Delfos: the Oracle to Predict Next Web User's Accesses

B. de la Ossa, J. A. Gil, J. Sahuquillo and A. Pont
Department of Computer Engineering, Polytechnic University of Valencia
Camino de Vera, s/n, 46022 Valencia (Spain)
berospe@doctor.upv.es, {jagil, jsahuqui, apont}@disca.upv.es *

Abstract

Despite the wide and intensive research efforts focused on web prediction and prefetching techniques aimed to reduce user's perceived latency, few attempts to implement and use them in real environments have been done, mainly due to their complexity and supposed limitations that low user available bandwidths imposed few years ago. Nevertheless, current user bandwidths open a new scenario for prefetching that becomes again an interesting option to improve web performance. This paper presents Delfos, a framework to perform web predictions and prefetching on a real environment that tries to cover the existing gap between research and praxis. Delfos is integrated in the web architecture without modifying the standard HTTP 1.1 protocol, and acts inserting predictions in the web server side, while prefetchs are carried out by the client. In addition, it can be also used as a flexible framework to evaluate and compare existing prefetching techniques and algorithms and to assist in the design of new ones because it provides detailed statistics reports.

1 Introduction

A lot of research effort has concentrated on techniques to improve the World Wide Web performance. Web caching and prefetching have been proposed to reduce user's perceived latency. While caching techniques are widely used in real environments, prefetching has been an interesting subject for research but few attempts to use in real environments can be found.

Prefetching techniques allow a web browser to request an object before the user asks for it. Obviously, the web browser must prefetch using accurate information in order to achieve reasonable performance that justifies the additional resources consumed (bandwidth, extra server load).

Prefetching can be done either by the web browser or by a proxy. To this end, predictions need to be provided in order to select the objects to prefetch. These predictions can be performed by the web server as described in most research works, but they can also be done by the web browser [7] or by an intermediate, e.g., a proxy [11].

The limitations on the available user's bandwidth constrained the benefits of prefetching in the past. This fact together with the difficulty of implementing these techniques without introducing changes in the current protocols have introduced a gap between academic results and available products. But the current user's bandwidth opens again new possibilities for prefetching to improve web performance.

This paper presents *Delfos*, a predicting framework integrated in a real web architecture, (i.e., the web server and the web browser). Delfos is the Spanish name of Delphi, the famous oracle of Apollo perched on the sides of Mt. Parnassos where ancient Greeks went to know the future. It implements prediction which is used to perform prefetching with no modification in the HTTP protocol, making it suitable to use with current browsers, web servers and protocols. *Delfos* considers predictions done on the web server side and prefetchs done on the client side, as current Gecko-based web browsers do. It also provides detailed statistic reports which permit to evaluate the performance of either the prediction engine, the prefetching engine or both, helping in the design of new and more efficient algorithms and structures.

Intensive research work has been published focusing on prediction and prefetching algorithms, but few of them include performance comparison results among the different proposals by using simulation or emulation tools. The main advantage of using these tools is their flexibility and speed providing results. Unfortunately, simulators may present significant result deviations since they are abstractions of the real world. As a consequence, there is a need to develop a tool in order to gather results when running prefetching algorithms in real environments. *Delfos* also covers this lack

*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 and by La Catedra Telefonica de Banda Ancha e Internet (e-BA) from the Polytechnic University of Valencia.

in web research topics.

In summary, the main contributions of this work are: the design and implementation of a framework to perform efficiently and easily web prefetching techniques, and to provide a flexible tool to evaluate and compare the performance of these techniques under real conditions.

The remainder of this paper is organized as follows. Section 2 describes the related work. Section 3 presents and gives details of our proposed framework. Section 4 presents some experiments and working examples using *Delfos*. Finally, section 5 presents the concluding remarks.

2 Related work

In this section we make a brief review of some previous attempts to implement web prefetching techniques or to evaluate their performance.

Below we discuss a representative subset of software products with certain ability to perform web prediction or prefetching. We grouped software products in three categories: servers, proxies and clients, as summarized on Table 1. Regarding web servers, only three products were found.

