Potential and Limits of Web Prefetching Between Low-Bandwidth Clients and Proxies

Quinn Jacobson
Pei Cao

Technical Report #1372

April 1998

Potential and Limits of Web Prefetching Between Low-Bandwidth Clients and Proxies

Quinn Jacobson
Department of Electrical and Computer Engineering
University of Wisconsin-Madison
qjacobso@ece.wisc.edu

Pei Cao
Department of Computer Science
University of Wisconsin-Madison
cao@cs.wisc.edu

Abstract

The majority of the Internet population access the World Wide Web via dial-up modem connections. Studies have shown that limited modem bandwidth is the main contributing factor to the Web access latency perceived by the users. In this paper, we investigate one approach to reduce the user-perceived latency: pre-pushing from the proxy to the browsers. The approach takes advantage of the idle time between user Web requests and uses prediction algorithms to predict what document a user might reference next. It then relies on proxies to send ("push") the documents to the user. Using existing modem Web access traces, we evaluate the potential of the technique at reducing user latency, examine the design of prediction algorithms and measure their accuracy as well as overhead, and evaluate the latency reduction of pre-push schemes using the algorithms.

Our results show that with perfect predictors, proxy-based Web pre-pushing with a 256K-byte pre-push buffer at the browser side can reduce latency by over 20%. Our results also show that prefix-based prediction algorithms works well for predicting user behavior. Proxy-based web pre-pushing driven by real prediction mechanisms can reduce latency by nearly 10%.

1 Introduction

Ever since the World-Wide Web becomes World-Wide Wait, reducing client latency has been among the primary concerns of the Internet research and development community. For a majority of the Internet population, who access the World-Wide Web through modem connections, the low modem bandwidth is a primary contribution to the client latency. For example, studies using commercial ISP traces shows that even if a Web proxy caching system is employed and has perfect performance, the client latency can only be reduced by 3% to 4% [7]. Other studies have shown that when image distillation techniques is used to reduce the document sizes, the client latency can be improved by a factor of 4 to 5, further demonstrating that the slow modem links are the bottleneck.

In this paper, we study one technique to reduce latency for modem users: prefetching between caching proxies and clients. The proxy, being exposed to the Web accesses of multiple users, can often predict what documents a user will access next. The modem link to the user often has idle periods as the user is reading the current Web document. Thus, a proxy can utilize the idle periods to "push" documents to the user's machine. If the user accesses any of the pushed documents, the latency perceived by the user is reduced. Since the proxy pushes the documents to the user side instead of the client fetching the documents from the proxy, we called this technique *proxy-side pre-push.* In the rest of the paper, we use the terms "pre-push" and "prefetch" interchangably.

We focus on the potential, limits and prediction mechanisms of the technique in reducing client latency. Using traces of modem users' Web accesses, we first study the upper bound on latency reduction from pre-push, assuming a perfect predictor at the proxy side. We analyze how this upper bound is determined by the inherent idle time distribution between requests from the same user, modem bandwidth, and client-size buffers.

We then study the prediction mechanisms for the pre-push approach. We investigate a family of prediction algorithms that are based on the Prediction by Partial Matching (PPM) data compression algorithm. We analyze the accuracy and bandwidth wastes of the algorithms as the algorithms' parameters vary. We also evaluate the impact of practical constraints such as pruning the history structure and interacting with the client-side buffers.

Finally, we evaluate the latency reduction under the prediction algorithms using trace-driven simulation. Our simulator takes into account of proxy latency and user request sequence. Our results show

that with perfect predictors proxy-based Web prepushing can reduce user observed latency by over 20% and with real predictors can reduce user observed latency by nearly 10%.

2 Related Work

Prefetching can be applied in three ways in the Web contexts: between Web servers and clients, between Web servers and proxies, and between proxies and clients.

Early studies have mostly focused on prefetching between Web servers and clients, since relatively few proxies were deployed then. Padmanabhan and Mogul [17] analyze the latency reduction and network traffic of prefetching using Web server traces and trace-driven simulation. The prediction algorithm they used is also based on the PPM data compressor, but with order of 1. The study shows that prefetching from Web servers to individual clients can reduce client latency by as high as 45%, at the expense of doubling the network traffic. Bestavros el al [2] presents a model for speculative dissemination of World Wide Web documents. The work shows that reference patterns observed at a Web server can be used as an effective source of information to drive prefetching. They reached similar conclusions in [17]. Finally, Cuhan and Bestavros [5] uses a collection of Web client traces and studies how effectively a user's future Web accesses can be predicted from his or her past Web accesses. They show that a number of models work well and can be used in prefetching.

We found that these early studies do not answer our questions about the performance of prefetching between clients and proxies. First, the studies investigate prefetching between user clients and Web servers; caching proxies were not considered or modelled. There are reasons to believe that a proxy's capability to predict users' Web accesses is different from the server's. Since a proxy only sees the Web requests to its users, its ability to predict is limited by the smaller sampling size. On the other hand, a proxy can predict documents across Web servers, increasing its chances at reducing client latency. Second, most of the studies use Web server traces which do not accurately capture user idle time. From a particular Web server's point of view, a user's accesses to other Web servers appear as idle time. Thus, it is difficult to estimate from Web server traces the true user idle time that can be exploited to push documents to users over the modem lines. Lastly, Cuhan and Bestavros [5] uses the trace of client requests that are not filtered by the browser cache; in practice, a proxy only sees requests after they are filtered by the browser cache. The presence of a browser cache often diminish the effect of prefetching: a predicted document may already be in cache and there is no need to prefetch it. Thus, it is not clear whether the conclusions from this particular study are applicable to our investigation.

As proxies become more widespread, a number of studies investigated prefetching between Web servers and proxies [14, 16, 10, 11]. Kroeger el al [14] investigates the performance limits of prefetching between Web servers and proxies, and shows that combining perfect caching and perfect prefetching at the proxies can at best reduce the client latency by 60% for relatively high-bandwidth clients (i.e. those that accesses the Web through a LAN instead of modems). Markatos and Chronaki [16] proposes that Web servers regularly push their most popular documents to Web proxies, which then push those documents to the clients. They evaluate the performance of the strategy using several Web server traces and find that the technique can anticipate more than 40% of a client's request. The technique requires cooperation from the Web servers. Furthermore, since the server traces do not capture each user's idle time, the study did not evaluate the client latency reduction resulted from the technique. Wcol [10] is a proxy software that prefetches documents from the Web servers. It parses HTML files and prefetches links and embedded images. The proxy, however, does not pre-push the documents to the clients. Finally, Gwertzman and Seltzer [11, 12] discussed a technique called Geographical Push-Caching where a Web server selectively sends its documents to the caches that are closest to its clients. The focus of the study is on deriving reasonably accurate network topology information and using the information to select caches.

In terms of prefetching between proxies and clients, Loon and Bharghavan [15] has proposed the same idea as we described here. However, the study did not address the potential and limits of the approach, or which prediction algorithms are the most successful. Rather, the study presents a design and an implementation of a proxy system that performs the prepushing. In this paper, we use recent, large-scale modem client traces to explore the design space and evaluate the tradeoffs in pre-pushing.

Outside the Web contexts, prefetching has been studied extensively in file systems and memory systems. Several studies investigate application-controlled prefetching in the file system contexts [3, 4, 13, 19], where an application gives prediction of what it might access next. Other studies also investigate the speculative approach to file prefetching [9, 20, 18]. The prediction algorithms used there are also derived from

data compressors, while the compressor and the parameters vary.

