DEPARTMENT OF
APPLIED PHYSICS AND ELECTRONICS
UMEÅ UNIVERISTY, SWEDEN

DIGITAL MEDIA LAB

Web prefetching through automatic categorization

Apostolos A. Georgakis
Dept. Applied Physics and Electronics
Umeå University
SE-90187, Umeå Sweden
e-mail: `apostolos.georgakis@tfe.umu.se`

H. Li
Dept. Applied Physics and Electronics
Umeå University
SE-90187, Umeå Sweden
e-mail: `haibo.li@tfe.umu.se`

Abstract

The present report provides a novel transparent and speculative algorithm for content based web page pre-fetching. The proposed algorithm relies on a user profile that is dynamically generated when the user is browsing the Internet and is updated over time. The objective is to reduce the user perceived latency by anticipating future actions. In doing so the adaboost algorithm is used in order to automatically annotate the outbound links of a page to a predefined set of "labels". Afterwards, the links that correspond to labels relevant to the user's preferences are pre-fetched in an effort to reduce the perceived latency when the user is surfing the Internet. A comparison between the proposed algorithm against two other pre-fetching algorithms yield improved cache-hit rates given a moderate bandwidth overhead.

Keywords

1 Introduction

The simplicity of access the variety of information stored on various remote server of the *World Wide Web* (WWW) in accordance with an increase in the size of the documents led to the fact that WWW services have grown to levels where major delays due to congestion are experienced very often. There are several factors influencing the retrieval time of a web document. These factors range from network bandwidth and propagation delay to data loss and the client and server load. A rather old and outdated but still useful study conducted in 1999 by Zona Research Inc. provides evidence that if a Web site takes more than eight seconds to load then the user is much more likely to leave the site [25].

Although several solutions have been implemented to reduce these delays, the problem still exists. The solutions range from the availability of faster connections for the users and alternative communication technologies to faster web servers and increased ISP bandwidth. Furthermore, the usage of proxy servers by the ISPs for cashing of "popular" documents also proved to lessen the *user perceived latency* (UPL). Finally, server side proxies and distributed servers for traffic balancing are also used and can substantially improve typical response times. Still, the UPL exist and is amplified when the user is visiting web pages and documents that never visited in the past.

The UPL depends on the performance of both the web servers and the network latency. Servers may take a while to process a request. The network latency depends on the network congestion and the propagation delay. While the propagation delay is a constant component, which can not be reduced, the network bandwidth is steadily increased by the increase of networks capacities and the installation of new networks. The UPL can also be reduced from the user's side as well without extra costs through the usage of the browser's cache [1, 30].

A crucial issue here is the effective prediction of the next user request and subsequently the modeling of the user actions over time. If an algorithm can accurately determine the next document to be requested and pre-fetch it in time then the UPL tends towards zero. Furthermore, there is no bandwidth misuse. A difficulty emerging here is to determine the document that is more probable to be requested. Unfortunately, since the probability of an accurate prediction is almost always below one, more than one files must be pre-fetched at the same time in order to achieve low UPL. In that case some pre-fetched documents may and will never be used which will result in bandwidth overhead and increased load on the web servers.

In what follows, section 2 provides a compact review of the available literature, section 3 covers the proposed pre-fetching algorithms along with the creation of the user's profiles and section 4 provides the experimental results for the evaluation of the proposed algorithms.

2 Related Work

The UPL can be reduced with the exploitation of one or more cache repositories located either remotely in a proxy server or the web server hosting the documents under consideration or locally by the browser's cache or local proxy server. Local proxy servers are helpful in cases where two or more users have access through the same connection. The usage of a cache can reduce the overall traffic and lessen the need for the client to contact a web server when the documents are stored in the cache. The fact that a proxy server stores documents without taking any initiative for predicting which documents are more probable to be requested (*passive mode*) is a major drawback. This drawback can be avoided and the effectiveness of a caching system can be elevated with the usage of algorithms that predict the demand that particular web documents might have (*active mode*). Subsequently, the documents that are more probable to be requested in the future are prime candidates for caching.

Figure 1 depicts the three alternative methods of communication between a client and a server. In Fig. 1(a) the client requests a document from the cache and the cache either returns the requested document to the client (if the document exist in the cache) or forwards the request to the Web server. In Fig. 1(b) the client requests the desired document from the cache (solid line) whereas the pre-fetching algorithm requests a plethora of relative documents (discontinuous line) that have high probability to be requested in the near future. Finally, in Fig. 1(c) the web server returns to the cache repository the requested document and the pre-loading algorithm, which works
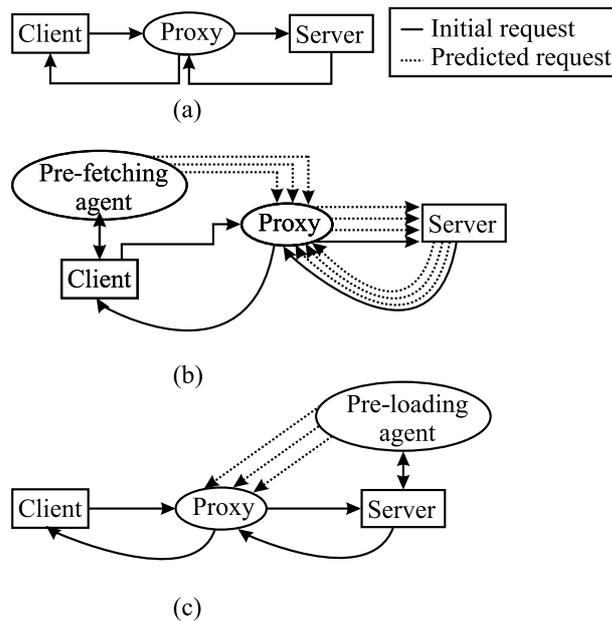
Figure 1: The three different methods of transaction between a client and a web server: (a) through cache repository, (b) a cache repository and a pre-fetching agent working along with the client, (c) a cache repository and a pre-loading agent working in cooperation with the web server.

in parallel with the server, "pushes" to the intermediate cache some extra documents with the anticipation that they might be requested in the future.