Kokku et al. [13] propose NPS, a system to perform non interfering web prefetching. The system monitors the network state and adapts the parameters of the prediction and prefetching system to prevent saturation. It does not require modifications neither in the web browser nor in the HTTP protocol since it includes specific JavaScript code in the served pages to perform the actual prefetching. It does not provide hints using HTTP standard headers, as it is possible nowadays. The learning process is done only in an initial step.

The results provided by Google search sometimes include the first page of the list as a hint embedded in the HTML code. If the web browser is capable of prefetching, it may request that page in advance.

Domènech et al. [8] propose a free available framework for prefetching, it is an hybrid implementation that combines both real and simulated parts in order to provide flexibility and accuracy. It implements state of the art prediction algorithms to produce hints on the emulated web server. It also emulates web clients that prefetch the objects and provides several performance results like precision, recall and response time. This framework is very useful to test prediction and prefetching algorithms, but it is not designed for a real world usage.

There are several web proxies with prediction and prefetching capabilities. Half of them (Wcol and AllegroSurf) prefetch all the hyperlinks of a html document, unnecessarily wasting bandwidth. This massive and indiscriminate prefetching can be problematic, and has been criticized by system administrators, web designers and users. No information is available about the prediction algorithms used in the other proxies, which leads to consider they use a similar method. Packeteer SkyX Accelerator is a gateway designed to accelerate connections in the local network using an undisclosed prefetching method. Viking Server is a commercial product for Microsoft Windows operating systems that is supposed to include a proxy with prefetching capabilities.

There are several products that provide prefetching capability to the end-user web client, but all of them use the same method as the proxies do: prefetch all hyperlinks. In this case, not only the web servers' bandwidth is wasted unnecessarily, but also the client's one. The only exception is Mozilla-based products, because they prefetch only the hints provided by the web server during idle time.

Mozilla Firefox is a web browser with web prefetching capacity. Other web browsers based on the same Mozilla Foundation technologies include this capacity, for example SeaMonkey Netscape, Camino, and Epiphany. Web prefetching was first available in Mozilla Suite 1.2 (published at the end of 2002). We use Mozilla Firefox in our experiments since it already implements all the required features regarding prefetching, it is widely used by both casual and expert users, it is published with a free and open source license and its full source code is also freely available.

Google Web Accelerator is a free web browser extension available for Mozilla Firefox and Microsoft Internet Explorer on Microsoft Windows operating systems. It includes, among other features, web prefetching. It prefetchs hints included in the HTML body, but also prefetchs all the links in the pages being visited, even if no hints are provided.

FasterFox is an open and free extension for Mozilla web browsers that prefetchs all of the links on the current page during idle time.

PeakJet is a commercial product for the end user that includes several tools to improve the user access to the web. It includes a web browser independent cache with prefetching capability, based either on history or on links, therefore it can prefetch links on the current web page that were visited by the user sometime in the past or all the links on the current web page.

Another commercial product for the end user that prefetchs all the links in the page being visited, and store the objects in the browser cache is NetAccelerator. It includes the possibility to refresh the cache content in order to avoid obsolete objects.

Wei Zhang et al. [18] present the design and implementation of a modified Mozilla web browser with prediction capability that includes two prediction algorithms. The main one is based on history and uses the Prediction by Partial Matching algorithm (PPM) [17]. In the case this one provides few hints, another algorithm based on the page content is additionally used.

