Exploiting User Behaviour in Prefetching WWW Documents: Abdulmotaleb El-Saddik, Carsten Griwodz, and Ralf Steinmetz
accepted for International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services'98, Oslo, Norway Sept. 08-11, 1998

Exploiting User Behaviour in Prefetching WWW Documents

Abdulmotaleb El-Saddik[1], *Carsten Griwodz*[1], *Ralf Steinmetz*[1,2]

[1]
Industrial Process and System Communications
Dept. of Electrical Eng. & Information Technology
Darmstadt University of Technology
Merckstr. 25 • D-64283 Darmstadt • Germany

[2]
GMD IPSI
German National Research Center
for Information Technology
Dolivostr. 15 • D-64293 Darmstadt • Germany

{Abdulmotaleb.El-Saddik,Carsten.Griwodz,Ralf.Steinmetz}@kom.tu-darmstadt.de

Abstract

As the popularity of the World Wide Web increases, the amount of traffic results in major congestion problems for the retrieval of data over wide distances. To react to this, users and browser builders have implemented various prefetching and parallel retrieval mechanisms, which initiate retrieval of documents that may be required later. This additional traffic is even worsening the situation. Since we believe that this will remain the general approach for quite a while, we try to make use of the general technique but try to reduce the destructive effects by retrieving less content which remains finally unread.

In our user-specific prefetch mechanism, the prefetching system gathers references by parsing the HTML pages the user browses, identifies the links to other pages, and puts the words describing the links into a keyword list. If such a word was already present in the list, its associated weight is incremented. Otherwise it is added to the table and a weighting factor allocated. We have designed and implemented a client based proxy-server with this mechanism. This paper shows the design and implementation of this prefetching proxy server, presents results and general considerations on this technique.

1 INTRODUCTION

The simplicity of access to a variety of information stored on remote locations led to the fact that World Wide Web 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 are network bandwidth, propagation delay, data loss, and the client and server load.

Although several approaches have been implemented to reduce these delays, the problem still exists.

The latency of retrieval operations depends on the performance of servers and on the network latency. Servers may take a while to process a request or may refuse to accept it due to over-load. 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 rest of the paper is organized as follows. Section 2 discusses related work. Section 3 describes our approach in designing a newer prefetch proxy. The description of the system architecture is followed by initial experiences in section 4, section 5 concludes the paper.

2 Related Work

Various approaches to solve the problem of retrieval delays were presented in the past. Most of them deal with caching ([Abra][Chan][Mark96]). However, the effectiveness of caching to reduce the WWW latency is small. Several papers report that the hit-rate of the caching proxy server is under 50% ([Abra][Chan]). Actually, we observe that the hit-rate is constantly falling because the number of documents and the size of those documents grows faster than the typical proxy server.

An alternative approach to reduce delay experienced by the end user is prefetching. The majority of users browse the Web by following hyperlinks from one page to another with a general idea or topic in mind. While users read the downloaded page, there is a communication pause. Since there is a general topic that drives the user´s navigation, this time can be used to prefetch pages that are likely to be accessed as a follow up to the current page. Actually, prefetching does not reduce latency, it only exploits the time the user spends reading, and thereby theoretically decreases the experienced access time. Practically, the growing number of users of this technique destroys the effects by increasing considerably the overall amount of data transfers on the networks.

Prefetching has some problems and drawbacks. One of these problems is to decide or to predict what and when to prefetch. Another problem is the large amount of traffic. Both problems can be addressed by increasing the hit-rate of the prefetching mechanism. There is a long list of references considering prefetching of WWW pages. Each of these references deals with different situations and different mechanisms. In general we can consider the following strategies:

- Non statistical prefetching
- Servers’ access statistics
- Users’ personal preferences

Chinen ([Chin]) prefetches referenced pages. The prefetching system he suggested is to prefetch all the referenced pages of an HTML document at each request of the client. This scheme reduces the relative latency experienced by the user, but it suffers from the fact that there is no speculative prediction.

