An Adaptive Network Prefetch Scheme *

Zhimei Jiang and Leonard Kleinrock †, *Fellow, IEEE*

Abstract

In this paper, we present an adaptive prefetch scheme for network use, in which we download files that will very likely be requested in the near future, based on the user access history and the network conditions. Our prefetch scheme consists of two parts: a prediction module and a threshold module. In the prediction module, we estimate the probability with which each file will be requested in the near future. In the threshold module, we compute the prefetch threshold for each related server, the idea being that the access probability is compared to the prefetch threshold. An important contribution of this paper is that we derive a formula for the prefetch threshold to determine its value dynamically based on system load, capacity, and the cost of time and system resources to the user. We also show that by prefetching those files whose access probability is greater than or equal to its server's prefetch threshold, a lower average cost can always be achieved. As an example, we present a prediction algorithm for web browsing. Simulations of this prediction algorithm show that, by using access information from the client, we can achieve high successful prediction rates, while using that from the server generally results in more hits.

1 Introduction

Due to the proliferation of the World Wide Web (WWW), there has been a large increase in the amount of information transmitted over the Internet. However, the growth of internet capacity is not keeping pace, and often users experience long delays in retrieving files from remote machines when running network related applications. Among these network applications, the most popular ones are Web browsing, email and news groups. Sometimes, the waiting does not end even after the files have made it to the local disk. For example, after an access, the client machine may need to decompress the received files, compile Java programs, etc. On the other hand, between two network accesses, users will be viewing the information just downloaded and the local machine is generally idle. The key idea of *prefetching* is to take advantage of these idle periods to fetch the files that will very likely be requested in

*This work was supported by the Advanced Research Projects Agency, ARPA/CSTO, under Contract DABT-63-94-C-0080 "Transparent Virtual Mobile Environment".

†The authors are with the Computer Science Department, University of California at Los Angeles, Los Angeles, CA. 90024. (email: {jiang, lk}@cs.ucla.edu)

the near future, so that the user's average waiting time can be reduced. More specifically, we add some intelligence to the network applications such that whenever the user requests a block of information (a file, an email message, etc.), the system can estimate what additional information (files, messages, etc.)* will be needed in the next few user accesses and transmit some of them to the local disk beforehand, according to certain criteria. If the prefetched information is indeed requested, the user can access it with negligible delay. In addition, prefetch allows more time for sophisticated processing including encryption, compression, and compilation to be carried out at both server and client sites, so that data can be transmitted more efficiently and securely without increasing the delay to the user.

If we knew exactly which files a user needed next, we would retrieve only those files in advance. Assuming that the prefetch operation always finishes before the user requests the next file, then we would enjoy zero latency with no extra bandwidth consumption. Unfortunately, in reality, some prefetched information may never be used, resulting in wasted bandwidth and increased delay to normal (nonprefetch) requests. This shows that the prefetch problem has two aspects. The first one is how to estimate the probability of each file being accessed in the near future. In general, this probability changes with time as new requests are issued by the user. The second aspect is how to determine which files to prefetch such that the overall system performance can be optimized relative to some criteria. Previous studies on internet prefetch only focus on the first part of the problem and either use a fixed threshold or prefetch a fixed number of the most popular links on a page [2, 8, 10]. In PeakJet prefetching software for web browsers, all available links on the page being viewed are prefetched[12]. If everyone were to use this method, the problem of surges in network traffic is obvious. In fact, prefetching without caution could be disastrous to the network. We solve the second part of the prefetch problem by determining the prefetch threshold based on the network conditions in real time, which is the key to safely achieving high efficiency and reduced average delay for the entire system.

Regarding the two aspects of the prefetch problem, our prefetch scheme consists of two modules: the prediction module and the threshold module. The complete scheme works as follows. After a user's request for a new file is satisfied, the prediction module immediately updates the history information if needed, and computes the *access probability* for each candidate file, where the access probability of a file is an estimate of the probability with which the file will be requested by the user in the near future. Different applications may

*For simplicity, in the rest of this paper, we use the term "file" to represent a block of information whether it is a file, directory listing, web page, or other information requested by the user.

use different prediction algorithms in this module. Then the threshold module determines the prefetch threshold for each related server, which contains at least one candidate file with nonzero access probability. The algorithm used in this module is independent of the application. Finally, each file whose access probability exceeds or equals its server's prefetch threshold is prefetched. When prefetching a file, the file is actually downloaded if and only if no up-to-date version of the file is available on the local disk; otherwise no action is taken.

As an example, we present a simple prediction algorithm for web browsing, and study its performance through simulations in which the prefetch threshold is set at various values. Although web browsing is an important application for prefetching, our prefetch scheme may be applied to basically any network application in which prefetching is possible. In general, for different applications, different prediction algorithms may be developed, while the same threshold algorithm can be employed, and the same prefetching criteria can always achieve the optimum performance.

Details about the two modules of our prefetch scheme are covered in the next two sections. In section 2, we also present a prediction algorithm for web browsing. In section 3, we derive a formula for determining the prefetch threshold based on the network conditions. We also study the efficiency of our prediction algorithm for web browsing through trace-driven simulations. The results are summarized in section 4. Conclusions are presented in section 5.

2 The prediction module

The task of the prediction module is to keep track of the user's access history and to compute access probabilities in order to determine which files to prefetch. Different prediction algorithms can be designed for different applications. For instance, an application which handles email may assign each message a probability based on certain criteria, for example the latest/earliest message has the largest probability, or messages with a subject similar to the one being read have larger probabilities, etc., such that while a user is reading an email, the messages he might read soon may be prefetched. Another example is when a user logs in to a remote machine and switches to a directory, the files and subdirectories in this directory may be assigned certain access probabilities and some of them can be prefetched since user may need them soon. Web browsing is an application where sophisticated prediction is necessary. In the rest of this section, we present a prediction algorithm which can be used in web browsers.

In web browsing, we define the *access probability* $p(B|A)$ as the conditional probability that page B will be requested by the user in the near future given that page A is being viewed. Our prediction algorithm is based on the fact that typical web surfing involves a sequence of clicking and downloading operations. Namely, first the user clicks on a link, and a new page is downloaded. After viewing the new page, the user usually clicks on a link on this page to download another new page. Sometimes, the user may use the *"back"* or *"forward"* button on the browser to go back to some previous page and then click on another link on that page. Rarely does the user type in a URL to switch to a new page. We take advantage of this feature by only keeping track of the click operations and assuming that $p(B|A)$ is zero if no link exists on page A to page B. In other words, whenever a new page is displayed, the candidate files we consider only include those which are linked to this page. In our algorithm, the access probabilities of embedded images or programs have been set to zero, but they could be prefetched when the page which includes them is prefetched.

Both [2] and [10] studied prediction algorithms for the WWW. Parameters similar to access probability are defined in these work. In both algorithms, a window is applied to the access sequence, either in terms of time or in terms of the number of requests. Files within the same window are all considered to be dependent. Since the content of each page varies greatly and how pages are browsed differs from user to user, it is difficult to adjust the size of the time window. In the WWW, after an html file is downloaded, the images embedded in this file are downloaded immediately. Therefore, the request window includes both html files and embedded images, which makes it hard to determine the request window size as well. Since both algorithms keep all access pairs that are in the same window for future prediction, the memory space required could be as large as the square of the total number of files requested. Our scheme is similar to using a request window of size two after removing all the requests that were not sent explicitly by the user, for example those for embedded images, etc., from the request sequence and eliminating all the unrelated pairs of requests. The latter can be accomplished because we keep track of each request as well as the page from which the request was initiated by clicking on a link. It is easy to get this information at the client site. Note that the page from which a request is initiated may not be the previous one in the request sequence, since the user can click the "back" button on the browser to go back to an earlier page. Assuming the number of links on a page is bounded, our algorithm requires memory space linear to the number of *pages* on the server.

