Abstract

Prefetching is one of the most popular techniques for dealing with the slow access speed of the World Wide Web. To provide a mobile user with effective real time online prefetching requires that the prefetch decision is able to adapt to different network systems. This article describes an adaptive network prefetch scheme which accomplishes this task. The basic scheme is comprised of a prediction module and a threshold module, which computes the access probabilities and prefetch thresholds respectively. The access probabilities indicate how likely files will be requested by the user, and the prefetch thresholds determine whether the performance may be improved by prefetching certain files. As a user changes network in a mobile environment, it is the prefetch threshold, which is computed based on system conditions as well as costs of bandwidth and time, that adjusts the number of prefetched files accordingly. In addition, by extending the method of computing the access probabilities, we are able to prefetch a group of files together for a user who is about to be disconnected from the network.

Web Prefetching in a Mobile Environment

Zhimei Jiang and Leonard Kleinrock
University of California at Los Angeles

As the popularity of the World Wide Web grows, so does the demand for faster and more efficient access to the Internet. Several techniques have been developed to meet this demand, including caching, prefetching, and pushing. Each of them has its own strengths and weaknesses, and all three approaches are being incorporated in the Internet. In this article, we investigate real time online prefetching. Specifically, while the user is browsing the WWW, pages that will very likely be accessed in the near future are downloaded according to certain criteria. Upon receiving a request from the user, the browser displays the prefetched version of the file whenever it is available and is up-to-date. As the user continues to issue new requests, new candidate pages for prefetching appear and more files may be prefetched as a result.

The idea of prefetching stems from the fact that, after retrieving a page from the remote server, a user usually spends some time viewing the page, and during this period the local machine, as well as the network link, is generally idle. If we can take advantage of this phenomenon by fetching some files that will likely be accessed soon using the otherwise idle system (in other words, prefetching them to the local disk), there will be no transmission delay when the user actually requests these files. In addition, prefetched files can be immediately processed if decryption or decompression is required, or if any Java applets need to be handled, which allows further reduction in the delay of loading a page. These are some of the benefits we can get from prefetching. The difficulty of realizing efficient prefetching lies in the fact that it is impossible to predict exactly what a user is going to need. So some of the prefetched files are never used, which means prefetching increases the system load. At some point, this increase in load may increase the delay of normal (nonprefetching) requests significantly and may eventually cause the overall system performance to degrade. Therefore, the key challenge in prefetching is to determine "what to prefetch" so that improvement in performance will be enhanced.

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

The prefetching problem may be solved in two steps. First, we need to find out the likelihood with which each file will be accessed. In this article, we estimate this likelihood with the *access probability*, which is defined as the conditional probability of a file being requested by the user in the near future, given the page currently being viewed. Once the access probability of a file is obtained, the next step is to determine whether it is more cost effective to prefetch this file. It turns out that, for the server on which the file is located, we can compute a *prefetch threshold* such that the average delay is guaranteed to be reduced by prefetching this file as long as its access probability is greater than its server's prefetch threshold. As mentioned before, the prefetch threshold is a function of current system conditions and certain cost parameters.

In a mobile environment a user moves around and connects to the backbone network through different types of links, or even becomes completely disconnected. Mobility brings new challenges to the prefetch problem, because each type of network link has its own characteristics in terms of capacity and bandwidth cost, which will certainly require that different numbers of files be prefetched for different links to achieve the largest performance improvement. In our prefetch scheme, since the prefetch thresholds are computed based on parameters including network capacity and network cost, the amount of data prefetched adapts naturally to the mobile environment. Furthermore, the prediction algorithm may be extended to download files in a group so that a user may access these files while disconnected from the network.

The Basic Prefetch Scheme

In [1], we proposed an adaptive network prefetch scheme comprising a prediction module and a threshold module. The two modules compute the access probabilities and the prefetch thresholds, respectively. The prefetch scheme works in the following way, as illustrated in Fig. 1. Basically, whenever a new page is displayed, the prediction module updates the local access history, if needed, and computes the access probability of each link on that page. At the same time, for each server which has at least one link on this page, the threshold module computes its prefetch threshold based on network and server conditions as well as the costs of time and bandwidth to the user. Finally, all the files with access probability greater than its server's prefetch threshold are prefetched. The prediction algorithm may also be run at the server, in which case access probabilities will be sent to the user along with the requested page. Details about these two modules will be discussed in the following sections. Our prefetch scheme is a general one that may be applied to almost any network application to decide what information to prefetch [1]. In this article we focus on its application to Web browsing.

The rest of this article is organized as follows. In the next two sections, we study the prediction algorithm and the threshold algorithm, respectively. We first summarize results from [1] and then discuss the implication of these results for implementing prefetching on the Internet. We then show how the basic prefetch scheme may be applied to mobile environments. The article goes on to discuss a prefetch program we have developed. In the last section, we compare prefetching with two other popular Internet technologies: caching and pushing.

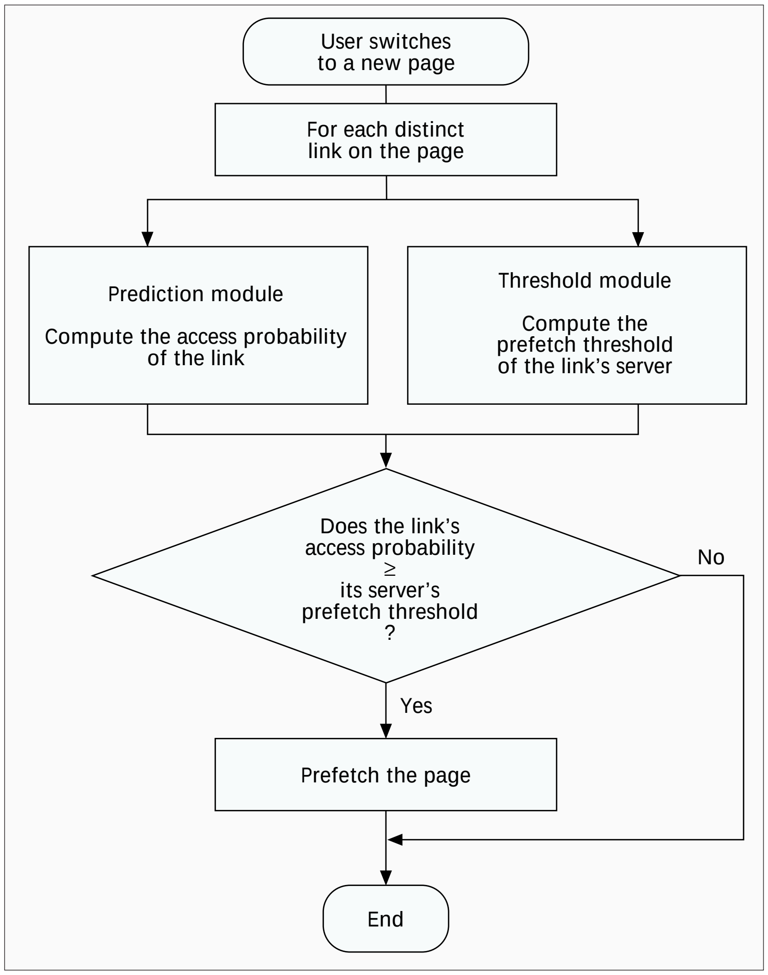

User switches to a new page
For each distinct link on the page
Prediction module
Compute the access probability of the link
Threshold module
Compute the prefetch threshold of the link's server
Does the link's access probability ≥ its server's prefetch threshold ?
No
Yes
Prefetch the page
End

Figure 1. *Adaptive prefetch scheme flowchart.*

Prediction Module

Prediction algorithms have been studied intensively in areas such as operating systems. Typically, they keep track of the sequence of actions taken by a user or a program to determine the pattern in which these actions take place. And most of them consider actions that are close to each other (or, in other words, fall in the same window) to be dependent, where the definition of window is determined by the specific algorithm. Based on this information, a dependency graph is created, in which nodes represent actions, and the weight of the directed edge from node *i* to *j* indicates how likely action *i* is followed by action *j* closely.