Very useful overviews on caching and pre-fetching techniques can be found in [31, 32]. Combinations of caching and pre-fetching have been studied in [10, 32, 34]. Due to inherent limitations in caching pre-fetching has gained much attention lately. Markatos et *al.* proposed the Top-10 pre-fetching algorithm which is a hint based method [17]. Other hint based pre-fetching techniques can be found in [5, 15, 23]. Bestavros in [3] proposed the utilization of a document transition matrix. Data mining techniques have also been used to extract patterns from Web logs [2, 21, 33]. Moreover, the *Prediction-by-Partial-Match* (PPM) has also been employed extensively in [9, 20]. Furthermore, recommendation systems and web usage mining techniques can be used in order to provide hints to a pre-fetching agent over to which links are more probable to be requested [11, 12, 18].

Many attempts have already been made to find the most efficient algorithm for client based pre-fetching. The most naive approach is to pre-fetch the embedded links in a top-to-bottom order regardless of the likelihood to be requested [4]. A similar approach is to pre-fetch randomly selected links. In [19] the server which gets to see requests from several clients makes predictions while individual clients initiate prefetching. A technique for prefetching composite-multimedia rich pages by using partial prefetching is presented in [13]. Discrete Markovian techniques have been applied on the history of Web page references to recognize patterns in the activity of the use [8, 26]. Pirolli and Pitkow in [22, 23] propose the usage of high order Markov Models and n-gram modeling in order to predict the behaviour and future actions of a user. Furthermore, they showed that longer sequence of user visits can lead to a reduced model in size without affecting the ability of accurate predictions. In [7] a combination of different order Markov models is used to devise a model with lower state-space complexity. An extension to the discrete model is the continues Markov chain found in [14].

3 Link pre-fetching: Some preliminary definitions

In this section we describe the formal setting used to address the web-link pre-fetching. After the retrieval of each page and before proceeding to any pre-fetching actions the following series of preprocessing steps are undertaken:

- Identification of the outbound links and the anchored text around them.

- Removal of links that definitely should not be pre-fetched. That links comprises of pages which contains time critical data and URLs that contains the term "?". Moreover, pages already in the cache either from an earlier access or pre-fetch are also ignored except when the content of a page needs to be refreshed. Finally, any dynamically generated pages are also excluded.

Let $\mathcal{W}(t)$ denote the set of web pages that the user have visited until the time instant t. Each individual web page in the set $\mathcal{W}(t)$ contains an arbitrary set of outbound links. Let $\mathcal{L}(t) = \{\mathbf{l}_1, \mathbf{l}_2, \ldots, \mathbf{l}_{|\mathcal{L}(t)|}\}$ denote the set of all the outbound links that comprise the web pages in the set $\mathcal{W}(t)$ where $|\cdot|$ denotes the cardinality of a set. Furthermore, it is generally admissible that in each link corresponds a portion of text which is anchored around the link. For each link and for the text anchored around it the following text processing actions are performed:

- Removal of numbers and punctuation marks. The sole punctuation mark left intact is the full stop. This is done in order to provide a rough sentence delimiter.

- Stopping and stemming is performed using the well-know Porter stemmer [24][1].

After the text preprocessing, the resulted anchored text contains probably a plethora of stems. In order to identify which of the stems are correlated with the user's preferences the so-called *low-to-medium* law will be borrowed from the information retrieval community. According to that law, the bigrams whose frequency is low-to-medium are the most informative ones [16, 29]. The above implies that the stems with high frequency are not informative regarding the actual content; they rather help in the formation of the text inside the sentences and therefore can be eliminated.

Let $\mathcal{B}(t) = \{b_1, b_2, \ldots, b_{|\mathcal{B}(t)|}\}$ denote the set of bigrams that comprise the anchored text around the link until the time instant t that survived the above thresholding process. Each outbound link, $\mathbf{l}_i \in \mathcal{L}(t)$ is associated with a set of bigrams drawn from the set $\mathcal{B}(t)$. For example in the link $\mathbf{l}_i$ corresponds the set $\{b^i_{(1)}, b^i_{(2)}, \ldots, b^i_{(|\mathbf{l}_i|)}\}$ where the notion $(\cdot)$ denotes sampling from a set. Finally, let $\mathcal{K}$ denote the label space. Each link in $\mathcal{L}(t)$ is labeled with only one of the labels in $\mathcal{K}$.

