Web Prefetching Using Partial Match Prediction

Themistoklis Palpanas themis@cs.toronto.edu

Alberto Mendelzon mendel@cs.toronto.edu

Department of Computer Science

University of Toronto

10 King's College Road, Toronto

Ontario, M5S 3G4, CANADA

December 19, 1998

Abstract

Web traffic is now one of the major components of Internet traffic. One of the main directions of research in this area is to reduce the time latencies users experience when navigating through Web sites. Caching is already being used in that direction, yet, the characteristics of the Web cause caching in this medium to have poor performance. Therefore, *prefetching* is now being studied in the Web context. This study investigates the use of partial match prediction, a technique taken from the data compression literature, for prefetching in the Web. The main concern when employing prefetching is to predict as many future requests as possible, while limiting the false predictions to a minimum. The simulation results suggest that a high fraction of the predictions are accurate (e.g., predicts 18%-23% of the requests with 90%-80% accuracy), so that additional network traffic is kept low. Furthermore, the simulations show that prefetching can substantially increase cache hit rates.

1 Introduction

One of the main directions of research in the Web is to reduce the time latencies users experience when navigating through Web sites. The work in this area originated from relevant research in operating systems, where the aim is to reduce file system latencies [1]. Caching is a technique that is already being used in the Web domain. Caches reduce latencies, because they allow fast retrievals of potentially frequently accessed documents.

However, recent studies [14, 9] indicate that the benefits from this technique are rather limited. The vast number of documents available in the Web, the quick rate of their change, and the diverse needs across users and over time, are the main factors that cause Web caching to perform poorly. Thus, another method that was first applied in operating systems, *prefetching*, is now being studied in the Web context. When prefetching is employed, Web pages that the user is likely to access in the near future are transferred to her cache without being requested. If the user does request one of the prefetched pages, it will already be in the local cache, thus, reducing the latency to minimum. Prefetching is complementary to the caching mechanism. It may take place while the I/O system is idle, and if the predictions are accurate enough the performance of the cache can be significantly improved.

This work investigates the applicability of multiple high-order Markov models for doing prefetching in the Web. Specifically, an algorithm that is also employed in the compression context, *Prediction by Partial Match*, is used to predict the users' future requests to a particular Web site. A trace-driven simulation emulates the network traffic with the additional (i.e., the prefetched pages) workload, the behaviour of the clients' browsers and their caches under the new scheme, and explores the characteristics of the prediction algorithm in this new environment. This study evaluates the costs and benefits of employing this technique, when different parameters of the model vary. The experiments show that 18%-23% of the requests can be predicted with 90%-80% accuracy, while the cache hits increase by up to 6.5 times. These results demonstrate that the algorithm can be efficiently tailored to suit the new environment, and deliver considerable benefits to the users, while achieving better performance than other approaches.

The remainder of this document is organized as follows. The theoretical model, and the implications of employing it to the Web environment are discussed in Section 2. Section 3 describes the simulation framework. The results of the simulations are discussed in Section 4. Section 5 presents the related work. The last section summarizes the work, and suggests directions for future research.

2 Prediction Model

In order to do successful prefetching, a model that describes the users' request patterns is essential. Such models, that may be used to drive a prefetching system, can be found in the compression literature. Previous work [7] indicates that the idea of applying techniques used in compression algorithms for making predictions is fruitful. Compressors are building a model of the data they are operating on in order to be efficient. In general, these models generate a probability distribution for the elements in the data, which is subsequently used to achieve effective coding. According to this scheme, data with high probability to occur are encoded with few bits, and infrequently occurring data with many bits. Therefore, an accurate model significantly improves the performance of the coding.

Likewise, in prefetching, the pages that are most likely to be requested next are prefetched. Thus, if a compressor can effectively compress a data sequence that means the model it is using accurately describes the data, and can be used for making predictions.

2.1 Context Modeling

As advocated earlier, the prediction model that a compressor uses is crucial for its efficacy. Research in the compression community has shown that the algorithms employing *context modeling* (for general purpose compression) tend to achieve superior performance [2].

Context models comprise a family of techniques that use the preceding few characters in order to calculate the probability of the next one. The simplest way to predict which character will follow a string is by selecting the one with the highest fixed probability. In this case the previous characters are ignored. A more sophisticated way of computing the letter probabilities is to take into account the preceding symbols. Evidently, the predictions are more accurate when they are produced with respect to some context. The length of a context is its *order*. So, if the m preceding symbols are used to determine the probability of the next character this is an *order-m* model.

Prediction by Partial Match (PPM) [4] is one of the context models that have been proposed in the literature, and upon which our work is based. A PPM compressor uses multiple high-order Markov models to store the contexts. Then, the algorithm uses the high-order contexts to make predictions, when these contexts are available. Otherwise, the predictions are based on the lower orders.

2.2 Building the Model

In the Web context the elements of the model are access events to particular Web pages (in the same server for our case), and the contexts correspond to sequences of such events. Therefore, if A, B, and C are three pages (all of them residing in the same Web server) the context $\{ABC\}$ denotes that these three pages have been accessed in that specific order by at least one of the clients.

An *order-m* prefetcher maintains $m+1$ Markov predictors[1] which correspond to contexts of length 0 to m. One way of describing this model is by using a trie [12]. Thus, contexts of different lengths can be mingled in a single structure.

