Prefetching Links on the WWW

Zhimei Jiang Leonard Kleinrock *
Department of Computer Science
University of California at Los Angeles
Los Angeles, CA 90024

Abstract

In this paper, we study prefetch techniques in the WWW, in which we predict which files will be needed in the near future and download some of them before they are requested by the user. Our prefetch scheme includes two algorithms: the prediction algorithm and the threshold algorithm. The prediction algorithm estimates the probability with which each file will be requested in the near future. The threshold algorithm computes the prefetch threshold for each server. An important contribution of this paper is a formula we derived to determine the prefetch threshold dynamically based on the system load, capacity and the cost of time and system resources to the user. Simulations driven by trace files show that using access information from the client can achieve high successful prediction rates, while using that from the server can result in more hits in general. We have also developed a prefetch program at the client site which assists users in browsing faster and more efficiently.

1 Introduction

Due to the proliferation of the World Wide Web (WWW), there has been a large increase in information transmitted over the Internet. This includes not only text and images but video and audio as well. The growth of internet capacity is not keeping pace and often users experience long delays in retrieving files from Web servers. Sometimes, the waiting does not end even after the file has made it to the local disk. For example, after an access, the client machine may need to decompress files, compile Java programs, etc. However while the user is viewing a page, the local machine is generally idle. These idle periods can be used to reduce the user's future waiting times. One way of doing this is to *prefetch* the files, namely to fetch the files that are likely to be accessed before the user requests for them. More specifically, some intelligence is added to web browsers and servers such that whenever a new page is displayed they can estimate which files (i.e. links) will be needed in the next few accesses and then choose some of them to transmit to the local disk beforehand according to certain criteria. If a prefetched file is indeed requested, the user can access it with negligible delay. In addition, prefetching allows more time for sophisticated processing including encryption, compression and compilation to be carried out at both the server and the client sites, so that data can be transmitted more efficiently and securely.

If we knew exactly what a user would need next, we would retrieve only those files in advance. Assuming that the prefetch operation always finishes before the user goes to the next page, then we could enjoy zero latency with no extra bandwidth requirement. Unfortunately, in the real world, some prefetched files may never be used, resulting in wasted bandwidth and increased delay to normal (nonprefetching) requests. More seriously, prefetching too many files could be disastrous to the network. Therefore, the problem to be solved is how to choose the files to prefetch such that the gain from the decreased latency by viewing prefetched files minus the loss suffered from prefetching those files which are never used, is maximized. We propose one solution in this paper.

Basically, our prefetch scheme includes two algorithms: the prediction algorithm, and the threshold algorithm. The prediction algorithm collects access history information and estimates the probability of each file being accessed in the near future by the user. The threshold algorithm computes a prefetch threshold for each server. A file is prefetched if and only if its access probability exceeds its server's prefetch threshold. By prefetching a file, we mean that the file is downloaded if no up-to-date version of the file is available on the local disk; otherwise no action will be taken. Details of these two algorithms are covered in the next two sections. We also studied the efficiency of our prediction algorithm through trace-driven simulations. The results are summarized in section 4. In addition, we developed a prefetch program running on top of the Netscape browser. The program, which is presented in section 5, also includes some interesting functions which can help the user browse more efficiently. We conclude in section 6.

2 The prediction algorithm

The task of the prediction algorithm is to keep track of the user/server access history and compute the access probabilities in order to determine which files to prefetch. The *access probability* $p(B|A)$ is defined as the conditional probability that file B will be requested by the user given that file A is being viewed. Our prediction algorithm is based on the fact that typically web surfing involves a sequence of clicking and downloading operations. Namely, first the user clicks on

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

a link to download a new page. After viewing the new page, the user usually clicks on a link on this page to download another new page and so on. Sometimes, the user may click the *'back'* button on the browser to go back to some previous page and then go to 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 assume $p(B|A)$ is 0 if no link exists on page A to page B. In our prefetch scheme, the prediction algorithm is required on the client site. The same algorithm can also be implemented on the server site to help the client generating the access probabilities.