The details of the prediction algorithm are as follows. For each client, in order to collect the history information for prediction, the algorithm maintains two kinds of counters, the

page counter and the link counter. More specifically, each page A is associated with a page counter, C_A. In addition, if page B can be accessed directly from page A, namely there exists a link on page A pointing to page B, then and only then is there a link counter $C_{(A,B)}$ for the pair. Whenever page A is *downloaded*, C_A is increased by one. In addition, if page B is accessed by clicking on the corresponding link on page A, counter $C_{(A,B)}$ is also increased by one. The access probabilities are obtained in the following way. When a page, A, is being viewed by the user, for each page B_i linked to A, the access probability of B_i is computed as $p(B_i|A) = min(1, \frac{C_{(A,B_i)}}{C_A})$ for $C_A \geq 5$, and 0 for $C_A < 5$. All the files which are not linked to the page being viewed have access probability zero. The values of the counters are based on the access history during a certain period of time, for example the last 30 days. The algorithm is also illustrated in figure 1.

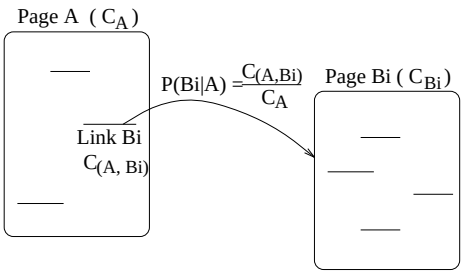

Figure 1: *Prediction algorithm for web browsing.*

Note that for a page A with k distinct links $(B_1, B_2, \cdots, B_k)$, $\sum_{i=1}^{k} P\{B_i|A\}$ can be greater than one. This is because a user may click link B_i on page A and come *"back"* later to go to another link B_j through page A. Therefore while page A is retrieved from the server only once, more than one file might be accessed from page A.

The prediction algorithm described above is for the client site browser. The same algorithm can also be implemented at the server site, where each server aggregates the access histories of all users to its pages and computes the access probabilities accordingly. The only difficulty at the server is that it is not straightforward to find out from which page a request was initiated. The best way to get this information is to require the client to include it in the http request. However, if we do not wish to change the http protocol, we can have the server keep track of the last page request from each of a certain number of users who accessed it most recently. When a new request from a tracked user arrives, the server assumes it was initiated from the previous page requested by that user if such a link exists.

Clearly, the access probabilities generated by the client site indicate the user's personal

interests, while those generated by the server site represent the popularity of the files over many users that have accessed that server. If a client has not visited a page often enough to obtain reliable access probabilities, the server's access probabilities are likely to be more accurate. Access probabilities from the server and client can be merged in the following way at the client site. If a page has been visited less than 5 times by the client, and the access probabilities of its links are available at the server, then the probabilities at the server can be used. Otherwise, the access probabilities from the client are used. More sophisticated algorithms may be developed to do this merging. In section 4, we compare the performance of the access probabilities obtained in different ways.

The algorithm described in this section is an essential component of a complete prediction algorithm for web browsing. More functions may be added for better prediction. Further discussion of this issue is beyond the scope of this paper. The idea we are trying convey through our prediction algorithm is that in order to predict a user's behavior in web browsing, it is not sufficient to just look at the sequence of requests independently, we must take advantage of the underlying link relations between pages to achieve better performance.

3 The threshold module

Our prefetching scheme tries to optimize the tradeoff between system resource (network link, server etc.) usage and latency, by predicting which files are likely to be needed soon and choosing some of them to download beforehand. The first part of this task is accomplished by the prediction module as we discussed in the last section. In the threshold module, we determine the prefetch threshold for the server in real time to determine which files to prefetch. Unlike the prediction module, the algorithm used in the threshold module is the same for all applications.

To optimize the tradeoff between system resource usage and latency, we choose to measure the system performance in terms of cost which is comprised of the *delay cost* (α_T \$/time unit) and the *system resource cost* (α_B \$/packet). The delay cost indicates how much the user suffers waiting for files. The system resource cost includes the cost of processing the packets at the end nodes and that of transmitting them from the source to the destination. In this section, we study how to determine which files to prefetch in order to minimize the average cost of requesting a file for several system models.

Previous work on prefetch uses either a fixed threshold for all servers or prefetches a

fixed number of files each time. The problem with these approaches is that they do not consider factors like system load and capacity, which can greatly affect the performance of prefetching. For example, we should be more cautious in prefetching files when the system load is high. In our algorithm, we compute the prefetch thresholds based on the network conditions in real time. Unless indicated otherwise, we assume that prefetches triggered by the previous request always finish before the user sends out the next request. In addition, we assume that when a prefetched file is requested for the first time, it is always up-to-date. We also assume infinite storage space at the client, so prefetched files are never deleted. These assumptions imply that the first request to a prefetched file will always result in a hit.

3.1 The single user system.

First, we consider a system in which there is one single user and multiple servers from which the user can retrieve files. In this system, the total cost of requesting a file is the sum of the system resource cost and the delay cost of waiting for the file to arrive[†]. Therefore, if the user requests a file that is not on the local disk, the total cost of retrieving this file is

$$c_1 = \alpha_B \cdot s + \alpha_T \cdot (t + t_0) = \alpha_B \cdot s + \alpha_T \cdot (\frac{s}{b} + t_0) \tag{1}$$

where s is the file size, t is the transmission time, t_0 is the startup time, and b is the capacity of the path to the server (packets/time unit). Assuming the cost of using a file stored on the local disk is negligible, if the requested file had previously been prefetched and saved on the local cache, then the cost associated with this request equals the system resource cost,

$$c_2 = \alpha_B \cdot s \tag{2}$$

because the delay cost is zero.

We now investigate the situation in which the user has just started using a new file, and currently there are L distinct files in the system with access probability greater than zero. The average cost, C, of satisfying all user requests stemming from this new file being used without any prefetching, is the summation of the costs to retrieve each of the L files times

[†]In this paper, we assume the processing delay is zero after the file has been received.

the access probability of the corresponding file [‡], i.e.

$$C = \sum_{i=1}^{L} p_i \cdot [\alpha_B \cdot s_i + \alpha_T \cdot (\frac{s_i}{b_i} + t_{0_i})] \tag{3}$$

where p_i is the access probability of file i, t_{0_i} is the start up time for retrieving ith file, and b_i is the capacity of the path to the server which contains the ith file.

If instead, m files among all the L candidate files are prefetched, say $i_1, i_2, \cdots, i_m$, this average cost becomes

$$C = \sum_{j=1}^{m} \alpha_B \cdot s_{i_j} + \sum_{j=m+1}^{L} p_{i_j} \cdot [\alpha_B \cdot s_{i_j} + \alpha_T \cdot (t_{i_j} + t_{0_{i_j}})] \tag{4}$$

where $i_j \in \{1, \cdots, L\}$, $j = 1 \cdots L$, and for any $j_1 \neq j_2$, $i_{j_1} \neq i_{j_2}$. Note that these L files could be on different servers and they are the only files that may be prefetched, because we never prefetch files with access probability zero.

Comparing (3) and (4), we conclude that the average cost is minimized if and only if we prefetch all and only those files which satisfy

$$p_i > \frac{1}{\frac{\alpha_T}{\alpha_B \cdot b_i}(1 + \frac{s_{0_i}}{s_i}) + 1} \tag{5}$$

where $s_{0_i} = b_i \cdot t_{0_i}$, and b_i is the capacity of the path to server i. We define the prefetch threshold $H = \frac{1}{\frac{\alpha_T}{\alpha_B \cdot b_i}(1+\frac{s_{0_i}}{s_i})+1}$ and let $r = \frac{\alpha_T}{\alpha_B}$. In figure 2, the prefetch threshold (H) is plotted as a function of the available bandwidth b, for several values of $r(=\frac{\alpha_T}{\alpha_B})$ at $\frac{s_{0_i}}{s_i} = 0.1$. Note that, the ratio $\frac{\alpha_T}{\alpha_B}$ indicates how valuable time is compared to the system resources.