3.1 The Adaboost algorithm

Let $\mathcal{S}(t) = \{(\mathbf{l}_1, \kappa_{(1)}), (\mathbf{l}_2, \kappa_{(2)})), \ldots, (\mathbf{l}_{|\mathcal{L}(t)|}, \kappa_{(|\mathcal{L}(t)|)}))\}$ denote the sequence of training examples where $\mathbf{l}_i \in \mathcal{L}(t)$ and label $\kappa_{(i)} \in \mathcal{K}$. Figure 2 depicts the formation of the set $\mathcal{S}(t)$. Let us assume that we are given a set of real-valued functions, called *weak* learners, which accept as input a sequence of training examples from the set $\mathcal{L}(t)$ and will be denoted by $h_1, \cdots, h_N$ ($h_j : \mathcal{L}(t) \times \mathcal{K} \to \mathbf{R}, j \in \{1, 2, \ldots, N\}$). The adaboost algorithm studies the problem of approximating the $\kappa_{(i)}$ for each $\mathbf{l}_i$ using a linear combination of the weak learners (*training phase*). That is, it tries to find a vector of *predictors* $\mathbf{a} = [a_1, a_2, \cdots, a_N] \in \mathbf{R}^N$ such that the function

$$f(\mathbf{l}_i, \kappa_{(i)}) = \sum_{k=1}^{N} a_k h_k (\mathbf{l}_i, \kappa_{(i)}) \tag{1}$$

to be a "good approximation" of $\kappa_{(i)}$. The previous can be interpreted as follows:

$$|f(\mathbf{l}_i, \kappa_{(i)}) - \kappa_{(i)}| \leq \varepsilon \tag{2}$$

---

[1]It should be noted that although the algorithm under consideration is meant to handle web pages with English content, this can be easily extended with the usage of different stopping lists and different language stemmers.
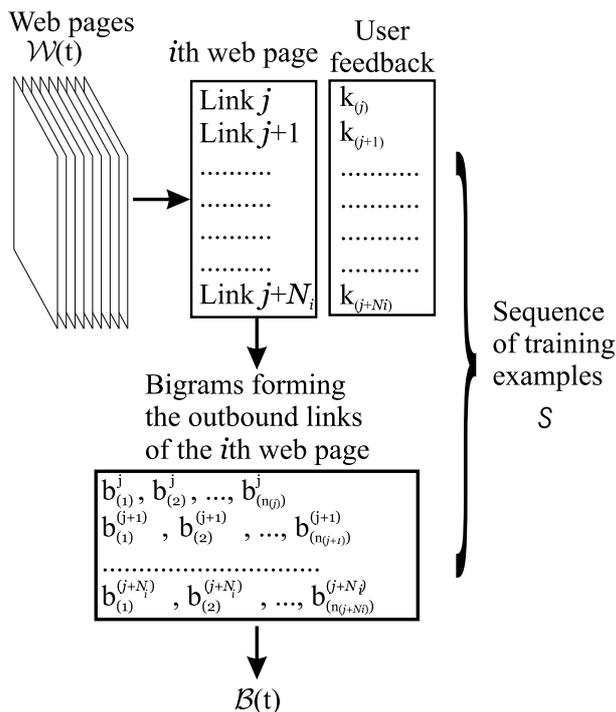
Figure 2: The formation of the sequence of training examples, $\mathcal{S}(t)$, employed by the adaboost algorithm.

where $\varepsilon = 0.5$. The value for epsilon is justified by the fact that the labels in $\mathcal{K}$ are taken from the set N.

Furthermore, Eq. (1) can be used in labelling a new and unseen link, $\mathbf{l}_{new}$, during the (*testing phase*). For example, the sign of the expression

$$\min_{\forall \kappa_i \in \mathcal{K}} \left(f(\mathbf{l}_{new}, \kappa_i) - \kappa_i \right) \tag{3}$$

is interpreted as the label to be assigned to $\mathbf{l}_{new}$. Moreover, the magnitude:

$$\left| f(\mathbf{l}_{new}, \kappa_{new}) - \kappa_{new} \right| \tag{4}$$

will be interpreted as the confidence of the prediction for the label given from Eq. (3).

Furthermore, let $\mathbf{D}_k = \{ d_1^k, d_2^k, \ldots, d_{|\mathcal{L}(t)|}^k \}$ denote a distribution, where $d_i^1 = 1/|\mathcal{L}(t)|, i \in \{1, 2, \ldots, |\mathcal{L}(t)|\}$, which corresponds to the importance that the adaboost algorithm places on each instance of the sample space $\mathcal{S}(t)$. On each iteration $k \in \{1, 2, \ldots, T\}$ the distribution $\mathbf{D}_k$ is used to compute a weak hypothesis h_k over the set $\mathcal{S}(t)$. The weak classifier is trained with more emphasis on certain patterns, using a cost function weighted by a probability distribution $\mathbf{D}_k$ over the training data. The idea behind adaboost is to modify the distribution $\mathbf{D}_k$ over $\mathcal{S}(t)$ in a way that will increase the probability mass of the misclassified parts of the domain $\mathcal{S}(t)$. In this case sampling with replacement (using the probability distribution $\mathbf{D}_k$) can be used to approximate a weighted cost function. Examples with high probability would then occur more often than those with low probability, while some examples may not occur in the sample at all although their probability is not zero. This will subsequently force the classifier to concentrate on the misclassified training examples and avoid misconceptions in the future.

