An intelligent technique for controlling web prefetching costs at the server side

Johann Márquez, Josep Domènech, José A. Gil and Ana Pont
Universitat Politècnica de València, Spain
{jmarquez,jdomenech}@ai2.upv.es;{jagil,apont}@disca.upv.es

Abstract—Prefetching is an interesting technique for improving web performance by reducing the user-perceived latency when surfing the web. Nevertheless, due to its speculative nature, prefetching can increase the network traffic and the server load. This could negatively affect the overall system performance and decrease the quality of service. To minimize and maintain under control these adverse effects, in this paper we propose an intelligent prefetching mechanism that dynamically adjusts the aggressiveness of the prefetching algorithm at the server side. To this end, we also propose a traffic estimation model that permits to accurately calculate, in the server side, the extra load and traffic generated by the prefetching. The performance evaluation study shows that our proposal effectively regulates the adverse effects of web prefetching without affecting its benefits.

I. INTRODUCTION

While web contents become more and more complex every day by offering new interesting services to users, web latency is still an issue to be improved. Latency is affected by many factors such as the network infrastructure and its interconnection elements, the use of different protocols, the generation of dynamic content and overloaded web servers, among others. As a result, the acronym for World Wide Web (WWW) is now ironically used as World Wide Wait because of the user’s frustration about the slow navigation.

Web performance has been improved thanks to the efforts made by researchers and the industry. Therefore, there have been proposed a wide range of techniques mainly focused on web latency reduction. Widely used Web architecture techniques (i.e., caching and replication) and promising solutions such as prefetching, must be taken into account.

Web prefetching takes advantage of the spatial locality property shown by the web objects to predict future requests by analyzing the user access patterns, and then using these predictions to prefetch web objects during idle times. The prefetched objects are stored in a local cache to reduce the user-perceived latency if they are finally requested by the user. Web prefetching techniques use a prediction algorithm to predict the future accesses.

Despite the benefits of prefetching [2], [3], [4], its speculative nature can increase the network traffic if the prefetched objects are never requested later by the user [5]. An aggressive prefetching can generate many extra requests overloading the web server and generating extra traffic on the overall network, hence decreasing the system performance.

In the open literature there have been proposed some approaches to regulate the prefetching actions and to control its adverse effects in order to preserve the system performance. However, to the best of our knowledge, none of them acts regulating directly the extra traffic generated by the prefetching from the server side.

Shi et al present in [6] a model to control the prefetch requests in the Proxy Cache server side. Their mechanism tries to prevent the cache pollution caused by the prefetched objects. Therefore, if the prefetched objects replace the most popular objects cached and the cache hit ratio is decremented, the mechanism reduces the prefetch request to avoid this effect.

Another proposal about prefetching regulation is presented by Wang et al in [7]. It takes into account the available residual bandwidth at the client side to decide whether to prefetch or not. To measure the available bandwidth, it sends ICMP packages to the original server. Depending on the resulting value, it will allow to prefetch the predictions, affecting negatively the client and the network resources is avoided.

A prefetch network scheme is presented by Jiang and Kleinrock in [8] in which the prediction module, set at the client side, manages a second threshold for the predicted objects, which is a function of the retrieving cost and available resources for each server.

With the raising up of the wireless networks, there are proposals to regulate the prefetching that take into account intrinsic properties of the wireless technologies, such as the signal power, the mobile device’s roaming and the mobility patterns, among others [9], [10], [11], [12].

All the above techniques regulate the prefetching at the client side individually by measuring the available resources (e.g. network bandwidth, computing resources...) for each client, which is not efficient from the overall system point of view.

Our goal is to develop an intelligent mechanism for controlling the traffic increase and the server load, able to be set at the server side (i.e. web server or proxy acting as a server) and managed by the system administrator in order to guarantee the QoS offered to the clients.

The intelligent web prefetching mechanism proposed adjusts dynamically the aggressiveness of the prediction algo-
algorithm according to the system performance. This mechanism increases or decreases the number of predictions given to generate more or less prefetch requests to the server, thus maintaining under control the traffic increase and the server load due to the prefetching.

In order to develop this adaptive prefetching we first propose a traffic estimation model that uses the available information in the server to accurately calculate the extra server load and the network traffic generated by the prefetching.

