the Web for developing countries.

1. INTRODUCTION

Generally speaking, people working in the same group may have similar interests and habits. In other words, it's very likely that the pages they've requested may relate to each other or even be the same one from the Web. According to this characteristic, we proposed an intelligent proxy technique for these people, who access the Web through a proxy server.

Compared to the existing proxy techniques, our intelligent proxy has two main distinguishing features. First, we enhanced the general proxy by adding a prefetch function to it. The prefetch technique has been used to combine with the client browser [1], but has never been implemented in a proxy. The proxy, being exposed to the Web accesses of multiple users who have similar interests, has the potential to predict more accurately which pages a user might access next, thus it can prefetch those pages to the user's machine on behalf of the user. Second, we made some improvements on the cache replacement algorithm of the general proxies, which usually use simple FIFO or pure LRU (Least Recently Used) algorithm. We use a variant of LRU algorithm, which achieved a much higher hit rate in the limited cache space.

In this paper, we first analysis group interests distribution, then we design our cache algorithm to be used on the proxy and investigate an approach to reduce the user-perceived latency —by prefetch from the Web server to the proxy. In the last section, we present the implementation scheme of our intelligent proxy server.

2. THE ANALYSIS OF GROUP INTERESTS DISTRIBUTION

We assume that members in one group may have similar interests and will access the Web for similar pages. To analyses this situation we have studied three different groups for their Web access traces. In studying the log files of three months, we count statistical data for all the visited URLs and their frequencies. We define

$$Q_n = \frac{specified\ \ page\ access\ count}{all\ \ pages\ access\ count}$$

$$P_n = \frac{page\ count\ having\ specified\ Q_n}{total\ count\ of\ \ access\ \ pages}$$

$$X_n = \sum_{Q_n > q_n} P_n$$

The relation between X_n and Q_n is shown in Figure 1.

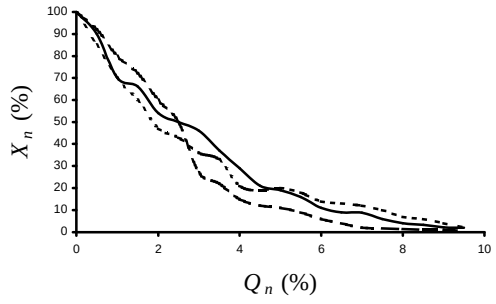

Figure 1: Distribution of group interests

The figure can be used to determine the similarity of the group. In this figure, we have observed that X_n (q_n =5%) of the three groups are 10%~20%, which denotes that group users have enough similarity.

3. IMPROVEMENT TO CACHE ALGORITHM

If a document is cached when it is accessed first time by a user, it will be retrieved more efficiently on subsequent accesses by the same user or by the same group members. A caching proxy does a difficult job. First, in order to have a cache hit, the same document has to be either requested by one user twice or more, or be requested by two or more users. Second, since most web browsers today have a built-in cache, a user is unlikely to request a single document twice in its life cycle. Thus the cache proxy can only have a hit when two or more users request the same document. This reduces the fraction of requests that the proxy can satisfy from its cache, known as the "hit rate".

How effective could a caching proxy ever be? We first simulate a proxy server with unlimited disk space, so that cached documents are never deleted due to space limitations. This gives an upper bound on the hit rate a real proxy server can ever achieve. The simulation input was gathered from several traces of all Web accesses during 3 months. We observed 30-50% hit rate. We then considered the situation when we have only limited disk space. Some files have to be replaced due to space limitations. Three replacement policies were examined: LRU and two variations of LRU.

Our simulation showed that basic LRU produces worst hit rate, while LRU-MIN and LRU-THOLD [2] achieve roughly the same results. So we choose a mixture of LRU-MIN and LRU-THOLD as our proposed cache scheme.

3.1. Cache Replacement Policy

Suppose that the size of an incoming request document is S, and the file is not in the cache.

3.1.1. LRU Policy

Classic least recently used: When the free space in the cache is smaller than S, repeat the following until the free space is at least S: replace the LRU document. (LRU may discard many small documents to make room for one large document.)

