Efficient Materialization of Dynamic Web Data to Improve Web Performance

Christos Bouras, Agisilaos Konidaris
Computer Engineering and Informatics Department, University of Patras, GR-26500, Patras, Greece and
Computer Technology Institute-CTI, Riga Feraiou 61, GR- 26221 Patras, Greece

Abstract

Abstract. The issues of performance, response efficiency and data consistency are among the most important ones for data intensive Web sites on the Internet today. In order to deal with these issues we analyze and evaluate a materialization policy that may be applied to data intensive Web sites. Our research relies on the performance evaluation of experimental client/server configurations. We propose a materialization policy that applies to different Web site request patterns and data update frequencies. The issue of Web and database data consistency is the driving force behind our approach. In some cases though, we prove that certain compromises to consistency can be beneficial to Web server performance and at the same time be unnoticeable to users. We evaluate the performance of our approach and compare it with other popular materialization approaches. The results of our evaluation show that the concept of "acceptable inconsistencies" between Web and Database data may be beneficial to Web servers in terms of performance.

1.0 Introduction

In our days the Web is the most popular application on the Internet. Even though most users don't know how the Web works, almost everyone experiences a phenomenon that is formally known as "Web latency". Web latency can generally be viewed as the delay between the time a user requests a Web page and the time that the Web page begins to appear on his/her computer. It can be traced down to all components of the client-server model [5, 15, 24]. One of the basic causes of latency is the Web/database interaction process that is invoked in order to keep Web data consistent, in cases where Web servers use a database as a back-end. The reason for using databases as back-ends to Web servers is their inherent efficiency in managing, upgrading and maintaining Web data [12, 13]. Even though there have been other approaches proposed (such as the integration of Web servers into databases), the independent Web and Database model, is the most popular WWW service approach in our days. This is the model that will be studied in this paper.

A solution for reducing Web latency and keeping Web/database consistency, is materialization policies. Materialization is a term used to represent the transformation of "dynamic" Web data into equivalent "static" Web data. In this paper the materialization of a dynamic Web page will cause the creation of a static instance of the page, at a certain point in time. The data contained inside the static

page are those found in the database at the time of materialization since, all the queries to the database are executed at that time.
In order to propose materialization policies for data intensive Web sites we first determine the meaning of data intensive Web sites. In this work we will refer to data intensive Web sites as sites that share two common characteristics:

- A lot of data is demanded of these Web sites at a certain point in time
- They use a frequently updated database as a back-end

We will propose a materialization policy that can be tailored to different Web site needs and then present its performance evaluation. The main aim of a materialization policy, as considered in this work, is to efficiently handle Web page request demand and at the same time perform efficient dynamic Web page materialization. We will look at Web page materialization through a fragment approach and show that it is more efficient than the classic page by page approach.
In this paper we present the generic framework of a materialization policy that relies on acceptable compromises in Web page data. We intend to show that even the most generic implementation of such a materialization policy has significant positive impact on the performance of a Web server. The framework that we propose can surely be enhanced much further. We intend to use it as the basis of our future work in the field.
The paper is structured as follows. Section 2 presents related work in the field of run-time management policies, Web page materialization, dynamic Web data and Web/Database consistency. In Section 3 we introduce the materialization policy and show how it can be applied in the general case. Section 4 presents our experimental methodology and section 5 presents our experimental results. Finally, section 6 contains the future work that can be carried out based on our work, and our conclusions.

2.0 Related Work

A lot of work has been done in the fields of specifying, executing and optimizing run-time management policies for data intensive Web sites. [1,2,3,9] present the approach for dealing with highly dynamic web data, which was followed by IBM, during the design and implementation of several Olympic games' Web sites. The approach introduces a fragmentation technique based on Server Side Includes (called fragments). This approach is the one used in this paper. The work at IBM, utilizes a triggering algorithm to propagate fragment updates to a materialization module. The basic concern is to keep Web/Database consistency. The ideas that will be presented in this paper have been inspired by the IBM approach. We contribute to their work by introducing an acceptable data freshness compromise scenario, which permits the execution of the materialization policy on the same hardware as the original Web site. [14] presents a case study for materializing and replicating dynamic data. The paper presents an interesting materialization architecture that is quite straightforward.
STRUDEL [22,16] is a Web site management tool that enables site builders to construct and manage a site declaratively. The basic idea is the separation of the Web site's data, the site's structure and the site's graphical representation. Araneus [4,21] is a management system for Web-bases. The system proposes a specific organization of Web documents and Web data inside a Web-base. This structure ensures efficient management and performance. These two projects require the design and the implementation of Web pages and Web sites with the use of specific steps and tools. This way, they can ensure performance. The projects have contributed to our work in the field of Web engineering.

