# Speculative validation of web objects for further reducing the user-perceived latency

Josep Domenech, Jose A. Gil, Julio Sahuquillo, Ana Pont

Department of Computing Engineering (DISCA)
Universitat Politecnica de Valencia. Spain
jdomenech@upvnet.upv.es; {jagil,jsahuqui,apont}@disca.upv.es

**Abstract.** Web caching techniques reduce user-perceived latency by serving the most popular web objects from an intermediate memory. In order to assure that reused objects are not stale, conditional requests are sent to the origin web servers before serving them. Most of the server responses to the conditional requests ratify that the object remains valid and, as a consequence, they do not include the object itself. Therefore, the object transfer time is completely saved when the object is still valid. However, the round-trip time (RTT) of these short responses cannot be saved. This time represents an important fraction of the response time in the current Internet scenario and makes conditional requests save less perceived latency than when they were proposed.
This paper proposes an approach to reduce the amount of conditional requests needed to maintain web cache consistency, thus completely saving both mentioned times (transfer and RTT) taken by such requests. To this end, our system uses a speculative approach similar to the one used in web prefetching which pre-sends freshness labels instead of web objects. The proposed technique has been evaluated using current and representative web traces. Experimental results show that the proposal dramatically reduces up to 55% of both the user-perceived latency and the amount of requests that the server receives.

## 1 Introduction

The latency perceived by users when they download a web page is affected by different factors like the availability of bandwidth between clients and servers, the server processing time of the request, the round-trip time and the size of the objects composing the page. Since the inception of the Web, many research efforts have concentrated on reducing the perceived latency by improving the infrastructure and by hiding the underlying latencies.

Web caching is based on the fact that users often request objects that were previously requested in the past. Although an object is in the local cache, a conditional request to the origin web server may be required to confirm that the reused object is not stale. In the current web, an important amount of the server responses to such requests certify that the object remains valid. In such a case, responses become much shorter since there is no need to resend the whole object

again, hence the fact that the object transfer time is removed. However, the user-perceived latency may not be significantly improved since the round-trip time (RTT) cannot be avoided.

Web prefetching is a technique that attempts to reduce the perceived latency by processing a user request before the user actually makes it. In current prefetching systems, the web server keeps track of the user's accesses *to predict* their next requests. Then, the server submits the hint list to the client, which will download the predicted objects in advance.

This paper proposes a new technique that aims to save conditional requests in order to reduce the user-perceived latency by anticipating the ETag of those objects likely to be requested in the near future. To this end, we propose a system similar in the prediction part to a prefetching system, but sending in advance the ETag for those objects included in the hint list (which contains the predictions).

The proposed system extends the prediction engine of the prefetching systems by adding the ETag for each object in the prediction list. Proceeding in this way, server responses allow to *prevalidate* other local objects in cache, although the user has not asked for them. The key performance advantage comes from the fact that subsequent accesses to prevalidated objects will not need to perform a conditional request to the server. Thus, we act on the amount of requests received by the web server and the latency perceived by users. Experimental results show that the prevalidation system dramatically reduces both the amount of requests to the server and the user-perceived latency between 45% and 58%, depending on the workload.

The remainder of this paper is organized as follows. Section 2 introduces some details on caching and prefetching in which our proposal lies. Section 3 presents the particulars of the prevalidation technique. Section 4 describes the experimental environment. Section 5 analyzes performance evaluation results. In Section 6 we discuss some related work in reducing user-perceived latency. Finally, Section 7 presents some concluding remarks.

## 2 Impact of caching and prefetching on user perceived latency

This section overviews some caching and prefetching principles. A brief discussion on downloading time issues is also included to provide some insights on performance.

**Downloading time issues** A web page is usually considered as the basic unit of a user navigation. For this reason, the time taken to download a page should be considered as the main metric to evaluate the user-perceived latency [1]. Pages are usually composed of an HTML acting as a container of an increasing number of embedded objects like images, javascript code, CSS, etc. [2, 3]. Most common user actions, i.e., clicking on a hyperlink or typing a URI, usually imply the download of an HTML. Embedded objects will only be downloaded when the HTML has been downloaded (or, at least, part of it).

