De La Ossa Perez, BA.; Sahuquillo Borrás, J.; Pont Sanjuan, A.; Gil Salinas, JA. (2012). Key factors in web latency savings in an experimental prefetching system. Journal of Intelligent Information Systems. 39(1):187-207. doi:10.1007/s10844-011-0188-x.

Noname manuscript No.
(will be inserted by the editor)

Key Factors in Web Latency Savings in an Experimental Prefetching System

B. de la Ossa · J. Sahuquillo · A. Pont · J. A. Gil

the date of receipt and acceptance should be inserted later

Abstract Although Internet service providers and communications companies are continuously offering higher and higher bandwidths, users still complain about the high latency they perceive when downloading pages from the web. Therefore, latency can be considered as the main web performance metric from the user's point of view. Many studies have demonstrated that web prefetching can be an interesting technique to reduce such latency at the expense of slightly increasing the network traffic. In this context, this paper presents an empirical study to investigate the maximum benefits that web users can expect from prefetching techniques in the current web. Unlike previous theoretical studies, this work considers a realistic prefetching architecture using real traces. In this way, the influence of real implementation constraints are considered and analyzed. The results obtained show that web prefetching could improve page latency up to 52% in the prefetchable part of the studied traces.

Keywords Web prediction and prefetching · performance evaluation and measurement · web latency reduction

1 Introduction

The reduction of user-perceived web latency has been the focus of many research efforts over the past few years. The most popular techniques proposed to reduce this latency are web caching, geographical replication, and prefetching. The former two have been widely implemented in real environments. However, few studies have focused on web prefetching in real environments.

In web prefetching, the web browser requests objects before the user demands them. To do so, the prediction engine predicts and sends hints listing the objects that will probably be demanded next by the user, and these are the objects that are prefetched by the browser. As prefetching is a speculative technique, it can consume extra resources like network bandwidth and server processing time when hints are not accurate enough.

Department of Computer Engineering, Universidad Politecnica de Valencia
Camino de Vera, s/n, 46022 Valencia (Spain)
E-mail: berospe@doctor.upv.es, {jsahuqui, apont, jagil}@disca.upv.es

Intensive research works focusing on prediction and prefetching algorithms to improve web performance have been published, for instance [1–6]. However, few studies [7–10] focus on the maximum performance achievable through web prefetching and the main constraints to reach it. Some of these works study the upper bounds in performance of web prefetching from a theoretical point of view but, to the best of our knowledge, none of them has empirically analyzed how real restrictions affect the benefits of prefetching.

Web prefetching had not been widely used in the real world until now due to two main reasons. The first one is the limited user bandwidth that restricted the benefits of prefetching in the early Web. The other reason is that some early proposals required modification to the standard web protocols in order to support web prefetching.

Regarding the first reason, the ever increasing bandwidth in the current Internet opens a new window for exploiting web prefetching, thus becoming an interesting option for improving web performance. Web browsers based on Mozilla already support the web prefetching technique. Concerning the second reason, it has been demonstrated [11] that web prefetching can be implemented in a real environment without modifying the standard HTTP 1.1 protocol, making it compatible with current web browsers and servers.

Most authors use web traces when evaluating the performance of a prediction algorithm, but performance depends on the characteristics of the traces used. The main goal of this paper is to evaluate the maximum benefits that web prefetching can achieve for a given trace, that is, for a given website and the associated web client access patterns. Thus, the results could be used as the baseline system to be compared with the proposed algorithm. For this purpose, we define a perfect prediction algorithm and use a simulation framework based on a realistic prototype of a web prefetching architecture. In a previous work [12] we presented and studied the potential performance of a *perfect* prediction algorithm which always provides accurate predictions. In this way, we can discern which performance losses come from miss-predictions and which ones are inherent to the prefetching technique itself. Latency savings achieved by such an algorithm was explored by varying the maximum number of provided hints. The results show that current web prefetching is still far from achieving its maximum performance, mainly due to the predictor inaccuracy. This paper extends the work presented in [12] in several ways: i) all sections have been either rewritten or extended, ii) the impact on performance of two more factors (browser idle time and type of hints) is evaluated and analyzed, and, iii) the benefits that can be achieved by using additional strategies to improve prefetching performance are also tested.

Therefore, in this work we claim that a much research effort should be addressed to improve current web prefetching techniques. Moreover, we feel that these results should encourage the industry to efficiently handle web prefetching in commercial products.

The remainder of this paper is organized as follows. Section 2 presents previous work related to web prediction and prefetching benefits. Section 3 provides a general overview of some web prefetching concepts. Section 4 describes the evaluation methodology and the framework used in the experiments. Section 5 analyzes some key conditions in web prefetching and how they affect performance improvements. Section 6 analyzes other proposed techniques for improving web prefetching performance. Section 7 provides comparative experimental results with realistic web prediction algorithms. Finally, Section 8 presents some concluding remarks.

2 Related Work

The first relevant study that analyzed the potential benefits of web prefetching was carried out by Kroeger *et al.* [7]. They analyzed the limits on latency savings that caching and prefetching reached in several scenarios. Experiments were performed with a perfect off-line prediction algorithm and using traces obtained from a proxy server in 1996. An interesting conclusion of that research work was that prefetching doubles the latency reduction achieved by caching (according to this work, local proxy caching allows reducing latency up to 26%, whereas prefetching could reduce latency by about 57%), which is limited by the frequent update of objects in the web. However, their results are difficult to compare due to the assumptions made about the environment and the workload characteristics.

In 1999, Li Fan *et al.* [8] investigated, using traces from 1996, the limits on the benefits that a perfect prediction algorithm located at the proxy server could achieve. With those traces, they conclude that combining a large browser cache with delta-compression could permit to reduce latency only by 14.4%, whereas a perfect predictor could save latency by 28.5%.

Domenech *et al.* [9] studied the impact of the web architecture on the limits of latency reduction. They identified that the main constraint to obtain the upper bound in latency savings is due to the situation in which a user access cannot be predicted, e.g., the first access of a session and the first time the predictor sees an object. Their results show that latency reduction clearly depends on the location of the predictor. They stated that latency can be reduced by 36%, 54%, and 67% when the predictor is located at the server side, client, or proxy, respectively. Latency reductions higher than 90% could be obtained if the predictor works collaboratively at different elements of the architecture.

Bouras *et al.* [13] stated that the results obtained in web prefetching experiments strongly depend on the architecture or model assumptions, and that the parameters that influence web prefetching are closely related to the web architecture itself.

Finally, Balamash *et al.* [10] proposed a mathematical model for a web prefetching architecture. Their results showed that prefetching was profitable even with the presence of a good caching system.

To sum up, the studies discussed above are quite theoretical either because they use analytical models or because prefetching is not applied under real conditions and scenarios. For instance, most of them assume that all the predicted objects will be prefetched, which is far from real implementations.

