Performance Analysis of a Client-Side Caching/Prefetching System for Web Traffic

Abdullah Balamash and Marwan Krunz
Department of Electrical & Computer Engineering
University of Arizona
Tucson, AZ 85721
{balamash,krunz}@ece.arizona.edu

Philippe Nain
INRIA
06902 Sophia Antipolis, France
nain@sophia.inria.fr

Abstract

Network congestion remains one of the main barriers to the continuing success of the Internet. For Web users, congestion manifests itself in unacceptably long response times. One possible remedy to the latency problem is to use caching at the client, at the proxy server, or within the Internet. However, Web documents are becoming increasingly dynamic (i.e., have short lifetimes), which limits the potential benefit of caching. The performance of a Web caching system can be dramatically increased by integrating document prefetching (a.k.a. "proactive caching") into its design. Although prefetching reduces the response time of a requested document, it also increases the network load, as some documents will be unnecessarily prefetched (due to the imprecision in the prediction algorithm). In this study, we analyze the confluence of the two effects through a tractable mathematical model that enables us to establish the conditions under which prefetching reduces the *average* response time of a requested document. The model accommodates both passive client and proxy caching along with prefetching. Our analysis is used to dynamically compute the "optimal" number of documents to prefetch in the subsequent client's idle (think) period. In general, this optimal number is determined through a simple numerical procedure. Closed-form expressions for this optimal number are obtained for special yet important cases. We discuss how our analytical results can be used to optimally adapt the parameters of an actual prefetching system. Simulations are used to validate our analysis and study the interactions among various system parameters.

keywords — Web modeling, caching, proxy, prefetching, multi-fractal traffic.

I. Introduction

A. Motivation and Related Work

Web users can experience response times in the order of several seconds. Such response times are often unacceptable, causing some users to request the delayed documents again. This, in turn, aggravates

This work was supported by the National Science Foundation through grant ANI-0095626. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

the situation and further increases the load and the perceived latency. Caching is considered an effective approach for reducing the response time by storing copies of popular Web documents in a local cache, a proxy server cache close to the end user, or even within the Internet. However, the benefit of caching diminishes as Web documents become more dynamic [21]. A cached document may be stale at the time of its request, given that most Web caching systems in use today are *passive* (i.e., documents are fetched or validated only when requested).

Prefetching (or proactive caching) aims at overcoming the limitations of passive caching by proactively fetching documents in anticipation of subsequent demand requests[1]. Several studies have demonstrated the effectiveness of prefetching in addressing the limitations of passive caching (e.g., [14], [17], [22], [23], [27], [31], [32], [35], [42], [46], [49]). Prefetched documents may include hyperlinked documents that have not been requested yet as well as dynamic objects [37], [42]. Stale cached documents may also be updated through prefetching. In principle, a prefetching scheme requires predicting the documents that are most likely to be accessed in the near future and determining how many documents to prefetch. Most research on Web prefetching focused on the prediction aspect. In many of these studies (e.g., [14], [35]), a *fixed-threshold-based approach* is used, whereby a set of candidate files and their access probabilities are first determined. Among these candidate files, those whose access probabilities exceed a certain prefetching threshold are prefetched. Other prefetching schemes involve prefetching a fixed number of popular documents [32]. Teng et. al [43] proposed the Integration of Web Caching and Prefetching (IWCP) cache replacement policy, which considers both demand requests and prefetched documents for caching based on a normalized profit function. The work in [30] focuses on prefetching pages of query results of search engines. In [47], the authors proposed three prefetching algorithms to be implemented at the proxy server: (1) the *hit-rate-greedy algorithm*, which greedily prefetches files so as to optimize the hit rate; (2) the *bandwidth-greedy algorithm*, which optimizes bandwidth consumption; and (3) the *H/B-greedy algorithm*, which optimizes the ratio between the hit rate and bandwidth consumption. The negative impact of prefetching on the average access time was not considered.

[1]The term *demand request* is used throughput the paper to refer to a user's request for a document that needs to be displayed right away.

Most of the above works rely on prediction algorithms that compute the likelihood of accessing a given file. Such computation can be done by employing Markovian models [20] [35] [41][36]. Other works rely on data mining for prediction of popular documents [38] [48] [29] [34].

Numerous tools and products that support Web prefetching have been developed [1]–[4], [6], [7], [9], [10]. Wcol [3] prefetches embedded hyperlinks and images, with a configurable maximum number of prefetched objects. PeakJet2000 [10] is similar to Wcol with the difference that it prefetches objects only if the client has accessed the object before. NetAccelerator [9] works as PeakJet2000, but does not use a separate cache for prefetching as in PeakJet2000. Google's Web accelerator [4] collects user statistics, and based on these statistics it decides on what links to prefetch. It also can take a prefetching action based on the user's mouse movements. Web browsers based on Mozilla Version 1.2 and higher also support link prefetching [1]. These include Firefox [6], FasterFox [2], and Netscape 7.01+ [7]. In these browsers, Web developers need to include html link tags or html meta-tags that give hints on what to prefetch.

In terms of protocol support for prefetching, Davison et al. [19] proposed a prefetching scheme that uses a connectionless protocol. They assumed that prefetched data are carried by low-priority datagrams that are treated differently at intermediate routers. Although such prioritization is possible in both IPv6 and IPv4, it is not yet widely deployed. Kokku et al. [26] proposed the use of the TCP-Nice congestion control protocol [45] for low-priority transfers to reduce network interference. They used an end-to-end monitor to measure the server's spare capacity. The reported results show that careful prefetching is beneficial, but the scheme seems to be conservative because it uses an additive increase (increase by 1), multiplicative decrease policy to decide on the amount of data to prefetch. Crovella et. al [17] showed that a rate-control strategy for prefetching can help reduce traffic burstiness and queuing delays.

Most previous prefetching designs relied on a *static* approach for determining the documents to prefetch. More specifically, such designs do not consider the state of the network (e.g., traffic load) in deciding how many documents to prefetch. For example, in threshold-based schemes, *all* documents whose access probabilities are greater than the prefetching threshold are prefetched. As shown in this paper, such a strategy may actually increase the average latency of a document.

B. Contributions and Paper Organization

In this paper, we advocate a *dynamic* prefetching approach, in which the prefetching threshold and the number of documents to prefetch are dynamically optimized (on a per idle/active period) so as to minimize the *average* response time for a demand requested document. Our analytical framework accounts for the impact of prefetching on the traffic load, and hence on network delays. It also incorporates the effects of client and proxy caching. The objective function of our optimization ensures that prefetching is performed only when it leads to a reduction in the average response time (compared with no prefetching).

Dynamic threshold-based prefetching was also considered in [24], [44] under a similar setup to the one assumed in this paper, but with only a single level of caching (browser cache). In our work, we also consider proxy caching, which is becoming commonplace in today's Internet access. Furthermore, in [24], [44], it was implicitly assumed that clients have high-bandwidth connections relative to the capacity of the shared access link (C). Consequently, the authors concluded that it is beneficial to prefetch *all* documents whose access probabilities exceed a given, network-state-dependent threshold. In our work, we consider a more generic model than [24], [44], with no assumptions on the predictor or caching policies (in [44], the authors assumed an LRU caching policy). In contrast to [24], [44], our model accommodates various connection speeds, including dialup connections in which the client-proxy link rate can be lower than C. Using this model, we find that it is *not* always good to prefetch all documents with access probabilities greater than some threshold value, irrespective of what this value is. More specifically, there exists an "optimal" number (N_p^*) of documents to prefetch in a given OFF period and for a given client. We provide a simple numerical procedure for determining N_p^* dynamically. For special cases, we express N_p^* is closed-form as a function of various system parameters (access speed, average document size, cache hit rate, etc.). We discuss how to integrate our optimization results into the design of a real prefetching protocol. Extensive simulations of such an optimization-based protocol are conducted. From these simulations, we observe that due to the variability of file sizes, the file hit ratio of the combined prefetching/caching system is not a good measure of the likelihood of finding an arbitrary file in the cache. A better measure is found in the *byte hit ratio*. Contrary to common belief, we observe that prefetching never degrades the

effectiveness of passive caching, so both can beneficially coexist in the same system.

The rest of the paper is organized as follows. In Section II, we present the network access model and derive an expression for the prefetching gain as a function of the system parameters. In Section III, we optimize the prefetching gain and determine the optimal number of prefetched documents that minimizes the average response time of a demand request. We use our analysis to study the effect of caching on the prefetching gain. In Section IV, we discuss how our analytical findings can be integrated into the design of a practical prefetching protocol. Simulations results are reported in Section V, followed by conclusions in Section VI.

II. Modeling Framework

A. System Architecture

As shown in Figure 1, we consider n *homogeneous* Web clients who are connected to a proxy server through dedicated lines (i.e., dial-up modems, cable, DSL, etc.), each of capacity r bits per second[2]. The proxy server is connected to the Internet via an access link of capacity C bps. A client is assumed to run one browsing session at a time. The case of multiple sessions will be treated in a future work. Each client maintains a local cache that implements an arbitrary cache replacement policy. Let h_c be the file hit ratio of the cache. A very small portion of the client cache is reserved for prefetching, and is called the *prefetching cache*. The remaining portion is called the *regular cache*. It was reported in several studies (e.g., [11], [15], [16], [18]) that the hit ratio is proportional to the logarithm of the cache size. Hence, reserving a small portion of the cache for prefetching should have a negligible effect on the hit ratio of the regular cache, making this hit ratio almost independent of prefetching. The regular cache stores demand-requested documents, whereas the prefetching cache stores prefetched documents. When a document that happens to be in the prefetching cache is demand-requested, it is moved to the regular cache. Accordingly, a document cannot be in both caches at the same time. Prefetched documents are brought to the client from either the proxy server (if available) or are retrieved from the original Web server. The proxy server maintains a cache for demand-requested documents, which is parameterized by

[2]In Appendix A, we show how our model can be extended to clients with heterogeneous characteristics.

its hit ratio h_{proxy}. We assume that h_{proxy} is independent of prefetching (the proxy server does not cache any prefetched files). We verify this point later in the simulations. Each client alternates between active (ON) periods, during which the client demand-requests documents, and idle (OFF) periods, during which the retrieved information is read by the user (see Figure 2). An ON period starts with the retrieval of an html file (the *main document*), which is usually followed by the retrieval of its inline objects.

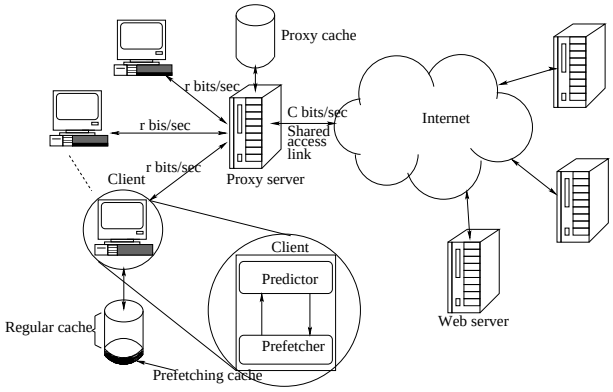


Fig. 1. Components of the prefetching system.