Table 1. Known software with prediction or prefetching capabilities

Type	Name	Description
Server	NPS	Non-interfering Web Prefetching System
	Google Search	Search results often embed hints on HTML
	JosepDom	Benchmarking framework (emulator-simulator)
Proxy	Wcol	Prefetchs all links
	Squid-prefetch	Prefetchs all links (small Perl script)
	AllegroSurf	Prefetchs all links
	Paketeer SkyX Accelerator	Prefetchs links
	Robtex Viking Server	Prefetchs links
Client	Mozilla	Used in our experiments
	Google Web Accelerator	Prefetchs all links in HTML
	FasterFox	Prefetchs all links in HTML
	PeakJet 2000	Prefetchs all links or only previously visited
	NetAccelerator	Prefetchs all links
	Personalized Mozilla	Predicts and prefetchs based on history

In summary, most prediction or prefetching implementations are either proprietary or do not even attempt to implement a smart prediction algorithm. The remaining implementations are either not ready for usage on a real environment, or do not take into account both the web server and the web client overload.

Regarding to prediction algorithms, different proposals can be found in the research literature. They can be classified depending on the type of information gathered and the data structure used for the prediction: object popularity [14], Markov models [2, 16, 19], web structure [5], and Prediction by Partial Matching [11, 17, 4].

But few research works have been addressed to compare the performance between different prediction algorithms mainly because the difficulty to reproduce environments and workloads [7]. Two algorithms based on Markov models, proposed by Zukerman [19] and by Bestavros [2], are compared in [1]. The comparison is only performed at the algorithmic level, without considering details related to the latency perceived by the user. Another work [3] compares two algorithms, one based on the idea of popular objects of Markatos [14] and another based on a variation of Prediction by Partial Matching. These comparisons were done from the point of view of the prediction and its precision, and to the knowledge of the authors there is only a fair attempt to compare them from the user's perspective [9].

3 *Delfos* proposal

Delfos is a framework to perform prefetching in a real system. Because its flexibility, it can also be used to develop, test and evaluate prefetching techniques. The current version of *Delfos* is integrated with Apache 2 web server and Mozilla web browser, although any web server or web client is suitable to work with *Delfos*.

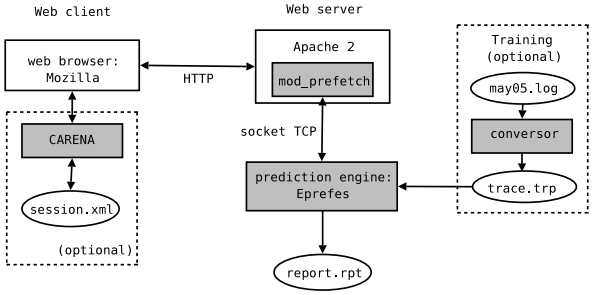

Figure 1. Framework architecture

Fig. 1 depicts the framework architecture. It comprises three main parts: the web client, the web server and the prediction engine. The web client includes a web browser with prefetching support (Mozilla) and a tool to capture and replay web navigation sessions (*CARENA*, [15]). The web server (Apache 2) includes a module (*Mod-prefetch*) to query predictions and provide them to the web client. The prediction engine (*Eprefes*) performs predictions and provides hints to the web server. Below we detail how these parts work.

3.1 Mozilla Firefox

Mozilla is able to prefetch hints if they are included in the response HTTP headers or embedded on the HTML file [12]. This prefetching mechanism was first proposed by Padmanabhan and Mogul [16], and standardized in HTTP/1.1 RFC 2616. The web server can provide one or more URIs if it considers that the user is likely to visit them soon.

These URIs or hints can be provided in three different

ways:

- in a response HTTP header:

```
Link: <ch3.html>; rel=prefetch
```

- in a 'meta' tag on the HTML header:

```
<meta HTTP-EQUIV="Link"
CONTENT="<ch3.html>; rel=prefetch">
```

- in a 'link' tag on the HTML body:

```
<link rel="prefetch" href="ch3.html">
```

When implementing prefetch in Mozilla, some interesting aspects concerning to what and when to prefetch must be considered. Only the provided URIs using the HTTP protocol are prefetched, without embedded objects. URIs that contain parameters (the query part of the URI) will not be prefetched. Prefetching will only occur when the web browser is idle. Web requests sent by Mozilla when prefetching include an additional HTTP request header, so web servers can filter those requests, for example, in case of overload. If the user clicks on a link while the browser is prefetching, the prefetch process is interrupted to satisfy the users' real request. If there was any prefetching queue, it will be discarded. The object partially downloaded will be kept on cache and completed if the user demands it. Later, when the browser is idle again, new hints can be prefetched.