The approach described above is also employed in several prediction algorithms for Web browsing [2, 3]. The window sizes in these algorithms are chosen in terms of either the number of page requests or the time between two requests. However, this kind of algorithm is generally not sufficient for Web browsing. As an example, let's consider the pages shown in Fig. 2. Assume the top-level news summary page has two links, one to the business news page and another to the sports news page. In addition, on the two second-level pages, there are many links to the latest news in the corresponding category. A user checks both pages at level two every day through the links on the summary page. If he/she always reads the business news page and the pages under it first, then in the request sequence the requests for the sports page are usually far away from those for the business and news summary pages. Hence, the dependency between them cannot be captured unless the window size is set very large, which is undesirable in many situations. If the user randomly picks one of the two second-level pages to visit first, the dependencies between them and the summary page will still be weakened. This situation occurs more often than one might think given that most of the time a page request is followed by several requests for images on that page, which may increase the distance between two page requests substantially.

Seeking a more effective prediction algorithm, we notice that Web pages are usually structured to link related pages together, and it is more likely that a user would click links on the pages or buttons on the browser, rather than type in a URL, to go to the page in which he/she is interested. Occasionally, bookmarks are used to switch to a different site with a related topic. Based on this observation, in [1] we proposed a prediction algorithm in which we assumed that at any time only pages linked directly to the currently viewed page may have nonzero access probabilities.

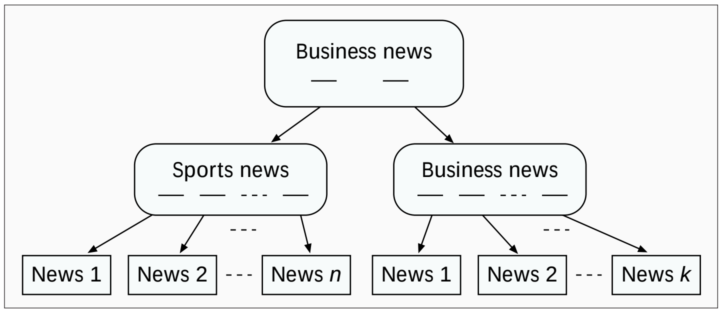
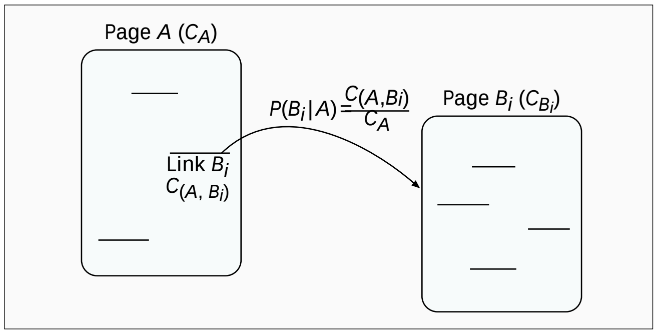


Figure 2. *An example of structured links in the WWW.*

A Prediction Algorithm

In our prediction algorithm, in order to compute the access probabilities we keep track of which pages are accessed and how they are accessed. Specifically, for a page, say page A, and a link B_i on A, we use a counter C_A to track the number of times page A has been *down-*



■ **Figure 3.** *A prediction algorithm for Web browsing.*

loaded, and use a counter C_{A,B_i} to track the number of times page B_i is accessed by clicking on its link on page A. There is exactly one counter for each page and one for each distinct link on a page. The related counters are updated each time the user issues a new request. The access probability $P\{B_i|A\}$, which is the conditional probability that link B_i will be accessed given that A is being viewed, is computed as follows:

$$P\{B_i \mid A\} = \begin{cases} \min\left(1, \dfrac{C_{A,B_i}}{C_A}\right) & \text{for } C_A \geq 5 \\ 0 & \text{for } C_A < 5 \end{cases} \qquad (1)$$

We arbitrarily require C_A to be at least 5 in Eq. 1 because if the user has not been to a page enough times, the access history may not represent his/her real interest. This algorithm is illustrated in Fig. 3. If a page, D, does not have a link on A, then $P\{D|A\}$ is simply zero according to our assumption. For the example above, as long as the news summary page is not updated with very high frequency, our prediction algorithm will generate access probabilities close to one for both the business and sports news pages, since both are accessed each time the user reads news.

Our prediction algorithm may be run at either the client or server site. In the latter case, the server aggregates the access patterns of recent visitors to compute the access probabilities for future users, and access probabilities of the links on a page are sent to the user together with the page to minimize delay and processing cost. Currently, most Internet Web servers save access requests from users in their log files, which can easily be used by our prediction algorithm. More details about the algorithm and its implementation may be found in [1].

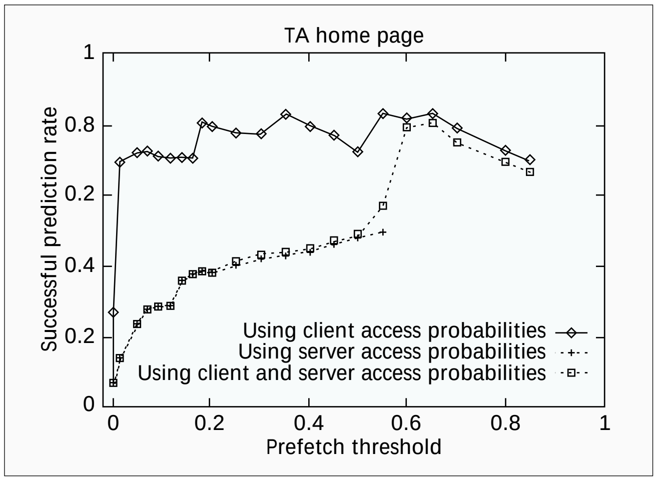
Clearly, access probabilities generated at a client site indicate the user's personal interests, while those from a server show the popularity of the files among many users who have accessed this server. If a user has not visited a page often enough to obtain reliable access probabilities based on his/her own access history, the server's access probabilities are likely to be more accurate. Access probabilities from server and client may be merged in the following way at the client site. If a page has been visited fewer than five times by the user, and the access probabilities of its links are available at the server, the probabilities from the server are used; otherwise, access probabilities from the client are used. We compared the performance of access probabilities from different sources through simulation. The results are summarized below.