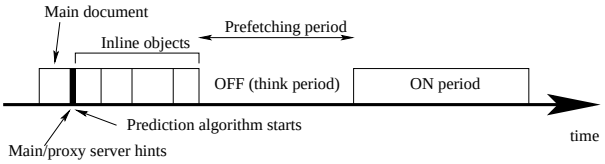


Fig. 2. Client behavior.

Each client runs a prediction algorithm that predicts future requests using the history of the client's requests along with hints from the proxy and original servers. The incorporation of such hints in the HTTP protocol is often done through the addition of new headers. These headers can have several directives that can be used by servers and clients to agree on the level of cooperation and to exchange information [22]. For example, the *HTTP link header*, specified in RFC 2068 [5], provides the means for describing a relationship between two resources (e.g., the requested file and other files). Other techniques for exchanging information include *prefetching agents*, which communicate with servers via separate HTTP requests for files that contain statistical information [32].

Typically, the outcome of the prediction algorithm becomes available right after the receipt of the main document. We assume a generic prediction model, where the predictor computes a set of k candidate files

$D_1, D_2, \ldots, D_k$, and the probabilities of accessing them in the next user's active period ($P_1, P_2, \ldots, P_k$). For example, one can adopt the scheme in [24] with a straightforward modification to account for hints from the proxy server (the details of such a modification are described in Section IV-B). Note that the events of requesting any two or more files in an ON period are not necessarily mutually exclusive, i.e., $\sum_{i=1}^{k} P_i$ can be greater than one. The prefetcher uses the information provided by the predictor to prefetch files in the subsequent OFF period of the underlying client, starting with the file that has the highest access probability. The number of prefetched files depends on the length of the OFF period and the state of the network. If the OFF period is long enough, prefetching ends before the start of the next ON period. Otherwise, if a demand-request is issued before the prefetching of a file has completed, the client instructs the proxy to stop forwarding the prefetched file in progress. Any partially prefetched file is kept in the prefetching cache to be used in any future access to such a file. A demand-request is first served from the local cache (regular or prefetching cache), if the file is available. Otherwise, the request is forwarded to the proxy server. If the proxy server does not have the requested file in its cache, it retrieves it from the original server.

B. Prefetching Gain

In this section, we study the benefit of client-side prefetching when the average access delay is used as the performance metric of interest. The improvement in the access delay is indicated by the ratio of the average access time of an arbitrary demand-requested file under prefetching (A_p) to the average access time of such a file without prefetching (A_{np}). We call this ratio *the access improvement index* (I). Prefetching is advantageous when $I < 1$. In the absence of client caching and prefetching, the proxy server is assumed to retrieve files from the original servers at a rate λ files per second in response to requests from all clients. Note that caching and prefetching can impact the rate of bringing files from their respective servers.

Prefetching always increases the hit ratio of the overall client cache system because prefetched files do not replace files in the regular cache (they are stored in the prefetching cache). Suppose that, on average,

a client prefetches N_p files in a given OFF period. Then, the average number of "useful" files is:

$$m = N_p \overline{P} \tag{1}$$

where $\overline{P} \stackrel{\text{def}}{=} \frac{\sum_{i=1}^{N_p} P_i}{N_p}$ $(0 \leq \bar{P} \leq 1)$ is called the *prefetching precision* [39]. The increase in the client-cache hit ratio due to prefetching is given by:

$$\Delta_h = \frac{m}{N_{on}} \tag{2}$$

where N_{on} is the average number of files in an ON period. This says that for each demand-requested file, there are $\frac{m}{N_{on}}$ useful prefetched ones.

If a client does not employ prefetching, a requested file will be brought from the local cache, the proxy cache, or the original server. The corresponding access times for a file of an average size $\overline{s}$ are 0, $t_{prox}(\overline{s})$, and $t_{serv}(\overline{s})$, respectively. Accordingly, the average access time without prefetching is

$$A_{np} = (1 - h_c).\left\{h_{proxy} t_{prox}(\overline{s}) + (1 - h_{proxy}) t_{serv}(\overline{s})\right\}. \tag{3}$$

We will come back to the determination of $t_{prox}(\overline{s})$ and $t_{serv}(\overline{s})$. Consider now the situation under prefetching. Because prefetching is performed on a per-client basis during the OFF periods and because clients communicate with the proxy via dedicated links, $t_{prox}(\overline{s})$ will be the same as in the no-prefetching case. Let $t'_{serv}(\overline{s})$ be the average access time from the original server when prefetching is employed. Note that $t'_{serv}(\overline{s}) \neq t_{serv}(\overline{s})$ because prefetching files for a given client increases the traffic seen by other clients that share the same access link, which as a result affects the average access delay for all clients. Accordingly,

$$A_p = (1 - h_c - \Delta_h).(h_{proxy} t_{prox}(\overline{s}) + (1 - h_{proxy}) t'_{serv}(\overline{s})). \tag{4}$$

From (3) and (4), the access improvement index becomes:

$$I = \frac{(1 - h_c - \Delta_h).(h_{proxy} t_{prox}(\overline{s}) + (1 - h_{proxy}) t'_{serv}(\overline{s}))}{(1 - h_c).(h_{proxy} t_{prox}(\overline{s}) + (1 - h_{proxy}) t_{serv}(\overline{s}))}. \tag{5}$$

We now turn our attention to the computation of $t_{prox}(\overline{s})$, $t_{serv}(\overline{s})$, and $t'_{serv}(\overline{s})$. The queuing delay

at the (dedicated) proxy-client link can be safely ignored, so $t_{prox}(\overline{s}) = \frac{\overline{s}}{r}$. For $t_{serv}(\overline{s})$ and $t'_{serv}(\overline{s})$, we assume they are dominated by the queueing/service delays at the (downlink) shared access link from the Internet to the proxy server. This assumption is justified when the pool of clients that share the access link is large, as is often the case in ISP networks. To compute $t_{serv}(\overline{s})$ and $t'_{serv}(\overline{s})$, we model the queueing/service delays at the proxy as an M/G/R Processor Sharing (M/G/R-PS) system. Riedl et al. [40] suggested the use of this model for the dimensioning of IP access networks with elastic traffic and concluded its suitability for Web delivery systems, particularly when file sizes are large. The rationale behind employing the M/G/R-PS approximation is that in the underlying Web delivery system, multiple file downloads occur simultaneously over different connections (clients). These downloads are serviced by a shared link (processor) of capacity C. In our case, a client is limited by the bandwidth r of the *dedicated* access link, which can be less than C. The shared link behaves approximately as a queuing system with $R = C/r$ servers. If there are n customers in the system, then each customer gets a fraction of the capacity C that depends on n. If $n \leq R$, then each customer gets a fixed fraction r/C, i.e., up to R flows can be served simultaneously, each at a rate r bps. If $n > R$, then each customer gets a fraction $1/n$ of the total capacity. A special case of the M/G/R-PS system is when $R = 1$. In this case, a single client can fully utilize the capacity of the shared access link.

For the M/G/R-PS system, the mean file transfer time is given by [40]:

$$\overline{t} = \frac{\overline{s}}{r} f_R(\rho) \tag{6}$$

where $\rho \stackrel{\text{def}}{=} \lambda \overline{s}/C$ is the traffic load over the access link and f_R is called the *delay factor*, a measure of how link congestion affects the response time. It is given by

$$f_R(\rho) \stackrel{\text{def}}{=} 1 + \frac{E_2(R,\rho)}{R(1-\rho)} \tag{7}$$

$$E_2(R,\rho) \stackrel{\text{def}}{=} \frac{\frac{(R\rho)^R}{R!}\frac{1}{1-\rho}}{\sum_{i=0}^{R-1}\frac{(R\rho)^i}{i!} + \frac{(R\rho)^R}{R!}\frac{1}{1-\rho}}. \tag{8}$$

Equation (8) is the familiar Erlang-C formula.

To apply the above model, we need to compute the traffic load with and without prefetching (ρ_p and

ρ_{np}, respectively). The average load in the case of no prefetching (with caching only) is given by:

$$\rho_{np} = \frac{(1 - h_{proxy})(1 - h_c)\lambda\overline{s}}{C}. \tag{9}$$

This represents the downlink traffic in response to client requests that cannot be satisfied from the client's regular cache or the proxy cache.

When prefetching is implemented, an average of N_p files are retrieved during the OFF period. Hence,

$$\rho_p = \frac{(1 - h_{proxy})(1 - h_c - \Delta_h + N_p/N_{on})\lambda\overline{s}}{C}. \tag{10}$$

This is the load on the downlink in response to requests that cannot be satisfied from the regular, the prefetching, or the proxy caches, plus the extra prefetched traffic ($\frac{N_p\lambda\overline{s}}{N_{on}}$). Note that for each demand-requested file, there are on average N_p/N_{on} prefetched ones.

From (5) and (6), the improvement index reduces to:

$$I = \frac{(1 - h_c - \Delta_h).(h_{proxy} + (1 - h_{proxy})f_R(\rho_p))}{(1 - h_c).(h_{proxy} + (1 - h_{proxy})f_R(\rho_{np}))}. \tag{11}$$

III. Optimization of Prefetching Gain

In this section, we study the performance of a generic prefetching system. We use the analysis in Section II to optimize the prefetching parameters. Intuitively, prefetching impacts the mean access time of a demand-requested file in two ways. It improves the overall cache hit ratio of the given client and, as a result, reduces the number of files that need to be retrieved from the remote servers. At the same time, prefetching increases the load on the shared access link, which affects the retrieval time of demand-requested files destined to other clients (such files are retrieved from their original servers following a miss at the local and proxy caches). Hence, a client should be careful not to prefetch every file suggested by the predictor, as this may lead to increasing the overall average access time.

Accordingly, we seek to compute the optimal number of files to prefetch in an OFF period. Before trying to find this optimal value, we need to study the behavior of I as a function of N_p. It can be mathematically shown (see below) that if prefetching a single file or a fraction of a file does not lead

to any gain, then prefetching more files can only worsen the performance. On the other hand, if there is a gain out of prefetching a single file or a fraction of a file, then there is a unique optimal value for the average number of prefetched files in an OFF period. The following theorem describes the general relationship between I and N_p. It also specifies the condition under which prefetching is beneficial.

Theorem 3.1: Suppose that files are prefetched in a decreasing order of their access probabilities, starting from the most likely one. Then the following holds:

1) Let $\bar{P}(N_p = 1)$ be the prefetching precision when, on average, only one document is prefetched in an OFF period. In other words, $\bar{P}(N_p = 1)$ is the average access probability of the first file to prefetch in the list of candidate files. For prefetching to be of a value, the following condition must be satisfied:

$$\bar{P}(N_p = 1) > P_{th} \stackrel{\text{def}}{=} \frac{M\rho_{np}^R}{(1 + M\rho_{np}^R)} \tag{12}$$

where

$$M \stackrel{\text{def}}{=} \frac{(1 - h_{proxy})R}{(1 - \rho_{np}^R)(1 - h_{proxy}\rho_{np}^R)} \tag{13}$$

2) If prefetching a single file or a fraction of a file does not improve the mean file access time, then increasing the number of prefetched files does not do any better.
3) If there is a gain out of prefetching, then the function I is convex in N_p, with its minimum point achieved at the optimal N_p (denoted by N_p^*).