Our approach can be considered more generic. We only require that Web sites must be designed with the fragment approach. Our materialization policy can then be utilized without any further constraint. Work related to the optimization of query workloads on DBMS can be found in [10,8,17,20]. This work mainly discusses view selection techniques in order to boost DBMS performance. The work can easily be utilized on the Web, in terms of choosing Web pages that should be materialized. In this paper we do not prioritize certain pages against others during the materialization procedure. We treat each page (each fragment actually) individually. It is part of our future work to add a prioritization and selection module before the materialization module. In [11] the authors shift the database view paradigm to the Web and show that it can be very useful. Their research work presents a simple cost model and experimental results that prove the efficiency of materialized Webviews (e.g. materialized Web pages). A similar approach to ours, but implemented in the field of data replication, is presented in [23]. This approach is similar to ours since it permits the stale replication of data to distributed servers in order to boost user query performance. The basic difference with the approach in this paper is that [23] is not transparent to users, since they can actually ask for a degree of acceptable staleness along with their queries.
Another interesting approach on web fragmentation and caching is presented in [7]. This approach does not require the "a priori" construction of Web pages with the use of Web fragments. It presents a novel caching approach that relies on automatic Web page fragmentation. This work can be very beneficial to our work when we move our proposal to already implemented Web pages that do not follow the Web fragment approach. We intend to utilize it in our future work.

3.0 The “Acceptable Compromise” Generic Framework

In this section we present a concept that can be beneficial to the key decision that must be made by a materialization policy. An efficient materialization policy must select materialization frequencies for all the pages in a Web site. The main goal of an efficient materialization policy should be:

- To include data that is as "fresh" as possible in materialized Web pages and
- To be executed on a Web server without degrading its response performance

It is clear that the two main materialization policy goals imply a balance (or trade-off) between data freshness and response efficiency. This work deals with this trade-off by introducing a balancing methodology.
Initially, we assume that Web pages are constructed of independent fragments. Many fragments make up a Web page. For simplicity reasons we will also assume that every fragment is a result of a unique query to a database. The model that we use in our evaluation is shown in Figure 1. In pure implementation terms, Web fragments are implemented as independent Server Side Includes. It is clear that the fragments included in every Web page, need to be updated with different update frequencies. For example, a stock quote fragment must be updated more frequently than a news-ticker fragment in order for it to contain data that is up-to-date. Let's assume that the fragments constructing a Web page, change with respective mean frequencies of 3, 5 and 8 changes/min. A change in a fragment, results in a change in the Web page that includes it. Consequently the whole Web page changes with a mean frequency of 16 (=3+5+8) changes/min. If this Web page was materialized with a mean frequency of 16 materializations per minute, one could say that "fresh" data would be sent to users at every request. The methodology that we will describe here, computes the

materialization frequency of Web fragments, in relevance to server conditions, fragment update frequencies and Web page request frequencies.

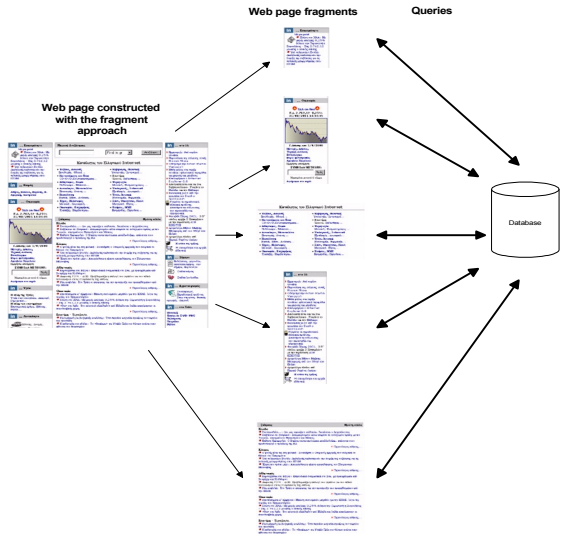


Figure 1 A Web page constructed by independent fragments

In the following paragraphs we present a technique for the appropriate selection of the fragment materialization frequency that requires a set-up period at the server. The architecture of the Web server that can utilize the concept that we propose in this paper is shown in Figure 2. Users request Web pages that are constructed with the use of several Web fragments. The Web fragments are frequently materialized by a Materialization module. This module receives the materialization frequencies for all fragments from a Materialization Frequency Optimizer. The optimizer in turn, is able to compute the materialization frequencies for every fragment by utilizing the data that it receives from:

The Request Propagation Module. This module propagates the requests for all Web pages received by the Web server. This is made possible through the implementation of a Request filter as a front-end of the Web server that receives all requests and then redirects them to the appropriate Web page and at the same time informs the Request Propagation module.