Figure 2 shows that, for fixed $\frac{\alpha_T}{\alpha_B}$, more files should be prefetched from a server to which the capacity of the path is smaller, because it takes longer for the user to fetch the file himself. In addition, the larger the ratio $\frac{\alpha_T}{\alpha_B}$, that is, the more expensive time is relative to the system resources, the lower the prefetch threshold is for fixed bandwidth.

3.2 The multi-user system.

The result in the last subsection is based on the fact that, in a single user system, the available bandwidth to each server is always the full capacity of the path. In a multi-user

[‡]A file can be viewed more than once. Since we have assumed that the cost of a request which does not invoke file transfer is negligible, when the file is requested again, if the copy the user viewed last time is still available on the local cache and up-to-date, the cost for this request is zero regardless of whether the first request was satisfied by a prefetched file or not. Therefore, we do not need to consider this case in the cost function. If there is not an up-to-date copy of the file that is available locally, the request is treated as a new request.

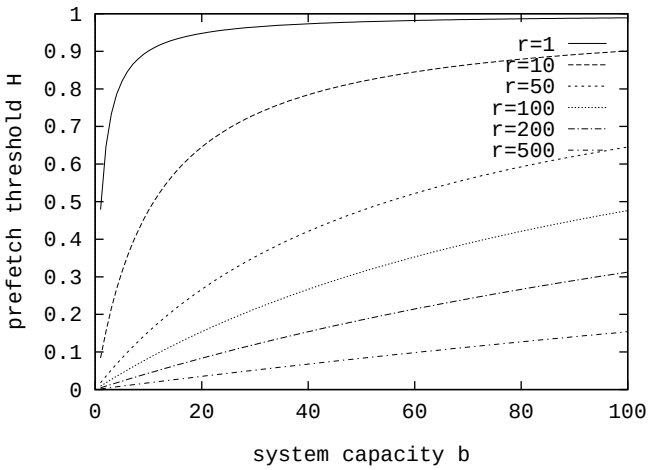

Figure 2: *Prefetch threshold H as a function of system capacity in the single user system, where* $H = \frac{1}{\frac{\alpha_T}{\alpha_B \cdot b_i}(1+\frac{s0_i}{s_i})+1}$, $r = \frac{\alpha_T}{\alpha_B}$, *and* $\frac{s0_i}{s_i} = 0.1$.

system where system resources are shared among users, prefetching increases the delay of other users' normal (nonprefetch) requests. Therefore, different criteria must be used to determine which files to prefetch in such a system.

The multi-user system we study consists of a single server and multiple users who share the same server and the network link as shown in figure 3a. Users issue requests for files on the shared server. Each user's local system can prefetch files while the user is working. We model this prefetch system as an $M/G/1$ Round-Robin processor-sharing system with two Poisson inputs, shown in figure 3b, where the server in the model represents both the network link and the server in the real system. The server handles two kinds of requests (inputs): *normal requests*, which are those user requests that can't be satisfied by the prefetched files on the local disk, and *prefetch requests*, which are the requests sent by the prefetch program. All requests to the server are of the same priority and they join the same queue waiting for service. We assume that those user requests which do not invoke a file transfer consume very little resources and are negligible. Thus, if a user request can be satisfied by a prefetched file, it adds no cost to the system.

We start with a special kind of system in which, at any time, the access probability of each file is either exactly p ($p > 0$) or 0, where p is fixed within one system but can vary for different systems. Files with access probability zero may actually be requested by the user but they are never prefetched. This implies that for a given system and its p, all the

prefetched files will be requested by the user with the same probability p. For simplicity, we initially assume that the startup time (t_0) is zero. The result for the system with nonzero startup time is given at the end of this section. Let us now derive the prefetch threshold step by step.

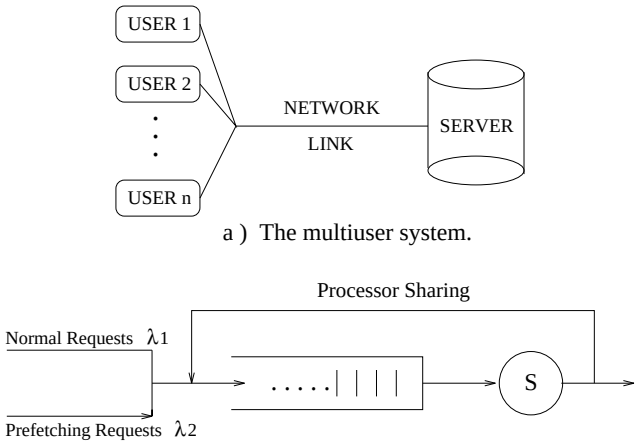

Figure 3: *The multi-user system and the prefetch model.*

a) The cost function C.

For a given system, let λ be the arrival rate of the file requests from the users when no prefetch is applied. We assume that prefetch does not affect the user's behavior regarding the likelihood of accessing files. In other words, when prefetching is employed, users still issue requests at rate λ in the same pattern, although they can get some files faster due to prefetching. Let the arrival rate of normal requests and prefetch requests be λ_1 and λ_2 respectively. Hence the rate at which user requests are satisfied by the prefetched files is $\lambda - \lambda_1$, which is simply $p\lambda_2$ because prefetched files are eventually requested by the user with probability p, where p is the access probability of the prefetched files. Thus $\lambda_1 + p \cdot \lambda_2 = \lambda$ or

$$\lambda_1 + \lambda_2 = \lambda + (1 - p)\lambda_2 \tag{6}$$

where $\lambda_1 + \lambda_2$ must be less than b/s in order to keep the system in a stable state.

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

$$t = \frac{x}{1 - \rho} = \frac{s}{b(1 - \rho)} \tag{7}$$

where ρ is the system load, s is the average file size, and b is the system capacity[6, 7]. For the system shown in figure 3b, $\rho = s(\lambda_1 + \lambda_2)/b$. This implies that, in a multi-user system, the cost of a normal request, which is the sum of the system resource cost and the delay cost, becomes

$$c_1 = \alpha_B \cdot s + \alpha_T \cdot t = \alpha_B \cdot s + \alpha_T \cdot \frac{s}{b - (\lambda_1 + \lambda_2)s} \tag{8}$$

where $b > (\lambda_1 + \lambda_2)s$. Notice that in equation (8), the effect of prefetching on the entire system is considered by including λ_2 in the formula. As more files are prefetched, the cost of normal requests increases because prefetching increases the system load, hence the delay of retrieving files increases. The average cost of a prefetch request is still

$$c_2 = \alpha_B \cdot s \tag{9}$$

Since users issue requests with rate λ, and some of them ($p\lambda_2$) are satisfied by prefetched files, the rest (λ_1) are sent to the server as normal requests; by equations (6), (8), and (9), we obtain that the average cost of an explicit user request is

$$\begin{aligned} C &= \frac{\lambda_1 \cdot c_1 + \lambda_2 \cdot c_2}{\lambda} \\ &= \frac{s}{\lambda}[(\lambda + (1-p)\lambda_2)\alpha_B + \frac{(\lambda - p\lambda_2)\alpha_T}{b - (\lambda + (1-p)\lambda_2)s}] \end{aligned} \tag{10}$$

This equation for the average cost C is plotted in figure 4 as a function of λ_2 for several values of p. Note that for equation (10), the valid range of λ_2 is $0 \leq \lambda_2 \leq \frac{\lambda}{p}$ for $0 < p \leq 1$.

