Neural Nets Based Predictive Pre-fetching to Tolerate WWW Latency[1]

Tamer I. Ibrahim and Cheng-Zhong Xu
Department of Electrical and Computer Engineering
Wayne State University, Detroit, MI 48202
czxu@ece.eng.wayne.edu

Abstract

With the explosive growth of WWW applications on the Internet, users are experiencing access delays more often than ever. Recent studies showed that pre-fetching could alleviate the WWW latency to a larger extent than caching. Existing pre-fetching methods are mostly based on URL graphs. They use the graphical nature of hypertext links to determine the possible paths through a hypertext system. While they have been demonstrated effective in pre-fetching of documents that are often accessed, they are incapable of pre-retrieving documents whose URLs had never been accessed.

In this paper, we propose a context-specific pre-fetching technique to overcome the limitation. It relies on keywords in anchor texts of URLs to characterize user access patterns and on neural networks over the keyword set to predict future requests. It features a self-learning capability and good adaptivity to the change of user surfing interest. The technique was implemented in a SmartNewsReader system and cross-examined in a daily browsing of MSNBC and CNN news sites. The experimental results showed an achievement of approximately 60% hit ratio due to pre-fetching. Of the pre-fetched documents, less than 30% was undesired.

Keywords: Neural networks, pre-fetching, SmartNewsReader, Web latency tolerance.

Technical Area: Web-Based Applications.

[1] Corresponding address: Dr. Cheng-Zhong Xu, ECE Department, Wayne State University, Detroit, MI 48202. czxu@ece.eng.wayne.edu

1 Introduction

Bandwidth demands for the Internet are growing at an exponential rate and quickly saturating the Internet capacity due to the pervasive use of WWW applications. Although upgrade of the Internet with higher bandwidth networks has never stopped, users are experiencing Web access delays more often than ever. Caching is widely used to alleviate the latency. However, recent studies showed that benefits from client-side caching are very limited due to the lack of enough temporal locality in Web references [13]. It was also shown that the potential for caching requested files was even declining over the past years [1][2]. An alternative is to *pre-fetch* Web documents (more generally, Web objects) that are likely to be browsed in the near future while users are processing previously retrieved documents [14]. A recent tracing experiment revealed that pre-fetching could offer more than twice the improvement of caching [13].

Previous studies on Web pre-fetching were mostly based on the history of user access patterns [6][16][19][20]. If the history information shows an access pattern of URL address *A* followed *B* with a high probability, then *B* will be pre-fetched once *A* is accessed. A limitation of this technique is that the pre-fetched URLs must be frequently accessed before. There is no way for the approach to pre-retrieve documents that are new or never accessed. For dynamic HTML sites where a URL has time-variant contents, the existing approaches could not retrieve desired documents by any means. In this paper, we propose a novel predictive pre-fetching technique to overcome the limitations. It relies on keywords in link anchor texts of URLs to capture user access patterns and on neural networks to predict user future requests.

Pre-fetching to hide communication latency has long been used in the design of parallel and distributed systems [5]. A closely related pre-fetching topic to Web surfing is file pre-fetching and caching in the design of remote file systems [10][15]. As a matter of fact, some of the early results on Web pre-fetching were due to the work of file pre-fetching. Padmanabhan and Mogul adapted a Giffioen-Appleton file predictive algorithm to reduce Web

latency [19]. Due to the nature of user non-deterministic browsing behaviors, Web pre-fetching is more challenge.

Web pre-fetching is an active topic in both academy and industries. Algorithms used in today's commercial products are either blind or user-guided. A blind pre-fetching technique does not keep track of any user Web access history. It pre-fetches documents based solely on the static URL relationships in a hypertext document. WebMirror[17] retrieves all anchored URLs of a document for a fast mirroring of the whole web site. NetAccelerator[12] pre-fetches the URLs on a link's downward path for a quick copy of HTML formatted documents.

User-guided pre-fetching techniques are based on user-provided preference URL links. Go-Ahead-Got-It![9] relies on a list of predefined favorite links, while Connectix-Surf-Express[4] maintains a record of user frequently accessed Web sites to guide pre-fetching of related URLs.

Researches in academy are targeted at intelligent predictors based on user access patterns. A popular approach is to represent the access pattern as a *URL graph*. Each edge between a pair of vertices represents a follow-up relationship between the two corresponding URLs. The edge weight denotes a probability of the relationship in statistics. Padmanabhan and Mogul [19] and Schechter, et al [20] presented server-side pre-fetching techniques to reduce the access latency of an average client. They established URL graphs based on access logs on a server. Cunha and Jaccoud [6] extended pre-fetching to client proxies with an attempt to provide users a smart browser or an intelligent local proxy server. Client-side pre-fetching takes advantage of user-specific behaviors. Most recently, Loon and Bharghavan presented a distributed mechanism of the profiling, pre-fetching, and related caching between client and server proxies [16].