The Fragment Update Propagation Module. This module propagates the update frequencies of every fragment by constantly monitoring the database.

The Server Utilization Propagation Module. This module receives the memory, CPU and hard disk utilization variables of the server and propagates them to the Materialization frequency optimizer for evaluation.

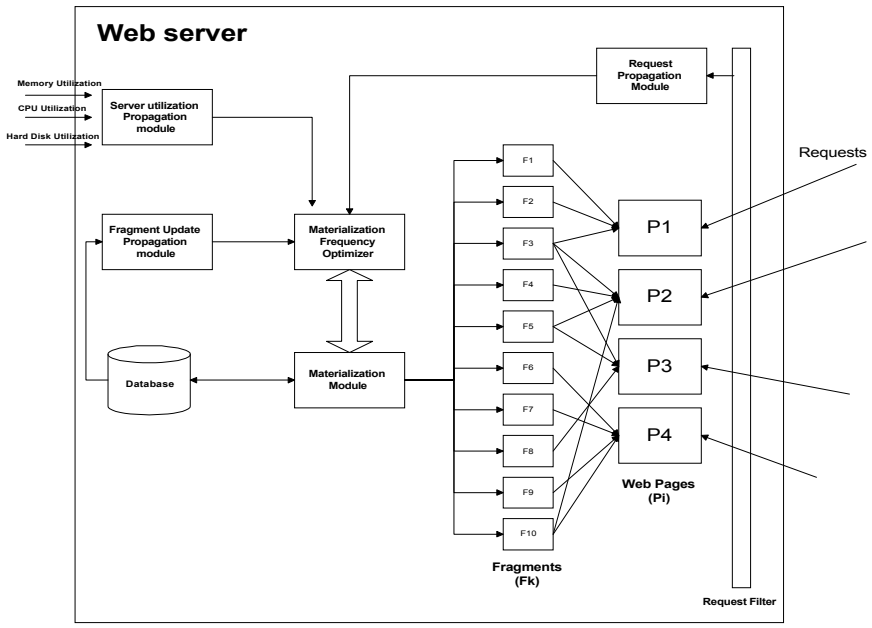


Figure 2 The proposed Web server architecture

We assume that we have selected a set-up period equal to T. During this period we will define the mean update (f_{up}) and request (freq) frequencies of every fragment contained in the Web page set under consideration. During T, the total requests for fragment k (F_k) are:

$$\text{TotalRequests}(F_k) = \sum_{i=1}^{N} \text{Occur}_{Fk}(P_i) * \text{TotalRequests}(P_i) \qquad (1)$$

In Equation 1, N is the total number of Web pages in the Web set under consideration and $\text{Occur}F_k(P_i)$ is the number of occurrences of fragment F_k in Web page i (P_i). This number is usually equal to 0 or 1. Conceptually this means that a Web fragment will normally appear at most once in a Web page, since it would be redundant to show the same information more than once in a Web page. In order to be as general as possible we assume that this variable can be even greater than 1. Equation 1 defines that the total requests for fragment k will be equal to the sum of the total requests to the pages that contain it, multiplied by its occurrence in those pages. But the total requests to web page P_i during T, are equal to its request frequency multiplied by the set-up time T:

$$\text{TotalRequests}(P_i) = f_{req}(P_i) * T \qquad (2)$$

Thus, Equation 1 becomes

$$\text{TotalRequests}(F_k) = \sum_{i=1}^{N} \text{Occur}_{Fk}(P_i) * f_{req}(P_I) * T \Leftrightarrow \qquad (3)$$

$$\text{TotalRequests}(F_k) / T = \sum_{i=1}^{N} \text{Occur}_{Fk}(P_i) * f_{req}(P_i) \Leftrightarrow$$

$$f_{req}(F_k) = \sum_{i=1}^{N} \text{Occur}_{Fk}(P_i) * f_{req}(P_i)$$

Equation 3 connects the request frequency of every fragment with the request frequencies of the Web pages under consideration. With a similar rationale we come to Equation 4 which connects the Web page update frequencies to the Web fragment update frequencies. In equation 4 we consider the same fragment contained in different pages as a different fragment. It is clear that this equation can be further optimized through the fragment approach. This is part of our future work. Here we use equation 4 to show how the update and request frequencies of Web pages can be related to the fragment update and request frequencies.