Simulation Results

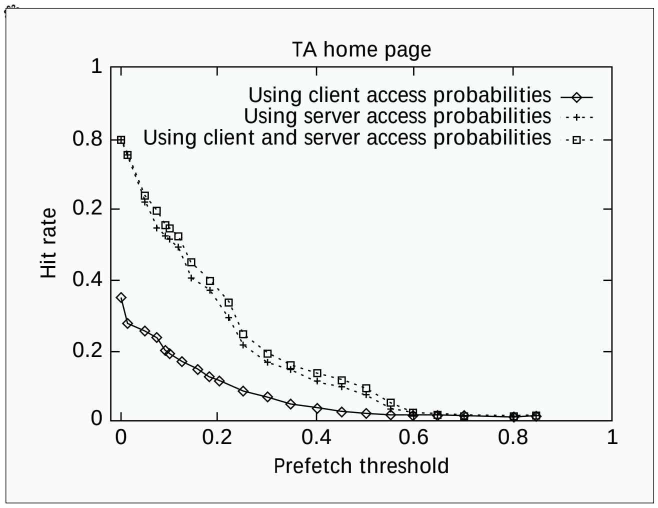
To further study our prediction algorithm, we compared the performance of prediction using access probabilities from either the client, the server, or a combination of them as described above through trace-driven simulations, where the trace was taken from the log file of the UCLA Computer Science Department Web server. In the simulation, we measured two important parameters: the hit rate and the successful prediction rate. The *hit rate* refers to the percentage of user requests satisfied by the prefetched files. Generally, the higher the hit rate, the more time saved by prefetching. The *successful prediction rate* is the probability of a prefetched file being used eventually. A high successful prediction rate indicates that less bandwidth is wasted due to prefetching unused files. A fixed prefetch threshold was used in each simulation run, and all the files with access probability greater than or equal to the prefetch threshold were prefetched.

One of the Web pages we examined is the homepage for TAs (teaching assistants), which contains links to all the class homepages. These class homepages are updated frequently to include new assignments, handouts, and so on. Figures 4 and 5 show the successful prediction rate and hit rate obtained for this TA home page, with prefetch thresholds ranging from 0.01 to 0.9 using the three different sources of access probabilities described above.

As expected, when using only access probabilities from the client site, the successful prediction rate is generally high. Figure 4 shows that its value is around 70 percent even when the threshold is just slightly higher than zero. The successful prediction rate is lower when access probabilities from the server



■ **Figure 4.** *Successful prediction rate vs. prefetch threshold for a TA homepage.*



■ **Figure 5.** *Hit rate vs. prefetch threshold for a TA homepage.*

or the combination of client and server access probabilities are used. However, the result is good considering that the server access probabilities are computed from the access history of hundreds of other clients who are totally unrelated to the user. While client access probabilities yield a high successful prediction rate, they are available only for a limited number of pages that are visited frequently by the user. Thus, the hit rate in this case is lower than that using server access probabilities, as shown in Fig. 5. More simulation results can be found in [1]. In the rest of this section, we briefly discuss several possible extensions to the basic prediction algorithm.

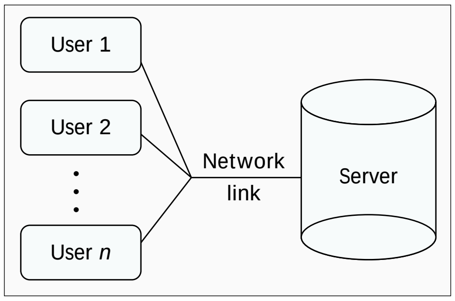


Figure 6. *The system model used for studying the prefetch threshold.*

Extensions to the Prediction Algorithm

Figure 4 shows that when using access probabilities from the server, the successful prediction rate is generally lower. To achieve a better prediction result, we may divide users into different categories according to their interests. For example, one category may be created for users who are interested in pages related to database research. Each category has its own set of access probabilities, which are computed from the access history of users within the same category. Initially, a new user is provided the access probabilities of the general category. Once a specific category in which the user belongs is determined based on his/her access pattern, access probabilities for this category will be sent instead. This categorization of users according to their interests may be realized using cluster algorithms such as the one proposed in [4].

Another possible extension is to predict deeper than one layer of links. More specifically, assuming there is no direct link for C on page A, if link B on page A has access probability $P\{B|A\}$ and link C on B has access probability $P\{C|B\}$, then we may let access probability $P\{C|A\}$ be $P\{B|A\}P\{C|B\}$, instead of zero as in the original algorithm. By extending the method of computing access probabilities in this way, pages that are not linked directly to the current page may also be prefetched. This extension becomes necessary in cases where a user would like to download a relatively large group of pages related to a certain topic for future reference without going through every intermediate page. We will continue to discuss this issue later as one of the strategies for dealing with mobility.

In addition to the linking relations among pages, there are other things we may take advantage of to improve the prediction results. One possibility is to use bookmarks. For example, when a user logs on to the Internet for the first time in a day, he usually goes through the same routine: checks stock prices, news, daily comics, and so on. If these pages are bookmarked, a prediction algorithm can then observe which sequence of bookmarks are likely to be used close to each other in a session, regardless of the number of page requests between them. When one bookmarked page is requested, others that might follow it may then be prefetched.

Other algorithms turn to the users for better prediction by asking them either to indicate what they are interested in or to rate the prediction results as feedback to the program [5, 6]. In general, the more user involvement required, the more inconvenience might be experienced. We believe that the way in which pages are linked together is one of the most important items of information we need to take advantage of for efficient Web browsing prediction.

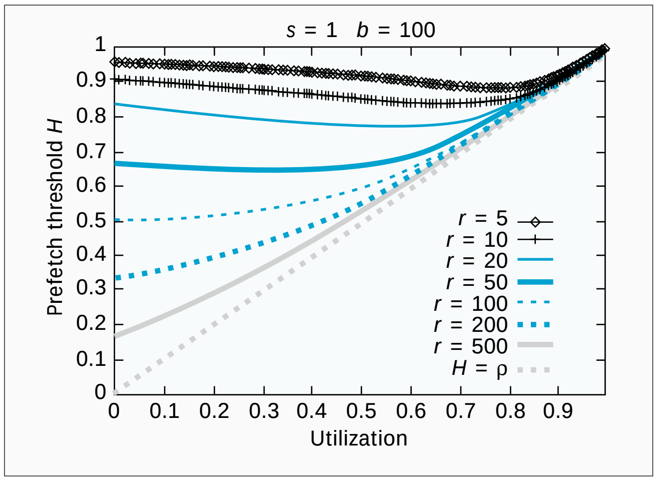


Figure 7. *Prefetch threshold* H *as a function of utilization* ρ *for different values of* r (r = α_T/α_B, b = *100).*

Threshold Module

Once the access probability of a link is obtained, the next step is to find out whether it is worthwhile to prefetch the corresponding file. There are three factors we must consider when selecting files to prefetch: the amount of time that might be saved by prefetching the file for the user who may need it; the amount of bandwidth that will be wasted if the file is not used; and, more important, the impact of the prefetch request on other users whose normal requests may be delayed by the prefetch request. The latter is a necessary consideration in order to ensure that the overall system performance can be improved by prefetching the file.