3.1.2. LRU-MIN Policy

A variant of LRU that tries to minimize the number of documents replace: Let L and T denote, respectively, a list and an integer. (1) Set T to S. (2) Set L to all documents equal to or larger than T. (L may be null.) (3) Remove the LRU documents of list L until the list is empty, or the free cache space is at least T. (4) If the free cache space is not at least S, set T to T/2 and goto (2).

3.1.3. LRU-THOLD Policy

A variant of LRU can avoid the situation in which a document that is large compared to the cache size causes replacement of a large number of smaller documents. This policy is identical to LRU, except that no document larger than a threshold size is cached. (Even if the cache still has room.)

3.2. Simulation Results

We compared document replacement policies LRU, LRU-MIN and LRU-THOLD, to identify which maximizes hit rate and minimizes disk size required for the cache. The results are in Table 1.

Table 1: Comparison of three cache replacement policies

	Hit Rate (%)		Cache Size (Mbytes)		Lifetime (Hours)	
	Min	Max	Min	Max	Min	Max
LRU	23.8	26.4	22.9	61.4	1.3	15
LRU-M	27.4	40.8	23.0	62.8	2.8	47.3
LRU-T	29.5	32.9	14.8	26.2	7.2	151.6

In this table, given an infinite disk space we confirmed that a caching proxy has an upper bound of 30-50% in its hit rate as described in the reference [3]. In no case did LRU outperform the other replacement policies in Table 1. Given that the optimal threshold for LRU-THOLD is a function of the workload and the ratio of the disk size available for the cache compared to the cache size needed for no replacement, LRU-MIN is the best policy. It requires no parameters and achieves the best performance most of the time. On the other hand, LRU-THOLD achieves dramatically smaller cache sizes with a small penalty in hit rate compared to LRU-MIN, thus LRU-THOLD is recommended when disk size is limited compared to the cache size required for no replacement. Finally, the lifetime reported in the table shows that a document stays in the cache for greatly varying times — as much as ten times longer for LRU-THOLD compared to LRU. The short lifetimes for LRU might explain why its hit rates are never higher than the other policies: documents are replaced too frequently.

Therefore we decide to use the adaptive policy: use LRU-MIN until the cache size approached 100% of the available disk size and then change to LRU-THOLD with a threshold that is gradually reduced until the cache size reaches a low water mark.

4. THE PREDICTION ALGORITHM

The prediction algorithms in this study observe information collected from past accesses from all the users to predict what each user might access next. The proxy needs to log all HTTP requests of the users and adapts its prefetch activity based on them. In our scheme, the proxy is required to maintain two kinds of counters, the page counters and the link counters, to keep track of each user's access history. Each page A is associated with a page counter C_A. In addition, if page B can be accessed directly from page A, in other words, there exists a hyperlink of B on page A, then we use counter $C_{(A,B)}$ to denote this link. Whenever page A is accessed, the counter C_A is increased by one. Similarly, the counter $C_{(A,B)}$ is also increased by one, if page B is accessed by clicking on the corresponding link on page A. The cumulative value of counter C_A and $C_{(A,B)}$ can be achieved by adding up each user's C_A and $C_{(A,B)}$ respectively.

We use conditional probability $P(B|A)$ to denote the probability of "a user is to access page B right after he or she accesses page A". Therefore, the personal access probability is obtained in the following way. When a page A is being viewed by the user, for each page B_i linked to A, the access probability of B_i is computed as $P(B_i|A) = C_{(Bi,A)}/C_A$.

As described above, the group using the proxy has very similar interests. Therefore, the definition of group access probability has significant meaning in the proxy-based prediction algorithm. Assuming there are k members in the group, the group access probability of B_i is defined as

$$P(B_i \mid A) = \frac{\sum_{j=1}^{k} C^{j}_{(A,B_i)}}{\sum_{j=1}^{k} C^{j}_{A}}$$

where C^{j}_{A} denotes the jth group member's C_A.

