DDG: An Efficient Prefetching Algorithm for Current Web Generation

Josep Domènech, José A. Gil, Julio Sahuquillo, Ana Pont
Department of Computer Engineering (DISCA)
Universitat Politècnica de València.
Camí de Vera, s/n. 46022 València (Spain)
jodode@doctor.upv.es; {jagil,jsahuqui,apont}@disca.upv.es

Abstract— **Web prefetching is one of the techniques proposed to reduce user's perceived latencies in the World Wide Web. The spatial locality shown by user's accesses makes it possible to predict future accesses based on the previous ones. A prefetching engine uses these predictions to prefetch the web objects before the user demands them. The existing prediction algorithms achieved an acceptable performance when they were proposed but the high increase in the amount of embedded objects per page has reduced their effectiveness in the current web. In this paper we show that most of the predictions made by the existing algorithms are useless to reduce the user's perceived latency because these algorithms do not take into account how current web pages are structured, i.e., an HTML object with several embedded objects. Thus, they predict the accesses to the embedded objects in an HTML after reading the HTML itself. For this reason, the prediction advance is not enough to prefetch the objects and therefore there is no latency reduction. As a result of a wide analysis of the behaviour of the most commonly used algorithms, in this paper we present the DDG algorithm that distinguishes between container objects (HTML) and embedded objects to create a new prediction model according to the structure of the current web. Results show that, for the same amount of extra requests to the server, DDG always outperforms the existing algorithms by reducing the perceived latency between 15% and 150% more without increasing the computing complexity.**

I. INTRODUCTION

Despite the high amount of research works and the improvements of the infrastructures, the latency that users perceive when exploring the Web is still high. Web prefetching is one of the techniques proposed in the literature to hide latencies, i.e., to reduce the perceived latency. It takes benefit from the spatial locality inherent to the users' accesses to web objects. In this way, it is possible to predict future accesses from the past ones. The prefetching engine makes use of these predictions to preprocess the predicted objects before the user actually demands them, so reducing the perceived latency.

To predict the users' accesses, researchers usually adapt 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 and Mogul [1], lies in a predictor of accesses to a local file system [2]. PPM algorithm, mainly used for lossless data compression [3], has been also used to predict web accesses by several authors [4], [5], [6], [7]. 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 [8], and the increase in the complexity of the web sites design. Despite this fact, to the knowledge of the authors, no prefetching algorithms taking into account the new Web sites characteristics have been proposed.

By analyzing an important set of current Web sites, we found that there is an noticeable increase in the amount of embedded objects per page [9]. In this context, most of the predictions made by the existing algorithms are useless to reduce the user's 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 objects embedded in an HTML after reading the HTML itself. In this paper 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 [10] 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 paper 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 in addition to dropping the need of resources, i.e., extra bandwidth and extra server load.

The remainder of the paper is organized as follows. Section II shows the experimental environment used to run the experiments. Section III presents the methodology used to evaluate prefetching algorithms. Section IV describes the existing algorithms implemented for comparison purposes. Section V presents the proposed prediction algorithm. Section VI analyzes the performance achieved by each algorithm. Finally, Section VII presents some concluding remarks.

II. Experimental Environment

This section presents the experimental framework used for the performance evaluation study and the workload used.

A. Framework

In [11] we proposed an experimental framework for testing web prefetching techniques. In this section we summarize the main features of such environment and the configuration used to carry out the experiments presented in this paper.

The architecture consists of three main parts (as shown in Fig. 1): the back end (server and surrogate), the front end (client) and optionally the proxy server, which is not used in the experiments presented in this paper. The framework implementation combines both real and simulated parts in order to provide flexibility and accuracy. To perform prefetching tasks, a prediction engine implementing different algorithms has been included in the server side. Clients take the generated predictions to download those objects in advance.

The back end part includes the web server and the surrogate server. The framework emulates a real surrogate, which is used to access a real web server. Although the main goal of surrogates is to act as a cache for the most popular server responses, we use it as a predictor. To this end, the surrogate adds new HTTP headers to the server response with the result provided by the prediction algorithms.

The server in the framework is an Apache web server set up to act as the original one. For this purpose, a CGI program returns objects with the same size and MIME type as those recorded in the traces.

The front end, or client part, represents the users' behavior exploring the Web with a prefetching enabled browser. To model the set of users that access concurrently to a given server, the simulator can be fed by using either real or synthetically generated traces. Simulated clients obtain the results of the prediction engine from the response, and prefetch the hinted object in their idle times, as implemented in Mozilla [12]. The simulator collects basic information for each request performed to the web server and writes it to a log file. By analyzing this log at post-simulation time, all the performance metrics related to the user and to the prefetching engine can be calculated.

The environment has been extended in several ways to perform the experiments presented in this paper, since originally it did not obtain information regarding the prediction engine. In this way, each hint received by the clients is also gathered into the output log. Together with the prediction hits, it is recorded the elapsed time since its prediction and since the first time the user was idle; i.e. time from which it is possible to prefetch objects.

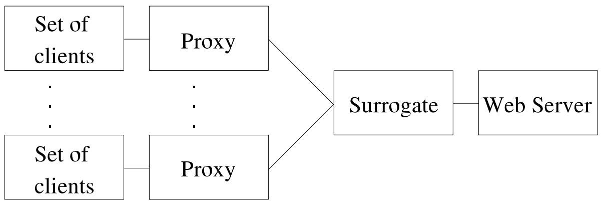


Fig. 1. Architecture of the simulation environment

TABLE I
Traces characteristics

Characteristics	Trace A	Trace B
Users	300	1,379
Page Accesses	2,263	10,282
Objects 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

B. Workload Description

The behavior pattern of users was taken from two different logs. Trace A contains accesses to a news web server and was collected during May 2003. It was obtained by filtering the web server accesses in the log file of a Squid proxy of the Polytechnic University of Valencia. 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 I. 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, as defined in Section III-A. 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's perceived latency can be reduced in 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's perceived latency by about 1%.