The algorithm for constructing the prediction model is depicted in Figure 1.

An example describing the above procedure will give an insight into the way the model is constructed. The example is illustrated in Figure 2. Assume that the model of order 2 is initially empty, and that the sequence of accesses $ABACDBCA$ is introduced. In the very first step the only available context is the order-0 context, which is represented by the dotted square surrounding the root node Λ (Figure 2.a). Thus, the element A becomes the child of Λ, and at the same time the new order-1 context. The number label associated with the node counts the occurrences of this context. Again, the dotted squares show the current context of each order (Figure 2.b). When element B comes, all contexts are updated in turn (Figure 2.c). The process continues in a similar fashion until the entire sequence is exhausted (Figure 2.e).

The algorithm described above makes only a single pass over the data, does not require any bulk processing, and there is no need for preprocessing either. Therefore, it can operate in real-time. The model can be updated as information on new user requests comes in, and at the same time provide predictions concerning the future accesses.

[1]An *order-k Markov predictor* is a scheme which calculates the conditional probability p of accessing page P, given that the previous accesses were to pages $\{P_1, P_2, \dots, P_k\}$ in that order. More formally, the predictor computes the probability $p(P|P_k P_{k-1} \dots P_1)$.

Objective: Build a prediction model based on the access patterns of the users.
Input: The trie structure T, representing the prediction model of order m constructed so far, and a set S of events deriving from the same user.
Output: The updated prediction model.

```
current_context[0]:=root node of T;
for length j=1 to m
   current_context[j]:=NULL;
for every event R in S
   for length j=m down to 0 {
      if current_context[j] has child-node C representing event R {
         node C occurrence_count:=occurrence_count+1;
         current_context[j+1]:=node C;
      }
      else {
         construct child-node C representing event R;
         node C occurrence_count:=1;
         current_context[j+1]:=node C;
      }
      current_context[0]:=root node of T;
   }
```

Figure 1: Algorithm for building the prediction model.

An interesting point is that in the case where predictions from different contexts are merged, subtle variations in the pattern sequences can be captured. In the model of Figure 2.e for example, assume that a specific user has already seen page D followed by B, and that the next two accesses will be for pages A and C. Both of these pages can be prefetched, since they are predicted by the contexts $\{B\}$ and $\{DB\}$ respectively. Consequently, no matter whether the actual request sequence is $\{DBAC\}$ or $\{DBCA\}$, it can potentially be predicted by the model.

3 Simulation Model

In order to estimate the performance of the algorithm a simulation model is used. The simulation is trace-driven, i.e., a real Web server log file is employed to emulate the requests of the clients, and to assess the operational behaviour of the prefetching system. An overview of the architecture of the simulator is depicted in Figure 3. The *dispatcher* is the module responsible for the coordination of the

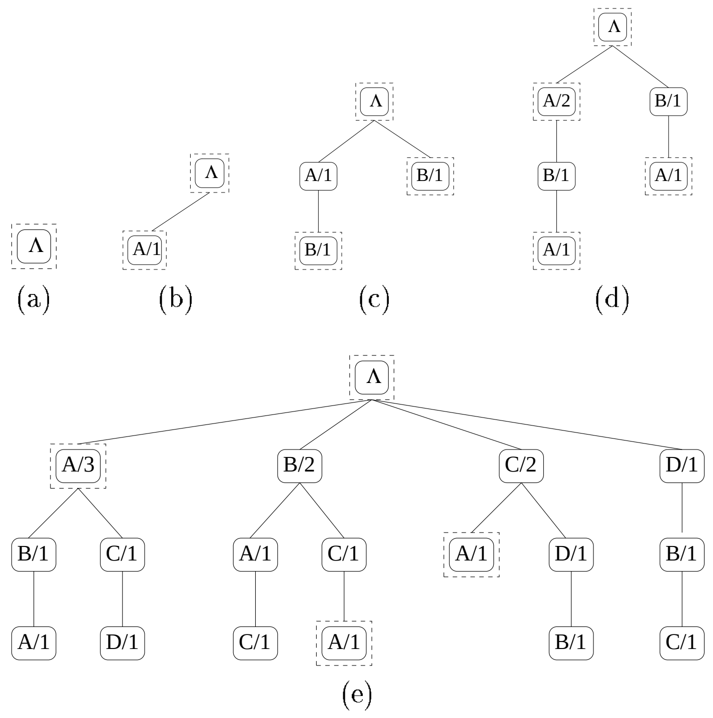


Figure 2: Structure of the model when inserting the sequence $ABACDBCA$.

other components of the system.

3.1 The Prediction Engine

This component constructs and updates the prediction model according to the requests issued by the users, and offers predictions independently to each client.

