A New Prefetch Cache Scheme

Shun-Zheng Yu and Hisashi Kobayashi
Department of Electrical Engineering
E-Quad, Princeton University, Princeton, NJ 08544

Abstract-**The criterion that existing prefetch schemes apply in prefetching documents from the origin servers into a proxy Web server is usually the probability of each document being accessed in the near future or the popularity of the document. This criterion is not optimum in minimizing the average access latency or maximizing the average hit probability because the factors that affect on the latency and the hit probability also include the response time and the updating cycle of the documents. In this paper, we derive expressions for the average latency, hit probability, cache capacity and required bandwidth for a general prefetch scheme. A new prefetch scheme that combines access probability, response time and updating cycle to determine the lowest average latency or the highest hit probability is proposed. Finally, some numerical results are presented. The required parameters of the prefetch scheme can be simply derived from the log data of the cache. Thus, our scheme can be implemented in practice.**

I. INTRODUCTION

The World Wide Web traffic is increasing very rapidly. Caching is a useful technique to reduce the Web traffic and speed up the Web access by storing copies of popular Web documents in a proxy server close to the end users. Most Web caching systems in use today are demand driven, i.e., documents are fetched or validated only when requested by users. However, the cache hit probability that can be achieved by such a caching scheme is usually less than 50%. We can increase the hit probability by prefetching those documents which are very likely to be requested in the near future and can reduce latency, i.e. the user's waiting time in accessing the desired documents.

There are many prefetch schemes reported in the literature. We believe that they can be divided into three types. The first type is *predictive prefetching,* which is usually based on the prediction of the access probability that a document will be requested in the near future. The prediction of the access probability can be based on the client's access history [1][2], links between the contents [3], and relationships between the contents and some other threads such as the client's mobility [4]. If its future access probability is high, the document will be prefetched into the cache. The second type is *popularity-based prefetching*, which depends on how frequently a given document has been accessed recently [5][6][7]. The most popular documents will be prefetched into the cache. The third type is *interactive prefetching,* which is based on the interaction between the cache server and the clients [8][9][10][11]. Activities between the cache and the clients are used to decide whether to prefetch specific documents. The prefetch decisions can be made either by the cache or the clients. This type of scheme can take advantage of the idle time between the moments when the client requests to push or pull the documents to the clients, and thereby can decrease the access time [12].

As noted earlier, the criterion based on the probability with which a document is accessed in the near future or the popularity of the document is not optimal in minimizing the average access latency or in maximizing the average hit ratio. There are other important factors that should be considered. For example, if a popular document has a long freshness lifetime, it needs not to be prefetched frequently because its current copy in the cache will remain "fresh" for a long time, hence can be accessed many times before it expires. Thus, the hit probability associated with such a document is high and the access latency is low. The latency to access a given document is also related to the size of the document: a large multimedia document, for instance, may cause high latency if it has no copy in the cache when requested, because it takes much transmission time to transfer such document into the cache. Therefore, the important factors that we should consider in minimizing the latency or maximizing the hit ratio include the access probability, the update cycle and the size of each document.

In this paper, we propose a new prefetch scheme. First, in Section II, we build a general caching model based on the caching mechanism of HTTP/1.1, and use this model to derive expressions for the average latency, the hit probability, the cache capacity and the bandwidth required for a general prefetch scheme. In Section III, based on the discussion of the derived expressions, we propose a new prefetch scheme, which will minimize the average latency or maximize the average hit probability based on sorting a combined parameters that are extracted from log data of the cache. Finally, in Section IV, some numerical results are presented to validate our analysis, which confirms that the prefetch scheme based on the criterion of access probabilities alone cannot achieve the lowest latency or the highest hit probability.

II. PERFORMANCE OF A GENERAL PREFETCH SCHEME

A majority of Web servers and clients use the Hypertext Transfer Protocol (HTTP) [14], which has several cache control features. The basic cache mechanism in HTTP/1.1 uses the origin server-specified expiration times and validators, as described below.