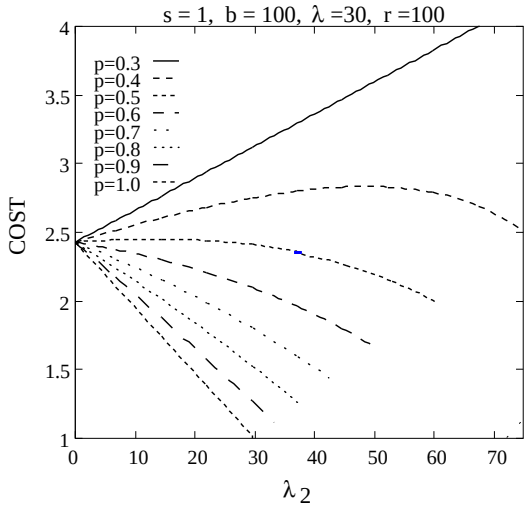

Figure 4: *Average cost of a user request as a function of* λ_2 *for p from* 0.3 *to* 1. ($s = 1, b = 100, \lambda = 30, r = \frac{\alpha_T}{\alpha_B} = 100$)

b) The optimum value of the prefetch rate λ_2.

Assume p and λ are known, we wish to find the value of the prefetch rate λ_2 which minimizes the average cost of each request in the system. Once λ_2 is determined, λ_1 can be computed by $\lambda_1 = \lambda - p\lambda_2$ according to (6).

From equation (10), clearly, at p=1, C is minimized when $\lambda_2 = \lambda$, i.e. all the files which have access probability 1 should be prefetched. On the other hand, if $p = 0$, C is minimized when $\lambda_2 = 0$, therefore no files should be prefetched. For $0 < p < 1$, we consider the following. From equation (10), we take the derivative of C with respect to λ_2, to obtain

$$\frac{dC}{d\lambda_2} = \frac{s}{\lambda}[(1-p)\alpha_B + \frac{\alpha_T(\lambda s - pb)}{(b-(\lambda+(1-p)\lambda_2)s)^2}] \tag{11}$$

Differentiating (11) again, we get

$$\frac{d^2C}{d^2\lambda_2} = \frac{2s^2}{\lambda}[\frac{\alpha_T(1-p)(\lambda s - pb)}{(b-(\lambda+(1-p)\lambda_2)s)^3}] \tag{12}$$

In a stable system, $(\lambda+(1-p)\lambda_2)s = (\lambda_1+\lambda_2)s$ must be less than b. Therefore equation (12) shows that for $pb > \lambda s$ ($0 < p \leq 1$), $\frac{d^2C}{d^2\lambda_2}$ is always less than zero. This implies that equation (10) is maximized at $\frac{dC}{d\lambda_2} = 0$ when $pb > \lambda s$. Solving $\frac{dC}{d\lambda_2} = 0$ for λ_2, we obtain the critical value λ_2', which is given by

$$\lambda_2' = \frac{1}{(1-p)s}\left(b - \lambda s - \sqrt{\frac{(pb-\lambda s)\alpha_T}{(1-p)\alpha_B}}\right) \tag{13}$$

Since function (10) is maximized at λ_2', where $\frac{dC}{d\lambda_2} = 0$ for $pb > \lambda s$, it follows that the cost decreases as λ_2 increases for $\lambda_2 > \lambda_2'$. Specifically, if $\lambda_2' \leq 0$ for the given p, λ, and $\frac{\alpha_T}{\alpha_B}$, then for any λ_2 in the range $[0, \frac{\lambda}{p}]$, the higher the λ_2, ie. the more that files with access probability p are prefetched, the lower the cost is. Thus, for the given p, λ, and $\frac{\alpha_T}{\alpha_B}$, if $\lambda_2' < 0$, prefetching *all* the files with access probability p will minimize the cost.

Notice that we have assumed the access probability of each file in the system is either p or 0. In addition, we assumed that files with access probability 0 may actually be requested, which implies that the rate at which files with access probability p appear in the system can be any value between 0 and $\frac{\lambda}{p}$. For example, if files with access probability zero are never requested by the user, then λ_2 is equal to $\frac{\lambda}{p}$. If all the files have access probabilities 0, then $\lambda_2 = 0$, because we never prefetch the files with access probability 0. Therefore, our conclusion may be stated as follows

For given p, λ, and $\frac{\alpha_T}{\alpha_B}$, independent of the rate at which files with access probability p appear in the system, if $\lambda_2' \leq 0$, then prefetching all the files with access probability p minimizes the cost.

The number of files that are actually prefetched is determined by the rate at which files with access probability p appear in the system, we do not need to know that to minimize the cost.

For given p, λ, and $\frac{\alpha_T}{\alpha_B}$, if $\lambda_2' \geq \frac{\lambda}{p}$, the cost increases as λ_2 increases for $\lambda_2 < \frac{\lambda}{p}$, therefore no files should be prefetched. If $0 < \lambda_2' < \frac{\lambda}{p}$, since the access probabilities of files in the system vary for different clients and with time in a real system, it is very hard to determine if λ_2 would be greater than λ_2' by prefetching all the files with access probability p. Therefore the lower cost can not be guaranteed at all times, and we choose not to prefetch any file in this case.

The above result is for the case when $pb > \lambda s$. If $pb \leq \lambda s$, (11) shows that $\frac{dC}{d\lambda_2} > 0$ for all λ_2, which means that the cost increases monotonically as λ_2 increases, therefore no files should be prefetched.

c) The prefetch threshold H.

Let us now find the prefetch threshold H for the system with $pb > \lambda s$ such that for $p \geq H$, $\lambda_2' \leq 0$ and the cost is minimized by prefetching all the files with access probabilities p. From equation (13), we obtain that $\lambda_2' \leq 0$, i.e. (10) is maximized at $\lambda_2 < 0$, if and only if

$$p \geq 1 - \frac{(1-\rho)\frac{\alpha_T}{\alpha_B}}{(1-\rho)^2 b + \frac{\alpha_T}{\alpha_B}} \tag{14}$$

where $\rho = \frac{\lambda \cdot s}{b}$. We then set the prefetch threshold to be

$$H = 1 - \frac{(1-\rho)\frac{\alpha_T}{\alpha_B}}{(1-\rho)^2 b + \frac{\alpha_T}{\alpha_B}} \tag{15}$$

It's easy to prove that H is always greater than ρ. Thus if $p > H$, then $p > \rho$, ie. $pb > \lambda s$. Therefore, equation (14) indicates that if the access probability p is greater than or equal to the threshold H, then $\lambda_2' \leq 0$ according to (13). Moreover, following our previous analysis this implies that prefetching all the files with access probability p minimizes the cost for $p \geq H$. The threshold H is plotted in figure 5 as a function of system utilization ρ for several different values of r $(= \frac{\alpha_T}{\alpha_B})$.

