UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

DEPARTMENT OF COMPUTER ENGINEERING

Evaluation, Analysis and Adaptation of Web Prefetching Techniques in Current Web

Thesis submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computing

Josep Domènech i de Soria

Ph.D. advisors:
Dr. Ana Pont Sanjuán
Dr. José A. Gil Salinas

València, March 2007

To my family.
To my sweetheart.
And, of course, to you.

Acknowledgments

Firstly, I would like to acknowledge the work of Dr. Julio Sahuquillo as advisor of this Ph.D. thesis, although his name does not appear on the cover due to the limit on the number of advisors established by the university. His help, together with the contribution and knowledge of Prof. Ana Pont and Dr. José A. Gil have been crucial for the development of this work.

I would also like to thank Ana, my girlfriend, for her help to improve my English and, therefore, the quality of the thesis; for her time and support, and for being always by my side.

My family has been also very important to encourage me to achieve this goal. Thank you Pepe, Silvia, Lluís, Paloma and Sergi.

Finally, I much appreciate the good times I have shared with my former and current colleagues, who have become good friends.

Thank you, one and all.

Abstract

This dissertation is focused on the study of the prefetching technique applied to the World Wide Web. This technique lies in processing (e.g., downloading) a Web request before the user actually makes it. By doing so, the waiting time perceived by the user can be reduced, which is the main goal of the Web prefetching techniques.

The study of the state of the art about Web prefetching showed the heterogeneity that exists in its performance evaluation. This heterogeneity is mainly focused on four issues: i) there was no open framework to simulate and evaluate the already proposed prefetching techniques; ii) no uniform selection of the performance indexes to be maximized, or even their definition; iii) no comparative studies of prediction algorithms taking into account the costs and benefits of web prefetching at the same time; and iv) the evaluation of techniques under very different or few significant workloads.

During the research work, we have contributed to homogenizing the evaluation of prefetching performance by developing an open simulation framework that reproduces in detail all the aspects that impact on prefetching performance. In addition, prefetching performance metrics have been analyzed in order to clarify their definition and detect the most meaningful from the user's point of view. We also proposed an evaluation methodology to consider the cost and the benefit of prefetching at the same time. Finally, the importance of using current workloads to evaluate prefetching techniques has been highlighted; otherwise wrong conclusions could be achieved.

The potential benefits of each web prefetching architecture were analyzed, finding that collaborative predictors could reduce almost all the latency perceived by users. The first step to develop a collaborative predictor is to make predictions at the server, so this thesis is focused on an architecture with a server-located predictor.

The environment conditions that can be found in the web are also heterogeneous, so the optimal way of conducting web prefetching may be different depending on these conditions. We studied under which environments it is better to maximize either the amount of prefetched objects or the size of these objects. The results showed that when the user's available bandwidth increases prefetching techniques should prefetch a high amount of objects; whereas when the server processing time decreases, the amount of prefetched bytes should be maximized.

By analyzing current prefetching techniques we found that, due to the evolution of the Web, some prefetching techniques that worked well some years ago have their performance dropped on the current web. This fact is mainly caused by the high amount of images or similar objects that are currently included in each page. These images notably interfere on the accesses prediction. Taking into account this unwanted effect, a novel algorithm (DDG) has been proposed and tested dealing with the characteristics of the current web. The DDG algorithm distinguishes between container objects (HTMLs) and embedded objects (e.g., images) to create the prediction model and to make the predictions. Results showed that, given an amount of extra requests, DDG always reduces the latency more than the other existing algorithms. In addition, these results were achieved without increasing the order of complexity of the existing algorithms.

Resum

Aquesta tesi tracta la tècnica de prebusca aplicada a la World Wide Web. La tècnica es fonamenta a processar (p.e. descarregar) una petició Web abans que l'usuari la realitze perquè, d'aquesta manera, aconseguim reduir el temps d'espera percebut per l'usuari final, objectiu últim de les tècniques de prebusca en la Web.

L'estudi de l'estat de l'art de la prebusca web va posar de manifest una gran heterogeneïtat en com s'avaluaven les prestacions. Hi podem destacar quatre aspectes principals: i) no existia un entorn de desenvolupament lliure en què simular i avaluar les tècniques de prebusca ja propostes; ii) el conjunt d'índexs de prestacions a maximitzar no era uniforme, o inclús la seua pròpia definició; iii) no n'hi havia cap estudi comparatiu d'algoritmes de predicció que tinguera en compte els costos i els beneficis de la prebusca de forma conjunta; i iv) l'avaluació de les tècniques propostes es realitzava baix càrregues de treball molt distintes o poc significatives.

Durant la investigació, hem contribuït a homogeneïtzar l'avaluació de prestacions per mitjà del desenvolupament d'un entorn de simulació que reprodueïx detalladament tots els aspectes que afecten les prestacions de la prebusca. A més, hem analitzat les mètriques de prestacions utilitzades en els estudis de prebusca per a aclarir la seua definició i detectar quines eren les més significatives des del punt de vista de l'usuari. També hem proposat una metodologia d'avaluació que considere el cost i el benefici de la prebusca de forma conjunta. Finalment, s'ha ressaltat la importància d'utilitzar càrregues de treball actuals per a avaluar les prestacions de la prebusca, ja que, en cas contrari, podríem obtindre conclusions incorrectes.

L'anàlisi dels beneficis potencials de cada possible arquitectura del sistema ens va mostrar que, potencialment, els predictors col·laboratius podrien reduir quasi tota la latència percebuda pels usuaris. El primer pas per a desenvolupar un predictor col·laboratiu és fer les prediccions des del servidor, per la qual cosa en aquesta tesi ens centrem en una arquitectura de prebusca en la que el predictor d'accessos està situat en el servidor web.

D'altra banda, les condicions de l'entorn que ens podem trobar a la web són també heterogènies, per la qual cosa la forma òptima d'utilitzar la prebusca podria ser també distinta depenent d'este entorn. Per això, hem estudiat davall quines condicions és millor maximitzar o el nombre d'objectes prebuscats o la grandària de tals objectes, per a obtindre les màximes prestacions. Els resultats mostren que, quan l'ample de

banda disponible per l'usuari augmenta, les tècniques de prebusca haurien de ser més propenses a prebuscar una gran quantitat d'objectes mentres que quan es redueïx el temps de procés del servidor, és la quantitat de bytes prebuscats la variable que s'ha de maximitzar.

Quan analitzem les tècniques de prebusca existents trobem que, a causa de l'evolució que ha patit la Web, algunes de les tècniques que van funcionar bé fa uns anys ja no obtenen les mateixes prestacions a la Web actual. Açò es deu principalment a la gran quantitat d'imatges o objectes incrustats que s'inclouen actualment a cada pàgina web. Aquestes imatges interfereixen notablement en la predicció d'accessos de l'usuari. Considerant aquest efecte no desitjat, hem proposat i avaluat un nou algoritme (DDG) que maneja aquestes característiques presents a la web actual. Per a crear el model de predicció, l'algoritme DDG diferència els objectes entre contenidors (HTMLs) i incrustats (p.e. imatges). Els resultats experimentals mostren que, donat un increment de peticions prefixat, l'algoritme DDG sempre reduïex més la latència que els algoritmes existents. A més, aquestos resultats s'han aconseguit sense augmentar l'orde de complexitat respecte als algoritmes existents.

Resumen

Esta tesis trata la técnica de prebúsqueda aplicada a la World Wide Web. Esta técnica se fundamenta en procesar (p.e. descargar) una petición Web antes de que el usuario la realice para que, de esta forma, consigamos reducir el tiempo de espera percibido por el usuario final, objetivo último de las técnicas de prebúsqueda en la Web.

El estudio del estado del arte de la prebúsqueda web puso de manifiesto una gran heterogeneidad en cómo se evaluaban las prestaciones. Podemos destacar cuatro aspectos principales: i) no existía un entorno libre de desarrollo en el que simular y evaluar las técnicas de prebúsqueda ya propuestas; ii) el conjunto de índices de prestaciones a maximizar no era uniforme, o incluso su propia definición; iii) no había ningún estudio comparativo de algoritmos de predicción que tuviera en cuenta los costes y los beneficios de la prebúsqueda de forma conjunta; y iv) la evaluación de las técnicas propuestas se realizaba bajo cargas de trabajo muy distintas o poco significativas.

Durante la investigación, hemos contribuido a homogeneizar la evaluación de prestaciones por medio del desarrollo de un entorno de simulación que reproduce detalladamente todos los aspectos que afectan a las prestaciones de la prebúsqueda. Además, hemos analizado las métricas de prestaciones utilizadas en los estudios de prebúsqueda para clarificar su definición y detectar cuáles eran las más significativas desde el punto de vista del usuario. También hemos propuesto una metodología de evaluación que considere el coste y el beneficio de la prebúsqueda de forma conjunta. Por último, se ha resaltado la importancia de utilizar cargas de trabajo actuales para evaluar las prestaciones de la prebúsqueda, ya que, en caso contrario, podríamos obtener conclusiones incorrectas.

El análisis de los beneficios potenciales de cada posible arquitectura del sistema nos mostró que, potencialmente, los predictores colaborativos podrían reducir casi toda la latencia percibida por los usuarios. El primer paso para desarrollar un predictor colaborativo es hacer las predicciones desde el servidor, por lo que en esta tesis nos centramos en una arquitectura de prebúsqueda en la que el predictor de accesos está situado en el servidor web.

Por otra parte, las condiciones del entorno que nos podemos encontrar en la web son también heterogéneas, por lo que la forma óptima de utilizar la prebúsqueda podría ser también distinta dependiendo de este entorno. Por ello, hemos estudiado

bajo que condiciones es mejor maximizar bien el número de objetos prebuscados, bien el tamaño de dichos objetos, para obtener las máximas prestaciones. Los resultados muestran que, cuando el ancho de banda disponible por el usuario aumenta, las técnicas de prebúsqueda deberían ser más propensas a prebuscar una gran cantidad de objetos mientras que cuando se reduce el tiempo de proceso del servidor, es la cantidad de bytes prebuscados la variable que se debe maximizar.

Al analizar de las técnicas de prebúsqueda existentes encontramos que, debido a la evolución que ha sufrido la Web, algunas de las técnicas que funcionaron bien hace unos años ya no obtienen las mismas prestaciones en la Web actual. Esto se debe principalmente a la gran cantidad de imágenes u objetos similares que se incluyen actualmente en cada página web. Estas imágenes interfieren notablemente en la predicción de accesos del usuario. Considerando este efecto no deseado, hemos propuesto y evaluado un nuevo algorithmo (DDG) que maneja estas características presentes en la web actual. Para crear el modelo de predicción, el algorithmo DDG diferencia los objetos entre contenedores (HTMLs) e incrustados (p.e. imágenes). Los resultados experimentales muestran que, dado un incremento de peticiones prefijado, el algoritmo DDG siempre reduce más la latencia que los algoritmos existentes. Además, estos resultados se han conseguido sin aumentar el orden de complejidad respecto a los algorithmos existentes.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The enormous spread of Internet and, consequently, the growth of the World Wide Web due to the popularization of new applications and services, like e-business, multimedia contents, e-learning, etc. have implied a dramatic increase in the number of connected users, and, therefore, a tremendous increase in the global traffic, damaging the quality of service (availability, reliability, security) and specially the latency perceived by the users.

Internet latency depends on the number of intermediate points and on the amount of traffic crossing them. The connection between two adjacent points is generally a cable line, but data must cross other elements such as modems, Ethernet cards, switches, routers, etc. All these elements are obstacles that increase the latency. Each point has a separate hardware, different cable bandwidths, and it even uses different protocols. Therefore, the number of intermediate steps must be minimized and individual bandwidth must also be enhanced in order to improve web performance. The international and global nature of Internet (Internet is a network of networks) makes it very difficult (or sometimes really impossible) to improve system performance working in the network hardware and in their interconnection elements. As a consequence, it is necessary to work on web architecture and organization using and exporting techniques already learned and widely used in computer architecture to improve web performance.

Those techniques take advantage of the locality properties shown by accesses to web objects. In the Web, locality of reference has three properties [Bestavros 97]: temporal, geographical and spatial.

Temporal locality means that objects referenced recently are likely to be accessed again in a near future. Web caching takes benefit of this property [Aggarwal 99, Jin 00].

Geographical locality implies that objects referenced by a user are likely to be accessed by users in the same geographic area. Replication techniques [Johnson 01,

Iyengar 02], including CDNs, profit from the geographical locality to reduce the global traffic as well as the user-perceived latency.

Spatial locality is related to the fact that, after accessing an object, another object located close to the first one will be probably accessed. *Located near* in this context refers to objects in the same directory, in the same web server or in the same group of servers. Web prefetching takes benefit of this property in order to predict future accesses [Markatos 98, Fan 99]. In this thesis we focus on how this technique can be used to reduce the user-perceived latency.

The basics of web prefetching is to preprocess a user's request before it is actually demanded. Therefore, the time that the user must wait for the requested document can be reduced by hiding the request latency. Prefetching is usually transparent to the user, that is, there is no interaction between the prefetching system and the user. For this reason, systems speculate on the following user's requests and thus the prediction can fail. In such a case, web prefetching increases the resources requirements, so it should be applied carefully.

1.1 Motivation

Despite being a technique researched for years, the commercial penetration of web prefetching is still poor. By studying the open literature, one can observe that the development, methodology and goal of the proposals are highly heterogeneous. We found this heterogeneity in several areas:

- There was no open framework to develop, test, and evaluate web prefetching techniques.
- The performance metrics and ever their definition and use differ in the studies.
- The workload used to check the proposals is old, corresponding to the first web generation, and does not represent current web trends and user's behavior.
- Most of the proposal evaluations do not consider the user's point of view, focusing only on the prediction performance.
- There was no standard methodology to compare prefetching algorithms that take into account the cost and benefit of web prefetching at the same time.

In addition, most of the proposed prediction algorithms are adaptations of algorithms used in other fields of computing and do not profit from the organization of web objects. These facts encouraged us to cover the methodological gaps found in the literature and develop a new prefetching algorithm able to deal with the main drawbacks of the existing ones.

1.2 Objectives of the thesis

Firstly, this thesis is aimed at providing the present state of the art with a solid framework and methodology to evaluate web prefetching techniques from the user's point of view. To do so, an open framework which permits the implementation of prefetching techniques in a flexible and easy way has been developed. Moreover, performance key metrics have been analyzed to detect the most meaningful indexes from the user's point of view. A comparison methodology that takes into account the costs and benefits of prefetching has also been proposed.

The second main goal of the thesis is, once the methodological issues are clarified, to find out how web prefetching algorithms can be improved from the user's point of view and propose a new one that outperforms those existing in the open literature. To do so, the most important algorithms in the literature are analyzed from the user's perspective. In the detailed analysis we found that their performance in current web can be improved by distinguishing two classes of objects: container objects and contained objects. We take benefit of this observation in order to design the new algorithm.

As secondary objectives, we explore the performance limits of web prefetching to know the potential benefits of this technique and analyze how to adapt prefetching algorithms to the environment conditions in order to achieve the best results in each situation.

1.3 Organization of the thesis

This thesis is organized as follows:

Chapter 2 introduces web prefetching and presents the current state of the art in this technique. A large set of prediction algorithms are surveyed as well as the commercial products implementing web prefetching.

Chapter 3 analyzes why web prefetching techniques are rarely compared. When they are compared, the evaluation is not done from the user's point of view. We found four main reasons: i) the proposed approaches are applied and tested in different baseline systems, ii) they use workloads that are not representative of current web, iii) different performance metrics are measured to quantify the benefits, and iv) studies usually do not focus on the user's perspective neither use a cost-benefit analysis. To overcome these limitations, this chapter presents: i) a general experimental framework which permits the implementation of prefetching techniques in a flexible and easy way, ii) an analysis of the main performance key metrics to find the most meaningful from the user's point of view, iii) a cost-benefit methodology to perform fair comparisons of prefetching algorithms, and iv) an analysis of how dependent results are on the workloads used to show the importance of using current workloads to evaluate current web.

Chapter 4 explores the potential limits of reducing user-perceived latencies depending on where the predictions are made, and on if they are made by a single element of the architecture or in a collaborative way.

Chapter 5 evaluates the most representative algorithms from the user's point of view. A detailed analysis is provided to find why performance differs among the algorithms.

Chapter 6 performs a rigorous statistical analysis to identify those environment conditions in which it is better to prefetch smaller objects instead of larger ones. This permits an algorithm to adapt the prefetching depending on the server and client environment.

Chapter 7 introduces a new prefetching algorithm that considers the characteristics of current web sites to improve the performance achieved by web prefetching.

Finally, Chapter 8 summarizes the main contributions of this thesis and presents some concluding remarks.

Chapter 2

Web Prefetching

2.1 Introduction

The goal of web prefetching is to preprocess user's requests before the user demands them explicitly, so reducing the user-perceived latency. The main prefetching mechanisms proposed in the literature are transparent to the user and, consequently, they are necessarily speculative. As any speculative technique, prefetch predictions can fail, i.e. downloading an object that will not be demanded later. That means that prefetching can involve many hazards because if the prediction is not accurate, it can pollute the cache, waste bandwidth and overload the original server. This implies that prefetching must be carefully applied, by using, for example, idle times in order to avoid performance degradation [Crovella 98]. Despite these hazards, it has been shown that many prefetching algorithms [Bestavros 95, Padmanabhan 96, Markatos 98, Fan 99, Chen 02] can considerably reduce the latency perceived by users.

The prefetch predictions can be performed by the server, by the proxy, or by the client itself. Some research studies suggest to perform the predictions at the server [Padmanabhan 96, Markatos 98] because it is visited by a high number of users and it has enough information about how they visit the site, therefore its predictions can be quite accurate. Some other studies suggest that proxy servers can perform more accurate predictions because their users are much more homogeneous than in an original server, and they can also predict cross-server links which can reach about 29% of the requests [Duchamp 99]. Finally, other authors consider that predictions must be performed by the client browser, because it knows users' preferences better [Kim 03, Lau 04]. There are also some studies indicating that the different parts of the web architecture (users, proxies and servers) must collaborate when performing predictions [Markatos 98].

This chapter presents the current state of the art in web prefetching. It is organized as follows: Section 2.2 presents a generic web prefetching architecture and describes the elements and functions that compose it. Section 2.3 surveys and classifies a large

set of web prediction algorithms appeared in the literature. The most used algorithms are examined in detail, since they will be used and analyzed in the following chapters. Section 2.4 reviews the commercial products that implement any web prefetching technique. Finally, Section 2.5 presents some concluding remarks about this study.

2.2 Generic Web Prefetching Architecture

Prefetching systems are usually based on a generic web architecture. This technique is implemented by means of two extra elements, the prediction and prefetching engines, that can be located in the same or different elements of the system. In this section, we describe the elements that compose a generic web architecture and how it can be extended and used to carry out prefetching.

2.2.1 Generic Web Architecture

There are two main elements in a generic web architecture: i) user agents, or clients, that is, the software employed by users to access the Web, and ii) web servers, which contain the information that users demand. The generic web architecture is an example of the client-server paradigm, which works as follows. Human users tell the client which page they want to retrieve by writing down a URI or by clicking a hyperlink on a previously loaded page. Then, the client demands each object the page consists of. Finally, the whole page is displayed to the user. Optionally, there may be more elements between clients and servers, as Figure 2.1 shows. A proxy is usually located near a group of clients to cache the most popular objects accessed by that group. By doing so, the user-perceived latency and the network traffic between the proxy and the servers can be reduced. Surrogates, also called reverse proxies, are proxies located by the server side. They cache the most popular server responses and are usually transparent to the clients, which access to the surrogate as if they accessed to the web servers.

2.2.2 Prediction Engine

The prediction engine is the part of the prefetching system aimed at guessing the following user's accesses. This engine can be located at any part of the web architecture: clients [Kim 03, Lau 04], proxies [Fan 99, Bouras 04], and servers [Schechter 98, Palpanas 99, Lee 06, Domènech 06b], or even in a collaborative way between several elements [Markatos 98]. To make the predictions, a high amount of algorithms that learn from the past access patterns to predict future ones have appeared in the literature, e.g., [Sarukkai 00, Teng 05, Domènech 06b]. These patterns of user's accesses differ depending on the element of the architecture in which the prediction engine is implemented because the information that each one can gather is completely different. For instance, a predictor located at the web server can only gather transitions between pages of the same server, but not cross-server transitions. Notice that it is

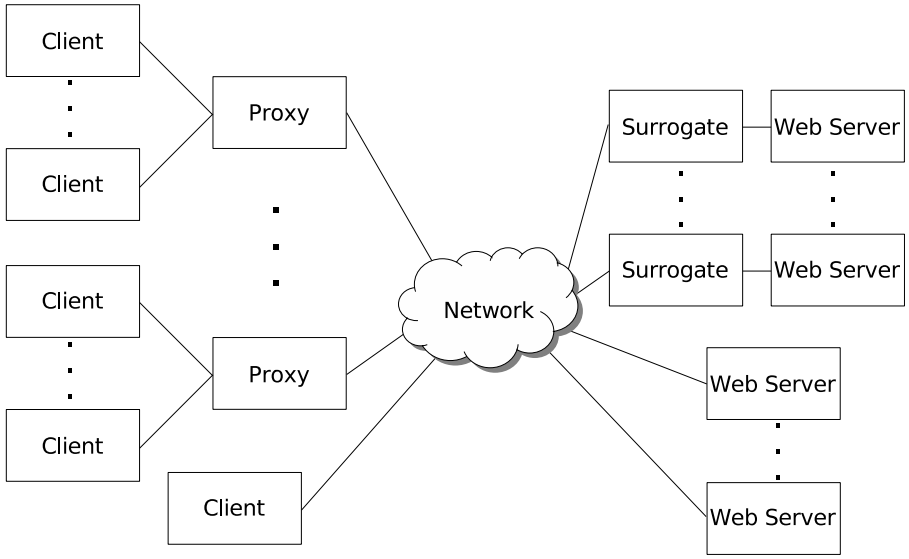


Figure 2.1: Generic Web Architecture

equivalent to predict from the surrogate or from a server without surrogate, since both gather the same requests.

The output of the prediction engine is the hint list, which is composed of a set of URIs which are likely to be requested by the user in a near future. Nevertheless, this list can also consist of a set of servers if only the connection is going to be prefetched. Due to the fact that in many proposals the hint list is included in the HTTP headers of the server response, the time taken by the predictor to provide this list should be short enough to avoid delaying every user request and, therefore, degrade overall performance.

The predictor must distinguish between those objects that can be prefetched (and therefore predicted) and those that cannot. For this reason, the hint list can only contain prefetchable URIs. According to the definition proposed in [Davison 01a], a web resource is prefetchable if and only if it is cacheable and its retrieval is safe. Browsers based on Mozilla [Fisher 03] consider that a resource is not prefetchable if the URI contains a query string.

2.2.3 Prefetching Engine

The prefetching engine is aimed at preprocessing those object requests predicted by the prediction engine. By processing the requests in advance, the user's waiting time when the object is actually demanded is reduced. In the literature, this preprocessing has been mainly concentrated on the transference of requested objects

in advance [Padmanabhan 96, Duchamp 99, Fan 99, Ibrahim 00, Kokku 03, Wu 06] although other approaches consider the preprocessing of a request by the server [Schechter 98, Lee 06], or focus on the pre-establishment of connections to the server [Cohen 02]. For this reason, the prefetching engine can be located at the client, at the proxy, at the server, or at several elements.

In addition, to avoid the interference of prefetching with the current user's requests, the prefetching engine can take into account external factors to decide whether to prefetch an object hinted by the prediction engine or when to prefetch it. These external factors could be related to any part of the web architecture. For instance, when the prefetching engine is located at the client, some commercial products wait for the user to be idle to start the prefetching [Fisher 03, goo b]. If it is located at the proxy, the prefetching engine could prefetch objects only when the bandwidth availability is higher than a given threshold. The case of the predictor located at the server, it could prefetch only when its current load permit an increase of requests.

The capability of reducing the user-perceived latency by means of web prefetching depends on where this engine is located. In this way, the closer to the client the prefetching engine is, the more latency is avoided. Therefore, a prefetching engine located at the client can reduce the whole user-perceived latency. Besides, it is the current trend as it is included in commercial products like Mozilla Firefox and Google Web Accelerator (see Section 2.4).

2.3 Web Prediction Algorithms

The previous section has shown that a quick prediction of user's accesses is the key of web prefetching. This section surveys the large variety of prediction algorithms that can be found in the literature. We have classified them into two main categories according to the data taken into account to make the predictions.

The first category includes those algorithms that predict future accesses considering the observed pattern of past accesses. The algorithms included in the second category make the prediction by analyzing the content of the recently visited web pages to find links with a high probability of being clicked.

2.3.1 Prediction from the accesses pattern

Algorithms in this category lie in the statistical inference about future user's accesses from the past ones. The prediction of users' accesses has usually been made by adapting prediction algorithms from other fields of computing to deal with web accesses. For instance, the DG algorithm described in Section 2.3.1.1, lies in a predictor of accesses to a local file system [Griffioen 94]. PPM algorithm (see Section 2.3.1.1) is mainly used for lossless data compression [Bell 90, Curewitz 93]; besides, it has also been used by several authors [Fan 99, Palpanas 99, Chen 03, Bouras 04] to predict web accesses. According to the amount of research works found, this is the most important category.

The algorithms have been classified in three subcategories depending on the statistical technique used to compute the predictions: i) those based on Markov models, ii) those using data mining techniques and iii) other techniques. The algorithms presented in the first subcategory are the ones that require less CPU time to make predictions. Consequently, the algorithms based on Markov models are the main candidates to be implemented in a web prefetching system to achieve the best reduction of user-perceived latency.

2.3.1.1 Markov models

A Markov model is a finite-state machine where the next state depends only on the current state. Associated with each arc of the finite-state machine network (a directed cyclic graph) is the probability of making the given transition [Mitchell 93]. When applied to the prediction of user accesses, each state represents the context of the user, i.e., their recent past accesses, and the transitions represent a user's demand. In this sense, the probability attached to each transition is the probability of accessing a given object.

The main parameter of any Markov model is the *order*, which refers to the number of accesses that defines each different context. Usually, the higher order the more precision. However, an increase in the order of the model also implies an increase in the resources needed to compute the prediction.

There exists a high amount of works that describe web prediction algorithms based on Markov models. Basic Markov models considering the accesses sequence of different orders are explored in the literature.

A Markov model of first order is implemented and evaluated by [Nicholson 98]. The top-10 algorithm proposed by [Markatos 98] can be considered as a Markov model of order 0, that is, the prediction result does not depend on the context (i.e., lasts requests). In [Sarukkai 00], the author considers first and second order Markov chains to predict requests. The optimal results in [Fan 99] with PPM, an algorithm based on Markov models described below, are obtained when a second order model is applied. However, [Su 00] obtains better results when implementing the model with orders higher than 3. Most of these studies only focus on the precision of the prediction to determine the appropriate order of the model without taking into account how it affects the user-perceived latency. This fact is analyzed in Chapter 5.

When Markov models of higher orders are explored, the use of techniques to reduce the resource requirements of the implementation is needed. Therefore, some papers explore how to deal with such a large amount of data. For instance, [Zhu 02b, Zhu 02a] discuss a method for compressing the representation of Markov models of high order to reduce the size of the model. With the same goal, a pruning method to reduce the size of the prediction tree without decreasing the performance is presented in [Li 01]. A variation of a Markov model to store only long branches with frequently accessed URLs is proposed by [Pitkow 99]. In [Chen 02, Chen 03], authors propose a PPM variant that classifies the branches of the tree root in four classes according to the

popularity of the branch, so that branches with higher popularity are allowed to be longer than the less popular ones.

Other authors take into account, in addition to the access sequence of the objects, the referrer information provided in the HTTP protocol. In this context, four models of first and second order considering these data are checked in [Zukerman 99]. They found that the best algorithm to predict user's accesses is the Markov model which takes into account referee and access sequence information at the same time. This algorithm is favourably compared in [Albrecht 99] against the temporal dependency graph algorithm of [Bestavros 96], described below.

Algorithms based on Markov models have been proposed to be applied either to each object access [Zukerman 99, Sarukkai 00, Davison 04] or to each page (i.e., to each container object) accessed by the user [Palpanas 99, Dongshan 02, Chen 03]. In addition, two ways have been used to select which object or page will be predicted: predicting the top-n likely objects [Sarukkai 00, Dongshan 02, Davison 04] and predicting those objects with more probability of being accessed than a given confidence threshold [Zukerman 99, Palpanas 99, Nanopoulos 03, Chen 03]. The four variants resulting from combining the unit of prediction (object or page) and the way of selecting the candidates (top-n or using a threshold) are analyzed in Chapter 5.

Prediction by Partial Matching (PPM) A commonly used algorithm used for predicting users' accesses is the PPM (prediction by partial matching) [Fan 99, Palpanas 99, Chen 02, Chen 03]. A similar algorithm is presented in [Dongshan 02], although PPM is not mentioned by the authors. PPM algorithm uses Markov models of m orders to store previous contexts. Predictions are obtained from the comparison of the current context with each Markov model. The algorithms for building the prediction model and for making the predictions are shown in Figures 2.2 and 2.3, respectively.

For illustrative purposes, we use an hypothetical example. Let's suppose that the algorithms are trained by two user sessions. The first one contains the following sequence of accesses: HTML1, IMG1, HTML2, IMG2. The second session includes the sequence: HTML1, IMG1, HTML3, IMG2. Note that IMG2 is embedded both in HTML2 and in HTML3. We found this behavior common through the analyzed workloads. For instance, in a news web server, different pieces of news (i.e., HTML files) contain the same embedded images, since they are included in the site structure. Figure 2.4 shows the graph obtained when applying the PPM algorithm to the described training. Each node represents a context, where the root node is in the first row, the order-0 context is in the second, and the order-1 context is in the third. The label of each node also includes the counter of times a context has appeared, so one can obtain the confidence of a transition by dividing the counter of a node by the counter of its parent, i.e., the node in the previous context. The arcs indicate the possible transitions. For instance, the label of the IMG2 in order-0 context is 2 because IMG2 appeared twice in the training; once after HTML2 and another after

```
1: Objective: Build a PPM prediction model
2: Input:
3:   S: set of users' sessions
4:   m: order of the Markov model
5: Output:
6:   T: prediction model
7: for each session s ∈ S do
8:    current-context[0] ← root node of T
9:    for j = 1 to m do
10:      current_context[j] ← NULL
11:   end for
12:   for each object i ∈ s do
13:      for j = m down to 0 do
14:         if current_context[j] has child_node C representing i then
15:            increment node C occurrence_count
16:            current_context[j + 1] ← node C
17:         else
18:            construct child_node C representing i
19:            node C occurrence_count ← 1
20:            current_context[j + 1] ← node C
21:         end if
22:         current_context[0] ← root node of T
23:      end for
24:   end for
25: end for
26: return T
```

Figure 2.2: Algorithm for making the prediction model in PPM

1: **Objective:** Obtain predictions of the next requests of the user
2: **Input:**
3: T: prediction model
4: R_i: last k user's accesses; $0 \leq i \leq k \leq$ order of the prediction model
5: th: threshold
6: **Output:**
7: P: Set of predictions
8: **for** $j = 1$ to k **do**
9: current_context[j] $\leftarrow$ node of depth j, representing the access sequence $R_{k-j+1}, \cdots, R_k$
10: **end for**
11: **for** $j = k$ down to 1 **do**
12: **for** each child_node C of current_context[j] **do**
13: **if** (occurrence_count of C) / (occurrence_count of parent) >= th **then**
14: $P \leftarrow P \bigcup \{a\}$
15: **end if**
16: **end for**
17: **end for**
18: remove duplicates from P
19: **return** P

Figure 2.3: Algorithm for making predictions in PPM

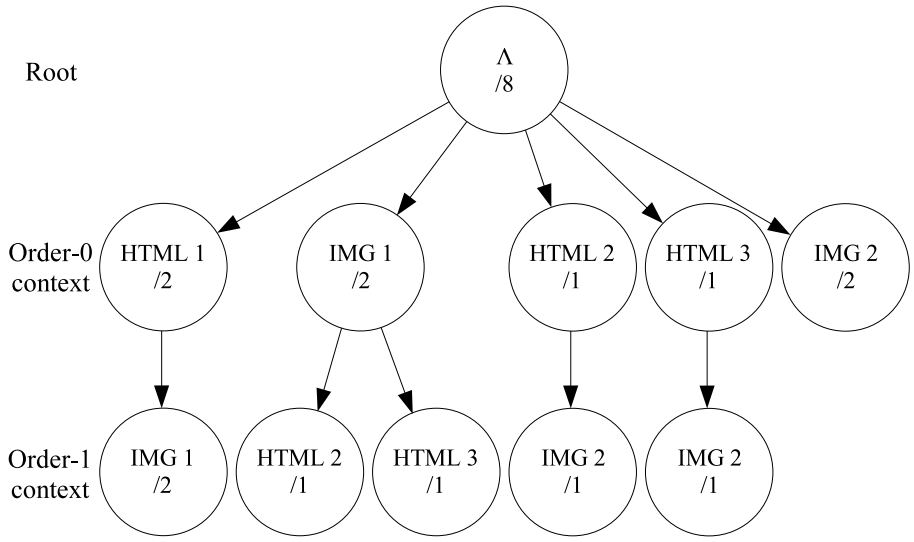


Figure 2.4: State of the graph of a first-order PPM algorithm after the accesses HTML1, IMG1, HTML2, IMG2 by one user; and HTML1, IMG1, HTML3, IMG2 by other user

HTML3; IMG2 has two nodes in the order-1 context, i.e., one per each HTML on which it depends.

Dependency graph (DG) The DG prediction algorithm constructs a dependency graph that depicts the pattern of accesses to the objects. The graph has a node for every object that has ever been accessed. As defined by [Griffioen 94] in file systems prediction, and implemented by [Padmanabhan 96, Jiang 98, Nanopoulos 03] in web prefetching, there is an arc from node A to B if and only if at some point in time a client accessed to B within w accesses after A, where w is the *lookahead window* size. The weight of the arc is the ratio of the number of accesses to B within a window after A to the number of accesses to A itself. Figure 2.5 shows the pseudo-code for constructing the DG prediction model. A variant of this algorithm is implemented in [Bestavros 95, Bestavros 96, Albrecht 99], where the *lookahead window* is based on the time elapsed between two accesses instead of on the amount of accesses between them. In both cases, the prefetching aggressiveness is controlled by a cutoff threshold parameter applied to the weight of the arcs. The algorithm for making the predictions is illustrated in Figure 2.6

To illustrate the algorithm, let's suppose that the algorithms are trained by the two user sessions used to show the PPM algorithm. The first one contains the following accesses: HTML1, IMG1, HTML2, IMG2. The second session includes the accesses:

HTML1, IMG1, HTML3, IMG2. Figure 2.7 shows the state of the graph of the DG algorithm after the aforementioned training for a lookahead window of 2 accesses. Each node in the graph represents an object, whereas the weight of each arc is the confidence level of the transition.

```
1: Objective: Build a DG prediction model
2: Input:
3:   S: set of users' sessions
4: Output:
5:   T: prediction model
6: for each session s ∈ S do
7:    create empty lookahead window l
8:    for each object i ∈ s do
9:       for each object o ∈ l from the newest to the oldest do
10:         increment weight of arc from o to i in T
11:      end for
12:      increment i access counter in T
13:      if l is full then
14:         remove the oldest object from l
15:      end if
16:      insert i in l
17:   end for
18: end for
19: for each object t ∈ T do
20:   for each output arc a ∈ t do
21:      a confidence ← weight of arc from t to a / t access counter
22:   end for
23: end for
24: return T
```

Figure 2.5: Algorithm for making the prediction model in DG

2.3.1.2 Web mining algorithms

Several prediction algorithms benefiting from web mining techniques have been developed. Web mining is defined in [Kargupta 04] as the application of data mining techniques to extract knowledge from Web data, where at least one of structure (hyperlink) or usage (Web log) data is used in the mining process (with or without other types of Web data). It has been found that two data mining techniques are used to make web accesses predictions: clustering and association rules mining.

