THE APPLICABILITY OF BALANCED ESI FOR WEB CACHING
A Proposed Algorithm and a Case of Study

Carlos Guerrero, Carlos Juiz and Ramon Puigjaner
Departamento de Ciencias Matemáticas e Informática, Universitat de les Illes Balears, Ctra Valldemossa km 7.5, 07122 Palma de Mallorca, Illes Balears, Spain

Keywords: Web performance, web cache, ESI (Edge Side Includes).

Abstract: In this paper, we present an algorithm to improve the cases in which the caching of fragments pages (ESI standard) has a worse performance than expected. A layer between the web cache and the clients has been developed and has tested over a web application using a fragment and non fragment version of web pages. Through the results of this case study we demonstrate that the proposed algorithm improve the general performance of the web application.

1 INTRODUCTION

The way that web servers deliver their contents has completely changed since the beginning of Internet. One of the major changes has been the inclusion of traditional information system applications on the www as the database ones. Nowadays, dynamic web applications are used to deliver contents to users. However, dynamic content creation places significant strain on traditional Web site architectures. This is because the same infrastructure used to generate the content it is also used for delivering the content. Generating dynamic content typically incurs: 1) user requests are dispatched to appropriate software modules that service these requests, thereby producing network overhead; 2) these software modules determine which data to fetch and present, thereby producing process overhead; 3) the disk I/O management querying the back-end database produces data overhead, and finally; 4) the assembled data needs to be formatted and delivered to the browser, thereby producing cache overhead. In short, building Web pages on-the-fly is always computationally very expensive. Therefore, the dynamic workload is higher than the workload generated by a static web server, which usually reads a file from the hard disk and sends it by the network. This is the reason why content caching has become an interesting research topic in web engineering (Cao, 1998).

Unfortunately, there are other more complex problems appearing in dynamic web caching (Veliskakis, 2005). For instance, the ratio in which web pages are changing is normally higher than the static ones and moreover, these changes are usually small. These small changes produce the page invalidation in the cache (miss) and the corresponding request to the server is sent, generating the additional respective workload. Another common case in dynamic server is the huge amount of information that the web pages are sharing. This shared information has to be generated for each page even though it is identical.

Since the minimum caching unit in web servers is a full document, the problem of regenerating web pages, in every content change, turns reusing shared information out almost an impossible task. However, we may improve the way of the web pages are cached by reducing the caching units to fragments of pages. This technique avoids the regeneration of the whole page. One way to brake down pages into fragments is given by ESI (Edge Side Includes). Unfortunately, not always the use of ESI guarantees the best response times for web pages. In this paper we present a case study of using ESI to improve web caching by using a heuristic balancing algorithm.

Thus, in section 2 a brief introduction to ESI benefits is provided. In section 3 we relate ESI use for web caching and some of the drawbacks of using ESI standard are provided. Section 4 is devoted to present our proposed balancing algorithm and in

Guerrero C., Juiz C. and Puigjaner R. (2007).
THE APPLICABILITY OF BALANCED ESI FOR WEB CACHING - A Proposed Algorithm and a Case of Study.
In *Proceedings of the Third International Conference on Web Information Systems and Technologies - Internet Technology*, pages 197-203
DOI: 10.5220/0001273501970203

section 5 we present our case study. Conclusions and future work are presented at the end of the paper.

2 EDGE SIDE INCLUDES (ESI)

ESI is a simple markup language that web application developers may use to identify content fragments for web dynamic assembly (ESI, 2006). ESI also specifies a content invalidation protocol for transparent content management across ESI-compliant solutions, such as application servers and content delivery networks. The ability to assemble dynamic pages from individual page fragments means that only non-cacheable or expired fragments need to be fetched from the origin Web site, thereby lowering the need to retrieve complete pages and decreasing the load on the Web site's content generation infrastructure.

The use of ESI maximizes its profit in scenarios where the response times are high in the server, in the network or in both cases. ESI can be use in the most layers of the web architecture: in a web cache proxy in the client side, in an inverse cache proxy in the server side, or in the edge of the network.