3.2 *Mod-prefetch* for Apache 2

Mod-prefetch is a module for the Apache 2 web server that request hints to the prediction engine and submits them in the HTTP response headers to the web browser. See section 3.3.2 for more information.

Fig. 2 shows the communication between *Mod-prefetch* and the prediction engine. When the web server receives a request from a web browser, *Mod-prefetch* establishes a TCP socket connection to the prediction engine and sends a message to it depending on the HTTP request: If it is a standard GET request, *Mod-prefetch* sends a *predict* message request. If the response includes hints, they are added to the HTTP response headers as described in HTTP/1.1, and sends them to the web client together with the rest of the HTTP message.

3.3 *Eprefes*

Eprefes is a prediction engine designed to be used in a real environment. It runs a prediction algorithm, gathers statistics and listens for TCP connections. When it receives a prediction request, it executes the prediction algorithm and returns the resulting hints. This process has a minor impact on the response time, being currently around 1 millisecond.

To verify *Eprefes* accuracy, experiments were run both on it and on the simulator proposed by Domènech et al [8], obtaining negligible deviations.

3.3.1 Features

The main features of *Eprefes* are: it is independent of the web server, it can be controlled externally, it is modular, different parameters of the modules can be reconfigured dynamically and the code can be modified, compiled and reloaded at runtime without restarting neither the entire engine nor any module. Let's discuss them in more detail.

Eprefes is independent of the web server that queries it and the communication between both is by means of a TCP socket. This design provides several advantages. The prediction engine can be used with different models of web server. It is only required to write a module for the new web server that connects, queries and adds the hints to the HTTP headers. The prediction engine and the web server can be implemented on different languages. The web server and *Eprefes* can be located in the same or in different machines, which sometimes is preferable due to security, stability or efficiency reasons. A single prediction engine may be capable of serving several web servers, and it is not required to install it in all of them.

All the functionalities available in *Eprefes* are distributed in different modules. Table 2 gives a general view of the available modules and their purpose.

Most modules have configurable parameters, for example, the maximum number of hints that can be provided as response to a prediction request. They can be set not only in the configuration file before start up, but can also be modified at runtime by other modules, i.e., a new module that would allow to modify such parameters using a web interface or shell commands which can be very useful for adaptive policies.

Runtime code swapping allows to add new functionalities, improve performance, or fix bugs on the source code and reload the newly compiled modules into memory without restarting the server or missing the internal data.

3.3.2 Connectivity

mod-socket provides connectivity by using a TCP connection and binary format messages. When started, this module opens a socket to listen for TCP connections in the configured port number. Once a connection is established, it creates a process that waits for requests. Each request will be parsed and submitted to the *mod-serve*module. The response is conveniently packaged and sent back throughout the TCP connection. The messages received include the

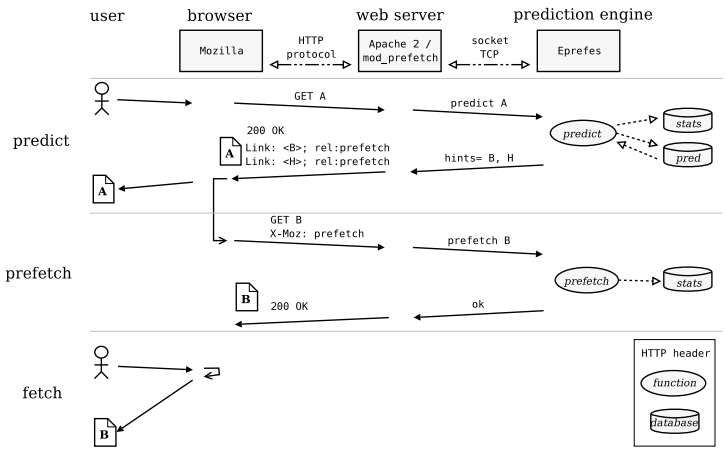

Figure 2. Communication between the web browser, the web server and the prediction engine