Most of the algorithms, including the one studies here, are inspired by an important early study by Krishnan and Vitter demonstrating the relationship between data compression and prediction [21, 6]. Their study shows that if the source behind a reference string can be modelled as an M-order Markov source, then the compressor-driven prediction algorithm will converge to the optimal predictor given a long enough reference string. Our algorithms extend the algorithms described in [21] and [6] by considering prediction depth of more than 1, that is, not only predicting which documents might be used next, but also predicting which documents will be used after the immediate next ones. Our experience show that for best performance, the prediction depth needs to be larger than 1.

3 Proxy-Side Pre-Pushing

3.1 Overview of Scheme

The basic assumption behind proxy-side pre-pushing is that users have idle times between requests (because users often read some parts of one document before jumping to the next one), the proxy can predict what documents users will access in the near future, and the proxy can take advantage of the idle time to *push* the documents to the users. A proxy observes the behavior of many users' Web accesses and from these may be able to learn common access patterns and use this to predict what document a user is likely to access in the near future. The general configurations of the proxy and clients are shown in Figure 1.

When a user has a request it can not service with its own cache it looks into the prefetch buffer. If the document is in the prefetch buffer the user sends a small update message to the proxy that it used the page. If the document is not in the prefetch buffer the user sends a request for the page to the proxy. There is a third possibility, the document the user want may be in the process of being tranferred into the prefetch buffer, in which case the user can get the document as it is being downloaded. In this last case the user waits until the document is downloaded and then sends an update to the proxy.

When the proxy receives a request from a user it immediately starts fetching the document from the correct content server (if the document is not in its own web cache) and sends the document to the user. A user request may preempt a document currently being pushed to the user side. In this case the partially pre-pushed document at the user end is discarded. After the proxy services a request, or when it receives an update, it updates the history structure for the user's last access. The proxy then predicts what pages, if any, are candidates for pre-pushing. It then orders these pages by their probabilities and starts sending them. When the next user request or update comes in the proxy clears the list of candidate pages it is sending and recomputes the list of candidate pages based on the new information.

A proxy will only pre-push documents to a user if the documents are in the proxy's web cache. This restricts the mechanism from producing extra traffic on the Internet. Note that prepushing can still cause increased traffic on the dial-up connection between the proxy and the users.

3.2 Prefetch Buffer

The prefetch buffer is implemented as a cache. We model various size prefetch buffers from 256 Kilobytes to 16 Megabytes. The replacement policy of the user's prefetch buffer is a combinations of two different policies. First, the buffer implements a LRU policy for the pages in the buffer. During the prefetching sequences the buffer management is slightly modified. The first document of a prefetch sequence is ordered as the most recently used (highest priority), the second document is ordered as the second most recently used and so on. This ordering is implemented because documents are sent in the order of decreasing probability.

A prefetched document should not replace a higher probability document of the same prefetch sequence. The proxy attempts to detect this case and stop prepushing when it occurs, but it uses a simple mechanism that can not detect all cases. When the prefetch buffer is full and there are more pages to prepush the least-recently used (lowest priority) document in the buffer is checked to see if it is a higher probability document of the same prefetch sequence. If the new document is larger than the lowest priority document it may force multiple documents out of the buffer, some of these additional documents may be of a higher probability than the new document.

3.3 Traces and the Simulator

The traces used in this study is the HTTP traces gathered from the University of California at Berkeley home dial-up populations from November 14, 1996 through November 19, 1996. For a detailed description of the trace see [8]. The traces are collected at a packet sniffing machine placed at the head-end of the UC Berkeley HomeIP modem bank. Thus, it cor-

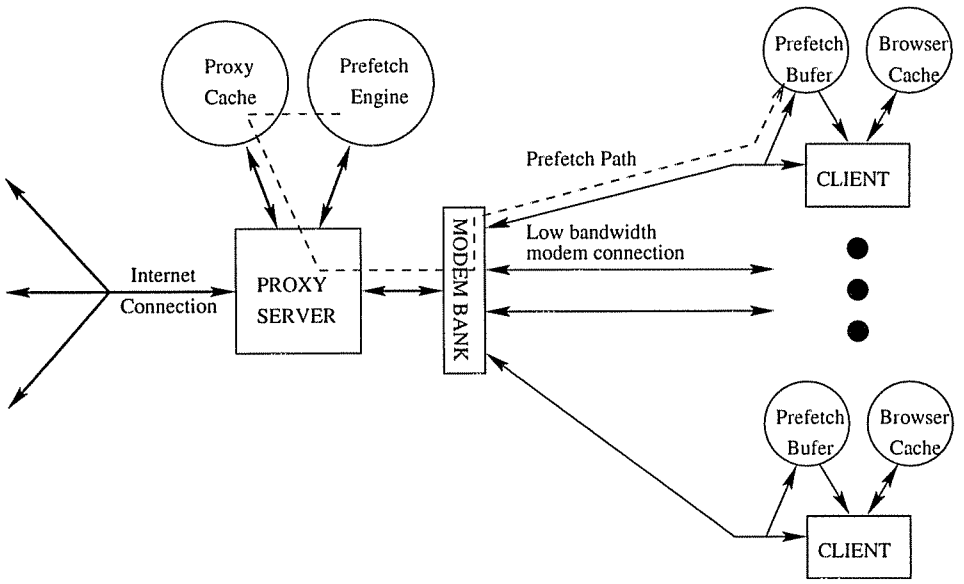

Figure 1: System Overview.

responds to requests that would be seen by a proxy between the modem clients and the Internet.

The simulator uses the timing information in the traces to estimate the latency seen by the modem client. The trace includes, for each request, the time the client made the request (T_0), the time at which the first byte of the Web server response was seen at the head-end machine (T_1), and the time at which the last byte of the Web server response was seen at the head-end machine (T_2). We assume that the transfer of the document from the Internet to the head-end machine and the transfer of the document from the head-end machine to the modem client can be overlapped, and since packets typically arrive at the modem bank faster than they can be sent to the modem clients [7], we estimate the client-seen latency to be ($T_1 - T_0$) plus the document size divided by the modem bandwidth [1].

Except in our analysis of user idle times, we always assume the existence of a proxy between the modem clients and the Internet. The proxy is assumed to have a cache size of 16GB, which for our traces results in no cache replacement. We do not simulation the latency of fetching a document from the proxy cache in detail since in practice there are many different proxy software that has quite different performance characteristics. Rather, we assume a 10ms fixed delay in fetching a cached file from disk in order to send to the client. We assume that the proxy has one disk and the fetching of different files is serialized.

[1] We are planning to improve our latency estimate and incorporate new results for the final version.

The simulator models a prefetch buffer associated with each unique client. The prefetch buffer resides on the local machine of the client. The prefetch buffer is assumed to be implemented in main memory and its access time is considered insignificant. From the traces we can not recognize when a user disconnects and reconnects to the modem bank. We approximate this behavior by assuming that a user connection idle for more than 30 minutes corresponds to a user disconnecting. When a user is idle for more than 30 minutes their prefetch buffer is cleared and any state the proxy has been maintaining for that user is discarded.

For most of the studies we assume that the proxy has perfect knowledge of the state of each user's prefetch buffer. In a real system this could be achieved by either the proxy maintaining state on all active users' buffers or by having the proxy query the user when it needs information. This query communication would most likely be very short messages whose latency would be insignificant as compared to document transfer times. We will also show that prepushing can be implemented with the proxy having no knowledge of the state of the user's prefetch buffer.

4 Potential of Proxy-Side Pre-Pushing

Before studying the design of the prediction mechanisms, one must answer the question: assuming perfect predictors, what's the maximum latency reduction possible using this technique? The upper bound

will determine whether pre-pushing is worthwhile.