Both [1] and [6] studied the prediction algorithm at the server site. Parameters similar to the access probability are defined in these references. In both algorithms, a window is applied to the access history, 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 hard to adjust the size of the time window. In the WWW, after an html file is downloaded, several images embedded in this file are downloaded immediately. The request window includes both html files and embedded images, this makes it hard to determine the window size as well. Our scheme is similar to using a request window of size two after removing all the requests that are not sent explicitly by the user, for example embedded images, etc., from the request sequence.

Our prediction algorithm requires each client to keep track of its own access history by maintaining two kinds of counters, the file counters and the link counters. Each file A is associated with a file counter C_A. In addition, if file B can be accessed directly from file A, namely there exists a link on file A pointing to file B, then there is a link counter $C_{(A,B)}$ for the pair. There are no other counters. Initially, all counters are set to 0. Whenever file A is downloaded, C_A is increased by one. In addition, if file B is accessed by clicking on the corresponding link on the previous file A, counter $C_{(A,B)}$ is also increased by one. These counters are used to predict the user behavior in the following way. Whenever file A is being viewed, the access probability of file B is given by $min(1, \frac{C_{(A,B)}}{C_A})$ for $C_A \geq 5$, and 0 for $C_A < 5$. When file A is not being viewed, $P\{B|A\}$ is always set to 0. Notice that for a page A with k distinct links, $\sum_{i=1}^{k} P\{B_i|A\}$ can be greater than one. This is because a user may click link i_1 on page A and come *'back'* later to go to another link i_2 through this page. Therefore while page A is retrieved from the server only once, more than one file might be accessed from page A. The values of these counters are based on the access history during a certain period of time, for example the last 30 days.

The access probabilities at the client site show the user's personal interest. The same prediction algorithm can also run on the server to keep track of the popularity of the files over all the users that have accessed that server. If a client hasn't visited a page often enough to obtain reliable access probabilities, the server's access probabilities are more likely to be accurate. Access probabilities from the server and client are merged in the following way at the client site. If a page has been visited less than 5 times by the client, and its access probability is available at the server then the server's probability is used. Otherwise, access probability from the client is used. More sophisticated algorithms may be developed to do the merging. We have implemented this algorithm in the program presented in section 5.

3 The threshold algorithm

Prefetching is a scheme which takes advantage of the tradeoff between bandwidth usage and latency by predicting which files will be needed and downloading some of them so that these files can be viewed immediatedly when the user requests them later. We choose to measure this tradeoff in terms of cost which is comprised of both the *delay cost* (α_T\$/time unit) and the *system resource cost* (α_B\$/packet), where 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 browsing a page.

Previous work on prefetching uses a fixed threshold for all servers. The problem with a fixed threshold is that it does 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 the rest of this section, we study a prefetch system model and derive an upper bound for the prefetch threshold based on system capacity, load, and cost. Unless indicated otherwise, we assume that a prefetch operation always finishes 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 *. In the real implementation, we may encounter the problem of running out of the disk space, in which case some cached files must be deleted. We will not discuss this problem any further in this paper.

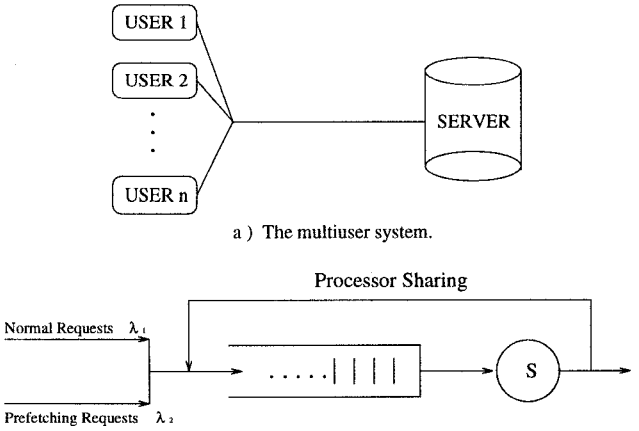
a) The system model.

