Abstract: The enormous potential of locality based strategies like caching and prefetching to improve web performance motivates us to propose a novel global framework for performance evaluation in scenarios where different parts of the web architecture interact. Unlike existing proposals our approach is a fast and flexible tool that allows to represent faithfully the behaviour of each element of the architecture in order to study, reproduce, evaluate and design web strategies to decrease the user’s perceived latency when surfing the web.

1. MOTIVATION

Despite of the many efforts done by the industry and the research community to improve the World Wide Web performance, the web latency perceived by the users is still a perennial issue to reduce.

The rising up of the Web architecture techniques such as web caching, prefetching and replication have become an important solution to reduce the user perceived latency. These techniques make use of the temporal, spatial and geographical locality properties of web objects to improve the web performance [1].

In the open literature, there are several proposals about the benefits of the above techniques applied at different elements of the generic Web architecture (i.e. clients, proxies, servers). Kroeger et al. in [2] suggests that the use of caching can reduce up to 26% the latency, also the use of prefetching can improve the web performance up to 57%. Furthermore, the combined use of caching and prefetching can reduce the latency perceived up to 60%. Nevertheless, Domenech et al., taking into account the current Web generation, point out a theoretical upper bound of 97% of latency reduction when prediction is done in a collaborative manner between proxies and servers [3].

The available development frameworks, test-beds and simulators in the Networking area do not take deeply into the account Web architecture including caching and prefetching techniques.

One of the most used tools in the literature is the NS-2 simulator [4] which allows to simulate at any level of the OSI Model but it only has a light module for caching and none for prefetching. NS-2 has many models for different applications but only a basic model for HTTP traffic that does not model accurately the main characteristics of these kind of applications.

Kokku et al. in [5] present a non-interfering web prefetch system between clients and servers, unfortunately there is not a caching module in its architecture.

An attempt to integrate web caching and prefetching was done by Teng et al. in an interesting proposal [6] where both techniques were applied only at the proxy server side and the used workload was generated by a synthetic workload generator. Bouras et al. in [7] present an extended study about a prefetching technique and its impact on the Proxy Cache Server in a real WAN environment (i.e. university campus). The later proposal contributes with many useful considerations (e.g. log analysis, session estimation, web object types) to take into account when prefetching is applied.

Motivated by the important potential of caching and prefetching techniques to reduce the web latency, our main goal in this paper is to propose a flexible framework to study, reproduce, check and compare the performance of caching and prefetching techniques to each element of the Web architecture using a discrete-event based simulation offering trace-driven simulation, speed and accuracy.

2. BASIC PRINCIPLES

Web caching is a widely deployed technique in the Web architecture that takes advantage of the web object’s temporal locality to reduce the user perceived latency and bandwidth consumption. Web caching stores the most popular web objects already requested by users into a pool close to the client-side to avoid requesting again the objects to the original web servers.

Web prefetching is a technique focused on web latency reduction based on predicting the next future web object to be accessed by the user and prefetching it in idle times. So, if finally the user request it, the object will be already at the client’s cache. This technique takes advantage of the spatial locality shown by the web objects.

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 decide to prefetch them or not depending on some conditions like available bandwidth or idle time. Each engine can work at any element of the Web architecture.

Both techniques, caching and prefetching, have as objective the reduction of the user perceived latency. Latency can be understand as the waiting time since the user clicks on a link until the web page or object requested can be fully displayed. Kroeger et al. [2] divide the latency into internal and external taking into account the use of a proxy server on the Web architecture.

3. DEVELOPED SIMULATION FRAMEWORK

The simulator proposed is an open framework that reproduces the generic Web architecture to study caching and prefetching techniques. Our framework proposal stems from a previous framework [8] which is a functional emulator able to model a Client-Server prefetching architecture and test diverse prediction algorithms under homogeneous conditions. Nevertheless, the main adverse effect of emulation is the long time that it takes to perform experiments and this is where our framework is an improvement.
time taken to perform an experiment. Therefore, we developed our framework as a discrete-event based simulator, fastest than emulation, which allows to set any fully scheme with clients, proxies and servers integrating web caching and prefetching techniques and able to simulate real user behavior.