We have three choices to define our final used access probability. The first one is to use personal access probability p_u. It is known that the accuracy of prediction is closely related to personal interests, which is well indicated by this probability. However, when the user's Web access history information is rather scant, the probability may not really reflect his or her real interests. The second choice is to use the group access probability p_g. Just as the group has some common interests, the accumulative data can represent the popularity of the pages over the group members. Therefore, this probability may reflect the user's interests to a certain extent, especially, the more similar the group members' interests are, and the better this probability fits. Last but not the least, we introduce the third choice, which we have adopted in this paper, integrating the advantages of first two choices. By introducing a weight b, here we present the integrated access probability as

$$p = b\,p_u + (1 - b)\,p_g \qquad (1)$$

where $0 < b < 1$. We can emphasize the personal data or group data by adjusting the value of b.

5. THE THRESHOLD MODULE

Our threshold algorithm is based on that described by Zhimei Jiang [1]. However, there are a few noteworthy differences. First, their scheme was designed for use by a single user to prefetch files on the Web, in which the prefetch engine was located on the user's local machine. Our model is to put the prefetch engine on the proxy server, taking advantage of the accumulative data collected on the proxy to achieve a better prediction result. Therefore, our threshold algorithm is designed for users in a group, who share the same proxy server. Second, by adjusting the parameter b, we can obtain their results as a limit case of our algorithm; that is to say, our model has a wider application in some senses.

In order to get a better tradeoff between system resource usage and latency, our prefetch strategy first predicts which files are likely to be needed soon and chooses some of them to download beforehand. The first part of this task is accomplished by the prediction module that we discuss in the last section. In the threshold module, we determine the prefetch threshold for the Web server in real time to determine which files to prefetch.

We use the term cost to measure the system performance, which is comprised of the delay cost (a_T \$/time unit) and the system resource cost (a_B \$/time unit). The delay cost indicates how valuable the time is to the user. The system resource cost includes the cost of processing the packets at the end nodes and that of transmitting them from the source to the destination. In this section, we study how to determine which files to prefetch in order to minimize the average cost of requesting a file for several system models.

5.1 The cost function C

For a given system, let l be the arrival rate of user requests for pages when no prefetch is applies. We assume that prefetch does not affect the user's behavior regarding the likelihood of accessed pages. In other words, when prefetch is employed, users still issue requests at rate l in the same pattern, although they can get some pages faster due to prefetch. Let the arrival rate of normal requests and prefetch requests are l_1 and l_2 respectively. Hence the rate at which user requests are satisfied by the prefetched files is $l - l_1$, which is simple $p\,l_2$ because prefetched pages are eventually requested by the user with probability p, where p is the access probability of the prefetched pages. Thus

$$l_1 + p\,l_2 = l, \text{ or}$$
$$l_1 + l_2 = l + (1 - p)\,l_2 \qquad (2)$$

In a Round-Robin processor-sharing system, the average response time for requests requiring an average of

x time units of processing is

$$t = \frac{x}{1-r} = \frac{s}{b(1-r)} \quad (3)$$

where r is the system load, s is the average file size, and b is the system capacity. For the system shown in Figure 2, $r = s(l_1 + l_2)/b$. Therefore, the cost of a normal request, which is the sum of the system resource cost and the delay cost, becomes

$$c_1 = a_B \cdot s + a_T \cdot t = a_B \cdot s + a_T \cdot \frac{s}{b-(l_1+l_2)} \quad (4)$$

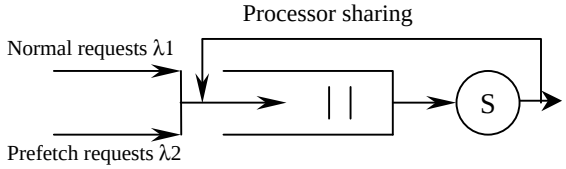

Figure 2: The prefetch system model

We can see that in the above equation, as more files are prefetched, the cost of normal requests increases because prefetch increases the system load, which will increase the delay of retrieving files.