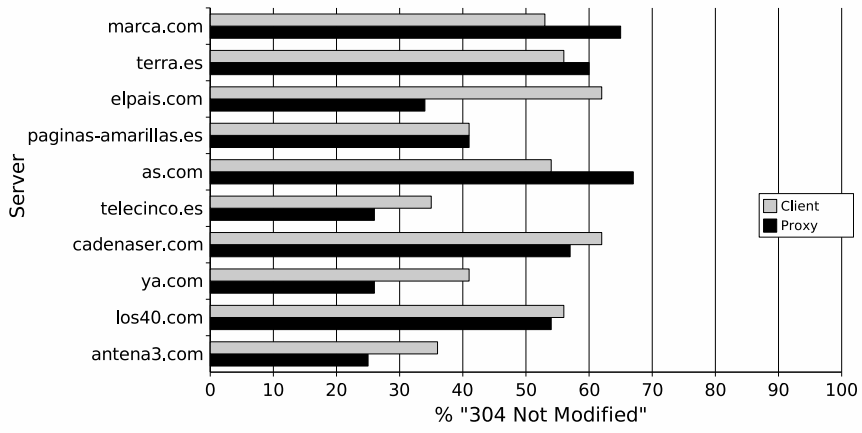



**Fig. 1.** Percentage of server responses with "304 Not Modified" response code in the top 10 most accessed servers in Spain received by the clients and by the proxy

Due to the fact that web pages are usually composed of several objects, the page downloading time is directly related to the time spent to download each object composing the web page. There are two main components in the downloading time of an object: i) the time taken by the request to reach the server plus the time taken by the response to reach the client (i.e., the round-trip time, RTT), and ii) the object transfer time (which depends on the bandwidth between the client and the server). Advances in technology have reduced both times, but mainly the object transfer time by increasing the available bandwidth. This fact has varied the weight that both components have in the overall downloading time.

Let's use an example in which a client downloads an object of 10KB. In 1996, according to [4], a typical value of bandwidth was 33Kbps and 1,130 ms of RTT. Downloading a 10KB object using the connection of 33Kbps takes 2,483 ms, assuming that there is no overload. So, the latency of downloading that object is 3,613 ms (i.e., 1,130 + 2,483). In this situation, the transfer time is 69% of the total latency.

With current technologies, typical values are 8Mbps of bandwidth and 40 ms of RTT. The transfer time of the same object is 10 ms, so the latency of downloading the object is 50 ms. In this new situation, the transfer time is just 20% of the total latency, which means that the RTT becomes the heaviest part of the current latency when navigating the web. In fact, several research studies show that the downloading time of most web objects does not depend on the file size, but mainly on the network latency [5, 6].

**Web caching details** When an object is requested, the server usually provides information about how long its response will remain valid. If the user accesses again to such object before its lifetime expires, the object can be safely used.

However, if the object is stale, the browser must check its freshness before serving it to the user. This is performed by sending a conditional request to the web server: the client requests the object to the server, but it will only be served if the object has changed since the last time the client downloaded it. This request is replied by the server with the object if it has been modified, or with a "304 Not Modified" response otherwise. With these conditional requests, the user saves the transfer time if the object has not changed. Taking into account the components of the downloading time of an object described above, this technique could highly reduce the user-perceived latency in the 1996 situation (69% in the example). However, in the current situation example, the reduction of the user-perceived latency is much more limited (only 20%).

The choice of the expiration date of an object is a critical issue. If a far date is given, it is possible that the browser shows an outdated version of the web page. If the date given is very close, the benefits of web caching are reduced. In practice, web consistency prevails over web caching benefits, so expiration dates are usually closer to the request time or even in the past. As a consequence, the number of conditional requests sent to web servers is higher in current navigations than earlier in the web [3, 7]. This remark can be also probed by analyzing Figure 1, which shows the percentage of server responses with "304 Not Modified" response code for the top 10 most accessed Spanish web servers (according to the EGM Internet audience dated at May 2007 [8]). Data represented in this figure were gathered by the Squid proxy of the Universitat Politècnica de València.

**Basic Web prefetching** In order to process user requests in advance, web prefetching is implemented by means of two main elements: the prediction and the prefetching engines. The prediction engine is aimed at guessing the following user accesses. This engine can be located at any part of the web architecture: clients, proxies, and servers, or even in a collaborative way at several elements. To predict future accesses, the most widely used option is to learn from past access patterns. The output of the prediction engine is a hint list, which is composed of a set of URIs likely to be requested by the user in a near future. Due to the fact that in many proposals the hint list is included in the HTTP headers of the server response, the time taken by the predictor to provide this list should be short enough to avoid delaying every user request and, therefore, degrading overall performance.