The remainder of this paper is organized as follows: Section II presents some concepts about prefetching. The simulation environment used is shown in Section III. In Section IV we describe the proposed model to estimate the traffic increase at the server side. The adaptive control mechanism is presented in Section V. Section VI shows the simulation results for the proposed technique. Finally, Section VII presents some concluding remarks.

II. BACKGROUND

Here we present some basic prefetching concepts to clarify the prefetching technique and to make the remainder sections of this paper easier to understand.

A. Prefetching Principles

Web prefetching is a technique for reducing web latency based on predicting the next future web objects to be accessed by the user and prefetching them during idle times. So, if finally the user requests any of these objects, it will be already on the client cache. This technique takes advantage of the spatial locality shown by the web objects [1].

The prefetching technique has two main components: The prediction engine and the prefetching engine. The prediction engine runs a prediction algorithm to predict the next user’s request and provide these predictions as hints to the prefetching engine. The prefetching engine handles the hints and decides whether to prefetch them or not depending on certain conditions like the available bandwidth or the idle time. Both engines can work at any component of the Web architecture (clients, proxies or servers) [1]. Figure 1 shows a generic prefetching scheme considering the prediction engine located at the web server side and the prefetching engine at the client side. The prediction engine delivers the predictions to the prefetching engine as hints attached to the server response. These hints will be prefetched during idle times or whenever there are enough available resources (i.e. bandwidth) [13], [3]. This scheme can be implemented in real world without modifying the HTTP standard, as demonstrated in [14].

B. Prefetching terminology

The Predictions (PD), also known as Hints, are the number of objects predicted by the prediction engine. Prefetch request (PR) represents the number of objects prefetched. The number of objects prefetched that are requested later by the user is the Prefetch hit (PH). The opposite of the Prefetch Hit is the Prefetch Miss (PM), which represents the number of prefetched objects that were never demanded by the user (i.e., extra traffic). Finally, User request (UR) refers to the total amount of objects requested by the user (prefetched or not), and the User request not prefetched (URnP) represents the number of objects demanded by the user that were not prefetched.

As shown in Figure 2, the set of Prefetch request (PR) is a subset of the Prediction set (PD). The result of the intersection between the User request set (UR) and Prefetch request set is the Prefetch hit subset (PH). This subset is the main factor to reduce the perceived latency. In Figure 2, A represents a User request not prefetched (URnP), which is a user request neither predicted nor prefetched. B is a Prefetch request made by the prefetching engine that is requested later by the user, thus becoming a Prefetch hit. C is a Prefetch Miss (PM) resulting from an unsuccessful prediction that was prefetched but never demanded by the user. This request becomes extra traffic and extra server load.

C. Prediction algorithms

Previous studies [15] demonstrated that the Double Dependency Graph (DDG) prediction algorithm presents, for the current web structure, a better cost-benefit relationship than others from the final user’s point of view. It achieves a noticeable latency reduction over other traditional algorithms.
Therefore, our proposal to control the prefetching traffic is designed for this algorithm.

The DDG algorithm is based on a graph that keeps track of the dependencies among the objects accessed by the user. It distinguishes two classes of dependencies: dependencies to an object of the same page and dependencies to an object of another page. The graph has a node for every object that has ever been accessed. There is an arc from one node (X) to another (Y) if, and only if, at some point in time a client accessed to node Y within w accesses to node X after node Y, where w is the lookahead window size. The arc is a primary arc if X and Y are objects of different pages, that is, either Y is an HTML object or the user accessed one HTML object between X and Y. If there are no HTML accesses between X and Y, 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.

III. EXPERIMENTAL ENVIRONMENT

In this section we describe the experimental framework used to develop and test the adaptive prefetching mechanism.

A. Simulation framework

To test and develop our proposal we used the framework presented and described in [16]. This discrete-event based simulator is a flexible tool for studying, reproducing, checking and comparing the performance of prefetching and caching techniques at any element of the Web architecture. It is a trace-driven simulator able to simulate the real user behavior, offering full result statistics and performance indexes with a low cost in terms of resource consumption.

For our study we use the following configuration: The Prediction engine, which is set at the web server, receives the user access pattern to feed the algorithm and adds the predicted hints to the server response so as to deliver them to the clients. The Prefetching engine, which is set at the client side, extracts the hints from the server responses and prefetches them only in idle times, as implemented in Mozilla browser [17].

B. Workload

The experiments were run using two different traces extracted from a Squid Proxy server at the Polytechnic University of Valencia (Spain), for 3 days from March 6th to 9th 2007. In our study we used the most popular accessed web servers. Both are Spanish news web sites (www.elpais.com, www.marca.com), and their main characteristics are shown in Table I.