$$f_{up}(F_k) = \sum_{i=1}^{N} \text{Occur}_{Fk}(P_i) * f_{up}(P_i) \qquad (4)$$

In order to connect the Web page update and request frequencies with the fragment materialization frequencies and the "Acceptable inconsistency" concept we follow the simple algorithm presented below:

If $f_{req}(F_k) < f_{up}(F_k)$ then

$$f_{mat}(F_k) = f_{req}(F_k)$$

else

$$f_{mat}(F_k) = f_{up}(F_k)$$

end if

The algorithm actually defines the values of the materialization frequency by comparing the values of the request and the update frequencies. When the update frequency of a fragment is greater than its request frequency, the materialization frequency of that fragment must by at least equal to the request frequency of the fragment. In this case a compromise period is introduced that is reflected in the compromise frequency ($f_{comp}(F_k)$) shown in Equation 5:

$$f_{comp}(F_k) = f_{up}(F_k) - f_{req}(F_k) \qquad (5)$$

When the request frequency is greater than the update frequency of a fragment the materialization frequency should be equal to the update frequency. In this case the materialization frequency is equal to the update frequency and the compromise frequency is equal to zero.

Since the values of the update and the request frequencies of the fragments are derived from the correspondent values of the Web page update and request frequencies, that can be directly computed through the Web server logs during the period T, the final values of the fragment materialization frequency in both cases will be:

$$f_{mat}(F_k) = \sum_{i=1}^{N} Occur_{Fk}(P_i) * f_{req}(F_k) \qquad \text{when } f_{req}(F_k) < f_{up}(F_k) \qquad (6)$$

and

$$f_{mat}(F_k) = \sum_{i=1}^{N} Occur_{Fk}(P_i) * f_{up}(F_k) \qquad \text{when } f_{req}(F_k) \geq f_{up}(F_k) \qquad (7)$$

It is clear that the materialization frequency in both cases does not cause any staleness in the actual viewed data. A compromise is introduced only to the materialization frequency of the fragments (when $f_{req}(F_k) < f_{up}(F_k)$) since more frequent materializations are considered unnecessary. The concept of the CF must also be dependent on the server utilization parameters. If certain parameters (disk, memory and CPU utilization) surpass a specific threshold, then the materialization frequencies must be reduced. This procedure is reflected in the technique that we have presented, since the Web server's utilization parameters are directly reflected in the Web page request and update frequencies since worse utilization would slow down request servicing and reduce the update frequency. Thus, if we select appropriate threshold values for the Web page request and update frequencies for a Web server, we can reduce all the fragment materialization frequencies proportionally when these values are surpassed, at some point in time.

4.0 Methodology

In order to evaluate the proposed materialization policy, we implemented four different experimental scenarios. Our primary goal was to simulate real Web site conditions. We gave special attention to Web page construction techniques and Web page request patterns. In the following paragraphs we describe our four experimental scenarios in detail. The scenarios that will be described here are experimentally evaluated in the following section. In order to give the notion of compromise, that is in the heart of our proposal, in the following experiments we name the materialization frequency as the “Compromise factor”. This means that a fragment that will be materialized once every 2 seconds will be considered to have a Compromise factor of 2 or simply CF=2.

The experimental set-up consisted of a Web server and several simulated clients. The server used a Pentium 4 CPU at 2Khz, 256Mb of RAM and a Hard disk of 30Gbytes, running under Windows NT. The client processes were implemented on another PC that was connected to the server through a network switch. The set of Web pages evaluated, was determined to be 30 and the clients were also selected to be 30 in order to have a one to one relationship. Every client was tuned to request a Web page at a certain frequency in order to evaluate the materialization policy. We could have done the same by using only one client that would request all the Web pages but we found that tuning the

experiment was much easier when using independent clients for every Web page. The number 30 was selected since it was the number of pages after which the server fell into the data intensive Web site category meaning that the memory, CPU and hard disk utilization parameters begun to surpass 50% of their potential in the worst case.

4. 1 Scenario 1

In this scenario we evaluated a Web site that consisted of static pages. The Web site consisted of 30 static Web pages. No policy was implemented. The pages were requested through 30 independent client processes with the use of a Zipf model [6,18]. Every page was requested through a client process. The request intervals followed the Zipf model which was used because it is the most popular request distribution model on the Web. In the following section this scenario will be referred to as the "All static technique".