The development framework covers the main components of the Web architecture such as clients, proxies, and servers; each component could work independently or perform a collaborate work among each other. Besides that, the proposed simulator could set the engines (prediction and prefetch) at any of these elements offering flexibility and facilities to reproduce and compare a plenty of proposals as well as test and propose new prefetching and caching studies.

The system uses SMPL [9], a discrete event simulation subsystem, to handle the event queue and the main actions of the components and it was developed using ANSI C.

Figure 1 shows the main components of the simulator architecture that are described in the following sections. Table 1 presents the simulator parameters which can be flexibly setup. The first column presents the name of the parameter and the second column shows what the parameter does. The way parameters work are also described in the following sections while the description of components is made.

### 3.1. Clients

The client component reproduces a standard user agent behaviour (i.e. Firefox browser), fulfilling the HTTP recommendations and being able to reproduce real user navigation traces or workload generator’s synthetic traces.

The system enables so many clients as the trace-driven simulation requires, therefore a maximum number of clients may be set via `MaxClients:Parameter` in the configuration parameters file shown in Table 1.

Each client has two SMPL servers simulating the two standard TCP sockets for connections to the web servers or proxies, so a client can request up to two objects at same time. When prefetching engine is set at the client it uses only one socket to prefetch objects in the same way as real Mozilla Firefox client performs prefetching [10].

Once the system reads the configuration settings file or receives the setting parameters via command-line arguments, each client is ready to start the simulation by reading line per line, from the trace specified by `ClientTrace:Parameter` and converting them into requests (SMPL tokens). Every line from the trace contains the request information such as time stamp, URI, linetype, among others. There are two kinds of linetypes:

- continent
- embedded

Embedded objects

etc).

The content line that has the HTML object (continent object) information which is used to start a web page request (setting the counters for a web page, page latency, etc).

- The content line that has the embedded objects for the HTML object such as images, scripts, css.

Table 1 – Simulator Setting Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Client</strong></td>
<td></td>
</tr>
<tr>
<td>MaxClients</td>
<td>The maximum amount of clients that can be simulated on the system</td>
</tr>
<tr>
<td>Intrace</td>
<td>The first session client’s trace file</td>
</tr>
<tr>
<td>Sesssiontime</td>
<td>The client’s workload trace directory path</td>
</tr>
<tr>
<td>Clearclientcache</td>
<td>It determines the length of the idle period to start a new session</td>
</tr>
<tr>
<td>NetLatency</td>
<td>It determines the network latency to be simulated</td>
</tr>
<tr>
<td><strong>Proxy</strong></td>
<td></td>
</tr>
<tr>
<td>EnableCache</td>
<td>It enables the proxy cache in the system</td>
</tr>
<tr>
<td>CachePolicies</td>
<td>It sets the caching policies and/or the HTTP cacheability rules</td>
</tr>
<tr>
<td>CacheSize</td>
<td>The object-store capacity.</td>
</tr>
<tr>
<td>Clientsatance</td>
<td>It determines the capacity to serve request simultaneously</td>
</tr>
<tr>
<td><strong>Server</strong></td>
<td></td>
</tr>
<tr>
<td>Webserverttrace</td>
<td>The file containing the object properties to be served</td>
</tr>
<tr>
<td>Maxthreads</td>
<td>It determines the capacity to serve request simultaneously</td>
</tr>
<tr>
<td><strong>Prediction engine</strong></td>
<td></td>
</tr>
<tr>
<td>Enableproxypred</td>
<td>It enables prediction at proxy</td>
</tr>
<tr>
<td>Enableserverpred</td>
<td>It enables prediction at server</td>
</tr>
<tr>
<td>Trauntrace</td>
<td>The warm-up predictor’s trace</td>
</tr>
<tr>
<td>Enableprediction</td>
<td>It chooses the prediction algorithm</td>
</tr>
<tr>
<td>Algconfigfile</td>
<td>The configuration file for the prediction algorithm</td>
</tr>
<tr>
<td><strong>Prefetching engine</strong></td>
<td></td>
</tr>
<tr>
<td>Enableclientpref</td>
<td>It enables prefetching at clients</td>
</tr>
<tr>
<td>Enableproxy pref</td>
<td>It enables prefetching at proxy</td>
</tr>
<tr>
<td>MaxLongHintList</td>
<td>The hint-database size</td>
</tr>
<tr>
<td>ClearList</td>
<td>It clears the hint-database if: new request arrives/after an idle time</td>
</tr>
<tr>
<td>OneLocalHit</td>
<td>It deletes (or not) a prefetch object if becomes a hit</td>
</tr>
<tr>
<td>StopPrefetch</td>
<td>It cancels a prefetch when a user request is demanded</td>
</tr>
<tr>
<td><strong>General Settings</strong></td>
<td></td>
</tr>
<tr>
<td>LogFile</td>
<td>The file where all events got recorded, used to generate results</td>
</tr>
<tr>
<td>SmplTrace</td>
<td>It determines the main buffer size to control the computing resources</td>
</tr>
<tr>
<td>Buffer</td>
<td></td>
</tr>
<tr>
<td>Limitrun</td>
<td>It limits the simulation by client requests</td>
</tr>
</tbody>
</table>
not empty, the next hint in the list will be prefetched until the next continent object is requested. If a new continent object is requested by the user the prefetching engine stops the prefetching to avoid interfering with the requests by demand. We consider the idle time as the time between the last embedded object request of a web page and the continent object of the next web page.