The URL graph based approaches are demonstrated effective in pre-fetching of documents that are often accessed in history. However, they are unable to pre-retrieve those documents whose URLs are never touched before. For example, all anchored URLs of a page are fresh when a client gets into a new web site and none of them will be pre-fetched by the approaches. This paper presents a context-specific pre-fetching technique to over the limitation. It was motivated by an observation that client surfing was often guided by some keywords in *anchor texts* of URLs. Anchor test refers to the text that surrounds hyperlink

definitions (hrefs) in Web pages. For example, a user with a strong interest in Year 2000 problem won't miss any relevant news or Web pages that are pointed to by a URL with a keyword "Y2K" in its anchor text. An on-line shopper with a favorite auto model, say "BMW X5", would like to see all consumer reports or reviews about the model. The pre-fetching approach monitors Web surfing behaviors and dynamically establishes keyword-oriented access preferences. It employs neural networks to predict future requests based on the preferences. Since it is keyword-oriented, rather than URL graph based, the approach is capable of pre-fetching documents whose URLs are never accessed. For example, the approach can help a client with interests in Y2K problems pre-fetch Y2K related documents when he/she gets into a new Web site.

The rest of the paper is organized as follows. Section 2 presents neural network basics, with an emphasis on key parameters that affect the prediction accuracy and self-learning rules. Section 3 presents designs of a news reading predictor and its architecture. Sections 4 and 5 present the details of our experimental methodology and results. Section 6 concludes the paper with remarks on future work.

2 Neural Network Basics

Neural nets have been around since the late 1950's and come into practical use for general applications since mid-1980's. Due to their resilience against distortions in the input data and their capability of learning, neural networks are often good at solving problems that do not have algorithmic solutions [11].

2.1 Neurons

A neural network comprises of a collection of neurons. Each neuron performs a very simple function: computing a weighted sum of a number of inputs and then comparing the result to a predefined threshold value. The result is often referred to as *net*. If the net is less than the threshold, then the neuron yields an output of zero, otherwise one. Specifically, given a set of inputs $x_1, x_2, x_3, \ldots, x_n$ associated with weights $w_1, w_2, w_3, \ldots, w_n$, respectively, and a threshold T, the binary output *y* of a neuron can be modeled as

$$y = \begin{cases} 1 & if \sum_{i=1}^{n} w_i x_i \geq T \\ 0 & otherwise \end{cases} \qquad \textit{Equation 2.1}$$

The output y is often referred to as *net* of the neuron. Figure 1 shows a graphical representation of the neuron. Although it is a over-simplified model of the functions performed by a real biological neuron, it has been shown a useful approximation. Neurons can be interconnected together to form powerful networks that are capable of solving problems of various levels of complexity.

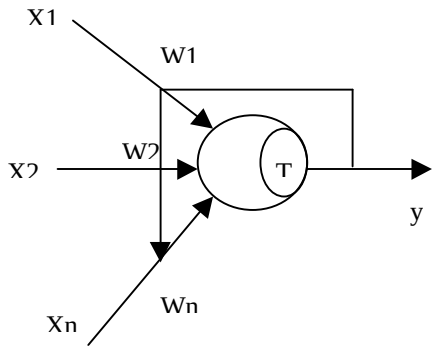


Figure 1. A graphical representation of neuron

To apply the neural network to Web pre-fetching, the predictor examines the URL anchor texts of a document and identifies a set of keywords for each URL. Using the keywords as an input, the predictor calculates the net of each URL neuron. Their nets determines the priority of URLs that are to be pre-fetched.

2.2 Learning Rules

A distinguishing feature of neurons (or neuron networks) is self-learning. It is accomplished through learning rules. A learning rule incrementally adjusts the weights of the network to improve a predefined performance measure over time. Generally, there are three types of learning rules; supervised, reinforced and unsupervised. In this study, we are interested mainly in supervised learning where each input pattern is associated with a specific desired target pattern. Usually, the weights are synthesized gradually and updated at each step of the learning process so that the error between the output and the corresponding desired target is reduced [11].

Many supervised learning rules are available for single unit training. A most widely used approach is μ–LMS learning rule. Let

$$J(w) = \frac{1}{2}\sum_{i=1}^{m}(d^i - y^i)^2 \qquad \textit{Equation} \quad 2.2$$

be the sum of squared error criterion function. Using the steepest gradient descent to minimize J(w) gives

$$w^{k+1} = w^k + m\sum_{i=1}^{m}(d^i - y^i)\,x^i \qquad \textit{Equation} \quad 2.3$$

where μ is often referred to as the learning rate of the net because it determines how fast the neuron can react to the new inputs. Its value normally ranges between 0 and 1. The learning rule governed by the equation is called the batch LMS rule. The μ-LMS is its incremental version, governed by

$$\begin{cases} w^1 = 0 \quad or \quad arbitrary \\ w^{k+1} = w^k + m(d^i - y^i)\,x^k \end{cases} \qquad \textit{Equation 2.4}$$

The LMS learning rule is employed to update the weight of keywords of URL neurons.

3 SmartNewsReader: A Web Pre-fetching Predictor

This section presents details of the keyword-oriented neuron network based pre-fetching technique, with an illustration of its application in news reading.