The system we are going to study consists of a single WWW server and multiple users which share the WWW server and the network link to it as shown in figure 1a. Each user's system can prefetch files while the user is browsing. We model this prefetching system as an $M/G/1$ Round-Robin

*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 will be zero regardless whether the first request is 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.

processor-sharing system with two Poisson inputs which is shown in figure 1b, where the server in the model represents both the network and the WWW server in the real system. The server handles two kinds of requests: *normal requests* are those user requests which can not be satisfied by the prefetched files on the local disk, *prefetch requests* are those sent by the prefetch program. All requests are of the same priority and they join the same queue waiting for service. If a user request can be satisfied by a prefetched file, no request will be sent to the server. We further assume that requests which do not invoke a file transfer consume very little resources and are negligible.

Here we assume that the access probability of each link on every page is either exactly p ($p > 0$) or 0, where p is fixed within a system but can vary for different systems. Files with access probability 0 are never prefetched. This implies that all the prefetched files will be requested by the user with the same probability p. Now we begin to derive the prefetch threshold step by step.



a) The multiuser system.

b) The prefetch system model.

Figure 1: *Multiuser system and its model.*

b) The cost function C.

As we defined earlier, the total cost of a user request is the sum of the system resource cost of transmitting the file and the delay cost of waiting for the file to arrive. Therefore, the cost of a normal request is

$$c_1 = \alpha_B \cdot s + \alpha_T \cdot t \tag{1}$$

where s is the file size, t is transmission time. If the user requests for a file that had previously been prefetched and saved on the local cache, the cost associated with this request is only

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

because the delay cost is zero.

Let λ be the arrival rate of user requests when no prefetching is applied. We assume that prefetching does not affect the user's behavior regarding the likelihood of accessing links. Let the arrival rate of normal requests and prefetch requests be λ_1 and λ_2 respectively when prefetching is employed. Therefore, the rate at which user requests are satisfied by the prefetched files is $\lambda - \lambda_1$, which is simply $p\lambda_2$ by definition. Thus $\lambda_1 + p \cdot \lambda_2 = \lambda$ or

$$\lambda_1 + \lambda_2 = \lambda + (1-p)\lambda_2 \tag{3}$$

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

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

$$t = \frac{x}{1-\rho} = \frac{s}{b(1-\rho)} \tag{4}$$

where $\rho = \lambda x$ is the system load, s is the average file size, and b is the system capacity [2,3]. Plugging (4) into (1), we obtain that, the cost of a normal request is

$$c_1 = \alpha_B \cdot s + \alpha_T \cdot \frac{s}{b - (\lambda_1 + \lambda_2)s} \tag{5}$$

where $b > (\lambda_1 + \lambda_2)s$. Notice that in equation (5), the effect of prefetching to other users in the system is reflected through the λ_2 in the formula. As more files are prefetched, the cost of normal requests increases since prefetching increases the system load and the delay of retrieving files. The average cost of a prefetch request is given by (2).

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 WWW server as normal requests; by equations (2),(4) and (5), it follows that the average cost of an explicit user request is

$$\begin{aligned} C &= \frac{\lambda_1 \cdot c_1 + \lambda_2 \cdot c_2}{\lambda} \\ &= \frac{s}{\lambda}[(\lambda + (1-p)\lambda_2)\alpha_B + \frac{(\lambda - p\lambda_2)\alpha_T}{b - (\lambda + (1-p)\lambda_2)s}] \end{aligned} \tag{6}$$

This equation for average cost C is plotted in figure 2 as a function of λ_2 for different values of p. Note that for expression (6), the valid range of λ_2 is $0 \le \lambda_2 \le \frac{\lambda}{p}$ for $0 < p \le 1$.

c) The optimum value of the prefetch rate λ_2.