The "expiration" caching mechanism is to expect that origin servers will use the "Expires header" (or the max-age directive of the Cache-Control header) to assign future explicit expiration times to responses. Before the expiration time is reached the document is not likely to change. If the origin servers do not provide explicit expiration times, a HTTP cache can use other header values (such as the Last-Modified time) to estimate a plausible expiration time.

The Last-Modified entity-header field value is often used as a cache "validator". When an origin server generates a full response, it attaches the validator to the response, which is kept with the cache entry. When a cache finds that a cached entry that a client is requesting has already expired, it makes a conditional request that includes the associated validator to the origin server. The origin server responds with a short code "Not_Modified" (no entity-body) to validate that the cached entry is still usable if the entity has not been modified since the Last-Modified time; otherwise, it returns a full response including entity-body. Thus, it avoids transmitting the full response if the validator matches, and avoid an extra round trip if it does not match.

In order to determine whether a cached entry is fresh, a cache needs to know if its *current age* has exceeded its *freshness lifetime*. The *current age* is an estimate of the time elapsed since the response was generated at the origin server. The *freshness lifetime* is the length of time between the generation of a response and its expiration time. HTTP/1.1 requires the origin server to send a *Date* header with every response, giving the time when the response was generated. The expiration judgement is performed in the cache when a cached entry is requested by a client:

$$entry\_is\_fresh = (freshness\_lifetime > current\_age) \qquad (1)$$

If the cached entry is fresh, then the cache sends the entry to the client; otherwise, it sends a conditional request with associated validator to the origin server. The validation check is performed in the origin server:

$$Not\_Modified = (\ Validator = Last\_Modified\ time\ ) \qquad (2)$$

The caching in HTTP/1.1 is shown in Fig. 1, where

Expiration time	is the time at which the origin server intends that an entity should no longer be returned by a cache without further validation.
Current age	is the time since the response was sent by, or successfully validated with, the origin server.
freshness lifetime	is the length of time between the generation of a response and its expiration time.
validator	is a protocol element (e.g., an entity tag or a Last-Modified time) that is used to find out whether a cache entry is an equivalent copy of an entity.
Date	is the value of the origin server's Date: header.
now	is the current (local) time at the host performing the calculation.
request_time	is the (local) time when the cache made the request that resulted in this cached response.
response_time	is the (local) time when the cache received the response.

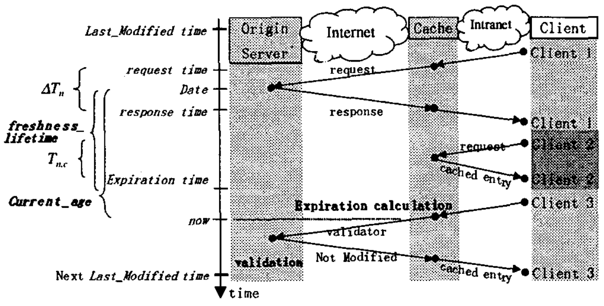


Fig. 1. Caching in HTTP/1.1

Based on the HTTP/1.1 caching mechanism, we build an analysis model of caching, as shown in Fig. 2. Document n is modified in its origin server with cycle μ_n. The first request from a client to the cache in a given cycle can not be satisfied by the cache (i.e., "miss" a fresh copy) and must fetch a copy of the document from the origin server. The consequent requests in the cycle are satisfied by the cache with the cached copy (i.e., "hit" its fresh copy). If a request arrives at the cache between the expiration time and the end of the cycle, the cache must validate the cached copy before using it. The inter-arrival time of requests to document n is governed by a distribution $f_n(t)$. Because origin servers specify the expiration time based on its estimation or schedule to the next modification time (i.e., the end of the current modification cycle), the interval between the expiration time and the end of the cycle may have a stochastic or deterministic distribution. For reduction of access traffic, origin servers intend to reduce the interval.

Now we define the variables required in the following analysis (see TABLE I).

In the definition of the variables, we assume the total rate R of access traffic to the Internet from a given Intranet is finite, given by:

$$R = \sum_{n=1}^{N} R_n\ , \qquad (3)$$

and the ratio

$$\gamma_n = R_n / R \qquad (4)$$

represents the probability of access to document n, n=1,..., N.