The traces pertaining to different clients should not be intermingled when building the model, since this would lead to a distorted view of the actual access patterns. In addition, the model should differentiate between discrete series of accesses even if they belong to the same user. The model tries to capture the above characteristics by introducing the notion of *browsing sessions*. A browsing session S_ϕ is a set of requests, $\{R_1, R_2, \ldots, R_i, \ldots, R_n\}$, where R_i represents an entry from the log (carrying information about the client, the request and the time she made it, and any error codes). Each request R_i in the set is constrained to belong to the same client ϕ, i.e., $R_i.host = \phi$ for $1 \leq i \leq n$. In addition, there is a total ordering in the set, $R_i.time \leq R_{i+1}.time$ for $1 \leq i < n$, and a time window is used to group

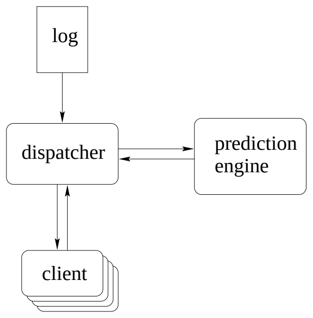


Figure 3: The architecture of the simulation model.

requests in sessions, $R_{i+1}.time - R_i.time \leq t^s_{max}$ for $1 \leq i < n$. Therefore, requests from discrete clients correspond to different browsing sessions, and a session ends when the relevant user has been idle for more than t^s_{max} time units[2]. A typical value for t^s_{max} would be in the order of few minutes. The model does not solve the problem of identifying different users behind the same proxy server. Nevertheless, prefetching can be beneficial for them as well.

It is important to note here that when building the model not all of the log file entries are taken into account. Obviously, requests for non-existent pages or requests that result in any other error do not contribute to the model. In addition, pages generated by *CGI programs* cannot be prefetched since they usually involve user selections that normally take place just before the request of the specific page. Finally, all the requests for the inlined images are also ignored. It is evident that when a user issues a request for some document, subsequent requests for all the embedded images will follow.

The prediction engine selects which pages to prefetch based on the occurrence count associated with each node in the model. In addition, the occurrence count of a certain node divided by the count of the father node yields the percentage of times that this particular path was followed among all the possible ones. This fraction also enables the prediction engine to set a *confidence* threshold, below which no pages are selected.

As discussed in the previous section, the strategy of combining predictions from contexts of multiple lengths may prove useful. The outline of such an algorithm is given in Figure 4. Each context of the

[2]The s superscript in the time window length stands for the server, where the prediction engine is located. Later on, a similar time window will be introduced for the simulation of the clients.

model produces predictions independently, and may be associated with a different level of confidence. Then, all the predictions are merged into a single set, and duplicates are removed.

Objective: Predict the next request, given the prediction model, the previous requests, and the confidence threshold for each order of the model.
Input: The trie structure T, representing the prediction model of order m; a set of the last k requests, R_i, $0 \leq i \leq k \leq m$; and the confidence threshold for each order, $c_{th}[j]$, $0 \leq j \leq m$.
Output: A set P of predicted pages.

(initialization is not actually a separate phase; current contexts are updated as new requests arrive)

```
for length j=1 to k
   current_context[j]:=node of depth j, representing the access
                       sequence {R[k-j+1],..., R[k]};
P:=NULL;
for length j=k down to 1
   for every child-node C of current_context[j]
      if (occurrence_count of C)/(occurrence_count of parent) >= c_th[j]
         P:=P+C;
remove duplicates from P;
```

Figure 4: Algorithm for making predictions, using multiple orders and different confidence thresholds.

3.2 The Client

This module simulates the client side of the system. It receives the page actually requested, as well as any additional pages sent by the prefetcher, and stores them in its cache (implemented using the LRU replacement policy).

At any time during the simulation there may exist many instances of the client module running, each one associated with a different client. These instances correspond to the *active* clients. A client C_ϕ is termed active if the last request it issued to the Web server is within a window of t^c_{max} time units of the present (simulation) time. Formally, let client C_ϕ be associated with the requests $\{R_1, \ldots, R_n\}$, issued at times $R_1.time, \ldots, R_n.time$. This client is considered active if $time_{present} - R_n.time \leq t^c_{max}$.

When a client becomes inactive its cache is emptied of all its contents. The intuition for this scheme is that pages originating from a particular server do not have infinite lifetime in the client's cache. This behaviour is simulated by emptying the cache at specific time intervals.

4 Results

All the simulations presented herein were driven by the access log files pertaining to the Web server of the Department of Computer Science of the University of Toronto [8].

Three quantities, expressed in percentages, are used to evaluate the performance of the prefetching system:

Usefulness of predictions: The number of prefetched pages that the users requested divided by the total number of requested pages.

Accuracy of predictions: The number of prefetched pages that the users requested divided by the total number of prefetched pages.

Network traffic: The volume of network traffic when prefetching is employed divided by that in the non-prefetching case.

A prefetching system aims at maximizing the first two metrics, and at the same time at minimizing the last one. It is obvious that these objectives are conflicting. The more pages are prefetched, the more probable it is for some of them to be accessed. Though, at the same time the return of value is lower (i.e., accuracy is decreasing), and the increase in network traffic is high. Thus, there is a trade-off among these quantities that the prefetching algorithm should take into account.

4.1 Trace Characteristics

The log file used in the experiments spans a time period of five days, and contains 211,300 user requests, originating from 14,007 unique hosts. As mentioned earlier, only the successful HTML and text requests are taken into account, leaving 86,000 requests. From these, the first 29,650 (i.e., the first two days) are used as training data in order to build the prediction model, and the rest to simulate a real system with prefetching.