On the other hand, Ossa *et al.* [14] focused on how to implement web prefetching in real environments making it compatible with commercial products and standard protocols.

This paper studies the upper bounds in latency savings of prefetching but, unlike previous studies, assuming such real conditions and analyzing how these conditions impact the design of a web prefetching architecture.

3 Baseline Web Prefetching System and Assumptions

Web prefetching is split in two main components: a predictor engine, and a prefetching engine. The former predicts the next user accesses by using previous usage information. This information usually differs depending on the element of the Web architecture (the

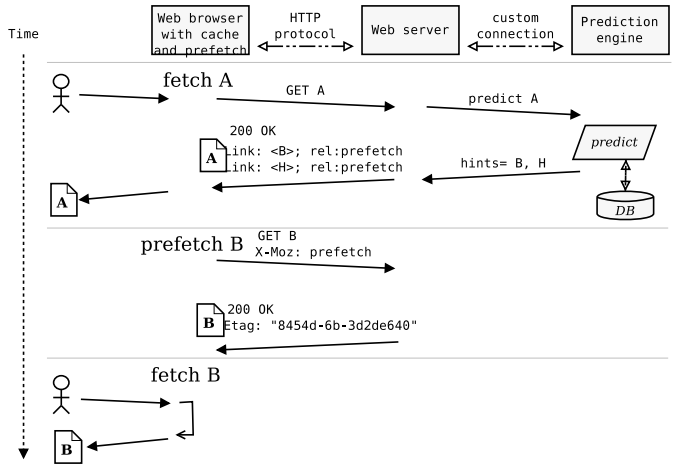


Fig. 1 Communication between the web browser, the web server and the prediction engine when using web prediction and prefetching

client, the proxy or the server) in which the prediction engine has been located. When it is located at the client, only one user access pattern is used to perform predictions [2, 15]. However, when the predictor is at the proxy server, it takes advantage of the multi-user and multi-server information gathered at this element to perform the predictions [8, 13, 16]. Finally, if the engine is located at the server, it makes predictions based on multi-user accesses to the same website [6, 17, 3, 18].

In the architecture used in this work, the predictor is located at the web server, and the prefetcher is at the client. More details can be found in [14]. Fig. 1 shows an example of how the client and the web server communicate with each other using the HTTP protocol. First, when the user clicks on a URI, the browser requests the object from the web server. Then, the prediction engine performs a prediction based on the URI of the requested object. The result of the prediction is a list of hints that the web server includes as HTTP headers in the response (e.g.: `Link: /news/190309; rel:prefetch`). This work assumes that the prefetching engine, located at the browser, prefetches the objects referred by the received hints during the browsing idle time, and this is how Mozilla-based browsers work [11]. Once prefetched, the web browser stores the objects in its local cache. Therefore, only those objects that can be stored at the web client can be safely prefetched. In this way, if the user demands any of these objects later, they will be served without any network latency. It is important to restrict prefetching to well-designed web pages where GET requests do not have side-effects such as website modification, file upload, session logout, etc.

As depicted in Fig. 2, a web page consists of a main object (the one demanded by the user) referred to as primary and many embedded objects called secondary objects. When the user demands a web page identified by its URI, the web browser firstly requests the primary object of the page. Once this primary object is received, the browser processes it to get the secondary objects from the network or from the local cache. Although the requested web page can be the result of a dynamic request, many of the objects that compose the page are usually static and, consequently, cacheable. Moreover, a dynamically generated object can be cached if it is properly labeled. As all the cacheable objects are useful to be prefetched (precached), this study covers, among

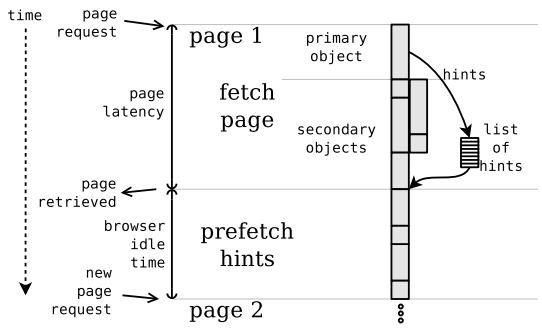


Fig. 2 Evolution in time of a user request with prefetch

other technologies, dynamic web server and application programming, and browser application programming like AJAX, Java, Flash, and other rich Internet applications. Two HTTP 1.1 persistent connections with no pipelining are assumed, since that is the maximum number of connections that the standard recommends [19]. So, the secondary objects are requested using two independent parallel network threads. When all the secondary objects are available in the browser, the download ends, since the whole page has arrived to the client. Our experimental environment, based on the mentioned architecture, assumes that secondary objects are retrieved once the primary object of the page is available (either from the network or from the browser cache). This assumption overestimates the page latency in all cases, but it is made in all the experiments both with and without prefetching, so its comparative impact on the page latency savings is negligible.

The page latency is defined as the elapsed time from when the user demands a page until all the objects in the page are received. This time finishes earlier if the page download is canceled. The browser idle time is defined as the elapsed time from when the last secondary object of a page is received until the next page is requested.

The hints received by the web browser in an object response are added to the *hints queue* of the page that contains that object. When the browser is idle (i.e., it is not downloading any object), it prefetches the objects referenced in the hint queue. This structure is handled as a FIFO queue; thus, it is important that the prediction engine provides the hints sorted according to the associated probability. In this way, if only a subset of the provided hints is going to be prefetched, these hints will be the most likely to be used. Remark that if a hint is already in the browser cache, it will not be prefetched. Mozilla web browsers include a special HTTP header (i.e., `X-moz:prefetch`) in prefetch requests. This allows the web server to filter or ignore such requests, for instance under overload conditions. When the user requests a new page, the hint queue is flushed.

The prefetching process is canceled and its hints queue is flushed if the user demands another page while the browser is prefetching. In such a case, the prefetch process is aborted to satisfy the user's current demand. When a prefetch cancellation occurs, those objects that have been partially predownloaded (i.e., a piece of them was already received and stored in cache) are allowed to reside in cache. Those parts can be reused if the objects are demanded later.