III. Evaluation Methodology

This section is aimed at introducing the performance metrics considered and the cost-benefit methodology used in the evaluation of the algorithms.

A. Performance Indexes

One of the most important steps in a performance evaluation study is the correct choice of the performance indexes. In this work, the performance of the algorithms has been evaluated by using the main metrics related to the user's perceived performance, prefetching costs and prediction performance [13]. Notice that 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. Therefore, each prediction index has a dual index; e.g., we can refer to the precision of the prediction engine and to the precision of the prefetching engine.

- Recall (Rc): The percentage of objects requested by the user that were previously predicted (or prefetched).
- Precision (Pc): The ratio of good predictions (or prefetched objects) to the number of predictions (or prefetched objects).
- Latency per page ratio (L_p): It is the ratio of the latency per page that prefetching achieves to the latency with no prefetching. The latency per page is calculated by comparing the time between the browser initiation of an HTML page GET and the browser reception of the last byte of the last embedded image or object for that page.
- Traffic Increase (ΔTr): The bytes transferred through the network when prefetching is employed divided by the bytes transferred in the non-prefetching case. Notice that this metric includes both the extra bytes wasted by prefetched objects that the user will never use and the network overhead caused by the transference of the prefetch hints. The variant *Object Traffic Increase* (ΔTr_{ob}) measures this cost taking into account only the amount of objects.

As it was demonstrated in [14], the metrics that concern the prediction are interrelated as Equation 1 shows:

$$\Delta Tr_{ob} = 1 - Rc + \frac{Rc}{Pc} \tag{1}$$

B. Cost-Benefit Analysis Methodology

Despite the fact that prefetching has been also used to reduce the peaks of bandwidth demand [15], its primary goal; i.e., the benefit, is usually the reduction of the user's perceived latency. Therefore, performance comparison between prefetching algorithms should be made from the user's point of view and using a cost-benefit analysis.

When predictions fail, prefetched objects waste user and/or 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 to achieve the latency reduction is the increment of the network load. This increment has two effects: the first is the increase in the amount of bytes transferred (measured through the *Traffic Increase* metric), and the second is the increase in the server requests (measured through the *Object Traffic Increase* metric). As a consequence, the performance analysis should consider the benefit of reducing the user's perceived latency at the expense of increasing the network traffic and the amount of requests to the server. Each simulation experiment in our prefetching environment considers as input the user's behavior, their available bandwidth and the prefetching parameters. The main results obtained are the traffic increase, the object traffic increase and the latency per page ratio.

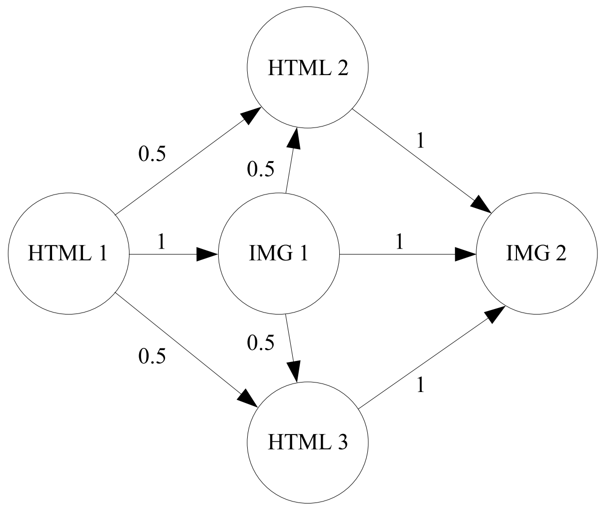


Fig. 2. State of the graph of the DG algorithm with a lookahead window size of 2 after the accesses HTML1, IMG1, HTML2, IMG2 by one user; and HTML1, IMG1, HTML3, IMG2 by other user

Comparisons of two different algorithms can only 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 paper 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 we varied the aggressiveness of the algorithm, i.e., how much an algorithm will predict. This aggressiveness is controlled by a confidence threshold parameter.

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. We can proceed in a similar way with the object traffic increase metric.

IV. Prefetching Algorithms

The experiments were run using two of the most widely used prediction algorithms in the literature: one variant of the *Prediction by Partial Match* (PPM) algorithm [16], [17], [5], [18], [7], [19] and the *Dependency Graph* (DG) based algorithm [1], [19].

A. 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. There is an arc from node A to B if and only if at some point in time

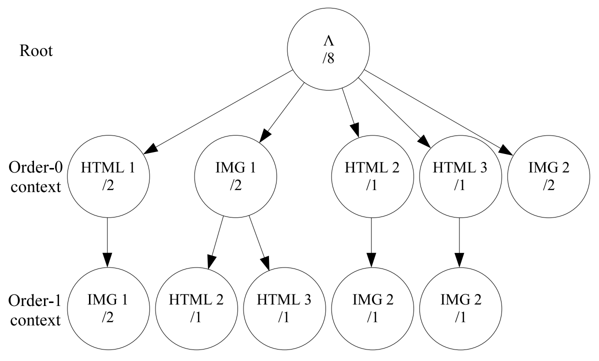


Fig. 3. State of the graph of a first-order PPM algorithm after the accesses HTML1, IMG1, HTML2, IMG2 by one user; and HTML1, IMG1, HTML3, IMG2 by other user

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. The prefetching aggressiveness can be controlled by a cutoff threshold parameter applied to the weight of the arcs.