Subject	Name	Purpose
Connectivity	*mod-socket*	Listen for TCP connections
Serve requests	*mod-serve*	Manage requests depending on the request type
	mod-trainer	Optional. Read log files to train prediction algorithm
Statistics	*mod-stats*	Maintain variables and calculate performance indexes
	mod-report	Generate reports periodically
Prediction	*mod-palmen*	Make predictions using the Palpanas and Meldelzon's algorithm (PPM)
	mod-padmog	Make predictions using the Padmanabhan and Mogul's algorithm (DG)

Table 2. Modules in *Eprefes*

client IP address, timestamp, and object URI, MIME type and file size. On the other hand, the response message is simply a list of hints.

3.3.3 Serve requests

mod-serve manages each received request depending on the message type, that can be a prediction, a prefetch or a fetch request. If the message is a prediction request, it is redirected to the prediction module that will answer with none, one or several hints. Finally, the hints are sent back to the calling module. Besides, these hints are also notified to the statistics module.

mod-trainer is an optional module designed to provide statistics when using web server log files as input for the prediction engine instead of real web clients with prefetching capability. It intercepts the hints provided by the prediction module and generates fictitious prefetch requests. If the legitimate user later requests an object that was virtually prefetched, the module intercepts this request and converts it to a fictitious fetch request.

Currently, *mod-trainer* prefetchs all hints if the corresponding objects are not on the client cache yet without considering whether the client has idle time enough or not to prefetch them. In a real scenario, browsers may not be able to prefetch all the hints. As a consequence the results obtained when using *mod-trainer* are optimistic and suppose an upper bound.

3.3.4 Statistics gathering

mod-stats maintains variables and calculates performance indexes that can be used for comparison purposes, e.g., to evaluate the prediction accuracy and usefulness or the resources consumed by the prediction algorithm. These data are calculated and written to disk periodically without stopping the process, so all statistics are avaliable immediately.

Some statistics available are: the received requests, fetched objects, hints sent to the web server, objects that were prefetched, prefetchs that were later fetched (prefetch hits), hints that were later proved right (good predictions). All these variables are measured both in number and byte size.

Additionally, four performance indexes are calculated: precision and recall measured both in number of objects and size of objects [10].

$$Precision = \frac{Prefetch\_hits}{Prefetchs}; Recall = \frac{Prefetch\_hits}{User\_requests}$$

For all of them, the mean value and the confidence inter-

val is calculated. Performance indexes are measured using two methods. The standard one, called *EXP*, measures the values from the beginning of the measurement session. The method called *INT* calculates the indexes using only information of the last measurement interval. Proceeding in this way, the evolution of the performance indexes is shown without the interference of very old values.

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

3.3.5 Prediction algorithms

We have currently implemented the two prediction algorithms more widely referred in the literature; Dependency Graph (DG) proposed by Padmanabhan and Mogul [16] and Prediction by Partial Matching (PPM) proposed by Palpanas and Mendelzon [17], on modules *mod-padmog* and *mod-palmen* respectively.

These algorithms learn dynamically with each prediction request, so a special training phase is not required, and this information will be updated with posterior changes on the web objects, web structure or users' patterns. The prediction algorithms include parameters to limit the growth of the data structures.

mod-padmog implements the prediction algorithm Dependency Graph (DG) described by Padmanabhan and Mogul [16]. It is based on a Markov model, and considers that two objects are more related as more frequently they are requested one after the other in a window containing the last accesses for that same client.

mod-palmen implements the prediction algorithm proposed by Palpanas and Mendelzon [17]. It is based on Prediction by Partial Matching (PPM), a particular version of Markov model algorithms.