The prefetching engine is aimed at preprocessing those objects included in the hint list by the prediction engine. By processing the requests in advance, the user-perceived latency when the object is actually demanded is reduced. In the literature, this preprocessing has been mainly concentrated on the transfer of requested objects in advance.

Our proposal takes the prediction engine from web prefetching, However, objects are not speculatively downloaded, so the main cost of prefetching is avoided. This means that prevalidation does not significatively increase neither the network traffic nor the requests to the server.

## 3 The Prevalidation technique

We propose the prevalidation technique to further improve caching benefits by acting over the conditional request aimed to ensure the freshness of the cached objects. To this end, the proposal uses a predictive approach similar to the one implemented in web prefetching techniques. Unlike other techniques analyzed below (see Section 6) that focus on the reduction of conditional requests, our proposal addresses specifically the freshness of web resources in the client's cache.

In our system, the web server provides the client with an extended hint list generated by the prediction engine, as shown in Figure 2. It includes the freshness labels of those objects that are likely to be requested in the near future. A freshness label can be any mechanism provided by the HTTP 1.1 for determining whether a resource is stale or not (e.g., last modified time and ETag).

In the implementation evaluated in this paper, the extended hint list is sent to the client piggybacked on the response. Link headers provided by the HTTP 1.1, which are currently being used by prefetching [9], can also be used for piggybacking the extended hint list. For instance: *Link: <image.gif>; ETag="efsi"; rel=preval*. With the ETag or last modified time included in the link header, the client can validate those cached objects that are still fresh according to the data provided by the server. To do so, the client compares the last modified time (or ETag) of the object in cache to the last modified time (or ETag) provided in the extended hint list. If the user requests a prevalidated object within $t$ seconds after its validation, it is served from the cache, where $t$ is the validation lifetime. In this way, the client avoids making a conditional GET request to the origin server and waiting for the "304 Not Modified" response. Therefore, the client saves the whole object latency. Notice also that the proposal reduces the amount of requests that the server has to fulfill. Taking into account this implementation, the basic working of a prevalidation-enabled browser is described in Figure 3.

Figure 4 depicts an example of communication between client and server in a system without prevalidation and in a prevalidation-enabled system. This example shows that the server can avoid the last request and, as a consequence, its user-perceived latency by anticipating the freshness label (i.e., the ETag) of the last object.

A side effect is that the prediction engine adds extra computation time for the server to fulfill a request. However, this effect can be mitigated since the prediction can be generated in parallel to the server response in order to keep the computational load of the server unaffected.

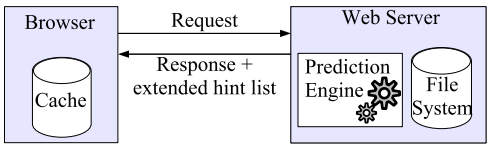



**Fig. 2.** Architecture of the proposed system

```
1: Algorithm: Working of a prevalidation-enabled browsing
2: Input: page: page demanded by the user, cache: current cache contents
3: Output: cache: updated cache contents
4: for each object in page do
5:    if object is fresh in cache then
6:       display object from cache
7:    else
8:       request object to server
9:       display object from server response
10:      for each predicted_object in predictions attached to the response do
11:         if predicted_object is stale in cache and ETag matches then
12:            extend predicted_object freshness by t seconds
13:         end if
14:      end for
15:   end if
16: end for
```