In [1], we studied this problem using the multi-user system model shown in Fig. 6, where users share the same server as well as the network link; we assumed a round-robin processor sharing system [7]. System performance is measured in terms of cost, which is the sum of the response time cost (α_T\$/s) and bandwidth cost (α_B\$/kb). By determining which files should be prefetched such that the total cost in the system may be safely reduced, we are able to prove that for an *arbitrary* distribution of access probabilities, an *upper bound* on the optimum prefetch threshold is given by

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

where b is the system capacity in kilobits per second and r is the average system utilization.

What this result indicates is that, for the system shown in Fig. 6, regardless of the access probability distribution, as long as each user only prefetches files with access probability greater than the prefetch threshold computed by Eq. 2, we are guaranteed to achieve a lower average cost per user request than without prefetching. Moreover, the more such files are prefetched, the lower the average cost. The fact that H is an upper bound on the optimum prefetch threshold determined by the actual distribution of access probabilities is particularly important, since it is almost impossible to find this distribution in a real system. From now on, we use this upper bound H as the prefetch threshold of the server.

According to Eq. 2, for a given system (i.e., for fixed b), the prefetch threshold is affected by only two parameters: ρ and α_T/α_B. Figure 7 shows how the threshold changes with the system load ρ for different values of α_T/α_B. As expected, the higher the ratio α_T/α_B (in other words, the more expensive the time compared to the bandwidth), the lower the threshold. Thus, more files are prefetched to save time. The relation between the threshold and system load is more complicated. In general, for $b \le \alpha_T/\alpha_B$, the threshold increases monotonically with system load; while for $b > \alpha_T/\alpha_B$, it first decreases slightly and then increases with increasing load [1].

Let us now consider two systems with the same α_T/α_B but different capacity b. If they also have the same load ρ, Eq. 2 indicates that the threshold is higher in the faster system (i.e., when b is larger). By multiplying both the denominator and the numerator of Eq. 2 by b, we obtain

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

Since the average delay of transmitting a file of size s is

$$\frac{s}{(1-\rho)b},$$

it follows that for these two systems, which have the same α_T/α_B but different capacity b, when the average delay and average file size are also the same, the threshold is again higher in the faster system.

Another interesting point is that the threshold is independent of individual file size. (Of course, it does depend on the average file size which affects the system load ρ.) This implies that for two files with the same access probability but different file sizes, we do not prefer to prefetch the smaller file over the larger one or vice versa. This is because, although prefetching the larger file consumes more bandwidth, it also saves more time if the user does request it later. Here, we assume that a file cannot be used unless the entire file has been downloaded. In Web browsing, a user may start viewing the top part of a page while the rest is still being downloaded, which gives preference to prefetching smaller files. However, if we change our strategy accordingly to prefetch just the top portion of big files, the individual file size would again make no difference to the prefetch decision.

Equation 2 is derived for a system in which all users have the same α_T. In reality, people value their time differently, which means α_T varies. Assume there are n different types of users in the system, and the time of type k users is worth α_{T_k}*($/s)*. We can prove that the prefetch threshold for type k users is

$$H_k = 1 - \frac{(1-\rho)\dfrac{\alpha_{T_k}}{\alpha_B}}{(1-\rho)^2 b + \dfrac{\alpha_{T_k}}{\alpha_B}(1-\rho) + \dfrac{s}{b\alpha_B}\sum_{i=1}^{n} \lambda_{i1}\alpha_{T_i}} \qquad (3)$$

where λ_{i1} is the arrival rate of normal (nonprefetching) requests from type i users and s is the average file size [1]. Similar to the previous system, by letting type k users only prefetch files with access probability greater than H_k, a lower average cost can always be achieved. And the more files are prefetched in this way, the lower the cost. Furthermore, the results we discussed in the last few paragraphs also hold for this more complex system. The main difference is that H_k also depends on the distribution of the requests from different types of users. The conservative way to estimate H_k is to assume that all other requests are from users with the highest α_T. A better estimation may be obtained if the system can provide such information.

Our threshold algorithm computes the prefetch threshold based on system conditions and costs to guarantee that prefetching improves system performance. This is certainly more efficient than fixing either the number of files prefetched on a page or the value of the prefetch threshold. In addition, it allows the prefetch threshold to adapt naturally to different systems in a mobile environment. In the next section, we investigate how mobility is supported by our prefetch scheme.

Prefetching in a Mobile Environment

So far, we have discussed the prediction and threshold algorithms, which compute the access probabilities and prefetch thresholds, respectively. In particular, we showed that by prefetching only those files with an access probability greater than its server's prefetch threshold, a lower average cost can always be achieved. And the more such files are prefetched, the greater the improvement. Our study of the prefetch threshold is based on the multi-user system shown in Fig. 6. For the real Internet, if we divide the entire path from a client to a server into three components, namely the user's intranet, the link between the intranet and the Internet, and the rest of the Internet[1] as illustrated in Fig. 8, each component may be modeled as a multi-user system in Fig. 6. Note that a user shares the system resources with different users in different components. Depending on the location of the user and the server, the Internet may look different in terms of delay, capacity, and load [8]. In general, it is relatively easy to obtain these network characteristics for the Intranet and the connection between the Intranet and the Internet backbone. Systems like the one proposed in [9] may help us getting information about backbone network and Web servers

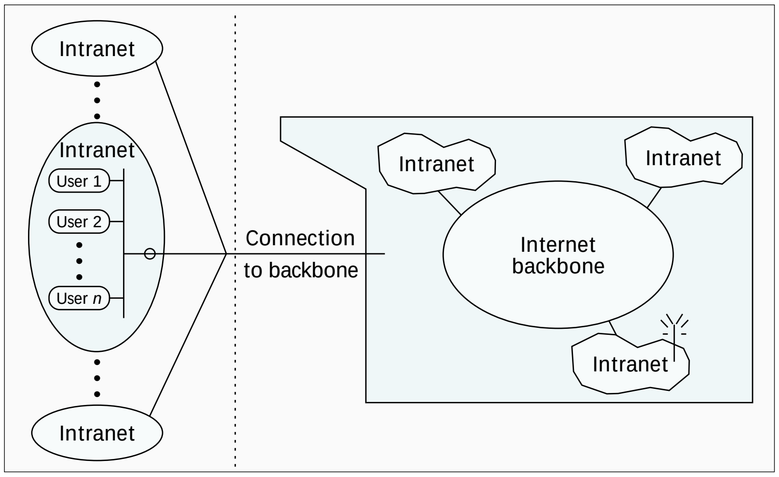


Figure 8. *Mapping between the multi-user system model and the Internet.*

[1] We may further divide the rest of the Internet into the backbone network and the network which contains the Web server.

efficiently. To estimate the prefetch threshold of the server for the user, Eqs. 2 or 3 may be used to compute the prefetch threshold for each part; then the maximum of the three is taken as the prefetch threshold for this server. As a user moves around in a mobile environment, all three components of the path may change, which will subsequently affect the prefetch threshold and hence the number of files prefetched.