Proof: See Appendix B. ■

It is clear from Theorem 3.1 that the prefetching protocol must first decide whether to prefetch or not based on the prefetching threshold P_{th}. If prefetching is deemed beneficial, then N_p^* is computed by solving $\frac{dI}{dN_p} = 0$ for N_p. The convexity of the function I (part 2 of Theorem 3.1) gives a simple numerical recipe for computing N_p^* using a logarithmic-time binary search over the set of possible (integer) values that N_p can take. For some special cases, a closed-form expression for N_p^* can be obtained, as described next.

A. Prefetching Precision Independent of N_p

Consider the special case when $\bar{P}$ is independent of N_p, i.e., all files have the same access probabilities. Accordingly, the condition in (12) translates into having the file access probability greater than P_{th}. In this case, N_p^* can be computed analytically for two special cases, as described in the following theorem.

Theorem 3.2: Consider the case when $\bar{P}$ is independent of N_p, $\bar{P} > P_{th}$, and $R = 1$. Then,

1)
$$N_p^* = \begin{cases} \frac{-B-\sqrt{B^2-4AC}}{2A}, & \text{if } B^2 - 4AC > 0 \\ \text{Largest value for } N_p \text{ s.t. } \rho_p < 1, & \text{otherwise} \end{cases} \tag{14}$$

where

$$A \stackrel{\text{def}}{=} \left(\frac{\rho_{np}(1-\bar{P})}{N_{on}(1-h_c)} \right)^2 \tag{15}$$

$$B \stackrel{\text{def}}{=} \frac{-2\rho_{np}(1-\bar{P})(1-\rho_{np})}{(1-h_c)N_{on}} \tag{16}$$

$$C \stackrel{\text{def}}{=} (1-\rho_{np})^2 - \frac{1-h_{proxy}}{h_{proxy}} \left(\frac{\rho_{np}}{\bar{P}} - 1 \right) \tag{17}$$

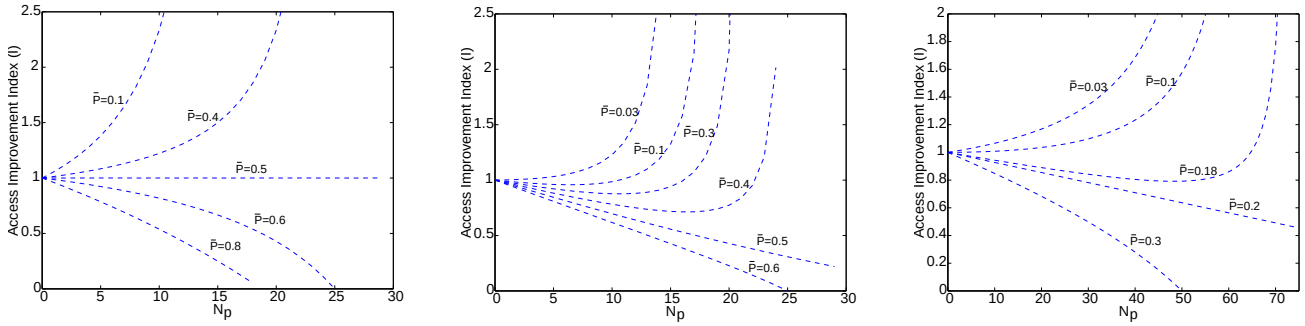
2) If no proxy caching is used ($h_{proxy} = 0$), then the higher the number of prefetched files, the higher is the prefetching gain.

Proof: See Appendix C. ■

Figure 3 depicts I versus N_p for the special case when $\bar{P}$ is constant (independent of N_p), with $h_c = 0$ (no regular client cache), $C = 500$ kbps, $\lambda = 6.25$ files/s, $\bar{s} = 40$ kbits, and $N_{on} = 15$ files. In part (a), we set $r = 500$ kbps (so $R = 1$), $h_{proxy} = 0$, and $P_{th} = 0.5$. It is clear that when $\bar{P} > P_{th}$, I decreases monotonically with the number of prefetched documents. In this case, I can be maximized by prefetching all documents with access probabilities greater than P_{th}, in line with [44]. For the other cases, I does not necessarily decrease monotonically with the number of prefetched files. This is shown in parts (b) and (c). For example, in part (b), when $\overline{P} = 0.4$, I decreases with the increase in N_p up to a certain point, after which the trend is reversed. Furthermore, when $\overline{P} \gg P_{th}$, the trend in the access improvement becomes monotone. This is because the improvement in the hit ratio is more significant than the loss due to the increased traffic. Moreover, prefetching cannot go beyond a point where the shared access link is 100%

loaded.

Note that the threshold value decreases with the increase in R and h_{proxy}, which is intuitive since increasing R or h_{proxy} moves the delay bottleneck towards the client-proxy link.



(a) $R = 1$, $h_{proxy} = 0$, $P_{th} = 0.5$

(b) $R = 10$, $h_{proxy} = 0$, $P_{th} = 0.05$

(c) $R = 1$, $h_{proxy} = 0.6$, $P_{th} = 0.102$

Fig. 3. I versus N_p ($C = 500$ kbps, $\lambda = 6.25$ files/s, $\bar{s} = 40$ kbits, $N_{on} = 15$ files, $h_c = 0$).

B. Prefetching Precision Varying with N_p

To study the performance of prefetching when $\bar{P}$ depends on N_p, we consider the following simple relationship between Δ_h and N_p:

$$\Delta_h = K\ (1 - e^{-a\ N_p}) \tag{18}$$

where $0 \leq K \leq 1 - h_c$. Based on this relationship, the lowest value for Δ_h is 0 (no prefetching), while its highest value is $1 - h_c$, since the overall cache hit ratio ($h_c + \Delta_h$) cannot exceed one. Accordingly, the prefetching precision is given by:

$$\bar{P} = \frac{H(1 - e^{-a\ N_p})}{N_p} \tag{19}$$

where $H \stackrel{\text{def}}{=} N_{on}\ K$. Because $\bar{P} \leq 1$, the constant a is bounded ($0 \leq a \leq -\ln(1 - \frac{1}{H})$).

Figure 4 shows the performance for the same system shown in Figure 3 but with $\overline{P}$ varying according to (19). Consider the case $R = 1$ and $h_{proxy} = 0$. In this case, $\bar{P} > P_{th}$ for $N_p \leq 7$. When $N_p = 7$, prefetching all seven files with access probabilities greater than P_{th} improves the performance ($I < 1$), but does not necessarily optimize it (e.g., prefetching six files is actually more beneficial than prefetching

seven files). For the other two cases shown in Figure 4, we can see that increasing the number of prefetched files can worsen the performance, sometimes even when $\bar{P} > P_{th}$.

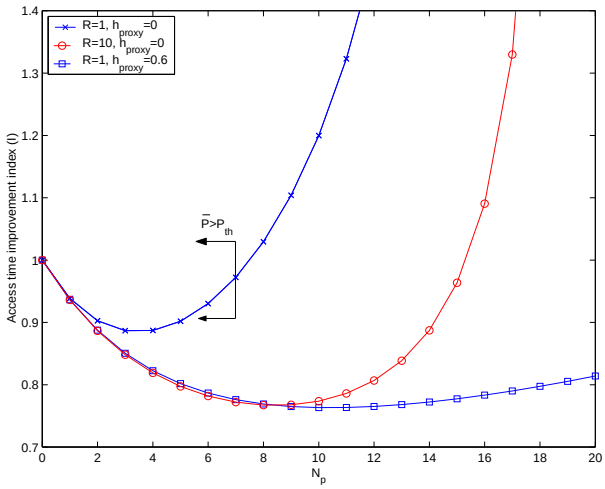


Fig. 4. I versus N_p for the case $\bar{P} = \frac{4.5(1-e^{-0.24N_p})}{N_p}$ ($C = 500$ kbps, $\lambda = 6.25$ files/s, $\bar{s} = 40$ kbits, $N_{on} = 15$ files, $P_{th} = 0.5$, $h_c = 0$).

Corollary 3.3: Consider the case when $R = 1$, $h_{proxy} = 0$, and $\bar{P}$ varies with N_p. Then, prefetching all files with access probabilities greater than P_{th} is guaranteed to reduce the average access time compared to no prefetching.

Proof: The proof follows readily from part 2 of Theorem 3.2. ■

Theorem 3.4: For given $\bar{P}$ and N_p, increasing R or h_{proxy} reduces the average access time.

Proof: See Appendix D. ■

According to Corollary 3.3 and Theorem 3.4, a prefetching system can prefetch all files with access probabilities greater than P_{th}. As can be seen in Figure 5, this solution reduces the average access time but does not necessarily minimize it with respect to N_p. This is because for a given N_p, the worst access delay is when $R = 1$ and $h_{proxy} = 0$.

Effect of Caching on Prefetching Gain

We now use our analytical results to study the interactions between prefetching, on the one hand, and proxy and client caching, on the other. First, we consider the interactions between prefetching and proxy caching, setting $h_c = 0$. Intuitively, one may think that proxy caching limits the value of prefetching. It turns out that this is not always true. Specifically, for clients with low-bandwidth connections ($r \ll C$),