The upper bound is determined by a number of factors, including:

- the inherent idle time distribution in a user's Web accesses. This determines how much time can be used for pushing documents to the user side.
- the modem bandwidth between the proxy and the user.
- the lookahead of the predictor, that is, how many requests can be predicted at a time. In this section we assume that the prediction are always perfect.
- the size of the user-side pre-push buffer. Pre-pushed documents are saved at the client machine. The total amount of storage that can be devoted to storing the prepushed documents is limited.

Figure 2 shows the idle time distribution observed in the UCB traces. Here we estimate idle time by looking at each user's requests, and calculate the difference between the estimated end-of-transmission of one request to the start of the next request. Differences of less than 1 second are presumed to be for concurrent transfers. The figure show that about 60% of the requests are preceded by 2 to over 120 seconds of idle time, indicating that there are plenty of opportunities to take advantage of the idle time to reduce client latency.

Our simulation assumes a perfect predictor at the proxy side; that is, at any time, it can predict the next k requests for a user for any k. Furthermore, the simulation assumes that at each user end a pre-push buffer exists to hold pre-pushed pages from the proxy. The pre-push buffer is separate from the user browser cache. User Web accesses that hit in the browser cache is hidden from the proxy However, we assume that for every accesses that hit in the pre-push buffer, the proxy receives a notification message.

Our simulation assumes that a document can only be pushed if it has been seen before in the trace. Obviously, unless Web servers provide prefetch-hints to proxies, this is the limitation in reality. We further assume that the proxy has infinite cache space, and proxy cache documents on disks, and for every document that is pushed, there is a 10ms latency to fetch the document from disk first.

Finally, the simulation assume that the buffer is flushed if the user's idle time exceed 30 minutes. This is because in reality, the idle time signifies a new session and we assume it to be impossible to predict across the sessions.

4.1 Performance Metrics

We are primarily interested in the following performance metrics for our prediction algorithms:

- *Latency Reduction*: the percentage reduction in the user-visible latency from the prepush technique. The latency reduction comes from two sources:
  - latency hidden: the user-visible latency that is either avoided because the requested document is already in the prefetch buffer (completely hidden), or reduced because it is being pre-pushed (partially hidden).
  - contention avoidance: the reduction in user latency that is due to the document being transfered to the user earlier than in the no-prefetching case. When multiple documents are being sent to the user, the transfers share the limited modem bandwidth. If prefetching can make one of the document transfers happen earlier, then the rest of the document transfers all complete quicker and result in reduction in user-visible latency.
- *Wasted Bandwidth*: the amount of bytes that are pre-pushed from the proxy to the client but are not read by the client. That is, it is the sum of the sizes of files that are pre-pushed from the proxy to the user but are replaced from the user-side prefetch buffer without ever being accessed by the user. In our following figures we show the ratio between bandwidth wasted and the total bytes accessed by the users.
- *Request Savings*: the number of time that the user requests a document that is already in the prefetch buffer, or is being sent from proxy to the user. We can further characterize the request savings into the following three categories:
  - prefetched: the document is in the prefetch buffer, and and the user is accessing it for the first time since it is in the prefetch buffer.
  - cached: the document is in the prefetch buffer, but this is not the first time that the uses has accesses it since it has been in the prefetch buffer.
  - partially prefetched: the document is still being sent from the proxy to the user.

  The reason for separating the request savings into the three categories is that we would like to separate the caching effect of the prefetch buffer

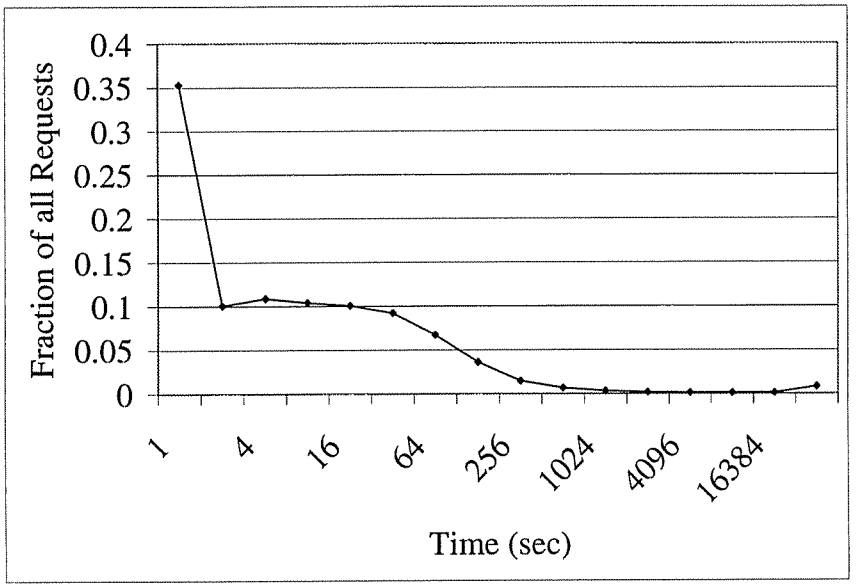

Figure 2: Distribution of idle time between user requests.

from the prefetching effect. The prefetch buffer also acts as an extended browser cache for the user, and to understand the latency reduction due to the prediction algorithm, it is important to separate the two effects.

Latency reduction is the primary goal of the prepush scheme. Bandwidth wasted measure the extra bandwidth consumed by the algorithm. Unlike in the wide-area network, wasted bandwidth on the modem line can be tolerated: the modem line would stay idle anyhow. For most users who do not initiate other network transfers (such as ftp) while Web-surfing, the wasted bandwidth has virtually no effect. Finally, we need to investigate request savings to understand the source of latency reduction.

4.2 Limit Study Results

Figure 3 shows the fraction of requests that are serviced by the prefetch buffer for users connected to the proxy via 56Kbps modems, under different lookaheads (256, 8 or 4) and with different prefetch buffer sizes (16MB, 2MB or 256KB). Around 60% of user requests (that miss in the browser cache) are serviced by the prefetch buffer. In the case of the 256 look ahead and large buffer every document that is not seen for the first time is serviced by the prefetch buffer. Even with the minimal look ahead there is opportunity to prefetch nearly every request. The requests serviced by the prefetch buffer are broken into three categories. The first category is the cache affect. This is the case when a requests is for a document in the prefetch buffer that has already been used at least once since being brought into the buffer. This occurs in limited cases where the browser either doesn't have a cache, or the cache doesn't use the LRU replacement policy. The second category are requests that first use a document that has been brought into the buffer. This is the case of true prefetching. The third category is a special case of the second where the document the request is for is in the process of being brought into the buffer when the user requests it. In this last case the user sees a partial reduction in latency.

Figure 4 shows, for the same lookahead and buffer size combination, the fraction of requests serviced by the prefetch buffer, the average reduction in latency seen by the user and the extra bandwidth consumed by prefetching. The latency reduction is considerably smaller than the fraction of requests serviced by the prefetch buffer. The major reasons for this is that prefetching can only be used on requests that hit in the proxy cache, these requests on average have a smaller latency than requests that require accessing a server over the Internet. We also see that there is wasted bandwidth even with perfect prediction and in one case it is significant. This is an artifact of the prefetch buffer management where we may replace a document with one that has a lower prefetch probability (occurs later). This is discussed in Section 3.2. In the case of the 256 look ahead and the small buffer there is extreme pressure on the prefetch buffer and we see that this effect is significant, in the other cases the impact is minimal.