The average cost of a prefetched request is

$$c_2 = a_B \cdot s \quad (5)$$

We obtain the final average function of cost as follows:

$$C = \frac{l_1 \cdot c_1 + l_2 \cdot c_2}{l}$$

$$= \frac{s}{l}\{[l+(1-p)l_2]a_B + \frac{(l-pl_2)a_T}{b-[l+(1-p)l_2]s}\} \quad (6)$$

Put (1) into (6), we obtain our objective cost function.

$$C = \frac{s}{l}\{[l+[1-b\cdot p_u-(1-b)p_g]l_2]a_B + \frac{[l-(b\cdot p_u+(1-b)\cdot p_g)l_2]a_T}{b-[l+[1-b\cdot p_u-(1-b)p_g]l_2]s}\} \quad (7)$$

5.2 The optimum value of the prefetch rate l_2

Let's assume the value of p and l are known, we wish to find at which point of l_2, the average request cost in the system can get its minimized value. From the equation (7), we take the second order derivative of C with respect to l_2, and obtain

$$\frac{d^2C}{dl_2^2} = \frac{2s^2}{l}[\frac{a_T(ls-b(bp_u+(1-b)p_g))(1-bp_u-(1-b)p_g)}{(b-s(l+l_2(1-bp_u-(1-b)p_g)))^3}] \quad (8)$$

The prerequisite of a system to be stable is that $l_1 + l_2$ must be less than b/s, that is to say

$s(l + (1 - b\,p_u - (1 - b)\,p_g)\,l_2) < b$

Therefore, as long as $ls < b\,(b\,p_u+(1 - b)\,p_g)$, the value of (8) will definitely below the zero.

It implies if $ls < b\,(b\,p_u+(1 - b)\,p_g)$, the equation (7) will get its maximum value at the zero point of $\frac{dC}{dl_2}$. Therefore, we can get the critical value l_2', which is

$$l_2' = \frac{1}{s(1-bp_u-(1-b)p_g)}(b-ls-\sqrt{\frac{a_T(b(bp_u+(1-b)p_g)-ls)}{a_B(1-bp_u-(1-b)p_g)}}) \quad (9)$$

Since function (7) is maximized at l_2', where $\frac{dC}{dl_2} = 0$ for $b\,(b\,p_u+(1 - b)\,p_g) < ls$, it follows that the cost decreases as l_2 increases for $l_2 > l_2'$. Specifically, if $l_2' \le 0$ for the given p, l, and r ($=a_T/a_B$), then for any l_2 in the range of $[0, l/p]$, the higher the l_2, i.e. the more that files with access probability p are prefetched, the lower the cost is. Thus, for the given p, l, and r, if $l_2' < 0$, then prefetch all the files with access probability p will minimize the cost.

5.3 The prefetch threshold H

Let us now find the prefetch threshold H such that the cost can be minimized by prefetch all the files with access probabilities p, for p greater than H. From equation (9), we obtain that $l_2' \le 0$ if and only if

$$(b\,p_u + (1-b)\,p_g) \ge 1 - \frac{(1-r)r}{(1-r)^2 b + r} \quad (10)$$

where $r = ls / b$. We then set the prefetch threshold to be

$$H = 1 - \frac{(1-r)r}{(1-r)^2 b + r} \quad (11)$$

Equation (10) shows that if the access probability p is greater than or equal to the threshold H, then $l_2' \le 0$ according to (9). Moreover, following our previous analysis, this implies that prefetch all the files with access probability p minimize the cost, for $p \ge H$. The threshold H is plotted in Figure 3 as a function of system utilization r for several different values of r.

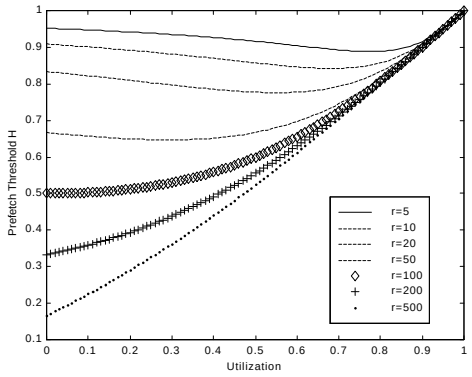

Figure 3: Prefetch threshold H as a function of utilization for different value of r.