Clustering algorithms are applied in [Yang 03] to a three dimensions cube representation of web accesses. To make the predictions, the transition probability matrix for pages that belong to the same clusters is calculated. The prediction model used by

1: **Objective:** Obtain predictions of the next user's requests
2: **Input:**
3: T: prediction model
4: u: last user's access
5: th: threshold
6: **Output:**
7: P: Set of predictions
8: **for** each output arc $a \in u$ in T **do**
9: **if** a confidence $>$ th **then**
10: $P \leftarrow P \bigcup \{a\}$
11: **end if**
12: **end for**
13: **return** P

Figure 2.6: Algorithm for making predictions in DG

[Gündüz 03] takes into account data both from the accesses sequence of each session and from the user's thinking time of each page in the session. Then, user's sessions are clusterized and the prediction is made by considering the most similar session in the cluster.

A rule-assisted prefetching strategy is developed to make predictions in [Lan 00, Pandey 01, Mobasher 02, Pandey 02, Huang 05]. The technique presented in [Lan 00] gives priority to recently added links over those older, whereas [Huang 05] tries first to find a rule generated from the user that is currently accessing the web. The proposal of [Nanopoulos 01, Nanopoulos 02, Nanopoulos 03] deals with random accesses that could be in the pattern (i.e., access patterns), although the other user's accesses must be in the same order as found in the pattern. Their algorithm (WM_o) also takes into account the graph of the web site structure to allow only those patterns with transitions included in the graph.

2.3.1.3 Other algorithms

Other approaches to web access prediction can be found in the literature. The prediction engine presented in [Bonino 03b, Bonino 03a] is based on a evolutionary finite-state machine (FSM). In this FSM, user requests are represented by transitions, but unlike the FSM representation of Markov models, each node in their model represents a prediction instead of a context. The algorithm is initially set up with a set of FSM that will evolve according to genetic operators: recombination, mutation and natural selection. Authors propose a fitness function to evaluate the goodness of each FSM related to the main prediction metrics. This function helps the evolution of the FSMs to achieve better performance.

The algorithm presented by [Wu 06] takes into account object characteristics not considered in other algorithms to maximize the ratio of hit rate to bandwidth ratio.

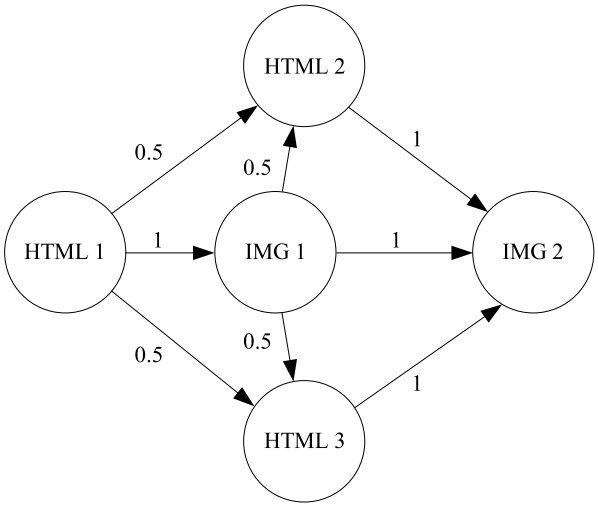


Figure 2.7: State of the graph of the DG algorithm with a lookahead window size of 2 after the accesses HTML1, IMG1, HTML2, IMG2 by one user; and HTML1, IMG1, HTML3, IMG2 by another user

This technique lies on accounting the contribution to the hit rate and to the bandwidth ratio of each object, but needs extra information about the objects like access frequency, size, lifetime, etc.

In [Angermann 02], the author explores how the prefetching should be conducted to minimize the user-perceived latency, despite not considering the costs of the prefetching. The proposed prefetching system sorts the objects with respect to their (unconditional) probability of being requested, and then fetches sequentially all the documents.

2.3.2 Prediction from web content

In [Khan 03], authors suggest that the prediction of user's accessed can be improved if the algorithm considers the organization of the webspace. Therefore, the second subset of web prediction algorithms are those whose predictions are made from the analysis of the content of the web pages (usually HTMLs) that the user has requested instead of using only the sequence of accesses. As a consequence, the complexity of these algorithms is significantly higher than the complexity of just analyzing the references.

The first attempt to predict next user's accesses from the HTML content was presented in [Duchamp 99]. This work combines the analysis of the content with usage profiles that are generated by the web servers.

In a more recent work, [Ibrahim 00] apply neural networks to keywords extracted from the HTML content in order to make the predictions. Their evaluation, which is performed from the prediction point of view, only focuses on news web servers where key word extraction is easier.

The prefetching technique presented by [Davison 02] sorts the links found in a HTML according to their probability of being clicked. This order is given by the similarity of each text context of the links to the text context of the previously clicked link. The context is composed of the words that appear near the anchor tag in the HTML.

2.4 Web Prefetching in Commercial Products

Despite being a subject researched for years, the commercial penetration of web prefetching is still poor, mainly caused by the reduced bandwidth availability and the modification of the HTTP standard proposed by some authors. The main commercial contribution to promote web prefetching as a mean for reducing user-perceived latency has been made by the Mozilla Firefox browser and Google.

As described in [Fisher 03], the *Firefox* web client contains a web prefetching engine that parses *link* HTTP headers to find prefetch hints included by the server or proxy. When the user is idle after downloading the page, Firefox prefetches sequentially all the hinted URLs found in the header. This technique requires the server to make the predictions, but this is not common. *Google Search* includes support for the prefetching technique implemented by Firefox since a prediction hint pointing to the URL of the first result is included in some situations [goo a].

Google Web Accelerator [goo b] is a software that benefits from web prefetching (and other techniques) to improve the user-perceived latency. It works together with Mozilla Firefox or Microsoft Internet Explorer and implements the prefetching engine described by [Fisher 03], as well as prefetchs other objects predicted by itself.

PeakJet 2000 and *NetAccelerator* implement two aggressive prefetching modes: historic and link based. In the former, the software refreshes all the objects in the cache accessed by the user. In the latter, all the linked documents in the current page are downloaded. The difference between Peakjet and NetAccelerator lies in where prefetched documents are saved, since Peakjet has its own cache and NetAccelerator uses the browser cache.

Other commercial software implementing web prefetching techniques are *Robtex Viking Server*, *Mentat SkyX Accelerator* and *AllegroSurf*, but they provide few details on how prefetching works in their products.

2.5 Conclusions

This chapter has shown the basics of any web prefetching system and have presented the current state of the art in the technique, both in research and in commercial areas.

We have described the elements that a web prefetching system adds to a generic web architecture. The elements are the prediction engine, which guesses future user accesses, and the prefetching engine, which takes the output of the predictor to preprocess the requests. We have highlighted that the predictor must be fast enough to avoid damaging the overall performance.

A large variety of prediction algorithms have appeared in the research literature, although most studies do not focus on the user's point of view. We found two main groups: those that analyze the accesses sequence of the users and those that analyze the content seen by the user. From those algorithms, the main candidates to achieve the best performance are those that analyze the sequences of accesses by means of Markov models, since their CPU requirements to make predictions are lower. Despite the important research effort, we have shown that the use of web prefetching in commercial products is still marginal.

Chapter 3

Performance Evaluation

3.1 Introduction

There is a large amount of research works focusing on web prefetching in the literature, but comparative studies are rare. Different proposals cannot be easily compared because of four main reasons: i) the proposed approaches are applied and tested in different baseline systems, ii) they use workloads that do not represent the current web, iii) different performance metrics are measured to quantify the benefits, and iv) studies do not usually focus on the user's perspective or use a cost-benefit analysis. This fact leads to the inability to quantify in a real working environment which proposal is better for the user.

In this chapter we show how these limitations have been overcome to provide the appropriate framework and methodology to evaluate and compare web prefetching algorithms from the user's point of view. To do so, Section 3.2 presents a general experimental framework which permits the implementation of prefetching techniques in a flexible and easy way using real workloads.

Section 3.3 tackles the selection of the performance key metrics. To this end, we analyze a large subset of key metrics and propose a taxonomy based on three main categories, which permits us to identify analogies and differences among the metrics commonly used, and to check experimentally their relation. As a consequence, we also introduce new byte based indexes as complementary metrics.

We propose in Section 3.4 a cost-benefit methodology to perform fair comparisons of web prefetching algorithms from the user's point of view. This methodology takes into account the main benefit of web prefetching, i.e., user-perceived latency reduction, versus its main costs: increase in the network use and increase of server load.

Finally, Section 3.5 deals with the selection of the workload that will be used to evaluate web prefetching techniques. In this section, current and old workloads are compared, since a high amount of current research works use old traces to evaluate prefetching algorithms. Results show that there is no reason to think that prefetching

studies should be conducted using old traces, since the performance of prefetching algorithms is highly different. Hence, conclusions obtained for old workload cannot be directly moved to the current web.

3.2 Experimental Framework

3.2.1 Introduction

Most of the prefetching research studies use different frameworks to check their proposals, therefore the assumptions, conditions, implementations and even performance metrics widely differ among them. This fact implies that performance comparison studies cannot be fairly done. Therefore, it is difficult to distinguish the aspects and conditions in which each technique presents better performance, or when its use is more convenient. Some of these studies presented in the open literature analyze the theoretical bounds of performance that prefetching can reach [Kroeger 97, Fan 99, Domènech 06h]. The different scenarios only permit to arrive at very general conclusions like, for instance, that performance can boost until 60% [Kroeger 97] or more [Domènech 06h].

Although the open literature presents very interesting proposals about prefetching techniques, it has not been possible to reproduce them yet (or to propose new ones) using similar environmental conditions in order to compare their main features, advantages and disadvantages. Despite the fact that there are some simulator environments to check cache replacement algorithms [Cárdenas 04, Cao] as well as network features [Davison 01b, UCB], there is a lack of simulation environments which concentrate on prefetching. This fact, together with the potential performance benefits that prefetching can reach, motivates us to propose a simulation environment to check the performance of different prefetching algorithms under similar conditions. Our environment permits the implementation of prefetching algorithms on any part of the web architecture (user browser, proxy or server), either individually or working together.

The simulation environment (Figure 3.1) is composed of three main parts: the server, the client, and the proxy. We have developed a hybrid implementation which combines both real and simulated parts in order to provide the environment flexibility and accuracy.

The back end part has both the real web server and the surrogate server. The server is external to our environment and the system can access to it through the surrogate where the prefetching engine is implemented. The surrogate is implemented on a real machine offering fully representativeness.

The front end contains the client component, which represents the user behavior in a prefetch enabled client browser. It is possible to use real traces or a synthetic workload generator for each set of users accessing concurrently to the same server.

The intermediate part, which is located between the clients and the surrogate system, models one or more proxy servers. Each one can implement the prefetching independently.

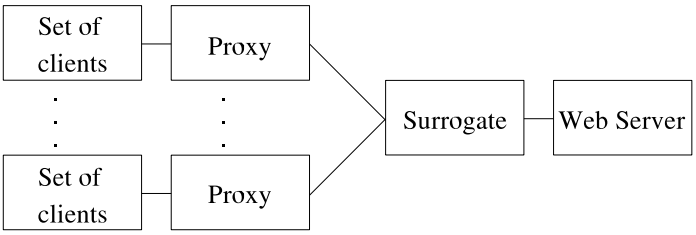


Figure 3.1: Architecture of the simulation environment

This organization provides a useful and operative environment since it permits the implementation of prefetching algorithms working in any part of the system (in the browser of the client, in the original server or in the proxy, or even in a collaborative manner) by modifying only few parts of the structure. In addition, the environment provides the appropriate interfaces to communicate the different modules between them in order to implement easily those prefetching algorithms in which the different elements collaborate. On the other hand, our simulator also includes a module to provide performance results, and the main metrics discussed in Section 3.3.

3.2.2 Surrogate

A surrogate proxy is an optional element located close to the origin server that generally acts as a cache of the most popular server responses ([Vahdat 98, Group 01] or accelerator mode in Squid [Pearson]). Usually surrogates are not used to perform prefetching predictions. Although it is equivalent to perform predictions from the server or the surrogate, the latter one allows much easier and more flexible software implementations. Therefore, we perform our implementation on the surrogate.

The surrogate hints clients about what objects should be prefetched by means of an optional header offered by the HTTP/1.1 standard [Fielding 99] in the same way as Mozilla (from version 1.2 on) admits [Mozilla , Fisher 03]. This implementation is based on the one described in [Padmanabhan 96]. The list of hints that the surrogate proxy piggybacks is generated for each response.

Figure 3.2 shows the block diagram of the proposed surrogate proxy. Each block represents a functional unit. Some of them are implemented as functions and others as threads, keeping separately all their functionality.

The listener block, which is the simplest one, is implemented as a thread that waits for clients to connect its socket, acting as a concurrent TCP server that listens at a proxy port (generally at port 80). By default, there are no restrictions on the number of simultaneous incoming connections. The main goal of this block is to create a new *Connection to server* thread.

The *Connection to server* block, which is the main one, handles the connections. The connection involves a request that is transferred to the web server, which processes

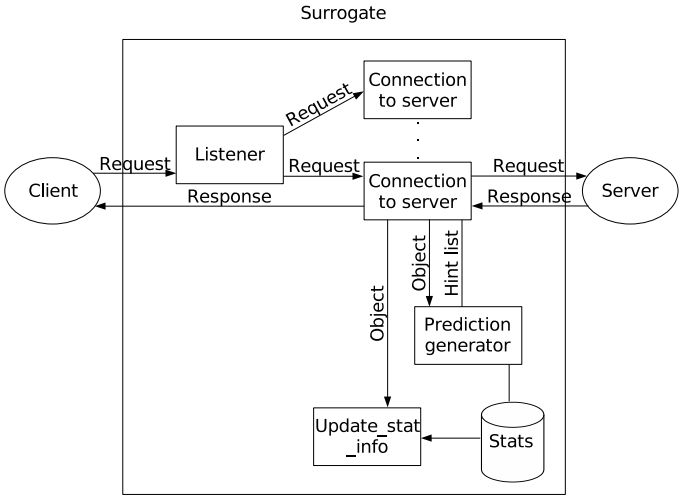


Figure 3.2: Block diagram of the surrogate

it and starts the transference. In that moment, the module first parses the response until the full header is received and then, asks for a prediction to the *Prediction generator* block. The prediction consists of a list of hints whose content depends on the implemented algorithm. This list will be piggybacked on the response headers, so the client can prefetch the object if the client application considers it appropriate.

When the whole object is received, the listener calls the *Update stat Info* block to notify that event as well as its headers (object size, content type, prefetch or user request, etc). This information is used to update the contents of the statistical database, which is used to generate predictions. The frequency at which statistics are updated depends on the prefetching scheme; e.g., once per hour, once per day, when the server is not too loaded, when the server finishes its response etc. As our aim is to maintain the simulator as flexible as possible, we implemented this module as a function which updates the information or put the fetched object into a queue for processing them later, although using threads to implement it is also possible.

The *Prediction generator* block represents the task of generating predictions each time an object is requested. This block generates the prediction list of the objects with the highest probability to be accessed in a near future, by taking the statistical database contents as inputs. This list depends on the prefetching algorithm and the information gathered, which can also include external information.

Once described the modules that the surrogate implements, we describe the interface it offers to accommodate new prefetching proposals. Table 3.1 shows this interface for the different functions that the surrogate part implements. This interface permits to model proposals which do not involve changes in the HTTP 1.1 standard, although

Table 3.1: Function interface to implement prefetching algorithms

Function	**Param.**	**Result**	**Description**
prefetchInit	None	None	It is called when surrogate is started, and it is used to initialize variables or to create a thread that updates some structure periodically. Server starts to work when this function returns.
getHints	Client IP, Input and output headers	Hints list	It is called when all response headers are received from the server. It returns a list of hints of those objects that will be probably requested in the future.
updateStats	Client IP, Input and output headers	None	It is called when all response data have been sent to the client for updating user access statistics.

it is also possible to model techniques requiring changes in the standard by modifying the corresponding modules. For instance, implementing proposals which require the client to inform the server about the prefeching hits in order to reach more accurate predictions.

3.2.3 Client

The client part of the system reproduces the behavior of a set of users browsing the WWW with a prefetching enabled web browser. The client is fed with real traces collected by a Squid proxy. This module can be replaced either by other log format or even by any synthetic workload generator.

In order to simulate the behavior of the client, it is important to respect the time intervals between two successive requests. Those times are obtained from timestamp differences observed on logs. As timestamps in logs are made at the end of the transfer time, we need to subtract the duration field to the timestamp on logs to obtain the start time of the transfer. For this purpose, the environment includes a simple *chgtime* script.

According to the real behavior of a prefetching enabled browser like Mozilla, and as several research studies suggest [Crovella 98, Kokku 03], the prefetching of objects should be performed only when the user is not downloading objects. Furthermore, those proposals suggest stopping the prefetch connection when a user request rises during the prefetch transference. The client simulator permits to stop the prefetch transference, if wanted, by setting the corresponding parameter (*StopPrefetch*) in the

Table 3.2: Client configuration file. Main parameters and description

Parameter	**Description**
StopPrefetch	Stopping or not the prefetch requests when a new user request is performed
HintsOfPrefetch	The moment the client includes the received hints attached to the prefetched objects in the hint list: 0, when received, 1, when prefetched object is accessed
MaxLongHintList	Maximum length of the list of hints (for the client)
Bandwidth	Max number of bytes a user can receive per second
EnablePrefetch	Type of prefetching hints:0, no prefetch; 1, prefetch when hints received
ConnectDelay	Time (in ms) that a new connection to the server is delayed
ServerDelay	Time (in ms) that a new request to the server is delayed
Connections	Max number of connections a client can do simultaneously to the server
http11	HTTP version (1.1 o 1.0)
ClearList	It tells the simulator whether to clear the hint list when a new user connection arrives after an idle period

configuration file. This option aborts the prefetch, so losing the connection. Table 3.2 shows the main parameters that this configuration file includes.

The client part has also the ability to introduce latency for each downloaded object simulating the time a client waits to get an object. There are three parameters in the simulator affecting the response time of an object: the connection time, the server processing time and the available bandwidth.

These parameters are not enough to model closely a real network. To this end, an external delay generator can be included in the simulator, like the ns2 proposed in [UCB]. By using a generator it is possible to simulate scenarios having any particular kind of clients; e.g., systems with wireless clients losing a lot of packets and/or cable users with a high bandwidth. Analogously, it is possible to model different network conditions on the server; e.g., congestion and packet loss.

When simulating cache clients, the simulator only stores the object references in order to avoid the fetching of the object from the server later. Note that, for our purposes, it is not necessary to store the content of each object. However, as explained below, this must be considered when working in the client part of the proxy cache.

Once the hints attached to the prefetched objects are received, they must be included into the prefetching list. The parameter shown in Table 3.2 as HintsOf-Prefetched states the moment when they are included. There are two alternatives: i) when received, and ii) when accessed the prefetched object.

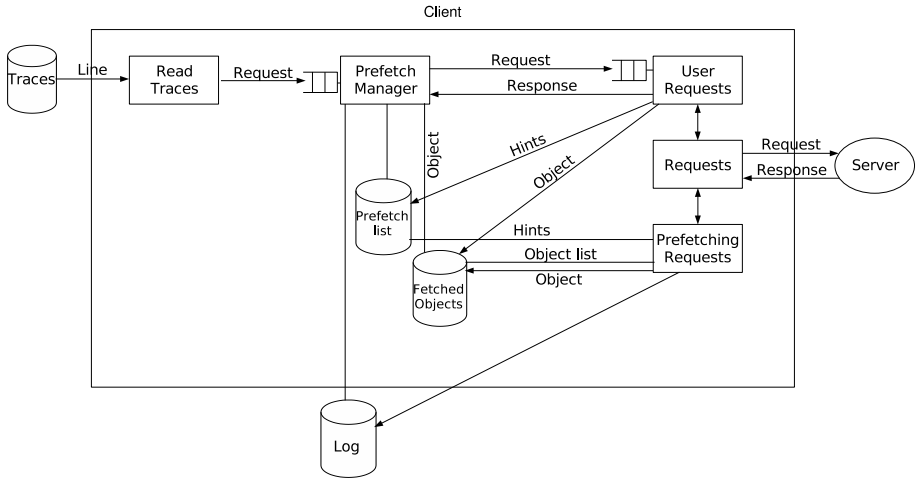


Figure 3.3: Block diagram of the client

Figure 3.3 shows the block diagram of the client simulator. As observed, the client part communicates only with the Squid traces, the log file (for analyzing the results achieved by prefetching later), and the server. The latter, depending on the modeled system, could be replaced by one or more proxies.

There are two data stores inside the client: the *Prefetch List* and the *Fetched Objects*. The *Prefetch List* contains the list of objects that the server hints the client to prefetch. These hints have been piggybacked with the requested object by the user, and accumulated for each burst of user requests. When there is a new user request after an idle period, the remaining hints in the list are removed. If one does not want to remove the hints, the corresponding variable (*ClearList*) in the configuration file must be set to zero. The *Fetched Objects* data store keeps all the fetched objects and, if they are prefetched, stores also the information on whether the user has accessed them later or not. This information can be used to simulate an infinite cache in the client, because objects in this data store were never fetched or prefetched. Of course, this behavior can be deactivated in the simulator by means of the corresponding parameter (*EnableCache*) in the configuration file.

The *Read Traces* block reads the users requests from a Squid log. This module reads each access and inserts it in a different FIFO queue for every client. Each insertion is timed according to the timestamps of the Squid log file to match the access pattern of the original user. It works in a similar way the HTTP Blast does [Vöckler], which is a tool that reads a trace file with URLs and submits them to a proxy cache, but in this case it is integrated into the client simulator. This block is implemented as an independent thread connected to the rest of the system through FIFO queues. As it is independent, it can be easily replaced by a synthetic workload

generator to feed the environment with the desired number of clients and requests. The way of managing multiple clients is by means of multiple processes (one per client) that work independently to request objects to the server.

The *Prefetch Manager* block reads requests from the *Read Traces* queue and asks for the object if it has not been prefetched yet (by checking the *Fetched Objects* data store). In order to request files to the server, the *Prefetch Manager* has several *User Request* threads (the connections parameter of the configuration file) created and waiting for a new request. When the object is received, if its URL is in the *Prefetch List*, it is removed. Then, that URL is inserted into the *Fetched Objects* data store in order to avoid prefetching the object later.

If the Read Traces queue is empty and the prefetch parameter is set to *Only when idle*, the *Prefetch Manager* allows the prefetching until a new user request rises. When a request is inserted into the queue, the *Prefetch Manager* indicates that the prefetched downloads must be cancelled and then the *Hint List* data store is cleared. Both behaviors are configurable by using the *EnablePrefetch* and *ClearList* parameters, respectively.

The *User Requests* block receives requests from the *Prefetch Manager* and redirects them to the Requests block, reusing the socket when using the HTTP 1.1 version. When an object is received, this block inserts the prefetching hints piggybacked with the object into the *Prefetch List* data store, and the URL of the object into the *Fetched Objects* data store in order to avoid the prefetching of the object later. If it already exists, the reference of that object is removed from the *Prefetch List*.

The *Request* block is a common interface (used both by user and prefetch requests) for communicating with the web server (or the proxy). This block handles the communication sockets at low level with the possibility of reusing connections if the client is configured to use HTTP/1.1 persistent connections [Fielding 99].

When the simulator is running, it writes a log file that gathers information about each request to the web server as well as the client request hits to the prefetched object cache. By analyzing the log file, the most relevant performance metrics (recall, precision, bandwidth overhead, etc) can be obtained. To this end, we implemented the results generator program, which obtains the following parameters: the amount of user accessed objects, cache hits, prefetch hits, prefetched objects, objects requested to the server, the sum of the response time of objects, total bytes transferred, precision of prefetch and recall. Note that the process of metrics generation is done at post-simulation time from the Log generated file.

3.2.4 Proxy Server

Prefetching in the proxy server has many advantages. First, it allows users of non-prefetching browsers to benefit from server prefetch hints, in the same way we discussed above with respect to the surrogate. Second, it has information from more sources (the clients and the servers), so it is the one that can make more complex and precise predictions about how many and which objects should be prefetched. For

example, one can decide to prefetch only those objects cached at the proxy server (as proposed in [Fan 99]) or restrict the amount of bandwidth overhead tolerated for prefetching purposes. As predictions are more accurate, prefetching uses less bandwidth and involves fewer overloads in the server.

When the different parts of the system support prefetch, many different behaviors can be chosen. For example, the proxy server could receive hints from the server and, after processing them, it could start a prefetch connection to cache the selected objects. Then, the proxy could redirect the received hints to the client. These hints could also be removed or even updated (omitting or adding new hints) by applying its own proxy prediction algorithm, since the proxy has a better knowledge of its users than the server has. In addition, the proxy can observe cross-server links.

The structure of the prefetching proxy server can be seen as a combination of a surrogate and a client. On one hand, the proxy waits for connections and performs predictions like the surrogate; on the other hand, the proxy requests files to the server and processes hints like the client. From this point of view, only few changes should be performed, i) the objects must be stored, and ii) requests to the server must be generated from client requests (instead of from log traces as done in the client part).

By having all these three parts (surrogate, client, and proxy) we are able to: i) study, test, and evaluate any proposal already appeared in the literature and ii) create novel test environments which inter-relate the three parts of the global system. In addition, the proxy side can be split in more parts to check the performance of new organizations; e.g. prefetching between proxies.

Simulations can be run by just setting up one set of clients and one server, which is the simplest system that can be modeled. The system can become more and more complex to represent more realistic scenarios.

To run simulations, the three parts of the modeled system can be placed in a single computer or placed distributed among computers of the same or different LANs. The number of clients that a computer machine can simulate depends on the CPU requirements of the client prefetching algorithm.

3.2.5 Implementation Example

As an example of how a prefetch technique can be implemented in our environment, we use the one proposed by [Padmanabhan 96]. We omit some implementation details since the goal of this section is only to present the simulation environment.

This proposal takes into account a set of clients and the server. The server implements a prefetching algorithm similar to the one proposed by [Griffioen 94], which uses previous client requests to generate the prediction list piggybacked on the response. Then, the client receives the list, and uses it when it is idle to ask the server for the hinted objects. When a user generates a real request, the prefetch is stopped.

The behavior pattern of the users was taken from logs collected by a Squid proxy of the Polytechnic University of Valencia. We selected those requests accessing to a popular news web server, which was mirrored to avoid external interferences. The

Table 3.3: Example of performance results

Prefetch Value	**0**	**2**
User requests	30 549	30 549
Avg. latency per page (s)	2.938	2.516
Latency per page ratio	N/A	0.856
Total Transfer (KB)	101 106	138 798
Traffic increase	N/A	1.373
Precision	N/A	17.31%
Recall	N/A	8.46%

response time parameters (50ms *ConnectionDelay* and 150ms *ServerDelay*) were obtained experimentally by timing connections to the real server. The others parameters were taken according to the HTTP/1.1 standard and Mozilla implementation:

- A maximum of two connections to the server per client.
- The system only prefetches when there are no users requests in course (including the stop of prefetch connections when the user makes a new request).
- Hints list (per client) is cleared after an idle period.
- The hints piggybacked with the prefetched objects are only processed when the container object is accessed.

According to the interface presented with the surrogate, accommodating the selected algorithm in the proposed environment is limited to the implementation of the next three functions:

- We create a thread in the *prefetchInit* to update statistical information of user requests (probability of next access for each file) every ten seconds.
- The *updateStats* function marks the object that is accessed as a possible successor of previous objects accessed by the user.
- The *getHints* function obtains a list of those objects having a probability equal or higher than 40% of being accessed within the next 4 accesses as suggested in [Padmanabhan 96].

Table 3.3 illustrates the results obtained by the environment for the selected prefetching algorithm when non-prefetching and with two different prefetching methods. The results were obtained considering an initial empty cache and a small trace, so that little past information can be used for prediction. However, we want to remark that the objective of this example is only for illustrative purposes.

Figure 3.4 profiles a small set of requests from three points of view: the web server, the surrogate, and the client logs. This profile shows the dynamic behavior of the requests and it can help us to debug the prefetching algorithms implemented. URLs are cut in the example due to their length. Some other characteristics that can be observed are:

- The IP address of the six requests that the web server receives is the corresponding to the surrogate. So the server sees all the requests as if they came from the surrogate. Timestamps are omitted.

- The surrogate distinguishes between user and prefetch requests through the X-moz header.

- The first column of the client is the timestamp of the request and the second one the request type (U: user, P: prefetch).

Web Server

```
158.42.58.137 - [timestamp] "GET / HTTP/1.1"                          200 72460  "-" "-"
158.42.58.137 - [timestamp] "GET …/relojsandoz2003_2.swf HTTP/1.1"    200  6799  "-" "-"
158.42.58.137 - [timestamp] "GET …/david_ferrer030505.jpg HTTP/1.1"   200  1138  "-" "-"
158.42.58.137 - [timestamp] "GET …/posicionhome.css HTTP/1.1"         200   851  "-" "-"
158.42.58.137 - [timestamp] "GET …/gerard030402.jpg HTTP/1.1"         200  1124  "-" "-"
```

Surrogate

```
Header 7-0<--GET / HTTP/1.1
Header 7-0-->HTTP/1.1 200 OK
Header 7-8-->Link: <http://www.marca.com/.../relojsandoz2003_2.swf>; rel=prefetch
Header 7-9-->Link: <http://www.marca.com/.../david_ferrer030505.jpg>; rel=prefetch
Header 7-0<--GET /…/relojsandoz2003_2.swf HTTP/1.1
Header 7-1<--X-moz: prefetch
Header 7-0-->HTTP/1.1 200 OK
Header 7-0<--GET /…/david_ferrer030505.jpg HTTP/1.1
Header 7-1<--X-moz: prefetch
Header 7-0-->HTTP/1.1 200 OK
Header 9-0<--GET /…/posicionhome.css HTTP/1.1
Header 9-0-->HTTP/1.1 200 OK
Header 9-8-->Link: <http://www.marca.com/.../relojsandoz2003_2.swf>; rel=prefetch
Header 9-9-->Link: <http://www.marca.com/.../gerard030402.jpg>; rel=prefetch
Header 9-0<--GET /…/gerard030402.jpg HTTP/1.1
Header 9-1<--X-moz: prefetch
Header 9-0-->HTTP/1.1 200 OK
```

Client

```
1056985840.876  U 73254  269  127.42.92.155 http://www.marca.com/
1056985841.088  P  7371  212  127.42.92.155 http://www.marca.com/.../relojsandoz2003_2.swf
1056985841.248  P  1827  160  127.42.92.155 http://www.marca.com/.../david_ferrer030505.jpg
1056985841.448  U  1391  220  127.42.92.155 http://www.marca.com/.../posicionhome.css
1056985841.668  P  1843  220  127.42.92.155 http://www.marca.com/.../gerard030402.jpg
```

Figure 3.4: Requests as seen by the server, the surrogate and the client

As the surrogate submits hints with the object, the size of the response that the user receives (third column of the client log) differs from that submitted by the server (sixth column of the server).

To summarize, the proposal offers an experimental framework for testing a wide variety of web architecture related aspects. Among them, the main contributions of the proposed environment reside in the following capabilities:

1. Quantifying how the prefetch requests can interfere in the performance of the user requests. Even more, we can quantify both the self-interference and the cross-interference [Kokku 03]. This fact is possible because our environment includes the entire system.

2. Standardizing the evaluation of prefetching techniques for comparison purposes.

3. Creating novel prefetching architectures that inter-relate the three parts of the global system. Therefore, we can implement prefetching algorithms in any point of the system as well as check their impact on the overall system performance.

4. Classifying which of the prefetching technique offers the best performance for each user profile under different assumptions and conditions.

In [Domènech 04a, Domènech 04b], a summary of the environment shown in this section and its capabilities was presented.

3.3 Performance Key Metrics

3.3.1 Introduction

Among the research works found in the literature, a large set of performance key metrics are used to evaluate web prefetching benefits. This section pursues to identify the most meaningful indexes when studying the performance of different prefetching techniques.

As described in Section 2.2, all prefetching related techniques start predicting or trying to guess the next objects to which the client will access. This part of the prefetching system is usually referred as the prediction engine. Then, the prediction results are submitted to the prefetching engine, which decides whether to prefetch such results or not depending on other variables; e.g., available bandwidth or server load. In this way, prefetched objects are a subset of the predicted objects. Notice that both the prediction engine and the prefetching engine can be found in the same element (client, proxy or server). We define below some basic concepts that will be used in Section 3.3.2:

- *Predictions*: amount of objects predicted by the prediction engine.
- *Prefetchs*: amount of objects prefetched by the prefetching engine.
- *GoodPredictions*: amount of predicted objects that are subsequently demanded by the user.
- *BadPredictions*: those predictions that do not result in good predictions.
- *PrefetchHits*: amount of prefetched objects that are subsequently demanded by the user.

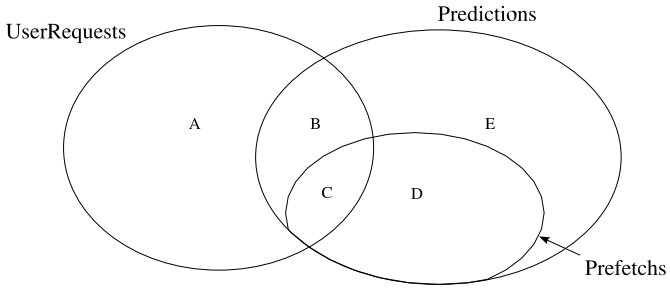


Figure 3.5: Sets representing the relation between *UserRequests*, *Predictions* and *Prefetchs*

- *ObjectsNotUsed*: amount of prefetched objects never demanded by the user.
- *UserRequests*: amount of objects that the user demands.

Figure 3.5 shows graphically the relations between the variables defined above. A represents an object requested by the user but neither predicted nor prefetched. B represents a GoodPrediction that has not been prefetched, while C represents a PrefetchHit, i.e. a GoodPrediction that has been prefetched. D is an ObjectNotUsed, which is also a BadPrediction. Finally, E represents a BadPrediction. Analogously, we can define byte related variables (Predictions_B, Prefetchs_B, and so on) by replacing the objects with the corresponding size in bytes in their definition.