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 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 behavior common through 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 2 shows the state of the graph of the DG algorithm for a lookahead window size of 2 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.

B. Prediction by Partial Matching (PPM)

The PPM prediction algorithm uses Markov models of *m* orders to store previous contexts. Predictions are obtained from the comparison of the current context to each Markov model. PPM algorithm has been proposed to be applied either to each object access [16], [17] or to each page (i.e., to each container object) accessed by the user [5], [18], [7]. In addition, two ways have been used to select which object or page shall be predicted: predicting the top-*n* likely objects [16], [18], [17] and using a confidence threshold [5], [19], [7]. In the experiments presented in this paper, PPM algorithm has been applied to objects instead of pages, and candidates are selected by means of a confidence threshold, since this is the configuration that performs better in the conditions we are studying in this paper [20].

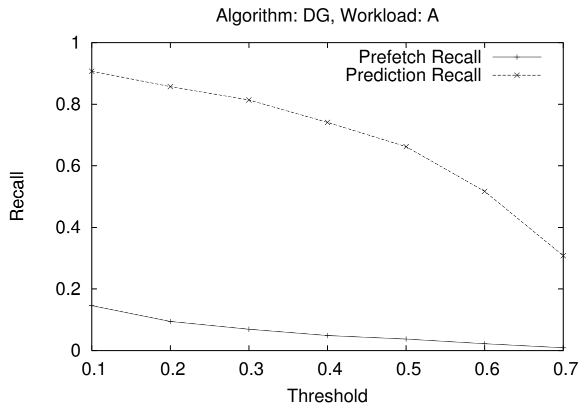
Figure 3 shows the graph obtained when applying the PPM algorithm to the training used in the previous example. Each node represents a context, where the root node is in the first row, the order-0 context is in the second, and the order-1 context is in the third. The label of each node also includes 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. 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 HTML3, IMG2 has two nodes in the order-1 context, i.e., one per each HTML on which it depends.

C. Uselessness Analysis

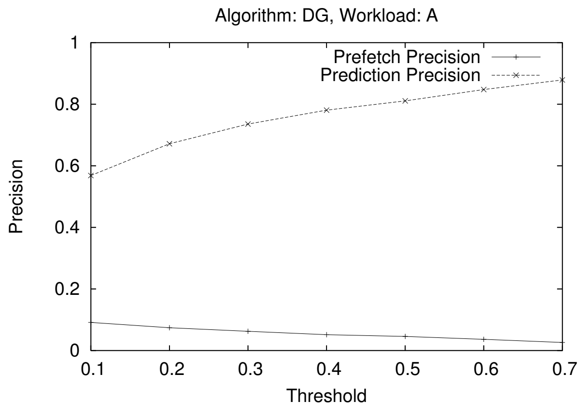
Figures 4 and 5 show the values of predictions metrics measured both at the prediction engine and at the prefetching engine under the workload A and B respectively. By analyzing the plots, we found that the prediction performance from the predictor point of view is quite good. However, this good performance 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 the 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, but these results are not included due to space restrictions.

We found that the prediction indexes measured at the prediction engine differ so much from their values at the prefetching engine in the fact that 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 [10] 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 is that the prefetch precision decreases when increasing the confidence threshold (see Figures 4(b) and 5(b)). This situation is caused by the useless predictions, which are hits from the point of view of the prediction engine, but they are 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 6 illustrates this fact showing the amount of useless predictions over the total amount of predictions. We consider that a prediction is 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 6 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%.

The aforementioned findings encouraged us to propose a novel algorithm aimed at dealing with the high amount

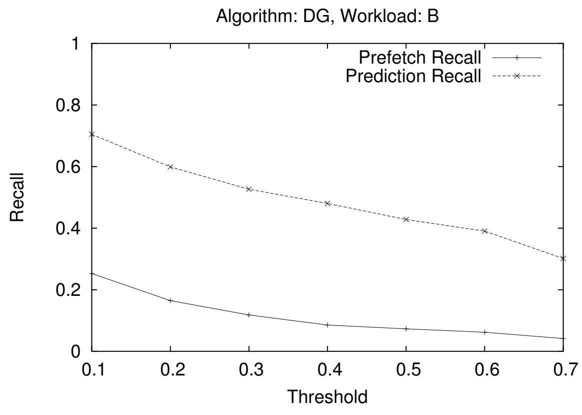


(a) Recall

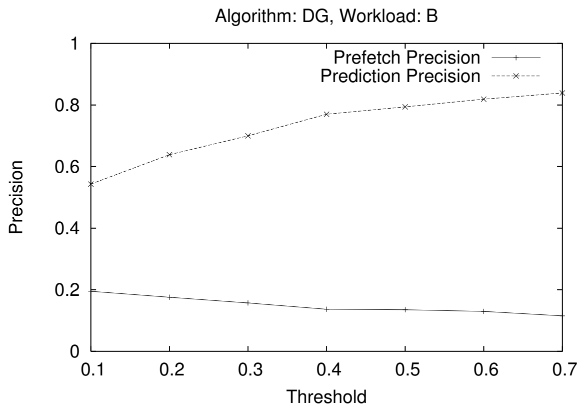


(b) Precision

Fig. 4. Predictive metrics when evaluated from the predictor point of view and from the prefetching point of view under workload A.