**Fig. 3.** Basic working of a prevalidation-enabled browser

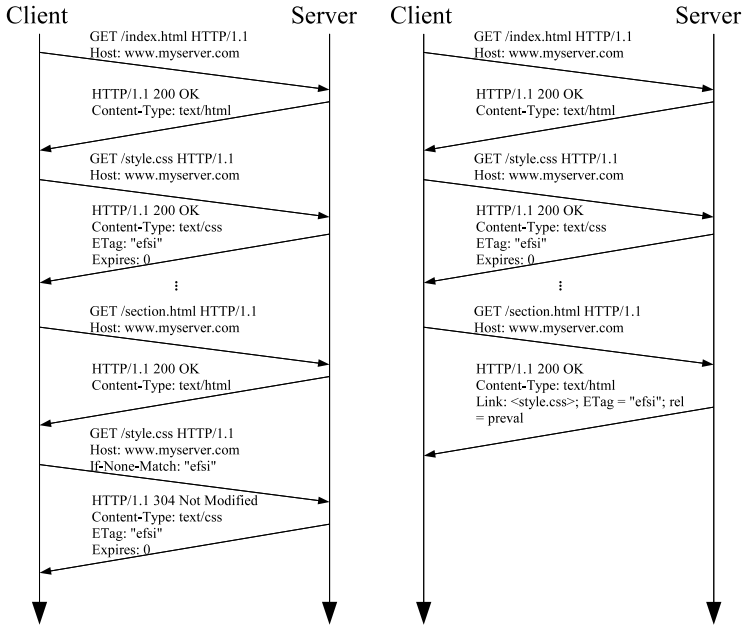
Finally, as it is known, many origin servers force the clients to send conditional requests mainly for statistics purposes; for instance, when keeping track of the users that see a banner. Nevertheless, this is not an obstacle for the proposal since, in such cases, those objects could be hosted at a different server (which is the current trend) or the prediction algorithm could have a black list with those objects that are not allowed to be predicted.

To sum up, we claim that the proposed mechanism has a double positive effect. First, it directly benefits the user whose client implements the prevalidation. Second, it indirectly benefits the rest of users because the technique decreases the amount of requests to the server. Even more, as experimental results will show, the proposal does not increase significatively the network traffic between clients and servers.

## 4 Experimental Environment

**Framework** The experimental framework presented in [10] has been extended to implement and evaluate the technique proposed in this paper. The framework, originally developed to test prefetching algorithms, consists of three main parts: the server, the surrogate, and the client. The implementation combines both real and simulated parts in order to provide flexibility and accuracy.

The framework emulates a real surrogate, which is used to access an Apache web server. The surrogate, which provides no caching, acts as a predictor by adding the corresponding HTTP headers to the server response together with the extended hint list, i.e., the URIs provided by the prediction algorithm and their freshness labels. Although the prediction can be made simultaneously to the generation of the response by the server, the surrogate does not start the algorithm for predicting next user accesses until all response headers are received




(a) Without prevalidation (b) With prevalidation

**Fig. 4.** This figure shows an example of HTTP client - server communication in a system without prevalidation (a) and with prevalidation (b). In both cases, the user demands two web pages (*index.html* and *section.html*) that use the same CSS file (*style.css*). This resource can be stored in the client's cache but must be revalidated due to the *Expires: 0* header, so a conditional request is made in (a). However, in the prevalidation-enabled system, the server provides the client with an extended hint list (i.e., the *Link* header in the last response) that the client uses to prevalidate *style.css*

from the server. Since the delivery of the response to the client is delayed by the time taken by the prediction algorithm, this time is also perceived by the client and, therefore, included to compute the user-perceived latency. Notice that our implementation represents the worst-case situation, since the prediction is made after the generation of the response.

The client part represents the behavior of the users surfing the web with a prevalidation enabled browser. To model the set of users that access concurrently to a given server, the simulator is fed by current and real traces. Simulated clients obtain the results of the prediction engine from the response and prevalidate those objects still fresh in cache, as described in Section 3.

The simulator also reproduces the time at which each object was modified. These data were taken from the Squid traces described in Section 4. We assume that an object was modified every time we find a response having a "200 OK" code.

**Workload** Two current and real traces, namely *elpais* and *los40*, were used to feed the simulator and to evaluate the prevalidation technique. Since the servers

of both traces are included in the top 10 most accessed Internet media according to the EGM Internet audience study [8], therefore they are representative of a Spanish web user. The traces were obtained by filtering the accesses to the servers in the log of the Squid proxy of the Universitat Politècnica de València. Below some characteristics of these traces are summarized.

The trace *elpais* contains accesses to the www.elpais.com web site, which is the 3rd most popular Internet media according to [8]. This trace collects the navigation of 827 users and 535,430 object requests for three days in June 2008. After several tests, we found that the system is, after 300,000 requests, in a steady state, so we used these first 300,000 requests to warm up client caches and to train the prediction algorithm. Some internal ads and banners hosted in this server, whose requests might be required for statistics purposes, were found in this server. In this work, we assume the worst-case, that is, the prediction algorithm never predicts those objects, which barely represent less than 1% of the requested objects.