3.4 *Trainer*

Prediction algorithms require a training process long enough before being able to provide precise hints. We developed a program to train the prediction engine using log files previously captured by the web server. In this way, it is not necessary to wait until user's accesses train the prediction engine. The program can also be used for stressing and benchmarking the prediction engine or for evaluation purposes. *Trainer* is an optional tool which accepts trace files in Common Log Format or Combined Log Format, both of them are standard formats in web server software. Those trace files are a set of time-ordered lines, being one line for each HTTP request received by the web server. The information of each line includes the web client IP address, the object URI, the timestamp when the HTTP response is sent, and the size of the requested object.

Trainer reads the trace file sequentially and sends messages to the prediction engine using the same communication method that the previously described *Mod-prefetch* does. The provided hints are printed on screen and can be written to a file, which permits to compare the results obtained on different experiments.

Trace files are slightly converted before being parsed by the *Trainer*. Additionally, trace files are filtered to select the appropriate HTTP method (i.e., *GET*) and the HTTP response code (i.e., *200 OK*, *304 Not Modified* and *206 Partial Content*).

3.5 *CARENA*

CARENA [15] is a Mozilla extension to capture and replay user navigation sessions. *CARENA* captures information about the user session, which can be used later to replay or mimic the gathered user navigation. *CARENA* emulates the original user think times as these times are important to obtain precise and reliable performance results. *CARENA* is multiplatform, open source, lightweight, standards based, easily installable and usable, programmed in JavaScript and XUL. We use it to test the correct behaviour of *Delfos*.

4 Experimental results

The purpose of the experiments presented in this section is to show how *Delfos* can implement prefetching techniques and how it permits to evaluate the performance obtained. Due to severe space restrictions, results shown in this paper include only a very brief selection of the ones obtained with *Delfos*, and using only the PPM prediction algorithm. More results can be seen in the associated technical report [6].

To allow fair comparisons, *Delfos* was configured in the same way through the different experiments. Common options are: maximum of 100 hints allowed on a HTTP response, interval length of 100000 user requests and subintervals length of 5000. Regarding *mod-palmen* (PPM) specific options: threshold 0.2, maximum order 1, minimum order 1 (see section 3.3.5 for references). Previous work [9] demonstrates that those values provide relatively good ratio cost-benefit.

4.1 Performance indexes

In this section we show how *Delfos* can be used for performance evaluation of prefetching techniques using trace driven experiments. The PPM prediction algorithm was used on the first experiment. It was configured to produce reasonably good results. Our prediction engine can be fed by a real web server that receives real requests from real users. However, in order to compare the performance of

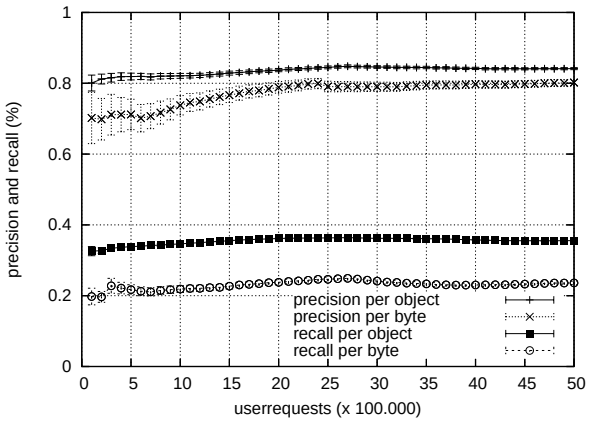

Figure 3. Precision and precision per byte, recall and recall per byte

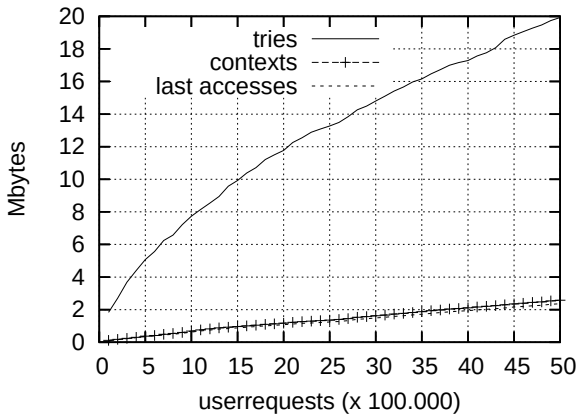

Figure 4. Memory consumed by *mod-palmen* data structures

prediction algorithms with different configurations, a reproducible workload must be used.