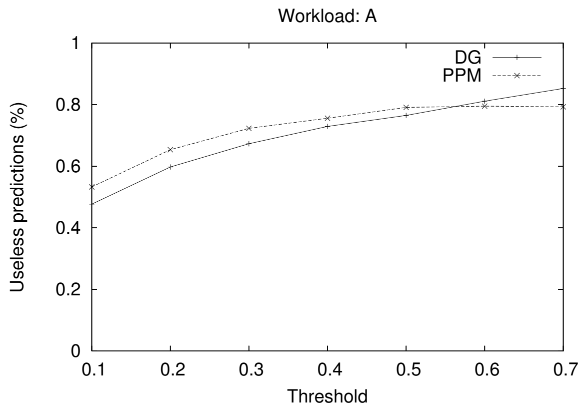


(a) Recall

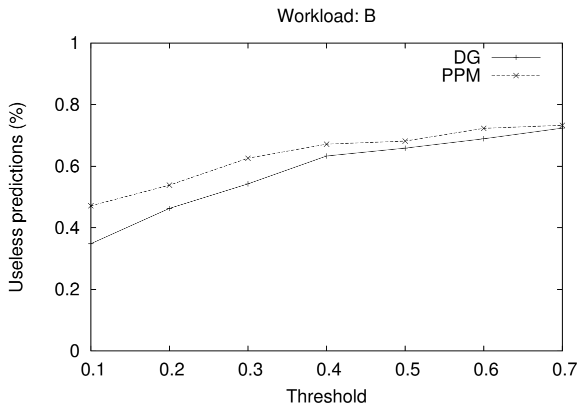


(b) Precision

Fig. 5. Predictive metrics when evaluated from the predictor point of view and from the prefetching point of view under workload B.



(a) Workload A



(b) Workload B

Fig. 6. Ratio of predictions that are made too late under current algorithms

```
Objective: Build a DDG prediction model
Input:
  $S$: set of users' sessions
Output:
  $DDG$: DDG prediction model
for each session $s \in S$ do
  create empty lookahead window $l$
  for each object $i \in s$ do
    $HTMLinWindow \leftarrow 0$
    if $i$ MIMEtype is "$text/html$" then
      increment $HTMLinWindow$
    end if
    for each object $o \in l$ from the newest to the oldest do
      if $HTMLinWindow = 1$ then
        increment weight of primary arc from $o$ to $i$ in $DDG$
      else
        if $HTMLinWindow = 0$ then
          increment weight of secondary arc from $o$ to $i$ in $DDG$
        end if
      end if
      if $o$ MIMEtype is "$text/html$" then
        increment $HTMLinWindow$
      end if
    end for
    increment $i$ access counter in $DDG$
    if $l$ is full then
      remove the oldest object from $l$
    end if
    insert $i$ in $l$
  end for
end for
for each object $t \in DDG$ do
  for each output primary arc $a \in t$ do
    $a$ confidence $\leftarrow$ weight of primary arc from $t$ to $a$ / $t$ access counter
  end for
  for each output secondary arc $a \in t$ do
    $a$ secondary confidence $\leftarrow$ weight of secondary arc from $a$ / $t$ access counter
  end for
end for
return $DDG$
```

Fig. 7. Algorithm for making the prediction model in DDG

of embedded objects that exist in current web to improve performance.

V. Double Dependency Graph Algorithm (DDG)

A. 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. 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.

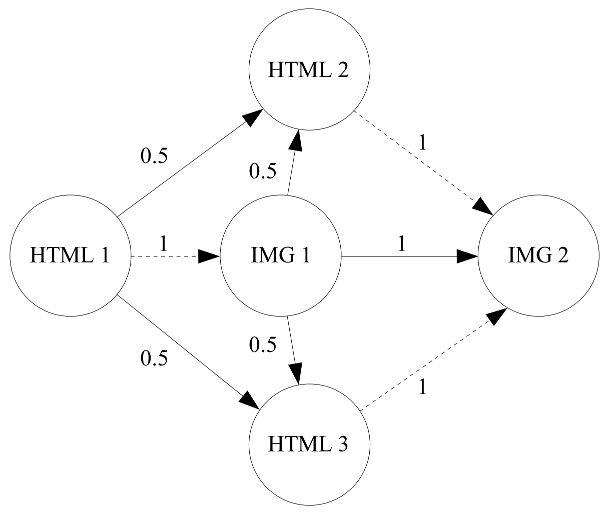


Fig. 8. State of the graph of the DDG algorithm with a lookahead window size of 2 after the accesses HTML1, IMG1, HTML2, IMG2 by one user; and HTML1, IMG1, HTML3, IMG2 by other user

Figure 8 shows the state of the DDG algorithm after the training example presented in Section IV-A. Arrows with continuous lines represent primary arcs while dashed lines represent secondary arcs. Primary and secondary arcs represent the relation between objects accesses of different pages and between objects accesses of the same page, respectively.

The predictions are obtained by firstly applying a cutoff *threshold* to the weight of the primary arcs that leaves from the node of the last user's 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 I), where HTMLs are 13 times larger than the embedded objects. The impact of this parameter on the performance is quantified in Section V-B. The pseudo-code of the algorithm for obtaining the predictions is illustrated in Figure 9.

```
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 $a \in u$ in $DDG$ do
11:   if $a$ confidence $>$ $th$ then
12:     if not $enableHTMLpred$ or $a$ MIMEtype is not $"text/html"$ then
13:       $P \leftarrow P \bigcup \{a\}$
14:     end if
15:     for each output secondary arc $e \in a$ in $DDG$ do
16:       if $e$ secondary confidence $>$ $thsec$ then
17:         $P \leftarrow P \bigcup \{e\}$
18:       end if
19:     end for
20:   end if
21: end for
22: return $P$
```

Fig. 9. Algorithm for making predictions in DDG

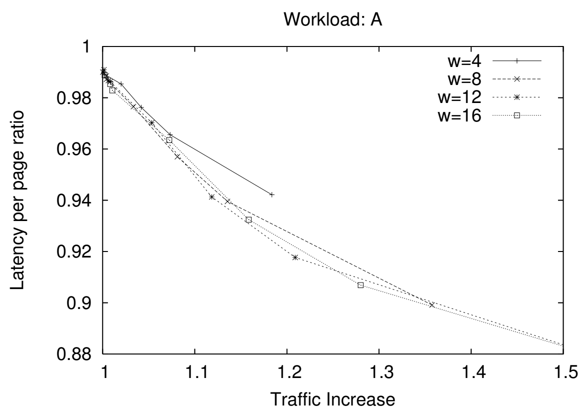
B. Selecting Parameters Values

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 [20].