The latency reduction for the long look ahead is significantly larger than the reduction for more mod-

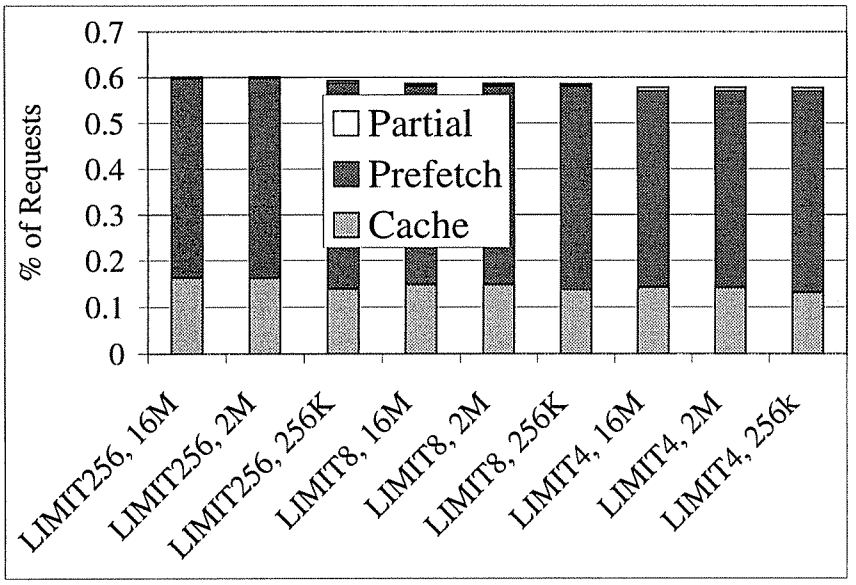

Figure 3: Fraction of requests that are serviced by the prefetch buffer (the first number specifies the lookahead, the second number specifies the prefetch buffer size).

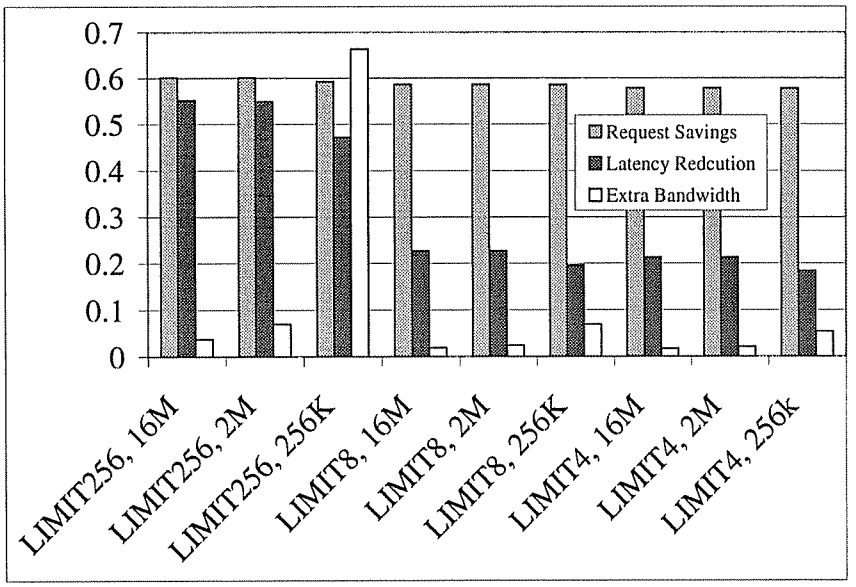

Figure 4: Request Savings, Latency Reduction and Extra Bandwidth.

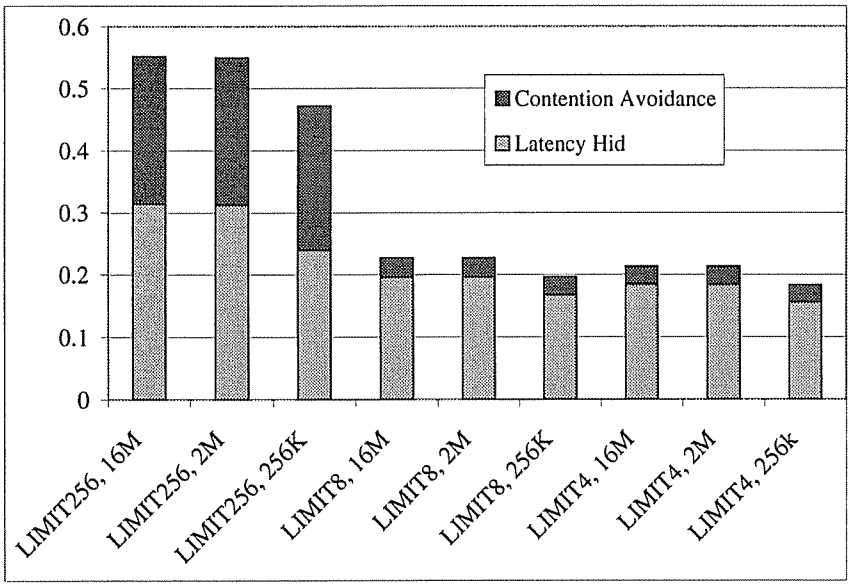

Figure 5: Breakdown of latency reduction.

erate look ahead even though the fraction of requests serviced by the prefetch buffer varies only slightly. The reason for this can be seen when we break the latency savings down to user latency hidden and contention avoidance, Figure 5. The user latency hidden is calculated by determining the amount of time the user would have waited for a document to be fetched if it had not been in the prefetch buffer. The contention avoidance is calculated by taking the total latency observed when there is no prefetching and comparing it to the total latency observed with prefetching plus the latency that was directly hidden by prefetching. We see that the long look ahead has significantly larger savings due to contention avoidance. This is logical because the ability to reduce contention is strongly dependent on how far you can move requests in order to capitalize on idle periods.

The latency directly hidden by prefetching is a direct result of prefetching. Some contention avoidance will occur from prefetching, but it is not clear that some, or even most, of the contention avoidance is an artifact of the trace driven nature of the simulation. Many requests are the direct result of a directive encoded into a preceding request. If the first request was serviced earlier the subsequent request would be issued earlier, but our simulations due not take this into account. In the case that there is a long chain of overlapping requests (each caused by the previous) our simulator may be able to move the first one earlier and it will observe that the requests now no longer overlap.

In the case of the moderate look ahead there is only minimal latency reduction due to contention avoidance and most of the user latency reduction is due to the direct hiding of latency due to prefetching. The next section will show that in for real predictors the contention avoidance is also a minimal impact.

5 Prediction Algorithms

5.1 Overview

The prediction algorithms in this study observe patterns from past accesses from all the clients to predict what each client might access next. The patterns we capture are in the form of "a user is likely to access URL B right after he or she accesses URL A." Clearly, only accesses from the same user should be correlated; accesses from different users are not related and may arrive at the proxy in any order.

The actual prediction algorithm is based on the Prediction-by-Partial-Match (PPM) data compressor [1, 6]. A m-th order predictor uses the context of the past m references to predict the next set of URLs. The algorithm maintains a data structure that keeps track of the URLs following another URL, following a sequence of two URLs, and so on up to a sequence of m URLs. In the context of the past m references, the immediate past reference, the past two references, the past three references, etc. are matched against this data structure to produce the set of URLs that are likely to be accessed next. The URLs are sorted first by giving preferences to longer prefixes, and then by giving preferences to URLs with higher probability within the same prefix.

There are two aspects of the algorithm that are

particular to the Web proxy contexts. First, the algorithms use patterns observed from all users' references to predict any particular user's future references. On the one hand, this may reduce the accuracy of the prediction because one user's patterns are used to predict a different user's accesses. On the other hand, this increases the number of times the algorithms can make predictions because the sampling size is increased. We feel that in the context of prepushing, the second factor is likely to be more important. A good browser cache should absorb most of the repeated accesses from the user, and it is important to predict the first-time access to a document from a user (i.e. a modem client).