In the remaining experiments the prediction engine receives requests from a special trainer program that reads preprocessed web server logs. The module *mod-trainer* (described on 3.3.3) is enabled to generate prefetchs based on the predictions and hits based on the real user requests logged. Those results were obtained using the particular behaviour of *mod-trainer*, therefore they are an upper bound of the results expected in real world conditions. An experiment with five million user requests takes around ten hours to complete on a standard PC (Intel Pentium 4 3.4 GHz, 6800 bogomips, 1 GB of RAM).

The length of the experiments is measured in processed user requests. The trace file was logged in combined log format by an Apache 2 serving the web site of School of Computer Science from the Polytechnic University of Valencia. The trace file used on those experiments includes five million user requests, starts on October, 1st 2005 and ends on March, 23th 2006, it contains on average 26000 user requests per day, 15000 different objects requested, 50 gigabytes transfered in total, 285 megabytes transfered per day in average, and no previous training phase was used.

Fig. 3 shows the evolution of the precision and the recall, both of them measured per object and per byte. Decreasing the prediction algorithm's threshold increases the prediction (and hence the prefetching) aggressiveness, which also increases the cost in bandwidth usage, and also the benefit in latency reduction. Since no training phase was used, the confidence intervals are considerably large at the beginning, but decrease slowly and consistently over the experiment. As this figure shows, the possibility to see not only average values but also confidence intervals helps to detect transitional phases.

4.2 System statistics

In addition to the performance indexes, *Delfos* allows the modules that implement prediction algorithms to report statistics that may be interesting on each case, for example, those related to data structures: number of registers on a database table, nodes and arcs on a graph, total memory consumption, etc.

Fig. 4 shows total memory consumption of data structures on the experiment using PPM. Memory consumption is important in real world implementations, since an algorithm providing great precision and recall may not be suitable for real world conditions if it has high memory requirements or computation requirements.

Other system statistics of interest to evaluate the resource consumption are the number of database operations required for the prediction, and the prediction service time. For algorithms based on a tree data structure (like PPM), it's possible to measure values like the mean number of children on a tree. On Markov-based prediction algorithms like *mod-padmog* (DG) it is possible to measure values like the total number of arcs and nodes, mean nodes occurrence, mean arcs occurrence, and mean arcs probability.

Delfos can be used to discover new insights into prefetching thanks to the detailed statistics. As an example, let's briefly observe the relation between performance indexes and resource consumption (memory and CPU). The figures show that data structures are still growing when the experiments end. Instead, performance indexes like precision and recall were mostly invariant during the last part of the experiments. This means that the prediction algorithm did not improve performance indexes after an initial learning phase. Allowing unlimited learning and size of data structures did not improve precision nor recall, but data structures grew, making the algorithm slower.

5 Conclusions

In this paper we presented *Delfos*, a framework that provides web prefetching capabilities in real environments. To the knowledge of the authors, it is the first implementation for real usage that features smart prediction algorithms and provides hints using the method described on HTTP 1.1. *Delfos* is also a flexible tool that can be used either for research purposes or performance evaluation analysis.

Concerning to its usage in real environments, the prediction engine is an independent program that connects to the web server to provide hints, and a module for Apache 2 is available for this purpose. Mozilla web browser is used since it already includes the required support for prefetching. An important novelty of the proposed framework is that it does not require any modification in the standard HTTP 1.1 protocol.

In order to make it useful and suitable for research and performance evaluation, it provides detailed statistic reports and allows easy implementation and replacement of prediction algorithms, since they are isolated on independent modules in the prediction engine. Statistics include both performance indexes like precision and recall (both per byte and per object) and resource utilization.