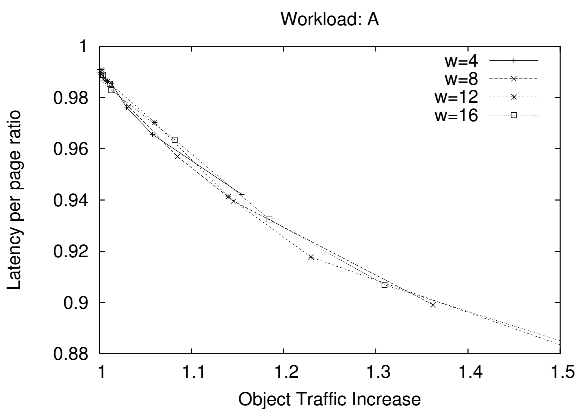
To find the best *lookahead window* size, experiments were run ranging it from 2 to 16. For the sake of clarity, only some of those values are represented in Figure 10 for the workload A and in Figure 11 for the workload B. Figure 10 shows minor performance differences under the workload A between the 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 VI the DDG algorithm under the workload A in order to have an aggressiveness similar to the DG and PPM algorithms. Under the workload B (see Figure 11), 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, it is the one with less computing complexity.

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 12 and 13 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 14 shows the performance achieved by the algorithms varying

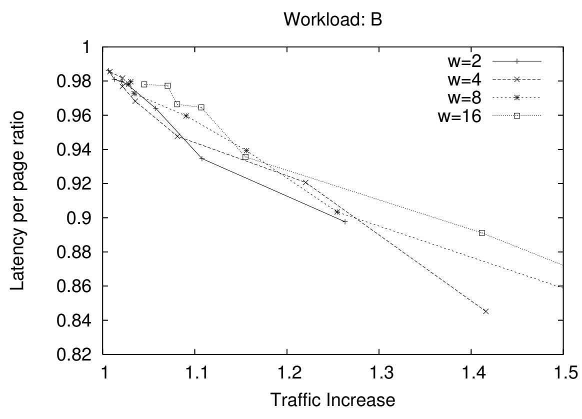


(a) Latency per page ratio as a function of Traffic increase

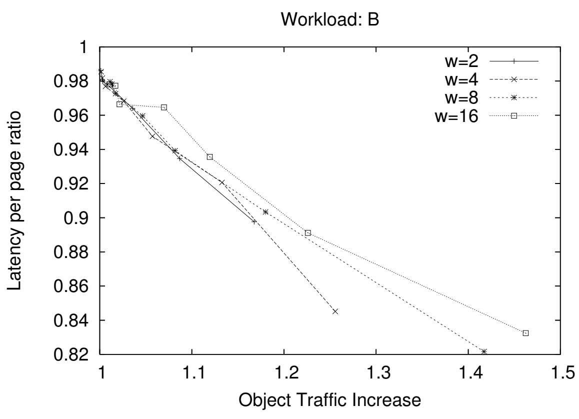


(b) Latency per page ratio as a function of Object Traffic increase

Fig. 10. Selection of the lookahead window size of the DDG algorithm under the workload A

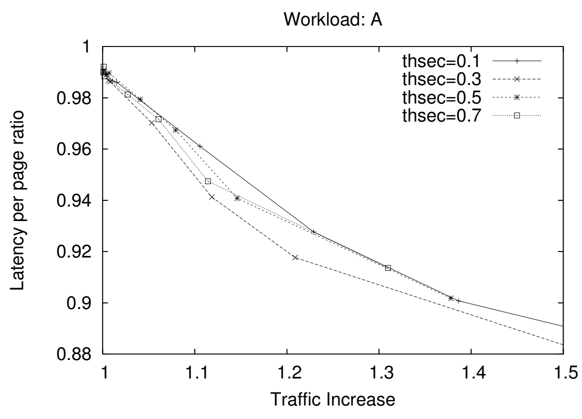


(a) Latency per page ratio as a function of Traffic increase

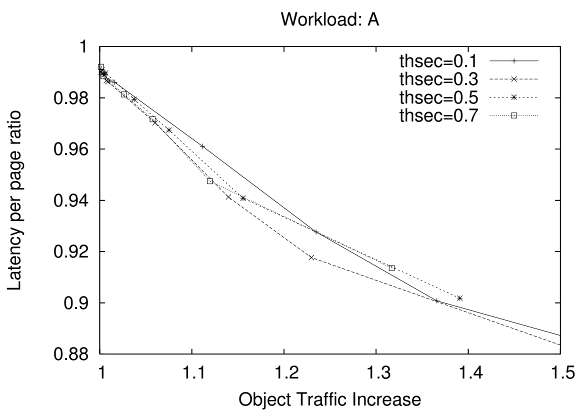


(b) Latency per page ratio as a function of Object Traffic increase

Fig. 11. Selection of the lookahead window size of the DDG algorithm under the workload B