A study about the benefits of ESI is presented in this paper. The results of a workload over an web application using ESI cache and standard cache are explained and analyzed. This analysis reaches to the conclusion that there are cases in which the use of ESI does not provide lower response times. Therefore, we have developed a web balancing algorithm in order to address web requests either to an ESI or to a standard cache version of the requested page. This heuristic selection tries to minimize the response times from the user point of view.

3 ESI AND WEB CACHING

Edge Side Includes (ESI) accelerates dynamic Web-based applications by defining a simple markup language to describe cacheable and non-cacheable Web page components that can be aggregated, assembled and delivered. ESI enables Web pages to be broken down into fragments of different cacheability profiles. These fragments are maintained as separate elements in the application server's local cache and/or on the content delivery network. ESI page fragments are assembled into HTML pages when requested by end users. This means that much more dynamically generated content can be cached, then assembled and delivered from the edge when requested. Furthermore, page assembly can be conditional, based on information provided in HTTP request headers or end-user cookies.

The basic structure a content provider uses to create dynamic content in ESI is a template page containing HTML fragments. The template consists of common elements such as a logo, navigation bars, and other "look and feel" elements of the page. The HTML fragments represent dynamic subsections of the page. The template is the file associated with the URL the end user requests. It is marked-up with ESI language (Figure 2) that tells the cache server or delivery network to fetch and include the HTML fragments. The fragments themselves are HTML/ESI marked-up files containing discrete text or other objects. Example of an ESI template:

```
<table><tr><td>
<esi:try>
   <esi include
src="http://www.myserver.com/main.php">
</esi:try>
</td></td></table>
```

Figure 1: Example of ESI template code with the source of a fragment.

3.1 ESI Problems

The technique of breaking down pages into fragments does not always benefit the web performance. When a page does not share information with other pages there is no advantage of having cached fragments of the document. Thus, it is necessary to generate all the fragments and to assemble all them. In these cases the use of ESI produces an additional overload in comparison with the traditional way of generating web pages (Guerrero, 2006).

Moreover, breaking the web page down in a high number of parts could cause a worse response time because the main part of the fragments are not shared with other pages. Then, the changes in that page only affect to the template or to one fragment. On the contrary, by breaking the page down in a small number of parts also could cause a worse response time that the traditional way to cache pages. If fragments are too big, it is less probable to find the same fragment in other pages, and little changes could invalidate the main part of documents.

The way in which the document is fragmented takes a crucial importance from the performance point of view. But it is not as easy as make the

higher possible number of fragments; it depends in the grade fragments are sharing information, the frequency of fragment updates and the workload needed to generate the fragment. Therefore, a fragment construction is more convenient as these three parameters are higher. However, this knowledge could not be known by the web developer or web analyst during the software engineering process. Additionally, these three parameters are not constant along time. Thus, fragments have to be dynamically built using the information gathered in previous web requests.

The work presented in this paper is part of a research project to build a tool for analyzing these parameters and fragments of the different pages in order to obtain the optimal performance in any time. These fragments would be made using the information gathered along requests, and would change dynamically so long as those three parameters are changing.

4 BALANCED CACHING

4.1 Balanced Caching Algorithm

The first step to achieve the global objective of the research project is to build a tool that selects between a fragmented page and the same content of this page without fragments. This tool uses the information gathered along different requests to decide which version of the requested page is going to be delivered to the user (Alcaraz, 2006).

The proposed algorithm for managing the tool answers to each request either with a fragmented version of the page or a non-fragmented one and gathers their corresponding response times. Both versions are identified in order to compare their response times and to select the fastest (Arasu, 2003).

The behaviour of the balanced cache can be defined by and three states machine diagram (Figure 3). This state machine defines the version of the page returned for each request. For each page or identification of the page a different state machine is associated. The three states of the diagram are: (a) “Non Fragments State”, the returned page is the non fragment version of the page; (b) “Fragments State”, the returned page is the fragment version of the page; (c) “Transient State”, the returned page is the version with less number of requests (Gilly, 2007).