Assume p and λ are known, we wish to find the values of λ_1 and λ_2 which minimize the average cost of each user request in the system. From equation (6), clearly, at p=1 C is minimized when $\lambda_2 = \lambda$, i.e. if all the files have access probability 1, all of them should be prefetched. On the other hand, if $p = 0$, C is minimized when $\lambda_1 = \lambda$, therefore no files should be prefetched. For $0 < p < 1$, we consider the following.

From equation (6), we take the derivative of C with respect to λ_2, to obtain

$$\frac{dC}{d\lambda_2} = \frac{s}{\lambda}[(1-p)\alpha_B + \frac{\alpha_T(\lambda s - pb)}{(b - (\lambda + (1-p)\lambda_2)s)^2}] \tag{7}$$

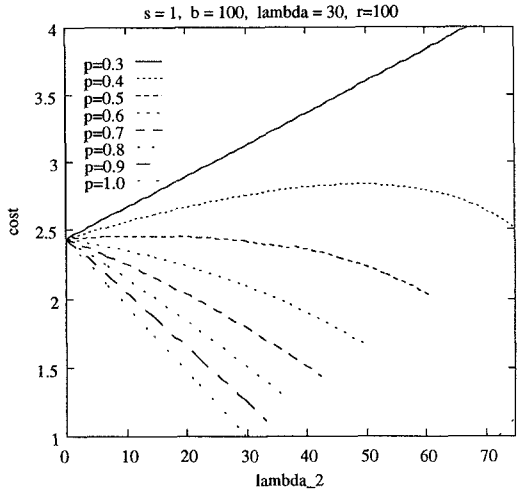


Figure 2: *Average cost of a user request as a function of* λ_2 *for* p *from* 0.3 *to* 1. $(s = 1, b = 100, \lambda = 30, r = \frac{\alpha_T}{\alpha_B} = 100)$

Differentiating again, we get

$$\frac{d^2C}{d^2\lambda_2} = \frac{2s^2}{\lambda}[\frac{\alpha_T(1-p)(\lambda s - pb)}{(b-(\lambda+(1-p)\lambda_2)s)^3}] \tag{8}$$

In a stable system, $(\lambda + (1-p)\lambda_2)s = (\lambda_1 + \lambda_2)s$ must be less than b. Therefore equation (8) shows that for $pb > \lambda s$ $(0 \leq p \leq 1)$, $\frac{d^2C}{d^2\lambda_2}$ is always less than zero. It follows that function (6) is maximized at $\frac{dC}{d\lambda_2} = 0$ on the half plane where $pb > \lambda s$. Solving $\frac{dC}{d\lambda_2} = 0$ for λ_2, we obtain the critical value λ_2'

$$\lambda_2' = \frac{1}{(1-p)s}[b - \lambda s - \sqrt{\frac{(pb-\lambda s)\alpha_T}{(1-p)\alpha_B}}] \tag{9}$$

Since function (6) is maximized at λ_2', where $\frac{dC}{d\lambda_2} = 0$, on the half plane $\rho b < \lambda s$. This implies that the cost decreases as λ_2 increases for $\lambda_2 > \lambda_2'$. Specifically, if $\lambda_2' \leq 0$ for a given p and λ, the more that files with access probability p are prefetched, the lower the cost is for any λ_2 in the range $[0, \frac{\lambda}{p}]$. Therefore, for a given p and λ, if $\lambda_2' < 0$, then prefetching all the files with access probability p will minimize the cost.

Notice that although we have assumed the access probability of each file on a page is either p or 0, we did not limit the exact number of links with access probability p on each page. For example, if the access probabilities of all the links on every page are all equal to p then λ_2 can be as large as $\frac{\lambda}{p}$. If all the files have access probabilities 0, then $\lambda_2 = 0$, because we never prefetch the files with access probability 0. Therefore, what we concluded in the last paragraph is that

> *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.*

But exactly how many files are prefetched is determined by the distribution of access probabilities, and we do not need to know that to minimize the cost.