3.3.2 Indexes Taxonomy

This subsection surveys the web performance indexes appeared in the open literature focusing on prefetch aspects. To the better understanding of the meaning of those indexes, we classify them into three main categories (see Figure 3.6), according to the system feature they evaluate:

- Category 1: prediction related indexes.
- Category 2: resource usage indexes.
- Category 3: end-to-end perceived latency indexes.

The first category is the main one when comparing prediction algorithms performance. It includes those indexes which quantify both the efficiency and the efficacy of the algorithm (e.g., precision). The second category quantifies the additional cost that prefetching incurs (e.g., traffic increase or processor time). This cost may become really high. For that reason, it must be taken into account when comparing

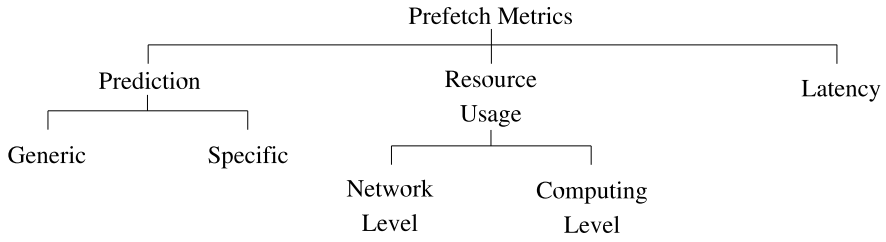


Figure 3.6: Prefetching metrics taxonomy

prefetching techniques. Therefore, those indexes are complementary measures. Finally, the third category summarizes the performance achieved by the system from the user's point of view. Notice that prefetching techniques must pay attention to the cost increase because they can negatively impact on the overall system performance (traffic increase, user-perceived latencies). Therefore, the three categories are closely related since in order to achieve a good overall performance (Category 3), prefetching systems must trade off the aggressiveness of the algorithm (Category 1) and the cost increase due to prefetching (Category 2).

Different definitions for the same index can be found in the literature (e.g., *precision*) and this fact increases the heterogeneity of the research efforts. In order to make this section more readable, we only include the definition that we consider more precise and appropriate for evaluation purposes. In the cases where several names match the same definition, we select the most appropriate index name from our point of view. The goal of this section is not only to contribute to the understanding of the indexes but also to discuss their usefulness, distinguishing those used for comparison purposes in any prefetching system from those applicable to a particular prefetching technique (i.e. specific). Specific indexes are only found in the Category 1 (Prediction), as shown in Figure 3.6.

3.3.2.1 Prediction Related Indexes

This group includes those indexes aimed at quantifying the performance that the prediction algorithm provides. Prediction performance can be measured at different moments or in different elements of the architecture, for instance when (where) the algorithm makes the prediction and when (where) prefetching is applied in the real system. Thus, each index in this category has a *dual* index; e.g., we can refer to the precision of the prediction algorithm and to the precision of the prefetch. Notice that those indexes measured when the prediction list is given do not take into account system latencies because the prediction algorithm works independently of the underlying network and the user's restrictions.

Generic Prediction Indexes

Precision (Pc) *Precision* measures the ratio of good predictions to the number of predictions [Cohen 99, Albrecht 99, Rabinovich 02, Bonino 03b, Davison 04] (see Equation 3.1). Precision, defined in this way, just evaluates the algorithm without considering physical system restrictions; e.g., cache, network or time restrictions; therefore, it can be seen as a theoretical index.

Some research studies refer to this index as *accuracy* [Loon 97, Cunha 97, Duchamp 99, Zukerman 99, Palpanas 99, Nanopoulos 01, Dongshan 02] or *hit ratio* [Bouras 04], while others use a probabilistic notation; e.g., *Pr(hit|match)* in some models based on Markov chains like [Pitkow 99].

There are other research works that measure the impact on performance of the *precision* [Loon 97, Duchamp 99]. In these cases, the number of prefetched objects and prefetch hits are used instead of the number of predictions and good predictions respectively (see Equation 3.2).

$$Pc = \frac{GoodPredictions}{Predictions} \tag{3.1}$$

$$Pc = \frac{PrefetchHits}{Prefetchs} \tag{3.2}$$

In [Ibrahim 00], they evaluate the prefetching performance through the *waste ratio*, which is defined as the percentage of undesired documents that are prefetched. As one can observe, this index is the complementary of *precision*.

Recall (Rc) *Recall* measures the percentage of user requested objects that were previously prefetched [Albrecht 99, Cohen 99, Rabinovich 02]. *Recall* quantifies the weight of the predicted (see Equation 3.3) or prefetched objects (see Equation 3.4) over the amount of objects requested by the user. Notice that this index only deals with the number of good predictions made but not with the total amount of predictions.

This metric is referred in some research works as *usefulness* [Palpanas 99, Nanopoulos 01, Bouras 04], *hit ratio* [Markatos 98, Ibrahim 00] or also *accuracy* [Davison 02, Davison 04]. *Predictability* has been employed by [Schechter 98] to refer to the upper limit of the *recall*.

$$Rc = \frac{GoodPredictions}{UserRequests} \tag{3.3}$$

$$Rc = \frac{PrefetchHits}{UserRequests} \tag{3.4}$$

Applicability *Applicability* is defined in [Bonino 03b, Bonino 03a] as the ratio of the number of predictions to the number of requests. In [Bonino 03b] authors incorrectly state that this index is a synonymous of *recall*. Notice that this index can be obtained from the previous ones (i.e., dividing *recall* by *precision*); therefore no additional information is given. This is the main reason why no other work uses this index.

$$Applicability = \frac{Predictions}{UserRequests} \tag{3.5}$$

Precision alone or together with *recall* has been the most widely used index to evaluate the goodness of prediction algorithms. The time taken between the client request making and the response receiving consists of four main components; i.e., connection establishment time, request transference time, request processing time, and response transference time. Both *precision* and *recall* are closely related to the three first components. Therefore, we consider that a complete comparison study about prediction algorithms should also include byte related indexes to quantify the last component. In this sense, similarly to some web proxy caching indexes (e.g., the *byte hit ratio* [Cherkasova 00]), we propose the use of *byte precision* and *byte recall* as indexes to estimate the impact of prefetching on the time that the user wastes when waiting for the bytes of the requested objects.

Byte Precision ($\mathbf{Pc}_B$) *Byte precision* measures the percentage of predicted (or prefetched) bytes that are subsequently requested. It can be calculated by replacing the number of predicted objects with their size in bytes in Equation 3.1.

Remark that, like *precision* does, *byte precision* quantifies how good the predictions are, but measured in bytes instead of in objects. An earlier approach to this index is the *Miss rate ratio* [Bestavros 95], described below. In [Bouras 04], authors also mentioned that different results could be obtained when using the number of bytes instead of the number of objects.

$$Pc_B = \frac{GoodPredictions_B}{Predictions_B} \tag{3.6}$$

Byte Recall ($\mathbf{Rc}_B$) *Byte recall* measures the percentage of demanded bytes that were previously predicted (or prefetched). As mentioned above, this index quantifies how many accurate predictions are made, measured in transferred bytes.

This index becomes more helpful than the previously mentioned *recall*, when the transmission time is an important component of the overall user-perceived latency.

$$Rc_B = \frac{GoodPredictions_B}{UserRequests_B} \tag{3.7}$$

Specific Prediction Indexes

Request savings *Request savings* measures the number of times that a user request hits in the browser cache or the requested object is being prefetched, as a percentage of the total number of user requests [Fan 99]. Furthermore, the request savings can be broken down into three groups depending on if they were previously prefetched, have been partially prefetched or cached as normal objects.

Notice that when prefetching is user-initiated, the number of requests increases; therefore, this index makes sense when prefetching is proxy or server initiated. Fan *et al.* use this index in a prefetch system where the proxy pushes objects to the client. Nevertheless, this index could also be used when a prefetching system pushes objects from the server to the proxy or from the server to the client (with no intermediate proxies).

Miss rate ratio *Miss rate ratio* is defined in [Bestavros 95] as the ratio between the byte miss rate when the prefetch is employed to the byte miss rate when the prefetch is not employed, where the byte miss rate for a given client is the ratio of bytes not found in the client cache to the total number of bytes accessed by that client.

As one can see, this index quantifies to what extent miss rate in the client cache drops due to the prefetching system. This is a specific index since it is only applicable to those systems storing the prefetched objects in the client's cache.

Probability of making a prediction In [Pitkow 99], they quantify the probability that the last accesses match the pattern prediction; in such case, the prefetch system computes the prediction outcomes. This index can be applied, for example, to those systems based on Markov models, but cannot be applied to a large subset of prefetching systems; e.g., the top-10 approaches [Markatos 98]; so, it is classified as specific.

3.3.2.2 Resource usage

The benefits of prefetching are achieved at the expense of using additional resources. This overhead, as mentioned above, must be quantified because they can negatively impact on performance.

Although some prediction algorithms may require huge memory or processor time (e.g., high order Markov models), it is not the current general trend, where the main prefetching bottleneck is the network traffic. Therefore, we divide indexes in this category into two subgroups: network level and computing level.

Network level

Traffic Increase ($\Delta \mathbf{Tr}_B$) *Traffic increase* quantifies the increase of traffic (in bytes) due to unsuccessfully prefetched documents [Markatos 98] (see Equation 3.8). It is also called *wasted bandwidth* [Fan 99], *extra bytes* [Rabinovich 02], *network traffic* [Palpanas 99, Bouras 04], and *bandwidth ratio* [Bestavros 95].

When using the prefetch, the network traffic usually increases due to two side effects of the prefetch: the objects not used and the extra information interchanged. Objects that are not used waste network bandwidth because these objects are never requested by the user. On the other hand, the network traffic increases due to the prefetch related information interchange, called *Network overhead* by [Duchamp 99]).

Several research studies fail to take into account that overhead [Markatos 98, Nanopoulos 01, Khan 03]; therefore, their results cannot estimate prefetching costs accurately.

$$\Delta Tr_B = \frac{ObjectsNotUsed_B + NetworkOverhead_B + UserRequests_B}{UserRequests_B} \tag{3.8}$$

Extra control data per byte *Extra control data per byte* quantifies the extra bytes transferred that are related to the prefetch information, averaged per each byte requested by the user. It is the *Network overhead* referred in [Duchamp 99], but quantified per requested bytes, and will be used below when relating *Traffic increase* to the prediction indexes.

$$ExtraControlData_B = \frac{NetworkOverhead_B}{UserRequests_B} \tag{3.9}$$

Object traffic increase ($\Delta \mathbf{Tr}_{ob}$) *Object traffic increase* quantifies in which percentage the number of documents that clients get when using prefetching increases. This index is referred by [Nanopoulos 01] as *network traffic* and by [Rabinovich 02] as *extra requests*.

As Equation 3.10 shows, this index estimates the ratio of the amount of prefetched objects never used with respect to the total user's requests. It is analogous to the *traffic increase*, but it measures the overhead in number of objects.

$$\Delta Tr_{ob} = \frac{ObjectsNotUsed + UserRequests}{UserRequests} \tag{3.10}$$

Computing level

Server load ratio *Server load ratio* is defined as the ratio between the number of requests for service when speculation is employed to the number of requests for service when speculation is not employed [Bestavros 95].

Space and Time overhead In addition to the server load, some research works discuss how the overhead impacts on performance. For instance, [Duchamp 99] discusses the memory and processor time that the prefetch could need.

Prediction time This index quantifies the time that the predictor takes to make a prediction. It is used in [Dongshan 02] to compare different predicting algorithms.

3.3.2.3 Latency related indexes

Indexes belonging to this category include those aimed at quantifying the end-to-end latencies; e.g., user or proxy related latencies. The main drawback of these indexes is that they include several time components, some of them difficult to quantify. Researchers do not usually detail which components they measure, although they use a typical index name; i.e., latency. Several names have been used instead; for instance, *access time* [Padmanabhan 96], *service time* [Bestavros 95] and *responsiveness* [Tao 02, Khan 03]. This situation is not the best for the research community, due to the fact that the different proposals cannot be fairly compared among them.

Through the different research works, latencies are measured both per page and per object. The *latency per page* (L_p) is calculated by comparing the time between browser's initiation of an HTML page GET and browser's reception of the last byte of the last embedded image or object for that page [Duchamp 99]. Analogously, the *latency per object* (L_{ob}) can be defined as the elapsed time since the browser requests an object until it receives the last byte of that object. In order to illustrate the benefits of prefetching, researchers calculate the ratio of the latency that prefetching achieves (either per page [Loon 97, Duchamp 99, Fan 99] or per object [Kroeger 97]) to the latency with no prefetching.

Unfortunately, some proposals that use *latency* when measuring performance do not specify to which latency they are referring; e.g., [Bestavros 95, Khan 03, Bouras 04]. This fact can be misleading because both indexes do not perform in the same way. In order to illustrate this point we present the following example: a user requests an HTML object embedding two images (IMG1 and IMG2). As Figure 3.7 shows, the transference of the HTML file starts at t_0 and ends at t_5. At times t_1 and t_3 the browser reads and processes the IMG tags of the embedded images. Then it starts the transference, which end at t_2 and t_5 respectively. In this case, the cumulative L_{ob} is the sum of the time taken by the three transferences (L_{ob}= t_4-t_0 + t_2-t_1 + t_5-t_3) where the *latency per page* (L_p) is t_5-t_0. If it is assumed that IMG1 was previously prefetched, no waiting time for such object will be plotted; i.e. t_1 will match t_2, in such a way that it will reduce L_{ob} but not L_p, which will remain the same value.

To observe this feature in a real environment for a given client it is necessary that the object retrieving times are independent. Nevertheless, although times are not independent, both indexes have different values, as experimental results show.

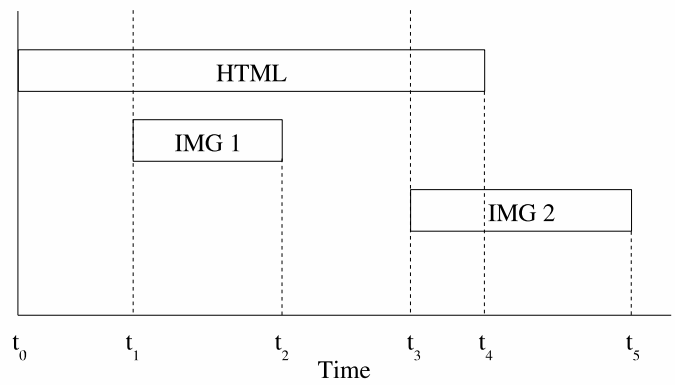


Figure 3.7: Example where latency metrics behave differently when the prefetch hits on one of the images

Furthermore, the *latency per object* measured when an object is downloaded by the web browser has two main components: i) the queuing time, since the browser has a limited number of connections, and ii) the request completion time, including the time of connection establishment (if needed) and the object transference. The first component is often ignored [Bouras 04] and other research studies [Bestavros 95, Kokku 03] do not specify whether the measured latency includes the queuing time. The first component should not be ignored because prefetch hits do not compete for a free connection so the queuing time of the remaining objects decreases and, consequently, their latency. Note that all the stated relations between both latency related indexes are valid for any web latency reduction technique and not only for prefetching. In this sense, the definitions of these indexes can be adapted to include any web technique aimed at reducing the final user-perceived latency. The latency per page (or per object) ratio can be defined, in a wider way, as the ratio between the latency per page (or per object) when a latency reduction technique is used to the latency per page (or per object) when that technique is not used.

3.3.2.4 Summary: synonymous and experimental index category used

Through the proposed taxonomy, we have discussed the large variety of index names (synonymous) used to refer to the same metric. This heterogeneity can be observed through web performance studies appeared in the literature, a fact that adds extra difficulty to obtain a general view of the subject. Table 3.4 offers a scheme of the current situation. In addition, we also have found that the same index name has been used when measuring different variables (e.g., accuracy appears both for precision and for recall). As one can observe, the widest heterogeneity is present in the first category due to the wide range of prediction algorithms appeared in the literature

and due to its importance in the prefetching systems, which are the main focus of this work.

Table 3.5 classifies a representative subset of research works that can be found in the open literature and shows the categories of indexes that are used in such works. As discussed above, a complete research work should include indexes belonging to the three categories. However, we only have observed this fact in 16% of the explored works, just four of the twenty-five works (row 1). Notice that the first column shows that 80% of the research works have at least one of the indexes belonging to the prediction category. Moreover, 44% of the studies only measure prediction metrics to evaluate the system performance. The second column states that 60% of the papers do not measure resource usage metrics. Finally, the third column shows that 60% of the research works do not quantify the end perceived latencies.

3.3.3 Relation between indexes

This subsection has three main goals. The first one is to illustrate the usefulness of the indexes introduced in this chapter (i.e., *byte precision, byte recall,* and *extra control data per byte*) as well as the way in which they differ from the analogous classical ones. The second one is to study and show how indexes are related among them. Finally, the third goal is to identify the most meaningful indexes when evaluating the overall system performance. We use both empirical and analytical approaches to show how indexes are interrelated. When analytical relations were found, results for the empirical approach are not shown.

3.3.3.1 Empirical tests

The relations among indexes are checked by comparing them by pairs that could be potentially related through different prefetching scenarios. Pairs were selected from the most widely used indexes as discussed in Section 3.3.2: *precision, byte precision, recall, byte recall, latency per page* and *latency per object*.

In order to detect the possible relations, a four-graphic figure is plotted for each pair of indexes as a result of combining the two prediction algorithms using two traces. Then, in those graphics where some sign of linear relation appears, we quantify it through the linear correlation coefficient. In order to observe how the indexes behave in a wide range of situations, the experiments were run ranging the *threshold* value of the prefetching algorithms from 0.1 to 0.9 (with a 0.1 increment). Points in the graphics refer to the performance indexes measured for every client in each experiment. Remark that the measured values of the prediction indexes refer to the prefetched objects instead of the predicted objects since they are closer to the system performance.

Prefetching Algorithms The experiments were run using two different prediction algorithms: the Dependency Graph (DG) and the Prediction by Partial Match (PPM), both described in detail in Section 2.3.

Table 3.4: Relation between the selected index name in this work and those appearing in the literature

Category	**Selected name**	**Literature**	
		Name	References
1. Prediction	*Precision*	*Precision*	[Cunha 97, Albrecht 99, Rabinovich 02, Bonino 03b, Bonino 03a, Davison 04]
		Accuracy	[Loon 97, Duchamp 99, Cohen 99, Zukerman 99, Palpanas 99, Nanopoulos 01, Dongshan 02]
		Pr(hit|match)	[Pitkow 99]
		Hit ratio	[Bouras 04]
		Waste ratio	[Ibrahim 00]
	Recall	*Recall*	[Albrecht 99, Cohen 99, Rabinovich 02]
		Usefulness	[Palpanas 99, Nanopoulos 01, Bouras 04]
		Hit ratio	[Markatos 98, Ibrahim 00]
		Predictability	[Schechter 98]
	Applicability	*Applicability*	[Bonino 03b, Bonino 03a]
2. Resource usage	*Traffic increase*	*Traffic increase*	[Padmanabhan 96, Markatos 98, Duchamp 99]
		Wasted bandwidth	[Fan 99]
		Bandwidth ratio	[Bestavros 95]
		Extra bytes	[Rabinovich 02]
		Data transfer	[Khan 03]
		Network traffic	[Palpanas 99, Bouras 04]
	Object traffic increase	*Network traffic*	[Nanopoulos 01]
		Extra requests	[Rabinovich 02]
	Prediction time	*Prediction time*	[Dongshan 02]
3. Latency	*Latency per page*	*Latency*	[Loon 97, Duchamp 99, Fan 99]
		Responsiveness	[Tao 02, Khan 03]
	Latency per object	*Latency*	[Kroeger 97, Kokku 03, Bouras 04]
		Access time	[Padmanabhan 96]
		Service time ratio	[Bestavros 95]

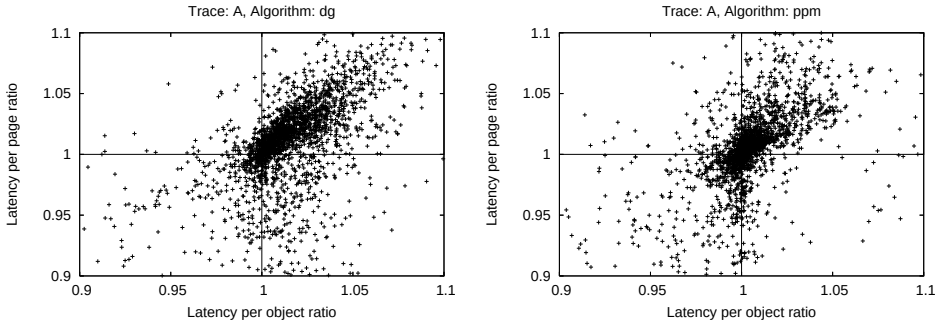
Table 3.5: Relation between the research studies and the indexes used, grouped by category

References	**Category**			**%**
	1. Prediction	2. Resource	3. Latency	
[Bestavros 95, Fan 99, Duchamp 99, Bouras 04]	X	X	X	16
[Padmanabhan 96, Khan 03]		X	X	8
[Loon 97]	X		X	4
[Kroeger 97, Tao 02, Kokku 03]			X	12
[Markatos 98, Palpanas 99, Nanopoulos 01, Dongshan 02]	X	X		16
[Cunha 97, Schechter 98, Zukerman 99, Cohen 99, Albrecht 99, Pitkow 99, Ibrahim 00, Davison 02, Bonino 03b, Bonino 03a, Davison 04]	X			44
TOTAL	80%	40%	40%	

Workload The behavior pattern of the users was taken from logs collected during eight days, between May 5th and May 12th 2003, by a Squid proxy of the Polytechnic University of Valencia. From this log, accesses to two of the most popular servers in this environment were taken and reproduced in the simulation, using the first week to train the prefetching algorithm and the following day to obtain the performance indexes. The first trace (A) contains accesses to a news web server, whereas the second one (B) includes the accesses to a student information web server. The main characteristics of trace can be found in Table 3.6.

Figure 3.8 plots the *latency per object* ratio to the *latency per page* ratio, both referring to different alternatives about how latency can be measured (as discussed in Section 3.3.2.3). The points refer to the performance index measured for every client in each experiment.

Notice that depending on the way the latency is measured, it is possible to reach not only different but also opposite conclusions. Suppose that we take a point in the upper left side quarter and we consider the *latency per object*; in such case, the prefetching system outperforms the non-prefetching system. However, if we considered the *latency per page*, we would conclude the opposite. The correlation coefficient



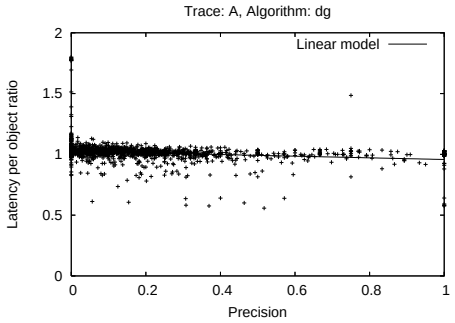
(a) Trace: A, algorithm: DG

(b) Trace: A, algorithm: PPM

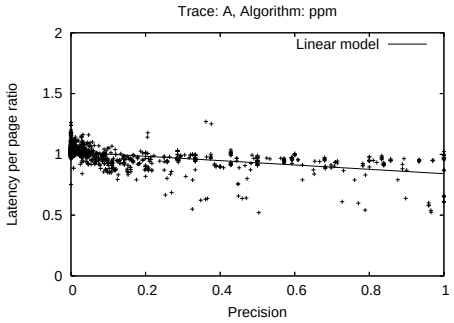
(c) Trace: B, algorithm: DG

(d) Trace: B, algorithm: PPM

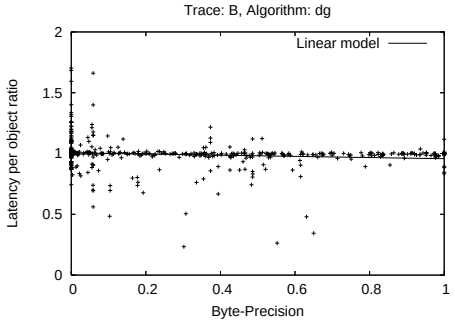
Figure 3.8: *Latency per object* ratio as a function of *latency per page* ratio



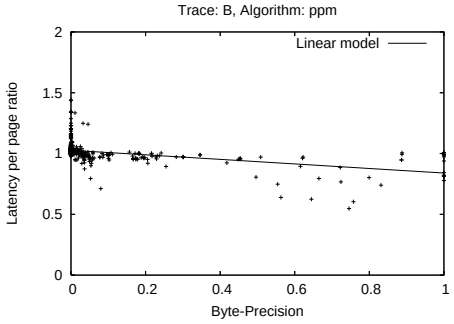
(a) Trace: A, algorithm: DG



(b) Trace: A, algorithm: PPM

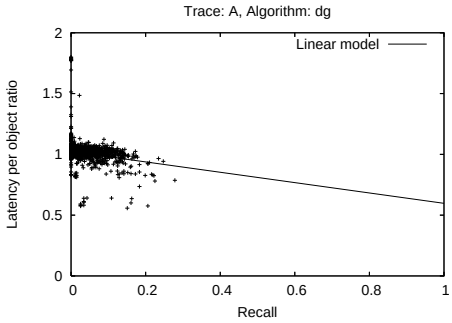


(c) Trace: B, algorithm: DG

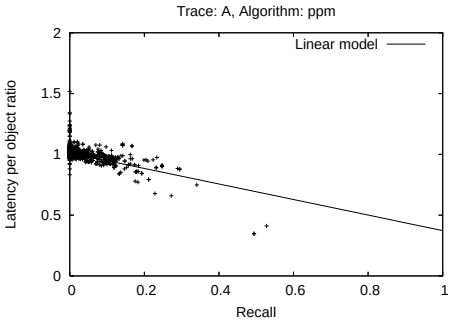


(d) Trace: B, algorithm: PPM

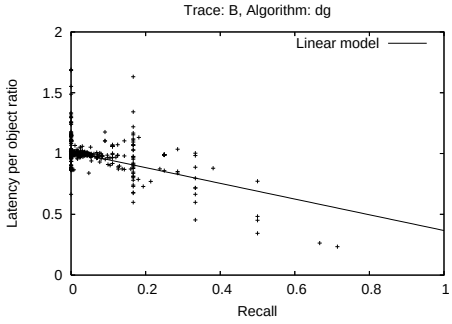
Figure 3.9: Relation between *precision* and latency related indexes



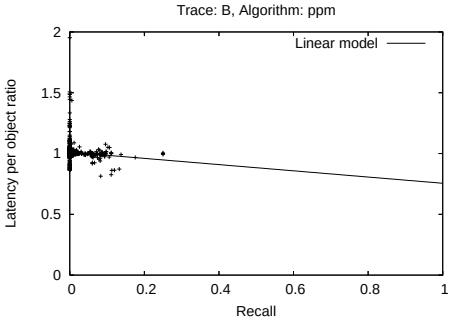
(a) Trace: A, algorithm: DG



(b) Trace: A, algorithm: PPM

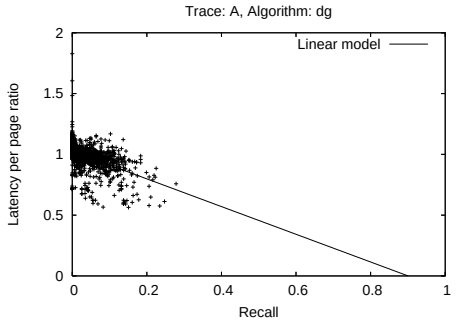


(c) Trace: B, algorithm: DG

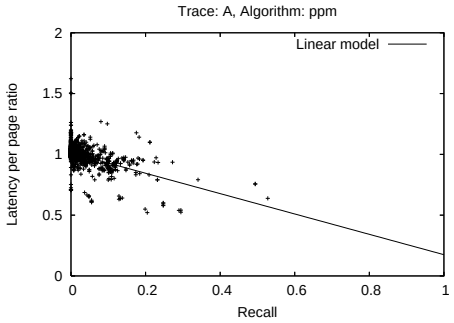


(d) Trace: B, algorithm: PPM

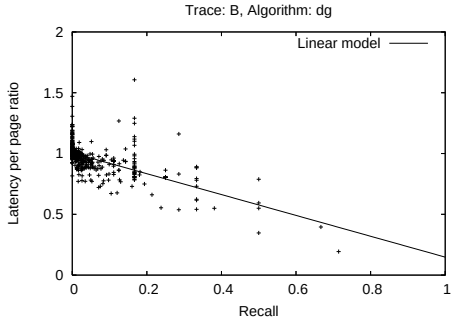
Figure 3.10: *Recall* as a function of *latency per object* ratio



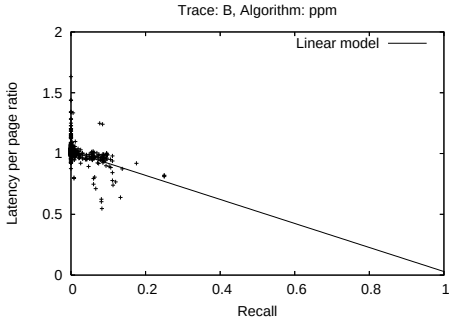
(a) Trace: A, algorithm: DG



(b) Trace: A, algorithm: PPM

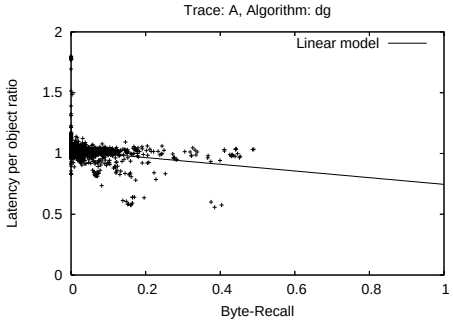


(c) Trace: B, algorithm: DG

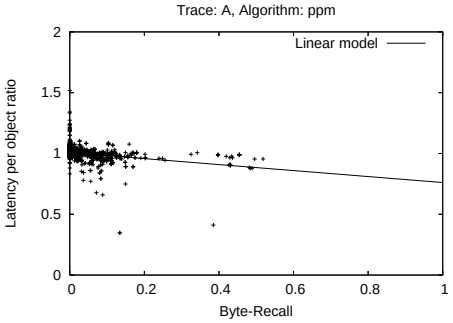


(d) Trace: B, algorithm: PPM

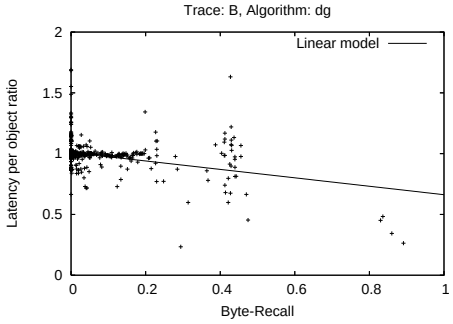
Figure 3.11: *Recall* as a function of *latency per page* ratio



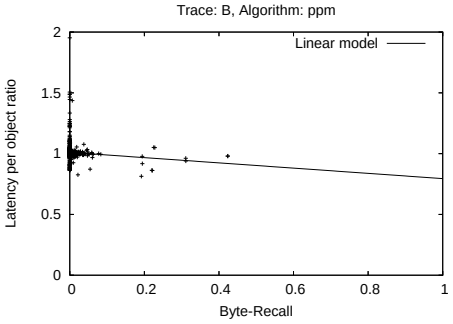
(a) Trace: A, algorithm: DG



(b) Trace: A, algorithm: PPM

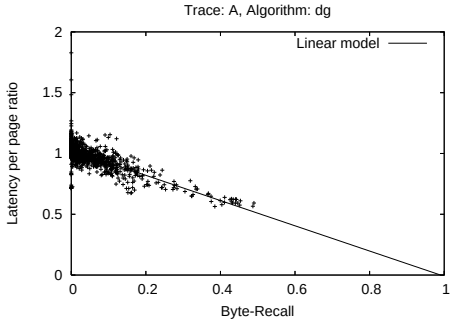


(c) Trace: B, algorithm: DG

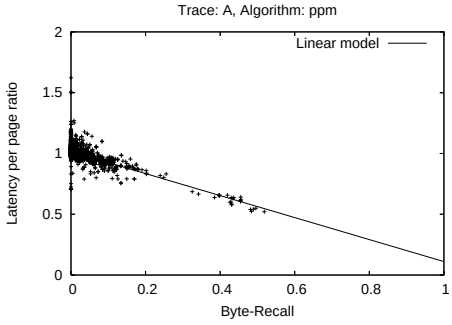


(d) Trace: B, algorithm: PPM

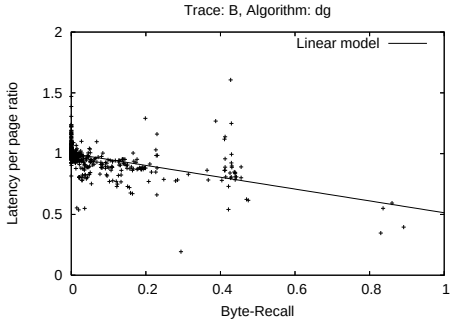
Figure 3.12: *Byte Recall* as a function of *latency per object* ratio



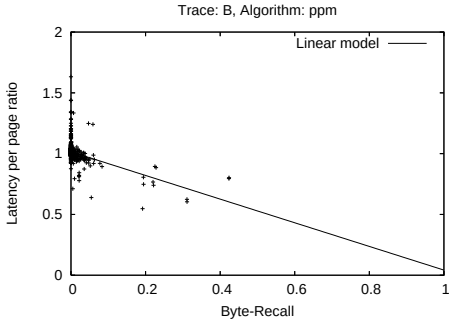
(a) Trace: A, algorithm: DG



(b) Trace: A, algorithm: PPM



(c) Trace: B, algorithm: DG



(d) Trace: B, algorithm: PPM

Figure 3.13: *Byte Recall* as a function of *latency per page* ratio

Table 3.6: Trace characteristics

Trace	**A**	**B**
Training accesses	251,271	148,213
Experiment accesses	65,866	37,655
HTMLs accesses	2,269	1,717
Users	300	132
Bytes transferred (KB)	188,600	68,428

Table 3.7: Linear correlation coefficient between precision and latency indexes

		Pc		Pc_B	
Lat.	Trace/Alg	DG	PPM	DG	PPM
L_p	A	–0.3333	–0.5453	–0.3684	–0.6107
	B	–0.3495	–0.3436	–0.1533	–0.4929
L_{ob}	A	–0.2068	–0.2667	–0.2663	–0.2114
	B	–0.1292	–0.1564	–0.3855	–0.1944

between both latency ratios corresponding to the points plotted in Figure 3.8 ranges from 0.55 to 0.77, i.e., the indexes present a certain linear correlation but it is far from being strong; so *latency per object* and *latency per page* ratios cannot be directly comparable.