In case the idle time is not enough to prefetch all the hints, the client deletes the remainder hints to avoid prefetching out-of-date hints on next idle times, that could become useless prefetched objects.

The client has a local cache which stores the prefetched objects. To avoid cache pollution with prefetched objects in old sessions there is a mechanism to clean up the cache every new session. We consider a new session if the client has been idle for a higher period than the sessiontime:Parameter.

In order to simulate as close as possible the real system, the idle time is fully respected. Each client starts according to the time stamp of its first session in the complete trace. The trace containing the timestamps for the clients’ first session is set via Intrace:Parameter. The ClientTrace:Parameter indicates the directory where the simulator must read the traces containing the workload for each client.

To simulate diverse underlying network technologies the Netlatency:Parameter can be used to set the latency of the network used in the experiments. This value can be obtained directly from the traces or can be set with any pre-established value.

All the simulation events are registered in the logs. At the end of a simulation the logs are processed to evaluate the results under the prefetching metrics. It is possible to register the SMPL events in a log set by SmplTrace:Parameter. To store the HTTP events the simulator writes entries in the log defined by LogFile:Parameter.

3.2. Proxy

The proxy server also is a SMPL server facility with simulated TCP connections.

Always considering about simulating the reality as close as possible, the proxy respects the HTTP recommendations opening no more than two connections to the same server per client. The number of clients allowed to connect to the proxy is defined by clientatonce:Parameter.

If the caching facility is enabled in the proxy -via enablecache:Parameter- then the proxy acts as an intermediary storage between the clients and the servers. Depending on the caching policy (i.e. mapping, HTTP cacheability, replacement) selected with cache policies:Parameter the proxy handles the objects to fulfill them to the clients in the following hits.

In our simulator, the proxy and the web server use the same web object pool for saving computing resources as shown in Figure 1. Instead of using two object databases we only use one with the same objects but different fields to tag cached objects and its policy. The proxy searches, adds and deletes tags into the pool. To manage the size of the cache the proxy component compares the whole size of cached objects against the CacheSize:Parameter.

The prediction and prefetching engines can be set to predict, to prefetch or combine both parts at this component.

3.3. Server

At the beginning of a simulation run, the web server is the first component to be set and to start working.

As well as the proxy, the server has a maximum of SMPL simulated TCP connections configured by Max-threads:Parameter to fulfill the incoming request from the clients or the proxy.