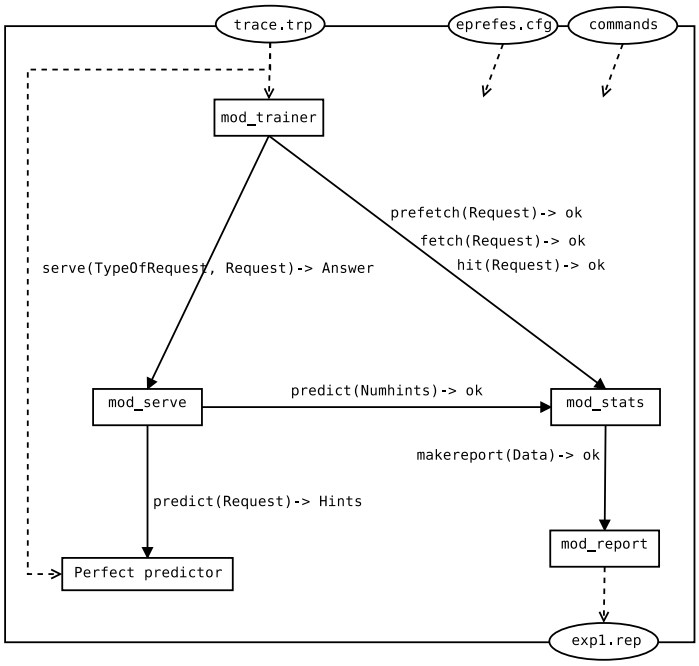


Fig. 3 Experimental framework

4 Experimental Environment

This section describes the framework, the evaluation methodology and the workload used to carry out the experiments.

4.1 Simulation Framework

To carry out the experiments we used *Delfos* [14], a framework developed to study prefetching in a real system. This framework allows developing, testing, and evaluating prefetching techniques. Figure 3 depicts the part of the framework architecture used in this study. Below its main modules and features are described.

Web clients operation is emulated by the *mod-trainer* module, which reads a trace file and generates the load to the prediction engine, thus making it suitable for training the prediction algorithm or for performing controlled experiments.

The *mod-trainer* module reads the trace file sequentially and sends objects' requests to the server (*mod-serve* and predictor) as indicated in the trace file. If hints are received within the response, the module sends prefetch requests during the browser idle time.

The *mod-serve* and the perfect predictor emulate the web server with prediction capabilities.

The *mod-trainer* and *mod-serve* modules notify their operations to the *mod-stats*, which maintains variables and calculates performance indexes that can be used for comparison purposes, for instance, to evaluate the prediction accuracy and usefulness or the resources consumed by the prediction algorithm. These data are calculated and

written to disk periodically without stopping the process, so all statistics are available immediately.

Some provided statistics are: the received requests, fetched objects, hints sent to the web server, objects that were prefetched, prefetchs that were later fetched (prefetch hits), hints that were later proved right (good predictions). All these variables are measured both in number and byte size. This module also calculates different performance indexes, which are described in the next section.

For all the performance indexes, the mean value and the confidence interval are calculated according to the methodology discussed in Section 4.4.

Finally, the *mod-report* module generates periodic statistic reports provided by the statistics module and writes them to data files for later usage by specific tools such as *Gnuplot*.

4.2 Methodology

The *Delfos* framework implements a pool of simulated clients with prefetching capabilities fed by real web traces. These clients perform requests to a simulated web server with a prediction engine that includes a perfect prediction algorithm according to the *Delfos* architecture (further details can be found in [14]). This predictor is perfect in the sense that it always provides useful hints, that is, those hints that are associated to objects that will be requested later by the user. Notice that this can be accomplished only in a testing framework, by reading the client requests in advance from the trace file.

To carry out the experiments, the client cache is assumed to be unbounded. The main reason of this assumption is that the simulated clients are fed with traces that were captured from real clients with limited cache sizes. In these traces, the sum of the object sizes for each session in each trace never exceeded 10 MB per navigation session. Consequently, all sessions can be completely stored in the browser cache of current web browsing devices, and a browser cache greater than 10 MB can be considered unbounded for the purpose of the experiments.

4.3 Workload Description

The experiments are performed using two web traces (A and B) from different websites. The trace files were logged by Apache 2 web servers in a custom format that included, among other request information, the referrer, the user agent, and the object service time.

The initial traces provided by the web server included all the HTTP requests, but as not all could be used by real prediction algorithms and prefetching engines, we filtered them by request type and response code: the request protocol is HTTP (neither HTTPS nor any other); the request method is GET; and the response code is 200, 206 or 304. The requests that meet this filtering criteria are considered cacheable for the purpose of this work. Table 1 summarizes the main characteristics of these traces after the filtering process.

Trace A was obtained from a dynamic website that acts as the home page of an open source project, which mainly contains news posts, documentation, and forums. Most of the content is generated dynamically by PHP scripts that query a database.

Table 1 Trace characteristics

Characteristic		**Trace A**	**Trace B**
Starting date		Sept, 27th 2007	Mar, 21st 2005
Ending date		Jun, 18th 2008	Nov, 22nd 2006
Original	Object requests	9,087,514	27,338,401
Filtered	Unique objects	6,790	1,330
	Object requests	5,654,371	3,411,307
	Requests of objects smaller 10 kB	77%	70%
	Bytes transferred (MB)	28,135	42,910
	Unique IP addresses	131,668	48,283
	Browsing sessions	317,268	271,736
	Page requests	992,037	1,159,191
	Avg. page latency (seconds)	0.877 ± 0.117	2.075 ± 0.378

However, the dynamic URIs are persistent, and they are written using an Apache 2 feature that does not use URI query strings. The site is visited by worldwide users using a wide variety of web browsers.

Trace B is from the website of the School of Computer Science at the Universidad Politecnica de Valencia. This site mainly contains news posts, and information addressed at students, staff, and visitors. Unlike the previous website, the content of this site is not dynamically generated and its visitor community is much more constrained. In the school there are some web based information points. These machines are configured to surf only local web pages and to refresh periodically the home page when idle. To prevent such automated navigations that could artificially increase the prefetching benefits, the requests originated from those IP addresses have been removed to obtain the trace used in the experiments.

A web session is a group of page requests made by the same web browser (identified by its IP address). The trace file does not indicate when a session finishes, so we assume that a browser idle time longer than 15 minutes represents the end of the web session.

Since it is not possible to know the browser idle time of the last page in a session, in the experiments we assume it to be 30 seconds long.

4.4 Performance Measures

The page latency savings have been used as the main performance metric for measuring prediction and prefetching effectiveness in this work because our aim is to study the maximum benefit perceived by web users. Other performance indexes measured in this work are: object latency savings, byte recall, and object recall [20].

The latency per object is obtained from the service time reported by the trace. The latency is zero if the object is already in the browser cache. Therefore, the latency reduction per object is the ratio of the latency perceived using prefetching to the latency without prefetching. For simplification purposes, the network latency has not been considered in the experiments, This simplification reduces the web prefetching potential benefits, as proven in previous works [21], which showed that higher network latency allows higher benefit for web prefetching.

The page latency is obtained by performing an experiment without prefetching, and these values are used as a baseline for comparison purposes in the experiments.