Figure 5 shows that as system load ρ increases, the prefetch threshold tends to increase as well, which means that fewer files should be prefetched. But the increase is not monotonic

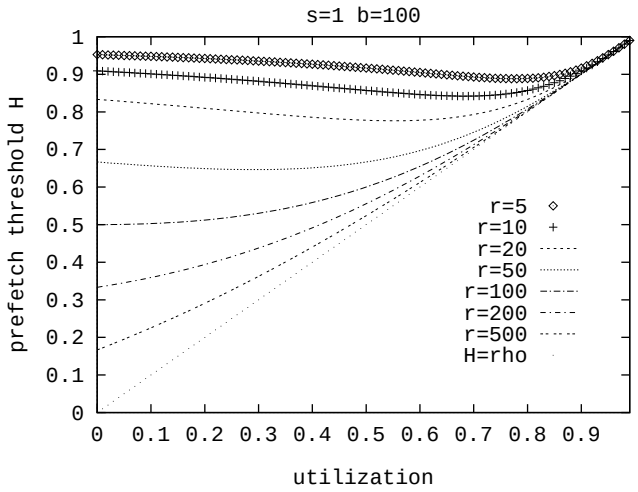

Figure 5: *Prefetch threshold H as a function of utilization ρ for different values of r.* ($r = \frac{\alpha_T}{\alpha_B}, s = 1, b = 100$)

for small values of $\frac{\alpha_T}{\alpha_B}$. The reason for this is, in those cases, when the load is low, prefetching does not save much time. As the load increases, it takes longer to transmit files and prefetch can save more time, therefore the threshold decreases. As the load continues to increase, prefetching files will add relatively long delay to the normal user requests, so the threshold needs to be increased again. We can prove that the threshold may decrease as ρ increases from zero, if and only if $b > \frac{\alpha_T}{\alpha_B}$; and for fixed $\frac{\alpha_T}{\alpha_B}$, the threshold is minimized at $\rho = 1-\sqrt{\frac{\alpha_T}{\alpha_B.b}}$. Furthermore, by comparing the prefetch threshold at $\rho = 0$ and at $\rho = 1-\sqrt{\frac{\alpha_T}{\alpha_B.b}}$, we can show that for any $b > \frac{\alpha_T}{\alpha_B}$ the maximum drop in the prefetch threshold value, $H_{\rho=0} - H_{\rho=1-\sqrt{\frac{\alpha_T}{\alpha_B.b}}}$, is 0.0674.

In both single and multi-user systems, for fixed $\frac{\alpha_T}{\alpha_B}$, the higher the system capacity, the higher the prefetch threshold, because the savings of time is less in a faster system. However, only in the multi-user system, is the prefetch threshold affected by system load.

d) The upper bound for the prefetch thresholds in systems with arbitrary access probability distribution.

Up to now, we have been studying the system in which, at any time, the access probability of each file is either p or 0, where p is fixed within one system. The following theorem shows that for an arbitrary distribution of access probabilities, where the access probability of each file in the system can be any value between 0 and 1, expression (15) gives an upper bound for the optimum prefetch threshold of this system. In other words, for any system, regardless

of the actual distribution of the access probabilities, the cost can always be reduced by prefetching files whose access probability is greater than the system's prefetch threshold computed from (15).

Theorem 1 *Consider a system in which b, s, λ, and $\frac{\alpha_T}{\alpha_B}$ are known. Let $f(q)$ be the rate at which files with access probability q appear in the system, where $\int_0^1 qf(q)dq = \lambda$ and $0 < q \leq 1$. Expression (15),*

$$H = 1 - \frac{(1-\rho)\frac{\alpha_T}{\alpha_B}}{(1-\rho)^2 b + \frac{\alpha_T}{\alpha_B}}$$

is an upper bound for the optimum prefetch threshold of this system.

This theorem can be proved based on our previous result. The details of the proof are given in appendix A. Unlike the system we studied previously, we assume that in this general system, files with access probability zero are never requested, hence we do not care about the value of $f(0)$. As an example, we examine the case when $f(q) = \frac{\lambda}{q}$ next.

Studies show that a small percentage of the files available on a given server usually account for a large portion of accesses to the server[3, 5]. This implies that, in general, a small number of files in the system have large access probabilities, while most of the files have small access probabilities. To model this, we choose a simple function for $f(q)$, namely we assume that the rate at which files with access probability q appear in the system is given by $f(q) = \frac{\lambda}{q}$, where $q \in (0, 1]$ and λ is the arrival rate of the user requests.

In this example system, if we set the prefetch threshold as H $(0 < H \leq 1)$ and prefetch all the files with access probability greater than or equal to H, then $\lambda_2 = \int_H^1 \frac{\lambda}{q} dq = -\lambda ln(H)$. Among them, $\int_H^1 q\frac{\lambda}{q}dq = (1-H)\lambda$ are eventually used on average, hence $\lambda_1 = H\lambda$. Similar to our previous analysis, we compute the average cost of each user request as

$$C = \frac{\lambda_1 c_1 + \lambda_2 c_2}{\lambda} = \alpha_B s[H - ln(H)] + \frac{\alpha_T \cdot s \cdot H}{b - [H - ln(H)]\lambda} \tag{16}$$

Equation (16) is plotted in figure 6 as a function of the prefetch threshold H for different values of user request arrival rate λ. Taking the derivative of equation (16) with respect to H yields that

$$\frac{dC}{dH} = \alpha_B \cdot s(1 - \frac{1}{H}) + \frac{\alpha_T \cdot s}{b - [H - ln(H)]\lambda} + \frac{p \cdot \alpha_T \cdot s \cdot \lambda(1 - 1/H)}{[b - (H - ln(H))\lambda]^2}$$

Figure 6 shows that the cost is minimized at $\frac{dC}{dH} = 0$. Let us evaluate $\frac{dC}{dH} = 0$ for several system loads and $\frac{\alpha_T}{\alpha_B}$ to determine the optimum value of H in each case such that the cost is

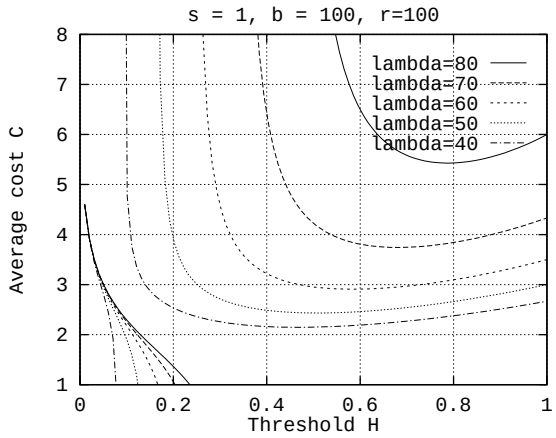

Figure 6: *Average cost C as a function of prefetch threshold H for different values of λ.* $(s = 1, b = 100, r = \frac{\alpha_T}{\alpha_B} = 100)$

minimized. The results are compared with the upper bound obtained using (15) in figure 7, where the upper bounds are shown as smooth lines and the results for $f(q) = \frac{\lambda}{q}$ are shown as points connected with lines.

e) The prefetch threshold for the system with nonzero start up delay.

So far, we have been studying the system with zero startup time. For the system with a startup time t_0 greater than zero, we can prove in a similar way that the prefetch threshold should be set to the following

$$H = 1 - \frac{(1-\rho)(1+\frac{s_0}{s})\frac{\alpha_T}{\alpha_B}}{(1-\rho)^2 b + (1+\frac{s_0}{s})\frac{\alpha_T}{\alpha_B}} \tag{17}$$

where $s_0 = t_0 \cdot b(1-\rho)$.

In summary, the prefetch threshold module works in the following way. For given λ and $\frac{\alpha_T}{\alpha_B}$, and a file with access probability $p_i > 0$, compute H using equation (17). The file is prefetched if and only if p_i is greater than or equal to the H obtained.

f) The prefetch threshold for more general systems.