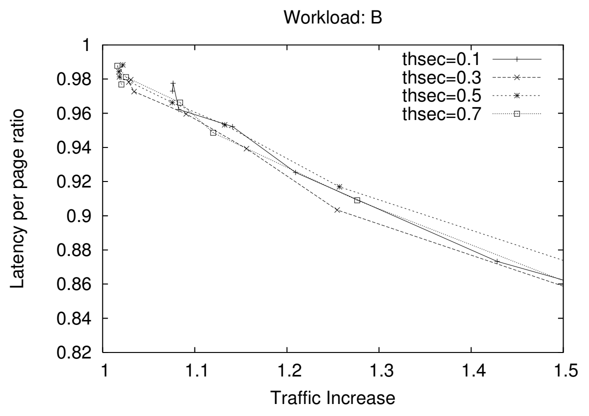


(a) Latency per page ratio as a function of Traffic increase

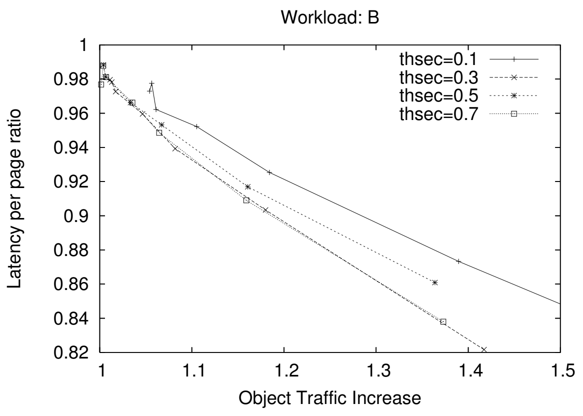


(b) Latency per page ratio as a function of Object Traffic increase

Fig. 12. Selection of the secondary threshold of the DDG algorithm under the workload A

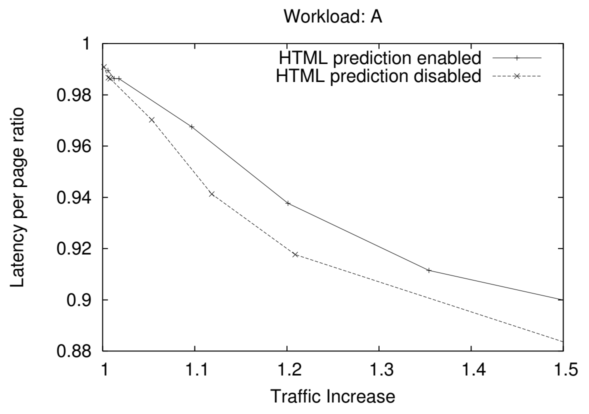


(a) Latency per page ratio as a function of Traffic increase

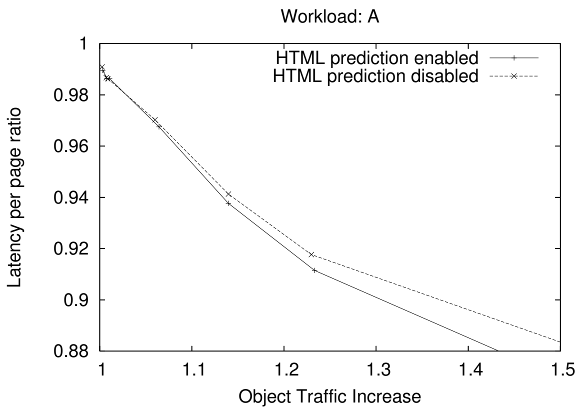


(b) Latency per page ratio as a function of Object Traffic increase

Fig. 13. 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

Fig. 14. Deciding whether to predict HTML files in the DDG algorithm under the workload A

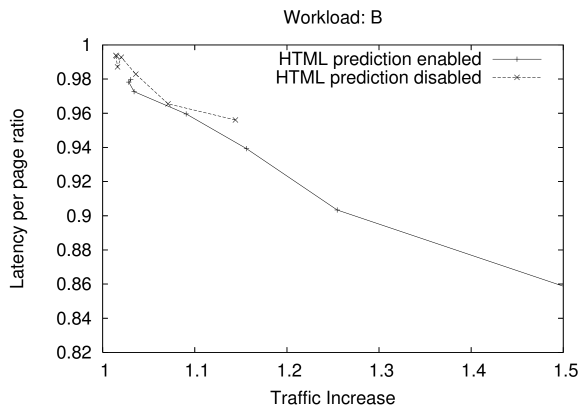
this parameter under the workload A. Looking at plot 14(a), it is better to disable the prediction of HTML objects since its curve is always below the other one. However, when taking into account the object traffic increase as the main cost metric (see plot 14(b)), it is slightly better to predict the 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 VI). 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 15, 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.

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.

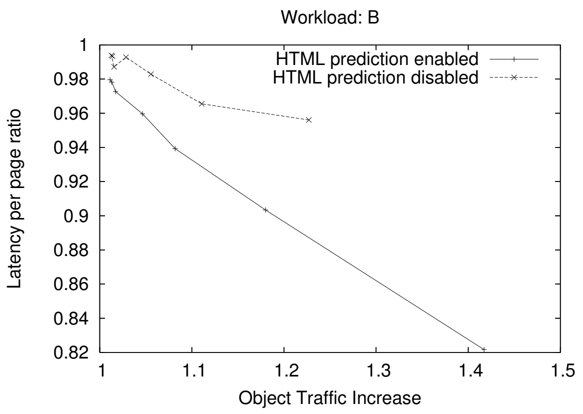
The paremeters values of DG and PPM algorithms were explored in [20]. We found that the best configuration for both workloads is to use a *lookahead window* size of 2 in DG, and a first-order Markov model in PPM.