The page latency savings percentage ($\nabla PL(\%)$) and the recall percentage have been calculated as:

$$\nabla PL(\%) = (1 - \frac{\text{Avg. PL with Prefetch}}{\text{Avg. PL no Prefetch}}) * 100$$

$$Recall(\%) = \frac{\text{Prefetch Hits}}{\text{User Requests}} * 100$$

All performance indexes have been obtained with a confidence interval of 95%. Nevertheless, for the sake of clarity, only average values are shown in the figures presented in this paper, as the interval lengths for latency related performance indexes are always lower than 15% of the average value, and lower than 5% for recall related indexes. In order to calculate average values with confidence intervals, each experiment is split into shorter successive runs. The run length for each single experiment is 120K-user requests and each run consists of 20 intervals equally sized. On the other hand, for each single experiment a preliminary warming up phase of 70 intervals ($420,000$ object requests) that use the initial part of the trace was carried out before collecting statistics. This represents 7.43% and 12.31% of the total requests of Trace A and Trace B, respectively. The remaining part of the trace was used to continue the experiment and obtain the results.

5 Perfect Prediction Algorithm and Key Conditions to Prefetch

This section first discusses the *perfect* prediction algorithm used in this work to study upper bounds in latency savings. Second, it analyzes critical issues (or factors) that can reduce the benefits on latency savings that prefetching could provide.

5.1 The Perfect Prediction Algorithm

We define a perfect prediction algorithm as a predictor having four main properties: i) it never provides wrong hints, ii) it provides at least as many hints as objects can be prefetched during the browser idle time, iii) it only provides hints for those objects that have been demanded before at least once by any user, and iv) it has no adverse impact on the server functionality.

Property i) means that it provides a precision of 100% (i.e., all the provided hints will be demanded later by the user) and implies that this algorithm does not inject additional traffic with respect to non-prefetching techniques, since the objects prefetched according to the provided hints will always be later demanded by the user.

According to property i), the more provided hints the more latency savings. However, this number cannot be unbounded since browsers have limited slots of time to prefetch (browser idle time). This time ranges from a few seconds to several minutes in the traces. In other words, there is no way to guarantee that there will always be enough time to prefetch the objects of all the provided hints. That is why property ii) is defined. Notice that a provided hint means that the user will request that object, but this will not necessarily be prefetched since its request may result in a cache hit.

Property iii) is introduced because any real predictor requires to be informed of the existence of an object before that object can be predicted as hint.

Finally, property iv) is defined because the algorithm consumes computational resources and this fact could affect the web server response time. Obviously, if the algorithm has to provide responses in real time, it must be fast enough to avoid delaying the normal web server functionality. There are different ways to avoid or alleviate them, for instance, locating the prediction engine in a computer different to the web server, or implementing the prediction algorithm in a custom hardware. On the other hand, the prefetching of the hinted objects must not have side-effects such as website modification, file upload, or session logout.

In summary, we define the perfect algorithm as the best algorithm that can be designed by exploring the trace file at simulation time in advance. The next sections analyze how limitations in the current real web prefetch architecture impact on the latency savings.

5.2 Maximum Number of Hints per Prediction

This section is aimed at determining the optimal number of hints to be sent to the prefetching engine in order to find the trade-off between bandwidth consumption and number of prefetched objects. The perfect prediction algorithm reports three main types of hints for a page: the primary object of the next page to be requested by the user, the secondary objects of that next page, and the next pages.

When the prediction algorithm makes a prediction, it may provide one, several or no hints at all. The resulting hints are sent from the web server to the web browser in HTTP response headers. Those headers consume extra bandwidth, a fact that can be considered negligible when the number of hints is low. As a simple estimation, let us assume that each HTTP header hint is 32 bytes long, and a prediction always provides 10 hints; the traffic overhead caused by the hint header transference is about 1.08% in Trace A and about 0.83% in Trace B. If the response includes a high number of hints, for example a thousand hints, the web browser usually prefetches only a small subset of them. The reason is that a hinted object is neither prefetched when there is not enough time nor when the corresponding object is already in the browser cache.

Fig. 4 illustrates how the number of provided hints affects the page latency savings for traces A and B. Experiments were run by varying the maximum number of hints (1, 5, 10, and 30) that the prediction algorithm can provide in each prediction. Each point in the figure is labeled with the average number of hints provided per response in the different experiments. Notice that this value widely differs from the number of allowed hints (with the only exception of one hint). As expected, the higher the number of allowed hints the higher the relative difference. For instance, there is no difference when allowing just one hint; however, when allowing 30 hints, the perfect prediction algorithm provides, on average, 9.66 hints (about one third of the allowed) in Trace A. An interesting observation is that by allowing only one hint, the page latency perceived by users is reduced by 34%, and when providing 10 perfect hints per prediction, the page latency is reduced by half. This means that all hints have not the same impact on latency savings but the first ones have much stronger impact.

Regarding Trace B, page latency savings are lower than in Trace A although the number of provided hints is slightly higher. As the prediction algorithm is perfect we can affirm that Trace B has more unpredictable web requests than Trace A. However, even in this trace, important latency savings ranging from 30% to 39% (depending on the number of allowed hints) are achieved. Providing more than 30 hints does not save

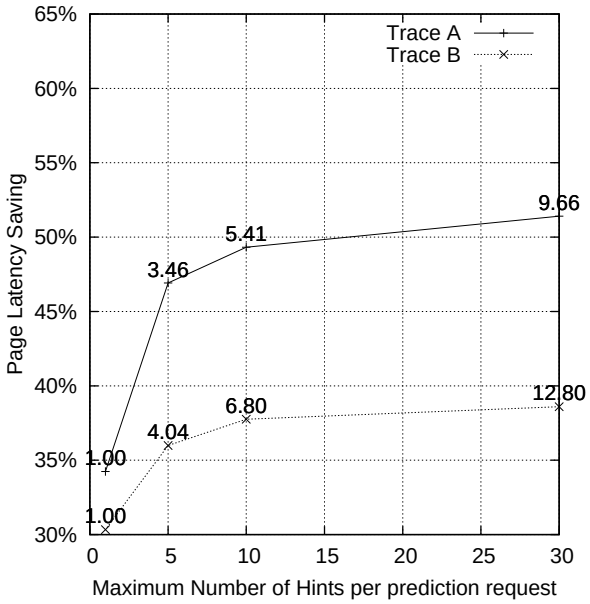


Fig. 4 Page Latency Saving with different number of maximum hints permitted per response

Table 2 The effect of the maximum number of hints per response

Trace	**Max Hints**	**Provided Hints**	**PCRH (%)**	**PERIT (%)**	**Recall (%)**	$\nabla PL(\%)$
A	01	1.00	0.02	67.55	6.80	34.24
	05	3.46	1.69	65.88	13.66	46.92
	10	5.41	2.18	65.39	17.70	49.32
	30	9.66	2.57	65.00	25.90	51.41
	∞	12.20	2.61	64.96	25.50	51.77
B	01	1.00	0.17	75.15	14.58	30.33
	05	4.04	3.47	71.85	35.21	35.99
	10	6.80	5.06	70.26	44.40	37.76
	30	12.80	5.29	70.02	46.26	38.60
	∞	19.37	5.14	70.17	46.27	38.56

PCRH: Prefetchs that are cancelled with remaining hints
PERIT: Prefetchs that end with remaining idle time

additional page latency, which means that the raw amount of hints cannot be the only concern of a prediction algorithm.