3.1 Keywords

A keyword is the most meaningful word within the anchor text of a URL. It is usually a noun referring to some role of an affair or some object of an event. For example, associated with the news "Microsoft is ongoing talks with DOJ" as of March 24, 1999 on MSNBC are two keywords "Microsoft" and "DOJ"; the news "Intel speeds up Celeron strategy" contains keywords "Intel" and "Celeron". Note that it is the predictor that identifies the keywords from anchor texts of URLs while users retrieving documents. Since a keyword tends to start with a capital letter, this simplifies the task of keyword identification.

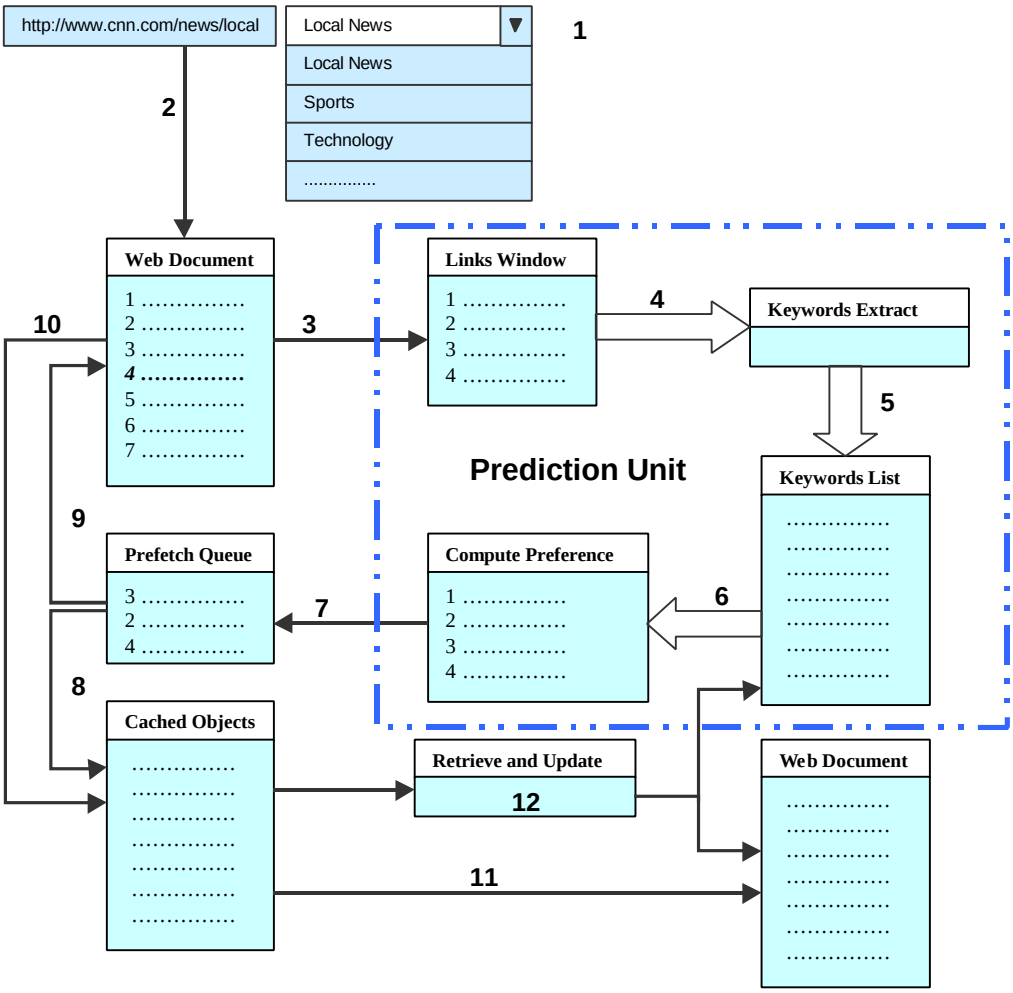
It is known that meaning of a keyword is often determined by its context. An interesting keyword in one context may be of no interest at all to the same user if it occurs in other contexts. For example, a Brazil soccer fan may be interested in all the sports news related with the team. In this sports context, the word "Brazil" will guide the fan to all related news. However, the same word may be of less importance to the same user if it occurs in general or world news categories, unless the user has a special tie with Brazil. The fact that a keyword in different contexts has different implications is well recognized in structural organizations of news sites. Since a news site usually, if not always, organizes the URLs into categories, we assume one pre-fetching predictor for each category for an accurate prediction.

It is also known that the significance of a keyword to a user changes with time. For example, news related to "Brazil" hit the headline during the Brazil economic crisis period in 1998. The keyword should be sensitive to any readers of market or financial news during the period. After the crisis was over, its significance decreased unless the reader has special interests in the Brazil market. Due to the self-learning property of a neural network, the weight of the key can be adapted to the change.

3.2 Architecture

Figure 2 shows the architecture of the SmartNewsReader. Its kernel is a prediction unit. It monitors the way of user browsing. For each user required document, the predictor examines

the keywords within its anchor text, calculate their preferences (nets), and pre-fetches those URLs which contain keywords with higher preferences than the neuron threshold.