As a consequence, we suggest that studies should differentiate the use of both latency ratios because they are aimed at exploring the performance from different points of view. The *latency per page* ratio evaluates the system performance from the user's point of view (since it measures the latency as perceived by the user) while the *latency per object* ratio measures the performance from the point of view of each individual object that composes the pages. Therefore, it should be used when the

Table 3.8: Linear correlation coefficient between recall and latency indexes

		Rc		Rc_B	
Lat.	Trace/Alg	DG	PPM	DG	PPM
L_p	A	–0.4088	–0.5089	–0.5780	–0.7227
	B	–0.5037	–0.4149	–0.5309	–0.4579
L_{ob}	A	–0.2311	–0.6023	–0.2318	–0.3049
	B	–0.6685	–0.1083	–0.3780	–0.1024

meaning of a page is not so clear; for instance when we are measuring the latency in a proxy server.

Figure 3.9(a) and 3.9(b) present an almost horizontal curve showing no apparent relation between precision and latency related indexes. The correlation coefficients showed in Table 3.7 quantify this negligible linear correlation, which confirms the visual appreciation. Graphics 3.9(c) and 3.9(d) show that there is a weak relation between the *byte precision* index and the latency related indexes. The remainder combinations of traces and algorithms are not shown because they are very similar.

Figure 3.10 shows the relation between *recall* and *latency per object* ratio. As one can observe, depending on the workload and the prediction algorithm considered, a linear relation between the recall index and the *latency per object* ratio might exist. Some extra elements not gathered by the recall, like the queuing time and the object size heterogeneity, justify why the relation is not stronger. Figure 3.11 shows that a certain correlation exists between the *recall* and the *latency per page* ratio, but it is far from being strong, as the linear correlation coefficient shows in Table 3.8. The *latency per page* ratio is not completely explained by the *recall* index since it involves more elements that affect it, like the simultaneous transference of objects (as explained in Section 3.3.2.3). These results corroborate the differences between both latency indexes observed in Figure 3.8.

Figures 3.12 and 3.13 show the relation between the *byte recall* and both latency indexes. The former presents a very weak correlation of the *byte recall* to the *latency per object* ratio; hence the fact that the *byte recall* does not explain the *latency per object* ratio. However, Figure 3.13 and its associated correlation coefficient show that the *byte recall* is stronger correlated to the *latency per page* ratio than to the *recall*, so this index explains the user-perceived latency better. More details about this relation can be found in Chapter 6.

3.3.3.2 Analytical Relations

In order to identify analytical relations among indexes we use the index definitions presented in Section 3.3.2. We found that the network level indexes can be obtained from the prediction related indexes. Equation 3.11 shows that the object traffic increase depends on the *recall* (Rc) and the *precision* (Pc) achieved by the prefetching system. The equation is consistent with the meaning of both indexes: a decrease in the *precision* means a higher *object traffic increase*, whereas the higher the *recall* is, the lower the *object traffic increase* is.

$$\begin{aligned} \Delta Tr_{ob} &= \frac{ObjectsNotUsed + UserRequests}{UserRequests} \\ &= \frac{P - Pc \cdot P + \frac{Pc \cdot P}{Rc}}{\frac{Pc \cdot P}{Rc}} \\ &= 1 - Rc + \frac{Rc}{Pc} \end{aligned} \tag{3.11}$$

Equation 3.12 shows the relation of the traffic increase index with those metrics related to the prediction; i.e., the *byte recall* (Rc_B), the *byte precision* (Pc_B) and the *extra control data per byte*. As this index is related to the number of bytes transferred, the main difference with the previous one lies in the fact that it depends on the *byte recall* and *byte precision* instead of on the *recall* and the *precision*. Notice that, when comparing this equation to the Eq. 3.11, a new term appears. This term refers to the *extra control data per byte*, so it can be understood as the increase in network traffic due to the transference of data used to control the prefetching engine (e.g., prefetch hints) per each byte requested by the user.

$$\begin{aligned}\Delta Tr_B &= \frac{ObjectsNotUsed_B + NetworkOverhead_B + UserRequests_B}{UserRequests_B} \\ &= \frac{P_B - Pc_B \cdot P_B + NO_B + \frac{Pc_B \cdot P_B}{Rc_B}}{\frac{Pc_B \cdot P_B}{Rc_B}} \\ &= 1 - Rc_B + \frac{Rc_B}{Pc_B} + ExtraControlData_B\end{aligned} \tag{3.12}$$

3.3.4 Key Metrics Summary

A wide variety of key metrics have appeared in the open literature in order to evaluate the performance of web prefetching systems. This section has analyzed this wide heterogeneity trying to i) clarify the index definitions and how they are interrelated, and ii) contribute to deciding which metric or index should be selected to evaluate the system performance.

We have proposed a taxonomy classifying the indexes related to prefetch in three main categories, according to the part of the system they pursue to evaluate: prediction, resource usage and latency. The goal of this taxonomy is not only to contribute to the understanding of the definition of the indexes, but it is also aimed at analyzing analogies and differences between them, in order to identify which are the most useful metrics when carrying out performance studies.

We have provided each metric or index with a definition (the one we considered more precise) from the wide variety appeared in the literature. Then, looking at these definitions we observed certain analogies. We used both empirical and analytical approaches to show how indexes are interrelated. We found analytical relations between network level indexes and prediction related indexes. To check other possible relations among indexes, we ran experiments and calculated the statistical correlation among a representative set of them. From these results, the main conclusions are:

- Performance studies should include at least latency related metrics. Depending on the goal of the performed study *latency per page* or per object is preferred. For instance, if the goal is to analyze the user's point of view, the *latency per page* must be included. However, when evaluating from the point of view of

a proxy server, the *latency per page* makes no sense due to the lack of page concept. Consequently, the *latency per object* could be the most useful metric.

- Latencies cannot be used as the only metrics to check performance. Studies must analyze how the latency reduction has been achieved for a given proposal. In this sense, resource usage indexes should be taken into account. Traffic increase is the one that provides more information; therefore, we suggest that performance studies should include at least this index.

- Our discussion has shown that it is not advisable to perform studies about the behavior of prefetching techniques just focusing on the algorithm point of view. Nevertheless, if some studies focus on this part, they should include *recall* and *byte recall* as performance metrics because they are the most correlated to the *latency per object* and *latency per page* respectively.

In [Domènech 04b, Domènech 04c, Domènech 06a, Domènech 06c], a summary of the results shown in this section was presented.

3.4 Comparison Methodology

Despite the large amount of research works focusing on web prefetching, comparative and evaluation studies from the user's point of view are rare. Some papers comparing the performance of prefetching algorithms have been published [Dongshan 02, Chen 03, Nanopoulos 03, Bouras 03, Bouras 04] but they mainly concentrate on predictive performance [Dongshan 02, Chen 03, Nanopoulos 03, Bouras 03].

In addition, performance comparisons are rarely made using a useful cost-benefit analysis, i.e., latency reduction as a function of the traffic increase. As examples of some timid attempts, Dongshan and Junyi [Dongshan 02] compare the accuracy, the model-building time, and the prediction time in three versions of a predictor based on Markov chains. Another current work by Chen and Zhang [Chen 03] implements three variants of the PPM predictor by measuring the hit ratio and traffic under different assumptions.

Nanopoulos *et al.* [Nanopoulos 03] show a cost-benefit analysis of the performance of four prediction algorithms by comparing the precision and the recall to the traffic increase. Nevertheless, they ignore how the prediction performance affects the final user. Bouras *et al.* in [Bouras 03] show the performance achieved by two configurations of the PPM algorithm and three of the n-most popular algorithm. They quantify the usefulness (recall), the hit ratio (precision) and the traffic increase but they present a low number of experiments, which make it difficult to obtain conclusions. In a more recent work [Bouras 04] they also show an estimated upper bound of the latency reduction for the same experiments.

In this section we propose a cost-benefit methodology to perform fair comparisons of web prefetching algorithms from the user's point of view.

3.4.1 Performance indexes

One of the most important steps in a performance evaluation study is the correct choice of the performance indexes. In this dissertation, the performance of the algorithms has been evaluated by using the main metrics related to the user-perceived performance and the prefetching costs as described in Section 3.3: Latency per page ratio, Traffic increase and Object traffic increase.

3.4.2 Methodology

Despite the fact that prefetching has been also used to reduce the peaks of bandwidth demand [Maltzahn 99], usually its primary goal; i.e., the benefit, is the reduction of the user's perceived latency. Therefore, the comparison of prefetching algorithms should be made from the user's point of view and using a cost-benefit analysis.

When predictions fail, the prefetched objects waste user and server resources, which can lead to a performance degradation either to the user himself or to the rest of users. Since in most proposals the client downloads the predicted objects in advance, the main cost of the latency reduction in prefetching systems is the increment of the network load due to the transference of objects that will not be used. This increment has two sides: the first is the increase in the amount of bytes transferred (measured through the *Traffic Increase* metric), and the second is the increase of the server requests (measured through the *Object Traffic Increase* metric). As a consequence, the performance analysis should consider the benefit of reducing the user-perceived latency at the expense of increasing the network traffic and the amount of requests to the server.

Each simulation experiment on a prefetching system takes as input the user behavior, its available bandwidth and the prefetching parameters. The main results obtained are traffic increase, object traffic increase and latency per page ratio values.

Comparisons of two different algorithms only can be fairly done if either the benefit or the cost have the same or close value. For instance, when two algorithms present the same or very close values of traffic increase, the best proposal is the one that presents less user perceived latency, and vice versa. For this reason, in this thesis performance comparisons are made through curves that include different pairs of traffic increase and latency per page ratio for each algorithm. To obtain each point in the curve, the aggressiveness of the algorithm is varied, i.e., how much an algorithm will predict. This aggressiveness is usually controlled by a threshold parameter in those algorithms that support it (e.g., DG, PPM-OB-TH and PPM-PG-TH) or by the number of returned predictions in those based on the top-n (e.g., PPM-OB-TOP and PPM-PG-TOP).

A plot can gather the curves obtained for each algorithm in order to be compared. By drawing a line over the desired latency reduction in this plot, one can obtain the traffic increase of each algorithm. The best algorithm for achieving that latency per page is the one having less traffic increase. In a similar way, we can proceed with the object traffic increase metric.

In [Domènech 06f, Domènech 06e, Domènech 06d], a summary of the methodology developed in this section was presented.

3.5 Workload

The benefits of a web prefetching system depend mostly on the workload chosen to evaluate its performance. For this reason, the choice of this workload is of paramount importance in order to estimate its benefits appropriately. By analyzing the open literature, we found that the workload used to evaluate prefetching is quite heterogeneous. A high amount of research works takes real traces (usually from a web server or from a proxy) to conduct the experiments [Jiang 97, Jiang 98, Venkataramani 02, Yang 03, Davison 04, Chen 05], whereas some others use synthetically generated workloads [Khan 01, Jiang 02, Nanopoulos 02, Teng 05]. Synthetic workloads do not usually take into account the structure of the web, i.e., container and embedded objects [Nanopoulos 02], while real traces are often stale when they are evaluated. For instance, [Davison 04] runs a nine-year old trace and [Chen 05] a six-year old trace.

In this section we run experiments to compare the performance of web prefetching in old and current web from two perspectives: user-perceived latency and prediction engine. Results show that there is no reason to think that prefetching studies should be conducted using old traces, since the performance of prefetching algorithms is quite different from both the prediction and the user's point of view. Hence, conclusions obtained for old workload cannot be directly moved to current web.

3.5.1 Old and Current Workload Differences

The following experiments are aimed at showing how web prefetching techniques have become less effective with the evolution of the World Wide Web.

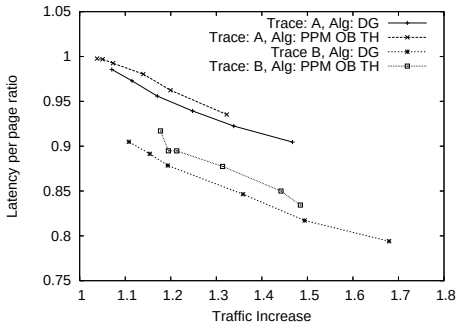
In this subsection we evaluate the performance from the user's point of view, by using the DG and the PPM-OB-TH algorithms described in Section 2.3. To show the different performance, two representative traces of current and old workload have been selected. Trace A is the same trace used in the experiments of Section 3.3.3. It contains accesses to a news web server. Trace B is the EPA-HTTP data set used in [Sarukkai 00], which is publicly available in the Web [tra]. It is a quite old trace (it dates from 1995) but it has been included in several recent works [Yang 03, Davison 04]. The main characteristics of the traces are shown in Table 3.9.

Figures 3.14 and 3.15 illustrate the results for two different user-available bandwidths, which represent dial-up and DSL users respectively. To draw the curves, the cutoff threshold of both algorithms has been ranged from 0.2 to 0.7, increasing in steps of 0.1.

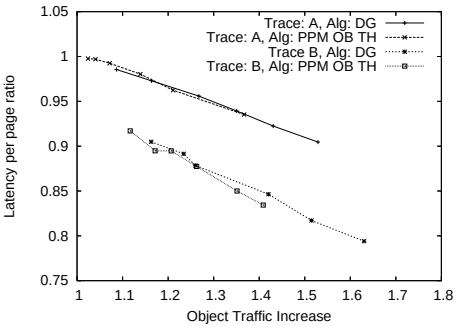
Figure 3.14 shows that for users of 1 Mbps of available bandwidth, web prefetching performance is significantly worse under the current workload (trace A). For instance, looking at Figure 3.14(a), taking into account the DG algorithm and permitting a traffic increase of 35%, web prefetching in trace C reduces the user-perceived latency

Table 3.9: Trace characteristics

Characteristics	Trace	
	A	**B**
Year	2003	1995
Users	300	2,330
Page Accesses	2,263	20,683
Object Accesses	65,569	47,748
Training accesses	35,000	22,000
Avg. Objects per Page	28.97	2.30
Bytes Transferred (MB)	218.09	285.29
Avg. Object Size (KB)	3.41	6.12
Avg. Page Size (KB)	98.68	14.12
Avg. HTML Size (KB)	32.77	8.00
Avg. Image Size (KB)	2.36	4.71

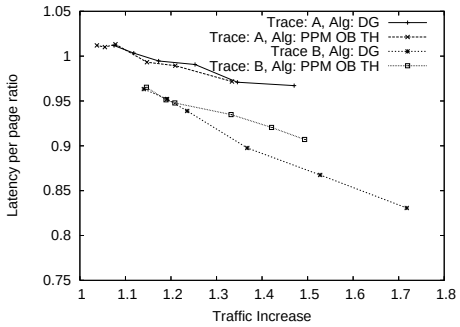


(a) Latency per page ratio as a function of Traffic Increase

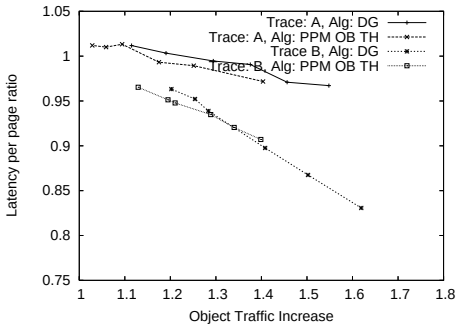


(b) Latency per page ratio as a function of Object Traffic Increase

Figure 3.14: Performance comparison between old and current workloads. Users of 1 Mbps of available bandwidth are simulated. Each point in the curves represents a given threshold, from 0.2 to 0.7



(a) Latency per page ratio versus Traffic Increase



(b) Latency per page ratio versus Object Traffic Increase

Figure 3.15: Performance comparison between old and current workloads. Users of 48 kbps of available bandwidth are simulated. Each point in the curves represents a given threshold, from 0.2 to 0.7

by about 15%. However, in trace A, this latency reduction is only about 8%. The same result is obtained if we consider as cost metric the object traffic increase, as shown in Figure 3.14(b).

Looking at Figure 3.15 one can observe that the simulation of 48 kbps of available bandwidth per user exhibits a similar trend. In this scenario, the worst case is even worse than the 1 Mbps one, since the latency per page ratio is greater than 1; i.e., the use of prefetching techniques adversely impacts on the user's perceived latency.

The prefetching techniques proposed until now in the literature achieve better results when applied to old traces, which are more or less contemporary with the prefetching proposals. One of the possible reasons of these discouraging results can be found in the dynamic nature of the current web applications that makes less predictable the user's accesses.

This remark must not be understood as meaning that prefetching techniques are not interesting to reduce the user-perceived latency in current scenarios . Quite the opposite is the case. The following chapters of this dissertation will show that prefetching techniques present better results for high bandwidth, which is the current trend.

3.5.2 Evolution of Prediction Performance

A large amount of research studies have focused on the prediction algorithm [Palpanas 99, Sarukkai 00, Nanopoulos 03, Chen 03, Lau 04, Bouras 04, Davison 04], which is designed to learn previous visiting patterns so as to *guess* user's future

accesses. In this sense, the period in which performance is measured is usually preceded by a training period for the algorithm to acquire the knowledge. During this period, the algorithm discovers, stores and classifies the universe of objects to which the user can access. The training stage finishes when the algorithm has a significant knowledge of the objects and their access patterns. In this sense, the resources used by the algorithm, e.g., CPU and memory, will increase with the size of this universe of objects.

Studying how prediction performance evolves in Web prefetching techniques is important to prevent a reduction of its potential benefits (which could turn even into performance losses) in a real working environment when considering long term scenarios. That is because this length affects the performance results when they vary through time, therefore it is important for researchers in order to determine which length of the training and testing periods makes their algorithms achieve the best performance.

In this subsection we analyze how prediction indexes vary through time in order to explore the performance evolution. To do so, we use both old (from 1995) and recent web accesses traces. Results show that prediction performance in old traces reaches the steady state in short time. However, in recent traces it may evolve negatively. This fact is the natural effect of the increasing dynamism in current web that has produced important changes in user's web accesses [Peña-Ortiz 05]. As a consequence, much of the stored information becomes stale, leading to a performance degradation as well as incurring in higher use of resources.

3.5.2.1 Background

A representative subset of research works appeared in the literature has been analyzed [Palpanas 99, Sarukkai 00, Nanopoulos 03, Chen 03, Lau 04, Bouras 04, Davison 04]. Table 3.10 summarizes the different decisions that have been made in the literature regarding the length of the training and test periods of the prediction algorithm.

Although performance results could vary significantly depending on this decision, it has been barely discussed, and most papers evaluating web prediction algorithms take an arbitrary value for the length of the training period. In fact, from those analyzed, only [Davison 04] and [Lau 04] slightly argue their training length. Prediction algorithms are evaluated in [Davison 04] without previous training. This procedure is argued to be more realistic than freezing the learning after a training period, and then evaluating the prediction algorithm. In [Lau 04], they train the algorithm during two weeks before evaluating its performance for one day. The authors state that this period is enough to show the web access patterns of each individual user based on their observations during the experimental period.

3.5.2.2 Experimental Environment

Framework As the main aim of this subsection is to analyze only the evolution of the prediction performance during a long period, there is no need to simulate the net-

Table 3.10: Test and training periods used in the literature

Reference	Trace year	Training	Test	Length
[Nanopoulos 03]	1995	1,200,000 acc.	400,000 acc.	7 days
[Lau 04]	N/A	14 days	1 day	15 days
[Sarukkai 00]	1995	40,000 acc.	7,000 acc.	1 day
[Chen 03]	1998	1 day	1 day	2 days
[Bouras 04]	N/A	400,000 acc.	100,000 acc.	7 days
[Davison 04]	1997/8	No training	Several	Several
[Palpanas 99]	N/A	30,000 acc.	56,000 acc.	5 days

work and other users' restrictions. In this way, the prediction engine module (located at the surrogate) and the prediction evaluation module (located at the client) are extracted from the framework described in Section 3.2 and adapted to work together. As a result, the simulation time is reduced dramatically.

Workload The behavior pattern of users was taken from four different logs. Traces A and B correspond to 50 days of the log used in the experiments of Section 3.3.3. Trace C is the EPA-HTTP data set used in Section 3.5.1 to evaluate the performance of web prefetching in current and old traces. Trace D is the first part of the ClarkNet Web server log that can be found in [cla]. Although it is a quite old trace, it has a significant number of users' accesses and has been used in recent research studies [Nanopoulos 03], [Gündüz 03]. The main characteristics of the traces can be found in Table 3.11. As one can observe, the differences between new and old traces are more evident in the amount of objects per page and, consequently, in the page size.

Table 3.11: Trace characteristics

Trace	A	B	C	D
Year	2003	2003	1995	1995
Duration	50 days	50 days	1 day	7 days
Users	1,627	1,115	2,330	90,499
Page Accesses	29,599	31,979	20,683	352,788
Object Accesses	658,323	506,148	47,748	1,654,849
Objects per Page	22.24	15.83	2.30	4.69
Bytes Transferred (MB)	1,907	1,996	275	13,104
Avg. Object Size (KB)	2.97	4.04	5.90	8.11
Avg. Page Size (KB)	65.97	63.91	13.61	38.03

Prefetching algorithms In order to check the evolution of the indexes in time, the experiments were run using five of the most implemented prediction algorithms in the literature: four main variants of the *Prediction by Partial Match* (PPM) algorithm and the *Dependency Graph* (DG) based algorithm, both described in Section 2.3. PPM algorithm has been proposed to be applied either to each object access [Sarukkai 00, Davison 04] or to each page (i.e. to each container object) [Palpanas 99, Dongshan 02, Chen 03] accessed by the user. In addition, two ways of selecting which object or page will be predicted have been used: predicting the top-n likely objects [Sarukkai 00, Dongshan 02, Davison 04] or using a cutoff threshold [Palpanas 99, Nanopoulos 03, Chen 03]. The four variants resulting from combining the unit of prediction (object or page) and the way of selecting the candidates (top-n or with a threshold) are implemented and analyzed.

Performance indexes The evolution of the prediction performance of an algorithm is analyzed through the main four prediction-related metrics described in Section 3.3: recall, byte recall, precision and byte precision.

3.5.2.3 Experimental results

To explore how prediction performance evolve, each index is analyzed as a function of the amount of the considered user's requests.

All plots in Figures 3.16 to 3.19 include five curves, and each one represents a different prediction algorithm. The curve labeled as *DG* corresponds to the *Dependency Graph* algorithm. The four PPM variants are labeled as *PPM x y*, where x can be *OB* or *PG* depending on the element of prediction (object or page respectively); and y can be *TOP* or *TH* depending on the way of selecting the candidates (the top-n most likely or with a threshold respectively).

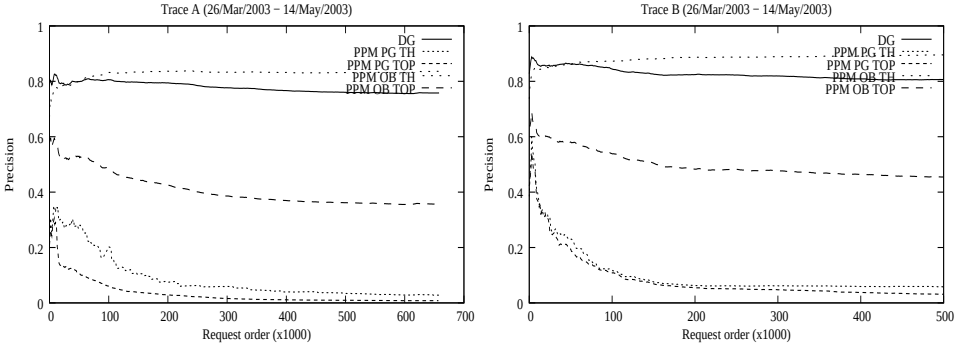
The DG algorithm was set up to use a 2 *lookahead window* and a 0.40 cutoff threshold. Similarly, PPM is based on a first-order Markov model with the same threshold for those variants, and n=*5* for the top-n ones.

Current Workload Figures 3.16 and 3.17 show how the prediction indexes evolve in current workloads as time goes by. These behaviors are also summarized in Table 3.12 to facilitate the understanding of the following analysis. The discussed metrics are divided into two main categories: precision-related and recall-related indexes. For each category and algorithm, the stationary trend is summarized as negative, steady or positive. Then, a general global trend (last column) is obtained from the trends of both categories.

Looking at Figure 3.16(a) one can observe that precision across trace A exhibits a marginally increase in the PPM OB TH algorithm in the initial state, i.e. until 100K requests. The DG presents a slight decrease and a noticeable decrease in the remaining three algorithms. After this period, all curves decline on a small scale with

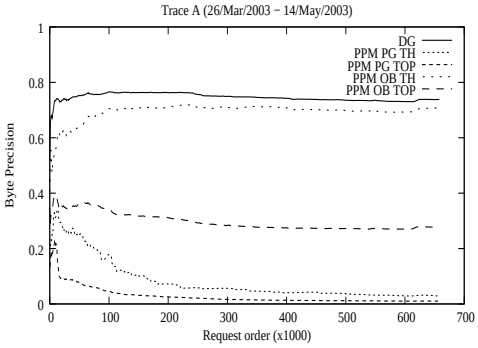
Table 3.12: Summary of the evolution of the prediction performance in the analyzed workload. Legend. I: Initial period, S: Stationary period, T: Trend, GT: Global Trend, – –: Strong negative trend, –: Slightly negative trend, =: Steady trend, +: Slightly positive trend

Trace	**Algorithm**	Pc		Pc_b		**T**	Rc		Rc_b		**T**	**GT**
		I	S	I	S		I	S	I	S		
A	DG	–	–	+	–	–	=	–	+	–	–	–
	PPM OB TH	+	=	+	–	–	– –	–	– –	–	–	–
	PPM OB TOP	– –	–	–	–	–	+	–	+	–	–	–
	PPM PG TH	– –	–	– –	–	–	+	=	=	=	=	–
	PPM PG TOP	– –	–	– –	–	–	+	–	+	–	–	–
B	DG	–	–	+	–	–	+	=	+	–	–	–
	PPM OB TH	+	=	+	=	=	–	–	– –	–	–	–
	PPM OB TOP	– –	–	– –	=	–	+	=	+	=	=	–
	PPM PG TH	– –	–	– –	=	–	+	–	+	–	–	–
	PPM PG TOP	– –	–	– –	–	–	+	=	+	=	=	–
C	DG	+	+	+	=	+	+	+	–	+	+	+
	PPM OB TH	+	+	+	–	Dif	=	=	=	=	=	Dif
	PPM OB TOP	+	=	+	=	=	+	+	+	+	+	+
	PPM PG TH	+	–	+	=	–	+	=	–	+	+	Dif
	PPM PG TOP	+	–	=	=	–	+	+	+	+	+	Dif
D	DG	+	=	+	=	=	+	=	+	=	=	=
	PPM OB TH	+	+	+	+	+	=	=	=	=	=	+
	PPM OB TOP	+	–	+	–	–	+	+	+	+	+	Dif
	PPM PG TH	–	–	– –	–	–	=	=	=	=	=	–
	PPM PG TOP	+	–	+	–	–	+	+	+	+	+	Dif

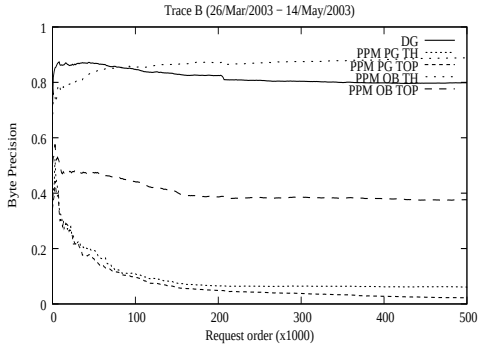


(a) Trace A. Precision

(b) Trace B. Precision

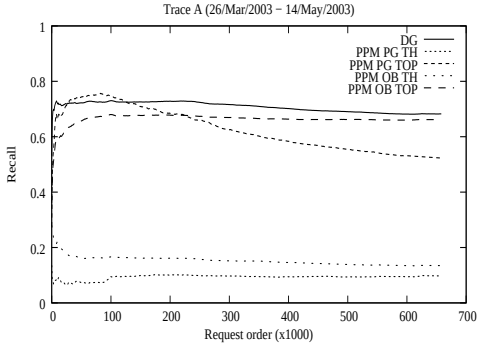


(c) Trace A. Byte Precision

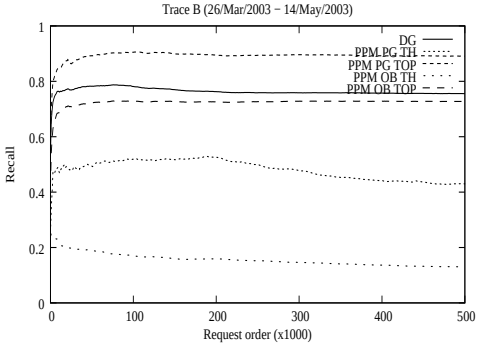


(d) Trace B. Byte Precision

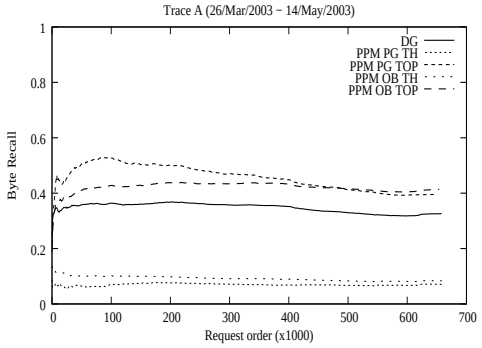
Figure 3.16: Evolution of precision metrics in current workloads



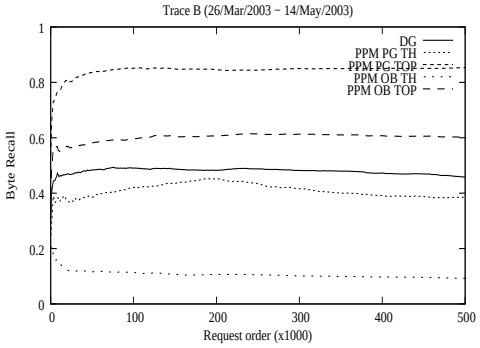
(a) Trace A. Recall



(b) Trace B. Recall



(c) Trace A. Byte Recall



(d) Trace B. Byte Recall

Figure 3.17: Evolution of recall metrics in current workloads

the only exception of PPM OB TH, which remains constant. Figure 3.16(b) shows the same behavior across trace B.

Byte precision differs from precision in the measured values, but similar trends are exhibited in trace A, as Figure 3.16(c) shows. In the initial period, the index shows distinct trends depending on the algorithm. After that, all trends become slightly descendant. In trace B (Figure 3.16(d)), the byte precision remains steady in the stationary state in three algorithms, while the other two present a weak descending trend.

Recall evolves across trace A as Figure 3.17(a) shows. In the initial period, its evolution is steady in the DG algorithm, negative in PPM OB TH and positive in the remaining three algorithms. In the stationary period, the recall has a weak descendant trend in all the algorithms with the only exception of PPM PG TOP, which is steady. Figure 3.17(b) shows that the recall, after an initial period of improvement, has a negative or steady trend.

Figures 3.17(c) and 3.17(d) show that the behavior of the byte recall index remains close to that shown in the other prediction indexes. Trends are showed in Table 3.12.

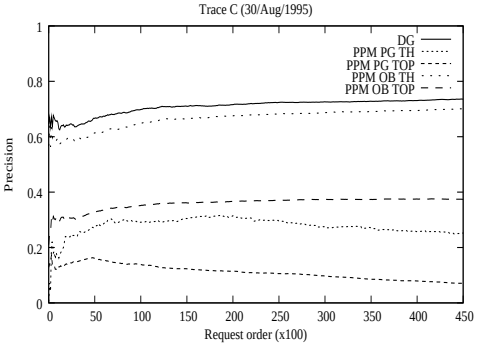
The global trend column in Table 3.12 shows that the prediction performance degrades as time goes by in all algorithms for both current traces.

Old Workload The evolution of the prediction indexes for the old workload is shown in Figures 3.18 and 3.19 and summarized in Table 3.12. Figure 3.18(a) shows that the precision in trace C exhibits a positive trend in the initial state, but then the evolution differs depending on the algorithm: It is ascendant for the DG and PPM OB TH algorithms, steady for the PPM OB TOP and descendant for the two remaining algorithms. Similarly, the stationary trend of the precision in trace D (Figure 3.18(b)) is positive in PPM OB TH, steady in DG, and negative in the remaining three algorithms. The recall index, in the traces C and D, shows a steady or positive stationary trend in all algorithms, as observed in Figures 3.19(a) and 3.19(b).

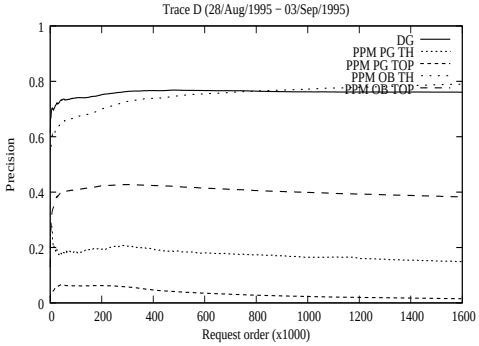
Unlike the results in current traces, when evaluating old workload a global trend for the prediction performance cannot be obtained, because it depends on the algorithm, as Table 3.12 shows. In those situations in which, for a given algorithm, the evaluated indexes exhibit opposite trends, no global trend can be provided, so we have labeled it as *Different* in the table. It represents a trade-off between two indexes, where one index can be improved at the expense of degrading the other.

3.5.3 Influence of the User-available Bandwidth on the Perceived Latency

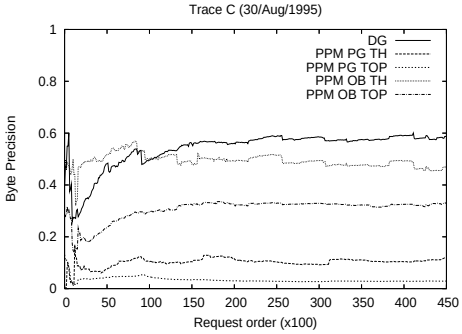
The user-available bandwidth has increased dramatically during the last few years. However, the latency perceived by users when browsing the web has not been proportionally reduced. To show how varies the user-perceived latency (by means of the latency per page metric) as a function of the increase of available bandwidth, we have simulated users with different bandwidths under the workload A used in Section 3.3.3.