IV. TRAFFIC ESTIMATION MODEL

Our intelligent prefetching mechanism works in the prediction engine, which is usually set at the server (or in a Proxy acting as server). Equation 1 shows how to calculate the extra traffic ratio (measured in objects) incurred by the prefetching, which is calculated as the objects transferred through the network when prefetching is employed divided by the objects transferred in the non-prefetching case. Nevertheless, this equation cannot be solved at the server side because in the current HTTP implementation there is no mechanism to notify the server when a prefetch hit occurs at the client side. Consequently, the number of Prefetch hits or Prefetch Misses are unknown to the server. Notice that this latter term is directly related to the traffic increase and the server overload.

\[
\Delta \text{Traffic}_{\text{obj}} = \frac{\text{Traffic}^{\text{pref}}}{\text{Traffic}^{\text{Notpref}}} = \frac{\text{Traffic}^{\text{pref}}}{\text{URnP + PR} + \text{PH}} = \frac{\text{UR} + \text{PM}}{\text{URnP + PR} - \text{PH}}
\]  

(1)

Therefore, we have developed a model to estimate this traffic increase based only on the type of requests known by the server (user request not prefetched and prefetch request). For this purpose we use the prefetch rate metric, that can be defined as:

\[
\text{PrefetchRate} = \frac{\text{PR}}{\text{URnP} + \text{PR}}
\]  

(2)

To clarify the concept of PrefetchRate and the variables used to calculate it, let’s refer to Figure 1 to describe the process of the prefetching technique. When the user requests the object A, it counts as UR at the client side. When the client requests it to the server, the request counts as a User Request not Prefetched (URnP) at the server side. Once the server has fulfilled the request and the hint B is attached to the response by the prediction engine, the prefetching engine prefetches the object B. Then, both the server and the client can identify this Prefetch Request as PR due to the a HTTP header included in the client request.

If the user requests the object B later, it is counted as an UR and also as a Prefetch Hit (PH) at the client side, since the object was already prefetched and stored in the local cache. But, since the request never reaches the server, it cannot gather accurate statistics about PH. So the server cannot calculate UR as \( UR = URnP + PH \) as the client does.

In search of a mathematical model which explains the relationship between the prefetch rate and the traffic increase, we evaluate the prefetch rate for a wide set of experiments. Each

<table>
<thead>
<tr>
<th>Trace</th>
<th>Elpais</th>
<th>Marca</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year</td>
<td>2007</td>
<td>2007</td>
</tr>
<tr>
<td>No. of Accesses</td>
<td>505868</td>
<td>423559</td>
</tr>
<tr>
<td>No. of Pages</td>
<td>20253</td>
<td>29942</td>
</tr>
<tr>
<td>Avg. objects per page</td>
<td>24</td>
<td>14</td>
</tr>
<tr>
<td>No. of Users</td>
<td>892</td>
<td>1180</td>
</tr>
<tr>
<td>Bytes transferred (GB)</td>
<td>1.48</td>
<td>2.06</td>
</tr>
<tr>
<td>Avg. objects size (KB)</td>
<td>3.08</td>
<td>5.10</td>
</tr>
<tr>
<td>Avg. page size (KB)</td>
<td>77.08</td>
<td>75.93</td>
</tr>
<tr>
<td>Avg. HTML size (KB)</td>
<td>30.55</td>
<td>14.52</td>
</tr>
<tr>
<td>Avg. image size (KB)</td>
<td>1.93</td>
<td>4.38</td>
</tr>
</tbody>
</table>
experiment was performed over the simulation framework described before, considering the DDG prediction algorithm with a primary and secondary threshold from 0.1 to 0.9. The workload used is the same described in Section III-B but only taking into account the accesses of the first day.

Figure 3 shows, on the horizontal axis, the prefetch rate calculated at the server side using Equation 2 and, on the vertical axis, the traffic increase at the client side calculated using Equation 1. The relationship between prefetch rate and traffic increase describes a quadratic behavior.