1. User requests a URL associated with a category.
2. The URL document is retrieved and displayed to the user.
3. The contents of the document are examined to identify the first set of links to evaluate.
4. Each link from the set is processed to extract the keywords.
5. Keywords are matched to the Keyword list. They are added to the list if they occur for the first time.
6. The corresponding preferences are computed using the weights associated with the keywords.
7. Links with preferences higher than the neuron threshold are sorted and placed in the prefetch queue.
8. Links are prefetched one by one and the objects are placed in the cache.
9. If the pre-fetching queue is empty, process the next set of links [Step 2].
10. When the user requests a link, the cache list is examined to see if it's there.
11. If it was there, then it is displayed.
12. If not, it is retrieved from the Web and the corresponding keyword's weights are updated.

Figure 2. Architecture of the SmartNewsReader

In light of the fact that a document may contain a large number of links, the predictor won't examine all anchor texts of URLs at a time. Instead, it examines and pre-fetches documents through a window. That is, a fixed number of URLs are considered at a time. It patterns after a user browsing behavior because users will look at a few links before deciding which to follow. Since evaluation of links takes non-negligible time, group evaluation also provides a quick response to user web surfing. The window size, referred to as *breadth of the pre-fetching*, was set to five in our experiment.

Once an anchored URL is clicked, the predictor starts to look for the requested document in the cache. If it has already been pre-fetched, the document is displayed. (The weights of its related keywords will remain unchanged by the rule of Equation 2.4.) Otherwise, the predictor suspends any ongoing pre-fetching process and starts to examine the caption of the requested link for keywords. If a keyword is introduced for the first time, a new weight will be assigned to it. Otherwise its existing weight will be *positively* updated ($d = 1$) by the rule of Equation 2.4.

3.3 Control of Pre-fetch Cache and Keyword List

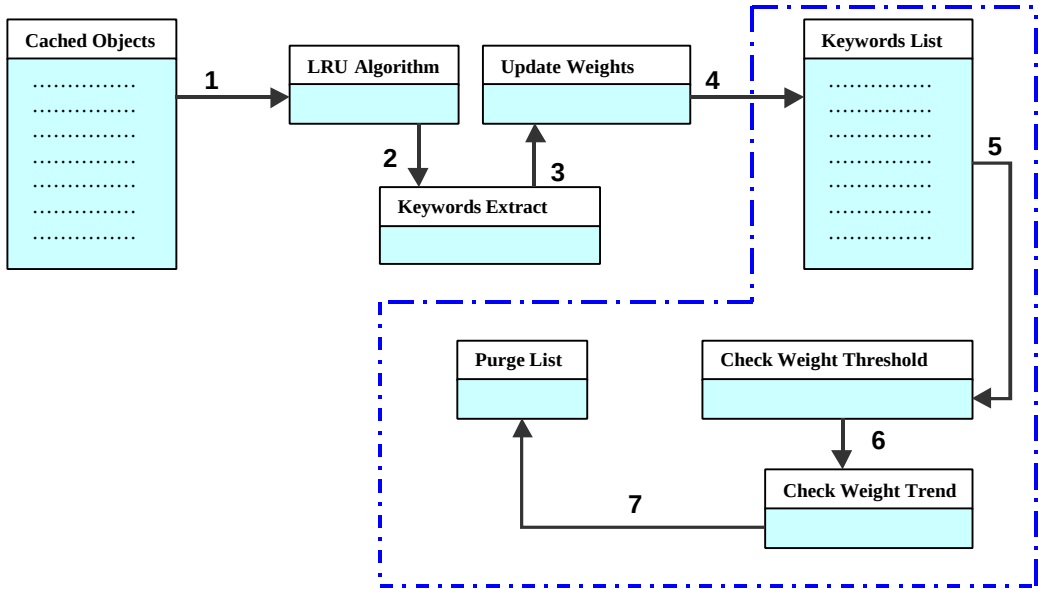
Notice that the keyword list could be full with unwanted items when the user loses interest in a long-standing topic. The pre-fetch cache may also contain undesired documents due to the presence of some trivial (non-essential) keywords in document captions. For example, the keyword "DOJ" is trivial in a sector of technology news. It is of interest only when it appears together with some other technology-related keywords like Microsoft or IBM. Figure 3 shows a mechanism of the SmartNewsReader to control the overflow of the keyword list and purge the undesired documents in the pre-fetch cache.

By checking the cache, we can identify documents that have been pre-fetched but never been accessed. The system will then decrement the weight of its related keywords. Also, we need to set a *purging threshold* value that determines whether a keyword should be eliminated from the keyword list or not. If the weight of a keyword is less than this threshold, then we examine the change trend of the weight. If it is negative, then the keyword is considered trivial and is eliminated from the list.

The keywords are purged once the keyword list is full. The list size should be set appropriately. A too small list will force the unit to purge legitimate keywords, while a too large

list would be filled with trivial keywords, wasting space and time in evaluation. We set the list size to 100 and the purging threshold value to 0.02 in our experiments. It was set based on our preliminary testing. The settings are by no means optimal choices.