If $\lambda_2' > \frac{\lambda}{p}$, the cost increases as λ_2 increases for $\lambda_2 < \frac{\lambda}{p}$, therefore no files should be prefetched.For $0 < \lambda_2' < \frac{\lambda}{p}$, because the distribution of access probabilities on a page varies for different clients and with time, 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 in this case we choose not to prefetch any file.

d) The prefetch threshold H.

Let us now find the prefetch threshold H such that the cost can be minimized by prefetching all the files with access probabilities p, for p greater than H. From equation (9), we can obtain that $\lambda_2' \leq 0$ if and only if

$$p \geq 1 - \frac{(1-\rho)\frac{\alpha_T}{\alpha_B}}{(1-\rho)^2 b + \frac{\alpha_T}{\alpha_B}} \tag{10}$$

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

$$H = 1 - \frac{(1-\rho)\frac{\alpha_T}{\alpha_B}}{(1-\rho)^2 b + \frac{\alpha_T}{\alpha_B}} \tag{11}$$

Expression (10) shows that if the access probability p is greater than or equal to the threshold H, then $\lambda_2' \leq 0$ according to (9). Moreover, following our previous analysis this implies that prefetching all the files with access probability p will minimize the cost for $p \geq H$. The threshold H is plotted in figure 3 as a function of system utilization ρ for several different values of $r = \frac{\alpha_T}{\alpha_B}$.

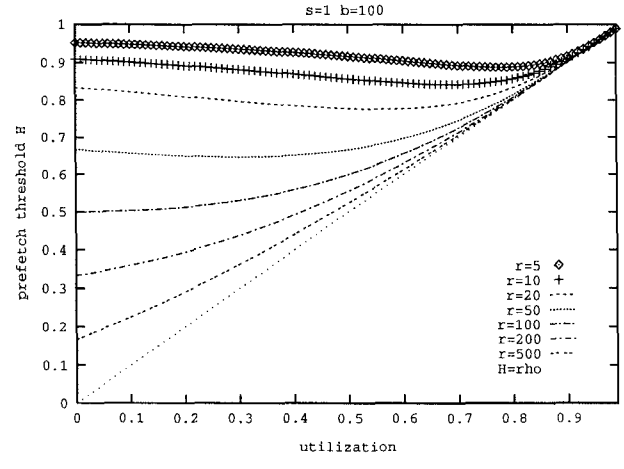


Figure 3: *Prefetch threshold* H *as a function of utilization* ρ *for different values of ratio* $r = \frac{\alpha_T}{\alpha_B}$.

Figure 3 shows that, for fixed system load ρ and capacity b, the higher the ratio $\frac{\alpha_T}{\alpha_B}$, the lower the prefetch threshold. This is because higher $\frac{\alpha_T}{\alpha_B}$ means that the time is more expensive. For fixed b and $\frac{\alpha_T}{\alpha_B}$, as ρ increases, the overall trend of the prefetch threshold increases as well, which means that fewer files should be prefetched. But the increase is not monotonic for small values of $\frac{\alpha_T}{\alpha_B}$. This is because in some situations, when the load is low, prefetching does not save much time. As the load increases, it takes longer to

transmit a file and prefetching 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. Another important thing indicated by expression (11) is, for fixed ρ and $\frac{\alpha_T}{\alpha_B}$, the prefetch threshold is higher for the system whose original capacity b is higher.

e) An upper bound for the prefetch threshold for the general system.

Up to now, we have assumed that the access probability of each file on a page is either p or 0, where p is fixed within a system but can vary for different systems. The following theorem states that for an arbitrary distribution of access probabilities, expression (11) gives an upper bound for the prefetch threshold.