In this section we investigate how to handle prefetching in two different situations involving mobility. In the first case, a user moves around carrying his laptop while being connected to different types of network links, such as a high-speed Ethernet, a phone line, a wireless network, and so forth. Since each type of link has its own characteristics in terms of cost and capacity, we need to find out how they affect the value of the prefetch threshold. In the second case, the user anticipates that he will be disconnected from the network for a long period of time. In order to be able to access network files during that period, the user may indicate what information will be needed and prefetch some of it when connection is available. This is different from the real-time prefetching model we have been discussing so far in this article. However, it turns out that our prediction algorithm may be extended to determine which files to prefetch in this situation as well. The mobile environment in the real world is more like a mixture of the above two cases. Namely, the user connects to the network via different links at different locations, while in between he is disconnected from the network for a certain period of time. In the following, we discuss them separately in detail.

Network	Capacity (B)	Cost[1]	Cost/kb (α_B)[2]	$\alpha_B \cdot b$
Dial-up	28–56 kb/s	$2/hr.	%6.9 x 10^{-6}	2.8 x 10^{-4}
CDPD	10.2 kb/2	$.14/kbyte[3]	$1.8 x 10^{-2}	3.4 x 10^{-1}
Cable modem	500 kb/s	$50/month	$1.5 x 10^{-7}	7.7 x 10^{-5}
Satellite (DirectPC)	400 kb/s	$.60/Mbyte[3]	$7.5 x 10^{-5}	3.0 x 10^{-2}
ISDN	64 kb/s	$200/month	$4.8 x 10^{-6}	3.1 x 10^{-4}
Frame relay	128 kb/s	$500/month	$6.0 x 10^{-6}	7.7 x 10^{-4}
T1	1.544 Mb/s	$1200/month	$1.2 x 10^{-6}	1.9 x 10^{-3}
T3	44.736 Mb/s	$25000;/month	$8.6 x 10^{-7}	3.8 x 10^{-2}

[1] All costs are averaged over several providers.
[2] Refer to "Prefetching for a Mobile User" for how the α_Bs are computed.
[3] Price varies with minimum monthly payment and time of day.

Table 1. *Capacity and cost of some popular network connections.*

Prefetching for a Mobile User

Using our adaptive prefetch scheme, it is straightforward to support prefetching for a user who is moving among networks. Basically, the prediction algorithm remains the same for different networks. To compute the prefetch threshold of a server for an Internet user, as mentioned before, we divide the entire path between them into three parts and calculate the prefetch threshold for each part separately. Then the maximum of the three thresholds is used as the prefetch threshold for the server. As the user moves around and is connected to different subnets, the α_B, b, and ρ in each of the three components may change. Therefore, the prefetch threshold for the server changes with the environment according to the formula presented in the last section. In this way, mobility is supported nicely with our prefetch scheme.

We now show how the prefetch threshold may differ in various networks. There are many options available to connect users within a subnet or to the backbone network, and the cost and capacity of these links vary over a large range. At one end are high-speed optical links, fast and relatively cheap; at the other end are wireless networks, slow and expensive; between are links with a fair amount of capacity and moderate cost. Table 1 lists the cost and capacity (b) of some popular network connections available on the market. In order to apply our prefetch threshold formula to a particular network, we need the value of its α_B, which indicates the cost of transmitting 1 kb of data. However, in Table 1 only the α_Bs for CDPD and DirectPC are given directly. Most providers charge for network usage according to time. In other words, a user pays to use network links for a certain period of time, regardless of the amount of data actually transmitted. Therefore, α_B is simply zero in this case. Note that the α_B in some networks (e.g., the LAN within a company) is inherently zero because people are not charged for using the network. When $\alpha_B = 0$, the prefetch threshold is given by

$$H_k = 1 - \frac{(1-\rho)\alpha_{T_k}}{(1-\rho)\alpha_{T_k} + \frac{s}{b}\sum_{i=1}^{n}\lambda_{i1}\alpha_{T_i}} \qquad (4)$$

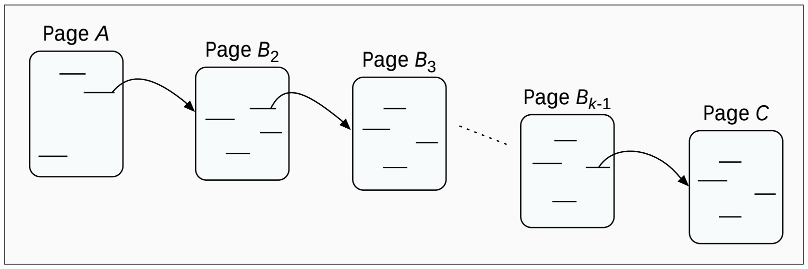
based on the proof in [1], where s, b, λ_{i1}, and α_{T_i} are defined in the same way as in Eq. 3. In particular, if α_T is the same for all users sharing the system, the prefetch threshold is simply $H = \rho$.

To further compare the prefetch threshold in different environments, for those networks that are charged by time we wish to find out how a user may be charged according to the actual bandwidth usage, and still end up paying the same cost on average for the same period. In particular, we estimate the actual bandwidth usage by assuming that half of the network capacity is wasted if charged by the hour and three quarters is wasted if charged by the month, since in the latter case the network is mostly idle at night. Then the cost per kilobit, α_B, may be obtained by dividing the cost per hour or month with the actual amount of data transmitted during that period. In Table 1, the α_Bs for links other than CDPD and DirectPC are computed in this way.

The following observation is quite interesting. From Eq. 2 for the prefetch threshold, if we multiply both denominator and numerator by α_B, we have

$$H = 1 - \frac{(1-\rho)\alpha_T}{(1-\rho)^2 \cdot \alpha_B b + \alpha_T}. \qquad (5)$$

Equation 5 shows that for two systems with the same r, if the user also sets his α_T the same for both links, the threshold only depends on the product of α_B and b; and the lower $\alpha_B b$, the lower the prefetch threshold. This implies that the prefetch threshold of a link which is expensive (high α_B) and slow (low b) may be the same or even lower than that of a faster link with low α_B. This seems to be somewhat contrary to our intuition that when bandwidth is scarce and expensive, fewer files should be prefetched. However, what this result really indicates is that if a user "has to" get the files regardless of the network link being used, the prefetch decision depends on $\alpha_B b$ and we might need to prefetch more files in a more costly and slower network. If we don't do so, the time cost to the



■ **Figure 9.** *A simple path between two pages,* A *and* C.

user will surpass the bandwidth cost and result in an even higher total cost. From another point of view, $\alpha_B b$ is actually the link cost per unit time, assuming that data is transmitted at the full speed given by b. In Table 1, the per-kilobit cost α_B of a dial-up network is higher than that of a T1 link; however, more files are prefetched in the former system which has lower $\alpha_B b$.