The keywords can also be purged when the cache is full (or up to a certain percentage) and the program needs to place a new document in it and the only way to do that is to replace an existing document or a number of documents. Documents to be replaced are those which have been in the cache for a long time without being accessed. The caption that resulted in pre-fetching those documents is first examined and the keywords are extracted. Then the weights associated with those weights are *negatively* updated ($d = 0$) by the rule of Equation 2.4. This will assure that the probability of those keywords to trigger the pre-fetching unit in the future is lower.



1. When the cache list is full, use an LRU algorithm to determine the least recently accessed objects.
2. Extract the corresponding keywords.
3. Negatively update those weights.
4. Compare the updated weights to the minimum weight threshold.
5. If they are, check the updating trend (Decreasing in this case).
6. Purge those keywords from the list if they are below the minimum weight threshold.
7. If the keyword list is full, consider steps 5, 6 & 7. However, the trend will not be necessarily decreasing in this case.

Figure 3. Overflow Control for the Keyword List and Pre-fetch Cache

4 Experimental Methodology

The smartNewsReader was developed in Java and integrated with a pure-Java Web browser, IceBrowser™ (www.IceSoft.com). We are interested in this browser because it allows us to test the algorithm with different parameters. Netscape source code wasn't available when this project is started. Since the Web is essentially a dynamic environment, its varying network traffic and server-load make the design of a repeatable experiment challenge.

There were studies dedicated to characterizing user access behaviors [2] and pre-fetching/caching evaluation methodologies; see [8] for a survey of the evaluation methods. Roughly, two major evaluation approaches are available: simulation based on a trace of user access and real-time simultaneous evaluation. A trace-based simulation works like an instruction execution trace based machine simulator. It assumes a parameterized test environment with varying network traffic and workload. However, unlike machine simulators that have deterministic timing of instructions and data size of memory reference, the trace-based Web surfing simulations have to make assumptions about when an access is issued and what is its data size. Although the user access timing and reference sizes can be logged in a trace, as well, few of the trace data sets available to public provide such information.

Real-time simultaneous evaluation [7] was initially designed to evaluate multiple Web proxies in parallel by duplicating and passing object requests to each proxy. It complemented to trace-based simulations because it reduced problems of unrealistic test environments, dated and/or inappropriate workloads. Along the line of the approach, we evaluate the performance of the SmartNewsReader by cross-examining multiple news sites. One site is used for neural network training. The trained smart reader then works, simultaneously with a browser without pre-fetching, to retrieve news information from other sites for a period. Since the news of a hot topic has coverage in a number of consecutive days, accuracy of the prediction is expected to increase with more reading of the related news. Following are our experimental settings.

4.1 Topic and Site Selection

In order to establish a user access pattern systematically, the experiment should start with a selection of one or more topics of the same category. Then, select two or more web sites that have intensive reports on the topic. Access to multiple sites will help cross-examine the

accuracy of predictive decisions. In our experiment, we selected the "Clinton Affair" to be the topic because its extensive exposure during the time we conducted the experiment helps collect more data within a short time period. The test sites were two USA local news services: *Today in America Front Page* at MSNBC.com and the *US News Page* at CNN.com**.** The first is full with multimedia documents, while the other is a combination of plain text and image illustrations.

We conducted the experiment during the period from November 18, 1998 to December 16, 1998, five days a week, and two half-an-hour sessions a day.

4.2 Access Pattern Establishment

The neural-net based predictor can identify keywords and establish user access patterns automatically by monitoring user-browsing behaviors. A number of keywords related to the "Clinton Affair" include *Starr, Lewinsky, Scandal, Testimony*, and *Clinton&Scandal*. Each related piece of news must have one or more keywords from the list. To reflect a fact that users may be detracted from their main topic of interest for a while in browsing, we intentionally introduced noises by periodically selecting headlines that do not belong to the topic. We accessed a random link every 10 trials and assumed each access be followed by one minute viewing.

Users may loose interest in a certain topic after a period of time. The system responded to this by updating the associated weights of the relevant keywords.

5 Experimental Results

The system was evaluated with respect to two major performance criteria: hit ratio and waste ratio. The hit ratio refers to the percentage of requested links that are found in the pre-fetch cache to the total number of requests. The waste ratio refers to the percentage of undesired documents that are pre-fetched in the cache. Evaluations focused on the effects of two predictor parameters: threshold and learning rate. We implemented multiple predictors with identical parameters, except the one under test, with the same inputs and accessing sequence.

5.1 Threshold Effect

Figure 4 plots the hit ratios due to pre-fetching with neuron thresholds of 0.25, 0.50 and 0.75 respectively. In agreement with Equation 2.1, the figure shows that a low threshold would yield a high hit ratio for a large cache. Since an extremely small threshold could trigger the pre-fetching unit frequently, it would lead to pre-retrieval of most of the objects on a given URL. By contrast, the higher the threshold, the smoother the curve is and the more accurate the prediction will be. Figure 5 shows the waste ratio in the cache. Expectedly, the lower the threshold value, the more undesired objects that get cached. The cache may be filled up with undesired objects quickly.