Second, the algorithms separate embedded objects from non-embedded objects, and treat a document and its embedded objects as the same entity. That is, if the algorithm decides to push the document to the client, it pushes the embedded objects as well. The reasons for doing so are two fold. First, embedded objects have an access probability of 1 following the access of the document, and finding the embedded objects in practice is easy — the proxy simply has to scan the document. Second, embedded objects are often fetched concurrently over four connections to the proxy, and the order of request arrival at the proxy can be random. Separating them out can simplify the data structures keeping track of the observed patterns.

Since the trace does not indicate which URLs are embedded in other documents, we estimate the information by treating all ".gif" files that are referenced after the reference of one ".html" file and before the reference of another ".html" file to be the embedded objects in the first ".html" file. This estimate errors on the side of overclassifying ".gif" files as embedded objects, and we are working on refining the estimate.

5.2 History Structure

The implementation uses a history structure, which maintains information about the order pages were seen in. There is one history structure shared and updated by all the users of a given proxy. Gif files are not included in this history, as they are considered to be sub parts of html pages.

The history structure is a forest of trees of a fixed depth K, where $K > m$. This history encodes all dynamic sequences of accesses (by any one user) up to a maximum length K. One root node is maintained for every page seen, and in this node is a count of how often this page was seen. Directly below each root node are all pages ever requested immediately after the root page and a count of how often the pair of requests occurred. The next level encodes all series of three pages and a count of how often this particular sequence of three pages occurred. This is continued to the K-th level.

Every time a user makes a request for a page the history structure is updated. For each user there is a list of the last K pages the user requested (this can be maintained as a list of pointers into the history structure). The update involves incrementing counters and possibly adding new nodes to the history structure. Each update involves changing one node at each level of the history structure. Figure 6 shows an example of the history structure. In this example $K = 3$, and the structure is being updated after a user accesses page C following accessing pages A and B. The sequence ABC is updated, with the counters for A and B and C incremented. The sequence BC is updated and so is the sequence C.

Encoded into the history structure is the observed relative probability of what page will follow a given sequence of references. This can be calculated by following a given sequence through the history structure and then looking at the relative counts of all nodes immediately following that sequence. The probability of a node following another node is the ratio of the count of the child to the count of the parent.

5.2.1 Aging and Pruning

There are a couple undesired effects if the history structure is allowed to continuously grow. First, the structure will grow very large which makes implementing the prefetch engine more costly. Second, users access patterns change over time and the history should dynamically adapt. Both of these issues are solved by what we refer to as aging.

All the nodes in the tree have a count, and we add to them a time of the last update. Periodically, in our case every hour, we initiate a pruning of the tree. When the tree is pruned all leaf nodes in the history structure are checked. For every eight hours (the threshold time used is arbitrary) of not being updated we half the count value by shifting the value right by one. We propagate these changes up by having the count of internal nodes be equal to the count of their immediate children. When a nodes count reaches zero we delete the node.

Choosing the threshold is complicated. If we make the threshold time used for pruning to short we will remove information from the tree that is still relevant and important for making predictions. If we make the threshold to long we have the problem that old information stays around to long. Old information can be a problem because we may issue a prefix for a page based on old information, or we may fail to issue a good prefetch for a page because with the old

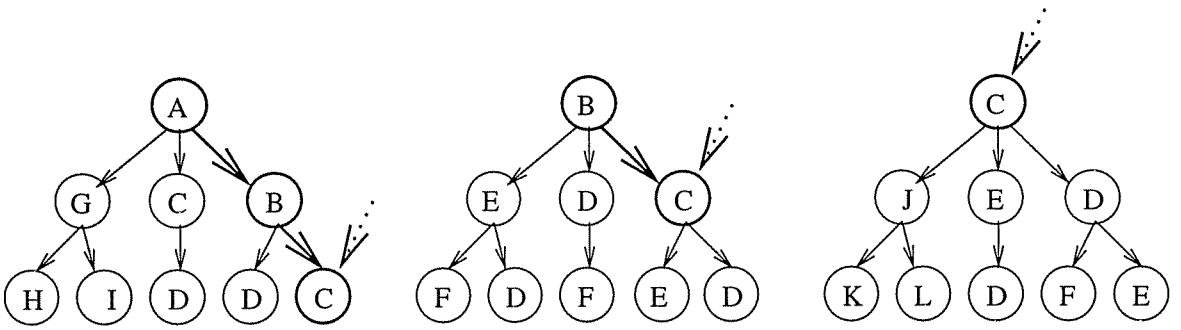

Figure 6: History Structure (being updated for user sequence A...B...C).

information the new dominate page to prefetch does not meet the prefetch threshold.

In case aging cannot remove enough history nodes to keep the structure under certain size, a "pruning" mechanism is triggered. The nodes that are least-recently traversed are replaced until the history structure is small enough.

5.3 The Predictor

The predictor uses the history structure to identify the set of pages a user is likely to access in the near future. It looks at a user's recent m accesses. For the sequence of the last l requests, where $l = m, ..., 1$, it finds the corresponding tree and the node in the history structure. The predictor then follows all paths from that node in the tree for k levels down to calculate the relative probabilities of those pages being accessed in the near future. This produces a list of candidate pages for prefetching. Each list first sorts its pages by their probabilities, then deletes the pages whose probability is below certain threshold t, where $0 \leq t \leq 1$. Finally, the lists are merged by putting lists corresponding to longer sequences first.

The final list is then sent to the proxy. The proxy pushes the documents on the lists in the order specified as long as the modem link is idle. When a new request from the user arrive, the ongoing push (if any) is stopped. A new round of prediction and push then starts again.

Clearly, there are three parameters for the prediction mechanism:

- m: the order of the predictor, that is, how many past accesses are used to predict. We call it the "prefix depth."
- l: how far down in the tree to search for eligible documents. We call it the "search depth."
- t: the threshold used to weed out candidates.

Their impacts on the predictor's performance are quantitatively analyzed in Section 6.

The algorithm is similar and yet different from previous proposed prefetching algorithms. Papadumanta and Mogul [17] studies the same algorithm, but with m always equal to 1. Pk and Vitter [6] also studies the PPM-based algorithm as one of their prefetching algorithms, but always have l equal to 1. The reason for considering algorithms with $m > 1$ is that more context can help improve the accuracy of the prediction. The reason for considering algorithms with $l > 1$ is that sometimes, a URL may not be always requested as the immediate next request after another URL, but rather within the next few requests. Thus, in some sense, our algorithm is the more comprehensive version among the existing algorithms.

6 Performance

Using the UCB traces, we study the performance of the prediction algorithm varying the parameters m, l, and t. The user-side prefetch buffers are assumed to be 2 Megabytes. The performance metrics are the same as described in Section 4.1. We also study the performance effect of limiting the size of the history structure, and that of imperfect interaction with user-side prefetch buffers.

In our first set of simulations, we assume that history structure can occupy up to 64MB, and the interaction between the proxy and the user-side prefetch buffers are as described in Section 3.2. We experimented with the prefix depth m being 4 and 1, the search depth l being 8, 4, and 1, and the threshold being 0.01, 0.10, and 0.25. We simulated two modem speeds: 56Kbps and 28.8Kbps.