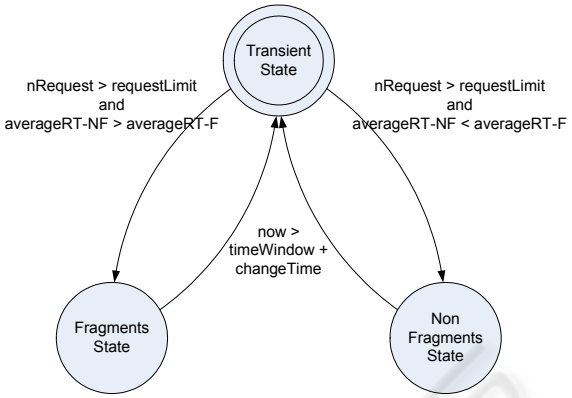

Figure 2: Balanced Caching Algorithm State Machine Diagram.

The state machine starts at “Transient State” and request `requestLimit` pages from both versions of the page, after the last request analyzes the average response time and changes the state to the state of the shorter response time version. After a period of time (`timeWindow`) either in the “Fragments State” or “Non Fragments State”, the state machine returns to the “Transient State” and requests `requestLimit` pages from the version with less access (`nRequest`), e.g. if the state before to the “Transient State” was the “Fragment State” non fragment versions of the pages would be requested, and vice versa.

Therefore the tool stores the following data for every page: the current state, the time of the last state change (`changeTime`), the accumulate response time and the number of requests (`nRequest`). These values are used to compute the average response time for both page versions (`averageRT-NF` and `averageRT-F`). When the state machine leaves the “Transient state” the tool initializes all the information gathered for both page versions.

4.2 Balanced Caching Architecture

Obviously the tool needs the fragment and non fragment version of the page in the web server. At this point of the research development, the tool is not capable break down web pages into fragments and/or to aggregate fragments. Therefore, the developer needs to develop both versions of each page. Our current task is to provide the aggregation/disaggregation technique and thereby it will not be necessary to manage fragments by web developers (Candan, 2001)).

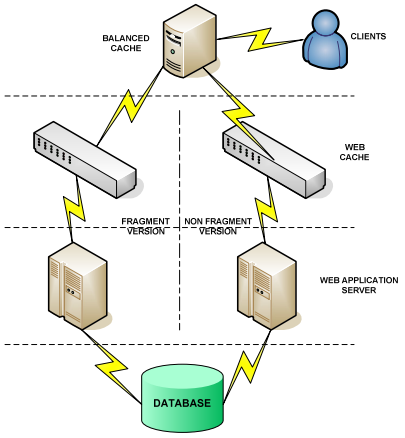

Figure 3: Web architecture diagram with balanced cache layer.

In order to minimize the development process, the tool has been built as a separate layer and not being part of a complete cache tool. Thus, it can be used with any cache software, providing tool portability. The balancing tool may be integrated by placing it between the web cache and the clients that make requests to the web server (see Figure 4).

5 CASE OF STUDY

A case of study is presented to show the improvement on average response time achieved with the balanced caching. The application example consists on a dynamic web with two versions of each page (fragmented and non fragmented). The database, application server, web cache and balanced cache layers are connected through the same local area network. The clients are connected by a wide area network to the server side of the application.

5.1 Hardware Architecture

The database and the application server share the same server, "Debian Linux" over a "Pentium II" 400 MHz, 192 MB RAM, 6 GB hard disk connected by a 100 Mbps Local area network to the web cache and the balanced cache. "MySQL" 3.23.56 and PHP 4.3.11 are also used. The web cache is "Oracle Application Server Web Cache 10g" (Oracle, 2006) running on a "Windows XP Service Pack 2" system over a "Pentium 4" 1,6 GHz, 1 GB RAM. In order to simplify the experiments the Balanced Cache and the Web Cache reside on the same machine. The client workload is generated by JMeter 2.1.1 benchmark running in similar computer to the cache server.