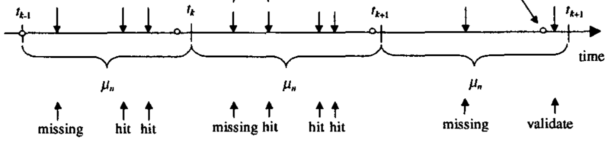
We assume that the inter-arrival time of requests to document n is exponentially distributed:

$$f_n(t) = R_n e^{-R_n t}\ , \qquad n=1, 2, \ldots, N. \qquad (5)$$

Then the probability g_n that there is at least one request to document n during a given modification cycle is given by

$$g_n = 1 - e^{-R_n \mu_n}\ , \qquad n=1, 2, \ldots, N. \qquad (6)$$

Suppose we observe k modification cycles. Then there will be on the average kg_n such cycles in which at least one request is made to document n. The first request in a given cycle will miss a fresh copy of the document and must fetch it from the origin server. The consequent requests in the cycle use the cached copy. Thus, the average hit rate for this document is given by



t_k is the *Last_Modification time*, and ° is the *expiration time*.

Fig. 2. Caching model

TABLE I
DEFINITIONS OF VARIABLES

N	the total number of documents residing in various origin servers in the Internet. n=1,..., N.
μ_n	the average modification cycle of document n, i.e., the interval between two successive modifications.
ΔT_n	the average delay imposed by the Internet, i.e., the interval between the "*request_time*" and the "*response_time*" of the cache, including the transmission time of the document, the round-trip time and the processing time, as shown in Fig. 1. Because persistent connections are the default behavior of any HTTP/1.1 connection, the time of establishing a connection is not included.
$T_{n,r}$	the average delay imposed by the Intranet between the request and the receiving of the response from the cache when a "fresh" or "valid" copy of the document is found in the cache, including the transmission time of the document, the round-trip time from the cache to the client and the processing time in the cache, as shown in Fig. 1.
$T_{n,s}$	the total average delay imposed by the Intranet and the Internet when there is no "fresh" copy of document n in the cache, i.e., $T_{n,s} = \Delta T_n + T_{n,c}$.
s_n	the size of document n.
R_n	the average rate of access to document n.
R	the total rate of access traffic to the Internet from the cache, i.e., the sum of $\{R_n\}$.
γ_n	the access probability, approximated by the ratio R_n /R.
$f_n(t)$	the distribution of inter-arrival time of requests to document n, which is assumed as exponential distribution.
g_n	the probability that there is at least one request to document n during a given modification cycle.
p_n	the average hit rate for document n.
η_n	the average latency increased when there is no "fresh" copy of document n in the cache, i.e., $\eta_n = \gamma_n (1 - p_n) \Delta T_n$
φ_n	the average missing probability of document n, i.e., $\varphi_n = \gamma_n (1 - p_n)$
r	the total number of documents that are prefetched to the cache.
P	the total hit probability.
L	the average latency for a prefetch scheme.
C	the required cache capacity.
B	the total transmission bandwidth required for transmitting documents from the origin servers to the cache.

$$p_n = 1 - \frac{kg_n}{k\mu_n R_n} = 1 - \frac{g_n}{\mu_n R_n} \tag{7}$$

where $k\mu_n R_n$ is the total number of requests made to document n during the k cycles, n=1, 2, ..., N.

Among the hit requests, there are some requests may arrive during the interval between the expiration time and the end of the cycle. They require the cache to validate the cached copy. Since the response with special code "Not_Modified" is a short message, the transmission time is small. For simplicity, we omit this delay or assume this delay is included in the average delay $T_{n,c}$ when a valid copy of the document is found in the cache.