For given α_T and ρ, the prefetch threshold for wireless networks (e.g., CDPD) is highest, since its transmission cost is much higher than the others. When bandwidth is expensive, users are often willing to delay requests or sacrifice the quality of the page by requesting less information. For instance, instead of sending a full page with large high-resolution images, the page may be re-authored to reduce its size, and distilled images which are much smaller may be sent [10, 11]. Or, in a case in which a file is only updated slightly, rather than sending the whole file, the difference between the two versions may be delivered [12]. Most of these methods require extra processing at a proxy server which is connected to the backbone network through a faster and less expensive link. By combining them with prefetching, even better performance can be achieved. For example, upon receiving a prefetch request from a user, the proxy server fetches the file from the server and starts processing it. Then the simplified version is forwarded to the client over the slow link. Even if the user decides not to prefetch any files through the slow link, a significant amount of processing time may still be saved by prefetching the files to the proxy server and having them processed in advance.

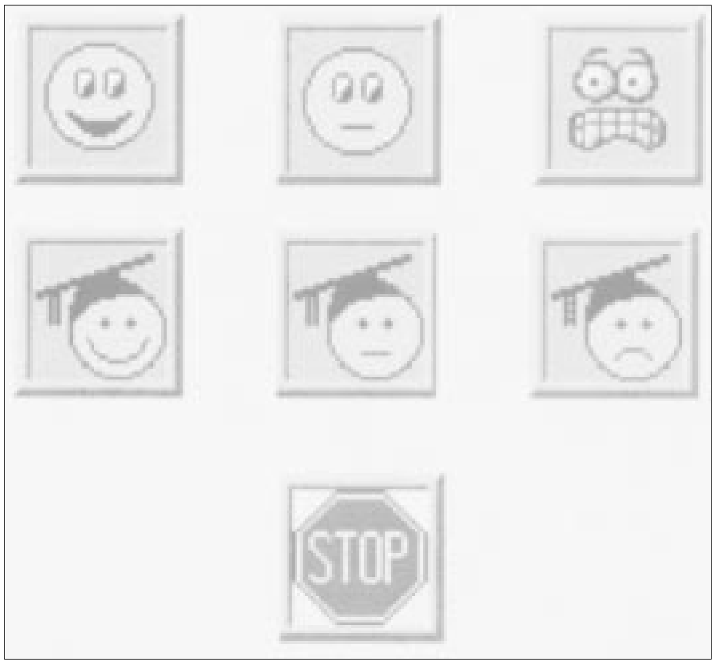
Off-Line Prefetching

Let us now consider another scenario in which files are prefetched to deal with disconnectivity. Suppose a businessman is going to fly to another city for a meeting. During the trip, he would like to do some research on a product that is available from several companies through their Web pages. However, it is very expensive to access the Internet during the trip. To avoid paying the high price, he might choose to download the information beforehand. One way of doing this is to go to each page that seems to be interesting and download it without reading the details. The more efficient way is to have it done automatically by the prefetch program, which we give the name of "off-line prefetching," as opposed to the real-time online prefetching we have been studying so far in this article.

For off-line prefetching, the threshold algorithm discussed earlier is no longer applicable. Instead, the user may simply indicate an upper bound for the bandwidth cost or the amount of data to be downloaded. However, to find out how likely it is that a file may be accessed, we may continue to use our prediction algorithm by applying our extended definition of access probability to pages that are not linked directly to the current page. Specifically, assume that page C can be reached from page A through a sequence of pages, say $A = B_1, B_2, \dots, B_k = C$, where $B_i \neq B_j$ for $i \neq j$ (i.e., there is no loop in the path; Fig. 9). Clearly, the probability that page C may be accessed from A through this particular path is given by $P\{B_2|A\}P\{B_3|B_2\}$ $\dots P\{B\{k-1|B_k-2\}P\{C|B_k-1\}$, where $P\{B_i + 1|B_i\}$ $(1 \leq i \leq k-1)$ is the access probability defined earlier. The value of $P\{B_i + 1|B_i\}$ is computed with Eq. 1, in which the counters used are the same as those for real-time prefetching. Then the access probability $P\{C|A\}$, which is defined as the conditional probability that C will eventually be accessed given that A is the current page, may be obtained by taking the summation of the probabilities for all the simple paths from A to C. Sometimes, the exact value of $P\{C|A\}$ is impossible to find, since there is no upper bound on the path length. In this case, we may estimate $P\{C|A\}$ by only considering paths shorter than a given length, or approaching its true value gradually as new paths are found. In our off-line prefetch scheme, we take the second approach, because it requires fewer pages to be expanded.

In order to prefetch a group of pages related to a certain topic, the user needs to specify some initial pages for the prefetch program to start with. The initial pages may be obtained either by searching for the topic or from the bookmarks. For simplicity, we assume there is a virtual page that has links to, and only to, all the initial pages. Pages waiting to be prefetched are sorted in decreasing order of their access probabilities from the virtual page based on all the paths discovered so far, and all the initial pages are assigned access probability 1. The page at the head of the queue is prefetched each time. Once a prefetched page is received, the access probability of each link on this new page is computed. If a new path to a page already in the queue is found, the access probability for the page is updated and its position in the queue changed accordingly. Otherwise, the page is simply inserted into the queue if it has not already been prefetched. Prefetching stops as soon as the total bandwidth cost, or the total amount of data downloaded, exceeds the limit placed by the user.

What we have described are some basic functions that

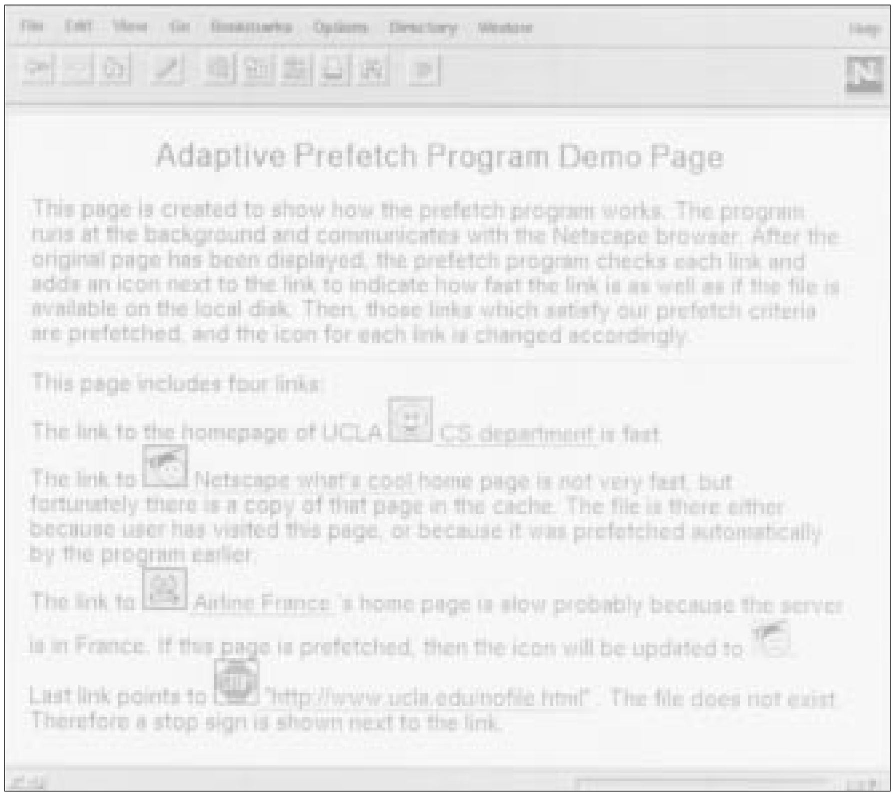


■ **Figure 10.** *Icons used in our prefetch program.*

should be included in an off-line prefetching algorithm. More sophisticated functions can be added to select files more intelligently. For instance, we may require that only pages containing certain keywords may be prefetched. Our goal here is simply to show how prefetching can be applied in this kind of situation, and how the access probabilities may be obtained.