The Balanced Cache Algorithm, that uses sockets libraries to accept requests from the client side and make the connections to the server site, was developed in Java 1.4.

Apache JMeter is a Java desktop application designed to load test functional behaviour and measure performance (JMeter, 2006). It may be used to test performance both on static and dynamic resources. It can be used to simulate a heavy load on a server, network or object to test its strength or to analyze overall performance under different load types. It can be used to make a graphical analysis of performance or to test your server/script/object behaviour under heavy concurrent load.

5.2 Web Application

The web application for benchmarking is the official site of the "Fundació Universitat Empresa de les Illes Balears FUEIB". The web application is a content management system (CMS) developed for this specific case. The application gives the possibility to manage pages and put them on the navigation tree.

Figure 4: Web application fragments.

For every web page, a non fragmented version and a fragmented version have been built (Ramaswamy, 2004). In the fragmented version, a template and six fragments have been defined (see Figure 5). Three fragments are shared by all the pages in the site (A, E, C), two fragments are shared by the pages of the same department (B, D) and only the template and one fragment (F) are exclusive of each page.

5.3 Non Fragment and Fragment Caching Test

Even it is expected that using a fragment caching (ESI) would have better performance than a non fragmented caching, the real situation is completely different (see section 3.1). There are mean response

times for some non-fragmented pages that are shorter than the corresponding fragmented ones. Therefore, two workload tests have been done to check these cases. One of these tests only requests the non fragmented version of the pages, and the other test only requests the fragmented version of the same pages. For both tests the response times have been gathered using JMeter benchmark to emulate the user's requests. The number of different pages requested in the experiment has been 60 pages, 10 for each different department of the web. The number of concurrent emulated users has been 100, requesting 100 pages each one, without thinking time, taking approximately 60 minutes to execute each workload test.

The results of both workloading tests show that some pages have best behaviour if the fragmented version of the page is request (for instance Page I in Table 1). On the other hand (surprisingly), there are a considerable number of pages that have a better response time when the non fragmented version is requested (for instance Page II in Table 1). Lastly, there are a no very significant number of pages (less than 16%) that the response times for each workload test are practically equal (for instance Page III in table 1). In Table 1 tree response times examples of pages of each type are presented.

Table 1: Average response times for fragmented version and non fragment version of the web application of tree pages (ms).

Pages Group	Fragment version test average response time (ms)	Non fragment version test average response time (ms)
Page I	2576	4673
Page II	3850	2281
Page III	2740	2734

5.4 Balanced Caching Algorithm Test

The algorithm presented in this paper is based on the differences shown in the last section. As it was explained in Section 4.1, the algorithm tries to find out the version that responds with the best behaviour ("Transient State") and when a considerable number of requests have been compared (`requestLimit`), it decides to request only one version of the page ("Fragments State" or "Non Fragment States"). As the conditions of the servers, networks and clients change along the time, this decision has to be evaluated after some period of time (`timeWindow`).

The goal of the Balanced Caching Algorithm is to achieve the minimum response time between the request of the non fragmented version and the fragmented version of the pages. However, the price to be paid is overhead time to these expected mean response times. In order to evaluate the overhead of the algorithm and the benefits of using this algorithm respect the other two alternatives, a third workload test has been executed.

The workload test using the Balanced Caching Algorithm (B.C.A. test) needs to set some parameters up during the experiment, the number of request in the "Transient State" (`requestLimit`) and the time window to return to the "Transient State" (`timeWindow`), respectively. In this case of study `requestLimit`=3 and `timeWindow`=240 seconds are used.

5.4.1 Overhead Analysis

To evaluate the overhead added by the algorithm, we compare the average response times in this last test execution with the minimum values of the mean response times obtained in the two previous tests. These minimum values are the expected response time of the algorithm. The subtraction of these values and the mean response times of the B.C.A. test correspond to the overhead that the algorithm adds. In Table 2 we have shown the expected mean response times and the real ones of three different pages (the same pages shown in Table 1).

Table 2: Overhead time calculated with the expected average response time and the measured Balanced Caching Algorithm test.