Now we derive general expressions for the average latency and hit probability of a generic prefetch scheme. Due to the limitation of the cache capacity, any prefetch scheme cannot cache all documents. Suppose that r documents are prefetched to the cache. By "prefetch" we mean the action that a proxy Web server takes by automatically caching and updating the r documents once they have expired, and this action is not driven by the client requests. Therefore, the average latency L for a prefetch scheme is given by

$$\begin{aligned} L &= \sum_{n=r+1}^{N} \gamma_n \left[p_n T_{n,c} + (1-p_n) T_{n,s} \right] + \sum_{n=1}^{r} \gamma_n T_{n,c} \\ &= \sum_{n=1}^{N} \gamma_n \left[p_n T_{n,c} + (1-p_n) T_{n,s} \right] - \sum_{n=1}^{r} \eta_n \, , \end{aligned} \tag{8}$$

where γ_n is the access probability given by (4), p_n is the hit rate given by (7), $T_{n,c}$ is the response time from the cache, $T_{n,s}$ is the response time from the origin server, and η_n is defined by

$$\eta_n = \gamma_n (1 - p_n) \Delta T_n \, , \quad n\text{=1,...,}N \tag{9}$$

As is seen from (8), the average latency consists of two terms: the first term is solely determined by the document access rates $\{R_n\}$, modification cycles $\{\mu_n\}$ and response times $\{T_{n,c}, T_{n,s}\}$, and is independent of a specific choice of prefetch scheme, which is the latency of a "no-prefetch" cache scheme (i.e., the conventional caching); the second term $\sum_{n=1}^{r} \eta_n$ is the latency reduction that a prefetch scheme makes. This reduction is determined by the number r and the selection of r prefetch documents and so depends on the specific prefetch scheme.

The total hit probability is given by

$$P = \left(\sum_{n=r+1}^{N} R_n p_n + \sum_{n=1}^{r} R_n \right) \Big/ R = \sum_{n=1}^{N} \gamma_n p_n + \sum_{n=1}^{r} \varphi_n \, , \tag{10}$$

where R_n is the access rate to document n, R is the total access rate, γ_n is the access probability given by (4), and φ_n is defined by

$$\varphi_n = \gamma_n (1 - p_n) \, , \quad n\text{=1, 2, ..., }N. \tag{11}$$

Obviously, $\sum_{n=1}^{r} \varphi_n$ is the increment of the hit probability compared with the "no-prefetch" cache scheme.

Now we proceed to derive expressions for the required cache capacity and bandwidth. Whenever a request misses a "fresh" copy of document n in the cache, the cache must fetch the current version of the document from the origin server. Thus, the fetching rate for document n in the cache is $(1\text{-}p_n)R_n$. The freshness lifetime of the document will be $\mu_n\text{-}t_1$, where t_1 is the origin server's "*Date*" in terms of HTTP/1.1, i.e., the time of the first request since the present modification cycle has started. Thus, the average freshness lifetime of the document is given by

$$\int_0^{\mu_n} (\mu_n - t_1) f_n(t_1) dt_1 \Big/ \int_0^{\mu_n} f_n(t_1) dt_1 = \frac{\mu_n}{g_n} - \frac{1}{R_n} \, , \tag{12}$$

where $f_n(t)$ is given by (5), and g_n is given by (6). The fetched document will be expectedly stored in the cache for the interval of the average freshness lifetime.

In a prefetch scheme, the selected r documents are prefetched into the cache, while the other requested documents are dynamically stored in the cache as is done in the ordinary cache scheme. Therefore, the required cache capacity C is

$$\begin{aligned} C &= \sum_{n=r+1}^{N} s_n R_n (1 - p_n) \left(\frac{\mu_n}{g_n} - \frac{1}{R_n} \right) + \sum_{n=1}^{r} s_n \\ &= \sum_{n=1}^{N} s_n p_n + \sum_{n=1}^{r} s_n (1 - p_n) \, , \end{aligned} \tag{13}$$

where s_n is the size of document n, and (7) is used to derive the last expression. Therefore, the prefetch scheme needs an extra cache capacity of $\sum_{n=1}^{r} s_n(1-p_n)$ compared with the ordinary cache scheme. Thus improvements in the average latency L and the hit probability P are obtained in exchange for the increased cache capacity.