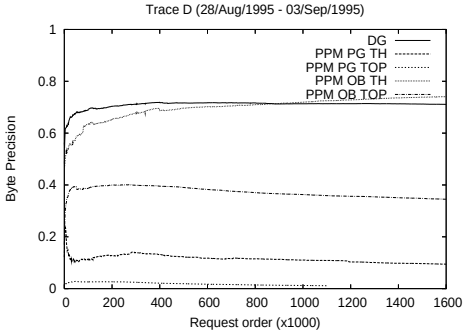
(a) Trace C. Precision



(b) Trace D. Precision

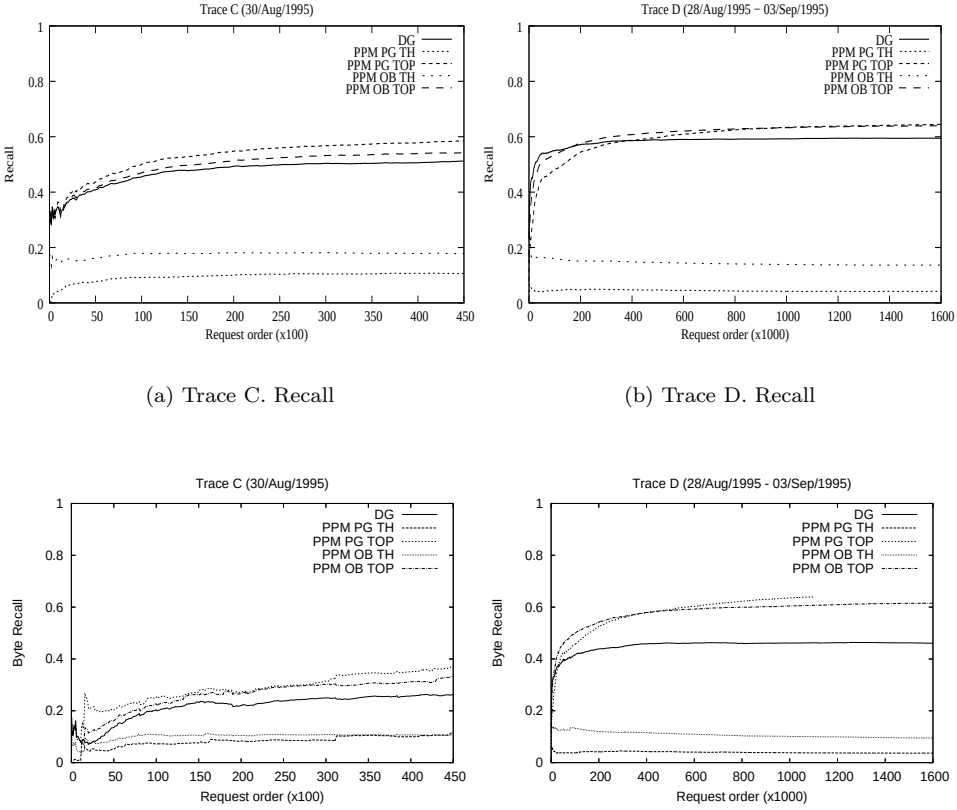


(c) Trace C. Byte Precision



(d) Trace D. Byte Precision

Figure 3.18: Evolution of precision metrics in old workloads



(a) Trace C. Recall

(b) Trace D. Recall

(c) Trace C. Byte Recall

(d) Trace D. Byte Recall

Figure 3.19: Evolution of recall metrics in old workloads

Figure 3.20 shows the evolution of the latency per page as the user-available bandwidth increases from 64 kbps to 8 Mbps. As one can observe, the perceived latency reaches its minimum value when users of 1 Mbps are simulated, and additional increases of available bandwidth do not involve reduction in the average latency per page. For this reason, the reduction of user-perceived latency by employing techniques such as prefetching acquires more importance.

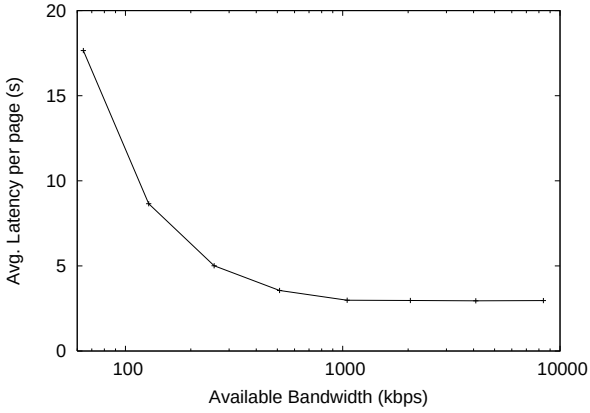


Figure 3.20: Average latency per page as a function of the user available bandwidth

3.5.4 Workload summary

In this section, we showed how prefetching techniques have become less effective with the evolution of the World Wide Web. Results showed that the techniques proposed in the literature achieved better results when applied to old traces than when they were applied under current workloads.

Section 3.5.2 explored how prediction algorithms indexes of web prefetching techniques, i.e. precision and recall, evolve according to different user's visiting patterns. For this purpose we have simulated a set of prefetching algorithms using current and old web traces, paying special attention to how these workloads affect the performance of the algorithms during the training period.

An interesting observation of our study is that when using old workloads, after the stationary state in the training is reached, the prediction performance either improves or is not affected, i.e., it remains steady or changes the trade-off between indexes. On the other hand, when current workloads are used to train any prediction algorithm, the performance degrades as time goes by. This is because the user's behavior in the current web is less foreseeable, and current contents change faster than in the past web. Therefore, prediction performance evaluation should be performed in current traces in a different way to the old ones.

The fact that performance degrades as time goes by means that some of the knowledge acquired during the training has become useless due to the changing behavior of users that is present in current web users. As a consequence, the lack of efficiency makes the algorithm increase the resource wasting, including CPU time, memory and network availability. This fact takes relevance especially when taking into account long term scenarios in which prefetching systems can run during long periods of time. In this sense, a corrector measure could help to avoid the resource wasting, like forgetting the training or freezing it after a period of time.

Finally, we have shown that, once the user-available bandwidth reaches 1 Mbps, additional increases of bandwidth hardly reduce the perceived latency when browsing the Web, so the use of techniques such as prefetching acquires more importance.

In [Domènech 05, Domènech 06g], a summary of the results shown in this section was presented.

3.6 Conclusions

In this chapter we have described and solved the main limitations when evaluating the performance of a web prefetching system. In order to perform a fair comparison study, an experimental framework has been developed. By simulating the whole web architecture, all performance issues, like the interference of prefetch requests with user requests, can be measured.

We have shown that a wide variety of performance metrics have appeared in the literature. We have analyzed this wide heterogeneity to clarify index definitions, to show how they are interrelated, and to decide which metrics should be selected to evaluate the system performance.

A comparison methodology has been provided to evaluate and compare prefetching algorithms from the user's point of view. This methodology uses a cost-benefit analysis to deal with the benefit of reducing the user-perceived latency at the expense of increasing the network traffic and the server load.

Finally, we have shown that the performance achieved by the prefetching techniques differs depending on the workload age. In addition, the evolution of the predictive performance also differs. Results also show that there is no reason to think that prefetching studies should be conducted using old traces.

Chapter 4

Theoretical Limits on Performance

4.1 Introduction

Web prefetching-related studies are performed through different prefetching architectures. For instance, [Fan 99] proposes the proxy servers to push web objects to low bandwidth clients. In a more recent work, [Bouras 04] describes a proxy where user's requests are predicted and requested to the server in advance. [Markatos 98] proposes different levels of proxies and servers collaborating to predict users' requests. Other works [Padmanabhan 96, Kokku 03, Domènech 06b] implement a system where servers predict users' future accesses and then, clients download those objects in advance.

These works provide results in different conditions and architectures, but do not show insights about how far these results are from the maximum benefits achievable by prefetching techniques. These benefits depend on several factors as well as on the considered web architecture. [Kroeger 97] presents an initial tentative to quantify the limits for latency reduction of web caching and prefetching but, unfortunately, they combine their benefits and they only focus on the proxy. In addition, some of their assumptions are not valid in current web.

This chapter analyzes the impact of the web prefetching architecture on the theoretical limits of reducing the user-perceived latency. To do so, we examine the factors that limit the predictive power of each prefeching architecture and quantify these theoretical limits both in amount of requests that can be predicted and in user-perceived latency that can be reduced. This study addresses two main issues: i) to identify the best architecture to reduce user-perceived latency when performing prefetch, and ii) to provide insights about the goodness of a given proposal by comparing it to the performance limits.

4.2 Metrics

There exist two situations in which a user access cannot be predicted. To quantify and analyze these situations, which may strongly limit the performance, new performance indexes are defined.

The first scenario in which a user access cannot be predicted is when accessing for the first time to the element of the web architecture in which the prediction engine is located. For instance, a predictor located at the web server cannot predict the first access of a user in a session. We quantify this limitation by means of the *session first-access ratio* (SFAR), described below.

The second situation in which the prediction is limited is when the prediction engine has never seen the object before. This is particularly important when the prediction engine is located at the client. We use the *first seen ratio* (FSR) to evaluate the impact of this limiting factor on the predictive performance. This index has two variants, FSR_{object} and $\text{FSR}_{latency}$, depending on whether the limits on performance are measured in number of objects that can be potentially predicted or in potential latency savings. The FSR_{object} is defined as the ratio of the amount of different objects to the total amount of requests (see Equation 4.1). When aggregating data from different clients, the FSR_{object} is calculated as the ratio of different pairs {URI, client IP} to the total amount of requests. $\text{FSR}_{latency}$ is defined as the ratio of the sum of the latency of the first appearance of each object to the latency of the total amount of accesses.

$$FSR_{object} = \frac{\#DifferentObjects}{\#ObjectRequests} \tag{4.1}$$

Like the FSR, the SFAR has different variants depending on how the limiting factor is quantified: object, latency and page. The index uses the *session* concept, defined as a sequence of accesses made by a single user with idle times between them no longer than t seconds. Since each session has always a first page, which cannot be predicted, we define the SFAR_{page} as the ratio of the number of sessions to the total amount of page accesses (see Equation 4.2). SFAR_{object} is used to quantify the amount of objects that this factor makes it impossible to predict. It is defined as the ratio of session first-accesses to the total amount of object requests, where a session first-access is a client request that is included in the first page of a session. Finally, to measure the impact of the SFAR factor on the perceived latency, the $\text{SFAR}_{latency}$ is calculated as the ratio of the latency of the session first-accesses to the latency of the total amount of accesses.

$$SFAR_{page} = \frac{\#Sessions}{\#PageAccesses} \tag{4.2}$$

Since the location of the prediction engine makes the pattern of accesses seen by the predictor vary, the described indexes are evaluated in each prefetching architecture to quantify the two situations in which the predictive performance is limited.

4.3 Workload

The workload used to quantify the limits of the reduction of web latencies when using prefetching was obtained from the log of the Squid proxy of the Polytechnic University of Valencia. The log includes 43,418,555 web accesses of 9,359 different users to 160,296 different servers. It is 7 days long, from May 31^{st} to Jun 6^{th} 2006. In the analysis, the first 6 days were taken as a training set while the trace of the last day was used to calculate performance metrics.

The log consists of one line per each client request with the standard information of the Squid logs. The main fields are the URL, the timestamp of the moment when the request was finished, the HTTP method, the proxy service time, the MIME type, the client IP and the amount of bytes transferred. In order to reproduce the original sequence of accesses made by the users, the trace was reordered by the starting time of the requests [Domènech 04a]. It was calculated by subtracting in each line the proxy service time from the timestamp. For those measurements that require that the log is grouped by page accesses, an alternative version of the log was created. To do so, all the embeddable objects (those with mime type like GIF, CSS or JS) were marked as embedded objects of the last HTML accessed by that user. Accesses not related to user's accesses, like automatic antivirus and O. S. updates were removed to avoid interferences with the user's accesses.

4.4 Predicting at the server

The main advantage of locating the prediction engine at the server is that it is the element that has the most knowledge about the objects that it contains. In addition, it is possible to avoid interferences with the user's requests that prefetching might produce by varying the prediction aggressiveness as a function of the server load. However, from the server point of view, it is not possible to gather dependencies between accesses to other servers, increasing the SFAR measured at the servers and, therefore, decreasing the potential reduction of the user-perceived latency.

Figure 4.1 shows the three variants of SFAR metric as a function of the maximum idle time allowed in a session (t parameter). As observed, for a t value of 1,800 seconds (as used in [Cooley 99]), the first pages of a session seen from the server point of view (SFAR_{page}) represent about 35% of the total pages visited by users. However, these pages contain about 45% of the accessed objects, as shown in the SFAR_{object} curve, but represent 63.4% of the latency perceived by the user, shown in the $\text{SFAR}_{latency}$ curve. Therefore, the percentage of latency that cannot be reduced due to this factor is 36.6%.

A deep analysis of this result shows that this poor potential reduction of latency is due to the fact that users access most of the servers only once. Figure 4.2 shows how the $\text{SFAR}_{latency}$ is distributed through the servers. Each point of the curve represents the ratio of servers that have a $\text{SFAR}_{latency}$ lower than its corresponding value in the X axis. As one can observe, more than 60% of servers have all their latency in the

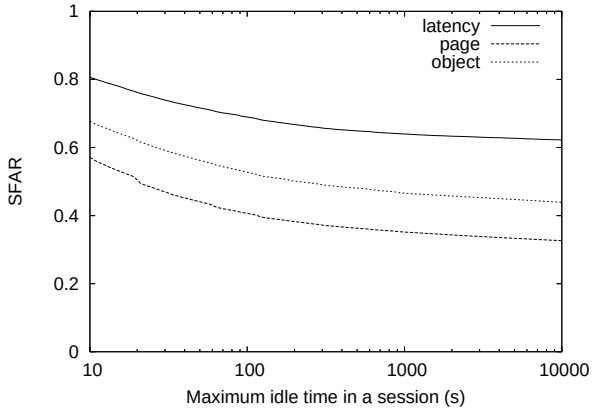


Figure 4.1: SFAR at the server as a function of the maximum idle time in a session

Table 4.1: First seen ratio at different servers

	Server				
Metric	**A**	**B**	**C**	**D**	**E**
FSR_{object}	0.5%	0.8%	0.6%	0.4%	0.6%
$FSR_{latency}$	3.9%	9.3%	1.7%	1.8%	1.7%

first pages. For this reason, the potential latency reduction highly depends on the server workload since, as shown in Figure 4.2, there is a noticeable amount of servers that could benefit from prefetching in a higher extent. For instance, 20% of servers have a $SFAR_{latency}$ lower than 37.4% (i.e., a potential latency reduction higher than 62.6%).

It is not possible to infer a precise FSR value at the servers from a proxy trace because it gathers only a part, in most cases not significant, of the requests received by a server. To obtain an approximate value, we calculated the FSR in the top-5 servers with more load gathered in the proxy. Table 4.1 shows that, among these servers, the FSR_{object} is lower than 0.8%, although the latency of these objects represents 9.3% of the user-perceived latency.

In summary, the main limitation to reduce the latency when the prediction engine is located at the server is the inability of predicting the first access to the server. This fact makes it impossible to reduce the user-perceived latency beyond 36.6% with this prefetching architecture.

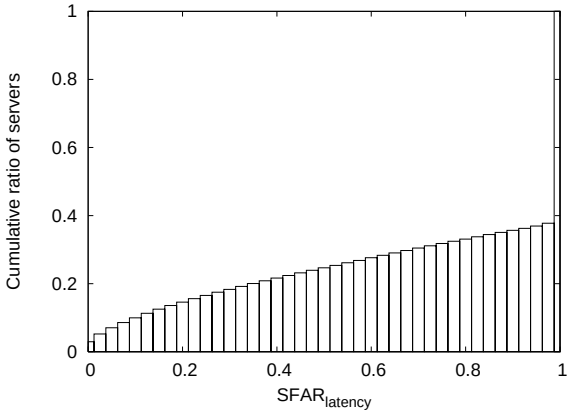


Figure 4.2: Cumulative relative frequency histogram of SFAR$_{latency}$ at the server with t=1,800 seconds to web servers

4.5 Predicting at the client

Locating the predictor engine at the client has two main advantages: i) the client is the element where the SFAR is the lowest since the predictor engine can gather all the user's accesses, and ii) it is possible to gather, and therefore, to predict, cross-server transitions. In contrast, this location has, as main shortcoming, the low amount of accesses that it receives, making FSR high.

Regarding the prediction limit that affects the prediction of objects never seen before, we found a FSR$_{object}$value of 41.6%, which is the percentage of objects demanded by a user that has never been accessed by that user before. The latency of these objects represent 45.6% of the total latency perceived by users, which is the latency that cannot be reduced when the predictor is located at the client.

Figure 4.3 shows the prediction limit related to the session first-access factor. As one can observe, the values are noticeably lower than those measured at the server. For a t value of 1,800 seconds, only 1.3% of pages are the first ones in a session, which represent 2.8% of the user-perceived latency.

In summary, unlike the predictor located at the server, the main limitation of the web prefetching techniques for reducing the user-perceived latency is the access to objects never seen before. This fact means that the maximum latency that this web prefetching architecture could reduce is about 54.4% of the latency perceived by users.

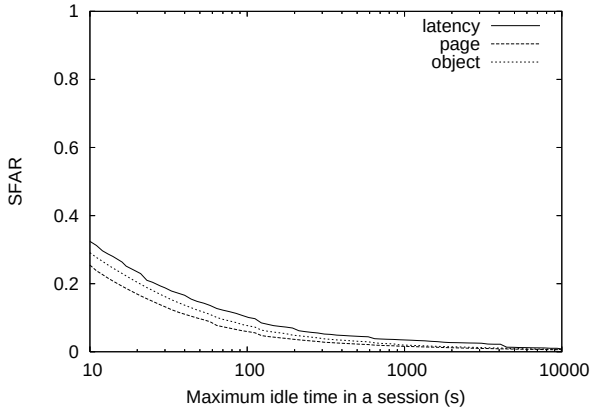


Figure 4.3: SFAR at the client as a function of the maximum idle time in a session

4.6 Predicting at the proxy

Locating the predictor at the proxy has similar advantages to locating it at the client, since both gather all the requests performed by the client. For this reason, their SFAR matches. However, the FSR has a better value in the proxy than in the client, due to the high amount of clients that are usually connected to the proxy. To analyze how the FSR changes depending on the amount of connected clients, experiments were run varying the number of users connected from 40 to 9,359, randomly selected from the original trace. As the results are highly dependent on the selected set of users that are connected to the scaled proxy, results for between 20 and 100 sets of users are averaged to achieve a 95% confidence interval less than 0.04.

Figure 4.4 shows that in a small sized proxy with 40 clients connected, 34% of user's accesses were made to an object never requested before. This FSR decreases logarithmically to 16% when considering all the users connected. It occurs in a similar way when looking at the latency curve, which decreases from 47% with a 40-client proxy to 33% with the 10,000-user proxy.

The values obtained for SFAR and FSR let us state that the first seen objects are the main limiting factor of prediction engines in the proxy in order to reduce the user-perceived latency.

4.7 Collaborative prediction

Previous subsections have shown that, depending on the location of the prediction engine, not only the potential latency reduction varies but also the source of this limitation. Therefore, if prediction engines were implemented in several elements of

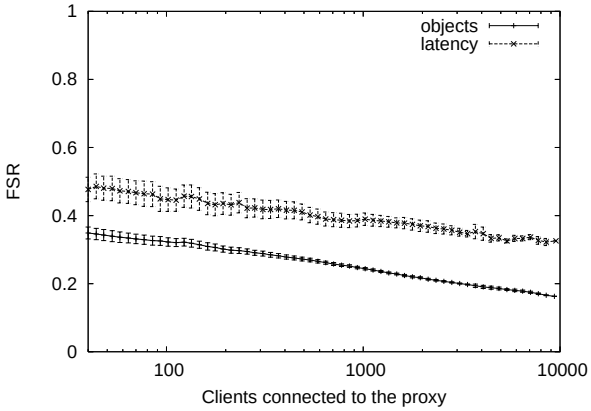


Figure 4.4: FSR at the proxy with 95% confidence intervals depending on the amount of clients connected to the proxy

the architecture, they could collaborate to improve the predictions and overcome the limitations.

In this subsection we quantify the latency reduction limits for the different combinations of collaborative prediction engines. Notice that when dealing with collaborative predictors, a client or a proxy can collaborate only with a server if it has been accessed before. To quantify this limitation, we define the *server first seen ratio* (SFSR) as the ratio of accesses to pages of servers that have been accessed for the first time to the total amount of accesses to pages.

4.7.1 Collaborative prediction between clients and servers

A collaborative prediction engine that uses predictors at the clients and at the servers takes benefit from the low SFAR of the client-located predictors and the good knowledge of the objects of server-located predictors. By analyzing the SFSR, we found that 5.0% of the pages accessed by the user are requested to a server that is accesed for the first time by that user. These requests represent 4.6% of the total user-perceived latency.

As discussed in Section 4.5, the latency that cannot be reduced due to the session first-accesses at clients is 2.8% of the perceived latency, whereas the FSR factor at the server limits the potential latency reduction between 90.7% and 98.3%. Since we are quantifying the theoretical maximum of the latency reduction, we consider the best situation (i.e., 98.3%).

All three limits provide a latency of 4.6% as a maximum value that cannot be reduced. Therefore, we conclude that a predictor engine based on collaborative predictors located at the clients and at the servers cannot reduce the user-perceived

latency more than 95.4%. Remark that this is only a theoretical limit of the capability of the prediction engine to potentially predict objects.

4.7.2 Collaborative prediction between proxy and servers

Like the previous scheme of collaboration, this one also takes benefit of the low SFAR of proxy prediction engines and the low FSR of predictors located at servers. The limits related to the session first-access factor and to the first seen objects are the same as the limits analyzed in the collaborative prediction between clients and servers, because the SFAR at the proxy matches the SFAR at the client (see Section 4.6).

The analysis of the trace provided a SFSR_{object} value of 0.8%, that is, only 0.8% of the pages requested by the proxy were made to a server that was not accessed before. These pages represented 0.7% of the latency recorded at the proxy. This result makes the session first-access factor the most restrictive limit to potentially reduce the user-perceived latency. Thus, a collaborative predictor engine located at the servers and at the proxy cannot reduce the user-perceived latency more than 97.2%.

4.7.3 Collaborative prediction between clients and proxy

A prediction engine located at the client could take benefit from collaborating with a predictor located at the proxy because the latter has a lower FSR. However, the opposite situation does not occur because the proxy predictor cannot benefit from the collaboration since SFAR matches at both elements. Therefore this collaborative scheme cannot outperform a single predictor located at the proxy.

For the same reason, a collaborative system between predictors in all the elements of the web architecture, i.e., clients, proxies and servers, cannot outperform the collaborative scheme between proxies and servers.

4.8 Conclusions

There are two previous works dealing with the limits of prefetching. The latency reduction limits of caching and prefetching are analyzed in [Kroeger 97] under several scenarios with a prefetching engine located at the proxy by using a proxy trace which dates from 1996. Most of their results are not comparable to the ones presented in this chapter because of the different assumptions. The only result that can be compared to those presented in this work is that, for a prediction engine located at the proxy, prefetching could reach a latency reduction of 45% with assumptions similar to those we considered. The comparison of this value to the one we found; i.e., 67%, illustrates that the changes of the Web during the last ten years have impacted noticeably on the potential benefits of web prefetching.

In [Fan 99], they also use a proxy-equivalent trace which dates from 1996 to evaluate their proposal of web prefetching between proxies and low-bandwidth clients. In their proposal, the proxy implements the predictor engine and then pushes the

Table 4.2: Summary of predicting limits depending on where the prediction engine is located

	Single Predictor			**Collaborative Predictors**	
Factor	Client	Proxy	Server	Client-Server	Proxy-Server
$SFAR_{object}$	1.3%	1.3%	45%	1.3%	1.3%
FSR_{object}	41.6%	16.3%	0.4%	0.4%	0.4%
$SFSR_{object}$	–	–	–	5.0%	0.8%
Max. object pred.	58.4%	83.7%	55%	95%	98.7%
$SFAR_{latency}$	2.8%	2.8%	63.4%	2.8%	2.8%
$FSR_{latency}$	45.6%	32.6%	1.7%	1.7%	1.7%
$SFSR_{latency}$	–	–	–	4.6%	0.7%
Max. lat. reduct.	54.4%	67.4%	36.6%	95.4%	97.2%

predicted objects to the clients when they are idle. In addition, the potential performance for their architecture is evaluated by means of a *perfect* prefetching algorithm. Their results show that, by combining several techniques (including prefetching), 50% of the requests can be predicted to avoid about 30% of the user-perceived latency.

Our study summarizes the potential prediction capabilities and the sources of the limits of each prefetching architecture (see Table 4.2). The results show that collaborative prediction engines could reduce almost all the user-perceived latency. These schemes largely improve the single predictor at the proxy, since a proxy-server collaborative prediction engine reaches a theoretical limit of reducing 97% of the user-perceived latency.

The best element to implement a single predictor is the proxy, which could reduce up to 67% of the perceived latency. The prediction engines located at servers can only reduce 36.6% the overall latency. This is due to the high amount of servers that are accessed only once per session. However, there is a noticeable amount of servers that can benefit from implementing the prediction engine in a higher extent.

In the remainder of this thesis, we will focus on this architecture because it is the one most studied in the open literature and the results obtained for a server-based predictor can be easily moved to the predictor located at the proxy. In addition, a predictor at the server is a compulsory step to implement any collaborative predictor.

Regarding the sources of the latency reduction limits, the session first-accesses are the main limiting factor in server predictors. The first seen objects represent the main limiting factor in client and proxy single predictors, as well as in the collaborative prediction engine of proxies with servers. Finally, accessing to servers never seen before is the most limiting factor to reduce the user-perceived latency in the collaborative predictors of clients with servers. In [Domènech 06h], a summary of the results of this chapter was presented.

Chapter 5

Evaluation of Current Prefetching Algorithms

5.1 Introduction

Despite the large amount of research works focusing on web prefetching, comparative and evaluation studies from the user's point of view are rare. This fact leads to the inability to quantify in a real working environment which proposal is better for the user.

Some papers comparing the performance of prefetching algorithms have been published [Dongshan 02, Chen 03, Nanopoulos 03, Bouras 03, Bouras 04, Wu 06]. These studies mainly concentrate on the performance of the predictor engine but, unfortunately, only [Bouras 04] attempts to evaluate the user-perceived latency. In spite of being an essential part of the prefetching techniques, performance measured at the prediction level does not completely explain the performance perceived by the user, as shown in Section 3.3, since the perceived latency is affected by a high amount of factors and not only by the hit ratio. In addition, performance comparisons are rarely made by means of a useful cost-benefit analysis, e.g., latency reduction as a function of the traffic increase.

In this chapter we use the methodology described in Chapter 3 to compare different prediction algorithms and to evaluate their performance by using current traces. That is, each algorithm is represented through a curve that relates the user-perceived latency reduction (by means of the *latency per page ratio* index) achieved by the algorithm to the cost of achieving such benefit, i.e., *traffic increase* or *object traffic increase*. A deep analysis through the predicted objects has also been carried out in order to identify the main reasons why performance among prediction algorithms differs.

5.2 Background

In this section we discuss recent works that deal with comparisons of web prefetching algorithms, focusing on the implemented algorithms and the metrics that they use to evaluate their performance.

Three versions of a predictor based on Markov are compared in [Dongshan 02]. They compare the accuracy, the model-building time, and the prediction time varying the order of the Markov model. In a similar way, [Chen 03] implements three variants of the PPM predictor. They show how the hit ratio and traffic increase vary as a function of the cutoff threshold of the algorithm.

In [Nanopoulos 03], it is shown a cost-benefit analysis of the performance of four prediction algorithms by comparing the precision and the recall to the traffic increase by means of a simple table. However, this analysis is only applied to a subset of the workload considered in the paper and ignores how the prediction performance affects the final user, i.e., the user-perceived latency.

The performance achieved by two configurations of the PPM algorithm and three of the n-most popular algorithm is compared in [Bouras 03]. For each one of the five configurations, they quantify the usefulness (recall), the hit ratio (precision) and the traffic increase. Both the number of experiments carried out and the metrics calculated make it difficult to identify which algorithm performs better. They only conclude that prefetching can be potentially beneficial to the considered architecture. In a more recent work [Bouras 04], they also show the latency reduction for the same experiments. Nevertheless, it is only an estimated latency calculated as the difference between the prediction time and the user request time. This value could be considered as an approximation of the latency reduction by the algorithm, since it does not include interactions between user and prefetching requests.

5.3 Experimental Environment

Workload Description The behavior pattern of users was taken from two different servers. Traces A and B correspond to the log used in the experiments of Section 3.3.3. The first one contains accesses to a news web server, whereas trace B has the accesses to a student information web server. The main characteristics of the traces are shown in Table 5.1. The training length of each trace has been adjusted to optimize the perceived latency reduction of the prefetching.

Prefetching Algorithms The experiments were run using the five algorithms evaluated in Section 3.5.2: four main variants of the *Prediction by Partial Match* (PPM) algorithm and the *Dependency Graph* (DG) based algorithm, both described in Section 2.3.1.1. The PPM variants are labeled as PPM-x-y, where x can be *OB* or *PG* depending on the element of prediction chosen (object or page respectively), and y can be *TOP* or *TH* depending on the way of selecting the candidates (the top-n most likely or using a threshold respectively).

Table 5.1: Trace characteristics

Characteristics	Trace	
	A	**B**
Year	2003	2003
Users	300	132
Page Accesses	2,263	1,646
Object Accesses	65,569	36,837
Training Accesses	35,000	5,000
Avg. Objects per Page	28.97	22.38
Bytes Transferred (MB)	218.09	142.49
Avg. Object Size (KB)	3.41	3.96
Avg. Page Size (KB)	98.68	88.64
Avg. HTML Size (KB)	32.77	28.21
Avg. Image Size (KB)	2.36	2.83
Avg. Page Download Time at 1 Mbps (s)	3.01	12.00
Avg. Page Download Time at 8 Mbps (s)	2.95	11.24

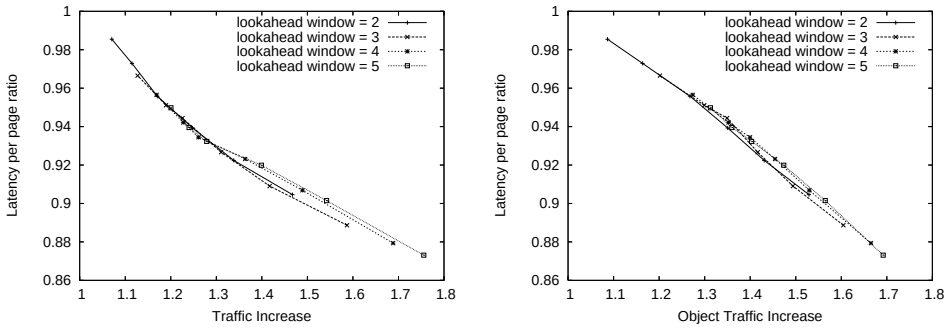
5.4 Selecting Algorithm Parameters

Each algorithm evaluated has a parameter that refers to the scope of the prediction. This parameter is the lookahead window size in the DG algorithm and the order in the PPM. To identify which parameter value achieves the best performance, a subset of possible values is explored by means of several experiments.

In this sense, we have simulated trace A with the DG and the PPM-OB-TH algorithms. As one can observe in Figure 5.1, the lookahead window in DG has been ranged from 2 to 5, whereas the maximum Markov chain order of the PPM has been ranged from 1 to 4, as shown in Figure 5.2. Each simulated user has a connection of 1 Mbps of available bandwidth. To build each curve, the cutoff threshold of both algorithms has been ranged from 0.2 to 0.7, increasing in steps of 0.1.

Results show that the lookahead window size in the DG algorithm hardly impacts on the performance, since all curves fall very close considering both cost factors, as shown in Figure 5.1. A deeper analysis of the results concludes that an increase in the lookahead window size has a similar effect to that of reducing the confidence threshold. This effect is shown by the fact that the 0.2 threshold point in the curves approaches the right side as the lookahead window size increases. Therefore, in the following experiments we have selected a window size of 2 since it is computationally more efficient.

In a similar way, an increase of the maximum order of the Markov model of the PPM predictor does not improve the performance. Figure 5.2(a) shows that curves



(a) Bytes

(b) Objects

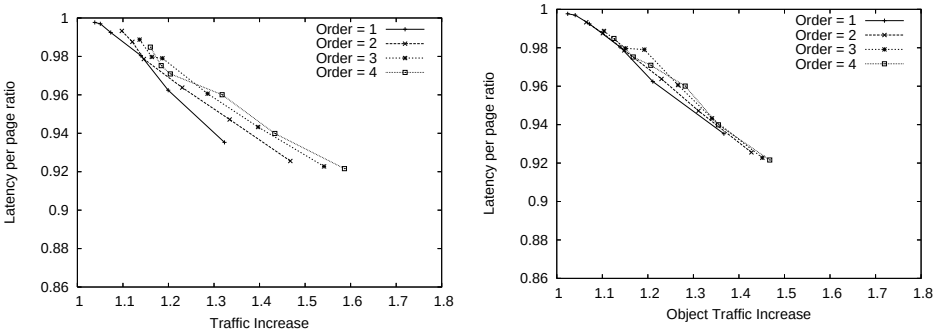
Figure 5.1: Lookahead window size parameter selection in DG algorithm. Users of 1 Mbps of available bandwidth with trace A are simulated. Each point in the curves represents a given threshold, from 0.2 (right) to 0.7 (left)

of higher order are further from the origin, which means that achieving the same reduction of latency requires extra traffic increase. When considering the object traffic increase (Figure 5.2(b)) the curves are closer among them, showing that similar performance is achieved through all the simulated orders. As a consequence, a Markov model of first-order will be used in the remaining experiments using the PPM algorithm. Similar results were obtained when trace B was used.

5.5 Comparison of Algorithms

The comparisons have been broken down into two groups, since we found that the results of the page-based algorithms (i.e., PPM-PG-TH and PPM-PG-TOP) show some odd behavior. Hence their separated analysis.

Figures 5.3 and 5.6 show the results of the performance comparison for the object-based algorithms, while results for the page-based algorithms are shown in Figure 5.8. Each algorithm is evaluated in four situations, as a result of combining two workloads (i.e., A and B) with two configurations of the user-available bandwidth (i.e., 1 Mbps and 8 Mbps). The curves of each plot in DG and PPM-x-TH algorithms are obtained by varying the confidence threshold of the algorithms, from 0.2 to 0.7 in steps of 0.1. To draw the curves of the PPM-x-TOP algorithms, the number of returned predictions are ranged from 1 to 9 in steps of 1, except for 6 and 8. Results for traffic increase and object traffic increase greater than 2 are not represented in order to keep the plot focused on the area where the algorithms can be compared.



(a) Bytes

(b) Objects

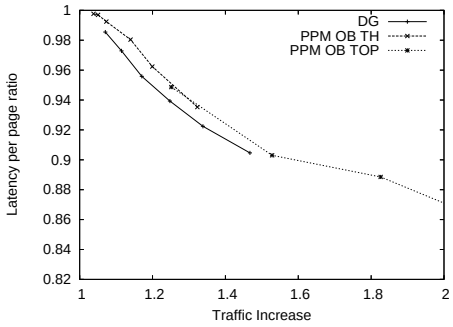
Figure 5.2: Selection of the maximum order of the Markov model in PPM-OB-TH algorithm. Users of 1 Mbps of available bandwidth with trace A are simulated. Each point in the curves represents a given threshold, from 0.2 (right) to 0.7 (left)

Figures 5.3(a) and 5.3(b) illustrate the performance evaluation of the algorithms simulating users who have 1 Mbps of available bandwidth and behave in accordance with the workload A. The first plot shows that, when considering traffic increase as the key cost, DG algorithm achieves better performance than the others in the range in which it is evaluated, since its curve falls always below the ones of the PPM algorithms. However, when the extra traffic is measured through the object traffic increase metric (Figure 5.3(b)), no significant performance differences can be found between the different algorithms.