At the time instance $k + 1$, through the usage of the week hypothesis h_k, the boosting algorithm generates the new distribution $\mathbf{D}_{k+1}$ using the following

$$d_n^{k+1} = \frac{d_n^k \exp \left\{ -a_k \kappa_{(n)} h_k(\mathbf{l}_n, \kappa_{(n)}) \right\}}{Z_{k+1}} \tag{5}$$

where Z_{k+1} is a normalization factor equal to

$$Z_{k+1} = \sum_{i=1}^{|\mathcal{L}(t)|} d_i^k \exp(-a_k \kappa_{(i)} h_k(\mathbf{l}_i, \kappa_{(i)})) \tag{6}$$

and $a_k \in \mathbf{a}$. At each iteration of the algorithm, the weak classifier misclassifies a portion of the training samples. The so-called *training error* of the training process is then given by

$$\varepsilon(\mathbf{D}_k) = \sum_{n=1}^{|\mathcal{L}(t)|} d_n^k \lambda(h_k(\mathbf{l}_n, \kappa_{(n)}), \kappa_{(n)}) \tag{7}$$

where

$$\lambda(h_k(\mathbf{l}_n, \kappa_{(n)}), \kappa_{(n)}) = \begin{cases} 1, & |h_k(\mathbf{l}_n, \kappa_{(n)}) - \kappa_{(n)}| \leq \varepsilon \\ 0, & otherwise \end{cases}. \tag{8}$$

The above equation ascertains whether there is a mismatch between the estimated label through h_k and the label already assigned to the link $\mathbf{l}_n$. An important note here is that the training error in Eq. (7) is measured using the distribution which is provided to the weak classifier at the time instant under consideration. The same misclassified training samples might produce different training errors when viewed under different time instances.

The week hypothesis h_k

In the case under consideration the weak hypotheses has the form of a *one-level* decision tree. The test at the root of the tree is a simple check for the presence or absence of a bigram in a given set $\mathbf{l}_i$. That is, if the bigram $b_{(j)}^i$ is present in the anchored text around the ith outbound link or not. Based only on the outcome of this test the weak hypothesis outputs predictions that each label in $\mathcal{K}$ is associated with the ith link. For example, lets assume that the user's profile comprises of web-pages relative to sports and the user is particularly interested in football. A bigram that might appear in a link can be "*scored goals*", and the corresponding predictor is: "If the bigram "*scored goals*" appears in the anchored text around the ith link, then predict that the link belongs to the *sports* category.

Let $b_{(j)}^i$ denote a bigram drawn randomly from $\mathcal{B}(t)$. Using this bigram, a weak hypotheses h_k will be introduced which makes a predictions of the following form [27, 28]:

$$h_k(\mathbf{l}_i, \kappa_{(i)}) = \begin{cases} c_{0r} & if \ b_{(j)}^i \notin \mathbf{l}_i \ and \ \kappa_{(i)} = \kappa_r \\ c_{1r} & if \ b_{(j)}^i \in \mathbf{l}_i \ and \ \kappa_{(i)} = \kappa_r \end{cases} \tag{9}$$

where the $c_{(\cdot)}$s are real numbers. In the above expression, we assign confidence values in both cases where a bigram is present or not in the set $\mathbf{l}_i$. However, intuition rules that we should not assign any confidence value when a bigram is not present ($c_{0r} = 0$) [28].

The weak hypotheses h_k searches all the possible bigrams in $\mathcal{B}(t)$. Given a bigram $b_{(j)}^i$ let the set $\mathcal{L}(t)$ be partitioned into two disjoint subsets $\mathcal{L}_m(t)$ where $m \in \{0, 1\}$, that is, the sets $\mathcal{L}_0(t) = \{\mathbf{l}_n | \mathbf{l}_n \in \mathcal{L}(t), b_{(j)}^i \notin \mathbf{l}_n\}$ and $\mathcal{L}_1(t) = \{\mathbf{l}_n | \mathbf{l}_n \in \mathcal{L}(t), b_{(j)}^i \in \mathbf{l}_n\}^2$. For each bigram $b_{(j)}$ and with the minimization of $\varepsilon(\mathbf{D}_k)$ in mind, the following value is calculated for each label

$$W_b^{mq}(t) = \sum_{i: \mathbf{l}_i \in \mathcal{L}_m(t)} d_i^{(t)} [\![|h_k(\mathbf{l}_n, \kappa_{(n)}) - \kappa_{(n)}| \leq \varepsilon]\!] \tag{10}$$

---

[2]Obviously $\mathcal{L}_0(t) \bigcup \mathcal{L}_1(t) = \mathcal{L}(t)$ and $\mathcal{L}_0(t) \bigcap \mathcal{L}_1(t) = \emptyset$.

where q runs for each available label, $b \in \{\pm1\}$, $[\![\pi]\!]$ equals one if π holds otherwise it is equal to zero and finally,

$$\kappa_{(i)}^q = \begin{cases} +1, & \text{if } q\text{th label was assigned to } i\text{th document} \\ -1, & \text{otherwise} \end{cases} \tag{11}$$

Subsequently, Eq. (6) can be written as follows with the usage of Eq. (10) and by assuming that $a_k h_k(\mathbf{l}_i, \kappa_{(i)}) = h_k(\mathbf{l}_i, \kappa_{(i)})$:

$$
\begin{aligned}
Z_{k+1} &= \sum_{i=1}^{|\mathcal{L}(t)|} d_i^k \exp\{-\kappa_{(i)} h_k(\mathbf{l}_i, \kappa_{(i)})\} \\
&= \sum_q \sum_b \left(\sum_{i:\mathbf{l}_i \in \mathcal{L}_0(t)} d_i^k \exp\{c_{0q}b\} + \sum_{i:\mathbf{l}_i \in \mathcal{L}_1(t)} d_i^k \exp\{-c_{1q}b\} \right) \\
&= \sum_q \sum_b \left(W_b^{0q}(t) + W_b^{1q}(t) \exp\{-c_{1q}b\} \right)
\end{aligned}
\tag{12}
$$