Up to now, we have been assuming that all users in the system have the same α_B and α_T. In the real world, people value their time differently; therefore, α_T can vary from user to user. Since generally users who are sharing the same network are supported by the same network carrier company, we can assume that α_B is the same throughout the system. In addition, after some users have started prefetching files, others can't use (15) to determine the prefetch thresholds. This is because the distribution of access probabilities in the system

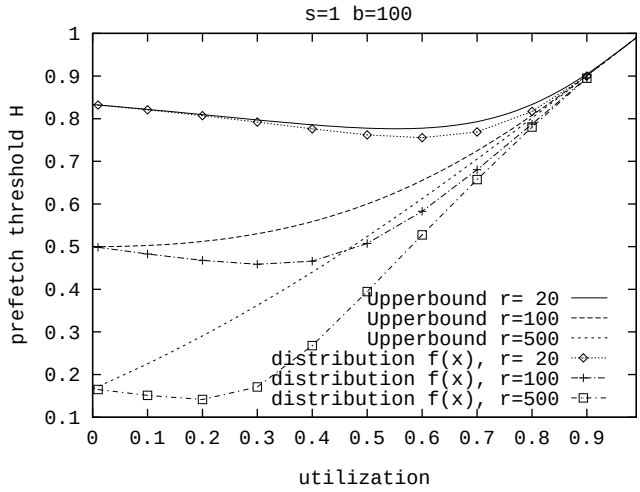

Figure 7: *The prefetch threshold upper bound and the prefetch threshold obtained for the system with* $f(q) = \frac{\lambda}{q}$. $(s = 1, b = 100, r = \frac{\alpha_T}{\alpha_B})$

is very complex, and a user may be unable to find the user request rate λ, from the current λ_1 and λ_2. In the following, we study the system with users who have different α_T's and in which some users have started prefetching files. Again, we assume the startup time is zero.

Assume there are n users in the system and the cost of a time unit for user i is α_{T_i}. In addition, we assume that user i issues requests at rate λ_i. Accordingly, the rate of normal requests from user i is λ_{i_1}, and the rate of prefetch requests from the same user's prefetch program is λ_{i_2}, where λ_{i_2} could be zero. We further assume that the arrival of requests to the server is a Poisson process. As in the previous system, the cost of a user request is the sum of the system resource cost and the time cost, and is given by

$$C = \frac{s}{\sum_{i=1}^{n} \lambda_i}[\sum_{i=1}^{n}(\lambda_{i_1} + \lambda_{i_2})\alpha_B + \frac{\sum_{i=1}^{n} \lambda_{i_1}\alpha_{T_i}}{b - \sum_{i=1}^{n}(\lambda_{i_1} + \lambda_{i_2})s}] \tag{18}$$

Assume that for each user in the system except user k, the current prefetch request rate λ_{i_2} is known, and $\lambda_{k_2} = 0$. We want to determine the prefetch threshold for user k, H_k, such that the cost C decreases if user k prefetches the files with access probabilities greater than or equal to H_k. Similar to the method we used before, we take the derivative of the cost function twice. For the case where $\frac{\partial^2 C}{\partial^2 \lambda_{k_2}} < 0$ for all possible values of λ_{k_2}, we find the critical value λ'_{k_2} which makes $\frac{\partial C}{\partial \lambda_{k_2}} = 0$. When $\lambda'_{k_2} < 0$, we can compute the prefetch threshold H_k, which is given by

$$H_k = 1 - \frac{\alpha_{T_k}[b - (\lambda_k + \sum_{i\neq k}^{n}(\lambda_{i_1} + \lambda_{i_2}))s]}{\alpha_B[b - (\lambda_k + \sum_{i\neq k}^{n}(\lambda_{i_1} + \lambda_{i_2}))s]^2 + s\sum_{i\neq k}^{n} \lambda_{i_1}\alpha_{T_i} + \alpha_{T_k}[b - \sum_{i\neq k}^{n}(\lambda_{i_1} + \lambda_{i_2})s]} \tag{19}$$

Since user k has not prefetched any file, we have that $\lambda_{k_1} = \lambda_k$ and $\frac{\sum_1^n(\lambda_{i_1}+\lambda_{i_2})}{b} = \rho$, and then (19) is equivalent to

$$\begin{aligned} H_k &= 1 - \frac{\alpha_{T_k}[b - \sum_{i=1}^n(\lambda_{i_1} + \lambda_{i_2})s]}{\alpha_B[b - \sum_{i=1}^n(\lambda_{i_1} + \lambda_{i_2})s]^2 + s\sum_{i=1}^n \lambda_{i_1}\alpha_{T_i} + \alpha_{T_k}[b - \sum_{i=1}^n(\lambda_{i_1} + \lambda_{i_2})s]} \\ &= 1 - \frac{(1-\rho)\frac{\alpha_{T_k}}{\alpha_B}}{(1-\rho)^2 b + \frac{\alpha_{T_k}}{\alpha_B}(1-\rho) + \frac{s}{b\alpha_B}\sum_{i=1}^n \lambda_{i_1}\alpha_{T_i}} \end{aligned} \tag{20}$$

Similar to the proof for the previous system, we can show that for fixed λ_{i_2}, $(1 \leq i \leq n, i \neq k)$, if user k prefetches all the files with access probability greater than H_k, the cost C is minimized. As the system changes, each user recomputes his prefetch threshold using (20) based on the new network conditions. In this way, the prefetch thresholds adapt to a changing environment. This algorithm is greedy in nature. And if the system is stable, it can achieve its locally optimal state.

4 Simulation results

In the last two sections, we studied the algorithms in the prediction module and the threshold module. We have also presented a prediction algorithm for web browsing which could be run on both the server and client. An important question regarding this prediction algorithm is how the access probabilities from the server and client should be used, and how this affects the system performance. In this section, we show the simulation results obtained from several approaches.

The simulations were driven by the UCLA Computer Science (CS) department web server's log file taken during the period from April 11th to June 5th of 1996. It consists of more than 300,000 actual user accesses. In the simulations, we concentrated on two representative pages: the CS department home page and the home page for TAs (teaching assistants). More precisely, only when one of these two pages was being viewed, was the prediction algorithm activated and the program simulated to prefetch files linked to the page. And only when a file linked to one of these two pages was requested, did the program check if it had been prefetched. The CS department home page contains links to research, students, faculty, and other general information home pages. These links are not updated very often. The TA home page has links to all the class home pages which are updated frequently to include new assignments, handouts, etc. Therefore, simulations on these two pages indicate how well our prediction algorithm works on pages that are revisited frequently by some particular users and on those that are generally not revisited frequently.

To study how the access probabilities from the server and the client affect the performance of prediction, we tested three cases. 1) Using only the access probabilities from the client site. 2) Using only the access probabilities from the server site. 3) Merging the access probabilities from server and client as designed in our original algorithm. Namely, assume A is the current page; if $C_A < 5$, then use the access probabilities from the server for the links on page A, otherwise use the access probabilities from the client. A fixed prefetch threshold was used in each simulation run and varied for different runs. All the files with access probability greater than or equal to the prefetch threshold were prefetched.

We measured several different system parameters. If the user requests a page and it is satisfied by a prefetched copy of the file, then we call it a *hit*. The *hit rate* is defined as the ratio of the number of hits over the total number of user requests. For a fixed number of user requests, the higher the hit rate, the more time is saved by prefetching. The *successful prediction rate* is the probability of a prefetched file being eventually used. A high successful prediction rate indicates that less bandwidth is wasted due to prefetching unused files. Figures 8 and 9 show the successful prediction rate and hit rate obtained for the pages linked to the CS home page and the TA home page respectively with prefetch thresholds from $0.01 - 0.9$ for the three cases described above.

As shown in figure 8, for both home pages, using only access probabilities on the client site yields successful prediction rates of around 70% when the threshold is just slightly higher than zero. As the threshold increases, the successful prediction rate tends to increase. In figure 8a, the tails of the top two curves drop because there were very few files prefetched at those threshold values, and the sample size was very small. Each curve ends at the point where no file is prefetched for larger prefetch thresholds. Although the successful prediction rates are lower when the access probabilities from the server were used, the result is good given that these probabilities are based on the accesses from hundreds of other clients who are totally unrelated to the user. Without the access probabilities from the server, no prefetch is possible when the user visits a page for the first few times. This is further supported by the hit rate distribution shown in figure 9.