The performance results assuming 56Kbps modem lines are shown in the following three figures. Figure 8 shows three bars for each algorithm configuration, illustrating the request savings, latency re-

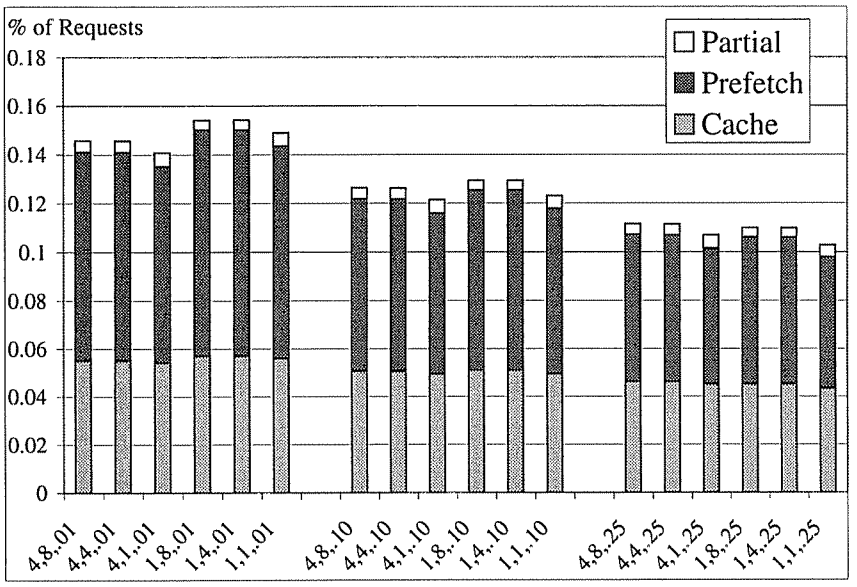

Figure 7: Fraction of requests that are serviced by the prefetch buffer (the first number specifies the prefix depth, the second specifies the search depth, the last specifies the threshold).

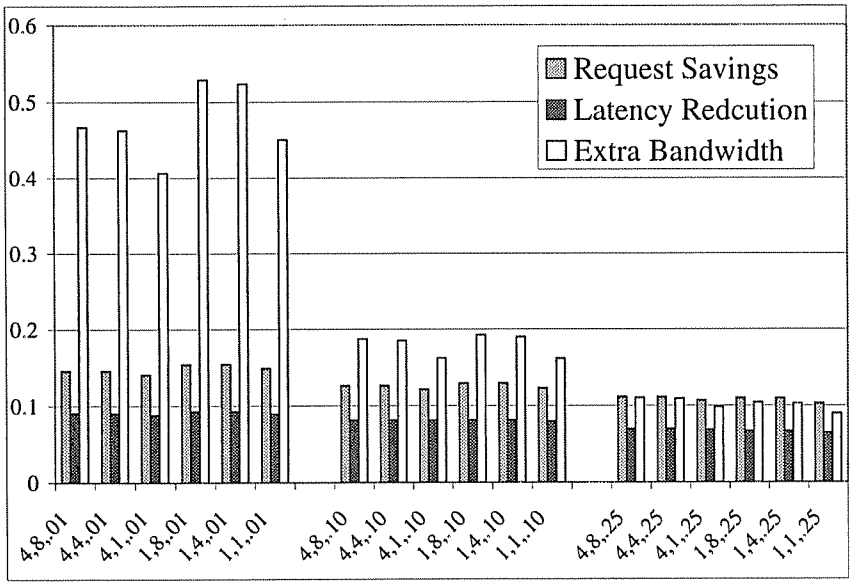

Figure 8: Requests serviced by prefetch buffer, latency savings, extra bandwidth.

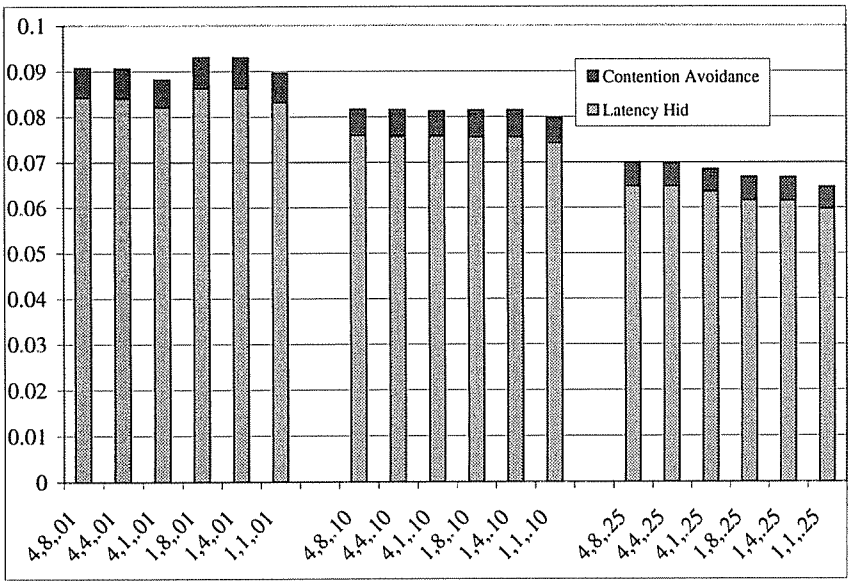

Figure 9: Breakdown of latency savings.

duction, and wasted bandwidth. Figure 9 shows the breakdown of latency reduction due to latency hidden and contention avoidance. Figure 7 shows the breakdown of requests savings in to three categories.

Figure 8 shows that for 56Kbps modem lines, the pre-pushing scheme can reduce user-visible latency by 9.3% at a cost of 52% wasted bandwidth (at $(1, 8, 0.01)$), by 8.2% at a cost of $< 20\%$ wasted bandwidth (at $(4, 8, 0.10)$ or $(1, 8, 0.10)$), and by about 7% at a cost of about 10% of wasted bandwidth (at $(4, 8, 0.25)$). Thus, there is an inherent tradeoff between the latency reduction and wasted banwidth: the more aggresive the prediction algorithm, the more chances it pushes useful documents, and the more chances it pushes the useless documents. On the other hand, a balance between the latency reduction and wasted bandwidth can be achieved. Depending on the environment and the effect of the wasted bandwidth, any one of the above configurations can be desirable.

Figure 8 also shows that the threshold t has the biggest effect on wasted bandwidth. This is to be expected since a lower threshold means more files are pre-pushed. Within the same threshold, increasing the search depth l tends to increase both wasted bandwidth and latency reduction, particularly for very low thresholds. This is because under low thresholds, higher search depth will generate a lot of candidate documents. Thus, for low threshold, if the wasted bandwidth is a concern, $l = 1$ seems to be most appropriate. However, for high threshold (e.g. 0.25), higher search depth will yield more documents that satisfy the threshold constraints and are useful, and a high search depth (for example, $l = 4$) is more appropriate. Finally, note that the performance of $l = 4$ is very close to that of $l = 8$.

Looking at the prefix depth m, we see that it has different effects depending on the threshold t. For $t = 0.01$, higher prefix depth reduces wasted bandwidth and results in slightly lower latency reduction. For $t = 0.10$, the effect of high prefix depth is minimal. For $t = 0.25$, higher prefix depth increases the latency reduction and wasted bandwidth. The reason is that for low threshold, higher m finds the documents that are more specific to the context (and therefore more likely to be references) and move them to the front of the prepush list. In other words, it improves the accuracy of the prediction. Thus, it reduces the bandwidth wastes without affecting latency reduction much. For high threshold, higher m uses more previous accesses as context information, and increase the probability of the context-specific documents because the sample size for longer contexts is smaller. Thus, more context-specific documents are made eligible for prediction.

Figure 9 shows that the latency reduction from contention avoidance is only a small percentage of the total latency reduction. This is consistent with our study of perfect predictors with small lookahead. Clearly, contention avoidance relies on predictions that are far into the future.