since $c_{0q} = 0, \forall q$ [28]. The local extrema minimum values of Z_{k+1} are given by solving the following:

$$
\begin{aligned}
\nabla Z_{k+1} &= 0 \Rightarrow \\
\frac{\partial Z_{k+1}}{\partial c_{1q}} &\overset{(\forall q)}{=} 0 \Rightarrow \\
W_{-1}^{1q}(t) \exp\{c_{1q}\} + W_{+1}^{1q}(t) \exp\{-c_{1q}\} &\overset{(\forall q)}{=} 0 \Rightarrow \\
c_{1q} &\overset{(\forall q)}{=} \frac{1}{2} \ln \left(\frac{W_{+1}^{1q}(t)}{W_{-1}^{1q}(t)} \right)
\end{aligned}
\tag{13}
$$

and since the second derivative is:

$$
\begin{aligned}
D_{Z_{k+1}} &= \begin{vmatrix} \frac{\partial^2 Z_{k+1}}{\partial c_{1\kappa_{nr}}^2} & \frac{\partial^2 Z_{k+1}}{\partial c_{1\kappa_{nr}} \partial c_{1\kappa_r}} \\ \frac{\partial^2 Z_{k+1}}{\partial c_{1\kappa_{nr}} \partial c_{1\kappa_r}} & \frac{\partial^2 Z_{k+1}}{\partial c_{1\kappa_r}^2} \end{vmatrix} \\
&= \prod_q \frac{\partial^2 Z_{k+1}}{\partial c_{1q}^2} = \prod_q \left(W_{-1}^{1q}(t) + W_{+1}^{1q}(t) \right) \sqrt{\frac{W_{+1}^{1q}(t)}{W_{-1}^{1q}(t)}}
\end{aligned}
\tag{14}
$$

which is positive, meaning that the minimum value for Z_{k+1} is achieved when the c_{1q} are given by Eq. (13). So, finally the value of Z_{k+1} is

$$Z_{k+1} = \sum_q \sum_b W_b^{0q}(t) + \sum_q \left(W_{-1}^{1q}(t) + W_{+1}^{1q}(t) \right) \sqrt{\frac{W_{+1}^{1q}(t)}{W_{-1}^{1q}(t)}}. \tag{15}$$

Prediction based on adaboost

After the training phase and when all the week learners h_k have been computed the algorithm can proceed to anticipate future user requests and pre-fetch some documents. In doing so let us suppose that currently there is page displayed in the user's browser. For each of the links in that page we evaluate the probable label using Eq. (3). For each label we keep a record of the time and frequency of visit. Based on that record we can decide which links to prefetch next.

3.2 General remarks

Some final notes concerning the pre-fetching algorithms in general must be stated here before proceeding to the next subsection:

- If there is a fast transition between web pages then the viewed pages are not taken into consideration. This is needed in order to ensure that the user has spend some time either reading or by simply searching around for something interesting in that particular web page.

- The number of links to be pre-fetched is bounded by the following threshold:

$$\lfloor \frac{\beta\mu(t)}{\mu(s)} \rfloor, \tag{16}$$

  where β corresponds to the total bandwidth available, $\mu(t)$ is the median operator over the time spend by the user in each page, $\mu(s)$ is the median operator over the size of the visited web pages and finally $\lfloor \cdot \rfloor$ denotes the floor function.

- Pre-fetching occurs when the weighting mechanism has assigned weights to each of the outbound links and it takes place during the browser's idle time. During that time multiple threads are employed and each thread receives one link to pre-fetch. After a thread has completed the pre-fetch of the URL that was assigned to it, a new URL is assigned to it and so on.

- If a user-driven event occurs before the completion of the pre-fetch session then all the threads are told to stop executing immediately in order to allow the browser to utilize the entire available bandwidth.

4 Assessment of the pre-fetching capabilities

The comparison of the capabilities for pre-fetching algorithms, particularly the content-based ones, poses up to a degree some difficulties. All the evaluation techniques rely either on artificial data sets or access logs. In the former case, the content based algorithms are useless due to the lack of textual content in the generated sets. In the later case, a problem that often emerges is that due to the dynamic nature of the Internet the content of the web pages tend to change continuously. Therefore, access logs that have been used in the past may not be prime candidates for present studies either due the change of the content or even worse due to the total extinction of a portion of the logged web pages.

Due to the above reasons a trace of the web activity for 10 volunteers in the the Umeå University has been gathered. The trace is divided into two sets, a *training set* which was used to build the personalized profiles for the users and a *test set* which was used in evaluating the performance of the proposed algorithm against two test-bed algorithms which are:

- *top-down* algorithm, corresponding from the first link visible in the top of the page to the bottom