In order to appraise the navigation patterns of the users as perceived by the prediction model, it is useful to know how many pages they request from the Web server before they leave. This information will also help in the fine-tuning of the prefetching algorithm. Figure 5 depicts the average number of

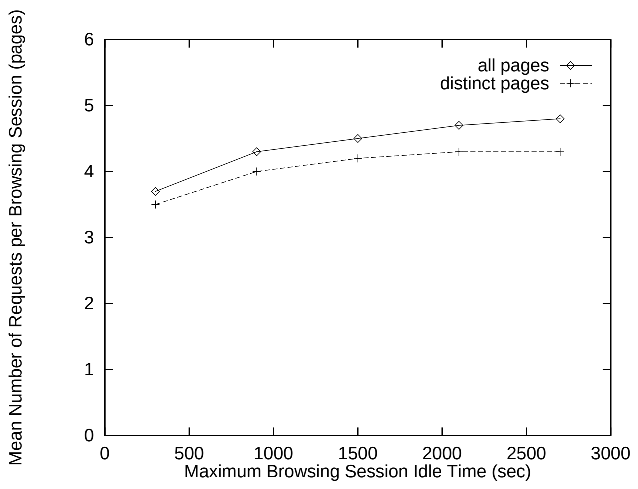


Figure 5: Mean number of page requests per browsing session, as a function of the maximum browsing session idle time.

pages requested in a single session, as the maximum browsing session idle time varies. The two curves represent the results for the cases where only the distinct pages in each session are taken into account, or all of them. The standard deviation for these experiments is reported in Table 1.

all pages		*distinct pages*	
mean	std. dev.	mean	std. dev.
3.7	5.9	3.5	5.6
4.3	7.2	4.0	6.3
4.5	7.9	4.2	6.8
4.7	8.2	4.3	7.0
4.8	8.7	4.3	7.1

Table 1: Arithmetic mean and standard deviation for the number of page requests per browsing session. Results reported for maximum browsing session idle time set to 5, 15, 25, 35, and 45 minutes.

The results indicate that most of the users tend to request only a few pages (≤ 3) during a single session, although there exist sessions with significantly higher numbers of requested pages. This observation implies that many users either are not really interested in navigating in the site, or they know exactly where to find the information they want. Therefore, prefetching will not be beneficial for them, and should be avoided.

Note that these results should not be interpreted as an exact description of the users' access behaviour. Nevertheless, they still represent the way the prediction model captures the navigation patterns of the users.

4.2 Experimental Results

The subsequent paragraphs illustrate and discuss the outcome of the simulations[3]. Each experiment investigates the influence of a single parameter at a time on the performance of the system. At the end, the best values obtained from each experiment are combined.

4.2.1 Varying the Number of Previous Requests

As the previous requests parameter increases, the number of prefetched pages decreases (Figure 6). The

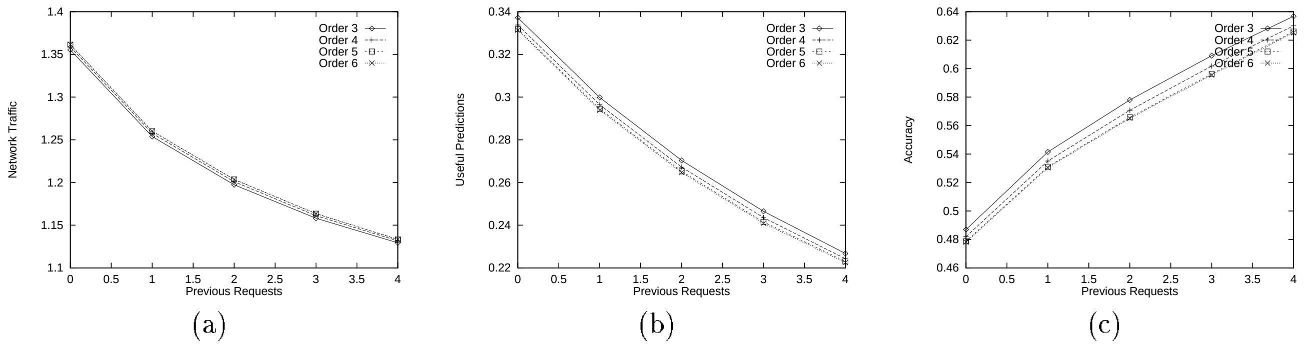


Figure 6: Varying the number of previous requests.

prefetcher is only making predictions for clients that exhibit an interest in the Web site and do not leave after the first requests. This results in more accurate predictions, yet, the predicted requests are fewer.

[3]For a more elaborate and complete presentation of the experimental results the interested reader should refer to [17].