Figure 7 shows that there is only a small percent of requests that hit on documents that are currently being pushed, and the caching effect of the prefetch buffer counts for about 38% of the request savings. In other words, about 38% of the latency reductions observed is in fact due to the caching effect of the

prefetch buffer. This first suggests that in the UCB traces, the users are probably using browser caches that are too small. In general, the browser cache should be increased. Second, assuming that the first step in reducing user latency is to increase browser cache size, we should then study the performance of the algorithm assuming a larger browser cache. Since requests that hit in the browser cache are not reflected to the proxy, the effect of a larger browser cache on the algorithm's performance is not clear. We are currently investigating this question. Third, comparing with the results in Section 4, we see that another 2MB increase to the browser cache seems to be sufficient.

Finally, comparing the latency reductions with the Limit studies in Section 4, we see that the prediction algorithm achieves about half of the latency reduction under a perfect predictor.

Figures 10 through 12 shows the corresponding results for 28.8Kbps modems. As we can see, the latency reduction and the request savings are generally reduced by 10% comparing to those under 56Kbps modems. This is because that in many cases, there isn't enough time to prepush a document to the client. The increase of the partial prefetch requests demonstrates it. The effect of the parameters are the same as for 56Kbps modems.

Although we only show the threshold values up to 0.25, we have experimented with higher threshold values, and the trend is the same: the wasted bandwidth is reduced, the latency reduction is reduced, the better performance relies more on higher search depth and higher prefix depth.

Effects of Limiting History Sizes In our second set of simulations, we pick the algorithm configuration $(1, 1, 0.01)$ and study how its performance is affected by smaller history structures. The size of the history structure is constrained by specifying a maximum number of nodes implementing an LRU replacement. This size constraint is in addition to the normal aging. The $(1, 1, 0.01)$ configuration is chosen because it only needs two-level trees and yet yields almost the highest latency reduction. Table 1 shows the results on request savings and latency reduction. The 32MB history size is sufficient that nodes are never prematurely removed.

The results show that when the history size is reduced, the latency reductions decreases, as the algorithm no longer has access to some patterns observed in the past. However, the degradation is gradual and as more memory can be devoted to the history structure, the performance improves.

History Size	Request Savings	Latency Reduction
32MB	15%	9%
4MB	12%	7%
1MB	9%	5%

Table 1: Performance of the $(1, 1, 0.01)$ under different history sizes.

Effects of Limiting Prefetch-Buffer Interactions We also study the effect of imperfect interactions with user-side prefetch buffers on the algorithm performance. We first simulate the scenario where the proxy is unaware of the content of the prefetch buffer, and pushes documents that are already in the prefetch buffer at the user-side. The results for the $(1, 1, 0.01)$ configuration show that the performance is degraded moderately; request savings changes from 15% to 12%, and the latency reduction changes from 9% to 7%.

We then simulate the scenario where the proxy is unaware of the user accesses that hit in the prefetch buffer. That is, the proxy would not know which of its predictions are right, and which are wrong. The results show that the performance is impacted significantly. The request savings is reduced from 15% to 8%, and the latency reduction changes from 9% to 5%.

Thus, it is important for the proxy to know the accesses that hit in the prefetch buffer. In practice, sending the information to the proxy requires a small update message from the client machine to the proxy. Our results show that this update message is important and should not be omitted.

7 Limitations of Our Study

There are many limitations in our study. First, we assume fixed user request arrival times in our simulation. In practice, as requests are serviced quicker, the requests also arrive faster. Thus, the latency reduction discussed only serves as an indication of how much pre-pushing schemes can speed up user requests. Second, our calculation of client latency is merely an estimate based on the timestamps recorded in the traces and the modem bandwidth. It does not include protocol overhead and is far from accurate. However, we feel that the estimate is good enough to give us an indication of how well the pre-pushing scheme might perform.

Third, our simulator does not model the proxies accurately. In practice, the latency that a proxy incurs to fetch a document from its cache depends on many factors, including whether the document is cached in main memory, the load on the proxy, and the number of disk arms in a proxy. Our simula-

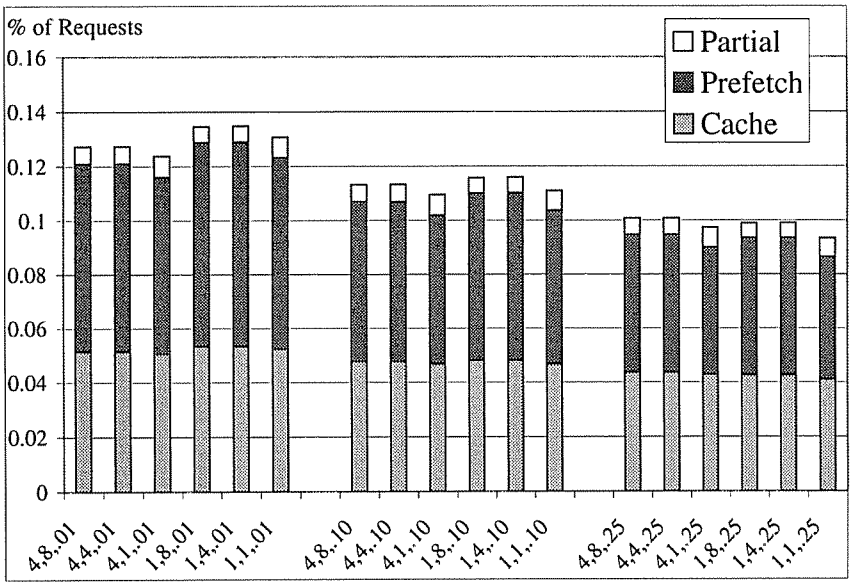

Figure 10: Fraction of requests that are serviced by the prefetch buffer (the first number specifies the prefix depth, the second specifies the search depth, the last specifies the threshold).

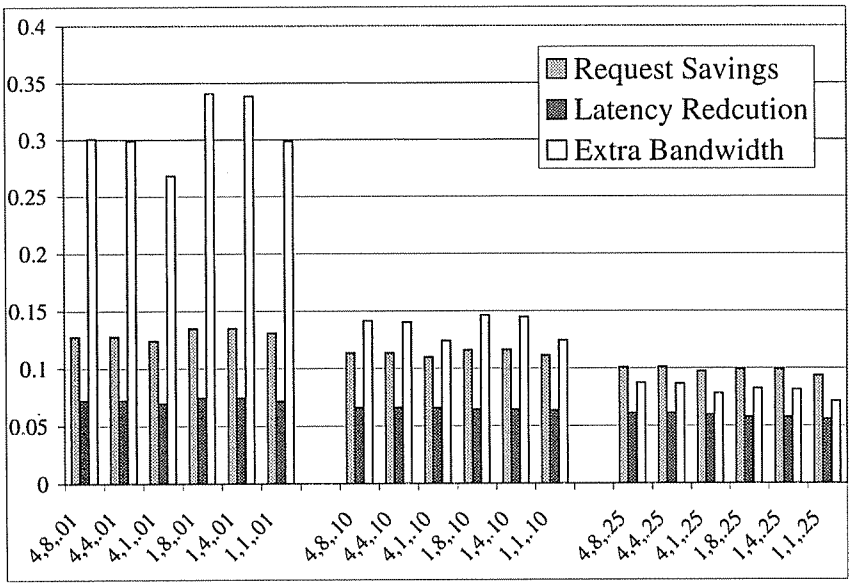

Figure 11: Requests serviced by prefetch buffer, latency savings, extra bandwidth.