Implementation: A Prefetch Program

We have developed a program on a PC Windows system based on the real-time prefetch scheme described in the last few sections. The program is integrated with the Netscape browser through the Netscape Client application programming interface (API). When executed on a PC for the first time, it needs to register with the machine's Netscape program as an http protocol handler. After that, each time Netscape is opened, the prefetch program is 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 the page being viewed to alert the user regarding the status of the links on that page (see below).



Adaptive Prefetch Program Demo Page

This page is created to show how the prefetch program works. The program runs at the background and communicates with the Netscape browser. After the original page has been displayed, the prefetch program checks each link and adds an icon next to the link to indicate how fast the link is as well as if the file is available on the local disk. Then, those links which satisfy our prefetch criteria are prefetched, and the icon for each link is changed accordingly.

This page includes four links:

The link to the homepage of UCLA CS department is fast.

The link to Netscape what's cool home page is not very fast, but fortunately there is a copy of that page in the cache. The file is there either because user has visited this page, or because it was prefetched automatically by the program earlier.

The link to Airline France 's home page is slow probably because the server is in France. If this page is prefetched, then the icon will be updated to .

Last link points to "http://www.ucla.edu/nofile.html" . The file does not exist. Therefore a stop sign is shown next to the link.

■ **Figure 11.** *A prefetch program sample page.*

In our program, whenever a new page is displayed, the status of each link on this page is checked by measuring the sum of the time needed to connect to the server and the time between sending out a header request and receiving the response from the server. The value obtained in this way is not very accurate and may vary with time. However, in general, it does indicate the quality of the link at that time. If the user has downloaded some pages from the server recently, say in the last 10 minutes, the average speed at which the files were downloaded is a better estimation of the actual link condition [8]. This link quality information is provided to the user through an icon placed next to the corresponding link. This can potentially reduce traffic on congested links, because most people would choose a faster link if several options are available. Figure 10 shows the icons used in our 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 cached on the local disk. For example, the first icon in Fig. 10 means that the connection is good. Requests to this server are expected to be satisfied 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 indicates that the file is cached on the local disk, but it may be slow if you follow that link further to access other pages on that server. The icon at the bottom indicates an error (e.g., the server is not reachable or the file was not found). 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 11 is a sample screen which demonstrates how the icons work.

Our prefetch program keeps track of a user's access history and computes access probabilities as described earlier. The access probabilities are only available to pages that are visited frequently, since we have not implemented the prediction algorithm on the server site yet. Because of this and also because we are currently unable to obtain the capacity information of the path to an arbitrary server, we limit the number of links that may be prefetched on each page to three. In general, links with the highest access probabilities, if they are available, or the slowest ones are prefetched. We are currently working on a better implementation of the original prefetch scheme.

In order to save bandwidth, the program does not prefetch images embedded in a prefetched page except for image maps. Instead, they are downloaded only when the user actually requests the page. The ongoing prefetch procedure is stopped immediately whenever a new request is issued by the user, unless the requested file is the one being prefetched. This results in some files not being transmitted completely. If the system is able to continue an interrupted transmission, it allows us to make better use of these partial documents.

One interesting thing we found from using this program is that users tend to go to pages which have been prefetched, even though they would not have been interested had these pages not been prefetched. This is because prefetched files are fast to load, which means it does not take the user much time if the page is not interesting. Moreover, if prefetching is carried out based on access probabilities from the server, the user may read them out of curiosity, since prefetched pages are presumably popular among other users according to our prediction algorithm.

Prefetching and Other Internet Technologies

In addition to prefetching, several other techniques have been proposed to improve network efficiency and reduce delay. Among them, the most popular ones are caching and pushing. In this section, we compare them with prefetching.

Caching and Prefetching

Caching and prefetching are operations complementary to each other. This includes not only caching requested files at the user's machine in case the user needs them again, but also caching the files at a proxy server or at Web caches distributed in the network so that they may be used to satisfy requests from other users. Files that are available on the local disk and are up to date need not be prefetched. In addition, prefetch requests may be satisfied by files cached in the network instead of being retrieved directly from the server. Reference [13] gives an excellent overview of current caching solutions.

However, caching alone is not sufficient for providing fast file access in many situations. Several recent studies show that although the hit rate resulting from caching can be as high as 50 percent or more, the reduction in delay is generally significantly lower, while prefetching can potentially achieve better performance especially when applied in combination with caching [14, 15]. Furthermore, pages transmitted due to prefetch requests can be cached in the network for future accesses from other users, which may help improve the cache hit ratio.

Prefetching and Pushing

Prefetching and pushing bear many similarities in the sense that both try to obtain files in advance before the user requests them. One advantage of real pushing is the same as that of multicasting, which is that if the same information is requested by several users, the bandwidth usage may be saved by sending out the information just once instead of sending one copy to every single user [16]. However, for Web browsing real pushing is hard to implement due to the diversity of users' needs in terms of content, as well as time differences in when users need to receive the information. Not surprisingly, the push functions included in the latest version of Internet Explorer (IE) and Netscape are actually "client pulling"[17]. In other words, the client machine is the active party. It automatically requests files periodically from the server at times set up by the user. Compared to pushing information all the way to the user's machine, push caching is in fact a more efficient way to save bandwidth [16].

From the point of view of retrieving information before it is requested, pushing (i.e., client pulling) is more limited than prefetching for the following reasons. First, pushing relies on the user predicting what he will need and when. However, in Web browsing the user's behavior is often highly unpredictable. In other words, users frequently access pages they had no intention of visiting; thus, we cannot expect them to accurately predict which pages should be pushed. With real-time online prefetching, popular pages may always be prefetched based on access probabilities from the client or the server as soon as the user starts visiting a server. Second, in the case of pushing, even for pages visited frequently a user has to either waste bandwidth by having lots of unneeded information pushed to his machine, or indicate in great detail what he wants; for example, "only push sports news about football, not news about hockey," which can become quite tedious very quickly. For prefetching, our simulation results discussed earlier show that access probabilities from the client can predict the user's behavior very accurately for pages visited frequently. The last issue is timing. Pushing a file every time it is updated is certainly not desirable. The best approach is probably to let the user schedule when and how often files should be pushed, as is done with IE and Netscape. The problem is, by the time the user reads the file, it might have become out of date and need to be downloaded again. Or, if the user checks the file before the scheduled push time, again, he/she would still have to wait for the file to be transmitted. However, with prefetching a file may be retrieved if and only if the user is browsing and its access probability is greater than the server's prefetch threshold.