Pages Group	Expected average response time (ms)	B.C.A. test average response time (ms)	Overhead (ms)
Page I	2576	2936	360
Page II	2281	3069	788
Page III	2734	3229	495

By performing this calculation for each page of the group of 60, the overhead for each page is obtained. Thus, the average overhead time calculated from the test is 662 ms.

The mean responses times of the 60 pages are between 2000 ms and 4000 ms. Therefore, if the mean overhead time is 662 ms, it adds a supplementary latency of 25% on the mean response times in worst cases. However, the benefits of the algorithm are considerable. For instance, if we observe Page I and Page II in Table 1 and Table 2

the mean response times of these two pages for the Balanced Caching Algorithm Test are quite better than the worst cases but the expected mean response time is not much better. In the next section the general analysis of the 60 pages are made.

5.4.2 Balanced Caching Response Time Analysis

The main goal of the case of study is to show that the average response times using the Balanced Cache are generally better than using fragment/non fragment cache even though the overhead produced by the tool.

Table 3: Mean values of the average response times for pages in "Group A" (ms).

Test	Average response time (ms)
Fragment version test	2866
Non fragment version test	4347
B.C.A. test	3528

Table 4: "Group A" Pages Speedup.

Case	Speedup
Worst case (Fragment)	0.81
Best case (Non Fragment)	1.23

In order to simplify the presentation of the experiments, results from 60 different pages are grouped in three different cases. As it was explained in Section 5.3, we distinguish three different cases: pages that obtain best response times when the fragmented version is requested (e.g. Page I in Table 1); pages that obtain best response times when the non fragmented version is request (e.g. Page II in Table 1); pages that do not present significant differences between both cases (e.g. Page III in Table 1). All the pages in the same situation that Page I are grouped as "Group A" (best response times with fragmented version of the pages); all the pages in the same situation that Page II are grouped as "Group B" (best response times with non fragmented version of the pages); finally, all the pages in the same situation that Page III are grouped as "Group C" (little differences between both versions). It is necessary to establish a value range to differences the pages that belongs to "Group C". For this analysis we have supposed this range is +/- the mean overhead time. Therefore, web pages (fragmented or not) that differ in mean response times less than overhead, are grouped together in "Group C".

First the behaviour of the pages belonging to "Group A" are analyzed. The three mean response times of the pages of each group are presented. We can observe that the Balanced Cache Algorithm improves the worst case but it has worse response times that the best case, as we had expected. In Table 4 the speedup is also shown.

The same analysis is made for pages of "Group B" in Table 5. In this group the benefits of the Caching Balanced Algorithm are smaller. The respective speedups are presented in Table 6.

Table 5: Mean values of the average response times for pages in "Group B" (ms).

Test	Average response time (ms)
Fragment version test	3589
Non fragment version test	2629
B.C.A. test	3205

Table 6: "Group B" Pages Speedup.

Case	Speedup
Worst case (Non Fragment)	0.82
Best case (Fragment)	1.12

Finally, the analysis performed for pages in "Group C" are presented in Table 7. In this case the proposed algorithm could not improve any of the both cases because the differences between the response times of the fragmented version and non fragmented version are smaller than the mean overhead as we can see in Table 8, where the speedup is presented. Thus, the general speedup of the algorithm would depend of the percentage of pages belonging to this group. In this case study the number of pages are quite small (8.3%, 5 pages of 60).

Table 7: Mean values of the average response times for pages in "Group C" (ms).

Test	Average response time (ms)
Fragment version test	3131
Non fragment version test	3016
B.C.A. test	3644

Table 8: "Group C" Pages Speedup.