From figure 9 we can see that, if the prediction algorithm is run only on the client site, the hit rate is not as good as the successful prediction rate. The reason for this is, in many cases, the user had not visited a page enough times and no file was prefetched from the page. On the other hand, by using only access probabilities from the server or merging those from server and client, we obtain much higher hit rates. In addition, for small prefetch thresholds, these two curves are very close because prefetches invoked due to high access probabilities

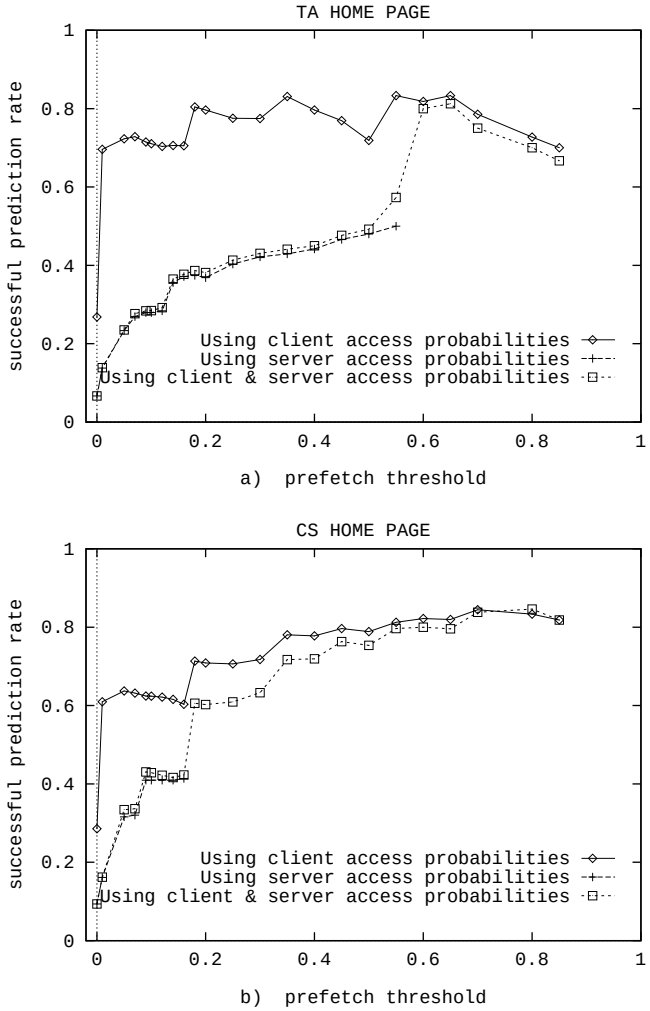

Figure 8: *Successful prediction rate vs prefetch threshold. a) TA home page, b) CS home page.*

at the client site only contribute a small portion of the total number of prefetches.

In figure 9, the hit rate of the pages linked to TA home page is lower than 40% when the prefetch threshold is zero. This is because, although the TA home page is the main place where requests to class home pages are generated, some students access a class home page using bookmarks or from pages other than the TA home page, for example the instructor's home page. In those cases no prefetch was done in our simulation since we did not apply the prefetch algorithm to pages other than the TA and CS home pages. The same problem exists for pages linked to the CS home page, but it is less significant. For the same reason, the hit rate in the real system should be higher than what is shown in figure 9.

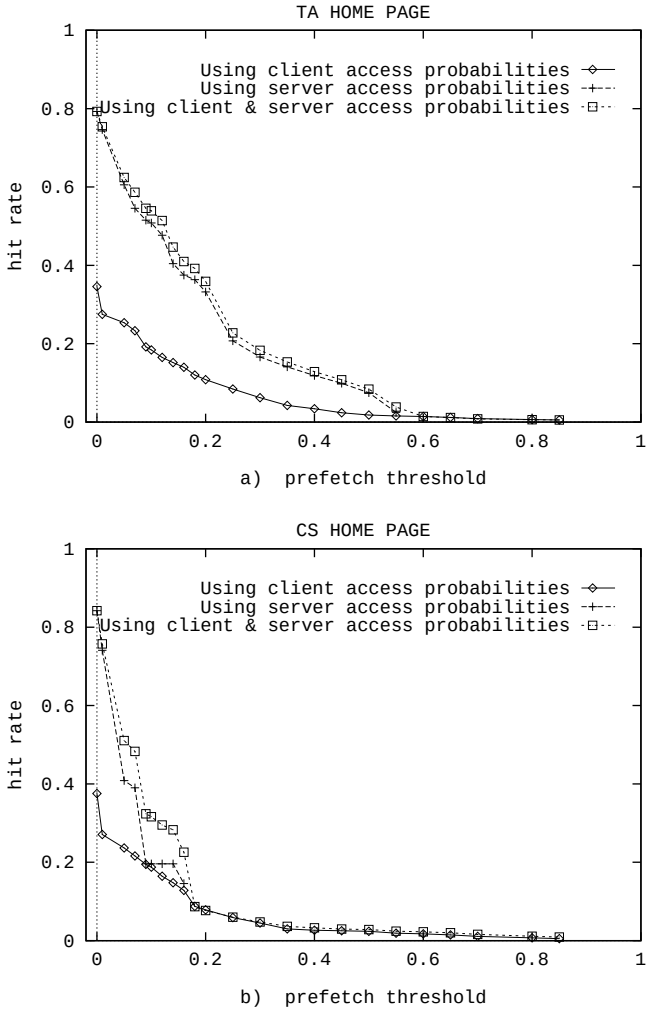

Figure 9: *Hit rate vs prefetch threshold. a) TA home page, b) CS home page.*

Figure 10 shows the relation between the prefetch threshold and the average number of files prefetched for each user request, which indicates the amount of traffic increase in the entire system due to prefetching. When only the access probabilities from the client was used, this average number is very low, since the prediction was very accurate and if the user had not visited the page at least 5 times, no link on it was prefetched. More files were prefetched for the other two cases, but still the average number of files prefetched for each user request is low. Comparing figure 9 and 10, we can see that for a hit rate of more than 60%, the average number of files prefetched for each user request is less than 2.

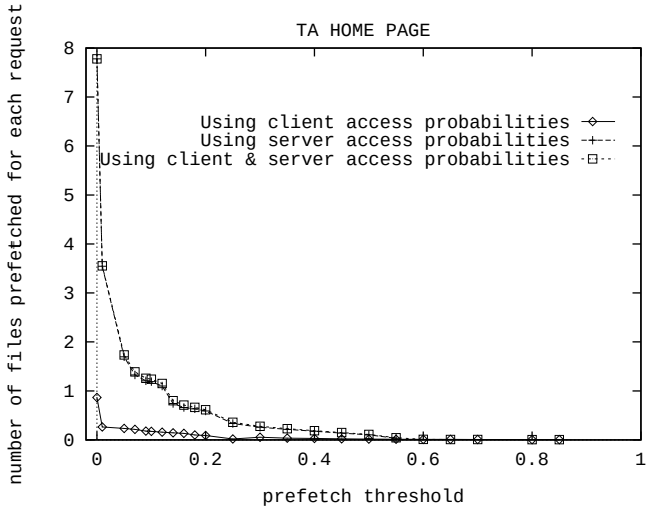

Figure 10: *Average number of files prefetched for each user request.* (*TA home page*)

5 Conclusion

In this paper, we have presented an adaptive network prefetch scheme, which is comprised of a prediction module and a threshold module. To achieve high efficiency with prefetching, in the prediction module, we must devise an algorithm to predict the access probability of each file as accurately as possible. One such prediction algorithm for web browsing has been discussed. In the threshold module, the prefetch threshold for each related server is computed. We have derived an upper bound for the prefetch threshold which is a function of system load, capacity, and cost of a time unit and a system resource unit to the user. Files with access probability greater than its server's prefetch threshold are prefetched. We showed that, by doing so, a lower average cost can always be achieved.