The server reads a trace containing the web object properties and stores it into the web object pool. Once the pool is built, the web server can perform searches into the pool. Similar to the client’s traces, the server trace also can be set via a configuration parameter file by Webservertrace:Parameter.

The server receives the incoming request, search for the properties of the requested object into the pool and then fulfills the request and send it to the client or proxy. If the prediction feature is enabled at this element, the prediction engine adds the hints to the HTTP response.

3.4. Prediction Engine

The prediction engine can be set at the server and/or the proxy. Enableproxypred:Parameter and Enableseverpred:Parameter allow to enable or disable the predicting capability.

The engine basically works by intercepting requests, passing the requests information to a prediction algorithm and adding the returned hints by the algorithm to the HTTP response.

The prediction algorithm considered can be chosen through Enableprediction:Parameter. If the algorithm requires any extra configuration it can be set via a configuration file by Algconfigfile:Parameter.

To interact with the prediction algorithm the prediction engine uses an interface which allows to execute the different processes to make the predictions. To start and train the algorithm, in order to improve the hints accuracy, a process is called by the Init:Process. The interface allows to update the user access patterns via the UpdateStats:Process, and finally GetHints:Process attaches the provided hints to the HTTP response.

3.5. Prefetch Engine

The Prefetch engine may be set to work with the clients and/or the proxy via Enableclientpref:Parameter and Enableproxypref:Parameter respectively.

When prefetching is done by the client, the engine prefetches the hints stored in the client’s hint-list using only idle times and if a user request is made during the prefetch, the engine stops the prefetching to avoid interfering with the user request. If for some reason (e.g. study adverse effects) the engine must always prefetch, the StopPrefetch:Parameter must be set off.

Prefetching at the proxy side is less restrictive than doing it at client side, because the amount of available resources (i.e. bandwidth) is higher.

4. TESTING EXPERIMENTS

To evaluate the simulator capabilities we run a set of experiments to test different algorithms and their performance in diverse prefetching and caching schemes.

To perform a fair comparison and evaluation, we run the experiments under the same conditions:

- Simulator parameters.
We set the following general parameters to the simulator: MaxHintlist=100, Sessiontimeout=600000 ms, Clearclient-cache=on, Clearlist=on, StopPrefetch=on.

- **Workload.** We used a one day Squid Proxy trace from the Polytechnic University of Valencia (March, 2007) to get a multi-user and multi-server accesses pattern. The main characteristics of this trace can be seen in Table 2.

- **Prediction algorithms**
  We have implemented a prediction algorithm widely referred in the literature, the Dependency Graph (DG) presented by Padmanabhan and Mogul in [11], and an improved DG algorithm developed by Domenech et al. in [12] called Double Dependency Graph (DDG).
  Both algorithms are based on Markov models, and take into account that objects are more related as more frequently they are requested one after another.

- **HTTP Cacheability**
  Based on the way Squid Proxy Cache [13] works, we configure the proxy to store only the “cacheable” objects. Squid does not store any response which comes from a request with dynamic characteristics (i.e. asp, php queries,...)
  Considering the current high capacity of the storing devices and the used workload we set an unlimited storage cache size.

- **Computing resources**
  We run the simulations over a 3.05 Ghz Intel Pentium IV processor with 2 GB RAM running the GNU/Linux operating system.

- **Performance indexes**
  Based on the taxonomy presented by Domenech et al. in [14] we use the following indexes to evaluate the results:
  - Latency\textsubscript{perPage}: The latency per page ratio is the ratio of the latency that prefetching achieves to the latency with no prefetching.
  - Latency\textsubscript{perObject}: Same as Latency\textsubscript{perPage}, but measured per objects.
  - Traffic\textsubscript{bytes}: The ratio of the transferred bytes when prefetching is applied over the amount of bytes when it is not applied.
  - Precision: The ratio of prefetch hits over the total number of objects prefetched.
  - Recall: The ratio of prefetch hits over all the user request. This metric is the prediction index that better explains the latency per page ratio.