- *random selection* of the links in the web page

4.1 Training set

The training trace consists of nearly 102000 individual web pages. In Fig. 3(a), which depicts the distribution of the outbound links in the trace, it can be seen that a big portion of the trace contains web pages with quite limited outbound links. Furthermore, Fig. 3(b) depicts the distribution of bigrams anchored on each of the outbound links.
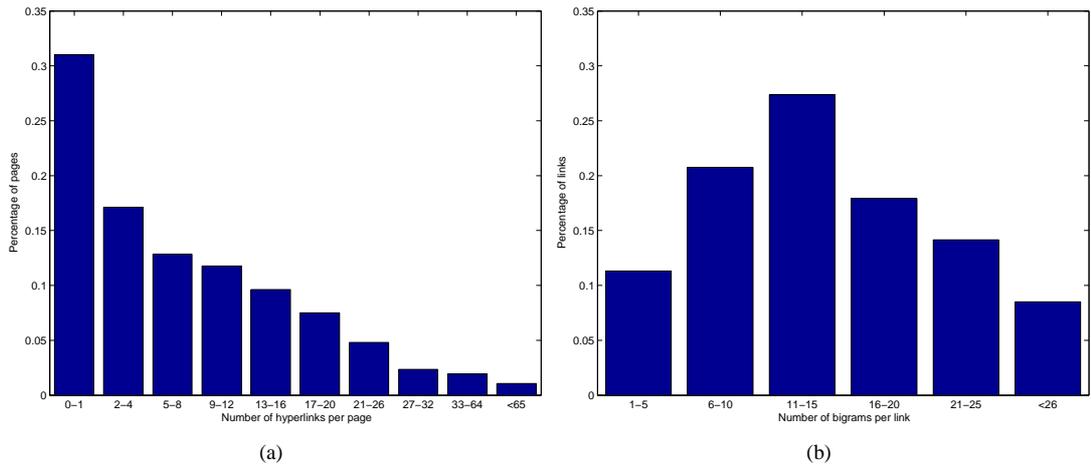
Figure 3: (a) The distribution of the outbound links. (b) The number of bigrams per outbound link.

After the processing of the trace, an individual user profile was created for each of the volunteers and for each of the proposed algorithms based solely on the web pages that comprised his or her activities. The profiles created in the *training phase* of the algorithm were exploited afterwards in the *testing phase*. In the testing phase each volunteer was asked to use a custom-made GUI that resembles a web browser in order to surf the Internet. They were instructed to follow their normal every-day activities regarding their surfing habits. The profiles of each particular volunteer was used by the stand-alone application in order to pre-fetch a portion of the outbound links for each of the web pages that this particular user visited.

4.2 Testing set

During the testing phase an access logs consisting of 12500 individual web pages was captured. This trace is employed in a performance evaluation that relies on the *cache-hit* ratio and the *fractional latency reduction* that the algorithms achieve compared to the *fractional network traffic* that might be caused to the network. In what follows, subsection 4.3 provides further details on the performance evaluation that is based on the cache-hit ratio and the rest of the metrics mentioned at the end of the previous paragraph.

4.3 Hit ratios and network overloads

The performance metrics covered in this subsection are the cache-hit ratio, the fractional latency reduction and the fractional network traffic [6].

Cache-hit ratio is the ratio of pre-fetched pages that a user requested to all the pages the pre-fetch agent retrieved and represents the accuracy of the prediction. The above metric constitutes the accuracy of the prediction. If the hit ratio is high, then the UPL will be low, since almost all requests are served by the local cache. A direct side-effect of the above approach is that if and when a pre-fetching algorithm is very successful in predicting the forthcoming actions the overall bandwidth overhead might be disappointing.

Fractional latency reduction is the ratio between the decrease due to pre-fetching of the observed UPL without a caching system (UPL_{np}) from the UPL with a caching system (UPL_p) to the observed UPL without caching (UPL_{np}). It is important here to describe the mechanism applied for the computation of the latency reduction. Since a pre-fetching algorithms will request, in theory, different documents that might be stored in different web servers, a mechanism is needed to detect and obviate any infrequent and abnormal delays that might affect and degrade the overall evaluation. Therefore, each document is requested from the corresponding web server a multiple number of times during a period of several days and in different time instances during each day.
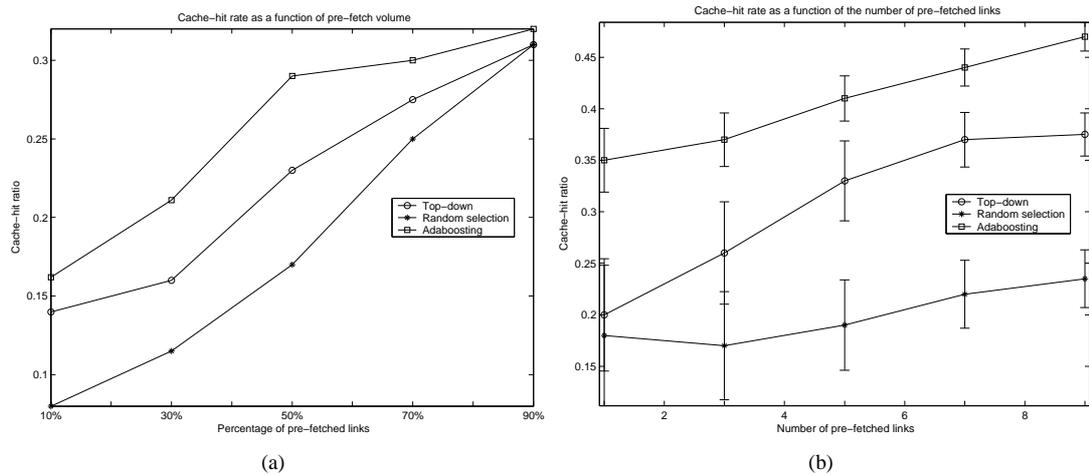
Figure 4: (a) The average cache-hit ratio curves for each of the pre-fetching algorithms in predefined step of the volume of pre-fetched outbound links. (b) The average cache-hit ratio curves for the pre-fetching algorithms for a given number of pre-fetched outbound links.