4.2.2 Varying the Confidence

Similar patterns are observed when confidence increases (Figure 7). Larger values for this parameter

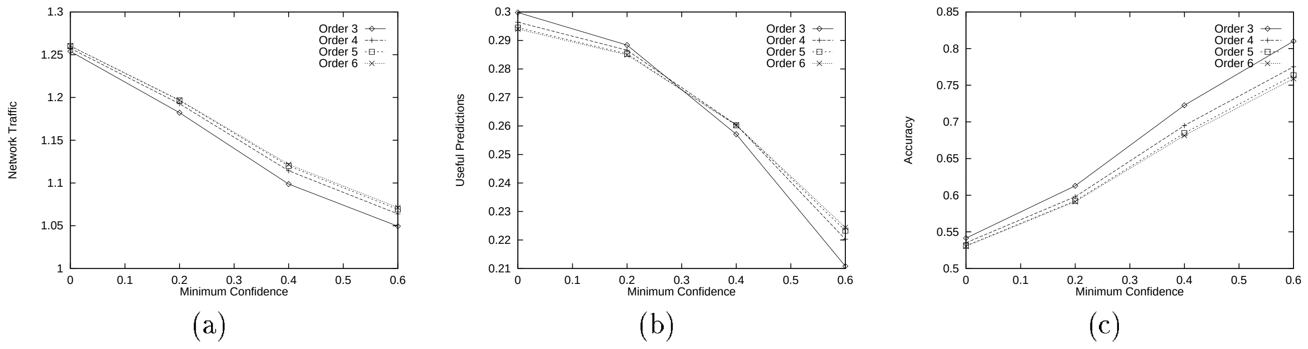


Figure 7: Varying the confidence.

cause fewer predictions to be made, thus, reducing the network traffic. However, the increment in prediction accuracy is significant since it is only the highly probable pages that are prefetched.

4.2.3 Varying the Number of Predictions

A way to predict as many requests as possible is by allowing the maximum predictions parameter to take large values, as represented in Figure 8. The number of useful predictions increase (note that

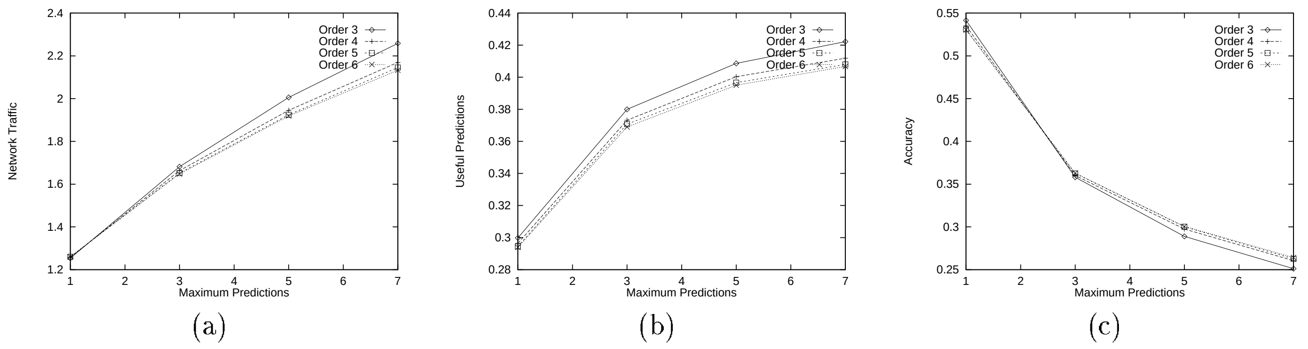
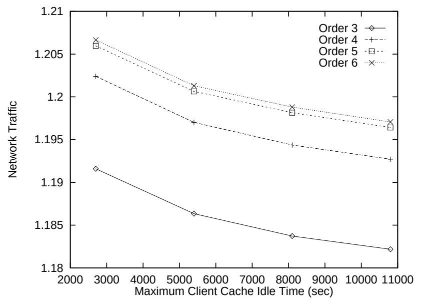


Figure 8: Varying the number of predictions.

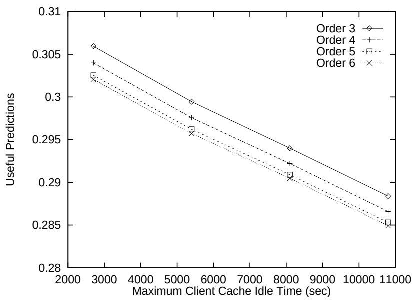
prefetching every possible page would result in a perfect figure for this metric), but the return of value is minimal. The increase in traffic is dramatic, and the accuracy degrades quickly.

4.2.4 Varying the Client Cache Idle Time

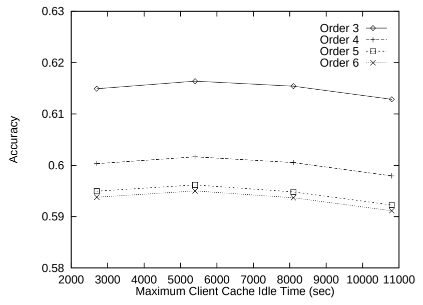
The results displayed in Figure 9 correspond to a confidence setting of 0.2.



(a)



(b)



(c)

Figure 9: Varying the client cache idle time.

As the maximum client cache idle time increases, network traffic and the number of useful predictions diminish. At the same time accuracy remains nearly constant, which suggests that the aforementioned reduction is due to an additional number of requests being serviced by the client cache, and not to a service degradation. When client caches in the simulation are left to live longer, it becomes more probable for a page already in the cache to be requested again.

4.2.5 Combining Predictions

The representation of the prediction model, which stores multiple context orders in the same structure, offers the opportunity to combine the predictions from different orders.

Figure 10 compares the use of a constant confidence value with *weighted* confidence, for a model of order 6. Note that in both cases predictions are made independently from each order of the model, and are then combined into a single answer set. When weighted confidence is employed, higher order contexts are trusted more, and are assigned a smaller confidence.