It is likely that caching, prefetching, and pushing will coexist in the future Internet. While caching speeds up repeated requests to the same file from one or multiple users, pushing saves time for those pages that are accessed regularly by users, and prefetching, which can potentially reduce the average access delay to almost any server based on access probabilities from either the server or the client, will fill in the gap between caching and pushing, and will sometimes achieve better performance. If pushing may be scheduled at off-peak hours and files are not updated very often, it may have less impact on the network than real-time prefetching.

Conclusion

Prefetching is one effective technique for reducing Internet access delay. In this article we describe an adaptive network prefetch scheme and show how it may be applied to a mobile environment. At the center of our prefetch scheme are the prediction and threshold modules, which compute the access probabilities and prefetch thresholds, respectively. The access probabilities indicate the likelihood of the user accessing certain links. Depending on whether the user has visited a page often enough, the access probabilities of the links on the page may be computed based on the user's own access history, or the access history of other users at the server. The prefetch threshold is a function of system capacity and load, as well as network and time cost. We find that it is more cost effective to prefetch a file if and only if its access probability is greater than its server's prefetch threshold.

In a mobile environment, a user may connect to different networks at different times. Since the prefetch threshold is computed based on parameters including network capacity and network cost, our prefetch scheme adapts naturally to the network to which the user is currently connected. Furthermore, for a given ρ and α_T, the prefetch threshold is only affected by the product of α_B and b. Another type of mobility support is to allow a user to be able to access a limited number of network files when he is disconnected from the network. To provide this support, we extend the definition of access probability to pages that are not linked directly. In this way, a group of files with high access probabilities from the initial pages indicated by the user may be prefetched before he is disconnected from the network. When the user is connected to a slow link, prefetching may be incorporated with techniques such as sending the distilled images or the difference between two versions, to reduce the delay and cost. Prefetching may also be used together with caching and pushing to further improve overall system performance.

References

[1] Z. Jiang and L. Kleinrock, "An Adaptive Network Prefetching Scheme," *IEEE JSAC*, Apr. 1998, vol. 16, pp. 358–68.

[2] A. Bestavros, "Speculative Data Dissemination and Service to Reduce Server Load, Network Traffic and Service Time for Distributed Information Systems," *Proc. ICDE '96*, New Orleans, LA, Mar. 1996.

[3] V. N. Padmanabhan, J. C. Mogul, "Using Predictive Prefetching to Improve World Wide Web Latency," *Comp. Commun. Rev.*, July 1996, pp. 22–36.

[4] T. W. Yan, M. Jacobsen, H. Garcia-Molina, and U. Dayal, "From User Access Patterns to Dynamic Hypertext Linking," *Comp. Networks and ISDN Sys.*, May 1996, vol. 28, pp. 1007–14.
[5] M. Banatre *et al.*, "Providing Quality of Service over the Web: A Newspaper-based Approach," *Comp. Networks and ISDN Sys.*, vol. 29, Sept. 1997, pp. 1457–65.
[6] H. Sakagami and T. Kamba, "Learning Personal Preferences on Online Newspaper Articles from User Behaviors," *Comp. Networks and ISDN Sys.*, vol. 29, Sept. 1997, pp. 1447–55.
[7] L. Kleinrock, *Queuing Systems Vol 2: Computer Applications*, New York: Wiley, 1975.
[8] H. Balakrishnan *al.*, "Analyzing Stability in Wide-Area Network Performance," *Proc. of ACM SIGMETRICS Conf. Measurement & Modeling of Comp. Sys.*, Seattle, WA, June 1997.
[9] S. Seshan, M. Stemm, and R. H. Katz, "SPAND: Shared passive network performance discovery," *Proc. USENIX Symp. Internet Technologies and Sys.*, Monterey, CA, Dec. 1997, pp. 135–46.
[10] T. W. Bickmore and B. N. Schilit, "Digestor: Device-Independent Access to the World Wide Web," *Proc. 6th Int'l WWW Conf.*, Santa Clara, CA, Apr. 1997, pp. 1075–82.
[11] A. Fox and E. A. Brewer, "Reducing WWW Latency and Bandwidth Requirements via Real-Time Distillation," *Proc. 5th Int'l. WWW Conf.*, Paris, France, May 1996, pp. 1444–56.
[12] G. Banga, F. Douglis, and M. Rabinovich, "Optimistic Deltas for WWW Latency Reduction," *Proc. USENIX 1997 Tech. Conf.*, Anaheim, CA, Jan. 1997.
[13] M. Baentsch *et al.*, "World Wide Web Caching: The Application-Level View of the Internet," *IEEE Commun. Mag.*, June 1997, pp. 170–78.
[14] F. Douglis, A. Feldmann, B. Krishnamurthy, and J. C. Mogul, "Rate of change and other metrics: A live study of the World-Wide Web," *Proc. USENIX Symp. Internet Technologies and Sys.*, Monterey, CA, Dec. 8–11, 1997, pp. 147–58.
[15] T. M. Kroeger, D. D. E. Long, and J. C. Mogul, "Exploring the bounds of Web latency reduction from caching and prefetching," *Proc. USENIX Symp. Internet Technologies and Sys.*, Monterey, CA, Dec. 8-11, 1997, pp. 13–22.
[16] J. S. Gwertzman, "Autonomous Replication Across Wide-area Internetworks," Thesis, Harvard College, Cambridge, MA, 1995.
[17] T. Spangler, "Push Servers Review," *PC Mag.*, June 10, 1997, pp. 156–80.

Additional Reading

[1] 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.
[2] J. Griffioen and R. Appleton, "Reducing File System Latency Using a Predictive Approach," *Proc. Summer 1994 USENIX Conf.*, Boston, MA, June 1994, pp. 197–207.
[3] G. H. Kuenning and G. J. Popek, "Automated Hoarding for Mobile Computers," *Proc. 16th ACM Symp. Op. Sys. Principles*, St. Malo, France, Oct. 5–8, 1997.
[4] H. Lei and D. Duchamp, "An Analytical Approach to File Prefetching," *Proc. 1997 USENIX Annual Tech. Conf.*, Anaheim, CA, Jan. 6–10, 1997.
[5] B. N. Schilit *et al.*, "TeleWeb: Loosely Connected Access to the World Wide Web," *Proc. 5th Int'l. WWW Conf.*, Paris, France, May 1996, pp. 1431–43.

Biographies

ZHIMEI JIANG (jiang@cs.ucla.edu) received a B.S. degree in computer science from Nankai University, China, in 1992. Since 1994 she has been working toward a Ph.D degree in computer science at the University of California at Los Angeles. Her current research interests are in performance analysis, mobile computing, video transmission, and Internet caching and prefetching.

LEONARD KLEINROCK [F] (lk@cs.ucla.edu) has been a professor of computer science at the University of California, Los Angeles, since 1963. He received his Ph.D degree from MIT. His research interests focus on performance evaluation of high-speed networks and parallel and distributed systems. He has had over 200 papers published and is the author of six books. He is a member of the National Academy of Engineering, and a Guggenheim Fellow.