Figure 3 shows that as system load r increases, the prefetch threshold tends to increase as well, which means that fewer files should be prefetched. But the increase is

not monotonic for small values of *r*. The reason is, in those cases, when the load is low, prefetch does not save much time. As the load increases, it takes longer to transmit files and prefetch can save more time, thus the threshold decrease. As the load continues to increase, prefetch files will add relatively long delay to the normal user requests, so the threshold needs to be increased again.

In determining the prefetch threshold, we should also take into consideration other network performance parameters, such as Web server load, number of routers in the end-to-end connection, bandwidth of links between routers and volume of traffic on each router. With the aid of some existing measurement software, we can obtain the accurate value of these data.

Here we use Response-time as composite index of network performance. We introduce a network performance factor f_n. The new threshold H_R is obtained by the following function:

$$H_R = f_n \cdot H \qquad (12)$$

The relationship between Response-time and f_n is shown in Table 2, which is determined by the thousands of measurement. Our results showed that settings in the table could achieve optimized effects.

Table 2: Relationship of Response-time and network performance factor f_n

Response Time (ms)	<100	100-300	300-500	500-1000	>1000
f_n	0.8	0.9	1	1.1	1.2

6. IMPLEMENTATION OF INTELLIGENT PROXY

Conceptually, a proxy server transfers characters between the clients' network application and the remote web servers. To provide a full-duplex connection between them, a proxy server must wait for and read data from two sides. Because the proxy cannot know which source of data will become available first, it cannot block indefinitely waiting for input from one of the two sources without also checking for input from the other.

The proxy server should be used for a group of people and an iterative implementation can perform poorly because it requires a given client to wait while it handles all prior requests. And the prefetch function will increase the response time for multiple users. A concurrent implementation avoids long delays because it dose not allow a single client to hold all resources. Finally, concurrency can help designer separate control and processing from normal input and output. Thus, we choose the concurrent and connection-oriented implementation for better-observed response.

We employ multithread method to realize such target. We have two main threads: one to read and manipulate console commands and the other to accept clients connecting request. When a client send requests, the server begin two new threads for every client, one for transferring data from the web server to the clients and the other for transferring data reverse direction. The proxy server responses as soon as possible whenever data is available on either end.

The prefetch function includes three main parts: the management of prefetched files, the interaction of prefetch threads and normal fetching threads, and prefetch algorithm. In order to control all the threads, we design global structure links to control all prefetch threads and other threads. Every user has one link. The link node is called as "Control Node". Each thread has one Control Node. A Control Node includes data fields to record threads' status and properties. It also has data fields for signals in order to deal with multithread synchronization and mutual exclusion.

In the next section we discuss the details of our implementation.

6.1 The computation of predictive access probability:

In order to compute the final used accessed probability in formula (1) we have to record all the access history for every user's personal counters and group counters as well. Using all these data, we can get the personal access probability and the access probability of the special group. As to the weight β, we choose an adaptive method to determine it. When one special user uses the proxy for longer time, we can get enough historical data to predict for him. That means the weight β will be increased slowly. We choose the following steps to determine the weight, and the steps are proved effective in practice. In following, we use C_u to denote the access frequency for a special page of one single user and use C_g for the collective access frequency of group.

$$b = \begin{cases} \dfrac{C_u}{C_g}, & C_u < 10 \ \textit{and} \ C_g > 100 \\ \dfrac{2}{p}\tan^{-1}\left(\dfrac{C_u}{10}\right), & C_u > 10 \\ \dfrac{2}{p}\tan^{-1}\left(\dfrac{C_g}{10}\right), & C < 10 \ \textit{and} \ C_g < 100 \end{cases}$$

When C_u is less than 10 and C_g is more than 100, the group access probability will take the significant part in the final probability. If C_u is less than 10 and C_g is also less than 100, we consider that using C_g is more accurate. This situation will vanish when the proxy server is used.

6.2 The modification of computed H_R

In formula (12) we get the threshold H_R for predictive

purpose. In practice, we have to include the influence of computer performance. We must take into account the number of concurrent multithreads, memory used and other network performance parameters.