Using a best fit recursive Least-Squares method we represent the behavior shown in Figure 3 by a mathematical expression. The estimation traffic model obtained is mathematically expressed in Equation 3 and graphically shown in Figure 3 as \( f(x) \). Table II presents the constants values for the set of experiments obtained with the fit recursive method.

\[
\Delta \text{Traffic}_{\text{est}} = a(\text{Pref Rate})^2 + b(\text{Pref Rate}) + c
\]

(3)

The calculated model and its constants depend on the prediction algorithm used. If other prediction algorithms are considered the traffic estimation must be recalculated. Once the traffic increase is estimated at the server side, the next step is to control it.

V. INTELLIGENT CONTROL MECHANISM

In the prediction algorithms the prefetching aggressiveness can be controlled by the mentioned threshold cutoff parameter and consequently, the number of predictions given directly depends on this parameter value. By decrementing the threshold value we increase the number of predictions and, in consequence, the number of prefetch requests.

Our proposal works over the DDG algorithm by dynamically modifying the threshold to increase or decrease the number of predictions so that the traffic increase can be controlled.

The adaptive prefetching mechanism proposed is designed to reach a certain value of aimed traffic, which can be set up by the system administrator. The mechanism periodically compares the estimated traffic with the aimed traffic obtaining the error as shown in Equation 4. It then recalculates the threshold proportionally to the estimated error using Equation 5. Figure 4 illustrates the adaptive prefetching mechanism proposed.

\[
\text{error} = \Delta \text{Traffic}_{\text{aim}} - \Delta \text{Traffic}_{\text{est}}
\]

(4)

\[
\text{Threshold} = \text{Threshold} + [k \times \text{error}]
\]

(5)

The suitable value for the proportional constant parameter \( k \) in Equation 5 should be the value which takes the system to the aimed range in a reasonable period of time and keeps the system as stable as possible within that range; as a reasonable period time we consider the equivalent time to 5,000 user requests because the experiments showed that the system becomes stable after this period. To evaluate the stability of the system under different values of \( k \), we consider the minimum error variance once the system has reached the reasonable period time.

The frequency to apply the control mechanism (control period) must be assessed based on the root mean square error (RMSE) of the \( \Delta \text{Traffic}_{\text{real}} \) with respect to the \( \Delta \text{Traffic}_{\text{aim}} \). Equation 6 shows the RMSE, where \( \Delta \text{Traffic}_{\text{aim}} \) and \( \Delta \text{Traffic}_{\text{real}} \) are measured at the end of an experiment at the client side. We consider that a reasonable control period is the value which achieves a lower RMSE.

\[
\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{n} (\Delta \text{Traffic}_{\text{aim}} - \Delta \text{Traffic}_{\text{real}})^2}{n}}
\]

(6)

In search of the best parameters for the adaptive prefetching mechanism, we performed a total of one hundred and ten experiments modifying the value of \( k \) from 0.01 to 0.25 and the control period equivalent to a certain number of user requests, varying this number between 50 and 250.

Table III shows the results for each experiment considering the RMSE metric. To select the best pair combination of \( k/\text{control period} \) we looked for the lower value of RMSE. Consequently, the best pair in the trace elpais is: \( k=1.5, \text{control period}=125 \text{ UR} \); for the trace marca it is \( k=1.5, \text{control period}=175 \). Therefore, we assume values for \( k=1.5 \) and \( \text{control period}=150 \) in our study.

VI. ADAPTIVE PREFETCHING BEHAVIOUR AND PERFORMANCE

In this section we first present the environmental conditions where the experiments were performed and the metrics used for the evaluation. Secondly, we present a set of experiments to evaluate the adaptive web prefetching.
A. Prediction algorithm parameters

The parameter values used for the prediction algorithm are:

- Primary threshold = adaptive but starting at 0.8 for a conservative experiment startup.
- Secondary threshold = 0.3.

B. Performance metrics

From the taxonomy of prefetching metrics presented in [18], we selected the following metrics to evaluate the performance of our proposal:

- $\nabla \text{Latency}_{\text{Page}}$: The latency per page ratio is the ratio between the latency achieved with prefetching and the latency with no prefetching.
- $\Delta \text{Traffic}_{\text{obj}}$: defined by Equation 1.
- $\Delta \text{Traffic}_{\text{bytes}}$: Same as $\Delta \text{Traffic}_{\text{obj}}$, but measured in bytes.
- Precision: The ratio of prefetch hits to the total number of prefetched objects.
- Recall: The ratio of requested objects by the user that were previously prefetched. This metric is the prediction index that better explains the latency per page ratio.
- RMSE or the real error defined by Equation 6

C. Performance Evaluation

To evaluate the effectiveness of the adaptive prefetching mechanism, we perform two sets of experiments for each trace. The first set consisted of experiments with the non-adaptive prefetching for both traces, while the second set of experiments was performed with the adaptive prefetching mechanism.