Server access statistics to prefetch WWW documents is investigated by Markatos ([Mark96]), Bestavros ([Best95]), Doi ([Doi96]), and Padmanhaban ([PaMo96]). Such a strategy is based on the observation of a client's access pattern. They exploit the fact that users do generally not access files at random. Although the access pattern of a user is not deterministic, the server can obtain a good idea of the files likely to be accessed next based on the currently accessed file. In order to achieve high hit-rates, a long observation time is required. It is difficult to react to new trends in user behaviour immediately using this scheme. Padmanhaban and Mogul ([PaMo96]) propose protocol modifications on both server and clients to keep state in the server.

The use of users' personal preferences is also a way to prefetch document in the WWW. This mechanism is implemented by Microsoft's Channel Bar ([Micr]) and Netscape's Netcaster ([Nets]), which enable (so-called) push delivery of information and off-line browsing. This technology enables users to subscribe to channels, which describe an interest profile for a user. When the user starts the browser, the prefetching mechanism that is built into the browser contacts the servers specified by the channel information and retrieves all appropriate information for off-line browsing by the user. The user doesn't have to request or search manually for the information.

All of these approaches work with long-term constant interests of the users. No approach considers the appropriateness of prefetching for short-term interests of the user. We consider questions such as "where can I find information on MPEG-2 encoders?" or "how do I travel in Tibet?" short-term interests, which concern the user for some minutes up to a few days but are not worth the manual specification of a user-specific profile. Since we believe that all of these prefetching techniques are not specific enough to cover the short-term interests of a user, and thus, that too much unnecessary information is retrieved, we have designed an alternative approach to prefetching.

3 Design of a new Prefetching-Proxy Algorithm

3.1 Design considerations

The user's browsing strategy can be determined by observing the user's earlier behaviour. Most user events are of the type "select hyperlink on current document" (52%) and "browser-back-button" (41%) ([Pitk95]). For effective prefetching, a mechanism should take these events into account. In other words, many web sessions start with an idea of what to search. In general, the user detects very quickly whether a piece of information is interesting for him or not and decides to move on. Other browsing strategies are less relevant to prefetching. It is not useful to prefetch when the major user event is "goto URL". Users who demonstrate this kind of behaviour are just surfing in unpredictable ways and find information randomly. However, this is rare and additionally, those users move on slowly because they don't look for specific topics but consume all input. Models which describe and determine the different browsing strategies are investigated in [CuJa97].

We believe that very specific interests are (in the mind of the user) expressed by a keyword or set of keywords; these keywords are also the basis of any search he performs, typically by starting with a typical search engine such as Alta Vista. For such a user behaviour, the prefetching of a subset of links is viable and we have implemented a mechanism to exploit it. Our idea is to make use of the Web technology as is and to implement a mechanism that can be used with all clients without replacing servers and protocols. Our mechanism differs from those which prefetch all pages referenced by the current page, by loading only those referenced pages which are predicted to be interesting to the user and thus, probably visited.

3.2 Structure of a Web page

Usually a Web page is a simple text document with in-line images and references to other pages (links). In general, all web documents can be grouped into two categories with regard to the time at which the content of the page is determined.

The first category consists of *static* documents. The content of these is determined at creation time. Although some documents of the static category are generated dynamically by server processes, these contents are independent from user interaction and are presented in the same way whenever they are accessed. Typically, the content remains unchanged for a while, so this kind of document is very suitable for caching and prefetching.

The second category are *dynamic* documents. This group can be subdivided into a *fully dynamical* category and an *active* category. A dynamical document is created by a web server when the document is requested (e.g. CGI-scripts). Dynamic documents can not be cached or prefetched because the results of a request are entirely dependent on a specific situation. An active document includes a program that runs on the client machine, e.g. a Java applet, and which may communicate with one or more server processes. The client parts of active documents may be prefetched and cached, since they remain unchanged for longer periods and their download operation is similar to the download of a computer program.