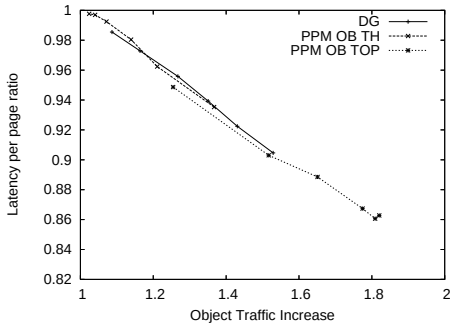
Curves of PPM-OB-TH and PPM-OB-TOP algorithms exhibit peculiar behavior in Figure 5.3(a). As one can observe, the PPM-OB-TOP curve seems to continue the curve of the PPM-OB-TH. This fact, which is also reflected in the other plots, means that selecting candidates by a cutoff threshold makes less predictions than selecting them by the top-n, although the overall performance is not considerably affected.

When considering 8 Mbps users, the DG algorithm also outperforms the others when the traffic increase is analyzed, as Figure 5.3(c) shows. However, Figure 5.3(d) reveals minor performance differences between algorithms in terms of object traffic increase.

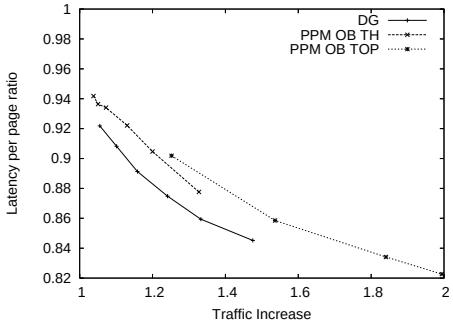
From the prediction perspective, the reason why the DG algorithm achieves better performance when considering the traffic increase lies in the fact that it predicts more user requests per each extra byte wasted by the prefetching; i.e., given a fixed recall value, DG generates less traffic increase than the other three algorithms. Plots 5.4(a) and 5.4(c) show this fact for both user's bandwidths.



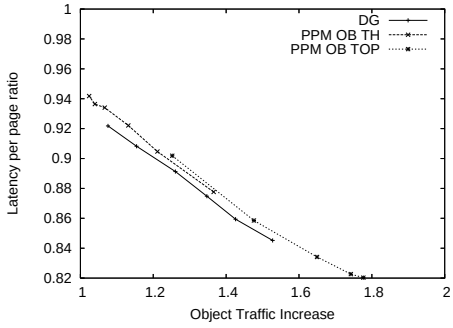
(a) 1 Mbps users, Bytes



(b) 1 Mbps users, Objects

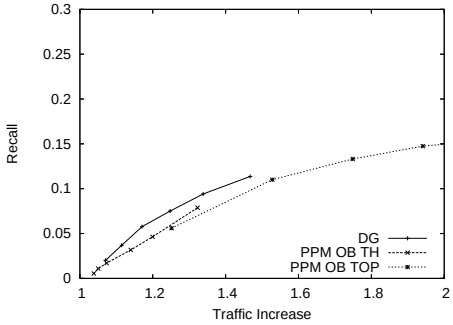


(c) 8 Mbps users, Bytes

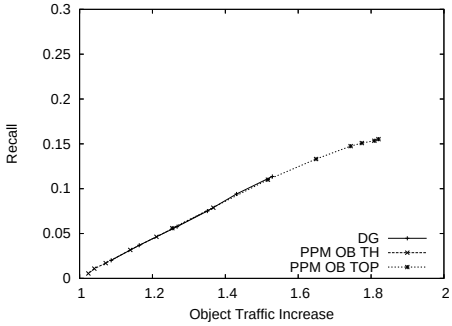


(d) 8 Mbps users, Objects

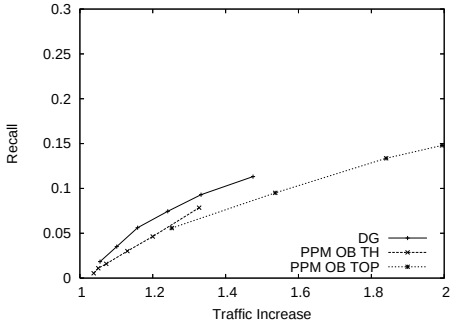
Figure 5.3: Performance comparison from the user's point of view between object-based algorithms running trace A. Each point in the curves represents a given threshold in PPM-OB-TH and DG, and a given number of returned hints in PPM-OB-TOP



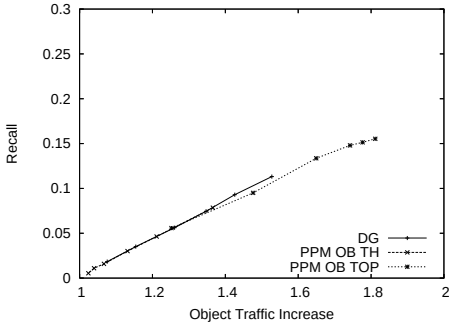
(a) Prefetching Recall versus Traffic Increase simulating 1 Mbps users



(b) Prefetching Recall versus Object Traffic Increase simulating 1 Mbps users



(c) Prefetching Recall versus Traffic Increase simulating 8 Mbps users



(d) Prefetching Recall versus Object Traffic Increase simulating 8 Mbps users

Figure 5.4: Performance comparison from the prediction point of view between object-based algorithms running trace A

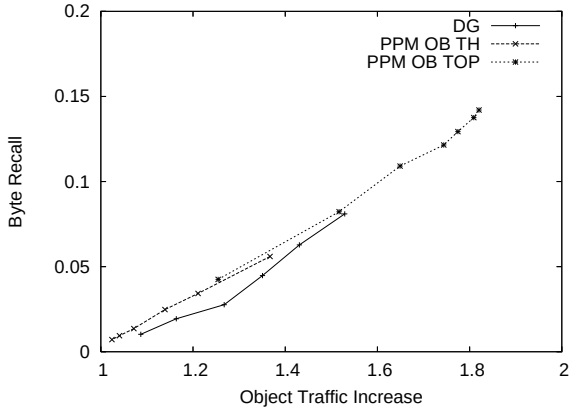
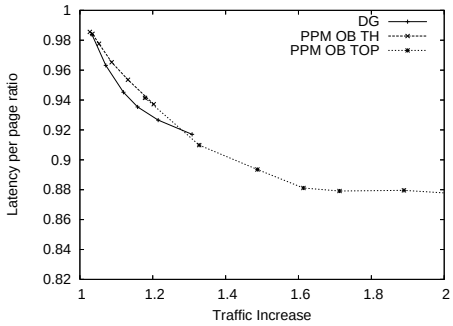


Figure 5.5: Byte Recall as a function of Object Traffic Increase. Users of 1 Mbps of available bandwidth under workload A are simulated

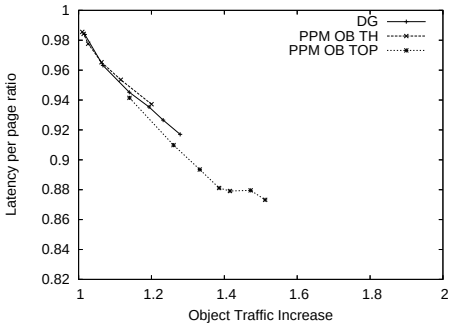
On the other hand, the reasons of the similar performance exhibited in the comparisons that consider the object traffic increase can be extracted from Figures 5.4(b) and 5.4(d). They show that given a fixed object traffic increase, the recall is almost the same in all three algorithms. This means that the algorithms predict a similar amount of user requests per each extra request to the server wasted by the prefetching system. Taking into account Equation 3.11, one can observe that all the algorithms have the same precision. This, together with the fact that DG requires less traffic increase than the other algorithms to reach the same recall value let us conclude that the objects prefetched by DG are, on average, smaller than those prefetched by the other algorithms. This result is also appreciated in Figure 5.5, where DG is the algorithm with less Byte Recall, i.e., given an object traffic increase, DG predicts the same amount of objects as the others algorithms (see Fig. 5.4(b)) but less bytes (see Fig. 5.5). The reasons why the size of the predicted objects differ among the different algorithms are explained in Section 5.6.

Figure 5.6 shows that the algorithms have less performance differences when using trace B for both user-available bandwidth. DG algorithm slightly outperforms the others when considering the wasted bytes as a cost in all its range, with the only exception of the most aggressive threshold (i.e., th=0.2), in which the PPM-OB-TOP algorithm achieves a slightly lower latency with the same traffic increase. When the cost analysis is focused on the amount of objects, plots show the same conclusion as when evaluating workload A: performance differences are negligible.

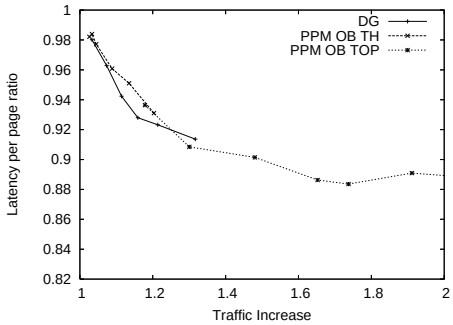
To complete this study, we also analyze results from the prediction perspective. Figures 5.7(a) and 5.7(c) reveal that the recall value of the DG algorithm is the highest when compared to the traffic increase, whereas there are no significant differences when compared to the object traffic increase (see Figures 5.7(b) and 5.7(d)). This



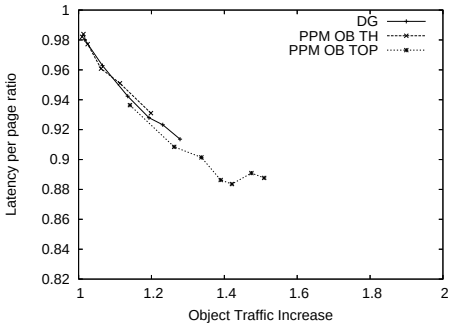
(a) 1 Mbps users, Bytes



(b) 1 Mbps users, Objects

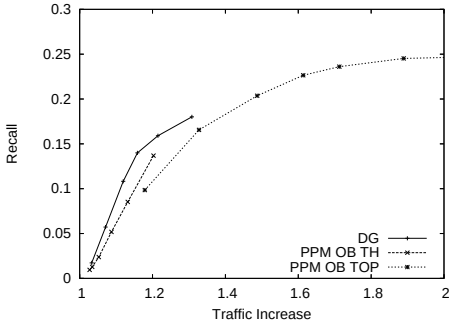


(c) 8 Mbps users, Bytes

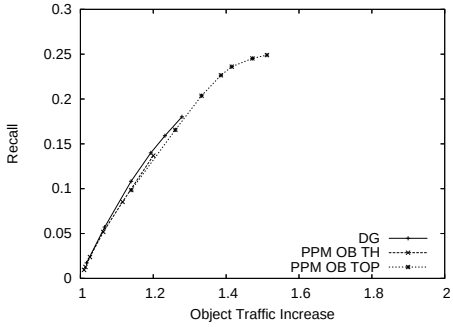


(d) 8 Mbps users, Objects

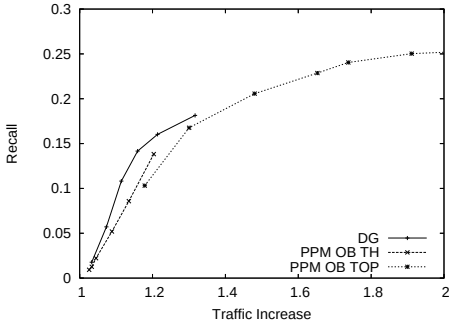
Figure 5.6: Performance comparison from the user's point of view between object-based algorithms running trace B



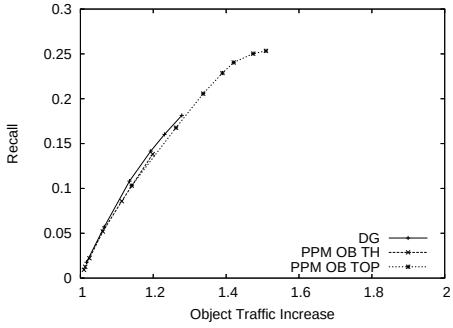
(a) Prefetching Recall versus Traffic Increase simulating 1 Mbps users



(b) Prefetching Recall versus Object Traffic Increase simulating 1 Mbps users

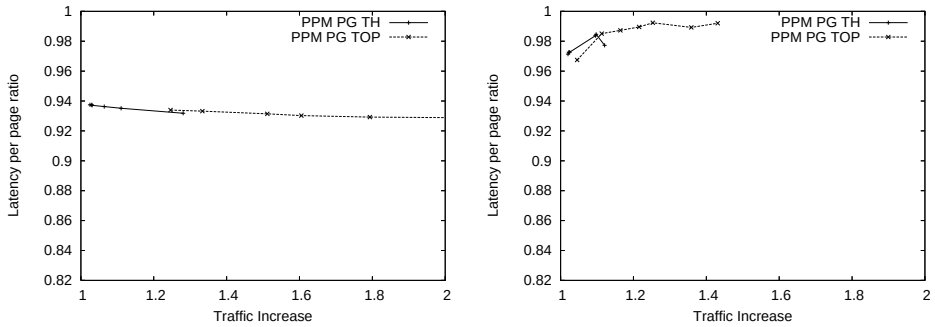


(c) Prefetching Recall versus Traffic Increase simulating 8 Mbps users



(d) Prefetching Recall versus Object Traffic Increase simulating 8 Mbps users

Figure 5.7: Performance comparison from the prediction point of view between object-based algorithms running trace B



(a) Workload A, Bytes

(b) Workload B, Bytes

Figure 5.8: Performance comparison between page-based algorithms with 8 Mbps users. Each point in the curves represents a given threshold in PPM-PG-TH and a given number of returned hints in PPM-PG-TOP

fact shows that the DG algorithm under workload B predicts smaller objects than the PPM algorithms, just like it occurs when workload A is used.

Figure 5.8 illustrates the performance of the page-based algorithms using both current traces and simulating users of 8 Mbps available bandwidth. Both plots in the figure refer to the latency versus traffic increase comparison and show almost horizontal curves. Figure 5.8(a) shows that a more aggressive policy will not reduce the perceived latency. Figure 5.8(b) manifests that increasing the traffic not only does not involve latency reduction but it also adversely impacts on the perceived latency. These results indicate that the page-based algorithms are not scalable, since they are only cost-effective when working in a non-aggressive manner, i.e., high confidence threshold (0.7 or 0.6) and few predictions returned (top-1 maximum). From the prediction point of view, the results are explained by the fact that recall hardly rises when increasing the aggressiveness, so dropping the precision.

As page-based algorithms predict only HTML files, the object traffic increase is negligible compared to the traffic increase in bytes of the same experiment. That is because HTML objects are, on average, much larger than others in the considered workload (see Table 5.1). In this sense, plots for comparing page-based algorithms using object traffic increase show almost all points near the left edge of the plot. These have not been included because all points concentrate in a small area. For instance, the run of the PPM-PG-TOP algorithm using a top-1 under workload A results in a traffic increase of 25% while the object traffic increase is just 1.3%. Despite the small value of the object traffic increase, the latency is not reduced with the extra

aggressiveness, so neither are the algorithms scalable when the cost analysis focuses on the object traffic increase.

5.6 Analysis of the Algorithms

To provide insights on the understanding of why DG predicts smaller objects than PPM, we analyzed carefully the requests contained in the traces. We found that the main reason why algorithms predict different sized objects lies in the fact that DG predicts image objects more likely than HTML files, which are larger (see Table 5.1).

For illustrative purposes, in this section we use an hypothetical example. Let's suppose that the algorithms are trained by two user sessions. The first one contains the following accesses: HTML1, IMG1, HTML2, IMG2. The second session includes the accesses: HTML1,IMG1, HTML3, IMG2. Note that IMG2 is embedded both in HTML2 and in HTML3. We found this fact to be common along the analyzed workloads, especially in workload A, where different pieces of news (i.e., HTML files) contain the same embedded images, since they are included in the site structure. Figure 5.9 shows the state of the graph of the current configuration of the DG algorithm after the aforementioned training. Each node in the graph represents an object, whereas the weight of each arc is the confidence level of the transition. The state of the PPM algorithm after the same training is illustrated in Figure 5.10. Each node represents a context, where the root node is in the first row, the order-0 context is in the second, and the order-1 context is in the third. The label of each node includes the counter of times that a context has appeared, so one can obtain the confidence of a transition by dividing the counter of a node by the counter of its parent. The arcs indicate the possible transitions. For instance, the label of the IMG2 in order-0 context is 2 because IMG2 appeared twice in the training. As it appeared once after HTML2 and another after HTML3, IMG2 has two nodes in the order-1 context, one per each HTML on which it depends.

As one can observe in Figure 5.9, the DG algorithm can predict the access to IMG2 after accessing the first page, i.e., HTML1 and IMG1. However, Figure 5.10 shows that PPM can only predict IMG2 when the user has requested HTML2 or HTML3, but at this time the prediction is useless since there is no time to prefetch the object. For this reason, it is more likely that DG algorithm predicts embedded objects of the next page than the PPM, which would predict only HTML files after the IMG1 object.

5.7 Conclusions

In this chapter we have applied the cost-benefit methodology presented in Section 3.4 to evaluate and compare prefetching algorithms from the user's perspective. This methodology has been also used to select the most effective algorithm parameters,

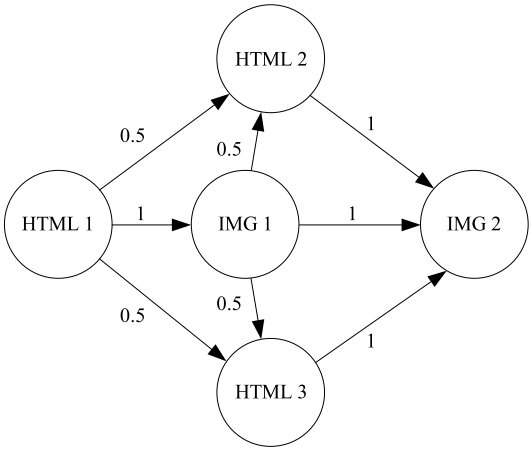


Figure 5.9: State of the graph of the DG algorithm with a lookahead window size of 2 after the accesses HTML1, IMG1, HTML2, IMG2 by one user; and HTML1, IMG1, HTML3, IMG2 by other user

concluding that a higher complexity of the algorithm involves more aggressiveness but not better performance.

Five prediction algorithms have been implemented and compared. Experimental results show that DG algorithm slightly outperforms the PPM-OB-TH and the PPM-OB-TOP algorithms in most of the studied cases when considering the traffic increase as the main cost. However, the aggressiveness (and, consequently, the latency reduction) of the DG is more limited than the PPM-OB-TOP one. For this reason, when prefetching is not desired to be very aggressive due to bandwidth or computational restrictions, DG achieves the best cost-effectiveness. However, for more aggressive policies, a larger DG *lookahead window* or the PPM-OB-TOP algorithm should be used.

We have analyzed why the DG algorithms achieve better performance, finding that it is because DG predicts, on average, smaller objects than the PPM-based algorithms. This result has been intuitively explained by understanding how the algorithms work.

The comparison of the algorithms when the cost analysis focuses on the object traffic increase shows that the differences between the object-based algorithms are negligible. Results also show that the PPM-OB-TOP algorithm is a more aggressive version of the PPM-OB-TH, but it offers similar cost-benefit ratio. Nevertheless, it is more difficult to adapt PPM-OB-TOP to the traffic increase restrictions since the number of returned hints is a discrete number whereas the threshold is continuous.

Page-based algorithms have shown some odd behavior, since extra aggressiveness involves extra traffic but the same or less latency reduction. This fact means that the most likely predicted pages reduce the perceived latency, but the others not only

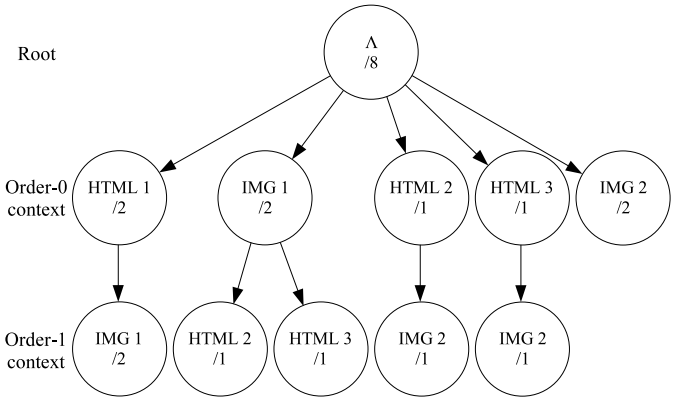


Figure 5.10: State of the graph of a first-order PPM algorithm after the accesses HTML1, IMG1, HTML2, IMG2 by one user; and HTML1, IMG1, HTML3, IMG2 by other user

do not reduce the latency but also they can degrade the performance. In this sense, these algorithms cannot modify their aggressiveness since they must work with a high confidence to be cost-effective.

In [Domènech 06f, Domènech 06e, Domènech 06g], a summary of the results shown in this section was presented.

Chapter 6

The Influence of the Environment Conditions on the Design of Prefetching Algorithms

6.1 Introduction

The difficulty to evaluate the performance of prefetching considering different environment conditions is an important obstacle for the design and improvement of prediction algorithms. As the main goal of prefetching is to reduce the user-perceived latency, the prediction algorithm should try to minimize it. However, latency-related indexes are not easy to calculate due to the complexity of the environment. As a consequence, prediction indexes are used instead. Prediction algorithms are usually designed to maximize the recall, therefore this is the most widely used metric to compare the algorithms performance [Schechter 98, Markatos 98, Albrecht 99, Palpanas 99, Nanopoulos 03]. In Section 3.3 we introduced the byte recall metric and showed that there are situations in which it quantifies the performance better than the classical recall since it is higher correlated to the main latency-related index: the latency per page ratio. In this context, algorithms designers have now the useful option to select, depending on the environment conditions, the appropriate index to optimize their proposal: recall or byte recall.

In this chapter we perform a rigorous statistical analysis to identify those situations in which it is better to optimize the byte recall instead of the classical recall and vice versa. If those situations are identified, the performance of prefetching algorithms could be improved by adapting them dynamically to optimize the best of both

indexes according to the situation. To this end, we select the analysis of variance method (ANOVA) to study the effect of those parameters (e.g., bandwidth and connection establishment time) that may have greater impact on the correlation between the latency per page ratio and the recall or the byte recall. In order to extend the conclusions to a wider range of situations, three different parameters values are selected according to current technologies, two prefetching algorithms are implemented, and two different workloads are taken into account. In addition, different configuration parameters values having impact on the prediction power for each algorithm are considered.

Our experimental results show that the recall index gains relative importance against the byte recall as higher the user available bandwidth is, while the byte recall gains relative importance against the recall as smaller the server processing time is.

6.2 Background

There is an important set of research works concentrating on prefetching techniques to reduce the user-perceived latency. Some of the studies presented in the open literature check their proposals only in a given and restricted scenario or conditions. Some more ambitious works test the performance of their proposal in different scenarios; e.g., users connecting through a modem or through broadband, loaded or unloaded server, and so on. Below we describe some prefetching research works in order to illustrate a representative subset of typical scenarios.

Fan *et al.* [Fan 99] study how the perceived latency can be reduced for low-bandwidth users by using compression and prefetching between clients and proxies. Fleming *et al.* [Fleming 97] also combine several techniques (e.g., prefetching, document adapting) to reduce the perceived latencies for wirelessly connected users. They check their technique using several bandwidths and both wire and wireless connections. Bouras *et al.* [Bouras 04] compare the performance of prefetching using two prediction algorithms over a server-proxy scheme (i.e., the proxy downloads objects from the server in advance). The studied network architecture includes high bandwidth users accessing to the Internet through a proxy. Kokku *et al.* [Kokku 03] propose a JavaScript prefetch implementation using the PPM algorithm as a predictor. Their proposal is verified against an unloaded system, a loaded server and a loaded network. The user bandwidth restrictions are applied considering all users connected to a LAN. Padmanabhan and Mogul [Padmanabhan 96] apply a file system prefetching algorithm to the web prefetching in order to reduce the perceived latency. They take into account two different scenarios: a 28.8 Kbps bandwidth and 1.5 seconds of connection establishment plus processing time, and a 149 Kbps bandwidth and 1.13 seconds of connection establishment plus processing time.

Although some proposals are explored in several scenarios, it is neither analyzed nor identified the cause why a given scenario can benefit a given algorithm. Note that this is the first step in order to suggest guidelines which permit to adapt the algorithm to the different situations.

6.3 Performance Indexes and Related Factors

The performance indexes analyzed in this chapter are the latency per page ratio, the recall and the byte recall, defined in Section 3.3. As experimental results will show, the correlation between the latency perceived by the client and the recall (or the byte recall) is not perfect. That is because the perceived latency depends not only on the amount of downloaded objects (or bytes) but also on those external factors that can impact on the download time. Those factors are mainly related to the user connection or the characteristics of the navigation itself. In Section 6.6 we study how those factors impact on the aforementioned correlation. The four factors selected for our study are: the user available bandwidth, the connection establishment time, the server processing time and the byte position of the first reference to an embedded object.

We select the bandwidth available for the user since the wider the bandwidth is, the less the object transfer time is. The connection establishment time is strongly related to the startup time of a user navigation session. The server processing time can speed up or slow down the startup time of each requested object.

While the impact of the three aforementioned indexes on the perceived latency is quite straightforward, the byte position is the less clear. The HTTP standard permits up to two simultaneous connections for a client to each server. Let's assume that the user has one active navigation session, and let's suppose that a container object (HTML) is requested so that it uses only one of the two available connections. The other connection is only used when the first reference to an embedded object is read from the HTML. Therefore, the position of this byte impacts on the startup time of downloading a web page.

6.4 Statistical Methodology

In this section we summarize the main characteristics of the analysis of variance method or ANOVA and the reduced experiment design that will be used to quantify the impact of the four factors described above in the correlation between indexes. Details on these methods can be found in [Taguchi 87].

6.4.1 Analysis of Variance (ANOVA)

The ANOVA method permits to identify which factors significantly affect the average value of a given variable when varying the values of a set of them. In our study, we select as factors the ones described above and as variable a new metric (SCD) that identifies which is the best index to optimize, as it will be described below. The idea behind this method is to split the total variance observed in the performed experiments among the studied effects. These effects can be associated to a factor nature (e.g., linear or quadratic) or to any factor interaction.

To perform the ANOVA, the data are usually represented in a table, where each row shows those values related to an effect – or a set of effects. The variability due to each factor is quantified by a sum of squares, and the corresponding values are specified in a particular column. The variability that is not due to any of the studied factors is gathered by a table row labeled as *residual*. The significance of an effect, with the desired level of confidence, is checked against the statistical F test.

Once the most relevant factors are identified and isolated, it is possible to quantify the direction and the degree that each effect has on the studied variable. From these data one can predict how the variable will behave for every factor value (e.g., 512 Kbps of user available bandwidth and 50 ms of connection establishment time), which is the purpose of this chapter.

6.4.2 Design of Experiments

The design of experiments is of paramount importance in order to guarantee the results validness of the analysis of variance. The ANOVA requires that the studied effects are orthogonal to each other, in order not to confuse effects.

The full factorial design is the simplest one that keeps all the possible effects orthogonal (related to factors or to interactions), but the amount of performed experiments rises at exponential rate since every combination of factor levels (the values tested for each factor) is explored once.

In general, high order interactions between factors are rarely significant. Therefore, to reduce the amount of experiments to be performed, their effects are not usually studied. In this context, [Taguchi 87] proposes several *orthogonal arrays* that minimize the number of experiments in a such way that the effects of the main factors remain orthogonal. In this paper we use the L_{18} orthogonal array, which explores up to 8 factors in 18 experiments.

6.5 Experimental Environment

Prefetching Algorithms To check the behavior of the indexes, the experiments were run using two different prediction algorithms: one variant of the *Prediction by Partial Match* (PPM) algorithm and the *Dependency Graph* (DG) based algorithm. To run the experiments presented in this chapter, from those variants of the PPM algorithm presented in Section 2.3.1.1, the algorithm has been applied to objects instead of pages, and candidates has been selected by means of a confidence threshold. This is the configuration that performs better in the studied conditions (see Chapter 5). Both algorithms have a parameter related to the number of future accesses that each one is able to predict. This parameter is the *lookahead window size* in the DG algorithm, and the *order* in the PPM. In addition, the *threshold* parameter of both algorithms refers to the minimum probability that a candidate object must exhibit to be hinted as a successor. As one can observe, the aggressiveness and the precision of the prefetch are sensitive to the described parameters; i.e., lookahead window size and

Table 6.1: Trace characteristics

Trace	A	B
Training accesses	251,271	148,213
Experiment accesses	65,866	37,655
HTMLs accesses	2,269	1,717
Users	300	132
Bytes Transferred (MB)	188,600	68,428

threshold for DG, and order and threshold for PPM. Therefore, in order to analyze the correlation in a wide range of scenarios, the corresponding parameters are ranged for each algorithm.

Workload Description The behavior pattern of users was taken from two different servers. Traces A and B correspond to the log used in the experiments of Section 3.3.3. The first one contains accesses to a news web server, whereas trace B has the accesses to a student information web server. The main characteristics of the traces can be found in Table 6.1.

6.6 Experimental Results

6.6.1 Experiments Design

In this section we analyze the impact of the four factors – discussed in Section 6.3 – on the indexes correlation. Three different values were selected for each factor according to some current technology. Table 6.2 shows these values broken down in columns called *levels* (in the statistical nomenclature). This table also includes some parameters related to the implemented prefetching algorithms (i.e., the order or window size, depending on the algorithm, and the threshold), and their corresponding values.

In addition to the values shown in Table 6.2, the two traces described above must be also taken into account for the experiment design. This gives us an amount of 2,916 ($2^2 \cdot 3^6$) possible combinations. In other words, 2,916 experiments will be needed to explore all the cases, which suppose a huge amount of simulation time. By using the Taguchi's Robust Design, a more efficient design was selected.

Among the orthogonal arrays proposed by Taguchi, we selected the L_{18} array to fit the desired factors (this method can support up to one two-level factor and seven three-level factors; i.e., $2 \cdot 3^7$), hence the fact that only 18 experiments need to be run. For each experiment, the correlation coefficient between the latency per page ratio

Table 6.2: Studied factors broken down into those really analyzed and those used to extend the conclusions to a wider range of prefetching situations

Category	Factors	Level 1	Level 2	Level 3
Analyzed Factors	User Bandwidth (Kbits/s)	50	300	550
	Connection estab. time (ms)	25	100	175
	Server Processing Time (ms)	100	300	500
	Byte of the first reference[1]	200	1,200	2,200
Prefetch Factors	Prefetching Algorithm	DG	PPM	—
	Order /lookahead window	2	5	8
	Confidence threshold	0.2	0.4	0.6

and both the recall and byte recall is calculated, taking into account data for each client in the experiment.

6.6.2 Correlation Analysis

Table 6.3 shows the entire experiments design and the results for each one, where each row represents one experiment. The eight columns corresponding to the factors level show the level (as labeled in Table 6.2) taken for each factor in each experiment, following the rules proposed by the Taguchi's L_{18} orthogonal array [Taguchi 87]. The correlation coefficient columns show the Pearson's coefficient correlation between the Latency per page ratio (L_p) and the Recall (Rc) and Byte Recall (Rc_B) indexes in each experiment.

The last column (SCD) refers to the difference between the square of the two previously calculated correlation coefficients. Each coefficient is powered to 2 to overweigh the differences of strongest correlations against weaker correlation differences. Note that a positive value means that the latency per page ratio is more correlated to the recall than to the byte recall, while a negative value means the opposite. The resulting SCD value is taken as input for the analysis of the variance method to find the most significant factors.

From the previous table we perform the first analysis of the variance (ANOVA), shown in Table 6.4. This analysis includes the linear and quadratic effects (for those factors having 3 levels) of the parameters. The null hypothesis (H_0), for each factor, means that the change of its level does not significantly affect the average of the observed variable, i.e., SCD. In such a case, the sample average of all the studied levels will be the same. The *Sum of Sq.* measures the variability of the SCD, while the *Mean Sq.* is the *Sum of Sq.* averaged by the degrees of freedom of each source of variation. The residual mean squares is demonstrated to be an estimation of the variance of studied populations. If the H_0 of a factor is true, the *Mean Sq.* of this factor is also an estimator of the variance independent from the previous one. In such

Table 6.3: Correlation coefficients and SCD value for each performed experiment. Legend. Tr: Trace, PA: Prefetch algorithm, FR: Byte position of the first reference. WS: Algorithm order (PPM) / Lookahead window size (DG). TH: Threshold. PT: Server processing time. ET: Connection Establishment time. BW: User-available bandwidth

Factors Level								***Correlation coef.***		
Tr	**PA**	**FR**	**WS**	**TH**	**PT**	**ET**	**BW**	**L_p, Rc**	**L_p, Rc_B**	**SCD**
A	1	1	1	1	1	1	1	-0.5636	-0.6054	-0.0489
A	1	2	2	2	2	2	2	-0.5893	-0.6957	-0.1368
A	1	3	3	3	3	3	3	-0.7064	-0.6943	0.0170
A	1	1	1	2	2	3	3	-0.7645	-0.8127	-0.0760
A	1	2	2	3	3	1	1	-0.4998	-0.6906	-0.2272
A	1	3	3	1	1	2	2	-0.7087	-0.8295	-0.1858
A	2	1	2	1	3	2	3	-0.9063	-0.8444	0.1083
A	2	2	3	2	1	3	1	-0.6095	-0.8480	-0.3476
A	2	3	1	3	2	1	2	-0.7724	-0.8593	-0.1418
B	1	1	3	3	2	2	1	-0.7370	-0.8177	-0.1255
B	1	2	1	1	3	3	2	-0.5762	-0.5999	-0.0278
B	1	3	2	2	1	1	3	-0.6565	-0.7398	-0.1164
B	1	1	2	3	1	3	2	-0.2428	-0.3811	-0.0862
B	1	2	3	1	2	1	3	-0.6728	-0.5363	0.1650
B	1	3	1	2	3	2	1	-0.3065	-0.2777	0.0168
B	2	1	3	2	3	1	2	-0.5801	-0.2854	0.2551
B	2	2	1	3	1	2	3	-0.3050	-0.0768	0.0871
B	2	3	2	1	2	3	1	-0.4268	-0.5478	-0.1180

Table 6.4: First ANOVA table for the final design. Legend. PA stands for prefetch algorithm, FR: refers to the byte position of the first reference. WS: refers to algorithm order (PPM) / lookahead window size (DG). TH: Threshold. PT: Server processing time. ET: Connection establishment time. BW: user available bandwidth

Src of Variation	**Sum of Sq.**	**DF**	**Mean Sq.**	**F-Ratio**	**F**
Main effects	0.3092	14	0.191	10.620	8.715
Trace	0.0658	1	0.066	3.657	10.128
PA	0.0075	1	0.007	0.414	10.128
FR	0.0319	2	0.016	0.885	9.552
Linear	0.0257	1	0.026	1.426	10.128
Quadratic	0.0062	1	0.006	0.345	10.128
WS	0.0153	2	0.008	0.425	9.552
Linear	0.0001	1	0.000	0.005	10.128
Quadratic	0.0152	1	0.015	0.845	10.128
TH	0.0128	2	0.006	0.355	9.552
Linear	0.0114	1	0.011	0.632	10.128
Quadratic	0.0014	1	0.001	0.079	10.128
PT	0.0615	2	0.031	1.707	9.552
Linear	0.0588	1	0.059	3.265	10.128
Quadratic	0.0027	1	0.003	0.149	10.128
ET	0.0251	2	0.013	0.697	9.552
Linear	0.0229	1	0.023	1.273	10.128
Quadratic	0.0022	1	0.002	0.122	10.128
BW	0.0893	2	0.045	2.480	9.552
Linear	0.0893	1	0.089	4.960	10.128
Quadratic	0.0000	1	0.000	0.001	10.128
Residual	0.0540	3	0.018		
Total	0.3632	17			

Table 6.5: Second ANOVA table for the final design. Legend. PA: Prefetch algorithm, FR: Byte position of the first reference. WS: Algorithm order (PPM) / Lookahead window size (DG). TH: Threshold. PT: Server processing time. ET: Connection establishment time. BW: User-available bandwidth

Src of Variation		Sum of Sq.	DF	Mean Sq.	F-Ratio	F
Main effects		0.2892	8	0.294	35.773	3.230
Trace		0.0658	1	0.066	8.009	5.117
FR	Linear	0.0257	1	0.026	3.123	5.117
WS		0.0153	2	0.008	0.930	4.256
	Linear	0.0001	1	0.000	0.010	5.117
	Quadratic	0.0152	1	0.015	1.850	5.117
TH	Linear	0.0114	1	0.011	1.383	5.117
PT	Linear	0.0588	1	0.059	7.153	5.117
ET	Linear	0.0229	1	0.023	2.787	5.117
BW	Linear	0.0893	1	0.089	10.865	5.117
Residual		0.0740	9	0.008		
Total		0.3632	17			

a case, *F-ratio*, defined as the ratio of both estimators, is distributed as F-distribution with the degrees of freedom of each estimator. For example, the F-ratio of PT factor in Table 6.4 is distributed as F with 2 and 3 degrees of freedom. Otherwise, if H_0 of a factor is not true (i.e., the change of the factor level significantly affects the average SCD), then the F-ratio value of this factor will be higher than F value, given a confidence level. We considered a confidence level of 95%, which is the value shown in the column labeled as F in the table.