In the graphs of Figure 10 the horizontal lines represent the performance of the algorithm with weighted confidence which for the orders 2 to 6 takes the values 0.8 to 0.5 (reducing in steps of 0.08). This algorithm is compared with four others which use constant confidence with values from 0.5 up to 0.8. The results suggest that the weighted confidence approach tries to achieve good performance in all

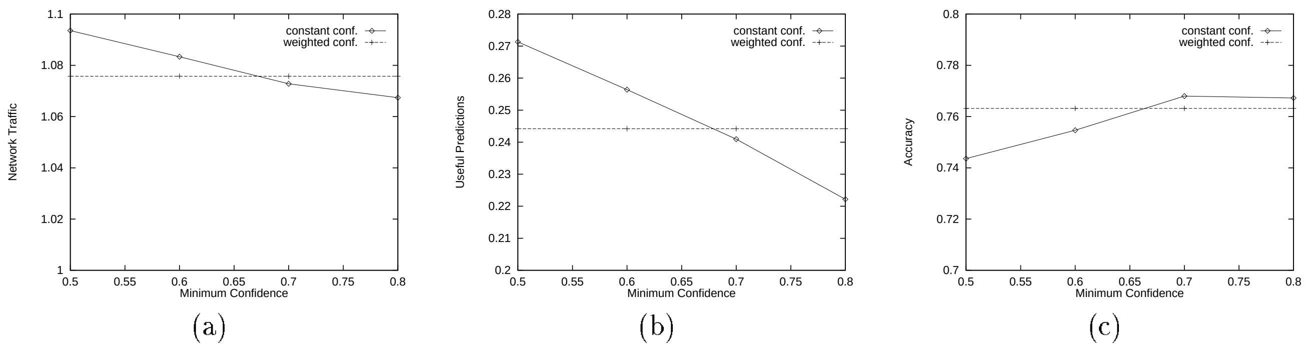


Figure 10: Constant and weighted confidence when combining predictions from different orders.

three metrics. Indeed, it performs above the average of the four constant-confidence algorithms in all categories.

In Figure 11 weighted-confidence algorithms using different maximum order models are compared. The prediction models have orders ranging from 3 up to 6, while the weighted confidence is set to 0.8

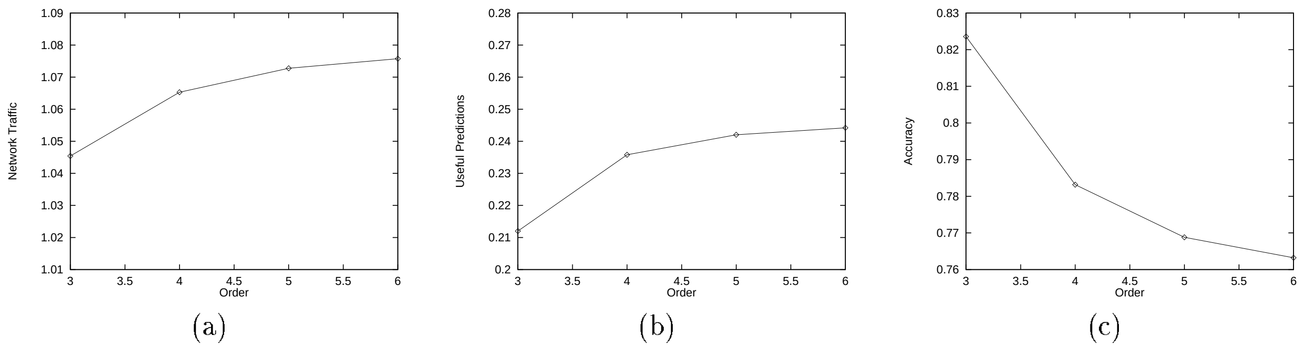


Figure 11: Varying the order of the model when combining predictions from different orders.

for the context of order 2 and reduces by 0.08 for each subsequent order. The results show that the number of useful predictions increase along with the maximum order of the model, at the cost of the accuracy.

4.3 Efficiency of the Algorithm

It is evident that the performance of the algorithm on any of the specified metrics will vary depending on the parameters. Different settings may limit the algorithm to predict only the highly probable pages, or relax the constraints and result in more page requests being predicted. Two such cases are presented in Table 2. The first experiment predicts almost a quarter of the total number of requests issued to the Web server, with an accuracy of 80%. The second one improves the accuracy to 90%, which causes the number of predicted requests to shrink to less than one fifth.

max order	*confidence*	*prev. requests*	*network traffic*	*useful pred.*	*accuracy*
6	0.6-0.8	1	1.05	0.23	0.8
3	0.8-0.9	4	1.02	0.18	0.9

Table 2: Efficiency of the algorithm (using weighted confidence). The parameters common to both experiments are maximum predictions 5 pages, client cache size 30 pages, browsing session idle time 600 seconds, and client cache idle time 5400 seconds.

4.3.1 Potential Predictions

In order to assess the ability of the algorithm to make predictions, it is useful to know how many predictions can be done and how many of those were actually made. The potential predictions are restricted by two factors. First, the initial request of each browsing session cannot be possibly predicted. Second, the model cannot predict any sequence of requests that has not been observed before.