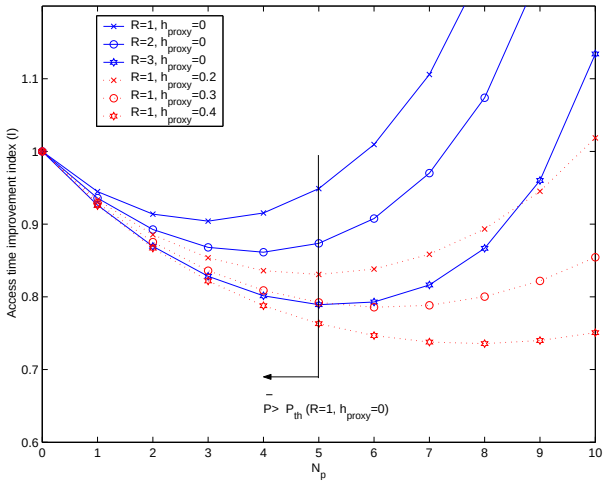
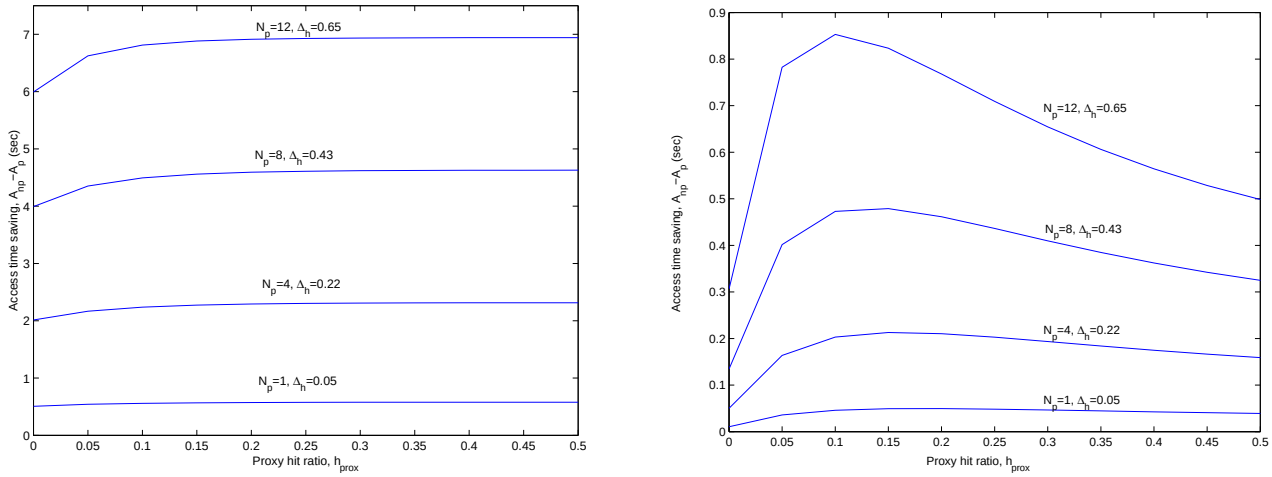


Fig. 5. Effects of R and h_{proxy} on I for the case $\bar{P} = \frac{4.5(1-e^{-0.18N_p})}{N_p}$ ($C = 500$ kbps, $\lambda = 6.25$ files/s, $\bar{s} = 40$ kbits, $N_{on} = 9$ files, $P_{th}(R = 1, h_{proxy} = 0) = 0.5$, $h_c = 0$).

the bottleneck is the client-proxy link. In this case, the reduction in the access time due to prefetching a file from the proxy cache is comparable with the reduction due to prefetching this file from its original server, especially when the load over the shared access link is light (i.e., small N_p values). This situation is depicted in Figure 6-a, where $A_{np} - A_p$ is observed to be insensitive to h_{proxy}. Note that as the load over the shared link increases (by increasing N_p), then $A_{np} - A_p$ starts to become more sensitive to h_{proxy}.

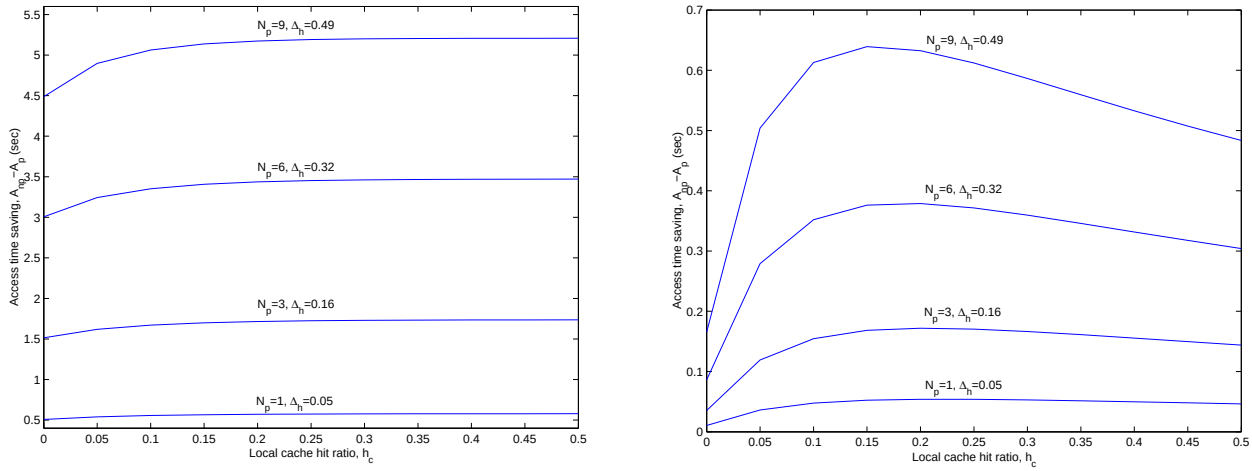
The behavior is different when r is large (i.e., high-bandwidth user access), as the bottleneck shifts to the shared access link between the proxy and the Internet. This is shown in part (b) of the figure with $R = C/r = 1$. At light load, not much change in $A_{np} - A_p$ is observed as h_{proxy} increases. However, at heavy load (e.g., $N_p = 12$), the time saving $A_{np} - A_p$ initially increases with h_{proxy} (i.e., the prefetching gain improves with better proxy caching), up to an optimal point $h^*_{proxy} \approx 0.1$, after which the trend is reversed. *This says that while proxy caching always contributes positively to the prefetching gain ($A_{np} - A_p$ under $h_{proxy} = 0$ is always smaller than $A_{np} - A_p$ under $h_{proxy} > 0$), the amount of positive contribution due to proxy caching is not always monotone with h_{proxy}.*

Similar interactions between prefetching and client caching are observed, as shown in Figure 7. The local cache limits the number of prefetched files, which in turn limits the prefetching gain. But it also reduces the load, which is advantageous for prefetching especially for clients with high-bandwidth connections ($R = 1$) and for a heavily loaded system.



(a) $r = 56$ kbps (b) $r = 1000$ kbps

Fig. 6. Impact of proxy caching on the effectiveness of prefetching ($C = 1000$ kbps, $\lambda = 20$ files/s, $\bar{s} = 40$ kbits, $N_{on} = 15$ files, $h_c = 0$).



(a) $r = 56$ kbps (b) $r = 1000$ kbps

Fig. 7. Impact of local (regular) caching on prefetching ($C = 1000$ kbps, $\lambda = 20$ files/s, $\bar{s} = 40$ kbits, $N_{on} = 15$ files, $h_{proxy} = 0$).

IV. Practical Considerations

In this section, we discuss how our analytical model can be integrated into the design of a prefetching protocol. We first address the issue of estimating the model's parameters and show how such estimates can be used in performing optimal prefetching.

A. Parameter Estimation and Protocol Support

Initially, each client goes through a no-prefetching warm-up period, during which the client estimates its own parameters, including h_c and N_{on} (the number of demand-requested files in an ON period). The client also estimates the relationship between $\bar{P}$ and N_p. This can be done by running a prediction algorithm

without performing any actual prefetching. Each client reports this information to the proxy server, which uses it in estimating P_{th} according to (12), determining the prefetching gain I, and computing N_p^* for each client. The proxy also estimates its own cache hit ratio h_{proxy}. By the end of the warm-up period, the proxy will have computed for each client an approximation of h_c, the relationship between $\bar{P}$ and N_p, and the average length of the ON period. Clients periodically update the proxy with estimates of their parameter values, which the proxy uses along with the estimated load (ρ_p) to recompute the prefetching parameters (P_{th} and N_p).

If the proxy determines that prefetching is beneficial (based on P_{th} and $\bar{P}$), it uses (11) to optimize the number of files each client can prefetch. The proxy provides each client with its N_p^* by piggybacking this information in its response to the client. Once a client has its N_p^*, it can start prefetching in the subsequent OFF period. We assume that N_p^* can take non-integer values, where the fractional part means that only a part of a file is prefetched using, for example, the HTTP *range request* message [8]. This feature is critical because of the high variability of file sizes in the Web. Upon receiving a demand-request, prefetching stops, and all prefetched data are saved. When a file that was partially prefetched is demand-requested, only the remaining portion of this file is retrieved.

Prefetching needs to be implemented fairly for clients with different traffic demands. A reasonable approach is to assign weights to clients depending on their (downlink) traffic demands. The higher the weight assigned to a client, the higher the volume of prefetched traffic that is allowed for that client. The assigned weights can be easily computed by the proxy based on the observed loads of different clients at the shared link.

B. Forecasting Demand-Requests

Several schemes for Web traffic prediction have been proposed in the literature (e.g., [14], [23], [24], [32], [35], [42], [49]). Any of these schemes can be integrated into our prefetching protocol. Without loss of generality, we can consider for our simulations the predictor by Jiang et. al [24], with some modifications to include hints from the proxy server. In [24], prediction is done at the client side using the client's history along with hints from the main server. Two types of counters are maintained at the

client for each html document: a page counter and a set of link counters. Any time an html document X is accessed, its page counter P_X is incremented. If X has a link to an html document Y and Y is accessed from X, then the link counter $L_{X,Y}$ is incremented. Following each access to document X, the predictor computes the probability of accessing every document that is linked from X. For a linked document Y, this probability is given by $\frac{L_{X,Y}}{P_X}$. If not enough historical information is available for computing this probability, the client relies on hints from the proxy, which runs a similar prediction algorithm but based on the aggregate traffic seen by all clients. The proxy also maintains some hints from the original servers that can be used if the information collected by the proxy is not statistically sufficient. The prediction algorithm at the proxy requires that clients provide the proxy with information about the html document from which the request is initiated. The proxy also provides the server with similar information.

Note that the above predictor does not consider dynamically generated files, i.e., files that are generated by a server-based script whose input parameters are supplied by the client. This does not change the qualitative nature of our results, since our analysis relies on a generic output for the prediction algorithm (the k files that are most likely to be demand requested in the next ON period along with their access probabilities). Predictors for dynamic content have been proposed in the literature (e.g., [28], [37]), and can be readily integrated into our adaptive prefetching design.

V. Simulation Results

The theoretical results in Section III were based on average-case analysis of an idealized queuing model. To validate the appropriateness of these results, we simulate a generic prefetching protocol that integrates into its design the optimization of the previous section.

A. Simulation Setup

We consider 50 clients who access the Web through a common proxy. The proxy cache implements an LRU policy with $h_{proxy} = 0.4$ (the cache hit ratio is controlled by adjusting the cache size). Each client has a large local cache. One percent of this cache is reserved for prefetching. Local caches also implement the LRU caching policy.

B. Traffic Model

We use model-based synthetic traces to drive our simulations. Although real (measured) traces would be preferable, we do not rely on them for two reasons. First, most Web traces available in the public domain are captured at the server, whereas our simulations require client-side traces. The few available client-side traces are not sufficient to reproduce the behavior of 50 *independent* clients, especially that they do not contain information about the client's ON/OFF behavior[3]. Secondly, when using real traces it is not possible to control the traffic parameters (e.g., average durations for the ON and OFF periods, average document size, etc.), which we need to study different scenarios.