Afterwards, a median value is computed which represents the "average" response and completion time for the couple <*requested document, hosting web server*>. Through this process, any isolated abnormalities that might affect the evaluation process are eliminated. Finally, before proceeding to the last metric to be employed, we should clarify that the median operator was used instead of the mean value due to its robustness properties against outlier observations.

Fractional network traffic is the ratio between the amount of bytes transmitted from a web server to the user's client to the total number of bytes requested. It represents the bandwidth overhead added to the network traffic of the non pre-fetched case, when pre-fetching is engaged. It is obvious that since some of the future behavior will not be predicted precisely some of the pre-fetched documents will never be requested. These redundant documents add undesired network traffic and should not have been pre-fetched.

Any pre-fetching algorithm should aim to balance the cache-hit ratio and the usefulness against the bandwidth overhead when anticipating future requests. A very strict approach that almost never pre-fetches documents will definitely keep the anticipated overhead low, but will not benefit the user either. If, on the other hand, we decide to pre-fetch a plethora of documents in anticipation of capturing the users next request, then we will definitely have to pay high bandwidth cost.

Figure 4(a) depicts the cache-hit ratio curves for each of the pre-fetching algorithms. The in question figure depicts the average cache-hit curves for all the volunteers in predefined steps of the outbound links volume. It is important to note that interpolation has been applied during the formation of these curves when it was needed. The necessity for interpolation is justified in the cases when the percentage of outbound links in a page is not an integer number. That is, when a page has for example nine outbound links, then pre-fetching 30% of the links implies 2.7 pre-fetches which is unfeasible. In that case, the cache-hit ratio was calculated by interpolating the closest values available. From the figure it is evident that all curves will converge to the same cache-hit point when all the outbound links will be pre-fetched. This is inevitable since the number of useful outbound links is not a function of the pre-fetching mechanism employed, rather than the personal preferences of each user. Furthermore, the adaboost algorithm exhibits better performance than its prime candidate in each pre-fetch volume, with a maximum difference of 6% higher cache-hit rate which is achieved at a 50% pre-fetch volume.

Moreover, in Fig. 4(b) one can see a comparison between the algorithms that is based on the number of outbound links that were pre-fetched. While the previous figure presents an overall evaluation in each pre-fetch level (up until 90% of the volume of outbound links) the later figure is more concise and covers only a fraction of the outbound links (until ten pre-fetched links per page). Again the proposed adaboost approach exhibits better
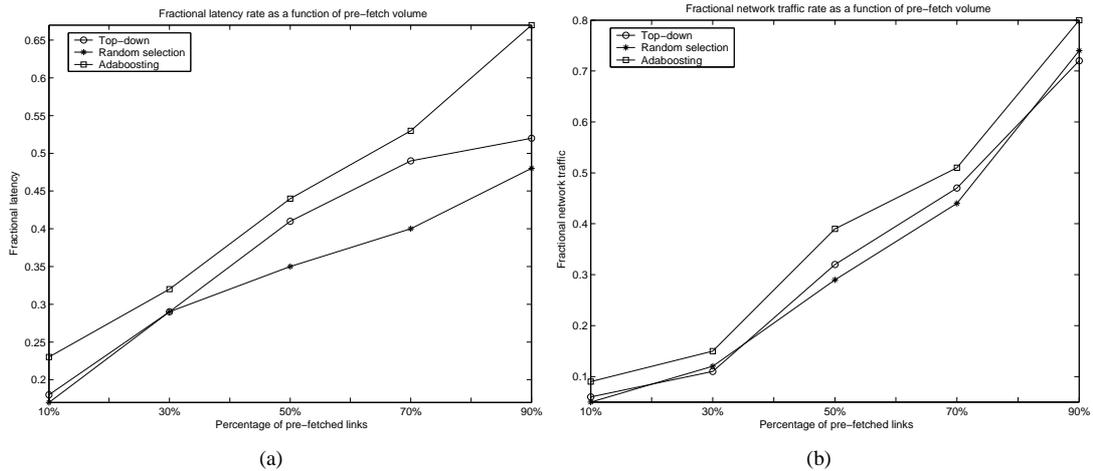
Figure 5: (a) The fractional latency reduction ratio for given volumes of pre-fetched outbound links. (b) The fractional network traffic ratio for specified volumes of pre-fetched outbound links.

characteristics than its prime candidate but at this time the difference is on average 10.1%. In the same figure it can be seen the deviations in each point of the curves.

Finally, as it can be seen in Fig. 5(a), prefetching does reduce network latency in all pre-fetch volumes whereas the bandwidth overhead is more or less the same for all the algorithms (see Fig. 5(b)). The latency is reduced from around 23% when only 10% of the links are retrieved to nearly 67% when 90% of the links have been pre-fetched. At the same time the prime candidate achieves similar but less satisfactory results with a difference in the reduction of the latency in the range from 18% to 52%.

5   Conclusions

In this report we provided a novel transparent and speculative algorithm to model the user's behavior when surfing the Internet. The proposed modeling scheme uses the adaboost algorithm in order to annotate the outbound links in given web page and subsequently pre-fetches links from the most frequently visited categories. Experimental results have proven the superiority of the proposed scheme.

References

[1] C. Aggarwal, J. Wolf, and P. S. Yu. Caching on the world wide web. *IEEE Trans. Knowledge and Data Eng.*, 11(1):95–107, 1999.