Table 2 extends the results presented in Fig. 4 to analyze in depth the effect that constraining the number of hints per response has on performance.

Experiments assume that the prefetching of hinted objects ends when a new page is requested by the user, as explained in Section 3. When this happens, three situations related to the provided hints can occur: i) there are hints that are still in the hint queue, waiting to be prefetched (labeled as *PCRH, Prefetch Cancelled with Remaining Hints* in Table 2); ii) all the hinted objects have been prefetched, and the browser is waiting for a new click (labeled as *PERIT, Prefetch Ends with Remaining Idle Time* in the aforementioned table); iii) it is the last page of the session so no more objects

will be prefetched. The latter case is not shown in the table because its value always remains constant (i.e., 32.43% of the total pages in Trace A, and 24.68% in Trace B). Notice that even when the perfect prediction algorithm is allowed to provide as many hints as possible, barely 2.61% of the page requests have remaining hints. This fact means that the browser idle time in the trace is long enough to prefetch the objects predicted by the perfect prediction algorithm.

The obtained recall is not higher than 30% in Trace A, even when allowing the perfect prediction algorithm to provide all the possible hints. The reason is that many pages in the website used in the experiments share most of their secondary objects. That is, the first page requested in a session requires many secondary objects to be fetched, and the subsequent pages reuse most of these secondary objects. When those secondary objects are required again, they are considered cache hits, not prefetch hits, and the recall does not increase. Since the objects of the first page in a session cannot be predicted, the prediction algorithm, even when it is perfect, can consider as hints only the objects of the second page visited as well as those of the following ones. As these pages have few different secondary objects, the amount of prefetch hits is very low compared to the total amount of objects requested by the web browser. Consequently, the recall in Trace A is rather low. Despite the low recall, the page latency savings is important in all cases, being higher than 51% when the number of hints allowed is over 30. This saving is much greater than the recall because the recall increases due to the secondary objects requested in the first page of a session. However, those objects represent a small amount of the page latency perceived by the user. As a consequence, the initial secondary objects prevent the increase of recall, but do not impede the increase of latency savings.

Results for Trace B show a higher recall, which almost doubles the recall obtained using Trace A. This happens because the algorithm provides, on average, more hints per prediction than in Trace A. Nevertheless, this fact does not result in a higher page latency savings. This situation is the opposite of the observed in Trace A: there are many objects that are prefetched and later required by users, so they are prefetch hits that increase the recall but barely reduce the overall page latency.

5.3 Browser Idle Time

The browser idle time required to run the experiments is taken from the collected traces. As this time widely differs from one request to another across the trace, depending on its value, the web browser can have enough time to prefetch all the objects, including those that have not been predicted.

This time depends on the user behavior when navigating through the website; that is, if the user navigates too fast then the web browser may not have enough time to prefetch all the predicted objects. This happens even if the prediction algorithm provides accurate hints that permits the system to reach the upper bound in latency savings. If the web browser does not have enough time to prefetch those objects, the latency saved in practice will not reach the upper bound. For this reason, it is important not only to provide good hints; a perfect predictor must also provide hints in the order that the user will request them. Then, the web browser will prefetch according to this order. In this way, if the browser is not able to prefetch all the objects, at least it will prefetch the ones going to be requested soon.

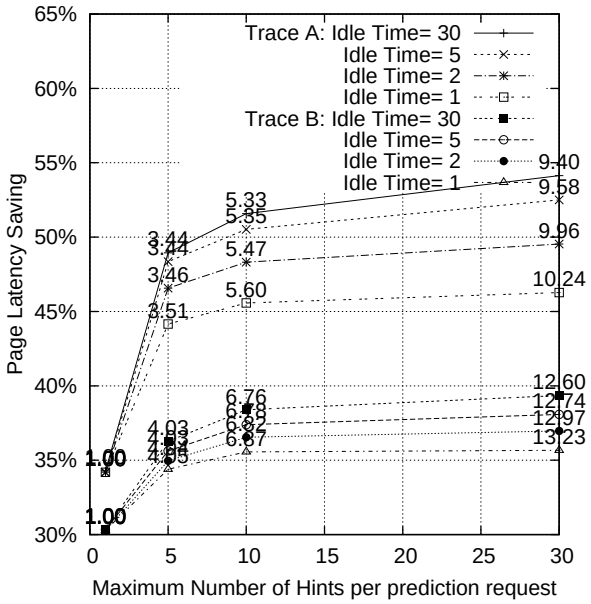


Fig. 5 Page Latency Saving with different number of maximum hints permitted per response and idle time

Fig. 5 shows how the idle time affects the page latency savings in Trace A and Trace B. For this purpose, browser idle times are fixed to 1, 2, 5, and 30 seconds in this experiment. Results show that values longer than 30 seconds (not shown) do not further benefit latency savings. This makes sense because although the algorithm is able to provide up to 30 hints per prediction, it only generates around 10 and 13 hints on average in traces A and B, respectively. When the perfect prediction algorithm is allowed to provide only one hint, an idle time of one second is enough to prefetch the object. This can be observed in the figure, since all curves draw the same point for abscissa 1, regardless of the assumed idle time. With the only exception of this point, the longer the idle time the higher the latency savings.

Table 3 shows the results without constraining the number of hints per response. It can be observed that increasing the idle time by 1 second reduces by half the amount of pages with remaining hints. This amount drops to 0% when the idle time is 30 seconds. However, when the number of hints per response is limited (e.g., to 5), the number of pages having remaining hints widely decreases by just providing an idle time 1 second longer. For an idle time of 3 seconds, the number of pages having remaining hints is about 1%.

5.4 Type of Hints

A prediction algorithm provides hints that can be primary objects or secondary (embedded) objects of web documents. It is unclear if it is appropriate to restrict the type of hints that can be provided by the prediction algorithm, as discussed below. By

Table 3 The effect of the browser idle time

Trace	Idle time (s)	Provided Hints	PCRH (%)	PERIT (%)	$\nabla PL(\%)$
A	1	13.50	8.84	58.73	46.04
	2	12.96	4.20	63.37	49.43
	3	12.58	2.49	65.08	51.02
	4	12.32	1.65	65.92	52.12
	5	12.15	1.18	66.39	52.65
	10	11.77	0.43	67.15	53.98
	15	11.66	0.26	67.31	54.27
	20	11.62	0.20	67.37	54.40
	30	11.59	0.14	67.43	54.57
B	1	20.54	9.16	66.15	35.61
	2	20.07	5.15	70.16	36.86
	3	19.81	3.64	71.67	37.45
	4	19.64	2.80	72.51	37.85
	5	19.52	2.28	73.03	38.06
	10	19.26	1.14	74.17	38.60
	15	19.17	0.74	74.57	38.82
	20	19.12	0.53	74.78	39.03
	30	19.09	0.34	74.98	39.35

PCRH: Prefetchs that are cancelled with remaining hints
PERIT: Prefetchs that end with remaining idle time

default, we allow the prediction algorithm to provide hints of any type, that is, both primary and secondary.