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 (11)*

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

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

We have proved this theorem based on our previous result. For the complete proof, please refer to our web page http://millennium.cs.ucla.edu/~jiang/Research/.

In summary, our prefetch algorithm is as follows. For each link i with access probability p_i on a page, compute its server's access probability H using equation (11); if $p_i \geq H$, then prefetch the file otherwise do not prefetch this file.

4 Simulation results

In the last two sections, we studied the prediction algorithm and the threshold algorithm. These two algorithms are executed whenever the user starts viewing a new page, and each file with access probability greater than or equal to its server's prefetch threshold is prefetched.

We conducted simulations to measure the efficiency of our prediction algorithm. The simulations were driven by the UCLA Computer Science 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 accesses. The simulations were concentrated on two representative pages: the CS department home page and the TA home page. More precisely speaking, the prediction algorithm is activated and the program tries to prefetch files only when these two pages are being viewed. A fixed prefetch threshold was used in each simulation run and varied for different runs.

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 or handouts etc. Therefore, simulations on these two pages indicate how well our prefetch scheme works on pages that need to be visited frequently and infrequently.

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 doing prefetch. 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 4-5 show the hit rate and successful prediction rate obtained for the CS home page and the TA home page with prefetch thresholds from $0.01 - 0.9$. In addition to merging the access probabilities from the server and the client as designed in the original scheme, we also tested the cases of using probabilities from only one of them. Results from these three approaches are compared in figures 4-5 as well.

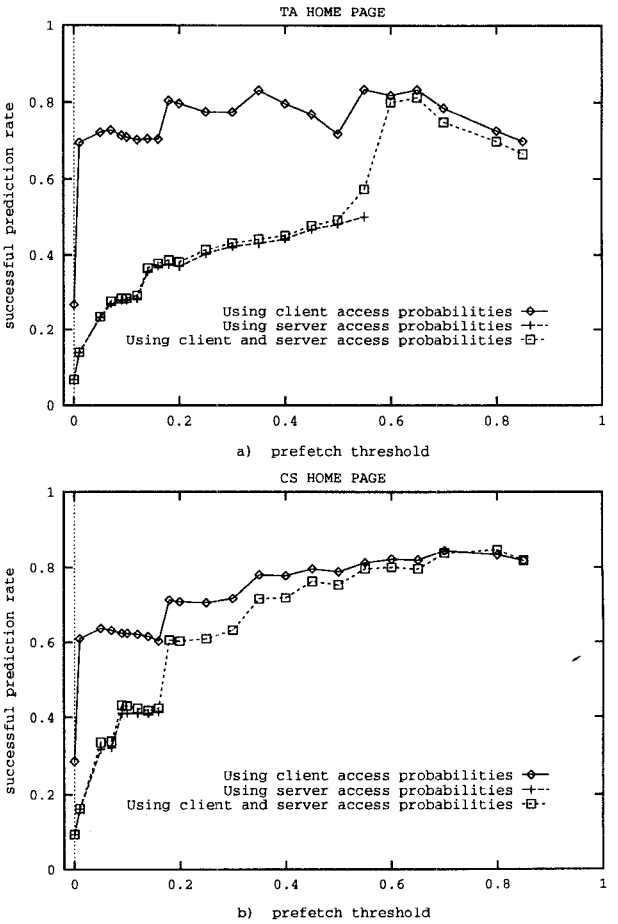


Figure 4: *Successful prediction rate vs prefetch threshold. a) TA home page, b) CS home page.*

As seen in figure 4, for both home pages, using only access probabilities on the client site yields successful predic-

tion rates of around 70% even when the threshold is zero. As the threshold increases, the successful prediction rate tends to increase slowly. In figure 4a, the tails of top two curves drop because there were very few files prefetched at those threshold values. Each curve ends at the point where no file is prefetched for larger prefetch threshold values. From figure 5 we see that, if the prediction algorithm runs only on the client site, the hit rate is much lower, about 35% compared to the successful prediction rate. On the other hand, by using only access probabilities from the server site or merging those from server and client sites, we obtain much higher hit rates. In addition, for small prefetch thresholds, these two curves are very close because prefetches invoked at the client site with high access probabilities only contribute a small portion of the total number of prefetches.