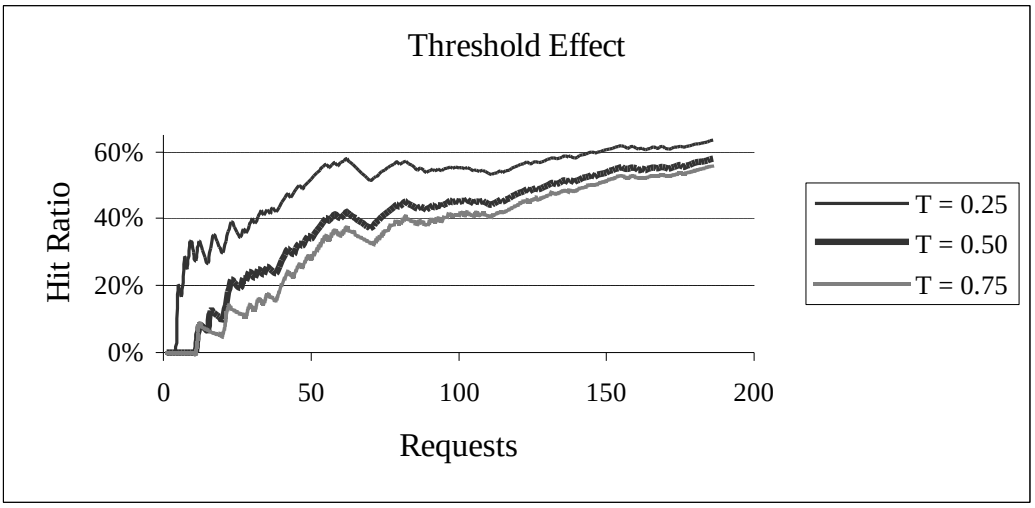


Figure 4: Effects of Threshold on Cache Hit Ratio

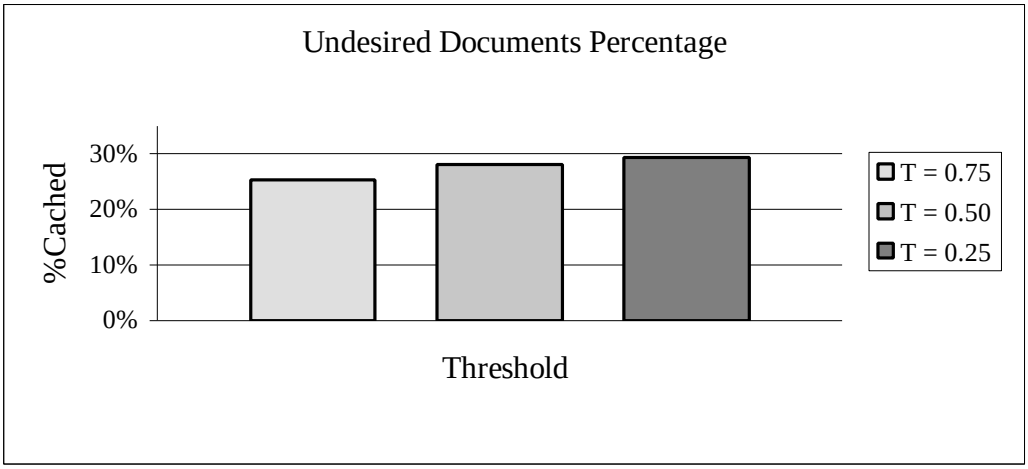


Figure 5. Effects of Threshold on Cache Waste Ratio

We are currently evaluating the effect of the threshold on caches with different cache sizes and trying to derive an optimum threshold for a balance between the hit ratio and utilization ratio of the cache.

5.2 Learning Rate

The learning rate controls the speed a prediction unit responds to a given input. Figure 6 shows the hit ratio due to different learning rates. From the figure, it can be seen that a high learning rate leads to a high hit ratio. It is because high learning rates tend to update the weights quickly. Ripples on the curve are due to the fast updating mechanism.

The figure also shows that the smaller the learning rate, the smoother the curve and the more accurate the prediction unit will be. It is because the weights of trivial keywords are updated with minimal magnitude. This generally affects cache utilization because insignificant weights won't trigger the prediction unit with a lower learning rate. This may not be the case for a unit with a higher learning rate, as shown in Figure 7. Thus, choosing a proper learning rate depends mainly on how fast we want the prediction unit to respond and what percentage of the cache waste we can tolerate.