The trace *los40* contains accesses to the www.los40.com web site, which is 9th most popular Internet media in Spain according to [8]. This trace collects the navigation of 285 users and 242,658 object requests for one week in May 2007. After several tests, we found that the system is in a steady state after 150,000 object requests, so we used these requests to warm up client caches and to train the prediction algorithm. We found no ads or banners hosted in this server that could interfere in the results.

Both websites are rich in dynamic pages since its content is frequently changed because *elpais* and *los40* traces provide up-to-date news regarding general and music information, respectively.

**Prediction algorithm** To implement the prediction part of the prevalidation technique, the *dependency graph* (DG) algorithm proposed in [4] has been used. This algorithm has been chosen because of its good performance at predicting embedded objects, as shown in [11]. Embedded objects are good candidates to be prevalidated since they are less often modified than HTML files.

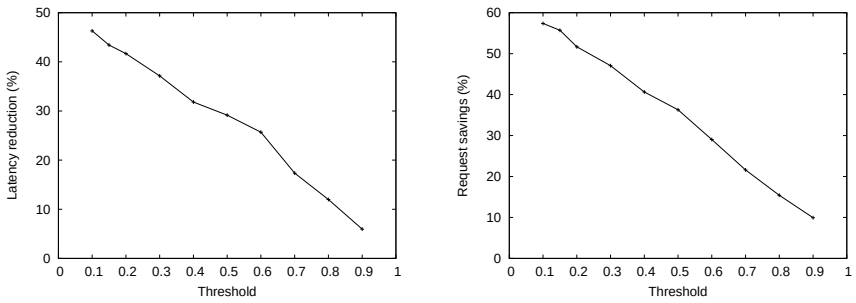
### 4.1 Performance metrics

The performance evaluation study is mainly focused on the following metrics, which represent the tradeoff of the prevalidation technique:

**Latency reduction:** Percentage of reduction in the user-perceived latency achieved by prevalidation compared to the perceived latency when it is not used. We consider as perceived latency the elapsed time between the demand of a page by the user until it is fully displayed, including all its embedded objects [1].

**Requests savings:** The number of requests, in percentage, that users do not make because the object has been prevalidated (within its validation lifetime). This metric does not include those requests saved due to cache hits.

**Traffic ratio:** The ratio of bytes transferred through the network when prevalidation is employed divided by the bytes transferred through the network when prevalidation is not used. Prevalidation increases the network traffic with re-




(a) Latency reduction (b) Request savings

**Fig. 5.** Benefits of prevalidation as a function of the prediction threshold used under workload *los40*

spect to the non-prevalidation case by sending the extended hint list. However, the network traffic is also reduced due to saved requests.

The goal of prevalidation is to achieve high latency reductions and request savings while keeping the traffic ratio close to 1.

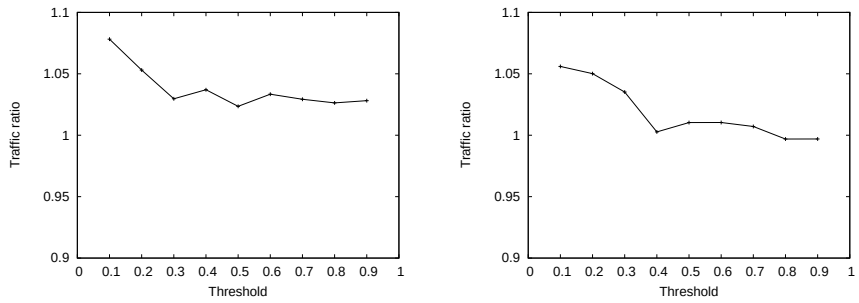
## 5 Performance Evaluation

This section presents the results of the experiments conducted to evaluate the performance of prevalidation. The base configuration for the DG algorithm is a lookahead window size of 8, a threshold of 0.3 and a prevalidation lifetime of 60 seconds. Some of these parameters are modified in the experiments with the aim to explore the performance under different scenarios.

Experimental results when evaluating the proposal are really encouraging, independently of the trace used. Under the trace *los40*, the reduction in the user-perceived latency (see Figure 5(a)) ranges from 5% to 48% depending on the prediction aggressivenes. Results are even better when quantifying the request savings metric, as Figure 5(b) shows. The percentage of requests that prevalidation saves to the server varies between 10% for a 0.9 threshold value and 59% when using a threshold value of 0.1.