3.3 Description of the Prefetching Algorithm

We have implemented a client side architecture that addresses the prefetching of static documents. Our approach includes criteria which are determined by observing the user behaviour. In our implementation the collection of the user-specific criteria is accomplished by a „clicked-word-list". This list is the database in which the frequencies of words extracted from the selected anchors are collected (see figure 1). As soon as an HTML-document is retrieved by the proxy, the prediction algorithm parses this document, builds a list of current links and anchor texts. The words in this list are compared to a user database of user-specific keywords. This database grows while the user is browsing. Each time a link is clicked, the words in the anchor text of the link are recorded in the database, respectively their counters are incremented. The database is kept for future sessions. The possibility of deleting this database is also given.

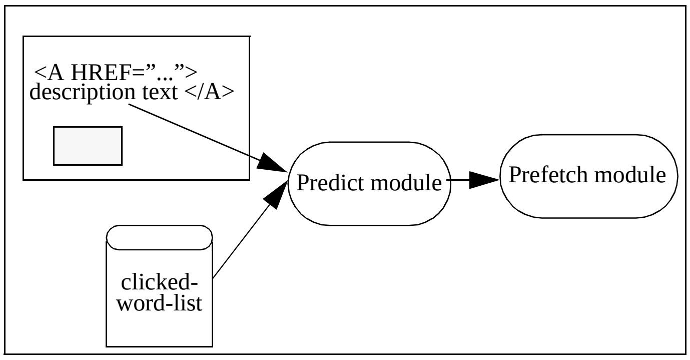

Figure 1: The predict module compares the anchor text with the user's clicked word list

The advantages of the mechanism are:

- Web latency is reduced: The system prefetches the predicted pages until all predicted pages are retrieved or until a new request is made by the user, whichever occurs first.
- The prefetching is not a recursive copying of all references on a page but only of the relevant documents.
- Using the user's preferences: The algorithm gathers the links by parsing the HTML document and by comparing the words in the anchor text with the user's database.
- abandon the use of statistics

The disadvantages of the mechanism are:

- Synonyms, and words that contain less than 4 lowercase letters are not taken into consideration: we have to add thesauri
- the large traffic that comes generally with prefetching

3.4 Modules

This section describes the module of the prefetching proxy server called MObile proxy helPER (MOPER). Figure 2 shows the system architecture of MOPER which consists of three modules: ClickedWord module, Predict module and Prefetch module.

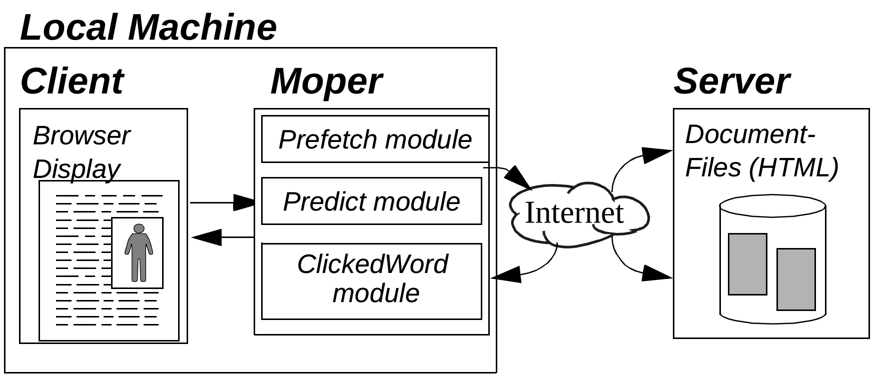

Figure 2: System Architecture

ClickedWord module

The main task of this module is to identify the anchor associated with a click, to extract the relevant words and to put them into a list with their associated URLs. In the current version of our implementation we consider only words which consists of more than three letters, except words in capital letters. In future implementations we will incorporate thesauri to decide which words are worth considering and which are not.

The requested HTML document is parsed on the fly. The list of all included URL and their anchor text are saved until the browser sends a new request. If the new requested URL matches one of the URLs in the list, the words of the anchor text are entered in a database. The words in the database reflect the user's preferences, which are used in the predictive algorithm. Every word entered in the list has an associated counter which will be incremented when the word occurs once again. We consider case insensitive words (Java = JAVA = java)

.

java	13
multimedia	10
computing	9
prefetching	9
caching	7

Figure 3: Excerpt from clicked-word-list

Prediction module

This module compares the words in the clicked-word-list (Figure 3) with the words describing the links on the actual HTML page, starts the prediction algorithm, and sends the URLs to the prefetch module according to the correct order of their importance to the user. A prediction range is assigned to these URLs.