As the design of experiments leave few residual degrees of freedom, this first ANOVA cannot be used to find what factors are significant. In this analysis, we identified the less significant factors and remove them from the ANOVA in order to increase the residual degrees of freedom. In this sense, *prefetching algorithm* (PA), and the quadratic effects of *byte of the first reference*, *threshold* (TH), *server processing time* (PT), *connection establishment time* (ET) and *user available bandwidth* (BW) have been removed since all of them have a small sum of squares.

The second analysis of the variance permits to identify the most significant factors. Table 6.5 presents the results of the second ANOVA. As one can observe, the three significant factors (i.e., those with F-ratio higher than F with 95% of confidence level) are the linear effect of the *user available bandwidth*, the *trace* used and the linear effect of the *server processing time* for each request.

Table 6.6 quantifies the effect and the direction that each of the three significant factors has on the average SCD. The *Level Average* column shows the average value

Table 6.6: Summary of the effect that each significant factor has on the average SCD

Factor	**Level**	**Level Average**	**Effect Estimate**
User Bandwidth	50	-0.1417	-0.0868
	300	-0.0539	0.0010
	550	0.0308	0.0858
Trace	A	-0.1154	-0.0605
	B	0.0056	0.0605
Server processing time	100	-0.1163	-0.0614
	300	-0.0722	-0.0172
	500	0.0237	0.0786

of the SCD for a given factor value. The *Effect Estimate* column is the difference between the level average and the global average (-0.0549).

On the one hand, considering the user-available bandwidth factor, it can be observed that when the bandwidth is higher, the latency per page ratio is strongly correlated to the recall rather than to the byte recall. On average, the SCD raises 0.0345 per each extra 100 Kbps of user available bandwidth.

On the other hand, for the server processing time factor, one can observe that when the server processing time for each request is smaller, the latency per page ratio is more strongly correlated to the byte recall than to the recall. On average, the SCD decreases 0.0350 per each 100 ms of server processing time reduction.

Finally, the trace factor values must be observed independently since the trace is a qualitative factor. As one can observe, the average SCD differs significantly between both traces. These large differences are mainly caused by the characteristics of each web design. For instance, as previously showed in Table 5.1, the average amount of embedded objects per page in trace A is higher than those in trace B (29 vs. 21.9 objects). Besides, each object in trace A is on average 57% times larger (2.86 vs. 1.82 KB) than an object in the B, resulting in a 108% times larger full page weight (82.9 vs. 39.8 KB).

6.7 Conclusions

In this chapter we analyzed in detail under which environment conditions the byte recall is more correlated to the perceived latency than the recall permitting to optimize the prefetch algorithm.

To identify these situations we have analyzed the main environment parameters (i.e., the user available bandwidth, the connection establishment time, the server

[1] In a HTML file, it is the byte position of the first reference to an embedded object.

processing time and the byte position of the first reference to an embedded object) in combination with other factors affecting the prefetching performance, and two different workloads.

Our experimental results show that the most significant parameters affecting the selection of the appropriate index are the user-available bandwidth and the server processing time. Although values of the bandwidth and processing time for selecting the best index depend on the workload, the significance and the direction of each one do not. The user-available bandwidth affects this selection in the sense that when the available bandwidth increases, the recall is relatively more correlated to the perceived latency, whereas when the processing time decreases, the byte recall is more relatively correlated to the perceived latency. Results suggest that, in these environment conditions, the index to optimize should be the byte recall instead of the classical recall. In this way, prediction probabilities could be calculated taking into account the amount of bytes instead of the traditional amount of objects.

These results take special relevance for current Internet infrastructure because we can find a large variety of ways of accessing the Internet, each one with different bandwidths and latencies, e.g., DSL, modem, cable, satellite, GPRS; as well as different web server loads and processing times.

In [Domènech 06i, Domènech 06d], a summary of the results shown in this section was presented.

Chapter 7

DDG: A Prefetching Algorithm for Current Web

7.1 Introduction

The prediction of users' accesses has been usually made by adapting prediction algorithms from other fields of computing to deal with web accesses. For instance, the DG algorithm, which was firstly used in web prefetching by [Padmanabhan 96], lies in a predictor of accesses to a local file system [Griffioen 94]. PPM algorithm, mainly used for lossless data compression [Bell 90, Curewitz 93], has been also used to predict web accesses by several authors [Fan 99, Palpanas 99, Bouras 04, Chen 03]. With these algorithms, web prefetching achieved an acceptable performance when they were proposed. However, the Web has changed noticeably during the last decade. There are two main differences between old and current Web generations: the increasing dynamism and customization of the content [Peña-Ortiz 05], and the increase in the complexity of the web site design. Despite this fact, prefetching algorithms taking into account the new Web sites characteristics have not been proposed.

We found in Section 3.5 that there is a noticeable increase in the amount of embedded objects per page. In this context, most of the predictions made by the existing algorithms are useless to reduce the user-perceived latency because these algorithms do not take into account how web pages are structured, i.e., an HTML object with several embedded objects. Thus, they predict the accesses to the embedded objects of an HTML after reading the HTML itself. In this chapter we present a fair performance evaluation study that shows that these predictions are useless since the client already knows that the following objects to be demanded are the embedded objects. However, these objects are not still demanded because the amount of simultaneous connections to a web server from the same client is limited, as suggested in the standard HTTP/1.1 [Fielding 99] and implemented in the most commonly used

web browsers (i.e., Internet Explorer and Mozilla Firefox). Therefore, there is neither time to prefetch nor perceived latency to reduce.

In this context, this chapter presents the DDG prediction algorithm that considers the characteristics of the current web sites to improve the performance achieved by web prefetching. It is based on the Dependency Graph (DG) algorithm, but it differentiates two classes of dependences: between objects of the same page and between objects of different pages. Performance evaluation results show that the latency reduction can be dramatically increased and the need of resources dropped, i.e., extra bandwidth and extra server load.

7.2 Experimental Environment

Workload Description The behavior pattern of users was taken from two different servers. Trace A corresponds to the log used in the experiments of Section 3.3.3, and it contains accesses to a news web server. Trace B includes users accessing the institutional web site of the Computer Science School of the Polytechnic University of Valencia. This trace was collected during February 2006 from its Apache web server log.

The main characteristics of the traces are shown in Table 7.1. The training length of each trace has been adjusted to optimize the perceived latency reduction of the prefetching. The simulation of the workload A considering that each user has 1 Mbps of available bandwidth results in an average latency per page of 3.01 seconds. By increasing the user available bandwidth to 8 Mbps it is only possible to reduce the perceived latency by about 2%. However, by employing prefetching techniques the user-perceived latency can be reduced in a higher extent with a reasonable cost, as experimental results show below. The simulation of 1 Mbps users under workload B results in an average latency per page of 0.87 seconds. The increase of the user available bandwidth to 8 Mbps only achieves the reduction of user-perceived latency by about 1%.

Prefetching Algorithms The DDG algorithm is compared in the experiments to two of the most widely used prediction algorithms in the literature: one variant of the *Prediction by Partial Match* (PPM) algorithm and the *Dependency Graph* (DG) based algorithm. To run the experiments presented in this chapter, from those variants of the PPM algorithm presented in Section 2.3.1.1, the algorithm has been applied to objects instead of to pages, and candidates are selected by means of a confidence threshold. This is the configuration that performs better in the studied conditions (see Chapter 5).

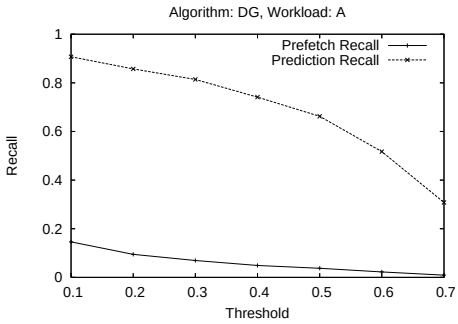
Table 7.1: Trace characteristics

Characteristics	Trace	
	A	**B**
Users	300	1,379
Page Accesses	2,263	10,282
Object Accesses	65,569	43,104
Training length (accesses)	35,000	23,104
Objects per Page	28.97	4.19
Bytes Transferred (MB)	218.09	386.05
Avg. Object Size (KB)	3.41	9.17
Avg. Page Size (KB)	98.68	38.44
Avg. HTML Size (KB)	32.77	12.94
Avg. Image Size (KB)	2.36	7.99

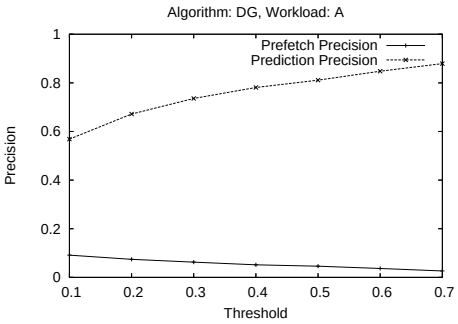
7.3 Uselessness Analysis of the Existing Algorithms

Figures 7.1 and 7.2 show the values of predictions metrics measured both at the prediction engine and at the prefetching engine under the workloads A and B respectively. By analyzing the plots, we found that the prediction performance from the predictor point of view is quite good. However, this good performance is not transferred to the prefetching engine, and therefore to the user's benefit, i.e. perceived latency reduction. As one can observe, *precision* and *recall* are substantially lower when evaluated at the prefetching engine side. This fact is more noticeably in workload A than in the workload B, mainly due to its higher amount of embedded objects per page. The algorithm simulated to plot the figures is the DG with a lookahead window size of 2. The same behavior was observed in the plots for the PPM algorithm (see Figures 7.3 and 7.4).

We found that the prediction indexes measured at the prediction engine differ noticeably from their values at the prefetching engine because most of the predictions made are useless. This is mainly due to the fact that the algorithm predicts objects that the user has already requested, but they are waiting for an available connection in the client to be requested. Remember that, as the standard HTTP/1.1 recommends, the amount of simultaneous connections to a server is limited to 2 per client. An unexpected situation that can be observed in both workloads and algorithms is that the prefetch precision decreases when increasing the confidence threshold (see Figures 7.1(b), 7.2(b), 7.3(b), and 7.4(b)). This situation is caused by the useless predictions, which are hits from the point of view of the prediction engine, and misses from the prefeching engine perspective. In other words, the algorithms are mainly predicting that a user will request the embedded objects of an HTML file after requesting the HTML itself. Figure 7.5 illustrates this fact showing the amount of useless predictions over the total amount of predictions. We consider that a prediction is

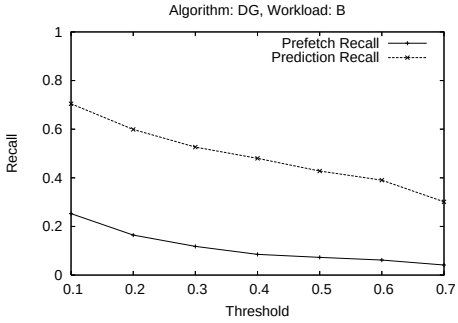


(a) Recall

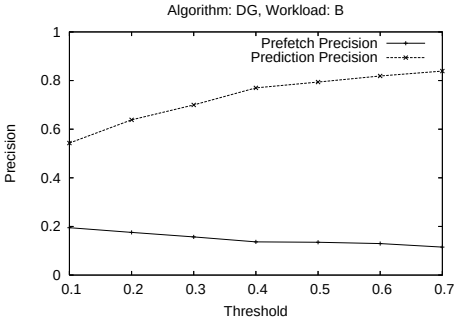


(b) Precision

Figure 7.1: Predictive metrics of DG algorithm when evaluated from the predictor point of view and from the prefetching point of view under workload A.

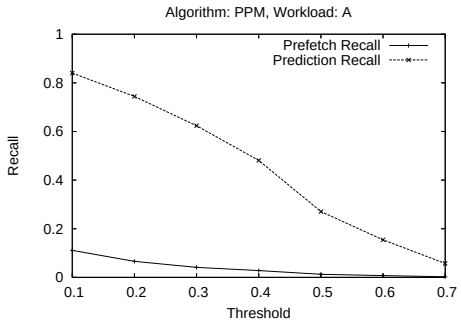


(a) Recall

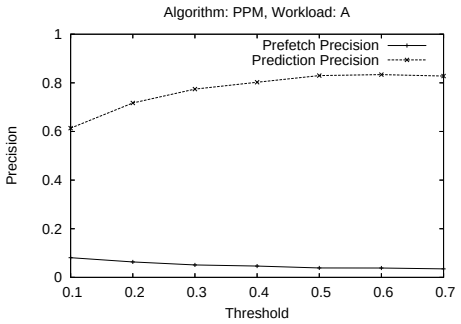


(b) Precision

Figure 7.2: Predictive metrics of DG algorithm when evaluated from the predictor point of view and from the prefetching point of view under workload B.

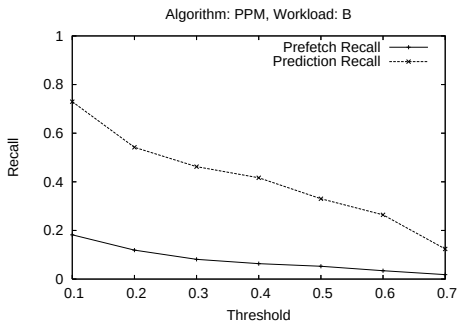


(a) Recall

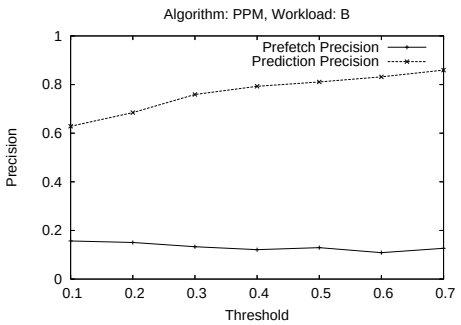


(b) Precision

Figure 7.3: Predictive metrics of PPM algorithm when evaluated from the predictor point of view and from the prefetching point of view under workload A.



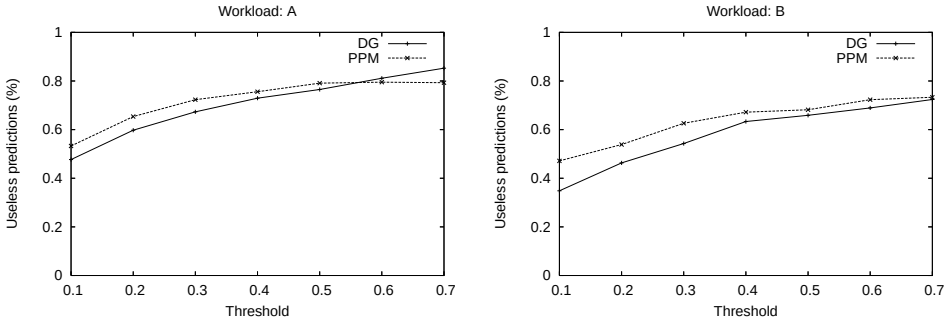
(a) Recall



(b) Precision

Figure 7.4: Predictive metrics of PPM algorithm when evaluated from the predictor point of view and from the prefetching point of view under workload B.

useless when the object predicted is already requested by the user but it is still in the browser queue waiting for an available connection. Figure 7.5 shows that, depending on the cutoff threshold of the algorithm, between 50% and 85% of the predictions made under the workload A refer to an embedded object. Under the workload B, the proportion is not so high due to the lower amount of embedded objects, but it is still high since it ranges from 35% to 75%.



(a) Workload A

(b) Workload B

Figure 7.5: Ratio of predictions that are made too late under current algorithms

The aforementioned findings encouraged us to propose a novel algorithm aimed at dealing with the high amount of embedded objects that exist in the current web to improve performance.

7.4 Double Dependency Graph Algorithm (DDG)

7.4.1 Description

The DDG algorithm is based on a graph that keeps track of the dependences among the objects accessed by the user. But unlike DG, it distinguishes two classes of dependences: dependences to an object of the same page and dependences to an object of another page. Like DG, the graph has a node for every object that has ever been accessed. There is an arc from node A to B, if and only if, at some point in time a client accessed to B within w accesses to A after B, where w is the *lookahead window* size. The arc is a primary arc if A and B are objects of different pages, that is, either B is an HTML object or the user accessed one HTML object between A and B. If there is no HTML accesses between A and B, the arc is secondary. The confidence of each primary or secondary transition, i.e., the confidence of each arc, is calculated by dividing the counter of the arc by the amount of appearances of the node, both for

primary and for secondary arcs. The pseudo-code for building the DDG prediction model is shown in Figure 7.6. As one can observe, the algorithm has the same order of complexity as the DG, since it makes the same graph but distinguishing two classes of arcs.

Figure 7.7 shows the state of the DDG algorithm after a training example with the following accesses: HTML1, IMG1, HTML2, IMG2 by one user; and HTML1, IMG1, HTML3, IMG2 by other user. Arrows with continuous lines represent primary arcs while dashed lines represent secondary arcs. Primary and secondary arcs represent the relation between object accesses of different pages and between object accesses of the same page, respectively.

The predictions are obtained firstly by applying a cutoff *threshold* to the weight of the primary arcs that leaves from the node of the last user access. In order to predict the embedded objects of the following page, a *secondary threshold* is applied to the secondary arcs that leave from the nodes of the objects predicted in the first step.

The DDG algorithm includes another useful parameter: the option of disabling the prediction of HTML objects. The algorithm determines if a requested object is an HTML file by looking at its MIME type in the response header given by the web server. This parameter is specially interesting when working in a dynamic web site. In this case, dynamic HTMLs will not be predicted by the algorithm. Other web sites that can benefit from disabling the prediction of HTMLs are those in which the cost of downloading an HTML object, i.e., its size, is very high when compared to the contained objects, e.g. images. One can observe that this fact occurs in the workload A (see Table 7.1), where HTMLs are 13 times larger than the embedded objects. The impact of this parameter on the performance is quantified below. The pseudo-code of the algorithm for obtaining the predictions is illustrated in Figure 7.8.

7.4.2 Selecting Parameter Values

In order to find the optimal values of the parameters of the prefetching algorithms, we checked different lookahead window sizes and secondary thresholds likewise DG and PPM algorithms were set up in Section 5.4.

To find the best *lookahead window* size, experiments were run ranging it from 2 to 16. For the sake of clarity, only some of those values are represented in Figure 7.9 for workload A and in Figure 7.10 for workload B. Figure 7.9 shows minor performance differences under the workload A between the different window sizes considering both cost perspectives, i.e., traffic increase and object traffic increase. A window size of 12 has been selected to compare in Section 7.5 the DDG algorithm under the workload A in order to have a similar aggressiveness to the DG and PPM algorithms. Under the workload B (see Figure 7.10), performance gets worse as long as the lookahead window size increases. For this reason, a window size of 2 has been selected to compare the algorithm under this workload. In addition, this window size is the one with less computing complexity.

```
1: Objective: Build a DDG prediction model
2: Input:
3:   S: set of users' sessions
4: Output:
5:   DDG: DDG prediction model
6: for each session s ∈ S do
7:    create empty lookahead window l
8:    for each object i ∈ s do
9:       HTMLinWindow ← 0
10:      if i MIMEtype is "text/html" then
11:         increment HTMLinWindow
12:      end if
13:      for each object o ∈ l from the newest to the oldest do
14:         if HTMLinWindow = 1 then
15:            increment weight of primary arc from o to i in DDG
16:         else
17:            if HTMLinWindow = 0 then
18:               increment weight of secondary arc from o to i in DDG
19:            end if
20:         end if
21:         if o MIMEtype is "text/html" then
22:            increment HTMLinWindow
23:         end if
24:      end for
25:      increment i access counter in DDG
26:      if l is full then
27:         remove the oldest object from l
28:      end if
29:      insert i in l
30:   end for
31: end for
32: for each object t ∈ DDG do
33:    for each output primary arc a ∈ t do
34:       a confidence ← weight of primary arc from t to a / t access counter
35:    end for
36:    for each output secondary arc a ∈ t do
37:       a secondary confidence ← weight of secondary arc from a / t access counter
38:    end for
39: end for
40: return DDG
```

Figure 7.6: Algorithm for making the prediction model in DDG

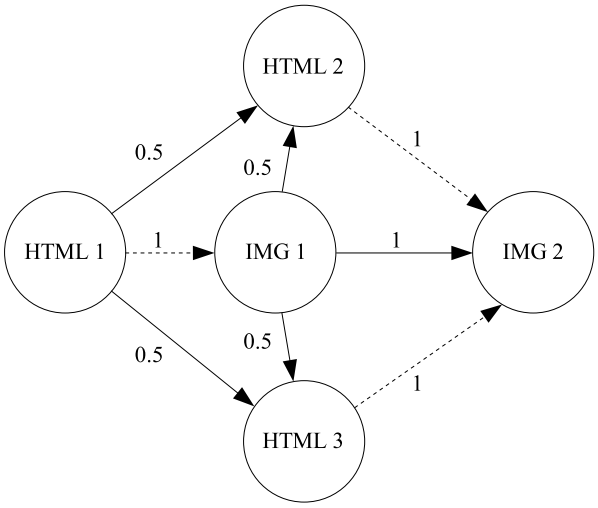


Figure 7.7: State of the graph of the DDG algorithm with a lookahead window size of 2 after the accesses HTML1, IMG1, HTML2, IMG2 by one user; and HTML1, IMG1, HTML3, IMG2 by other user

In a similar way, experiments to find out the optimal value for the *secondary threshold* were carried out. This parameter was ranged from 0.1 to 0.7 in steps of 0.1. As Figures 7.11 and 7.12 show, the best performance is achieved for a secondary threshold of 0.3 under the two selected workloads. This better behavior is evidenced both when comparing perceived latency to the traffic increase in bytes and to the traffic increase in amount of objects. Likewise when exploring the lookahead window size, results for only part of the experiments are shown in order to keep the plot readable.

Finally, experiments to determine the effect of not allowing the prediction of HTML files were carried out. Figure 7.13 shows the performance achieved by the algorithms varying this parameter under the workload A. Looking at plot 7.13(a), it is better to disable the prediction of HTML objects since its curve is always below the other one. However, when taking into account the object traffic increase as the main cost metric (see plot 7.13(b)), it is slightly better to predict the HTML files. Therefore, since performance differences are smaller when taking into account the object traffic increase, we selected the algorithm version that does not predict the HTML files in order to compare the algorithms (see Section 7.5). However, if the bandwidth consumption is not a critical issue, the version that predicts HTML objects should be selected. As one can observe in Figure 7.14, the selection of the better value of the parameter is easier under the workload B, since both plots give the same order, that is, the algorithm that predicts HTML files outperforms the version that does not predict them.

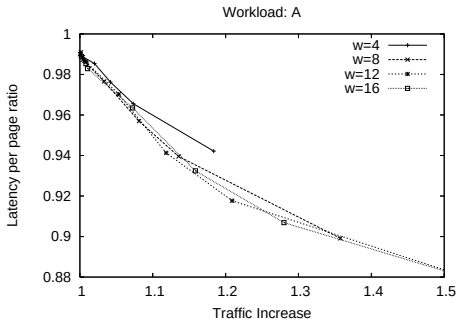
```
1: Objective: Obtain predictions of the next requests of the user
2: Input:
3:   DDG: DDG prediction model
4:   u: last user's access
5:   th: primary threshold
6:   thsec: secondary threshold
7:   enableHTMLpred: enable/disable the prediction of HTML objects
8: Output:
9:   P: Set of predictions
10: for each output primary arc a ∈ u in DDG do
11:   if a confidence > th then
12:     if not enableHTMLpred or a MIMEtype is not "text/html" then
13:       P ← P ⋃ {a}
14:     end if
15:     for each output secondary arc e ∈ a in DDG do
16:       if e secondary confidence > thsec then
17:         P ← P ⋃ {e}
18:       end if
19:     end for
20:   end if
21: end for
22: return P
```

Figure 7.8: Algorithm for making predictions in DDG

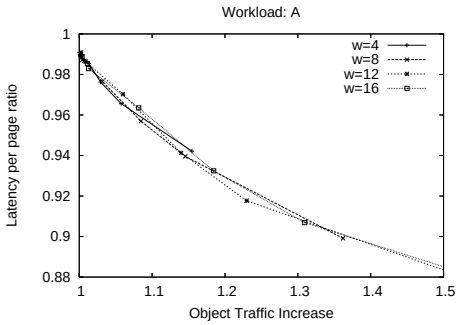
In short, the parameters that maximize the performance of the DDG algorithm are a 12 sized *lookahead window* with a *secondary threshold* of 0.3 with the prediction of HTML disabled for the workload A, and a *lookahead window* size of 2 and a *secondary threshold* of 0.3 allowing the prediction of HTML objects for the workload B.

7.5 Algorithm Comparison

Once the algorithms are tuned to have the best performance, they are compared in order to check the performance differences. Figure 7.15(a) shows that given any traffic increase in bytes, the algorithm that provides the lowest user-perceived latency under workload A is always the DDG. Similar results can be extracted when taking into account as a cost factor the traffic increase measured in amount of objects (see Figure 7.15(b)): DDG curve always falls below the other algorithms curves. To highlight the benefits of the new algorithm, it has been compared in Figure 7.16 against the DG algorithm since it performs better than the PPM. Figure 7.16(a) shows that given a value of latency per page ratio, the DDG algorithm requires between 25% and 60% less of bytes transference and between 15% and 65% less requests to

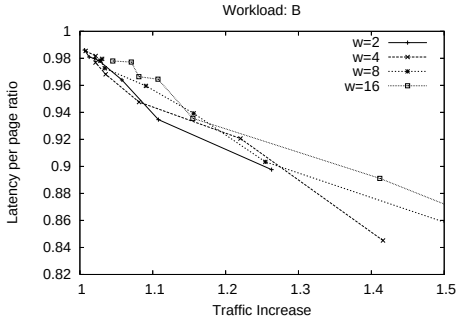


(a) Latency per page ratio as a function of Traffic increase

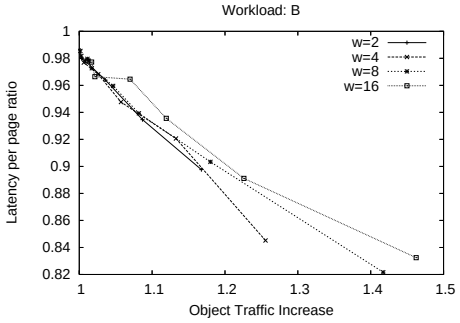


(b) Latency per page ratio as a function of Object Traffic increase

Figure 7.9: Selection of the lookahead window size of the DDG algorithm under the workload A

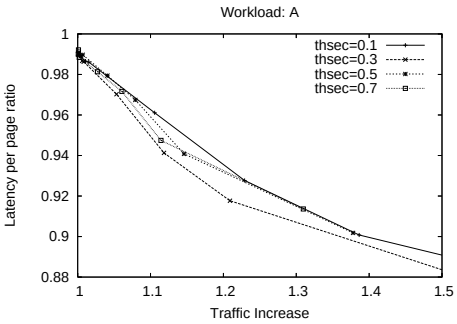


(a) Latency per page ratio as a function of Traffic increase

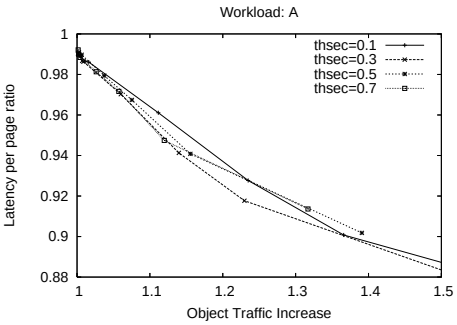


(b) Latency per page ratio as a function of Object Traffic increase

Figure 7.10: Selection of the lookahead window size of the DDG algorithm under the workload B

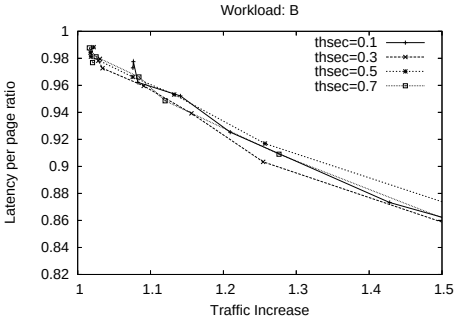


(a) Latency per page ratio as a function of Traffic increase

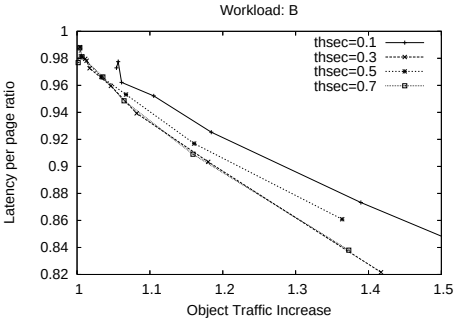


(b) Latency per page ratio as a function of Object Traffic increase

Figure 7.11: Selection of the secondary threshold of the DDG algorithm under the workload A

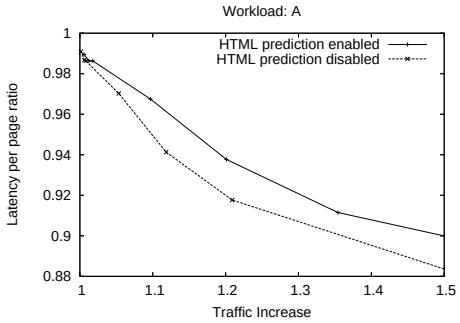


(a) Latency per page ratio as a function of Traffic increase

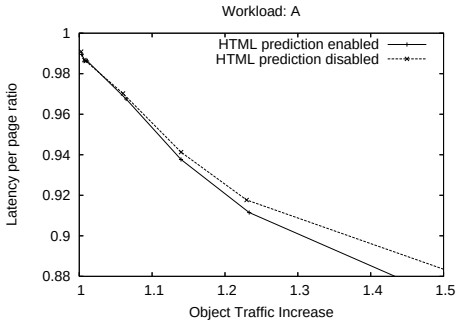


(b) Latency per page ratio as a function of Object Traffic increase

Figure 7.12: Selection of the secondary threshold of the DDG algorithm under the workload B



(a) Latency per page ratio as a function of Traffic increase

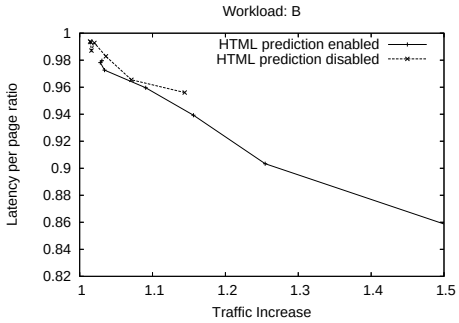


(b) Latency per page ratio as a function of Object Traffic increase

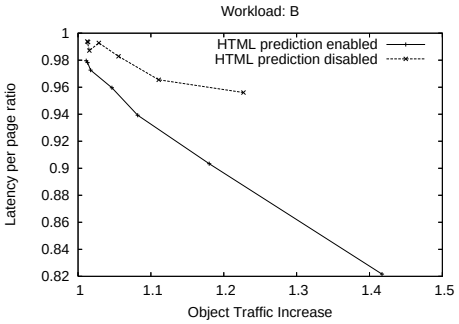
Figure 7.13: Deciding whether to predict HTML files in the DDG algorithm under the workload A

the server than the DG. On the other hand, given a value of traffic increase in bytes, DDG achieves a latency reduction between 15% and 45% higher than DG (see the continuous curve in Figure 7.16(b)). The benefits could rise up to 60% of extra perceived latency reduction when taking into account the object traffic increase, as the dashed curve in Figure 7.16(b) shows.

Performance comparison under the workload B is presented in Figure 7.17. Plot 7.17(a) shows that the selection of the best algorithm depends on which range the analysis is focused on. In this context, for a traffic increase lower than around 1.15, the best algorithm is the DDG, on the contrary, the best performance is achieved by the PPM, since its curve falls below the others. However, when looking at the object traffic increase (see plot 7.17(b)), the DDG always outperforms clearly the DG and the PPM algorithms. Figure 7.18 presents the relative comparison of the DDG algorithm against the PPM. By looking at plot 7.18(a), one can observe that, in the best situation, given a value of latency per page ratio, DDG requires about 60% less of traffic increase to achieve that latency reduction than the PPM, but in the worst situation it requires a 30% more of traffic increase. The dashed line in the same plot reveals that the DDG algorithm requires between about 35% and 90% less of requests to the server than the PPM to reduce a given amount the user-perceived latency. Plot 7.18(b) shows that in the best situation, with the same traffic increase (in bytes), DDG algorithms reduces around 65% more the user-perceived latency than the PPM. In the worst situation, DDG reduces about 15% less the perceived latency than PPM. When the object traffic increase is analyzed, the dashed line in



(a) Latency per page ratio as a function of Traffic increase



(b) Latency per page ratio as a function of Object Traffic increase

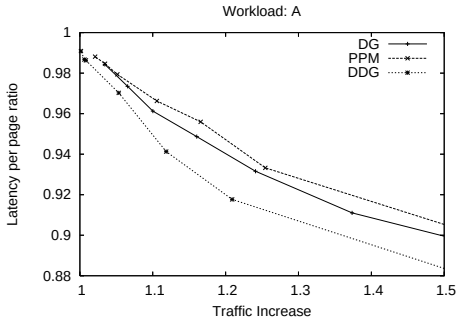
Figure 7.14: Deciding whether to predict HTML files in the DDG algorithm under the workload B

the plot shows that the DDG algorithm reduces the user-perceived latency between about 50% and 150% more than the PPM.