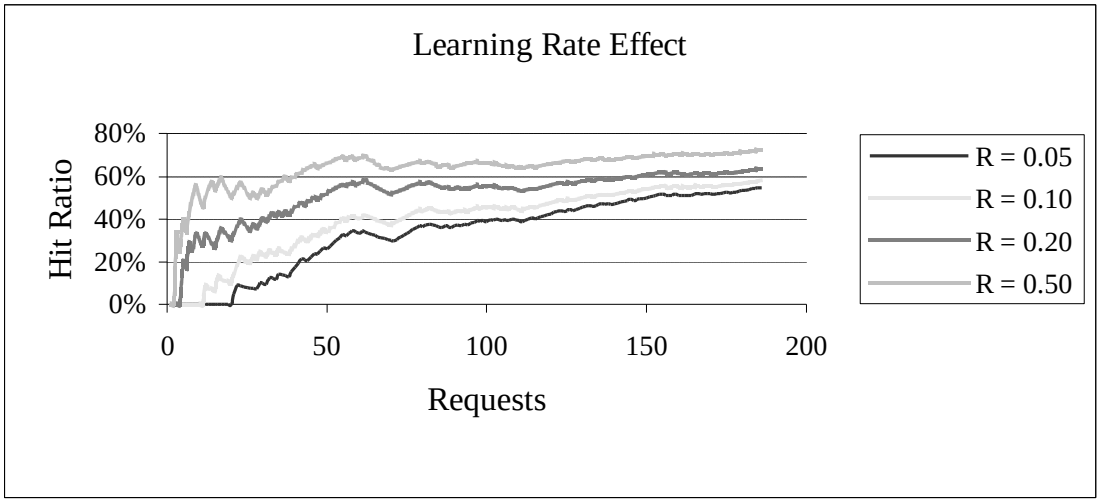


Figure 6. Effect of Learning Rate on Hit Ratio

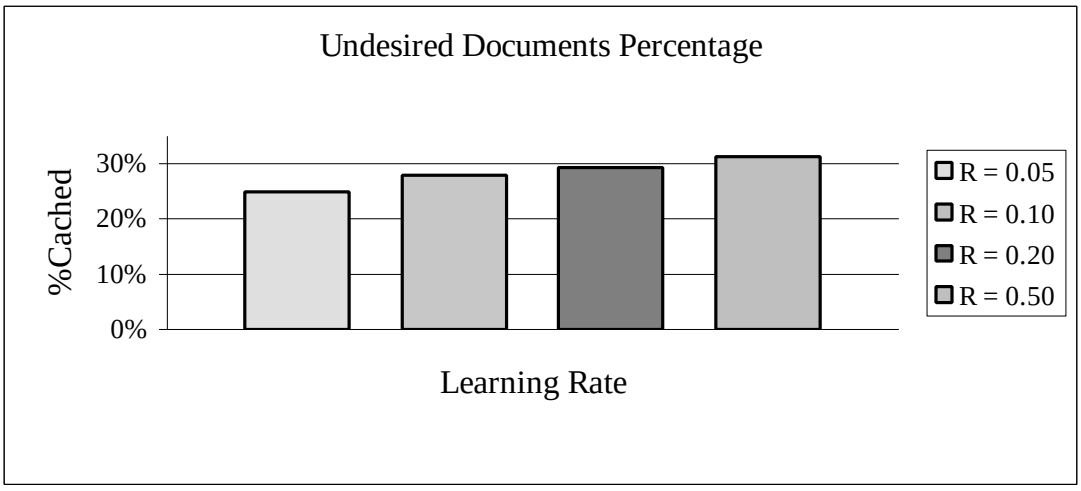


Figure 7. Effect of Learning Rate on Cache Waste Ratio

5.3 Update of Keyword and Purge of Trivial Keywords

Recall that the pre-fetching unit relies on a purging mechanism to identify trivial weights and decrease their impact on the prediction unit so as not to be triggered by accident. Through our tests, we found that on average approximately 4 to 5 trivial keywords are introduced for

each essential keyword. Figure 8 plots the distribution of keyword weights due to Smart-NewsReader. The figure shows the system bounded the trivial weight and nearly minimized the weight in the browsing process.

As outlined in Figure 3, weights of the keywords are reduced whenever their related objects in the cache are to be replaced by the pre-fetching unit. The objects that were never referenced were have the lowest preference value and are to be replaced first. Then weights of the keywords associated with the replaced object were updated accordingly. This guarantees that trivial keywords had minimal effects on the prediction unit.