In order to make a fair comparison between the non-adaptive and the adaptive prefetching mechanisms, the aimed traffic value to be achieved by the adaptive prefetching mechanism was set to the average traffic obtained in the experiment using non-adaptive prefetching. So, both sets of experiments generate a similar mean traffic increase, and their performance can be evaluated through the rest of the measured parameters, specially through the RMSE.

Figures 5 to 10 show the evolution over time, measured in user requests, of the experiments for the trace Elpais and the trace Marca. In these figures the traffic increase, the threshold and the RMSE can be graphically observed and
evaluated during the experiment. Concerning the trace Elpais, Figure 5 shows the non-adaptive prefetching traffic increase and its average obtained from an experiment with a fixed threshold of 0.4. In order to fairly evaluate both experiments, the traffic increase average value in this non-adaptive experiment becomes the aimed traffic to be reached by the adaptive prefetching experiment. On the other hand, Figure 6 shows the effect of the adaptive prefetching mechanism through the controlled traffic increase and its average; the variations of the dynamic threshold can be seen in Figure 7.

A secondary positive effect of the intelligent prefetching technique is that the algorithm requires less training time to reach an stable state. This effect can be observed in Figure 5, where the first approximation to the aimed traffic is done approximately at 50,000 UR in contrast to the 5,000 UR achieved by the adaptive technique in Figure 6. Similar results and effects can be seen in Figures 8 and 9 for the marca trace.

Finally, as we can appreciate in Table IV, the adaptive prefetching mechanism provides similar benefits to the non-adaptive, decreasing the page latency. Since the aimed traffic value set for the adaptive mechanism is the $\Delta Traffic_{obj}$ average from the non-adaptive, the cost for both techniques is almost the same. Precision and recall are hardly affected by the control mechanism. The key metric to evaluate the performance of the adaptive control mechanism is the $RMSE$ because it calculates the traffic increase deviation from the aimed traffic. A higher $RMSE$ value means that the traffic is far from the aimed traffic, while a smaller value indicates that the traffic increase is lower. Therefore, a smaller value is always desirable when control is applied. The adaptive prefetching experiment results show a $RMSE$ value at least three times lower in comparison to the non-adaptive experiment.
Through a wide set of experiments we have mathematically and graphically demonstrated the effectiveness of the estimation model and the adaptive control mechanism when regulating the traffic increase and the server load, hence guaranteeing the overall QoS.

Applying the proposed adaptive mechanism, the prefetching technique can be also improved since the prediction algorithm requires neither a long period to reach a stable state nor extra resources. Since our proposal proves that the negative effects of web prefetching can be controlled even at the server side, the use of prefetching can be safely spread among users and system administrators.

**REFERENCES**


**TABLE IV**

<table>
<thead>
<tr>
<th>Trace</th>
<th>Metrics</th>
<th>Pref non-adap</th>
<th>Pref adap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elpais</td>
<td>(\nabla Latency_{Page}) [%]</td>
<td>7.8</td>
<td>7.71</td>
</tr>
<tr>
<td></td>
<td>(\Delta Traffic_{obj}) [%]</td>
<td>11.9</td>
<td>11.7</td>
</tr>
<tr>
<td></td>
<td>Precision [%]</td>
<td>40.23</td>
<td>40.67</td>
</tr>
<tr>
<td></td>
<td>Recall [%]</td>
<td>6.16</td>
<td>5.94</td>
</tr>
<tr>
<td></td>
<td>Avg. Threshold</td>
<td>0.4</td>
<td>0.4317</td>
</tr>
<tr>
<td></td>
<td>RMSE</td>
<td>0.1376</td>
<td>0.0363</td>
</tr>
</tbody>
</table>

| Marca  | \(\nabla Latency_{Page}\) [%] | 14.1          | 13.49     |
|        | \(\Delta Traffic_{obj}\) [%] | 53.5          | 52.76     |
|        | Precision [%]             | 28.67         | 28.69     |
|        | Recall [%]                | 21.33         | 21.19     |
|        | Avg. Threshold            | 0.2           | 0.2281    |
|        | RMSE                      | 0.1931        | 0.0758    |

Fig. 10. Marca Adaptive Threshold