The total transmission rate required for transmitting documents from the origin servers to the cache should be

$$B = \sum_{n=r+1}^{N} R_n(1-p_n)s_n + \sum_{n=1}^{r} s_n/\mu_n$$
$$= \sum_{n=1}^{N} g_n \frac{s_n}{\mu_n} + \sum_{n=1}^{r}(1-g_n)\frac{s_n}{\mu_n}, \qquad (14)$$

where we used (7). Therefore, the prefetch scheme needs an extra transmission bandwidth of $\sum_{n=1}^{r}(1-g_n)\frac{s_n}{\mu_n}$ than the ordinary cache scheme. Thus, the improvement in the average latency and the hit probability is achieved at the expense of increased bandwidth usage.

III. A New Prefetch Scheme

We now proceed to derive a new prefetch scheme. First we discuss the derived general expressions for the average latency, hit probability, average cache capacity and average bandwidth usage of a generic prefetch scheme as follows.

From (8), if we sort $\{\eta_n, n=1, \ldots, N\}$ in the descending order and relabel them as: $\eta_1 \geq \eta_2 \geq \eta_3 \geq \ldots \geq \eta_N$, then we choose the r documents that correspond to the r largest $\eta_1, \ldots, \eta_r$, and prefetch them into the cache. In this case, the average latency L is minimized. Obviously, the reduction in the average latency is determined by three factors: the access probability, the hit rate and the response time. The previous prefetch schemes in the literature cannot achieve the minimum average latency because their criterion is usually the access probability $\{\gamma_n\}$ alone. Among the three kinds of parameters, the average delay ΔT_n imposed by the Internet has significant effect on the average latency. If it is small, that is, the bandwidth that the cache connects to the Internet is fast enough and the size of the document is small, then the prefetch scheme cannot be very effective in reducing the latency.

From (10), if we sort $\{\varphi_n, n=1, \ldots, N\}$ in the descending order and relabel them as: $\varphi_1 \geq \varphi_2 \geq \varphi_3 \ldots \geq \varphi_N$, then we choose the r documents that correspond to the r largest $\varphi_1, \ldots, \varphi_r$, and prefetch them into the cache. In this case, the hit probability P is maximized. Since $\eta_n = \varphi_n \Delta T_n$, the two criteria for sorting are almost equivalent if ΔT_n's are nearly equal. Obviously, a prefetch strategy based on the access probabilities $\{\gamma_n\}$ alone cannot achieve the maximum hit probability.

Equations (13) and (14) show that the average latency and the hit probability are improved at the expenses of increased capacity and increased bandwidth usage. The maximum number r of the documents that can be prefetched must satisfy all of the following constraints:

(a) Increased cache capacity ΔC:

$$\sum_{n=1}^{r} s_n(1-p_n) \leq \Delta C, \qquad (15)$$

where ΔC is the redundant cache capacity that may be utilized by the prefetch caching, which is the difference between the given cache capacity and the part required for the conventional caching. That is, the prefetch scheme can be performed only when the cache capacity has redundancy.

(b) Increased bandwidth usage ΔB:

$$\sum_{n=1}^{r}(1-g_n)\frac{s_n}{\mu_n} \leq \Delta B, \qquad (16)$$

where ΔB is the redundant bandwidth that may be utilized by the prefetching, which is the difference between the possible bandwidth and the bandwidth required for the conventional fetch of documents. That is, the prefetch scheme should not congest the network or increase bandwidth cost.

(c) Tolerable latency L_0:

$$\sum_{n=1}^{r}\eta_n \geq \Delta L, \qquad (17)$$

where L_0 is the lower bound of the average latency, and ΔL is the difference between the higher average latency necessary for conventional fetch of documents and the tolerable latency L_0. That is, the prefetch scheme must be efficient enough to reduce the latency under the limit.

(d) Minimum hit probability required P_0:

$$\sum_{n=1}^{r}\varphi_n \geq \Delta P. \qquad (18)$$

where ΔP is the difference between the required minimum hit probability P_0 and the average hit probability for conventional access to documents. That is , the prefetch scheme must be efficient enough to increase the total hit probability over the limit.