Acknowledgments

The authors would like to thank the technical staff of the School of Computer Science from the Polytechnic University of Valencia (www.ei.upv.es) for providing us recent and customized trace files logged by the web server from that school web site.

References

[1] D. Albrecht, I. Zukerman, and A. Nicholson. Pre-sending socuments on the www: A comparative study. *Proceedings of the 16th International Joint Conference on Artificial Intelligence, Stockholm, Sweden*, 1999.

[2] A. Bestavros. Using speculation to reduce server load and service time on the www. *Proceedings of the 4th ACM International Conference on Information and Knowledge Management, Baltimore, USA*, 1995.

[3] C. Bouras, A. Konidaris, and D. Kostoulas. Predictive prefetching on the web and its potential impact in the wide area. *World Wide Web: Internet and Web Information Systems, 7, Kluwer Academic Publishers, The Netherlands*, 2004.

[4] X. Chen and X. Zhang. Popularity-based PPM: An effective web prefetching technique for high accuracy and low storage. *Proceedings of the 2002 International Conference on Parallel Processing, Vancouver, Canada*, 2002.

[5] B. D. Davison. Predicting web actions from html content. *Proceedings of the 13th ACM Conference on Hypertext and Hypermedia, College Park, USA*, 2002.

[6] B. de la Ossa, J. A. Gil, J. Sahuquillo, and A. Pont. Delfos: the oracle to predict next web user's accesses. Technical Report available at http://www.gii.upv.es/web_architecture/, Dept. of Computer Engineering, Polytechnic University of Valencia, Spain, 2007.

[7] J. Domènech, J. A. Gil, J. Sahuquillo, and A. Pont. Web prefetching performance metrics: A survey. *Performance Evaluation*, 63(9-10):988–1004, 2006.

[8] J. Domènech, A. Pont, J. Sahuquillo, and J. A. Gil. An experimental framework for testing web prefetching techniques. In *30th EUROMICRO Conference*, pages 214–221. IEEE, 2004.

[9] J. Domènech, A. Pont, J. Sahuquillo, and J. A. Gil. Cost-benefit analysis of web prefetching algorithms from the user's point of view. In *Proceedings of the 5th International IFIP Networking Conference, Coimbra, Portugal*, May 2006.

[10] J. Domènech, J. Sahuquillo, J. A. Gil, and A. Pont. About the heterogeneity of web prefetching performance key metrics. *Proceedings of the 2004 International Conference on Intelligence in Communication Systems (INTELLCOMM 04), Bangkok, Thailand*, November 2004.

[11] 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*, pages 178–187, Atlanta, USA, 1999.

[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] R. Kokku, P. Yalagandula, A. Venkataramani, and M. Dahlin. Nps: A non-interfering deployable web prefetching system. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, Palo Alto, USA, 2003.

[14] E. Markatos and C. Chronaki. A top-10 approach to prefetching on the web. *Proceedings of INET '98, Geneva, Switzerland*, 1998.

[15] I. J. Niño, B. de la Ossa, J. A. Gil, J. Sahuquillo, and A. Pont. CARENA: A tool to capture and replay web navigation sessions. In *Proceedings of the Third IEEE/IFIP Workshop on End-to-End Monitoring Techniques and Services (E2EMON'05)*, Nice, France, May 2005.

[16] V. Padmanabhan and J. C. Mogul. Using preditive prefetching to improve world wide web latency. *Proceedings of the ACM SIGCOMM'96 Conference, Palo Alto, USA*, 1996.

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

[18] W. Zhang, D. B. Lewanda, C. D. Janneck, and B. D. Davison. Personalized web prefetching in mozilla. Technical Report LU-CSE-03-006, Dept. of Computer Science and Engineering, Lehigh University, Bethlehem, USA, 2003.

[19] I. Zukerman, D. W. Albrecht, and A. E. Nicholson. Predicting users' requests on the www. In *UM '99: Proceedings of the seventh international conference on User modeling*, pages 275–284, Secaucus, NJ, USA, 1999. Springer-Verlag New York, Inc.