Prefetch module

The prefetch module preserves a list of URLs to load. URLs with a higher prediction range will be prefetched first. For more interactivity this module implements a stop method which enables the user to stop all on-line prefetching activities.

4 Implementation results

Moper (MObile proxy helPER) is a WWW proxy server. Moper is installed on the local machine to support prefetching of web pages based on user preferences, thus reducing the waiting time of the user. Our application is a proxy server for the actual browser and poses as a browser for the remote server. We decided to implement our mechanism in Java ([Sun]) to overcome platform dependencies. We tested and evaluated our system on various platforms (Win 95, Win NT, Linux and Solaris).

To compare the efficiency of profile-dependent prefetching with the prefetching of all referenced pages, Moper is equipped with a couple of switches to make this decision. A small survey on educational web sites related to multimedia was made to inquire about relevant settings for these switches. We found that bigger cross-linking pages contain references (links) to 100 other pages and more, but we found only some pages with less than 7 links. The average number of links on the set of pages that were taken into account in our survey was 17.4. We consider it noteworthy that only 6.5% of these pages were greater than 15 kilobytes when the referenced images were not considered. Based on the results of the cross-linking survey, we chose to restrict the number of prefetched links per retrieved page to 15.

To compare our algorithm with unrestricted prefetching operations, we tested Moper in two different configurations. In the first one Moper made use of our approach and was configured to prefetch a maximum of 15 referenced pages if the words in the anchor text match the words in the clicked-word-list. The second configuration did not use the restriction and was set to prefetch any 15 referenced pages in the requested page, which reflects the approach taken by other prefetch mechanisms ([Doi96][Chin]).

Using these settings, we made a couple of sample web surfing sessions without taking the download delay per page into account (all pages were loaded completely before the next request was issued). This approach does not give us any indication of the speed increase or decrease of our approach in comparison with other prefetching approaches but instead, provides us with an indication of the better efficiency of our prefetching approach in terms of transmission overhead, as well as hit-rate comparisons between ours and the simpler approach.

We defined the induced traffic as the number of prefetched pages and the hit-rate as

$$\text{Hit-rate} = \frac{\text{Responses from prefetch-proxy}}{\text{Requests}}$$

Figure 4(a) and Figure 5(a) present the results of various surfing sessions when our approach is used, Figure 4(b) and Figure 5(b) present the results of the typical approach. For each exemplary surfing sessions, a different general topic was chosen, ranging from technical issues such as Java programming to private interests such as travelling to South America.

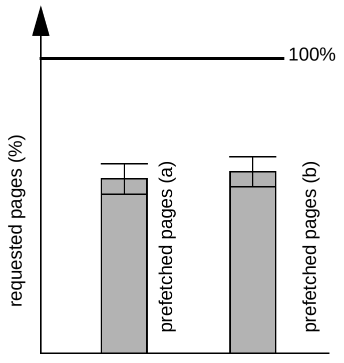

Figure 4: Hit-rate

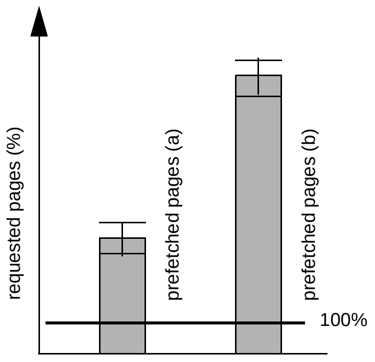

Figure 5: Induced traffic

As shown in Figure 4, both configurations make approximately the same hit-rate (60%). While the configuration according to our idea of the user's behaviour has a hit-rate of about 61.05%, the random prefetching 15 referenced pages in the requested page achieves a slightly better hit-rate of 63.86%. Obviously, our approach will never have a better hit-rate than the trivial approach, but the difference is marginal in the experiments.