Therefore, we design the new prefetch scheme as follows:

(A) From the log file of the cache that record its access activities and the headers of responses from origin servers, determine the access rates $\{R_n\}$, the total access rate R, the modification cycles $\{\mu_n\}$, the document sizes $\{s_n\}$, and the response times $\{\Delta T_n\}$. For simplicity, the scheme only calculates the parameters associated with the potential candidates that may be included into the list of the prefetched documents. The document that has only one access record (i.e., no revisit) in the log or has not more than one modification cycle during the statistic period will not treated as "potential candidate".

(B) Calculate $\{\gamma_n\}$ from (4); $\{p_n\}$ from (7); $\{\eta_n\}$ from (9); $\{\varphi_n\}$ from (11).

(C) The objective of the system performance is to minimize the average latency L. Sort $\{\eta_n\}$ in the descending order and relabel them as: $\eta_1 \geq \eta_2 \geq \eta_3 \geq \ldots$

(D) Determine the number r of prefetching documents, subject to the constraints (15), (16), (17) and (18) for given ΔC, ΔB, ΔL, and ΔP.

(E) Choose the r documents that correspond to the r largest $\eta_1, \ldots, \eta_r$. Prefetch the r documents from origin severs to the cache as soon as the documents has expired in the cache.

(F) Repeat the procedures from (A) to (E) with a given statistical cycle, such as several hours, several days, or several times the expected modification cycle:

$$\bar{\mu} = \sum_{n=1}^{N} \gamma_n \mu_n \, . \tag{19}$$

We can also use the hit probability as the performance objective. The procedure is the same as minimizing the average latency except that we now sort parameters $\{\varphi_n\}$ instead of $\{\eta_n\}$ in step (C).

Note that the prefetch scheme does not need to know the total number N of documents in origin servers and the accurate value of the parameters $\{\eta_n\}$ and $\{\varphi_n\}$ over extremely long statistic period, because it needs only their relative values to sort the documents that have been accessed. Therefore, it can make statistics over a finite long period and based on a finite log data that records the accessed documents instead of all documents in the Internet. The reasonable length of the statistic period is several times the expected modification cycle of documents given by (19).

IV. Numerical Results

In our numerical analysis, we assume the results from Breslau *et al* [15]. They comprised more than 17M requests traced from six academic, corporate and ISP environments. They found that the distribution of documents requests from a fixed group of clients follows a Zipf-like distribution with a parameter α ranging from 0.64 to 0.83, and showed that the model where the web requests are independent and distributed according to a Zipf-like distribution can yield the asymptotic behaviors that the various observed properties of hit-ratios and temporal locality are indeed inherent to web accesses observed by proxies. Hence, we assume that the access distribution $\{R_n\}$ of documents follows a Zipf-like distribution, and select the distribution parameter α=0.75 in our numerical analysis. The total number of accessed documents recorded by the cache is assumed 500,000. The total access rate R is assumed as 4400 requests per hour. The modification cycles $\{\mu_n\}$ are assumed uniformly ranging from 2 to 720 hours. The document sizes $\{s_n\}$ are assumed uniformly ranging from 0.5 to 25k bytes. The response times $\{T_{n,s}\}$ and $\{T_{n,c}\}$ are related to the document sizes and the transmission rates as well as the round-trip and processing times. Because persistent connections are the default behavior of any HTTP/1.1 connection, the time of establishing a connection is not included here. We assume that the average Intranet transmission rate is 160kbps, and the sum of the round trip and processing time is 10ms, for all documents. The Internet transmission rates for individual documents must be different, depending on the bandwidth of paths from an origin server to the proxy Web server. We assume that the Internet transmission rates for different documents are uniformly ranging from 4 to 40 kbps, and the sum of round-trip and processing times are uniformly ranging from 50 to 600ms.

The numerical results are shown in Fig. 3 - Fig. 6. From Fig. 3 we can see that the average latency of the conventional caching (i.e., r=0) is 2.95 seconds. If we use our prefetch scheme based on the criteria of descending order of $\{\eta_n\}$ of (9) and prefetch 10% of the accessed documents, i.e., r=50,000, the average latency L of our prefetch scheme will be 1.993 seconds, which means 32.45% reduction compared with the conventional caching. It is obvious that the conventional prefetch scheme that prefetches based on the access probability $\{\gamma_n\}$ only cannot achieve the lowest average latency, which can achieve only 16.71% latency reduction.