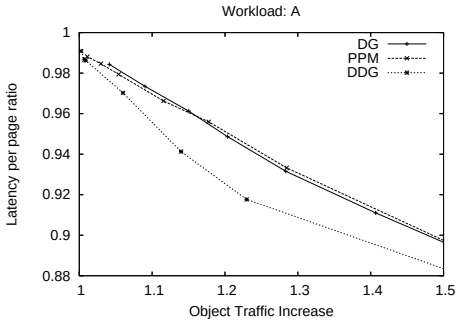
7.6 Conclusions

In this chapter we have presented an analysis of the performance achieved by two widely studied prefetching algorithms. We found that, due to the high amount of embedded objects in current web, most of the predictions provided by the existing algorithms are useless since they mainly predict the embedded objects of an HTML after accessing the HTML itself. These predictions, which are good from the predictor engine perspective, become useless for the client since it has no time to prefetch the hinted objects and, therefore, the user-perceived latency cannot be reduced. Taking into account this unwanted effect, a novel algorithm has been proposed and tested dealing with the characteristics of the current web. The DDG algorithm distinguishes between container objects (HTMLs) and embedded objects (e.g., images) to create the prediction model and to make the predictions. Results show that, given an amount of requests to the server, DDG always outperforms the existing algorithms by reducing the perceived latency between 15% and 150% more with the same extra requests. In addition, these results were achieved without incrementing the order of complexity of the existing algorithms.

In [Domènech 06b], a summary of the results shown in this section was presented.

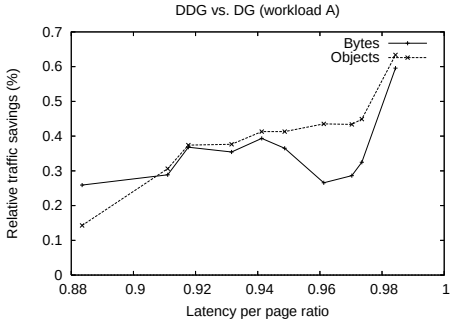


(a) Latency per page ratio as a function of Traffic increase

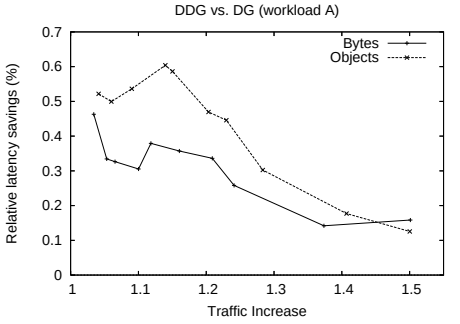


(b) Latency per page ratio as a function of Object Traffic increase

Figure 7.15: Algorithms comparison under the workload A

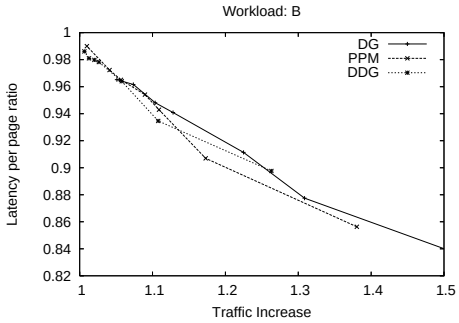


(a) Relative improvement in traffic increase as a function of latency per page ratio

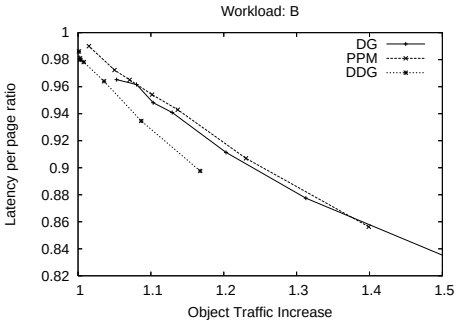


(b) Relative improvement in latency reduction as a function of traffic increase

Figure 7.16: Relative performance comparison of DDG vs DG algorithms under the workload A. Positive values mean that DDG outperforms DG, while negative ones would mean that DG outperforms DDG in the selected metrics.

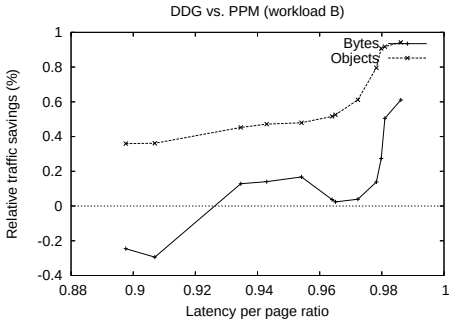


(a) Latency per page ratio as a function of Traffic increase

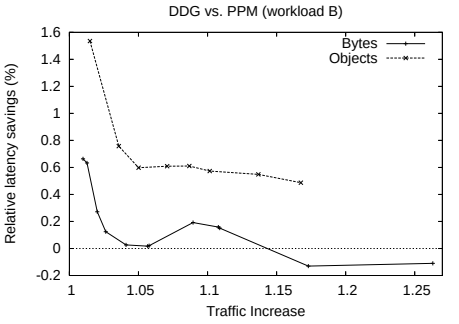


(b) Latency per page ratio as a function of Object Traffic increase

Figure 7.17: Algorithms comparison under the workload B



(a) Relative improvement in traffic increase as a function of latency per page ratio



(b) Relative improvement in latency reduction as a function of traffic increase

Figure 7.18: Relative performance comparison of DDG vs PPM algorithms under the workload B. Positive values mean that DDG outperforms PPM, while negative ones mean that PPM outperforms DDG in the selected metrics.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

Web prefetching has been researched for years with the aim of reducing the user-perceived latency; however, studies mainly focus on prediction performance rather than on the user's point of view. This dissertation has shown that a fast and accurate prediction is crucial for prefetching performance, but there are more factors involving the user's benefit. We have described and solved the main limitations when evaluating the performance from the user's perspective. To fairly evaluate web prefetching techniques, an experimental framework has been developed. By simulating the whole web architecture, all performance issues were measured.

One of the most important steps in a performance evaluation study is the correct choice of the performance indexes, but we found a wide heterogeneity of metrics in the literature. After its analysis, we clarified indexes definitions and detected the most meaningful metrics for the users. In spite of the focus on metrics used in web prefetching, some of the conclusions obtained for these metrics are common in any web performance evaluation study.

We also proposed an evaluation methodology to consider the cost and the benefit of web prefetching at the same time, although this approach could be used to any technique that implies a tradeoff.

Regarding the workload, we showed that prefetching performance differs depending on its age; and therefore, there is no reason to think that studies should be conducted using old traces.

We analyzed the limits on performance depending on the web prefetching architecture and found that a proxy-located predictor is the one less limited to reduce user-perceived latency since the potential limit of latency reduction is 67%. The prediction engines located at servers can only reduce up to 37% the overall latency. However, there is a noticeable amount of servers that can benefit from implementing

this prediction engine in a higher extent. The analysis also showed that a collaborative prediction engine could reduce almost all the user-perceived latency.

Once the methodological basics were established, the main prediction algorithms were tuned and compared. We found that a higher complexity of the algorithms does not involve better system performance. The algorithms comparison resulted in minor differences between DG and PPM, caused by the different average size of the predicted objects.

The environment conditions that can be found in the web are heterogeneous, so the optimal way of conducting web prefetching may be different depending on these conditions. We studied under which environments it is better to maximize either the amount of prefetched objects or the size of these objects. The results showed that, when the user's available bandwidth increases, prefetching techniques should prefetch a high amount of objects; whereas when the server processing time decreases, the amount of prefetched bytes should be maximized.

A deeper analysis of the results of the main existing algorithms showed that, due to the high amount of embedded objects in current web, most of the predictions provided by the existing algorithms are useless. This is because they mainly predict the objects embedded in an HTML after accessing the HTML itself, so there is no time to prefetch the hinted objects and therefore the user's cannot be reduced. Taking into account this unwanted effect, a novel algorithm dealing with the characteristics of the current web has been proposed and tested. The DDG algorithm distinguishes between container objects (HTMLs) and embedded objects (e.g., images) to create the prediction model and to make the predictions. Results showed that, given an amount of extra requests, DDG always reduces the latency more than the other existing algorithms. In addition, these results were achieved without increasing the order of complexity of the existing algorithms.

8.2 Summary of contributions

The main contributions of this dissertation are:

- An experimental framework for testing web prefetching techniques was developed.
- Performance key metrics were surveyed and their definition clarified.
- The most meaningful metrics from the user's point of view were identified.
- How the workload age affects the prefetching performance was shown both from the point of view of the user and of the prediction engine.
- The impact of the web prefetching architecture on the limits of reducing user-perceived latency was analyzed.

- A methodology for comparing algorithms from the user's point of view and for dealing with the costs and benefits of prefetching was developed.
- Design keys to adapt web prefetching algorithms to the environment conditions were provided.
- A comparative study of web prefetching techniques focusing on the user's perspective was performed and its results analyzed from the prediction point of view.
- A novel prefetching algorithm (DDG) that outperforms those existing without increasing the complexity was proposed and evaluated.

8.3 Future Work

8.3.1 Future Directions in Our Work

Despite the large amount of topics about web prefetching with which this dissertation deals, there still exist several scenarios to be explored. In this subsection, we describe some of them, showing new challenges and expected interesting results.

Through this thesis, we explored the prefetching algorithms for relative small traffic and object traffic increases. For those servers in which the increase of requests is not problem, a more aggressive policy should be adopted. In such a case, the prediction algorithms should be probably redesigned to consider a wider range of relation between object requests. In this context, it would be interesting to quantify the increase in network traffic and object requests that a given algorithm requires to achieve the maximum latency reduction for that algorithm.

How a prefetching system evolves in a long-term scenario is also an interesting issue to be explored. Probably, a mechanism for aging the learning of a prediction algorithm would be needed in order to adapt its predictions to the changes in the users' behavior. In this scenario there is another situation that requires attention. Since prediction engines cannot account the prefetch hits, the prediction performance will be negatively affected. An efficient system of notifying these prefetch hits could help the algorithms to improve the performance.

An issue not analyzed that could contribute to existing techniques is to classify users according to their navigation profile, and apply a prefetching strategy appropriate for their profile. For instance, if the prefetching system detects that a given user will have a large session in the server, an aggressive policy should be used. However, if the system detects that a given user is going to browse only a few pages in the server, the prefetching system should be more conservative.

8.3.2 Future Directions in Web Prefetching

Web prefetching performance is still far from their potential benefit, so there is a lot of research to be done. Once the increase in user's available bandwidths does

not involve better web performance and the requests processing time are difficult to reduce, the improvement of the perceived latency can only be done by means of predictive approaches.

As we described in Chapter 4, the implementation of prefetching at the proxy dramatically increases the potential benefits of this technique, so this direction should be explored in a near future. Any algorithm shown in this thesis can be implemented without changes in a proxy-based prefetching system. However, in this system, other benefits that current algorithms do not consider could be achieved. The main one is, probably, the fact that the prefetching between clients and the proxy will not require external traffic increase if only cached objects are predicted.

Bibliography

[Aggarwal 99] Charu C. Aggarwal, Joel L. Wolf and Philip S. Yu. *Caching on the World Wide Web.* IEEE Transactions on Knowledge and Data Engineering, vol. 11, no. 1, pages 95–107, 1999.

[Albrecht 99] David W. Albrecht, Ingrid Zukerman and Ann E. Nicholson. *Presending Documents on the WWW: A Comparative Study.* In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, Stockholm, Sweden, 1999.

[Angermann 02] Michael Angermann. *Analysis of Speculative Prefetching.* ACM Mobile Computing and Communications Review, vol. 6, no. 2, 2002.

[Bell 90] Timothy C. Bell, John G. Cleary and Ian H. Witten. Text compression. Prentice Hall, 1990.

[Bestavros 95] Azer Bestavros. *Using Speculation to Reduce Server Load and Service Time on the WWW.* In Proceedings of the 4th ACM International Conference on Information and Knowledge Management, Baltimore, USA, 1995.

[Bestavros 96] Azer Bestavros. *Speculative Data Dissemination and Service to Reduce Server Load, Network Traffic and Service Time in Distributed Information Systems.* In Proceedings of the Twelfth International Conference on Data Engineering, pages 180–187, New Orleans, USA, 1996.

[Bestavros 97] Azer Bestavros. *WWW Traffic Reduction and Load Balancing through Server-Based Caching.* IEEE Concurrency, vol. 5, no. 1, pages 56–67, 1997.

[Bonino 03a] Dario Bonino, Fulvio Corno and Giovanni Squillero. *Dynamic Prediction of Web Requests.* In Proceedings of the IEEE Congress on Evolutionary Computation, Canberra, Australia, 2003.

[Bonino 03b] Dario Bonino, Fulvio Corno and Giovanni Squillero. *A Real-Time Evolutionary Algorithm for Web Prediction.* In Proceedings of the IEEE/WIC International Conference on Web Intelligence, Halifax, Canada, 2003.

[Bouras 03] Christos Bouras, Agisilaos Konidaris and Dionysios Kostoulas. *Efficient Reduction of Web Latency through Predictive Prefetching on a WAN.* In Proceedings of the 4th International Conference on Advances in Web-Age Information Management, pages 25–36, Chengdu, China, 2003.

[Bouras 04] Christos Bouras, Agisilaos Konidaris and Dionysios Kostoulas. *Predictive Prefetching on the Web and Its Potential Impact in the Wide Area.* World Wide Web, vol. 7, no. 2, pages 143–179, 2004.

[Cao] Pei Cao. *Wisconsin web cache simulator.* http://www.cs.wisc.edu/ cao.

[Chen 02] Xin Chen and Xiaodong Zhang. *Popularity-Based PPM: An Effective Web Prefetching Technique for High Accuracy and Low Storage.* In Proceedings of the International Conference on Parallel Processing, Vancouver, Canada, 2002.

[Chen 03] Xin Chen and Xiaodong Zhang. *A Popularity-Based Prediction Model for Web Prefetching.* IEEE Computer, vol. 36, no. 3, pages 63–70, 2003.

[Chen 05] Xin Chen and Xiaodong Zhang. *Coordinated data prefetching for web contents.* Computer Communications, vol. 28, pages 1947–1958, 2005.

[Cherkasova 00] Ludmila Cherkasova and Gianfranco Ciardo. *Characterizing Temporal Locality and its Impact on Web Server Performance.* In Proceedings of the 9th International Conference on Computer Communication and Networks, Las Vegas, USA, 2000.

[cla] *ClarkNet WWW Server Log.* http://ita.ee.lbl.gov/html/contrib/ClarkNet-HTTP.html.

[Cohen 99] Edith Cohen, Balachander Krishnamurthy and Jennifer Rexford. *Efficient Algorithms for Predicting Requests to Web Servers.* In Proceedings of the IEEE INFOCOM '99 Conference, New York, USA, 1999.

[Cohen 02] Edith Cohen and Haim Kaplan. *Prefetching the means for document transfer: a new approach for reducing Web latency.* Computer Networks, vol. 39, no. 4, pages 437–455, 2002.

[Cooley 99] Robert Cooley, Bamshad Mobasher and Jaideep Srivastava. *Data preparation for mining World Wide Web browsing patterns.* Knowledge and information systems, vol. 1, no. 1, pages 5–32, 1999.

[Crovella 98] Mark Crovella and Paul Barford. *The network effects of prefetching.* In Proceedings of the IEEE INFOCOM'98 Conference, San Francisco, USA, 1998.

[Cunha 97] Carlos Cunha and Carlos F.B. Jaccoud. *Determining WWW User's Next Access and its Application to Pre-fetching.* In Proceedings of the Second IEEE Symposium on Computers and Communications, Alexandria, Egypt, 1997.

[Curewitz 93] Kenneth M. Curewitz, P. Krishnan and Jeffrey Scott Vitter. *Practical Prefetching via Data Compression.* In ACM SIGMOD International Conference on Management of Data, pages 257–266, Washington D.C., USA, 1993.

[Cárdenas 04] Luis Guillermo Cárdenas, Julio Sahuquillo, Ana Pont and José A. Gil. *The Multikey web cache simulator: A platform for designing proxy cache management techniques.* In Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network based Processing, La Coruña, Spain, 2004.

[Davison 01a] Brian D. Davison. *Assertion: Prefetching with GET is not Good.* In Proceedings of the 6th International Workshop on Web Caching and Content Distribution, Boston, USA, 2001.

[Davison 01b] Brian D. Davison. *NCS: Network and cache simulator – An introduction.* Technical report, Department of Computer Science, Rutgers University, 2001.

[Davison 02] Brian D. Davison. *Predicting Web Actions from HTML Content.* In Proceedings of the 13th ACM Conference on Hypertext and Hypermedia, College Park, USA, 2002.

[Davison 04] Brian D. Davison. *Learning Web Request Patterns.* In Web Dynamics - Adapting to Change in Content, Size, Topology and Use, pages 435–460. Springer, 2004.

[Domènech 04a] Josep Domènech, Ana Pont, Julio Sahuquillo and José A. Gil. *An Experimental Framework for Testing Web Prefetching Techniques.* In Proceedings of the 30th EUROMICRO Conference 2004, pages 214–221, Rennes, France, 2004.

[Domènech 04b] Josep Domènech, Ana Pont, Julio Sahuquillo and José A. Gil. *Giving Facilities for the Design and Test of Web Prefetching Techniques.* In Proceedings of the 2nd International Working Conference on Performance Modelling and Evaluation of Heterogeneous Networks (HET-NETs), Ilkley, United Kingdom, 2004.

[Domènech 04c] Josep Domènech, Julio Sahuquillo, José A. Gil and Ana Pont. *About the heterogeneity of web prefetching performance key metrics.* In Proceedings of the IFIP International Conference on Intelligence in Communication Systems (INTELLCOMM), Bangkok, Thailand, 2004.

[Domènech 05] Josep Domènech, Julio Sahuquillo, Ana Pont and José A. Gil. *How current web generation affects prediction algorithms performance.* In Proceedings of the 13th International Conference on Software, Telecommunications and Computer Networks (SoftCOM), Split, Croatia, 2005.

[Domènech 06a] Josep Domènech, José A. Gil, Julio Sahuquillo, Johann Márquez and Ana Pont. *La Heterogeneidad de los Índices de Prestaciones de la Prebúsqueda Web.* In Proceedings of the XXXII Conferencia Latinoamericana de Informática (CLEI 2006), Santiago, Chile, 2006.

[Domènech 06b] Josep Domènech, José A. Gil, Julio Sahuquillo and Ana Pont. *DDG: An Efficient Prefetching Algorithm for Current Web Generation.* In Proceedings of the 1st IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb), Boston, USA, 2006.

[Domènech 06c] Josep Domènech, José A. Gil, Julio Sahuquillo and Ana Pont. *Web prefetching performance metrics: A survey.* Performance Evaluation, vol. 63, no. 9–10, pages 988–1004, 2006.

[Domènech 06d] Josep Domènech, Ana Pont, José A. Gil and Julio Sahuquillo. *Guidelines for Evaluating and Adapting Web Prefetching Techniques.* In Proceedings of the XVII Jornadas de Paralelismo, Albacete, Spain, 2006.

[Domènech 06e] Josep Domènech, Ana Pont, Julio Sahuquillo and José A. Gil. *A Comparative Study of Web Prefetching Techniques Focusing on User's Perspective.* In Proceedings of the IFIP International Conference on Network and Parallel Computing (NPC 2006), Tokyo, Japan, 2006.

[Domènech 06f] Josep Domènech, Ana Pont, Julio Sahuquillo and José A. Gil. *Cost-Benefit Analysis of Web Prefetching Algorithms from the*

User's Point of View. In Proceedings of the 5th International IFIP Networking Conference, Coimbra, Portugal, 2006.

[Domènech 06g] Josep Domènech, Ana Pont, Julio Sahuquillo and José A. Gil. *A User-Focused Evaluation of Web Prefetching Algorithms*. Submitted, 2006.

[Domènech 06h] Josep Domènech, Julio Sahuquillo, José A. Gil and Ana Pont. *The Impact of the Web Prefetching Architecture on the Limits of Reducing User's Perceived Latency*. In Proceedings of the 2006 IEEE / WIC / ACM International Conference on Web Intelligence, Hong Kong, China, 2006.

[Domènech 06i] Josep Domènech, Julio Sahuquillo, Ana Pont and José A. Gil. *Design Keys to Adapt Web Prefetching Algorithms to Environment Conditions*. In Proceedings of the 1st International Conference on Communication Software and Middleware (COMSWARE), Delhi, India, 2006.

[Dongshan 02] Xing Dongshan and Shen Junyi. *A New Markov Model For Web Access Prediction*. Computing in Science and Engineering, vol. 4, no. 6, pages 34–39, 2002.

[Duchamp 99] Dan Duchamp. *Prefetching Hyperlinks*. In Proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems, Boulder, USA, 1999.

[Fan 99] Li Fan, Pei Cao, Wei Lin and Quinn Jacobson. *Web Prefetching Between Low-Bandwidth Clients and Proxies: Potential and Performance.* In Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling Of Computer Systems, pages 178–187, Atlanta, USA, 1999.

[Fielding 99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach and T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*, 1999.

[Fisher 03] Darin Fisher and Gagin Saksena. *Link prefetching in Mozilla: A Server driven approach*. In Proceedings of the 8th International Workshop on Web Content Caching and Distribution (WCW 2003), New York, USA, 2003.

[Fleming 97] Todd B. Fleming, Scott F. Midkiff and Nathaniel J. Davis. *Improving the Performance of the World Wide Web over Wireless Networks*. In Proceedings of the Global Telecommunications Conference, Phoenix, USA, 1997.

[goo a] *Google Search.* http://google.com/intl/en/help/features.html.

[goo b] *Google Web Accelerator.* http://webaccelerator.google.com/.

[Griffioen 94] James Griffioen and Randy Appleton. *Reducing File System Latency using a Predictive Approach.* Technical report, University of Kentucky, 1994.

[Group 01] Network Working Group. *Internet Web Replication and Caching Taxonomy.* RFC 3040, 2001.

[Gündüz 03] Sule Gündüz and M. Tamer Özsu. *A Web page prediction model based on click-stream tree representation of user behavior.* In KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, pages 535–540, New York, USA, 2003. ACM Press.

[Huang 05] Yin-Fu Huang and Jhao-Min Hsu. *Mining Web Logs to Improve Hit Ratios of Prefetching and Caching.* In Proceedings of the 2005 IEEE / WIC / ACM International Conference on Web Intelligence, Compiegne, France, 2005.

[Ibrahim 00] Tamer I. Ibrahim and Cheng Zhong Xu. *Neural Nets based Predictive Prefetching to Tolerate WWW Latency.* In Proceedings of the 20th IEEE International Conference on Distributed Computing Systems, Taipei, Taiwan, 2000.

[Iyengar 02] Arun Iyengar, Erich Nahum, Anees Shaikh and Renu Tewari. *Enhancing Web Performance.* In Proceedings of the IFIP World Computer Congress, Montreal, Canada, 2002.

[Jiang 97] Zhimei Jiang and Leonard Kleinrock. *Prefetching Links on the WWW.* In Proceedings of the IEEE International Conference on Communications, Montreal, Canada, 1997.

[Jiang 98] Zhimei Jiang and Leonard Kleinrock. *An Adaptive Network Prefetch Scheme.* IEEE Journal on Selected Areas in Communications, vol. 16, no. 3, pages 358–368, 1998.

[Jiang 02] Yingyin Jiang, Min-You Wu and Wei Shu. *Web Prefetching: Costs, Benefits and Performance.* In Proceedings of the 7th International Workshop on Web Content Caching and Content Distribution, Boulder, USA, 2002.

[Jin 00] Shudong Jin and Azer Bestavros. *Popularity-Aware GreedyDual-Size Web Proxy Caching Algorithms.* In Proceedings of the 20th International Conference on Distributed Computing Systems, Taipei, Taiwan, 2000.

[Johnson 01] Kirk L. Johnson, John F. Carr, Mark S. Day and M. Frans Kaashoek. *The measured performance of content distribution networks.* Computer Communications, vol. 24, no. 2, pages 202–206, 2001.

[Kargupta 04] Hillol Kargupta, Anupam Joshi, Krishnamoorthy Sivakumar and Yelena Yesha, editors. Data mining: Next generation challenges and future directions, chapter 3. AAAI Press, 2004.

[Khan 01] Javed I. Khan and Qingping Tao. *Partial Prefetch for Faster Surfing in Composite Hypermedia.* In Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems, San Francisco, USA, 2001.

[Khan 03] Javed I. Khan and Qingping Tao. *Exploiting Webspace Organization for Accelerating Web Prefetching.* In Proceedings of the IEEE/WIC International Conference on Web Intelligence, Halifax, Canada, 2003.

[Kim 03] Yuna Kim and Jong Kim. *Web Prefetching Using Display-Based Prediction.* In Proceedings of the IEEE/WIC International Conference on Web Intelligence, Halifax, Canada, 2003.

[Kokku 03] Ravi Kokku, Praveen Yalagandula, Arun Venkataramani and Michael Dahlin. *NPS: A Non-Interfering Deployable Web Prefetching System.* In Proceedings of the USENIX Symposium on Internet Technologies and Systems, Palo Alto, USA, 2003.

[Kroeger 97] Thomas M. Kroeger, Darrell D.E. Long and Jeffrey C. Mogul. *Exploring the Bounds of Web Latency Reduction from Caching and Prefetching.* In Proceedings of the 1st USENIX Symposium on Internet Technologies and Systems, Monterey, USA, 1997.

[Lan 00] Bin Lan, Stephane Bressan, Beng Chin Ooi and Kian-Lee Tan. *Rule-assisted prefetching in Web-server caching.* In CIKM '00: Proceedings of the ninth international conference on Information and knowledge management, pages 504–511, New York, USA, 2000. ACM Press.

[Lau 04] Kelvin Lau and Yiu-Kai Ng. *A Client-Based Web Prefetching Management System Based on Detection Theory.* In Proceedings of the Web Content Caching and Distribution: 9th International Workshop (WCW 2004), pages 129–143, Beijing, China, 2004.

[Lee 06] Heung Ki Lee, Gopinath Vageesan, Ki Hwan Yum and Eun Jung Kim. *A PROactive Request Distribution (PRORD) Using Web*

Log Mining in a Cluster-Based Web Server. In Proceedings of the International Conference on Parallel Processing (ICPP'06), Columbus, USA, 2006.

[Li 01] Ian TianYi Li, Qiang Yang and Ke Wang. *Classification Pruning for Web-request Prediction.* In Poster Proceedings of the 10th World Wide Web Conference, Hong Kong, China, 2001.

[Loon 97] Tong Sau Loon and Vaduvur Bharghavan. *Alleviating the Latency Reduction and Bandwidth Problems in WWW Browsing.* In Proceedings of the 1st USENIX Symposium on Internet Technologies and Systems, Monterey, USA, 1997.

[Maltzahn 99] Carlos Maltzahn, Kathy J. Richardson, Dirk Grunwald and James H. Martin. *On bandwidth smoothing.* In Proceedings of the 4th International Web Caching Workshop, San Diego, USA, 1999.

[Markatos 98] Evangelos Markatos and Catherine Chronaki. *A Top-10 Approach to Prefetching on the Web.* In Proceedings of the INET' 98, Geneva, Switzerland, 1998.

[Mitchell 93] Carl D. Mitchell, Randall A. Helzerman, Leah H. Jamieson and Mary P. Harper. *A parallel implementation of a hidden Markov model with durationmodeling for speech recognition.* In Proceedings of the Fifth IEEE Symposium on Parallel and Distributed Processing, Dallas, USA, 1993.

[Mobasher 02] Bamshad Mobasher, Honghua Dai, Tao Luo and Miki Nakagawa. *Using Sequential and Non-Sequential Patterns in Predictive Web Usage Mining Tasks.* In Proceedings of the IEEE International Conference on Data Mining, Maebashi City, Japan, 2002.

[Mozilla] Mozilla. *Link Prefetching FAQ.* http://www.mozilla.org/projects/netlib/Link Prefetching FAQ.html.

[Nanopoulos 01] Alexandros Nanopoulos, Dimitrios Katsaros and Yannis Manolopoulos. *Effective Prediction of Web-user Accesses: A Data Mining Approach.* In Proceedings of the Third International Workshop WEBKDD – Mining Web Log Data Across All Customers Touch Points, San Francisco, USA, 2001.

[Nanopoulos 02] Alexandros Nanopoulos, Dimitrios Katsaros and Yannis Manolopoulos. Exploiting web log mining for web cache enhancement, volume 2356, chapter in Lecture Notes in Artificial Intelligence (LNAI), pages 68–87. Springer-Verlag, 2002.

[Nanopoulos 03] Alexandros Nanopoulos, Dimitrios Katsaros and Yannis Manolopoulos. *A Data Mining Algorithm for Generalized Web Prefetching.* IEEE Trans. Knowl. Data Eng., vol. 15, no. 5, pages 1155–1169, 2003.

[Nicholson 98] Ann E. Nicholson, Ingrid Zukerman and David W. Albrecht. *A Decision-Theoretic Approach for Pre-Sending Information on the WWW.* In Proceedings of the 5th Pacific Rim International Conference on Artificial Intelligence, Singapore, Singapore, 1998.

[Padmanabhan 96] Venkata N. Padmanabhan and Jeffrey C. Mogul. *Using Predictive Prefetching to Improve World Wide Web Latency.* Computer Communication Review, vol. 26, no. 3, pages 22–36, 1996.

[Palpanas 99] Themistoklis Palpanas and Alberto Mendelzon. *Web Prefetching Using Partial Match Prediction.* In Proceedings of the 4th International Web Caching Workshop, San Diego, USA, 1999.

[Pandey 01] Ajay Pandey, Jaideep Srivastava and Shashi Shekhar. *A web intelligent prefetcher for dynamic pages using association rules – A summary of results.* In Proceedings of the First SIAM International Conference on Data Mining, Workshop on Web Mining, Chicago, USA, 2001.

[Pandey 02] Ajay Pandey, Ranga R. Vatsavai, Xiaobin Ma, Jaideep Srivastava and Shashi Shekhar. *Data Mining for Intelligent Web Prefetching.* In Proceedings of the Workshop on Mining Across Multiple Customer Touchpoints for CRM, Taipei, Taiwan, 2002.

[Pearson] Oskar Pearson. *Squid User's Guide.* http://squid-docs.sourceforge.net/latest/book-full.html.

[Peña-Ortiz 05] Raúl Peña-Ortiz, Julio Sahuquillo, Ana Pont and José Antonio Gil. *Modeling continous changes of the users' web dynamic behavior in the WWW.* In Proceedings of the Fith International Workshop on Software and Performance (WOSP 2005), Palma de Mallorca, Spain, 2005.

[Pitkow 99] James Pitkow and Peter Pirolli. *Mining Longest Repeating Subsequences to Predict World Wide Web Surfing.* In Proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems, Boulder, USA, 1999.

[Rabinovich 02] Michael Rabinovich and Oliver Spatscheck. Web caching and replication. Addison-Wesley, 2002.

[Sarukkai 00] Ramesh Sarukkai. *Link prediction and path analysis using Markov chains.* Computer Networks, vol. 33, no. 1-6, pages 377–386, 2000.

[Schechter 98] Stuart Schechter, Murali Krishnan and Michael D. Smith. *Using Path Profiles to Predict HTTP Requests.* In Proceedings of the 7th International World Wide Web Conference, Brisbane, Australia, 1998.

[Su 00] Zhong Su, Qiang Yang, Ye Lu and Hong-Jiang Zhang. *WhatNext: A prediction system for web requests using N-gram sequence models.* In Proceedings of the 1st International Conference on Web Information Systems and Engineering Conference, Hong Kong, China, 2000.

[Taguchi 87] Genichi Taguchi. System of experimental design. Unipub/Kraus International Publications, White Plain, 1987.

[Tao 02] Qingping Tao. *Impact of Webspace Organization and User Interaction Behavior on a Prefetching Proxy.* PhD thesis, Kent State University, 2002.

[Teng 05] Wei-Guang Teng, Cheng-Yue Chang and Ming-Syan Chen. *Integrating Web Caching and Web Prefetching in Client-Side Proxies.* IEEE Transactions on Parallel and Distributed Systems, vol. 16, no. 5, pages 444–455, 2005.

[tra] *The Internet Traffic Archive.* http://ita.ee.lbl.gov/index.html.

[UCB] UCB, LBNL and VINT. *Network Simulator ns (version 2).* http://www.isi.edu/nsnam/ns.

[Vahdat 98] Amin Vahdat, Thomas Anderson, Michael Dahlin, Eshwar Belani, David Culler, Paul Eastham and Chad Yoshikawa. *WebOS: Operating System Services for Wide Area Applications.* In Proceedings of the 7th Symposium on High Performance Distributed Computing Systems, Chicago, USA, 1998.

[Venkataramani 02] Arun Venkataramani, Praveen Yalagandula, Ravindranath Kokku, Sadia Sharif and Mike Dahlin. *The potential costs and benefits of long-term prefetching for content distribution.* Computer Communications, vol. 25, pages 367–375, 2002.

[Vöckler] Jens Vöckler. *HTTP Blast.* http://www.cache.dfn.de/DFN-Cache/Development/blast.htm.

[Wu 06] Bin Wu and Ajay D. Kshemkalyani. *Objective-Optimal Algorithms for Long-term Web Prefetching.* IEEE Transactions on Computers, vol. 55, no. 1, pages 2–17, 2006.

[Yang 03] Qiang Yang, Joshua Zhexue Huang and Michael Ng. *A data cube model for prediction-based web prefetching.* Journal of Intelligent Information Systems, vol. 20, no. 1, pages 11–30, 2003.

[Zhu 02a] Jianhan Zhu, Jun Hong and John G. Hughes. *Using Markov Chains for Link Prediction in Adaptive Web Sites.* In Proceedings of the First International Conference on Computing in an Imperfect World, Belfast, United Kingdom, 2002.

[Zhu 02b] Jianhan Zhu, Jun Hong and John G. Hughes. *Using Markov Models for Web Site Link Prediction.* In Proceedings of the 13th ACM Conference on Hypertext and Hypermedia, College Park, USA, 2002.

[Zukerman 99] Ingrid Zukerman, David W. Albrecht and Ann E. Nicholson. *Predicting Users' Requests on the WWW.* In Proceedings of the 7th International Conference on User Modeling, Banff, Canada, 1999.