To capture the essential statistical properties of Web traffic, we extend the model in [12] to generate client-side traffic. The model in [12] is based on multifractal processes, which are more flexible than monofractal (self-similar) models in describing traffic "irregularities" at different time scales. In its original form, the model in [12] captures the temporal locality, spatial locality, and popularity of the aggregate Web traffic seen by the proxy server. Such traffic represents responses to requests for main html documents from all clients. Each html document can have one or more inline files (e.g., images). As suggested in [13], [33], a heavy-tailed Pareto distribution is used for the number of inline objects in an html document. The OFF period and the file size are generated according to heavy-tailed lognormal distributions [13], [18], [33]. The duration of the ON period is specified by the requested main document and the time it takes the client to retrieve such a document and its inline files. Table I summarizes the distributions used in traffic generation along with their parameter values (taken from [13], [33]).

The model in [12] was not intended for client-side traffic, but rather to capture the properties of the aggregate traffic destined to a group of clients. To synthesize client-side traffic, we start with a no-prefetching simulation run, in which each client is represented by an ON/OFF profile based on the distributions shown in Table I. The aggregate stream is arranged as a vector. When a client starts a new ON period, it selects a document from the top of that vector. This document is considered as the main html document in the current ON period. Each unique document in the vector is assigned a group of

[3]It is possible to extract multiple sequences from one real trace by arranging this trace as a circular chain and extracting several sequences by randomizing their starting times within the circular chain. However, this method results in correlated sequences.

Component	Distribution	$f(x)$	Parameters
OFF period	Lognormal	$\frac{1}{x\sqrt{2\pi\sigma^2}}e^{\frac{-(\ln(x)-\mu)^2}{2\sigma^2}}$	$\sigma = 1.57$ $\mu = 2.75$
File size	Lognormal	$\frac{1}{x\sqrt{2\pi\sigma^2}}e^{\frac{-(\ln(x)-\mu)^2}{2\sigma^2}}$	$\sigma = 1.82$ $\mu = 6.78$
Files per Web page	Pareto	$f(x) = ak^a x^{-(a+1)}$	$k = 1$ $a = 1.42$

TABLE I

PROBABILITY DISTRIBUTIONS USED IN THE SIMULATIONS.

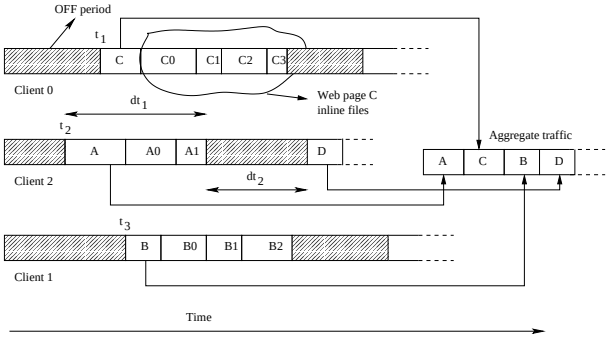


Fig. 8. Client-side traffic generation process.

unique inline files. Moreover, each file (main document or inline file) is assigned a size that is sampled from the proper distribution. The client retrieves the main document with its inline files from the local cache, proxy cache, or from the original server if the document has not been cached before. The outcome of this simulation run is streams of client requests that are saved in several files to be used in the main simulation experiments. Figure 8 depicts an example with three clients. The three clients start their first ON periods at times t_1, t_2, and t_3, respectively, where $t_3 > t_1 > t_2$. According to these times, Client 2 selects the top document (A) in the vector of aggregate requests. The first and the third clients select documents C and B, respectively. It takes the second client a period of dt_1 seconds to retrieve document A and its preassigned inline files (A_0 and A_1), and it takes it a period of dt_2 seconds to read the retrieved information (OFF period). At the end of the OFF period, this client starts a new ON period, while the other clients are still in their OFF periods. Hence, Client 2 selects document D as its next main document, and so on.

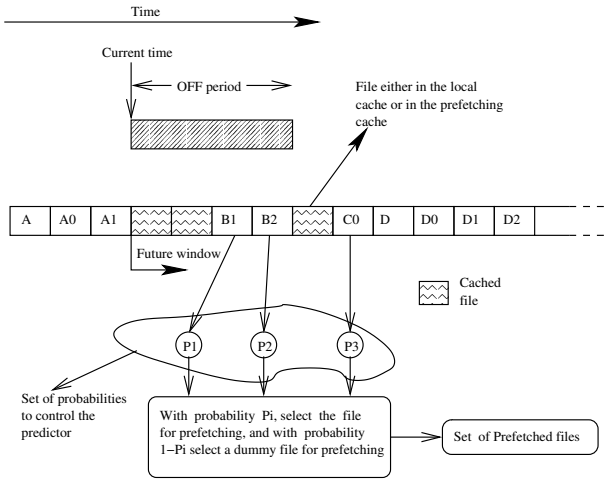


Fig. 9. Mechanism for traffic prediction.

C. Prediction Model

Because prediction is not the focus of our work, for our model-validation purposes, we adopt an artificial predictor whose accuracy can be controlled. The predictor works as follows. Each client is assumed to know the future with certain accuracy and has a window through which it sees this future. To emulate a particular relationship between $\bar{P}$ and N_p, the client considers a window of m files (number of files to prefetch) that are not in the local cache. The ith file in the candidate list is considered for prefetching with probability P_i. If a file is not selected for prefetching, it means that the predictor made a wrong decision. In this case, the client retrieves a dummy file whose size is sampled from the file size distribution. This dummy file is either retrieved from the proxy or the original server based on the estimated value of h_{proxy}. Figure 9 illustrates the main idea behind this artificial predictor. In this figure, the client needs to prefetch three files in the current OFF period. The first three files that are in the future window and are not locally cached are B_1, B_2, and C_0. To capture a specific relationship between $\bar{P}$ and N_p, the access probabilities P_1, P_2, and P_3 for the three candidate files to be prefetched are set to $P_i = i\bar{P}(i) - \sum_{j=1}^{j=i-1} P_j,\ i = 1, 2, 3$. The client prefetches file B_1 with probability P_1, and with probability $1 - P_1$ an alternative dummy file is prefetched. The same thing is done for files B_2 and C_0. Accordingly, the precision in predicting the three files is $\frac{\sum_{i=1}^{3} P_i}{3}$, which reflects the mimicked $\bar{P}(N_p)$.

D. Validation of Δ_h and ρ_p

In this section, we validate our analysis with regard to the effects of prefetching on the client's overall cache hit rate and on the system load (ρ_p). In a given simulation run, each client tries to prefetch a fixed number of files (n) in every OFF period, if possible. Each run outputs the access improvement index (I), the average hit ratios for all caches, the average system load, and the average number of prefetched documents in an OFF period (N_p). Note that N_p can be less than n because some OFF periods are not long enough to retrieve all n files. Figure 10 compares the increase in the client cache hit ratio due to prefetching with its numerical counterpart computed using (2). It is clear from the figure that the model is very accurate. The average load versus N_p is depicted in Figure 11. Overall, the modeled and simulated loads are sufficiently close to each other, with a slight deviation when N_p is high. This deviation comes from the slight change in h_{proxy} due to prefetching, which we assumed in our analysis to be independent of prefetching. Although we assumed that prefetched documents are not cached at the proxy, prefetching can affect h_{proxy} as it changes the stream of Web requests seen by the proxy.

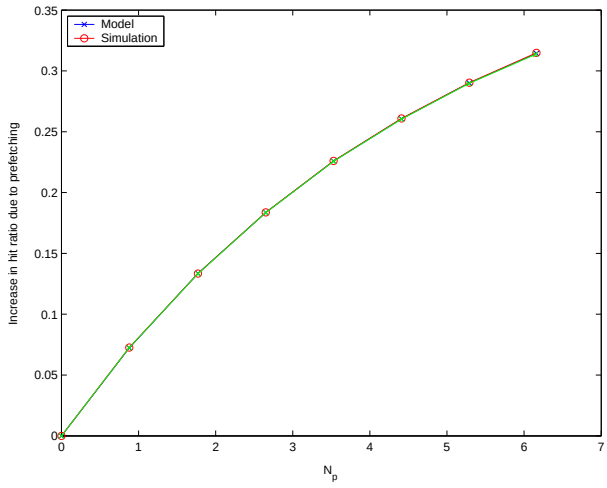


Fig. 10. Increase in the client's cache hit ratio due to prefetching versus N_p when $\bar{P} = \frac{3.5(1-e^{-0.18N_p})}{N_p}$ ($r = 500$ kbps, $C = 500$ kbps, $\lambda = 8$ files/s, $\bar{s} = 38$ kbits, $h_{proxy} = 0.39$, $h_c = 0.31$).

E. Validating the Access Improvement Index

Figure 12 depicts I versus N_p, computed using the analytical model and the simulations. The two plots depict a similar trend. Surprisingly, the prefetching gain in the simulations is lower than the one obtained

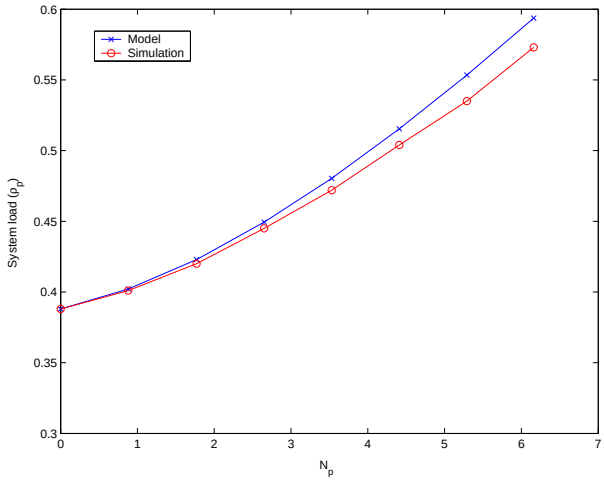


Fig. 11. Average system load under prefetching versus N_p for the case $\bar{P} = \frac{3.5(1-e^{-0.18N_p})}{N_p}$ ($r = 500$ kbps, $C = 500$ kbps, $\lambda = 8$ files/s, $\bar{s} = 38$ kbits, $h_{proxy} = 0.39$, $h_c = 0.31$).

using the analysis. One reason for the difference is that the analysis relies on the average file size, whereas the file size is highly variable (follows a heavy-tail distribution). To test the effect of the file size on the average access delay, we reran the simulations, assigning to all files the same size (average file size). The outcome of this simulation experiment is shown in Figure 13. It is clear that our analysis needs to account for the high variability in the file size. This can be done by modelling the average access delay for a single byte of data. Hence, we use the byte hit ratio of the caching system to compute the probability of finding an arbitrary byte of data in a given cache. Accordingly, the average access time of an arbitrary byte is computed as:

$$A_P(byte) = \frac{1}{r}(1 - \widetilde{h_c} - \widetilde{\Delta_h})(\widetilde{h_{proxy}} + (1 - \widetilde{h_{proxy}})f_R(\rho_p)) \tag{20}$$

where $\widetilde{h_c}$ is the regular-cache *byte* hit ratio, $\widetilde{h_{proxy}}$ is the proxy cache byte hit ratio, and $\widetilde{\Delta_h}$ is the increase in the local cache byte hit ratio due to prefetching. To validate this revised model, we reran the simulations to compute the byte hit ratios for all caches. Figure 14 shows the numerical results for the original and revised models, along with the simulation results. It is clear that the results for the revised model are quite close to the simulations.