VI. Algorithms Comparison

Once the algorithms are tuned to have the best performance, they are compared in order to check the performance differences. Figure 16(a) shows that given any traffic increase in bytes, the algorithm that provides the lowest user's perceived latency under the workload A is always the DDG. Similar

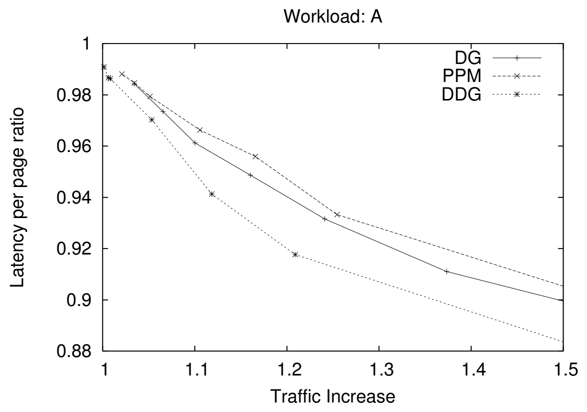


(a) Latency per page ratio as a function of Traffic increase

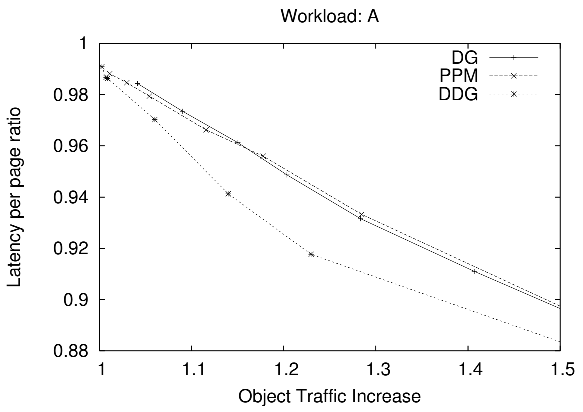


(b) Latency per page ratio as a function of Object Traffic increase

Fig. 15. Deciding weather to predict HTML files in the DDG algorithm under the workload B



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

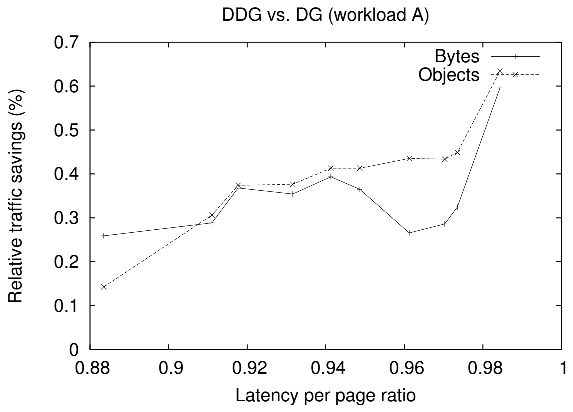


(b) Latency per page ratio as a function of Object Traffic increase

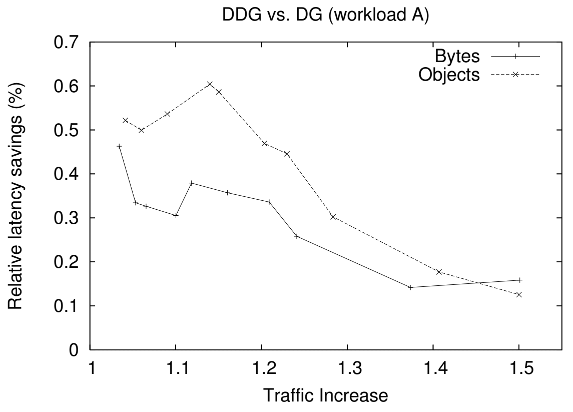
Fig. 16. Algorithms comparison under the workload A

results can be extracted when taking into account as a cost factor the traffic increase measured in amount of objects (see Figure 16(b)): DDG curve always falls below the other algorithms curves. To highlight the benefits of the new algorithm, it has been compared in Figure 17 against the DG algorithm since it performs better than the PPM. Figure 17(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 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 17(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 17(b) shows.

Performance comparison under the workload B is presented in Figure 18. Plot 18(a) shows that the selection of the best algorithm depends on which range the analysis is focused. In this context, for a traffic increase lower than around 1.15, the best algorithm is the DDG, otherwise, 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 18(b)), the DDG always outperforms clearly the DG and the PPM algorithms. Figure 19 presents the relative comparison of the DDG algorithm against the PPM. By looking at plot 19(a), one can observe that given a value of latency per page ratio, DDG requires between about 60% less and 30% more of traffic to achieve that latency reduction than the PPM. 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

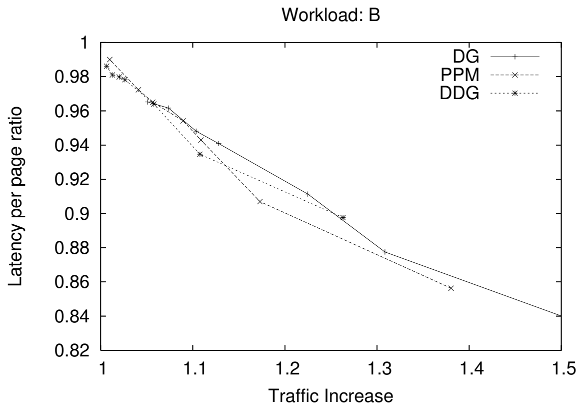


(a) Relative improvement in traffic increase as a function of latency per page ratio

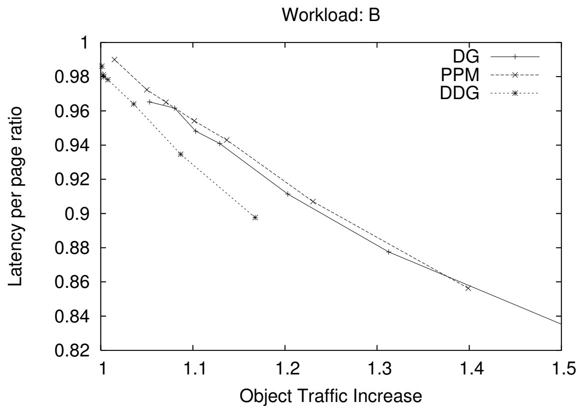


(b) Relative improvement in latency reduction as a function of traffic increase

Fig. 17. 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.