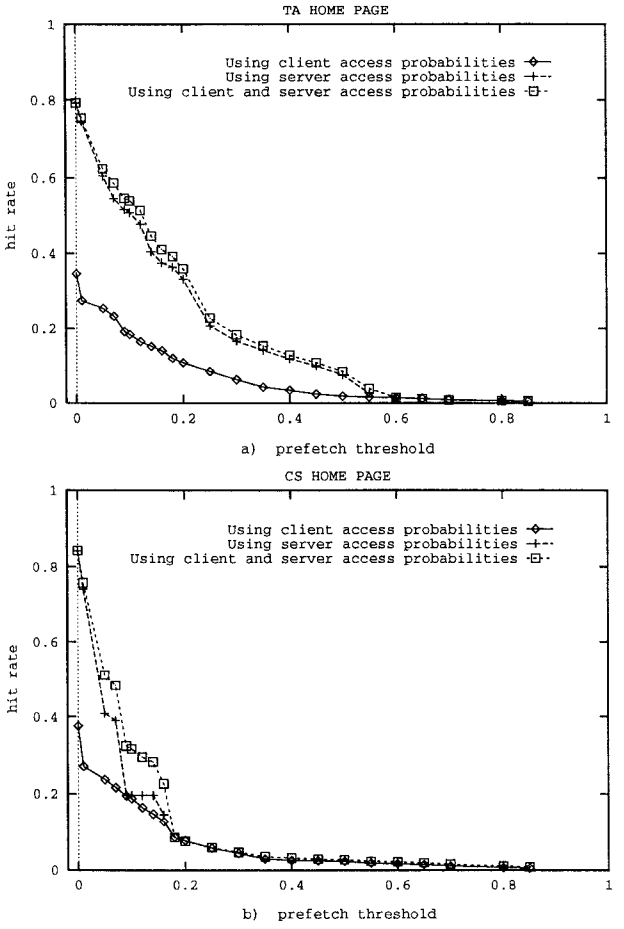


Figure 5: *Hit rate vs prefetch threshold. a) TA home page, b) CS home page.*

The hit rate of the TA home page is lower than 40% even with a threshold of zero, because some students access a class home page using bookmarks or from pages other than the TA home page. In this case no prefetch is 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 the CS home page, but is less significant.

As a further test of the prediction algorithm we divided all the requests into two subsets, those from inside UCLA and those from outside UCLA. We then compared the results of using different subsets of access history for prediction. Please refer to our home page for detail and more simulation results.

5 The prefetch program

We have developed a prefetch program at the client site on PC Windows. The program basically follows the scheme we described previously except for the threshold algorithm. Since currently we have no convenient way to obtain the capacity of the link from a user to an arbitrary server, a fixed threshold is used, but we allow at most three links be prefetched for each new page downloaded. We are working on the approximation algorithm to solve this problem.

The program is intergrated with Netscape through an Application Programming Interface (API). No modification is made to the Netscape source code. When being executed on a machine for the first time, the program needs to register with Netscape as an http protocol handler. After that, each time Netscape is opened, it will be started automatically. When the prefetch function is activated, the user may use Netscape as usual. The Netscape window looks the same, except that some icons will be added to each page being viewed. (see below)

In our program, whenever a new page is downloaded, we check the status of each distinct link on this page by measuring the time needed to connect to the server and the time between sending out a header request and receiving the response from the server. We then provide this information to the user by translating it into an icon placed next to the corresponding link. This can potentially reduce traffic at congested links or servers, because most users would choose a fast link. Figure 6 shows the icons we used in our program.