Based on the above revised model, we run new simulations using the following adaptive prefetching mechanism. Each client dynamically adjusts the number of files to prefetch at the beginning of each

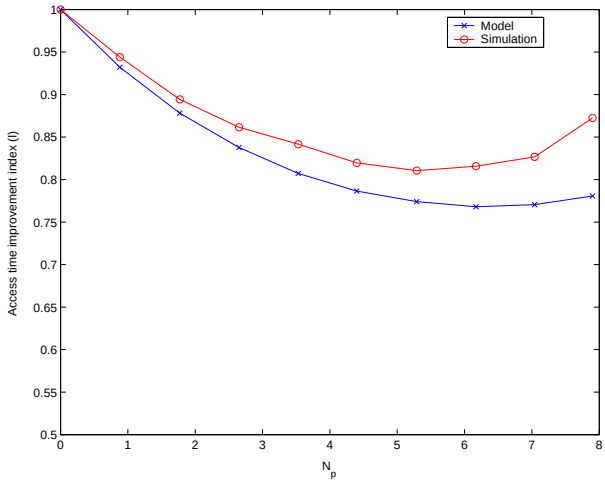


Fig. 12. I versus N_p ($\bar{P} = \frac{3.5(1-e^{-0.18N_p})}{N_p}$, $r = 500$ kbps, $C = 500$ kbps, $\lambda = 8$ files/s, $\bar{s} = 38$ kbits, $h_{proxy} = 0.39$, $h_c = 0.31$).

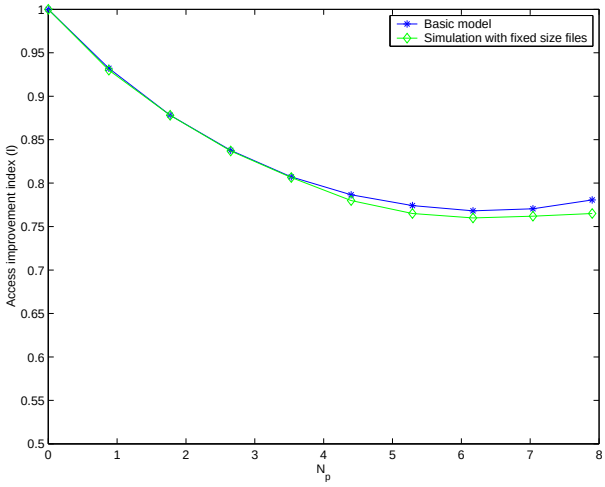


Fig. 13. I versus N_p under fixed-size files ($\bar{P} = \frac{3.5(1-e^{-0.18N_p})}{N_p}$, $r = 500$ kbps, $C = 500$ kbps, $\lambda = 8$ files/s, $\bar{s} = 38$ kbits, $h_{proxy} = 0.39$, $h_c = 0.31$).

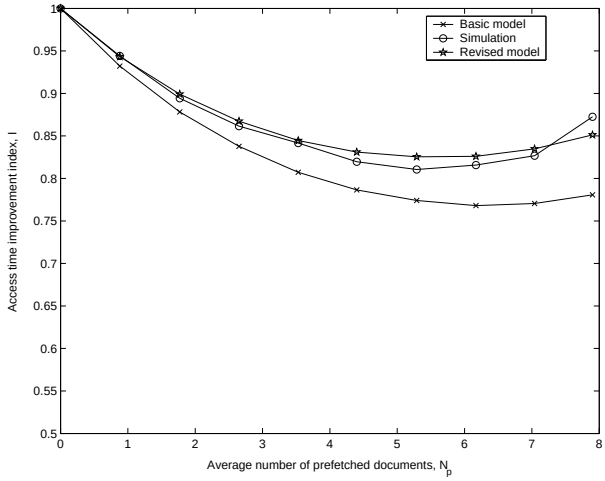


Fig. 14. I versus N_p ($\bar{P} = \frac{3.5(1-e^{-0.18N_p})}{N_p}$, $r = 500$ kbps, $C = 500$ kbps, $\lambda = 8$ files/s, $\bar{s} = 38$ kbits, $h_{proxy} = 0.39$, $h_c = 0.31$).

OFF period based on the estimated system load, the prefetching precision, and the proxy and regular caches' *byte* hit ratios. As before, these hit ratios are estimated from historical data. The increase in the local cache hit ratio due to prefetching (Δ_h) is estimated based on the number of files the client intends to prefetch. The estimated Δ_h is used to compute the increase in the local cache byte hit ratio due to prefetching ($\widetilde{\Delta_h}$). This is done by multiplying Δ_h by a correction factor α, *which represents the average ratio of prefetching cache byte hit ratio to its file hit ratio that can start with one and gets updated during continual prefetching*. Note that for the purpose of optimizing N_p, Δ_h must be estimated for several values of N_p. Figure 15 shows the simulation results for the adaptive prefetching protocol. In this plot, we also show the results under non-adaptive prefetching, where we run several simulation experiments and in each experiment we set N_p to a given value. From the non-adaptive prefetching simulation, we found that $N_p^* \approx 5.3$ files. Based on the simulation of the adaptive protocol, the average number of prefetched files was found to be $N_p = 5.6$, which is very close to N_p^*. Moreover, I for the adaptive protocol is very close to $I(N_P^*)$.

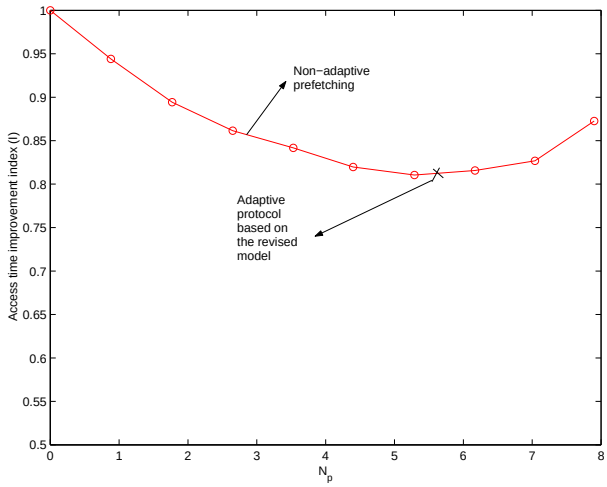


Fig. 15. I versus N_p ($\bar{P} = \frac{3.5(1-e^{-0.18N_p})}{N_p}$, $r = 500$ kbps, $C = 500$ kbps, $\lambda = 8$ files/s, $S = 38$ kbits, $h_{proxy} = 0.39$, $h_c = 0.31$).

VI. Conclusions

In this paper, we analyzed the performance of a generic client-side prefetching system. We considered the access time improvement as the performance metric. Our model considers both proxy and client caches. Based on our analysis, we obtained an expression for the prefetching threshold that can be set

dynamically to optimize the effectiveness of prefetching. We proposed a prefetching protocol that uses our analytical results for optimizing the prefetching gain. We investigated the effect of the caching system on the effectiveness of prefetching. We discovered that prefetching all documents with access probabilities greater than a given threshold value does *not* always lead to the minimum access delay, as was reported in previous works. This is only true for the case when clients have high access speeds relative to the access speed of their common proxy. For the other cases, the access delay improves with the increase in the number of prefetched documents until a certain point, after which the trend is reversed. Moreover, we found that prefetching is always profitable even with the existence of a good caching system. We also observed that the high variability in Web file sizes limits the effectiveness of prefetching.

In this work, we assumed that each client runs one browsing session at a time. The one-session assumption is acceptable for clients with low-bandwidth connections. The case of multiple sessions is more common for clients with high-bandwidth connections, which we leave for a future work.

APPENDIX

A. Model Extension to Heterogenous Clients

The model in Section II deals with a homogenous environment in which clients have the same access speed r, the same cache hit rate h_c, and the same ON/OFF statistics. In this section, we explain how our model can be extended to clients with heterogenous characteristics by employing a generalized version of the M/G/R-PS system. Such a queueing system has recently been studied by Kawahara et al. [25]. According to the extended model, client i, $i = 1, \ldots, n$, is characterized by an access speed r_i, a cache hit rate h_{c_i}, an average demand λ_i, and an average ON duration N_{on_i} (in files). This client gets $\min\{r_i, \frac{Cr_i}{\sum_{j=1}^{n} r_j}\}$ of the shared capacity. We now show how to derived the access improvement index under this model.

First, we determine the mean file transfer time for client i ($\bar{t}_i$). Similar to $\bar{t}$ in (6), $\bar{t}_i$ is given by:

$$\bar{t}_i = \frac{\overline{s}}{r_i} f_{R_i}(\rho) \tag{21}$$

where f_{R_i} is the same as the delay factor f_R in (7) but with $R_i \stackrel{\text{def}}{=} C/r_i$ replacing R. To determine $\bar{t}_i$ with and without prefetching, we need to compute the corresponding loads over the shared access link, ρ_p and

ρ_{np}, respectively, which are given by:

$$\rho_{np} = \frac{(1-h_{proxy})\sum_{i=1}^{n}(1-h_{c_i})\lambda_i\overline{s}}{C} \quad (22)$$

$$\rho_{p} = \frac{(1-h_{proxy})\left[\sum_{i=1}^{n}(1-h_{c_i}-\Delta_{h_i}+N_{p_i}/N_{on_i})\lambda_i\right]\overline{s}}{C} \quad (23)$$

where $\Delta_{h_i} = N_{p_i}\overline{P}/N_{on_i}$ is the increase in client i's hit ratio due to prefetching and N_{p_i} is the average number of files that client i prefetches in its next OFF period. Equations (22) and (23) are the counterparts of (9) and (10) for the homogeneous case. Accordingly, the mean access times without and with prefetching for an arbitrary file that is demand-requested by client i are given by:

$$A_{np}^{(i)} = (1-h_{c_i}).\left[h_{proxy}t_{prox,i}(\overline{s}) + (1-h_{proxy})t_{serv,i}(\overline{s})\right] \quad (24)$$

$$A_{p}^{(i)} = (1-h_{c_i}-\Delta_{h_i}).\left[h_{proxy}t_{prox,i}(\overline{s}) + (1-h_{proxy})t'_{serv,i}(\overline{s})\right] \quad (25)$$

where $t_{prox,i}(\overline{s})$, $t_{serv,i}(\overline{s})$, and $t'_{serv,i}(\overline{s})$ are the same as $t_{prox}(\overline{s})$, $t_{serv}(\overline{s})$, and $t'_{serv}(\overline{s})$, but for client i.