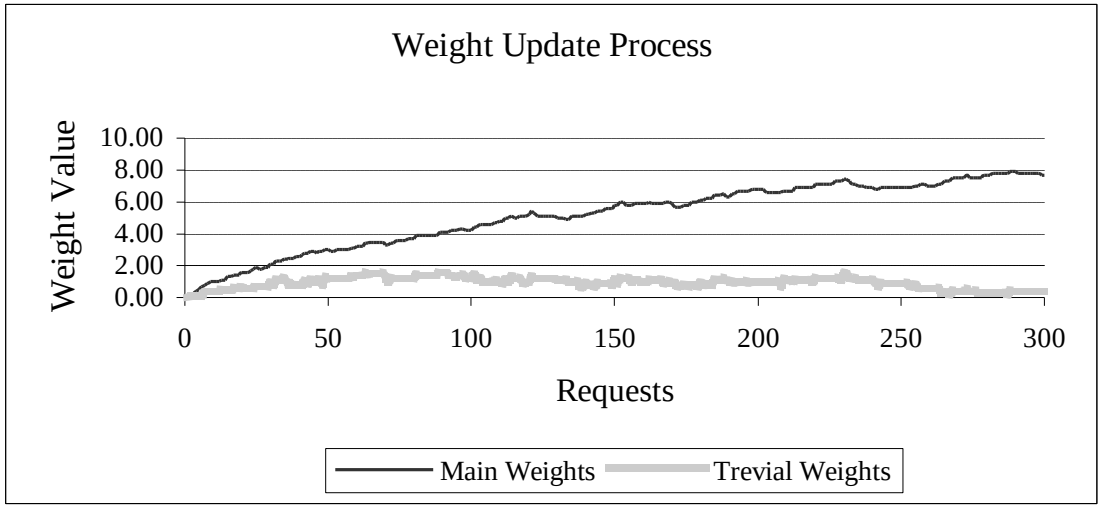


Figure 8. Effect of Unrelated Keywords on Prediction Accuracy

6 Conclusion

This paper has presented a context-specific pre-fetching technique to tolerate WWW latency. It relies on keywords in anchor texts of URLs to capture user access patterns and on neural networks over the keyword set to predict user's future requests. Unlike previous URL graph based techniques, this approach can pre-fetch documents whose URLs are never accessed before. It also features a self-learning capability and good adaptivity to the change of user surfing interests. The technique was implemented in a SmartNewsReader system and cross-examined in a daily browsing of MSNBC and CNN news sites. Experimental results showed

an achievement of approximately 60% hit ratio due to the pre-fetching approach. We note that the SmartNewsReader system was proposed from the perspective of clients. Its predictive pre-fetching technique is readily applicable to client proxies.

Future work will be focusing on the following aspects. First is to construct a prediction unit using neuron networks, instead of single neurons. Besides the keywords, other factors, such as document size and the URL graphs of a hypertext document, will also be taken into account in the design of the predictor. Second is to improve cache utilization. We noticed that the hit ratio saturates at around 60-70% after a reasonable number of requests. More work need to be done in this aspect to reduce the cache waste ratio.

The approach was demonstrated in news reading. We are planning to extend its application to other problem domains, such as on-line shopping and auction, where keywords can be easily identified. For example, in the on-line shopping application, the technique is expected to pre-fetch information on a favorite model while users surf its information from one store to another.

References

[1] V. Almeida, A. Bestavros, M. Crovella, and A. de Oliveira. Characterizing reference locality in the WWW. In *Proceedings of IEEE Conference on Parallel and Distributed Information Systems*, December 1996.

[2] P. Barford, A. Bestavros, A. Bradley, and M. Crovella. Changes in Web client access patterns: Characteristics and caching implications. *World Wide Web: Special Issue on Characterization and Performance Evaluation*.

[3] CNN United States News Home Page. http://www.cnn.com/US/ .

[4] Connectix Home Page. http://www.connectix.com/.

[5] D. Culler, J. Singh, with A. Gupta. *Parallel Computer Architecture: A Hardware/Software Interface*. Morgan Kaufmann Pub. 1998.

[6] C. R. Cunha and C. F. B. Jaccoud. Determining WWW user's next access and its application to prefetching. In *Proc. of the International Symposium on Computers and Communication'97*, July 1997.

[7] B. Davison. Simultaneous proxy evaluation. In *Proceedings of 4th International Web Caching Workshop*, March 1999.

[8] B. Davison. A survey of proxy cache evaluation techniques. In *Proceedings of 4th International Web Caching Workshop*, March 1999.

[9] Go Ahead Home Page. http://www.goahead.com/.

[10] J. Griffioen and R. Appleton. Reducing file system latency using a predictive approach. In *Proceedings of the 1994 Summer USENIX Conference*.

[11] H. Hassoun. *Fundamentals of Artificial Neural Networks*. The MIT Press. 1995

[12] ImsiSoft Home Page. http://www.imsisoft.com/.

[13] T. Kroeger, D. Long, and J. Mogul. Exploring the bounds of Web latency reduction from caching and prefetching. In *Proceedings of USENIX Symposium on Internet Technologies and Systems (USITS'97)*, December 1997.

[14] D. Lee. Methods for Web bandwidth and response time improvement In M. Abrams (ed), *World Wide Web: Beyond the Basics*, Chapter 25, Prentice-Hall Pub. April 1998.

[15] H. Lei and D. Duchamp. An analytical approach to file prefetching. In *Proceedings of USENIX 1997 Annual Technical Conference*, January 1997, pages 275—288.

[16] T. S. Loon and V. Bharghavan. Alleviating the latency and bandwidth problems in WWW browsing. In *Proceedings of USENIX Symposium on Internet Technologies and Systems*, December 1997.

[17] Macca Soft Home Page. http://www.maccasoft.com/webmirror/.

[18] MSNBC Today in America. http://www.msnbc.com/news/affiliates_front.asp

[19] V. Padmanabhan and J. Mogul. Using predictive prefetching to improve world wide web latency. In *Proc. of ACM SIGCOMM Computer Comm. Review*, July 1996.

[20] S. Schechter, M. Krishnan, and M. Smith. Using path profiles to predict HTTP requests. In *Proccedings of the 7th International World Wide Web Conference. Also appeared in Computer Networks and ISDN Systems*, 20(1998), pages 457—467.