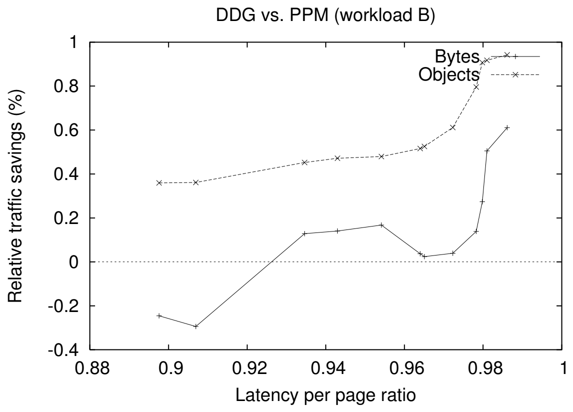


(a) Latency per page ratio as a function of Traffic increase

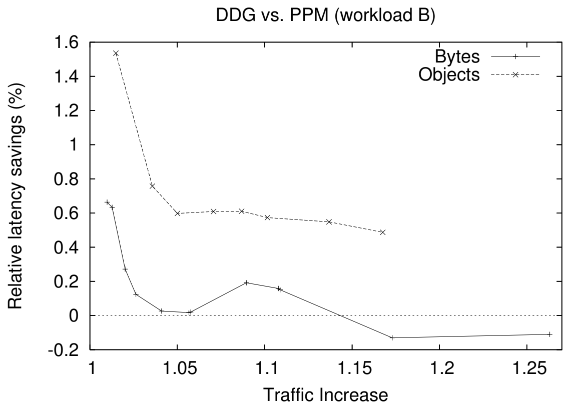


(b) Latency per page ratio as a function of Object Traffic increase

Fig. 18. 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

Fig. 19. 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.

the user's perceived latency. Plot 19(b) shows that, with the same traffic increase (in bytes), DDG algorithms reduces the user's perceived latency between around 65% more and 15% less than the PPM. When the object traffic increase is analyzed, the dashed line in the plot shows that the DDG algorithm reduces the user's perceived latency between about 50% and 150% more than the PPM.

The plots presented in this section were generated by simulating 1 Mbps users. The simulation of users with 8 Mbps of available bandwidth exhibit similar results, but their plots are not shown due to space restrictions.

VII. Conclusions

In this paper 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 objects embedded in 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's perceived latency can not 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.

Acknowledgments

This work has been partially supported by Spanish Ministry of Education and Science and the European Investment Fund for Regional Development (FEDER) under grant TSI 2005-07876-C03-01.

References

[1] V. N. Padmanabhan and J. C. Mogul, "Using predictive prefetching to improve World Wide Web latency," *Computer Communication Review*, vol. 26, no. 3, pp. 22–36, 1996.

[2] J. Griffioen and R. Appleton, "Reducing file system latency using a predictive approach," University of Kentucky, Tech. Rep., 1994.

[3] T. C. Bell, J. G. Cleary, and I. H. Witten, *Text Compression*. Prentice Hall, 1990.

[4] L. Fan, P. Cao, W. Lin, and Q. 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*, Atlanta, USA, 1999.

[5] T. Palpanas and A. Mendelzon, "Web prefetching using partial match prediction," in *Proceedings of the 4th International Web Caching Workshop*, San Diego, USA, 1999.

[6] C. Bouras, A. Konidaris, and D. Kostoulas, "Predictive prefetching on the web and its potential impact in the wide area." *World Wide Web*, vol. 7, no. 2, pp. 143–179, 2004.

[7] X. Chen and X. Zhang, "A popularity-based prediction model for web prefetching." *IEEE Computer*, vol. 36, no. 3, pp. 63–70, 2003.

[8] R. Peña-Ortiz, J. Sahuquillo, A. Pont, and J. A. 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.

[9] J. Domènech, J. Sahuquillo, A. Pont, and J. 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.

[10] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol – HTTP/1.1," 1999.

[11] J. Domènech, A. Pont, J. Sahuquillo, and J. A. Gil, "An experimental framework for testing web prefetching techniques," in *Proceedings of the 30th EUROMICRO Conference 2004*, Rennes, France, 2004.

[12] D. Fisher and G. 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.

[13] J. Domènech, J. Sahuquillo, J. A. Gil, and A. 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.

[14] 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.

[15] C. Maltzahn, K. J. Richardson, D. Grunwald, and J. H. Martin, "On bandwidth smoothing," in *Proceedings of the 4th International Web Caching Workshop*, San Diego, USA, 1999.

[16] R. Sarukkai, "Link prediction and path analysis using Markov chains." *Computer Networks*, vol. 33, no. 1-6, pp. 377–386, 2000.

[17] B. D. Davison, "Learning web request patterns," in *Web Dynamics - Adapting to Change in Content, Size, Topology and Use*. Springer, 2004, pp. 435–460.

[18] X. Dongshan and S. Junyi, "A new Markov model for web access prediction," *Computing in Science and Engineering*, vol. 4, no. 6, pp. 34–39, 2002.

[19] A. Nanopoulos, D. Katsaros, and Y. Manolopoulos, "A data mining algorithm for generalized web prefetching." *IEEE Trans. Knowl. Data Eng.*, vol. 15, no. 5, pp. 1155–1169, 2003.

[20] J. Domènech, A. Pont, J. Sahuquillo, and J. 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.