Case	Speedup
Worst case (Non Fragment and Fragment)	0.83

5.4.3 Balanced Caching Response General Speedup

The global speedup of the Balanced Caching Algorithm depends on the proportion of pages that belongs to each group and could be calculated by the formula 1. If we group the speedups of "Group A" and "Group B" and consider the same number of pages in each group we could calculate the maximum proportion of pages that belong to "Group C" to have shorter response times using the Balanced Caching Algorithm. The maximum percentage of pages that belong to "Group C" to have a general Speedup above 1 is 42%. This value is enough higher to suggest that, in most cases, the use of Balanced Caching Algorithm improve the scenarios where is used only either fragmented versions of pages or only non fragmented ones.

$$Speedup = \frac{1}{(1 - f_{enhanced}) + \frac{f_{enhanced}}{Speedup_{enhanced}}}$$

$$1 \leq \frac{1}{\frac{f_{GroupA}}{Speedup_{GroupA}} + \frac{f_{GroupB}}{Speedup_{GroupB}} + \frac{f_{GroupC}}{Speedup_{GroupC}}} \quad (1)$$

6 CONCLUSIONS AND FUTURE WORK

An algorithm to improve the cases in which the caching of fragments pages (ESI standard) has a worse behaviour than expected has been presented. A layer between the web cache and the clients has been developed and also it has been tested over a web application with a fragmented and non fragmented version. The results demonstrate that the proposed algorithm improves the general performance of the application.

This algorithm is the only a first step in order to develop a future tool that brakes down and aggregate fragments, dynamically. In this way, we shall try to achieve the optimal division of the pages depending on their mean response times. This division has to be done dynamically because the networks, servers and clients' status change along the time. Therefore, the algorithm shall manage the pages using the performance information of the last requests.

Another future research subject appears with the need to know what are the conditions that web pages have to fulfil to be faster o slower using or not cached fragments.

Next future work will be to improve the algorithm by reducing the period of time that the algorithm remains in "Transient State". When the algorithm is in this state both versions of the page are requested, thereby the slower version of the page is also requested. Thus, we should dynamically adjust the `timeWindow` and `requestLimit` parameters, along the time, by getting higher `timeWindow` and shortening the `requestLimit` as long as always resolve the "Transient State" state change to the same stationary state.

ACKNOWLEDGEMENTS

The authors acknowledge the partial financial support of this research through the project code TIN2006-02265 included in the programme *Programas Nacionales del Plan Nacional de Investigación Científica, Desarrollo e Innovación Tecnológica* from *Comisión Interministerial de Ciencia y Tecnología (Ministerio de Educación y Ciencia)*.

REFERENCES

Alcaraz, S., Juiz, C., Gilly, K., Puigjaner, R., 2006. A New Bucket DiffServ Policy for Web Traffic. *Proccedings of TEMU 2006*.

Arasu, A., Garcia-Molina, H., 2003. Extracting structured data from Web pages. *In SIGMOD.*

Candan, K., D. Agrawal, D., 2001. View Invalidation for Dynamic Content Caching in Multi tiered Architectures. *Proceedings of ICDCS-2001*.

Cao, P., Zhang, J., Beach, k., 1998. Active Cache: Caching Dynamic Contents on the Web, *Middleware*, England.

Edge Side Incluyes – Estándar Specification. *http://www.esi.org.*

Gilly, K., Alcaraz, S., Juiz, C., Puigjaner, R., 2007. Service differentiation and QoS in a scalable content-aware load balancing algorithm. *Proceedings of the 40th Annual Simulationn Symposium 2007*.

Guerrero, C., Juiz, C., Puigjaner, R., 2006. Estudio de viabilidad de ESI en aplicaciones Web dinámicas. *Proceedings of IADIS WWW/Internet 2006*.

JMeter – User Manual. http*://jakarta.apache.org/jmeter/*

Oracle Application Server Web Cache 10g – Technical White Paper Information. *http://www.oracle.com/ technology/products/ias/web_cache/*

Ramaswamy, L., Iyengar, A., Liu, L., 2004. Automatic Detection of Fragments in Dynamically Generated Web Pages. *Proceedings of the Thirteenth International World Wide Web Conference.*

Veliskakis, M., Roussos, J., 2005. DOMProxy: Enabling Dynamic-Content Front-end Web Caching. *Proceedings of the 10th WCW 2005.*