To explore whether the type of the hints impact the prefetching benefits, experiments are run by providing only primary hints, only secondary, or both types. The object latency savings and page latency savings are measured, and Table 4 shows the results. As one can observe, the page latency is reduced substantially by just providing as hint the primary object of the next page. This reduction is about 49% in Trace A and 33% in Trace B. Nevertheless, prefetch hits of secondary objects only provide about 4% and 6% of page latency savings in traces A and B.

There are three main reasons why primary objects provide much more page latency savings. The first reason is that the service time of primary objects is much longer than that of secondary objects. Nowadays, primary objects in most websites are dynamic files generated with PHP or other real-time programming languages, which often involve queries to a database. This is the case of the website in which Trace A was collected. Secondary objects are usually plain binary files like images, or text files that only need to be read from disk, such as JavaScript code or Cascade Style Sheets. In addition, the size of HTML files in most current websites is far larger than that of secondary objects, as discussed in [22]. The second reason is that primary objects are requested by web browsers using a single connection, while secondary objects are requested simultaneously using two parallel connections. The third reason is that the browser (in this experimental system) requests the secondary objects of the page only after the primary object has been received and parsed. This makes the primary object even more relevant to page latency.

Table 4 The effect of the type of hints permitted

Trace	Hint Type	Max Hints	Provided Hints	$\nabla OL(\%)$	$\nabla PL(\%)$
A	both	01	1.00	31.55	34.24
		05	3.46	43.77	46.92
		10	5.41	46.42	49.32
		30	9.66	49.27	51.41
	primary	01	1.00	31.60	34.30
		05	2.90	43.34	46.88
		10	3.94	44.85	48.49
		30	5.19	45.04	48.72
	secondary	01	1.00	1.58	1.46
		05	3.10	2.69	2.24
		10	4.95	3.73	2.67
		30	9.41	5.32	3.62
B	both	01	1.00	27.46	30.33
		05	4.04	35.96	35.99
		10	6.80	38.56	37.76
		30	12.80	39.58	38.60
	primary	01	1.00	27.28	30.13
		05	3.28	30.40	33.26
		10	4.95	30.51	33.37
		30	7.91	30.55	33.39
	secondary	01	1.00	2.96	1.66
		05	4.27	9.09	5.09
		10	7.02	10.06	5.81
		30	12.22	10.12	5.89

6 Enhancing Web Prefetching

From previous sections one can conclude that the more provided hints the higher the latency savings. This section analyzes the impact on latency savings of two techniques aimed at acting as a catalyst to enable the predictor to provide more hints than the baseline predictor explained above. The former technique, namely Predict On Secondary (POS), allows the perfect predictor to provide hints both in primary and in secondary objects. In the latter technique, Predict at Prefetch (P@P) [23], the predictor performs predictions not only when the user explicitly requests objects, but also when the browser prefetches previously predicted objects.

6.1 POS and P@P Techniques

In general, a web document consists of a primary object and many secondary objects (images, background, style sheet, etc). In the analysis presented above, we assumed that no prediction is performed when the browser requests a secondary object. Below we discuss the reason for this assumption.

In most web prefetching research works it is assumed that a prediction is performed for each object requested from the web server. This means that the web browser receives hints in any of the requested objects. However, this does not reproduce the real user navigation pattern for two main reasons: i) the user visits web documents and not individual objects; and ii) the user visits web documents, including those that are in the cache and, consequently, are not requested from the server. In this context, Mozilla web browsers do not prefetch objects referred by hints provided in secondary objects.

In a typical web prefetching architecture only the web browser is aware of which requests refer to primary objects and which ones to secondary objects. Nevertheless, predictions are made at the server side as assumed in Section 3. We extend this model so that the prediction algorithm firstly receives all the object requests, and then considers if an object is primary or secondary. The perfect prediction algorithm, by definition, knows in advance which objects requested by a browser are primary and which ones are secondary. In this way, we also study the effect of making predictions in secondary objects. The objects referred by hints provided with secondary objects are prefetched during the browser idle time while the object is being displayed. We refer to this technique as Predicting On Secondary objects (POS).

Predict at Prefetch (P@P) is a technique that improves overall web prefetching performance by giving hints also in prefetch requests. Web browsers receive more hints without negatively affecting the precision of the prediction algorithm. This proposal works well when there is unused bandwidth because the algorithm reports more hints, and the web browsers prefetches more objects. Those objects are prefetched if the user demands the prefetched object that contains the hints. They are prefetched later during the browser idle time while the object is being displayed. Further details can be found in [23].

6.2 POS and P@P Experimental Results

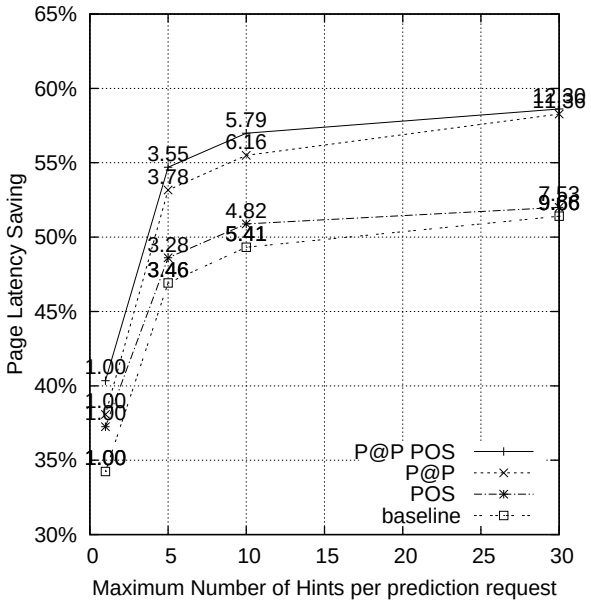
This section evaluates how much the techniques described above can help to reduce user-perceived latency.

As both techniques are orthogonal to each other, we measured the page latency savings that both techniques provide when working in an isolated way and when working together. Figures 6(a) and 6(b) show the results for the traces A and B, respectively. For comparison purposes, the figures show the latency savings obtained with a baseline web prefetch system (that is, without P@P or POS techniques).