From (24) and (25), we can compute the access improvement index for client i:

$$I_i \stackrel{\text{def}}{\equiv} \frac{A_p^{(i)}}{A_{np}^{(i)}} = \frac{(1-h_{c_i}-\Delta_{h_i}).\left[h_{proxy}t_{prox,i}(\overline{s}) + (1-h_{proxy})t'_{serv,i}(\overline{s})\right]}{(1-h_{c_i}).\left[h_{proxy}t_{prox,i}(\overline{s}) + (1-h_{proxy})t_{serv,i}(\overline{s})\right]}. \quad (26)$$

Note that for $i = 1, \ldots, n$, I_i is a function of N_{p_i}, which is the parameter to be optimized. For the underlying heterogenous case, determining the cost function to minimize with respect to $N_{p_1}, N_{p_2}, \ldots, N_{p_n}$ requires specifying a notion of fairness. For example, if clients are to be treated equally, then the cost function to minimize is simply given by $I = (\sum_{i=1}^{n} I_i)/n$. However, it may be argued that such a cost function is not fair to clients with fast connections, which should, arguably, be allowed to prefetch more files than clients with slower connections. On the other hand, it may also be argued that slow clients benefit more from prefetching than fast clients, and so they should be given more weight in the cost function. In general, all of these notions of fairness can be handled by minimizing a weighted cost function $\sum_{i=1}^{n} w_i I_i$, where the weights $w_1, \ldots, w_n$ are determined based on whatever concept of fairness is being adopted. Note that similar to I, the function $\sum_{i=1}^{n} w_i I_i$ is convex in the optimization parameters, so it can be easily minimized using numerical approaches.

B. Proof of Theorem 3.1

First, we show that if prefetching a single file or a fraction of a file does not improve the mean file access time, then increasing the number of prefetched files does not do any better. To do that, we express I as the product of two functions $f_1(x)$ and $f_2(x)$, where x is the average number of prefetched files in an OFF period:

$$I = f_1(x).f_2(x) \tag{27}$$

where

$$f_1(x) \stackrel{\text{def}}{=} \frac{1 - h_c - \Delta_h(x)}{1 - h_c} \leq 1 \tag{28}$$

$$f_2(x) \stackrel{\text{def}}{=} A + B f_R(\rho_p(x)) \geq 1 \tag{29}$$

$$A \stackrel{\text{def}}{=} \frac{h_{proxy}}{h_{proxy} + (1 - h_{proxy}) f_R(\rho_{np})} \tag{30}$$

$$B \stackrel{\text{def}}{=} \frac{1 - h_{proxy}}{h_{proxy} + (1 - h_{proxy}) f_R(\rho_{np})}. \tag{31}$$

We approximate $f_R(\rho)$ in (7) by:

$$f_R(\rho) \approx \frac{1}{1 - \rho^R}. \tag{32}$$

The goodness of this approximation is demonstrated in Figure 16.

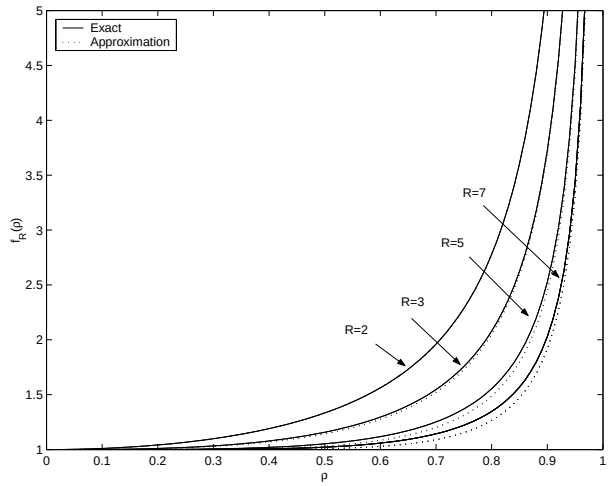


Fig. 16. Approximation of $f_R(\rho)$ by $1/(1 - \rho^R)$.

It is easy to show that $f_1(x)$ deceases monotonically with x, since $\frac{df_1(x)}{dx} = \frac{-\frac{d\Delta_h}{dx}}{1-h_c} < 0$ for all $0 \leq x < \infty$. Note that $\frac{d\Delta_h}{dx} > 0$, as prefetching always increases the overall cache hit ratio.

On the other hand, $f_2(x)$ increases monotonically with x, considering that for all $0 \leq x < \infty$ we have

$$\frac{df_2(x)}{dx} = B\frac{df_R(\rho_p)}{dx} = B\frac{R\rho_p^{R-1}}{(1-\rho_p^R)^2}\frac{d\rho_p}{dx} > 0. \tag{33}$$

Note that $\frac{d\rho_p}{dx} > 0$, because prefetching always increases the network traffic unless the prediction is 100% accurate.

Now if we can show that $f_1(x)$ decreases at a slower rate than the rate at which $f_2(x)$ increases, then we can say for sure that there is no gain out of prefetching more files if prefetching a single or a fraction of a file is not beneficial. This also assures that if there is a gain out of prefetching, then there is a unique value for N_p^*. Formally, we need to show that $\frac{d|f_1^{'}(x)|}{dx} < 0$ and $\frac{d|f_2^{'}(x)|}{dx} > 0$. Consider the first inequality. Recall that $\Delta_h = \frac{x\,\bar{P}(x)}{N_{on}}$. Then,

$$\begin{aligned}\frac{d|f_1^{'}(x)|}{dx} &= \frac{1}{1-h_c}\frac{d^2\Delta_h}{dx^2} \\ &= \frac{1}{(1-h_c)N_{on}}(\bar{P}^{''}(x)x + 2\bar{P}^{'}(x)) < 0.\end{aligned} \tag{34}$$

Note that $\bar{P}^{'}(x) < 0$ because $\bar{P}(x)$ is a monotonically decreasing function in x. Also, $\bar{P}^{''}(x) < 0$ since the popularity of Web files follows a Zipf-like distribution ($P_i \sim \frac{1}{i^\alpha}$, $0 \leq \alpha \leq 1$). For $\frac{d|f_2^{'}(x)|}{dx}$, we have

$$\begin{aligned}&\frac{d|f_2^{'}(x)|}{dx} = \frac{BR\rho_p^{R-2}((R-1)(1-\rho_p^R) + 2\rho_p^R R(\frac{d\rho_p}{dx})^2)}{(1-\rho_p^R)^2} + \\ &\frac{B.R\rho_p^{R-1}}{(1-\rho_p^R)^2}.\frac{d^2\rho_p}{dx^2} > 0.\end{aligned} \tag{35}$$

Formally, for prefetching to be beneficial, the following condition must be satisfied:

$$\lim_{x\to 0^+}\left|\frac{df_1}{dx}\right| > \lim_{x\to 0^+}\left|\frac{df_2}{dx}\right|$$

$$
\begin{aligned}
&\text{iff} \quad \lim_{x\to 0^+}\left|\frac{\frac{-d\Delta_h}{dx}}{1-h_c}\right| > \lim_{x\to 0^+}\left|\frac{BR\rho_p^{R-1}}{(1-\rho_p^R)^2}\frac{d\rho_p}{dx}\right| \\
&\text{iff} \quad \lim_{x\to 0^+}\left|\frac{\frac{-d\Delta_h}{dx}}{1-h_c}\right| > \\
&\lim_{x\to 0^+}\left|\frac{BR\rho_p^{R-1}}{(1-\rho_p^R)^2}(1-h_{proxy})(\frac{1}{N_{on}}-\frac{d\Delta_h}{dx})\frac{\lambda\bar{s}}{C}\right| \\
&\text{iff} \quad \lim_{x\to 0^+}\left|\frac{\bar{P}'(x)x+\bar{P}(x)}{N_{on}(1-h_c)}\right| > \\
&\lim_{x\to 0^+}\left|\frac{BR\rho_p^{R-1}}{(1-\rho_p^R)^2}(1-h_{proxy})(\frac{1}{N_{on}}-\frac{\bar{P}'(x)x+\bar{P}(x)}{N_{on}})\frac{\lambda\bar{s}}{C}\right| \\
&\text{iff} \quad \frac{\lim_{x\to 0^+}\bar{P}(x)}{N_{on}(1-h_c)} > \\
&\frac{BR\rho_{np}^{R-1}}{(1-\rho_{np}^R)^2}(1-h_{proxy})(\frac{1}{N_{on}}-\frac{\lim_{x\to 0^+}\bar{P}(x)}{N_{on}})\frac{\lambda\bar{s}}{C}.
\end{aligned}
$$

Note that $\lim_{x\to 0^+}\bar{P}(x)=\bar{P}(1)$. Defining $M \stackrel{\text{def}}{=} \frac{BR}{(1-\rho_{np}^R)^2} = \frac{(1-h_{proxy})R}{(1-\rho_{np}^R)(1-h_{proxy}\rho_{np}^R)}$, we end up with

$$\bar{P}(1) > \frac{M\rho_{np}^R}{1+M\rho_{np}^R}. \tag{36}$$

C. Proof of Theorem 3.2

Let x be the number of prefetched files. Then, I can be expressed as

$$I = Lg(x) + \frac{Mg(x)}{1-\alpha g(x)-\beta x} \tag{37}$$

where

$$g(x) \stackrel{\text{def}}{=} 1-h_c-\Delta_h(x) \tag{38}$$

$$L \stackrel{\text{def}}{=} \frac{h_{proxy}}{(1-h_c)(h_{proxy}+(1-h_{proxy})f_R(\rho_{np}))} \tag{39}$$

$$M \stackrel{\text{def}}{=} \frac{1-h_{proxy}}{(1-h_c)(h_{proxy}+(1-h_{proxy})f_R(\rho_{np}))} \tag{40}$$

$$\alpha \stackrel{\text{def}}{=} (1-h_{proxy})\rho \tag{41}$$

$$\beta \stackrel{\text{def}}{=} \frac{\alpha}{N_{on}}. \tag{42}$$

Now to optimize I, we let $\frac{dI}{dx} = 0$ and solve for x:

$$\frac{dI}{dx} = Lg'(x) + \frac{Mg'(x)(1-\beta x) + M\beta g(x)}{(1-\alpha g(x) - \beta x)^2} = 0. \tag{43}$$

With $\Delta_h(x)$ defined according to (18), $g'(x)$ reduces to $\frac{-\bar{P}}{N_{on}}$. Solving (43) for x yields

$$x = \begin{cases} \frac{-B-\sqrt{B^2-4AC}}{2A}, & \text{if } B^2 - 4AC > 0 \\ \text{Largest number of candidate} & \\ \text{files subject to } \rho_p < 1, & \text{otherwise} \end{cases} \tag{44}$$

where, A, B, and C are given in (15), (16), and (17), respectively.