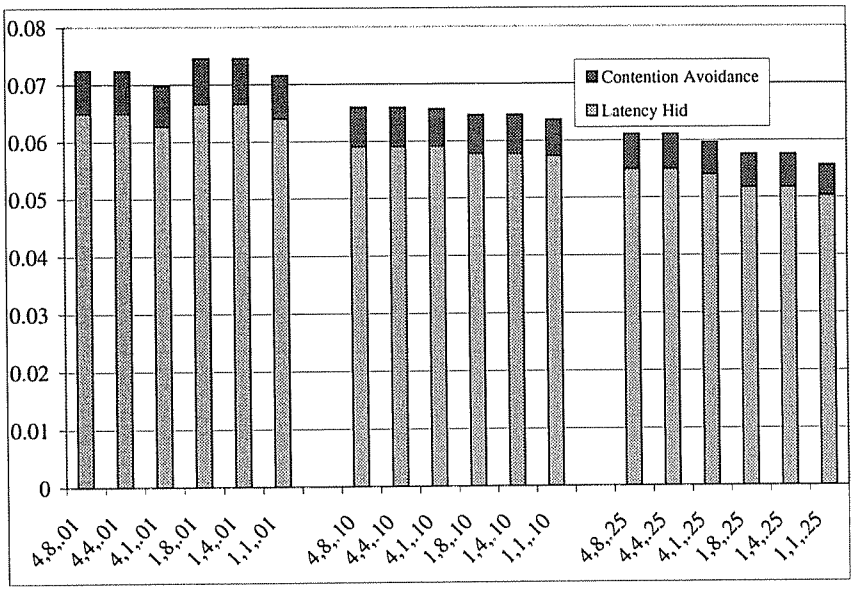

Figure 12: Breakdown of latency savings.

tor takes the simplifying assumption that all cached documents are on disk, there is only one disk arm, and the proxy always takes 10ms to fetch a document. Clearly, only implementation experiments can demonstrate the real performance of pre-pushing schemes, and we are currently working on an implementation.

Finally, since the traces we use do not mark whether a response from the Web server carries cookies or not, our simulation considers all responses cacheable at the proxies. This inflates the proxy cache hit ratio. In addition, our simulation does not consider document changes, and treat modified documents as cache hits. This also has the effect of inflating proxy cache hit ratio. On the one hand, high proxy cache hit ratio may make our latency reduction appear higher. On the other hand, there are other techniques such as delta compression that can reduce client latency for modified documents, which we did not simulate. Thus, we believe that the results reported here still serve as a good indication of achievable latency reduction from the pre-push technique.

8 Conclusion and Future Work

Today, a large percentage of the Internet population access the World Wide Web via low bandwidth modem connections. The users spend a lot of time impatiently waiting for web pages to come up on their browsers. New infrastructure to lessen the latency will not be available for some time. In the mean time anything that can reduce the latency over the existing infrastructure will be a big win. We have investigate the potential of prefetching between the low-bandwidth clients and caching proxies (called proxy-based pre-pushing) can be implemented in hiding the observed latency of the web over these low bandwidth connections.

Our results show that with perfect predictors proxy-based Web pre-pushing can reduce user observed latency by over 20%. Using a PPM compressor-based predictor, the latency can be reduced by nearly 10%. Thus, proxy-side pre-pushing is a promising technique that can have a considerable effect on user's Web surfing experience.

Much future work remains. We are planning to implement the pre-pushing scheme and measure the latency reduction in reality. We are also looking into ways to improve prediction accuracies for the proxies, including better predictors. Finally, we are investigating how the scheme might perform in high-latency high-bandwidth environments such as satellite transfers.

References

[1] T. C. Bell, J. C. Cleary, and I. H. Witten. *Text Compression.* Prentice Hall Advanced Reference Series, 1990.

[2] Azer Bestavros and Carlos Cunha. Server-initiated document dissemination for the www. *IEEE Data Engineering Bulletin*, September 1996.

[3] Pei Cao, Edward W. Felten, Anna R. Karlin, and Kai Li. A study of integrated prefetching and caching strategies. In *Proc. 1995 ACM SIGMETRICS*, pages 188–197, May 1995.

[4] Pei Cao, Edward W. Felten, Anna R. Karlin, and Kai Li. Implementation and performance of integrated application-controlled file caching, prefetching and disk scheduling. In *TOCS*, November 1996.

[5] Carlos Cunha and Carlos F. B. Jaccoud. Determining www user's next access and its application to pre-fetching. In *Proceedings of ISCC'97: The Second IEEE Symposium on Computers and Communications*, July 1997. URL: http://www.cs.bu.edu/students/alumni/carro/Home.html.

[6] Kenneth M. Curewitz, P. Krishnan, and Jeffrey Scott Vitter. Practical prefetching via data compression. In *Proceedings of SIGMOD'93*, pages 257–266, May 1993.

[7] Fred Douglis and Gideon Glass. Performance of caching proxies. *Private Communication*, 1997.

[8] Steven Gribble and Eric Brewer. Ucb home IP HTTP traces. *Available at http://www.cs.berkeley.edu/ gribble/traces/index.html*, June 1997.

[9] Jim Griffioen and Randy Appleton. Reducing file system latency using a predictive approach. In *Conference Proceedings of the USENIX Summer 1994 Technical Conference*, pages 197–208, June 1994.

[10] Wcol Group. Www collector - the prefetching proxy server for www. *http://shika.aist-nara.ac.jp/products/wcol/wcol.html*, 1997.

[11] James Gwertzman and Margo Seltzer. The case for geographical push-caching. In *Proceedings of the Fifth Workshop on Hot Topics in Operating Systems*, 1995.

[12] James Gwertzman and Margo Seltzer. An analysis of geographical push-caching. *http://www.eecs.harvard.edu/vino/web/server.cache/icdcs.ps*, 1997.

[13] Tracy Kimbrel, Pei Cao, Edward W. Felten, Anna R. Karlin, and Kai Li. Integrated parallel prefetching and caching. Technical Report TR-502-95, Princeton University, Department of Computer Science, November 1995.

[14] Thomas M. Kroeger, Darrell D. E. Long, and Jeffrey C. Mogul. Exploring the bounds of web latency reduction from caching and prefetching. In *Proceedings of USENIX Symposium on Internet Technology and Systems*, December 1997.

[15] Tong Sau Loon and Vaduvur Bharghavan. Alleviating the latency and bandwidth problems in www browsing. In *Proceedings of the 1997 USENIX Symposium on Internet Technology and Systems*, December 1997. URL: http://timely.crhc.uiuc.edu/.

[16] Evangelos P. Markatos and Catherine E. Chronaki. A top-10 approach to prefetching on the web. Technical report, Technical Report No. 173, ICS-FORTH, Heraklion, Crete, Greece, August 1996. URL http://www.ics.forth.gr/proj/arch-vlsi/www.html.

[17] Venkata N. Padmanabhan and Jeffrey C. Mogul. Using predictive prefetching to improve world wide web latency. *ACM SIGCOMM Computer Communication Review*, July 1996.

[18] Mark Palmer and Stanley B. Zdonik. Fido: A cache that learns to fetch. In *Proceedings of the 17th International Conference on Very Large Data Bases*, pages 255–264, September 1991.

[19] R. Hugo Patterson, Garth A. Gibson, Eka Ginting, Daniel Stodolsky, and Jim Zelenka. Informed prefetching and caching. In *Proceedings of 15th ACM Symposium on Operating Systems Principles*, December 1995.

[20] Carl D. Tait and Dan Duchamp. Detection and exploitation of file working sets. Technical Report CUCS-050-90, Computer Science Department, Columbia University, 1990.

[21] Jeffrey Scott Vitter and P. Krishnan. Optimal prefetching via data compression. In *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science*, pages 121–130, also appears as Brown Univ. Tech. Rep. No. CS–91–46, October 1991.