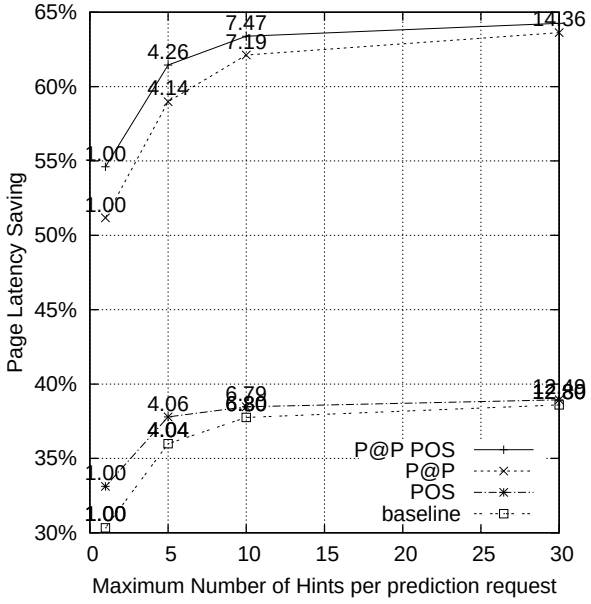
P@P always achieves about an additional 6% of page latency savings in Trace A, regardless of whether POS is enabled or not. P@P allows to predict in prefetch requests by providing the hints jointly with the prefetched objects. This technique exploits better the browser idle time and permits to increase the number of prefetch requests. In this way, when using a perfect prediction algorithm, the browser always has a pool of hints ready to be processed. For this reason, when using this technique, the perfect prediction algorithm permits to prefetch almost all the pages except the first one in the session. The benefit of P@P grows in Trace B up to 12%.

As expected, POS provides scarce page latency savings (about 1%) when it works either combined with the baseline or in conjunction with the P@P technique. POS obtains more benefits when the number of hints that can be predicted is restricted (*MaxHints*). The reason is that POS allows the prediction algorithm to make more predictions when the number of hints per prediction is severely limited, or when few primary objects are requested in comparison to secondary objects. Thus, providing more predictions makes it possible to increase the total amount of hints provided.

Giving hints in responses of secondary objects consumes additional computational resources to perform those additional predictions. Those hints consume bandwidth when they are sent to the browser, but they are not more valuable than the hints already provided when predicting for the primary object in the document. The results



(a) Trace A



(b) Trace B

Fig. 6 Page Latency Saving with different number of maximum hints permitted per response, POS and P@P

suggest that the prediction algorithm should not provide hints for secondary objects, because those hints will not significantly improve the latency savings.

7 Performance Comparison With Realistic Prediction Algorithms

This section compares the page latency savings of several real prediction algorithms that are known to provide the best cost-benefit ratio to the perfect prediction algorithm. The main objective of this comparison is to show that the benefits achieved by real proposals are still far from the performance capabilities of web prefetching. These algorithms are the Prediction by Partial Match (PPM) [5], the Dependency Graph (DG) [3], the Double Dependency Graph (DDG) [6], and the Referrer Graph (RG) [24].

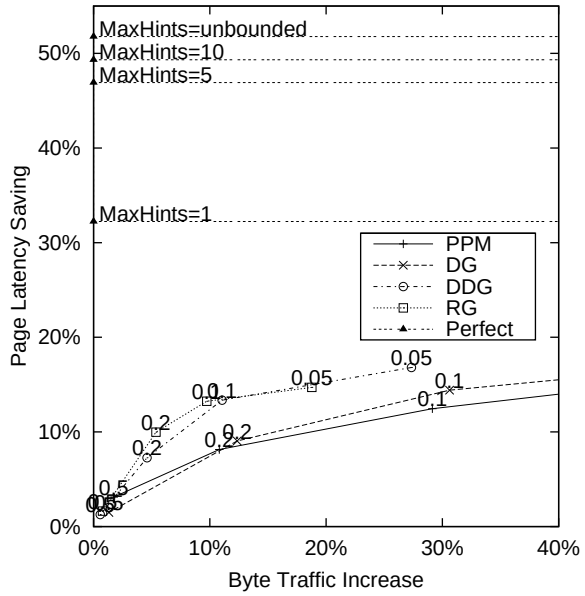
The algorithms parameters were individually tuned according to their intrinsic characteristics in order to obtain the best cost-benefit ratio with the lowest resource consumption in each individual case.

Figures 7(a) and 7(b) illustrate the cost and benefit obtained by the studied prediction algorithms in Trace A and B, respectively. In the real algorithms, the number represented in each line indicates the configured threshold, which defines the algorithm aggressiveness. The maximum number of hints allowed per prediction is 10, which is enough to obtain the best results [24]. The perfect prediction algorithm is represented by horizontal lines. The number in each line indicates the maximum number of hints allowed per prediction. Four different values have been studied (1, 5, 10 and unbounded).

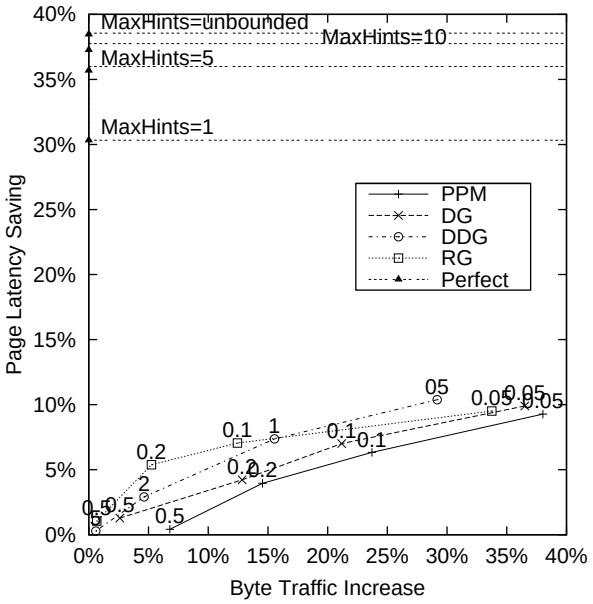
The page latency savings quantify the benefit, while the byte traffic increase measures the incurred cost. Experiments were run for different values of the prediction algorithms threshold parameter, because this is the most appropriate parameter to modify the aggressiveness of the algorithm prediction. Only those hints with a higher probability than the given threshold are returned to the web client.

DDG and RG obtain better cost-benefit ratio than PPM and DG. This difference is specially noticeable in Trace A, but it can also be appreciated in Trace B. RG is preferable when the byte traffic is restricted, while DDG is preferable when higher byte traffic is acceptable to obtain higher page latency savings.

As depicted in both figures, all those real prediction algorithms are far from the maximum page latency savings that can be achieved by the perfect prediction algorithm in these traces. In Trace A, the best real prediction algorithms (RG and DDG) can obtain page latency savings of 14% with a byte traffic increase of 10% and provide up to ten hints per prediction, while the perfect prediction algorithm obtains 32% latency savings by just providing one perfect hint per prediction. If this algorithm is allowed to provide up to ten hints per prediction, then it obtains almost 50% of page latency savings. In Trace B, the real algorithms can only obtain up to 12% page latency savings, and the perfect prediction algorithm is almost four times better. When comparing the results of both traces, it can be observed that Trace A allows us to obtain higher latency savings not only for the real algorithms used in the experiments, but also for a perfect prediction algorithm. That is, Trace A is more suitable for web prefetching than Trace B.