The advantage of our mechanism concerning the reduction of unnecessary traffic in the network, however, is considerable. Figure 5 shows that the overhead induced by the general prefetching technique (Figure 5(b)) is 10.04 times the amount of data compared with the pages that are actually visited by the user, our approach (Figure 5(a)) reduces this overhead to 4.29 times the number of actually visited pages.

5 Conclusion and Future work

Prefetching is a speculative process. If the guess is wrong, a (high) price may have been paid by the community of Internet users at large for nothing. If the guess is correct, on the other hand, time is saved for the individual user. Prefetching is only sensible if the payment for the Internet connection depends on time, not on the amount of transferred data.

In this paper we have described a predictive prefetching mechanism for the World Wide Web to improve the access time without the extraneous penalty in network load that is typical for applications that prefetch speculatively. We consider that the links appropriate for prefetching come from the current user page. In our model the system guesses the next user request and prefetched those referenced pages, whose words in the anchor text are found in the user's clicked-word-list. We found out that our model reduces the bandwidth used by other prefetch systems which prefetch all referenced pages by the factor 2.34 for browsing sessions aimed at a focused information search, and that the hit-rate is approximately still the same.

We are now in the process of incorporating thesauri inside our prefetch module to increase the hit-rate, and to have better decision about words which may be entered in the user's database.

Another way of works in which we are interested is to use our proxy as a blocker, like per example, porno blocker, advertisements blocker or racism blocker. We do not need to know the IP-address or the domain name of servers related to the topic to be blocked, all we need is to define the words which should not be requested. The concept may even be extended to editing such links out of the presented pages.

6 Acknowledgments

We thank testers and users of our system. We also thank our colleagues Cornelia Seeberg, Stephan Fischer and Michael Liepert for useful comments on this paper.

References

[BeFF96] T.Berners-Lee, R. Fiedling, and H.Frystyk. "Hypertext Transfer Protocol-HTTP/1.0", RFC 1945, May, 1996.

[Best95] Azer Bestavros, Using Speculation to Reduce Server Load and Service Time on the WWW. Technical Report TR-95-006, CS Dept., Boston University, 1995

[CuJa97] Carlos Cuncha, Carlos Jaccoud, Determining WWW User's Next Access and its Application to prefetching, International Symposium on Computer and Communication, 1997

[Doi96] Katsuo Doi, “WWW Access by Proactively Controlled Caching Proxy”, Sharp Technical Journal, No. 66, December 1996.

[Mark96] Evangelos Markatos, Main Memory Caching of Web Documents. In Electronic Proccedings of the fifth International World Wide Web Conferece, Paris, France, May 6-10, 1996.

[PaMo96] V.N. Padmanabhan, J.C. Mogul, "Using Predictive Prefetching to Improve World Wide Web Latency", ACM SIGCOM, Computer Communication Review, 1996.

[Pitk95] Catledge Pitkow, Characterizing Browsing Strategies in the World Wide Web , Technical Report 95-13, Graphics, Visualization and Usability Center, Georgis Tech, USA, 1995

[YeMc96] N.J. Yeager, R.E.McGrath, Web Server Technology, Morgan Kaufmann Publishers Inc., 1996.

[Abra] Marc Abrams et al. "Caching Proxies: Limitations and Potentials", http://ei.cs.vt.edu/~succeed/WWW4/WWW4.html

[Chan] Anawat Chankhunthod et al, "A Hierarchical Internet Object Cache", Usenix 1996 Technical Conference, http://excalibur.usc.edu/cache-html/cache.html

[Chin] Ken-ichi Chinen, "WWW Collector Home Page", http://shika.aist-nara.ac.jp/products/wcol/wcol.html

[Micr] Microsoft active Channel guide, http://www.iechannelguide.com/

[Nets] Netscape Netcaster,http://home.netscape.com

[Sun] Sun Microsystems Inc.: "The Java Tutorial``, http://Java.sun.com