From Fig. 4 we can see that the average hit ratio of the conventional caching (i.e., r=0) is 0.56. If we use our prefetch scheme based on the criteria of descending order of $\{\varphi_n\}$ of (11) and prefetch r=50,000 documents, the average hit ratio P of our prefetch scheme will increase to 0.65. It is obvious that the ordinary prefetch scheme that prefetches based on the access probability $\{\gamma_n\}$ only cannot achieve the highest average hit ratio, which achieves 0.63.

As the number r increases, the cache capacity for any prefetch scheme increases, as shown in Fig. 6. When r=50,000, the extra cache capacity required are 165MB, 400MB and 576MB, respectively for the ordinary prefetch scheme, the maximum hit probability scheme and the minimum latency scheme. The expenses of increased bandwidth usage are 0.69 KB/s, 3.1KB/s and 2.9KB/s, respectively for the conventional prefetch scheme, the maximum hit probability scheme and the minimum latency scheme when r =50,000, as shown in Fig. 6. Therefore, the expenses of increased cache capacity and bandwidth usage are omissible than the significant improvements in the average latency and the hit ratio.

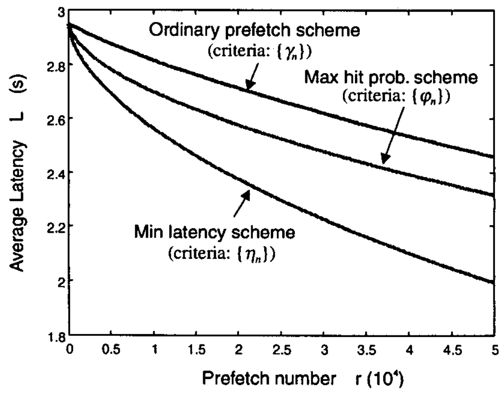


Fig. 3. Average latency

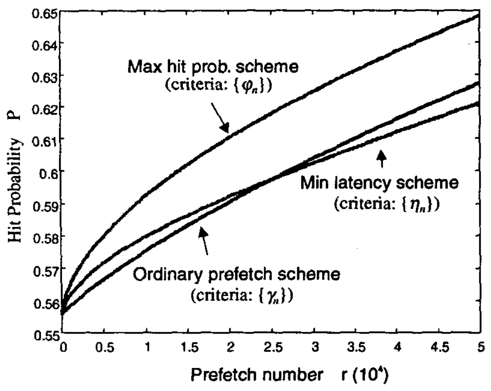


Fig. 4. Hit probability

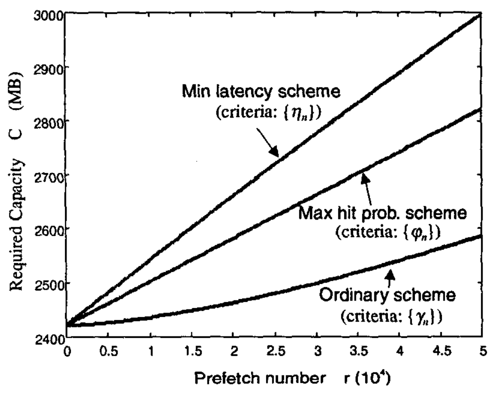


Fig. 5. Required capacity

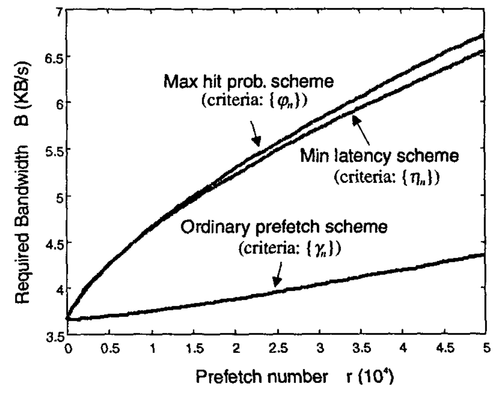


Fig. 6. Required bandwidth

V. CONCLUSIONS

From the analytical expressions and simulation results we obtained for the average latency, hit probability, cache capacity and required bandwidth for a general prefetch scheme, we have confirmed that the conventional scheme that prefetches documents solely based on their access probabilities $\{\gamma_n\}$ is not effective in reducing the average latency or in increasing the hit probability. The optimal criterion should be our newly defined parameters $\{\eta_n\}$ to minimize the average latency, or the parameters $\{\varphi_n\}$ to maximize the hit probability. These parameters to be used to rank order the documents for prefetching are readily obtainable from the log data of the cache. Thus, our prefetch algorithm is simple to implement in practical systems.

REFERENCES

[1] Z. Jiang and L. Kleinrock. "An adaptive network prefetch scheme," *IEEE J. on Selec. Areas in Commun.*, vol. 16, no. 3, April 1998, pp. 358-368.

[2] Z. Jiang and L. Kleinrock. "Web prefetching in a client environment," *IEEE Personal Communications*, vol. 5, no. 5, Oct. 1998, pp. 25-34.