(a) Trace A



(b) Trace B

Fig. 7 Page Latency Saving obtained by perfect and real prediction algorithms with different aggressiveness

8 Conclusions

Despite the wide research efforts performed in web prefetching, few attempts have implemented prefetching in the real world. This paper has focused on the maximum performance that web prefetching can achieve, and has analyzed the main constraints to reach it in a real scenario. To this end, we defined a *perfect* prediction algorithm. Experimental results, obtained using two real traces, show that the baseline perfect prediction algorithm could provide latency savings ranging from 39% to 52%, depending on the input trace. The results also show that the latency savings obtained by the current realistic prediction algorithms are far from those potentially reachable. Consequently, more research efforts are needed to reduce this gap.

Regarding how experimental conditions impact performance, we can conclude: i) providing just five or ten perfect hints makes it possible to obtain the maximum page latency savings; ii) an idle time of about 10 seconds is enough to prefetch all the objects of the provided hints; and iii) most page latency savings are obtained by providing primary objects as hints, in contrast to secondary objects.

Two techniques were studied with the aim of improving the performance over the baseline perfect prediction algorithm. Predict on Secondary (POS) proposes to provide hints both in primary object requests and in secondary object requests. This technique only allowed us to obtain additional page latency savings of 1% in both traces. Predict at Prefetch (P@P) provides hints both in primary object requests and in prefetch requests. Results show that this technique obtains noticeable additional savings in page latency: about 6% in the first trace and 12% in the second trace.

References

1. E. Markatos and C. Chronaki, "A top-10 approach to prefetching on the web," *Proc. of INET, Geneva, Switzerland*, 1998.
2. A. Bestavros, "Using speculation to reduce server load and service time on the www," *Proc. of the 4th ACM International Conference on Information and Knowledge Management, Baltimore, USA*, 1995.
3. V. Padmanabhan and J. C. Mogul, "Using predictive prefetching to improve World Wide Web latency," in *Proc. of the ACM SIGCOMM Conference*, Stanford University, USA, 1996.
4. B. D. Davison, "The design and evaluation of web prefetching and caching techniques," Ph.D. dissertation, Rutgers University, October 2002.
5. T. Palpanas and A. Mendelzon, "Web prefetching using partial match prediction," *Proc. of the 4th International Web Caching Workshop, San Diego, USA*, 1999.
6. J. Domènech, J. A. Gil, J. Sahuquillo, and A. Pont, "DDG: An efficient prefetching algorithm for current web generation," *Proc. of the 1st IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb), Boston, USA*, 2006.
7. T. M. Kroeger, D. Long, and J. C. Mogul, "Exploring the bounds of web latency reduction from caching and prefetching," in *Proc. of the 1st USENIX Symposium on Internet Technologies and Systems*, Monterrey, USA, 1997.
8. L. Fan, P. Cao, W. Lin, and Q. Jacobson, "Web prefetching between low-bandwidth clients and proxies: Potential and performance," *Proc. of the ACM SIGMETRICS Conference on Measurement and Modeling Of Computer Systems*, pp. 178–187, 1999.
9. J. Domènech, J. Sahuquillo, J. A. Gil, and A. Pont, "The impact of the web prefetching architecture on the limits of reducing user's perceived latency," in *Proc. of the International Conference on Web Intelligence*. IEEE, 2006.
10. A. Balamash, M. Krunz, and P. Nain, "Performance analysis of a client-side caching/prefetching system for web traffic," *Comput. Netw.*, vol. 51, no. 13, pp. 3673–3692, 2007.

11. "Link prefetching in mozilla faq." [Online]. Available: https://developer.mozilla.org/en/Link_prefetching_FAQ
12. B. de la Ossa, J. Sahuquillo, A. Pont, and J. A. Gil, "An empirical study on maximum latency saving in web prefetching," *Proc. of the 2009 IEEE / WIC / ACM International Conference on Web Intelligence (WI'09)*, 2009.
13. C. Bouras, A. Konidaris, and D. Kostoulas, "Predictive prefetching on the web and its potential impact in the wide area," *World Wide Web: Internet and Web Information Systems, 7, Kluwer Academic Publishers, The Netherlands*, vol. 7, no. 2, pp. 143–179, 2004.
14. B. de la Ossa, J. A. Gil, J. Sahuquillo, and A. Pont, "Delfos: the oracle to predict next web user's accesses," *Proc. of the IEEE 21st International Conference on Advanced Information Networking and Applications, Niagara Falls, Canada*, 2007.
15. D. Duchamp, "Prefetching hyperlinks," in *Proc. of the 2nd USENIX Symposium on Internet Technologies and Systems*, Boulder, USA, 1999.
16. W. Teng, C. Chang, and M. Chen, "Integrating web caching and web prefetching in client-side proxies," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 5, pp. 444–455, 2005.
17. S. Schechter, M. Krishnan, and M. D. Smith, "Using path profiles to predict http requests," in *Proc. of the 7th International World Wide Web Conference*, Brisbane, Australia, 1998.
18. A. Bestavros and C. Cunha, "Server-initiated document dissemination for the WWW," *IEEE Data Engineering Bulletin*, 1996. [Online]. Available: citeseer.ist.psu.edu/bestavros96serverinitiated.html
19. "HTTP/1.1." [Online]. Available: http://www.faqs.org/rfcs/rfc2616.html
20. J. Domènech, J. A. Gil, J. Sahuquillo, and A. Pont, "Web prefetching performance metrics: A survey," *Performance Evaluation*, vol. 63, no. 9-10, pp. 988–1004, 2006.
21. J. Márquez., J. Domènech, A. Pont, and J. A. Gil, "Exploring the benefits of caching and prefetching in the mobile web," in *Second IFIP Symposium on Wireless Communications and Information Technology for Developing Countries (WCITD 2008)*, 2008.
22. T. Changa, Z. Zhuangb, A. Velayuthamc, and R. Sivakumara, "WebAccel: Accelerating web access for low-bandwidth hosts," *Computer Networks*, vol. 52, no. 11, pp. 2129–2147, 2008.
23. B. de la Ossa, J. A. Gil, J. Sahuquillo, and A. Pont, "Improving web prefetching by making predictions at prefetch," *Proc. of the 3rd EURO-NGI Conference on Next Generation Internet Networks Design and Engineering for Heterogeneity (NGI'07)*, pp. 21–27, 2007.
24. B. de la Ossa, A. Pont, J. Sahuquillo, and J. A. Gil, "Referrer graph: a low-cost web prediction algorithm," *Proc. of the 25th ACM Symposium On Applied Computing*, pp. 831–838, 2010. [Online]. Available: http://doi.acm.org/10.1145/1774088.1774260