The reduction in perceived latencies and number of requests to the server is achieved, under trace *los40*, at the expense of increasing the network traffic slightly. As Figure 6(a) shows, this increase is related to the prediction aggressiveness and falls between 3% and 8%, i.e., traffic ratio is between 1.03 and 1.08. This extra traffic is due to the increase of the response headers size needed to include the prevalidation hint list.

The validation lifetime is a critical parameter because its value is closely related to the consistency of the web when using prevalidation. A high value means that a prevalidated object can be served from cache for a longer time, so it is the upper bound for a user to notice the changes in web objects. However, the performance of prevalidation is hardly related to this parameter, as Figure 7 shows. This is because most objects are prevalidated few seconds before their




(a) *los40* (b) *elpais*

**Fig. 6.** Traffic ratio of prevalidation depending on the prediction threshold used

use. Common situations are those in which the prevalidation of an embedded object is included in the response of the HTML where it is contained.

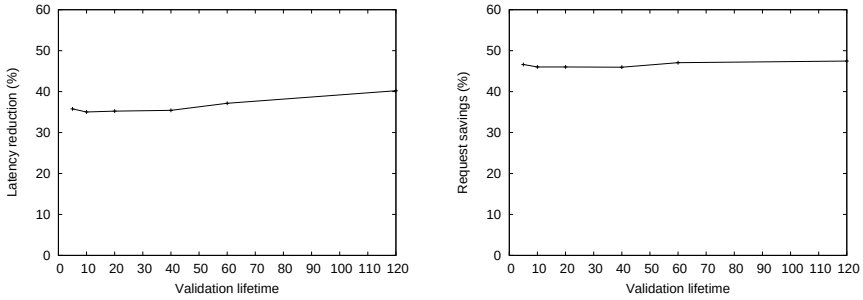
We found the best latency reduction of the prevalidation when simulating the workload *elpais*. As shown in Figure 8(a), the latency reduction achieved by the technique ranges from about 54% when using the most aggressive threshold (0.1) to almost no reduction when using the 0.9 threshold. As expected, Figure 8(b) presents similar values for the request savings metric: the reduction of requests to the server ranges from 56% to 4%.

When looking at the traffic costs of the prevalidation, one can observe in Figure 6(b) that, under some configurations (0.8 or 0.9 threshold values), this cost becomes a benefit since the network traffic is below the traffic of the non-prevalidation case (i.e., traffic ratio below 1). In this situation, the amount of bytes not sent due to saved requests (and responses) is higher than the amount of bytes transferred to send the extended hint list. In any case, the increase in network traffic in the other configurations is always below 6%.

## 6 Related Work

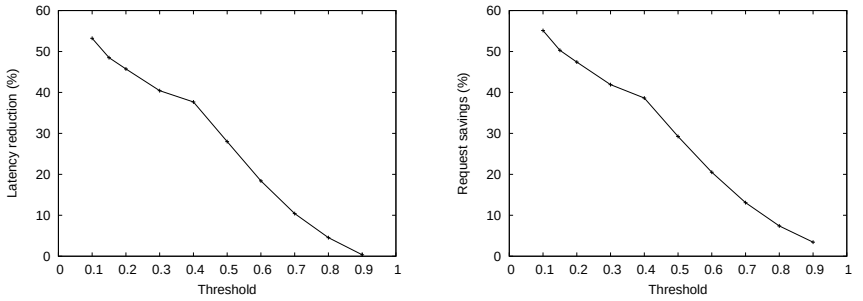
Since the beginning of the web, researchers have been working on reducing the associated latencies using a wide range of techniques. Caching the most popular objects was rapidly adopted by web browsers and extended to other elements of the architecture like proxy servers and surrogates.