The histogram of Figure 12 shows the percentage of requests that can be predicted for the log file

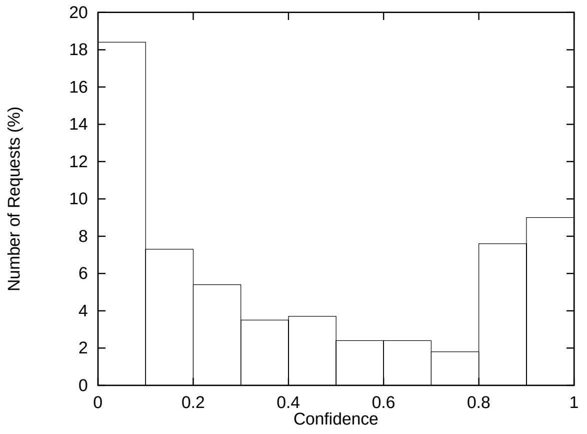


Figure 12: Percentage of requests that can potentially be predicted.

used in this study. These are the requests of the form $\{AB\}$, where page B is accessed after page A, and this pattern has occurred before. The total number of potential predictions (i.e., the sum of all the bars in the histogram) accounts for only 60% of the requests.

The experiments presented in Table 2 indicate that the predictions made are actually more than anticipated by the figures cited in the histogram. Indeed, taking into consideration the confidence levels used in the experiments, the predictions are about 15% more than expected. Note that even this estimate is pessimistic since the predictions in the simulations do not start right after the first access to the server. This outcome is explained by the fact that in the experiments predictions are also derived from high-order models, which usually exhibit greater confidence levels.

4.3.2 Cache Benefits

The simulations also show that prefetching can be used in a complementary fashion to the caching technique. Figure 13 reports the cache hits for various cache sizes, with and without the use of prefetching. In the former case, the parameters of the algorithm were the same as in the first experiment of Table 2. When prefetching is employed, the cache experiences up to 6.5 times more cache hits than in the non-prefetching case. As Figure 13 shows, the relative benefits increase as the cache size gets smaller.

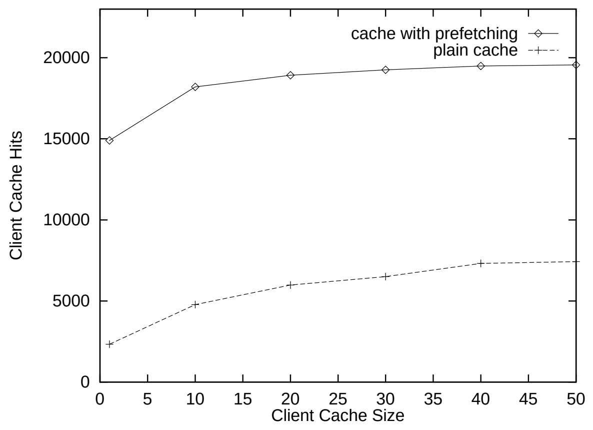


Figure 13: Cache hits with and without prefetching, for different cache sizes.

5 Related Work

In what follows we present an overview of the cache and prefetching techniques. Work in both the operating systems and Web communities is cited, so as to observe the similarities and differences between the two.

5.1 Operating Systems Context

Griffioen and Appleton [10] propose a predictive cache which prefetches file pages based on a *probability graph* of past accesses. The nodes in the graph represent accessed files, while a directed weighted arc from node A to node B expresses a metric for the probability of requesting file B within w requests after file A. A trace-driven simulation revealed improvements in the time an application has to wait for read operations to complete ranging from 22% to 51% compared with the non-prefetching case.

Curewitz et al. [7] claim that data compression techniques can be successfully used for prefetching. An implementation of this idea, based on *Prediction by Partial Match (PPM)*, is presented by Kroeger and Long [13].

Lei and Duchamp [15] employ *access trees* to make predictions. Access trees record all files accessed during one execution of each program. When an application is re-executed, the current activity is compared against the saved access trees. If a similarity is detected, the files in the access tree are prefetched. The trace-driven simulation shows a substantial reduction in applications latency, despite the significant CPU overhead.

5.2 World Wide Web Context

Bestavros proposes a server-initiated protocol [3], which sends to the client —along with the requested document— the server's predictions. The predictions are produced by the closure of the matrix whose *(i,j)* element represents the probability that document j will be requested within a certain time window after the request of document i.

Previous work on operating systems [10] has been the basis of Padmanabhan and Mogul's research

[16] on prediction of future requests. A trace-driven simulation along with a linear model for the network is used to test the algorithm. The results indicate that the benefits from prefetching are significant. However, the increase in network traffic is considerable (an order of magnitude greater than the numbers reported herein). The accuracy of the predictions is not reported.

Independently from our work, Jacobson and Cao [11] applied a version of the partial match prediction technique for prefetching between proxies and low-bandwidth clients. This work shows that prefetching can reduce latency by less than 10% (predicting 12% of the requests, and increasing traffic by 18%), though, a significant part of this reduction is attributed to the caching effect of the prefetch buffer. The experiments indicate that our algorithm has superior performance (in terms of the metrics we discussed). Nevertheless, in our case the prefetching process is coupled with the server, as opposed to the proxy.