To test and validate our simulator multiple testing-experiments were run with our predecessor emulator and using our proposal. For these experiments we set a Client-Server scheme and disable the Proxy Cache. The prediction engine is set at the web server and the prefetching engine runs at the client side. The workload only considers requests to a popular news web server (www.elpais.es) gathering 181617 user request from 509 different clients that means 10703 web pages with an average of 15.96 embedded images.

Table 3 shows the percentage of the latency reduction obtained from one experiment performed to evaluate the performance improvements using prefetching. According to the obtained results we can confirm that DDG algorithm performs better offering higher benefits: up-to 16.1% against the 12.2% accomplished by the DG algorithm for the same 24.6% traffic increase.

Table 4 presents the average time and the computing resource consumption for both frameworks to perform the testing-experiment.

Table 5 shows some results for one experiment as an example of the simulator capability to gather statistics at different elements of the system, giving the possibility to evaluate different metrics and also to control the evolution of the experiment.

Some studies suggests that applying caching and prefetching in a collaborative manner can improve the web performance reaching higher bounds [2], [3], [15].

To study both techniques working together, we set a scenario composed by clients, a proxy cache and web servers. In our experiment the prefetching engine is set at each client and the prediction engine is set at the proxy using the prediction algorithm DDG fed with multi-user and cross-server patterns of the trace shown in Table 2 and given hints that are only “cacheable” objects.

To test our simulator in different scenarios and to see the effects of prefetching when clients have different types of connectivity to the proxy server we have the Netlatency:Parameter presenting three cases of study.

Ethernet Connectivity. In this first scenario clients are close to the proxy connected through an Ethernet.

Table 6 shows that the percentage of latency reduction obtained by applying only caching in the Proxy server is up-to 25.29% for web pages and up-to 28.34% for objects. Applying prefetching between the clients and the proxy the improvement in latency reduction is only a 1.31% due to the network proximity between the clients and the proxy. We conclude that if latency between clients and proxy (internal latency) is negligible the benefits obtained predicting objects in the proxy and prefetching them in the clients are also negligible.

Line Modem Connectivity. In this second scenario we consider a 56Kbps transfer rate between the clients and the proxy simulating clients connected to the network via a line modem. We consider this type of connectivity since it is the main type of connectivity in many developing countries and sometimes
the only way to access to the Internet.

The baseline and environment conditions used in this experiment are exactly the same as the used in case A.

As we can see in Table 6 the benefits obtained using both caching and prefetching techniques are quite significant permitting page latency reductions up-to 40.57% and reducing the object latency up-to 43.91% when they are applied over networks with high internal latencies thus improving the whole web performance.

Wireless Connectivity. In order to study the effect of caching and prefetching techniques in the "Mobile Web" we have model wireless network characteristics (i.e. GPRS, UMTS, 802.11b/g). Therefore we have considered different last mile latencies depending on the underlying wireless technology. Taking as a base latency the Ethernet latency, we have added 56 ms for WIFI case, 365 ms for UMTS case and 890 ms for GPRS case to the NetLatency parameter.

Table 6 shows the benefits for these technologies when caching and prefetching techniques are applied. It clearly seems that networks with high internal latencies show better performance when both techniques are applied in a collaborative manner.

We have seen that our simulator is flexible enough to consider different Web architecture scenarios and provide the most appropriate evaluation indexes to fairly evaluate performance.

5. Conclusions

To improve the WWW performance further than networking issues, it is necessary to make use of techniques such a caching, prefetching and replication.

In this paper we have presented a flexible simulation framework to study the effect of caching and prefetching techniques considering all elements of the Web architecture.

Our proposal offers full result statistics and performance indexes with low cost of resource consumption (CPU, memory, simulation time).

We have presented some test experiments to show that the simulator allows to set different prefetching and caching strategies, compare prediction algorithms under the same environment conditions, and evaluate new proposals.

The simulation framework source code and the experiments data can be found at www.gii.upv.es/web_ architecture under the GPL Licence.

REFERENCES