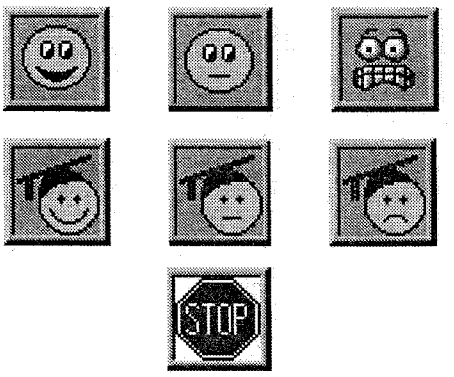


Figure 6: *The icons used in the prefetch program.*

Basically, the mouth shape of an icon indicates whether a link is fast, slow or in between, and the hat indicates if a copy of the file is available on the local disk. For example, the first icon in figure 6 means that the connection is good. Requests to this server are expected to be finished very fast. The icon below it shows not only that the link is fast, but also that the file is available on the local disk. The last icon in the second row means that the file is cached on the local disk, but

it is slow if you follow that link further through that server. The icon on the bottom row indicates an error, for example the server is not reachable or file not found, etc. After a file is prefetched, the icon next to the corresponding link will be switched to a face wearing a hat with the same mouth shape. Figure 7 is an example screen which demonstrates how the icons work.

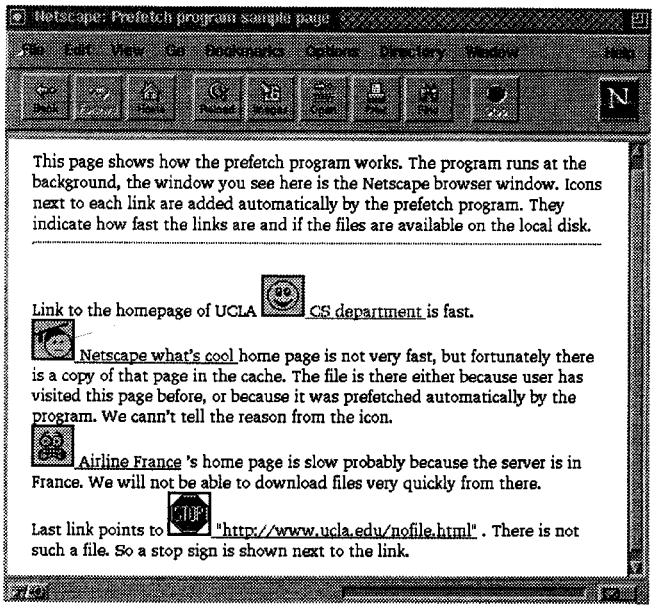


Figure 7: *Prefetch program sample page.*

Our prefetch program only handles the http protocol. In addition, images embedded in the page will generally not be prefetched except for image maps to save the bandwidth. The procedure of prefetching files is stopped immediately whenever a new request is issued by the user, except if the requested file is the one being prefetched. This may result in many files not being transmitted completely. One of the ongoing research efforts on http protocol [4] is to continue an interrupted transmission. If that is added in the http standard, we would be able to make better use of these partial documents.

6 Conclusion

Prefetching is a technique which takes advantage of the idle time between user requests to download files so that delay can be reduced when a user requests these files later. To achieve high efficiency with prefetching, we have devised an algorithm to predict as accurately as possible the probability of a file being requested by the user in the near future. In addition, given the access probabilities of files and the system conditions, we determine which files should be prefetched to maximize the savings. We do this by deriving 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. Simulations show that in general 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.

References

[1] 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,* March 1996.

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

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

[4] A. Luotonen, J. Franks, "Byte Range Retrieval Extension to HTTP", Internet Draft, draft-ietf-http-v10-spec-04.html, February 1996.

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

[6] V.N. Padmanabhan, "Improving World Wide Web Latency", Technical Report UCB/CSD-95-875, UC Berkeley, May 1995.