We also studied the performance of our prediction algorithm for web browsing by setting the prefetch threshold at various values in the simulation. The results show that generally, using the access probabilities from the client can ensure that a large portion of prefetched files are used, while the access probabilities from the server can help improve the number of user requests satisfied by the prefetched files.

In summary, we believe that prefetch is a good approach to reduce latency for network applications. However it must be implemented in a way such that the overall system performance can be improved, and the tradeoff between saving user's time and the bandwidth usage must be considered. We think the significance of this issue goes beyond the prefetch problem. For example, in the case of push server or push caching[3, 5, 11], it is important

to study its efficiency, this means not only the percentage of the pushed files being used, but also and more importantly, the impact of pushing to other users who are sharing the same network but may or may not be able to use these pushed files at all. As in most of the other prefetch algorithms, most commercial push servers use fixed (possibly customized by the user) values to determine what and when to push, without taking into account the network conditions at the time. This will almost certainly not achieve the best efficiency for the overall system. Clearly, the solution to this problem is adaptivity - adaptive to the user's behavior and adaptive to the network conditions. This is also what we are trying to show in our prefetch scheme. In fact, we can use the same kind of approach for the threshold algorithm to analyze the push server or push caching problem. As more functions like prefetching or pushing are added to the network applications, it is very important to consider the tradeoffs involved as we have shown here.

References

[1] A. Agarwal, *Analysis of Cache Performance for Operating Systems and Multiprogramming,* Kluwer Academic Publishers, Boston, 1989.

[2] A. Bestavros, "Speculative Data Dissemination and Service to Reduce Server Load, Network Traffic and Service Time for Distributed Information Systems", *Proceedings of ICDE'96: The 1996 International Conference on Data Engineering,* New Orleans, Louisiana, March 1996.

[3] A. Bestavros, "WWW Traffic Reduction and Load Balancing Through Server-Based Caching", *IEEE Concurrency: Special Issue on Parallel and Distributed Technology,* vol. 5, Jan-Mar 1997, pp. 56-67.

[4] J. Griffioen, R. Appleton, "Reducing File System Latency using a Predictive Approach", *Proceedings of the Summer 1994 USENIX Conference,* Boston, Massachusetts, June 1994, pp. 197-207.

[5] J.S Gwertzman, "Autonomous Replication Across Wide-area Internetworks", Thesis, Harvard College, Cambridge, Massachusetts, 1995.

[6] L. Kleinrock, *Queueing Systems Vol 1: Theory,* John Wiley & Sons, New York, NY, 1975.

[7] L. Kleinrock, *Queueing Systems Vol 2: Computer Applications,* John Wiley & Sons, New York, NY, 1975.

[8] E. Markatos, C. Chronaki, "A Top-10 Approach to Prefetching on the Web", Technical Report 173, ICS-FORTH, http://www.ics.forth.gr/proj/arch-vlsi/www.html, August 1996.

[9] M. Mroz, "A Client Based Prefetching Implementation for WWW", MS dissertation, Dep. Comp. Sci. Univ. of Boston, 1995.

[10] V.N. Padmanabhan, J.C. Mogul, "Using Predictive Prefetching to Improve World Wide Web Latency", *Computer Communication Review*, July 1996, pp. 22-36.

[11] T. Spangler, "Push Servers Review", *PC Magazine*, June 10, 1997, pp. 156-180.

[12] *PeakJet Software Homepage,* http://www.peak-media.com/PeakJet/PeakJet.html.

A Proof for the threshold upper bound.

We now prove theorem 1.

Theorem 1 *Consider a system in which b, s, λ, and $\frac{\alpha_T}{\alpha_B}$ are known. Let $f(q)$ be the rate at which files with access probability q appear in the system, where $\int_0^1 qf(q)dq = \lambda$ and $0 < q \leq 1$. Expression (15)*

$$H = 1 - \frac{(1-\rho)\frac{\alpha_T}{\alpha_B}}{(1-\rho)^2 b + \frac{\alpha_T}{\alpha_B}}$$

is an upper bound for the optimum prefetch threshold of this system.

Proof:

We assume $f(q)$ is continuous. The proof is similar for $f(q)$ discrete.

Since $f(q)$ is the rate at which files with access probability q appear in the system, for any h $(0 < h \leq 1)$, if all and only those files with access probability greater than or equal to h are prefetched, then the arrival rate of the prefetch requests is

$$\lambda_2 = \int_h^1 f(q)dq \tag{21}$$

hence, the normal requests arrive with rate

$$\lambda_1 = \lambda - \int_h^1 qf(q)dq \tag{22}$$

When the prefetch request rate is λ_2, the average cost of each user request is given by

$$C(\lambda_2) = \frac{\lambda_1 c_1 + \lambda_2 c_2}{\lambda} \tag{23}$$

where, the same as in our previous analysis,

$$c_1 = \alpha_B \cdot s + \alpha_T \cdot \frac{s}{b - (\lambda_1 + \lambda_2) \cdot s}$$

$$c_2 = \alpha_B \cdot s$$

Assume that, by plugging b, s, λ, and $\frac{\alpha_T}{\alpha_B}$ into (15), we obtain a value H'. Now we consider another system with the same b, s, λ, and $\frac{\alpha_T}{\alpha_B}$. When the user is using a file, for each file in the new system, we set its access probability to H' if and only if it is at least H' when the same file is being used in the original system, otherwise we set it to 0. Since two system have the same b, s, λ and $\frac{\alpha_T}{\alpha_B}$, the prefetch threshold for the new system is H' by (15). This implies that the average cost of a user request can be minimized by prefetching all the files with access probability H'. In this new system, let $\overline{\lambda_2}$ be the corresponding prefetch request rate at which the cost is minimized. By our definition of the new system and the result in section 3.2, it is given by

$$\overline{\lambda_2} = \int_{H'}^{1} f(q)dq$$

Let $C_{new}(\lambda_2)$ be the average cost of viewing a page when the prefetch rate is λ_2. The analysis in section 3.2 shows that

$$\frac{dC_{new}(\lambda_2)}{d\lambda_2} = \frac{s}{\lambda}[(1 - H')\alpha_B + \frac{\alpha_T(\lambda s - H'b)}{[b - (\lambda + (1 - H')\lambda_2)s]^2}] \leq 0 \tag{24}$$

for $0 \leq \lambda_2 \leq \overline{\lambda_2}$.

Based on equations (21) and (22), we take the derivative of equation (23) with respect to λ_2 to obtain that

$$\frac{dC(\lambda_2)}{d\lambda_2} = \frac{s}{\lambda}[(1 - h)\alpha_B + \frac{\alpha_T[(\lambda_2 h + \lambda_1)s - hb]}{[b - (\lambda_1 + \lambda_2)s]^2}] \tag{25}$$

Comparing equation (24) and (25) term wise yields that

$$\frac{dC(\lambda_2)}{d\lambda_2} \leq \frac{dC_{new}(\lambda_2)}{d\lambda_2} \leq 0$$

for $0 \leq \lambda_2 \leq \overline{\lambda_2}$.

Since for $0 \leq \lambda_2 \leq \overline{\lambda_2}$, $H' \leq h \leq 1$. This implies that for $H' \leq h \leq 1$, the average cost of viewing a file decreases as h decreases. Therefore the prefetch threshold which is going to minimize the cost in the original system must be less than or equal to H'. In other words, H' obtained from (15) is the upper bound of the prefetch thresholds for the general systems with the same b, s, λ, and $\frac{\alpha_T}{\alpha_B}$. This completes the proof. □