[2] B. Berendt and M. Spiliopoulou. Analysis of navigation behavior in web sites integrating multiple information systems. *The VLDB Journal*, 9(1):56–75, 2000.

[3] A. Bestavros. Using speculation to reduce server load and service time on the WWW. In *Proc. of the CIKM95: The 4th ACM Int. Conf. on Information and Knowledge Management*, pages 403–410, November 1995.

[4] K. Chinen and S. Yamaguchi. An interactive prefetching proxy server for improvement of www latency. In *Proc. of the INET'97*, 1997.

[5] E. Cohen, B. Krishnamurthy, , and J. Rexford. Improving end-to-end performance of the web using server volumes and proxy filters. In *Proc. of the SIGCOMM98*, pages 241–253, 1998.

[6] B. D. Davison. *The design and evaluation of web prefetching and caching techniques*. PhD thesis, State University of New Jersey, 2002.

[7] M. Deshpande and G. Karypis. Selective markov models for predicting web-page accesses. In *Proc. of the SIAM Int. Conference on Data Mining (SDM'2001)*, 2001.

[8] D. Duchamp. Prefetching hyperlinks. In *Proc. of the 2nd USENIX Symposium on Internet Technologies and Systems, USITS-99*, October 1999.

[9] L. Fan, P. Cao, and Q. Jacobson. Web prefetching between low-bandwidth clients and proxies: Potential and performance. In *Proc. of the Joint Int. Conf. on Measurement and Modeling of Computer Systems (SIGMETRICS99)*, pages 178–187, 1999.

[10] D. Foygel and D. Strelow. Reducing web latency with hierarchical cache based prefetching. In *Proc. of the Int. Workshop on Scalable Web Services*, pages 103–110, 2000.

[11] W. Gaul and L. Schmidt-Thieme. Recommender systems based on user navigational behavior in the internet. *Behaviormetrika*, 29:1–22, 2002.

[12] M. Géry and H. Haddad. Evaluation of web usage mining approaches for users next request prediction. In *Proc. of the 5th Int. Workshop on Web Information and Data Management (WIDM 2005)*, pages 74–81, 2003.

[13] J. I. Khan and Q. Tao. Partial prefetch for faster surfing in composite hypermedia. In *Proc. of the 3rd USENIX Symposium on Internet Technologies USITS01*, pages 13–24, 2001.

[14] A. Kraiss and G. Weikum. Integrated document caching and prefetching in storage hierarchies based on markov-chain predictions. *The VLDB Journal*, 1998.

[15] T. M. Kroeger, D. D. E. Long, and J. C. Mogul. Exploring the bounds of web latency reduction from caching and pre-fetching. In *Proc. of the USENIX Symposium on Internet Technologies and Systems USITS'97*, pages 13–22, 1997.

[16] D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. Cambridge, MA: MIT Press, 1999.

[17] E. P. Markatos and C. Chronaki. A top-10 approach to prefetching on the Web. In *Proc. of the INET'98*, 1998.

[18] B. Mobasher, H. Dai, T. Luo, and M. Nakagawa. Effective personalization based on association rule discovery from web usage data. In *Proc. of the 3rd Int. Workshop on Web Information and Data Management (WIDM 2001)*, pages 9–15, 2001.

[19] V. N. Padmanabhan and J. C. Mogul. Using predictive prefetching to improve world wide web latency. *Computer Communication Review*, 26(3):22–36, 1996.

[20] T. Palpanas and A. Mendelzon. Web prefetching using partial match prediction. In *Proc. of the Web Caching Workshop*, 1999.

[21] J. Pei, J. Han, B. Mortazavi-Asl, and H. Zhu. Mining access patterns efficiently from web logs. In *Proc. of the Pacific-Asia Conf. in Knowledge Discovery and Data Mining (PAKDD 00)*, 2000.

[22] P. Pirolli and J. E. Pitkow. Distributions of surfers' paths through the World Wide Web: Empirical characterization. *World Wide Web*, 2(1–2):29–45, 1999.

[23] J. E. Pitkow and P. L. Pirolli. Mining longest repeated subsequences to predict World Wide Web surfing. In *Proc. of the 2nd USENIX Symposium on Internet Technologies and Systems, USITS-99*, 1999.

[24] M.F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[25] Zona Research. The economic impacts of unacceptable web-site download speeds, April 1999.

[26] R. R. Sarukkai. Link prediction and path analysis using markov chains. In *Proc. of the 19th International World Wide Web Conf.*, 2000.

[27] R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.

[28] R. E. Schapire and Y. Singer. BoosTexter: A boosting-based system for text categorization. *Machine Learning*, 2/3(39):135–168, 2000.

[29] F. Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, 2002.

[30] J. Shim, P. Scheuermann, and R. Vingralek. Proxy cache algorithms: Design, implementation, and performance. *IEEE Trans. Knowledge and Data Eng.*, 11(4):549–562, 1999.

[31] J. Wang. A survey of web caching schemes for the internet. *ACM Computer Communication Review*, 29(5):36–46, October 1999.

[32] Z. Wang and J. Crowcroft. Prefetching in world wide web. In *Proc. of the IEEE Global Internet*, pages 28–32, 1996.

[33] Q. Yang and H. H. Zhang. Web-log mining for predictive web caching. *IEEE Trans. on Knowledge and Data Eng.*, 15(4):1050–1053, 2003.

[34] Q. Yang, H. H. Zhang, and I. T. Y. Li. Mining Web logs for prediction models in WWW caching and prefetching. In *Proc. of the Seventh ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 473–478, 2001.