A recent study attempts to determine bounds in the performance of proxy caching and prefetching [14]. The authors performed a trace-driven simulation with infinite-size cache and complete knowledge of future requests. The external latency between the proxy cache and the server accounts for 77% of the total latency. Caching achieved a reduction of 26%, prefetching 57%, and the combination of both 60%.

Crovella and Barford [6] discuss the effects of prefetching on the network. A trace driven simulation indicates that straightforward approaches to prefetching increase the burstiness of traffic. Instead, the authors propose a transport rate control mechanism. The simulation denotes that rate-controlled prefetching significantly improves network performance compared not only with the straightforward approach, but also with the non-prefetching case, while delivering the requested documents on time.

Several commercial products promise to enhance the performance of browsers [5, 18, 19], as well. However, the algorithms used are not sophisticated. They merely exploit the idle time of the client's network connection to prefetch all the links of the current page, or the user's popular pages. Thus the increase in network traffic is considerable.

6 Conclusions and Future Work

This work investigated the application of a multi-order context modeling technique for prefetching in the Web domain. The special characteristics of the Web context were recognized, and the algorithm was tailored to fit this new environment. Despite the heterogeneous nature of the users (their geographic origin, the reason they are visiting the Web site, the information they are looking for) which imposes limitations on the performance of the prefetching system, the simulation still suggests that the advantages of employing prediction are significant, and the performance is comparable or better to similar approaches. A considerable fraction of the users' requests can be predicted with high accuracy (e.g., predicts 18%-23% of the requests with 90%-80% accuracy), thus, reducing retrieval latencies and boosting the cache performance.

We are currently looking into the following ways of extending the work presented herein:

- Perform time simulations, in order to evaluate the time savings for the users.
- Introduce flexibility in the algorithm; cooperate with the network layer in order to make more informed decisions.
- Incorporate an aging mechanism, which will allow the model to automatically respond to changes in the user access patterns; a similar technique can be used to restrict the model's memory requirements.

References

[1] Mary Baker, Satoshi Asami, Etienne Deprit, John Ousterhout, and Margo Seltzer. Non-Volatile Memory for Fast, Reliable File Systems. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 10–22, September 1992.

[2] T. C. Bell, J. C. Cleary, and I. H. Witten. *Text Compression.* Prentice Hall, 1990.

[3] Azer Bestavros. Speculative Data Dissemination and Service to Reduce Server Load, Network Traffic and Service Time in Distributed Information Systems. In *International Conference on Data Engineering*, pages 180–189, New Orleans, LO, February 1996.

[4] John G. Cleary and Ian H. Witten. Data Compression Using Adaptive Coding and Partial String Matching. *IEEE Transactions on Communications*, 32(4):396–402, 1984.

[5] Connectix Corp. http://www.connectix.com, March 1998.

[6] Mark Crovella and Paul Barford. The Network Effects of Prefetching. In *IEEE Infocom*, San Francisco, CA, USA, 1998.

[7] Kenneth M. Curewitz, P. Krishnan, and Jeffrey Scott Vitter. Practical Prefetching via Data Compression. In *ACM SIGMOD International Conference*, pages 257–266, Washington, DC, USA, June 1993.

[8] Department of Computer Science, University of Toronto. http://www.cs.toronto.edu, March 1998.

[9] Fred Douglis, Anja Feldmann, Balachander Krishnamurthy, and Jeffrey C. Mogul. Rate of Change and other Metrics: a Live Study of the World Wide Web. In *USENIX Symposium on Internet Technology and Systems*, pages 147–158, Berkeley, CA, USA, December 1997.

[10] James Griffioen and Randy Appleton. The Design, Implementation, and Evaluation of a Predictive Caching File System. Technical Report CS-264-96, Department of Computer Science, University of Kentucky, June 1996.

[11] Quinn Jacobson and Pei Cao. Potential and Limits of Web Prefetching Between Low-Bandwidth Clients and Proxies. In *Web Caching Workshop*, June 1998.

[12] Donald E. Knuth. *The Art of Computer Programming: Sorting and Searching*, volume 3. Addison-Wesley, 1973.

[13] Thomas M. Kroeger and Darell D. E. Long. Predicting File System Actions from Prior Events. In *Winter USENIX Conference*, 1996.

[14] Thomas M. Kroeger, Darrell D. E. Long, and Jeffrey C. Mogul. Exploring the Bounds of Web Latency Reduction from Caching and Prefetching. In *USENIX Symposium on Internet Technologies and Systems*, pages 319–328, San Diego, CA, USA, January 1997.

[15] Hui Lei and Dan Duchamp. An Analytical Approach to File Prefetching. In *USENIX Annual Technical Conference*, pages 275–288, Berkeley, CA, USA, January 1997.

[16] Venkata N. Padmanabhan and Jeffrey C. Mogul. Using Predictive Prefetching to Improve World Wide Web Latency. *ACM SIGCOMM Computer Communication Review*, 27(3):22–36, 1996.

[17] Themistoklis Palpanas. Web Prefetching Using Partial Match Prediction. Technical Report CSRG-376, Department of Computer Science, University of Toronto, March 1998.

[18] PeakSoft Corp. http://www.peak.com, March 1998.

[19] Web3000 Inc. http://www.web3000.com, March 1998.

Contents