[3] K. Chinen and S. Yamaguchi. "An interactive prefetching proxy server for improvement of WWW latency," in *INET'97*, Kuala Lumpur, Malaysia. [Online]. Available http:// www.isoc.org/ inet97/ proceedings/ Al/ Al_3.HTM

[4] V. N. Persone, V. Grassi, and A. Morlupi, "Modeling and evaluation of prefetching policies for context-aware information services," *Proc. of the Fourth Annual ACM/IEEE Int. Conf. on Client Computing and Networking (MOBICOM 98)*, Dallas Texas, pp. 55-65.

[5] E. P. Markatos, "Main memory caching of Web documents," *Computer Networks and ISDN Systems*, vol. 28, no. 7-11, pages 893-906, 1996. [Online] Available http://www.ics.forth.gr/proj/arch-vlsi/publications.html

[6] A. Bestavros, "WWW traffic reduction and load balancing through server-based caching," *IEEE Concurrency*, January 1997, pp. 56-67.

[7] A. Besavros, "Speculative data dissemination and service to reduce server load, network traffic and service time for distributed information systems," *Proc. ICDE'96: 12th Int. Conf. Data Eng.*, Mar. 1996, pp. 180-187.

[8] V. N. Padmanabhan and J. C. Mogul. "Using predictive prefetching to improve Wold Wide Web latency," *ACM SIGCOMM Comput. Commun. Rev.* July 1996, pp. 22-36.

[9] J. Griffioen and R. Appleton, "Reducing file system latency using a predictive approach," in *Proc. Summer 1994 USENIX Conf.*, June 1994, pp. 197-207.

[10] E. P. Markatos and C. E. Chronaki, "A top-10 approach to prefetching the Web," In *Proceedings of INET' 98* [Online]. Available: http://www.ics.forth.gr/proj/arch-vlsi/publications.html.

[11] R. P. Klemm, "WebCompanion: a friendly client-side Web prefetching agent," *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, no. 4, July-Aug. 1999, pp. 577-94.

[12] F. Li, C. Pei, L. Wei, and J. Quinn, "Web prefetching between low-bandwidth clients and proxies: potential and performance," *Performance Evaluation Review*, vol. 27, no. 1, 1999, pp. 178-187.

[13] G. V. Dias, G. Cope and R. Wijayaratne, "A smart Internet caching system," *INET 96 Conference*. available: http://www.isoc.org/inet96/proceedings/a4/a4_3.htm

[14] R. Fielding et al. "Hypertext transport protocol-HTTP/1.1," *Network Working Group RFC 1945, May 1996*, URL: ftp://ftp.isi.edu/in-notes/rfc2068.txt.

[15] L. Breslau, C. Pei, F. Li, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: evidence and implications," *IEEE INFOCOM*, 1999, vol. 1, 126-134.