To prove the second part of Theorem 3.2, we know that for prefetching to be of a value, we must have $I < 1$. Therefore,

$$\begin{aligned} & I < 1 \\ \text{iff} \quad & \frac{(1-h_c-\Delta_h)(h_{proxy} + (1-h_{proxy})f_R(\rho_p))}{(1-h_c)(h_{proxy} + (1-h_{proxy})f_R(\rho_{np}))} < 1 \\ \text{iff} \quad & \frac{(1-h_c-\Delta_h)}{(1-h_c)} < \frac{h_{proxy} + (1-h_{proxy})f_R(\rho_{np})}{h_{proxy} + (1-h_{proxy})f_R(\rho_p)} \\ \text{iff} \quad & 1 - \frac{\overline{P}N_p}{(1-h_c)N_{on}} < \frac{h_{proxy} + (1-h_{proxy})f_R(\rho_{np})}{h_{proxy} + (1-h_{proxy})f_R(\rho_p)}. \end{aligned}$$

Taking $h_{proxy} = 0$, we end up with

$$\begin{aligned} 1 - \frac{\overline{P}x}{(1-h_c)N_{on}} &< \frac{f_R(\rho_{np})}{f_R(\rho_p)} \\ &< f_R(\rho_{np})(1-\rho_p). \end{aligned}$$

Because ρ_p is linear in x, both sides of the above inequality decrease linearly with x. Hence, if the rate at which the left-hand side (LHS) decreases at $x = 0$ is greater than the rate of the right-hand side (RHS), then increasing the value of x increases the reduction in I (improves I).

For the rate of the LHS to be greater than the rate of the RHS, we must have the following:

$$\begin{aligned} \frac{\bar{P}}{(1-h_c)N_{on}} &> \frac{(1-h_{proxy})(1-\bar{P})\rho}{(1-\rho_{np})N_{on}} \\ \Rightarrow \bar{P} &> \rho_{np} \end{aligned}$$

which is the threshold value that is necessary for prefetching when $R = 1$ and $h_{proxy} = 0$.

D. Proof of Theorem 3.4

Consider A_p as defined in (4) and $f_R(\rho)$ as defined in (32). Then,

$$\frac{dA_p}{dR} = (1 - h_c - \Delta_h)(1 - h_{proxy})\frac{\rho_p^R \ln(\rho_p)}{(1-\rho_p^R)^2}, \tag{45}$$

which is less than zero because $\ln(\rho_p) < 0$. Accordingly, the access time with prefetching decreases with R.

For h_{proxy}, we have

$$\begin{aligned} &\frac{dA_p}{dh_{proxy}} = (1 - h_c - \Delta_h) \\ &(1 - \frac{1-(1-h_{proxy})^R W + R(1-h_{proxy}^R)W}{(1-(1-h_{proxy})^R W)^2}) \end{aligned} \tag{46}$$

where

$$W \stackrel{\text{def}}{=} (1 - h_c - \Delta_h + \frac{N_p}{N_{on}})\frac{\lambda \bar{s}}{C}. \tag{47}$$

But $1-(1-h_{proxy})^R W = (1-\rho_p^R) < 1$. Therefore, $1-(1-h_{proxy})^R W > (1-(1-h_{proxy})^R W)^2$. Accordingly, $\frac{dh_{A_p}}{h_{proxy}} < 0$ and the access time with prefetching decreases with h_{proxy}.

REFERENCES

[1] http://developer.mozilla.org/en/docs/link_prefetching_faq.

[2] http://fasterfox.mozdev.org/.

[3] http://infonet.naist.jp/products/.

[4] http://webaccelerator.google.com/.

[5] http://www.faqs.org/rfcs/rfc2068.html.

[6] http://www.mozilla.com/en-us/firefox/.

[7] http://www.netscape.com/.

[8] http://www.w3.org/Protocols.

[9] Web 3000 Inc. (NetSonic Internet Accelerator). http://www.web3000.com/.

[10] PeakSoft corporation. PeakJet 2000 web page. http://www.peak.com/peakjet2.html, 2002.

[11] V. Almeida, A. Bestavros, M. Crovella, and A. deOliveira. Characterizing reference locality in the WWW. In *Proceedings of the Fourth International Conference on Parallel and Distributed Information Systems (PDIS)*, pages 92–103, 1996.

[12] A. Balamash and M. Krunz. WWW traffic modeling: A multifractal approach. *Computer Networks Journal*, 43(2):211–226, October 2003.

[13] P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. In *Proceedings of the ACM SIGMETRICS Conference*, pages 151–160, 1998.

[14] A. Bestavros. Using speculation to reduce server load and service time on the WWW. In *Proceedings of the 4th ACM International Conference on Information and Knowledge Management*, pages 403–410, 1995.

[15] L. Breslau, P. Cao, L. Fan, G. Philips, and S. Shenker. Web caching and zipf-like distributions: Evidence and implications. In *Proceedings of the INFOCOM Conference*, pages 126–134, 1999.

[16] P. Cao and S. Irani. Cost-aware WWW proxy caching algorithms. In *Proceedings of the 1997 USENIX Symposium on Internet Technology and System*, pages 193–206, 1997.

[17] M. Crovella and P. Barford. The network effects of prefetching. In *Proceedings of IEEE INFOCOM Conference*, pages 1232–1239, 1998.

[18] C. Cunha, A. Bestavros, and M. Crovella. Characteristics of WWW client-based traces. *IEEE/ACM Transactions on Networking*, 1(3):134–233, Jan 1999.

[19] B. Davison and V. Liberatore. Pushing politely: Improving web responsiveness one packet at a time. *Performance Evaluation Review*, 28(2):43–49, September 2000.

[20] M. Deshpande and G. Karypis. Selective Markov models for predicting Web page accesses. *ACM Transactions on Internet Technology*, 4(2):163–184, May 2004.

[21] F. Douglis, A. Feldmann, and B. Krishnamurthy. Rate of change and other metrics: a live study of the world wide web. In *Proceedings of USENIX Symposium on Internet Technologies and Systems*, 1997.

[22] D. Duchamp. Prefetching hyperlinks. In *Proceedings of 2nd USENIX Symposium on Internet Technologies and Systems*, pages 127–138, 1999.

[23] L. Fan, P. Cao, W. Lin, and Q. Jacobson. Web prefetching between low-bandwidth clients and proxies: Potential and performance. In *Proceedings of ACM SIGMETRICS Conference on Measurment and Modeling of Computer Systems*, pages 178–187, May 1999.

[24] Z. Jiang and L. Kleinrock. An adaptive network prefetch scheme. *IEEE Journal on Selected Areas in Communications (JSAC)*, 17(4):358–368, April 1998.

[25] R. Kawahara, K. Ishibashi, T. Mori, T. Ozawa, and T. Abe. Method of bandwidth dimensioning and management for aggregated TCP flows with heterogeneous access links. *IEICE Transactions on Communications*, EE88-B(12):4605–4615, December 2005.

[26] R. Kokku, P. Yalagandula, A. Venkataramani, and M. Dahlin. NPS: A non-interfering deployable web prefetching system. In *USENIX Symposium on Internet Technologies and Systems*, 2003.

[27] T. Kroeger, D. Long, and J. Mogul. Exploring the bounds of web latency reduction from caching and prefetching. In *USENIX Symposium on Internet Technologies and Systems*, 1997.

[28] K.-Y. Lam and C. Ngan. Temporal pre-fetching of dynamic web pages. *ACM Information Systems*, 31(1):149–169, May 2006.

[29] B. Lan, S. Bressan, B. C. Ooi, and K.-L. Tan. Rule-assisted prefetching in web-server caching. In *Proceedings of the ninth international conference on Information and knowledge management*, pages 504–511, McLean, Virginia, United States, 2000.

[30] R. Lempel and S. Moran. Optimizing result prefetching in web search engines with segmented indices. *ACM Transactions on Internet Technology*, 4(1):31–59, February 2004.

[31] T. S. Loon and V. Bharghavan. Alleviating the latency and bandwidth problems in WWW browsing. In *Proceedings of USENIX Symposium on Internet Technologies and Systems*, pages 219–230, 1997.

[32] E. Markatos and C. Chronaki. A top-10 approach to prefetching on the web. In *Proceedings of the INET Conference*, 1998.

[33] M. Molina, P. Castelli, and G. Foddis. Web traffic modeling exploiting tcp connections' temporal clustering through html-reduce. *IEEE Network Magazine*, 14(3):46–55, May 2000.

[34] A. Nanopoulos, D. Katsaros, and Y. Manolopoulos. Effective prediction of web-user accesses: a data mining approach. In *Proceedings of of web usage analysis and user profiling workshop*, pages 504–511, San Fransisco, CA, 2001.

[35] V. Padmanabhan and J. Mogul. Using predictive prefetching to improve world wide web latency. In *Proceedings of the ACM SIGCOMM Conference*, pages 26–36, 1996.

[36] T. Palpanas and A. Mendelzon. Web prefetching using partial match prediction. In *Proceedings of Web Caching Workshop*, San Diego, California, March 1999.

[37] A. B. Pandey, J. Srivastava, and S. Shekhar. Web proxy server with intelligent prefetcher for dynamic pages using association rules. Technical 01-004, University of Minnesota, Computer Science and Engineering, January 2001.

[38] J. Pitkow and P. Pirolli. Mining longest repeated subsequences to predict world wide web surfing. In *Proceedings of the Second USENIX Symposium on Internet Technologies and Systems*, Boulder, Colorado, USA, October 1999.

[39] M. Rabinovich and O. Spatscheck. *Web Caching and Replication*. Addison Wesley, 1st edition, December 2001.

[40] A. Riedl, T. Bauschert, M. Perske, and A. Probst. Inverstigation of the $M/G/R$ processor sharing model for dimensioning IP networks with elastic traffic. In *First Polish-German Teletraffic Symposium PGTSDresden*, September 2000.

[41] R. Sarukkai and S. Clara. Link prediction and path analysis using Markov chains. In *Proceedings of the Ninth International World Wide Web Conference*, pages 377–386, Amsterdam, The Netherlands, 2000.

[42] S. Schechtera, M. Krishnanb, and M. Smithc. Using path profiles to predict http requests. In *Proceedings of the 7th International World Wide Web Conference*, pages 457–467, April 1998.

[43] W.-G. Teng, C.-Y. Chang, and M.-S. Chen. Integrating web caching and web prefetching in client-side proxies. *IEEE Transactions on Parallel and Distributed Systems*, 16(5):444–455, May 2005.

[44] N. J. Tuah, M. Kumar, and S. Venkatesh. Resource-aware speculative prefetching in wireless networks. In *Wireless Networks Journal*, volume 9, pages 61–72, 2003.

[45] A. Venkataramani, R. Kokku, and M. Dahlin. TCP Nice: A mechanism for background transfers. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, volume 36, pages 329–343, 2002.

[46] A. Venkataramani, P. Yalagandula, R. Kokku, S. Sharif, and M. Dahlin. The potential costs and benefits of long-term prefetching for content distribution. In *The Sixth Web Caching and Content Distribution Workshop*, 2001.

[47] B. Wu and A. D. Kshemkalyani. Objective-optimal algorithms for long-term Web prefetching. *IEEE Transactions on Computers*, 55(1):2–17, 2006.

[48] Q. Yang, H. H. Zhang, and T. Li. Mining web logs for prediction models in WWW caching and prefetching.

[49] I. Zukerman, D. Albrecht, and A. Nicholson. Predicting users' requests on the WWW. In *Proceedings of the 7th International Conference on User Modeling*, pages 275–284, 1999.