To enhance the proxy server's whole performance, we arrange the threads to several ranks having different processing privilege. All the normal fetching threads have the highest level privilege and the prefetch threads have lower privilege sorted according to the access probability. The privilege will induce a bias in manipulation of perspective prefetch acts.

When the CPU is rather busy to deal with clients' normal requests, we have to reduce prefetch threads. So in practice we modify the threshold H_R to a little higher as follows: $H' = H_R + 0.1 * H_R * U$. The U is defined as the utilization of computer resource and can be computed approximately using a given number dividing the number of current threads. In our realization, we use a computer with Intel Pentium II 350, 128M RAM and 6.3G HD as a proxy server for 20 users. We set the given number to 500. In practice, the number of threads existed generally is less than 150 and the clients are satisfied with the response time.

Figure 4 is the logic diagram of our prefetch proxy server.

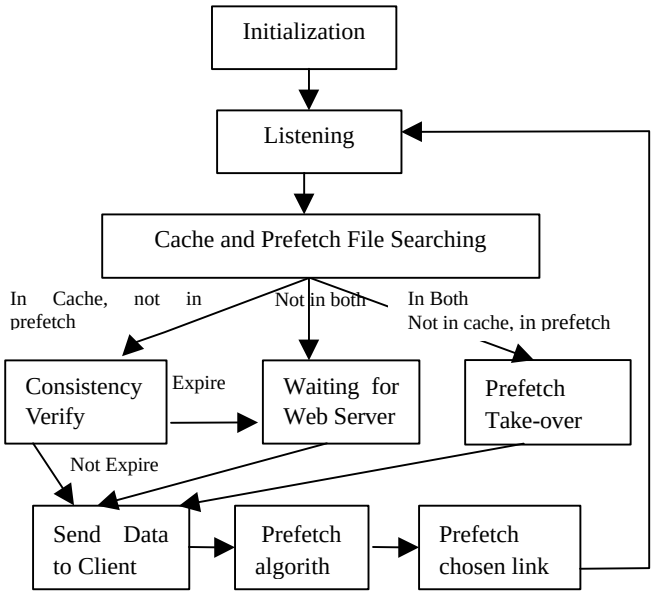

Figure 4: The implementation of intelligent proxy server

7. CONCLUSION

The Web traffic continues to increase at exponential rates [4]. Caching documents closer to users reduces the number of server requests and the traffic associated with them. Unfortunately, several recent studies suggest that the maximum hit rate achievable by any caching algorithm is usually no more than 30% to 50%. The reason is simple: Most people browse and explore the web, trying to find new information.

One way to further raise the caching hit ratio is to anticipate future document requests and prefetch these documents in a local cache. Thus, successful prefetch plus cache reduce the web latency observed by users, and lower both server and network load.

It is known that limited network bandwidth in the developing countries are the main reason to the Web access latency perceived by the users. Our intelligent proxy techniques in the Web can reduce the users' waiting time, and reduce the total number of access requests to Web server and network communication cost. One practical application is for people in developing countries to make frequent access to the Web servers to obtain technology and economic information from developed countries.

ACKNOWLEDGMENTS

This research was supported in part by the National Natural Science Foundation of China (NSFC) under grant No. 69672031. Here we thank Oliver W. W. Yang, professor of School of Information Technology and Engineering at University of Ottawa, who provided useful comments on a draft of this paper.

REFERENCES:

[1] Zhimei Jiang, "An Adaptive Network Prefetch Scheme," *IEEE International Conference on Communications*, Part 1 (of 3), vol. 1, June 8-12 1997.

[2] Marc Abrams, Charles R. Standridge, "Caching Proxies: Limitations and Potentials," *http://ei.cs.vt.edu/~succeed/www4/www4.html.*

[3] S. Glassman, "A Caching Relay for the World-Wide Web," *In First International World Wide Web Conference*, pp. 69-76, May 1994.

[4] J. Gwertzman, "Autonomous Replication in Wide-Area Networks," *Technical Report*, Harvard University, pp. 17-95, 1995.