Ensuring the freshness and consistency of proxy caches has been a widely researched topic. [12] proposes that servers and proxies keep track of changes in the web by grouping the objects into volumes and using version identifiers. The proxy piggybacks these identifiers in the request. Then, servers return a list of resources that have changed between the supplied and current version, so the proxy can invalidate them. The main drawback of this approach is that servers need to keep track of the evolution of every web resource. Another important issue with this approach is how to group sets of resources into volumes. To deal with volumes, [13] classifies objects according to the frequency and predictability of their changes. To provide strong consistency on caches, the server also




(a) Latency reduction (b) Request savings

**Fig. 7.** Effect on performance of changing the validation lifetime under workload *los40*




(a) Latency reduction (b) Request savings

**Fig. 8.** Benefits of prevalidation as a function of the prediction threshold used under workload *elpais*

provides explicit instructions to caches on how to handle each object, including invalidations for those objects that have changed. Other approach [14] works on reducing the delay of the validation requests for the users. This work proposes different policies for caches to proactively validate objects as they become stale. To do so, each refreshment policy associates a renewal credit with each cached object, so that only those with a positive credit are refreshed. An important disadvantage of this approach is the increase of requests due to the polling done by the proxies to the servers.

Unlike the prevalidation technique, these proposals are addressed to work between servers and proxies and are not appropriate to deal with browser's cache and, therefore, to hide the last-mile latency. However, it is possible to combine them to improve system overall performance.

## 7 Conclusions

To maintain consistency and to avoid sending stale responses to web users, web caches must confirm the freshness of the cached objects by asking the origin

servers before serving them. To do so, conditional requests are used. Unfortunately, because of the current Internet characteristics, the performance benefits due to this conditional requests have been notably reduced. This is because even in the best case –when the object has not been modified– the RTT (which currently represents a high percentage of the perceived latency) cannot be avoided.

To improve web performance and to further reduce the latency perceived by users, we proposed a novel technique that use a predictive approach. The technique permits to prevalidate objects cached without sending conditional requests to the origin server. Results show that the prevalidation technique has a double benefit: it can dramatically decrease the user-perceived latency and reduce the number of requests to the server at the expense of small overhead in network traffic.

## References


[1] Domenech, J., Gil, J.A., Sahuquillo, J., Pont, A.: Web prefetching performance metrics: A survey. Performance Evaluation **63** (2006)

[2] Domenech, J., Pont, A., Sahuquillo, J., Gil, J.A.: A user-focused evaluation of web prefetching algorithms. Computer Comm. **30** (2007)

[3] Bent, L., Rabinovich, M., Voelker, G.M., Xiao, Z.: Characterization of a large web site population with implications for content delivery. In: Proc. of the Thirteenth Int. World Wide Web Conf., New York, USA (2004)

[4] Padmanabhan, V.N., Mogul, J.C.: Using predictive prefetching to improve World Wide Web latency. Computer Comm. Review **26** (1996)

[5] Sharma, M., Byers, J.W.: How well does file size predict wide-area transfer time? In: IEEE Global Telecommunications Conf., Taipei, Taiwan (2002)

[6] Murta, C.D., Dutra, G.N.: Modeling http service times. In: IEEE Global Telecommunications Conf. (GLOBECOM), Dallas, USA (2004)

[7] Arlitt, M.F., Williamson, C.L.: Internet web servers: Workload characterization and performance implications. IEEE/ACM Trans. on Networking **5** (1997)

[8] Asociación para la Investigación de Medios de Comunicación (AIMC): Audiencia de Internet EGM. (http://www.aimc.es/03internet/31.html)

[9] Fisher, D., Saksena, G.: Link prefetching in Mozilla: A server driven approach. In: Proc. of 8th Int. Workshop on Web Content Caching and Distribution (WCW 2003), New York, USA (2003)

[10] Domenech, J., Pont, A., Sahuquillo, J., Gil, J.A.: An experimental framework for testing web prefetching techniques. In: Proc. of 30th EUROMICRO Conf., Rennes, France (2004)

[11] Domenech, J., Gil, J.A., Sahuquillo, J., Pont, A.: DDG: An efficient prefetching algorithm for current web generation. In: Proc. of 1st IEEE Workshop on Hot Topics in Web Systems and Technology, Boston, USA (2006)

[12] Krishnamurthy, B., Wills, C.E.: Piggyback server invalidation for proxy cache coherency. In: Proc. of 7th World Wide Web Conf., Brisbane, Australia (1998)

[13] Mikhailov, M., Wills, C.E.: Evaluating a new approach to strong web cache consistency with snapshots of collected content. In: Proc. of Twelfth Int. World Wide Web Conf., Budapest, Hungary (2003)

[14] Cohen, E., Kaplan, H.: Refreshment policies for web content caches. Computer Networks **38** (2002)