4.2 Scenario 2

In this scenario we evaluated the same Web site as in Scenario 1, but this time it consisted of dynamically constructed pages. The Web site consisted of 30 dynamic Web pages. No policy was implemented. The pages were requested through 30 independent client processes with the use of a Zipf model. The dynamic Web pages contained a total of 40 queries to database tables. We assumed that all the data contained in a Web page were the result of a database query. In order to be as consistent to real Web site scenarios as possible, we constructed the dynamic pages in a way that each one issued 10 specific queries to the same database tables and also 10 differentiating queries to other database tables. This way we wanted to simulate the construction of real Web site pages, that usually consist of standard (e.g. menus, headers, footers etc.) fragments that represent the Web page's template, and also fragments of data that change from page to page in the Web site. In the following section this scenario will be referred to as the "All dynamic technique".

4.3 Scenario 3

In this scenario we evaluated a Web site that consisted of frequently updated static Web pages. The Web site consisted of 30 static Web pages. A materialization policy was applied. The policy caused the materialization of all the Web pages in the Web site at a certain frequency. This means that every static Web page also had a dynamic equivalent. Every n seconds a request was issued for the dynamic equivalent page. This caused the execution of the required database queries and the construction of the page. The resulting HTML was saved as the static equivalent, overwriting the older copy. The static pages were then requested by our 30 client processes. The request model was the same Zipf model of scenario 1. This scenario was executed for three different values of the Compromise Factor. The materialization policy was executed every 2, 5 and 10 seconds. In the following section this scenario will be referred to as the CF=2, CF=5 and CF=10 policies.

4.4 Scenario 4

In this scenario we evaluated a Web site that consisted of dynamic Web pages. The difference between the dynamic pages of this scenario and the dynamic pages of scenario 3 was that this scenario included dynamic pages that did not execute the required database queries every time they were requested. They were constructed with the use of the fragment approach that we have already

described. We assumed that every database query represented a Web page fragment. This means that every Web page consisted of 10 standard fragments and 10 differentiating fragments. Every fragment was represented by a server side include statement to a static txt file on the Web server. The static txt files represented Web page fragments and contained the HTML of every fragment. In this scenario we executed the materialization dynamic page of the standard fragments every 10 seconds and the materialization of the differentiating fragments every 2 seconds. In the following section this scenario will be referred to as "Fragment policy".

5.0 Performance evaluation of scenarios

The performance evaluation had two specific goals. These were:

The performance evaluation at the server. This evaluation step aimed at measuring server parameters such as CPU utilization, available memory and throughput at the server. We measured these server parameters during all the experiments that we implemented.

The performance evaluation at the clients. This evaluation step aimed at measuring mean response times at the clients for all the experimental configurations that we implemented.

5.1 Server performance

In this section, we evaluate the performance of the Web server at the time of execution of every policy and technique. We focus on three basic server parameters. The parameters are the following:

- The server's memory utilization
- The server's CPU utilization
- The server's throughput

For every parameter we present the results through graphs and then present our conclusions.

5.1.1 Memory utilization

The following graph shows the available kbytes of memory on server at the time of the execution of the experiments. Higher bars represent better memory utilization. The All static technique shows the best memory utilization which was expected since the technique demands only file retrievals from the server's disk and their transfer to the network. The worst memory utilization is shown by the All dynamic technique. This was also expected since this technique demands the execution of dynamic pages every time they are requested. This procedure results in the extensive use of memory at run-time. The CF=2, CF=5 and CF=10 policies show very similar memory utilization. The proposed fragment policy shows better memory utilization than the dynamic policy and on average 6% worse utilization than the CF=2, CF=5 and CF=10 policies.

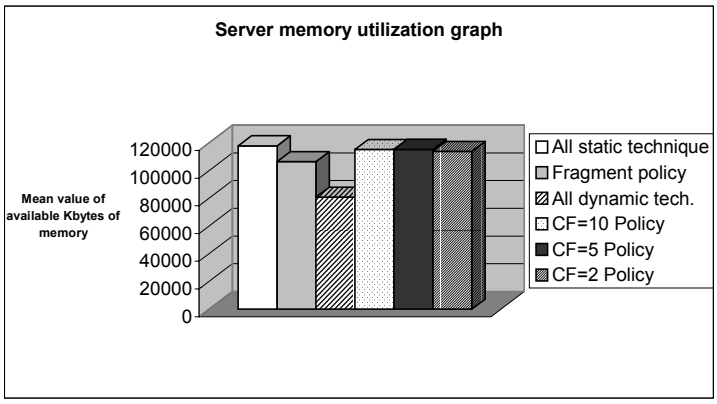


Figure 3 The server memory utilization graph for all approaches

5.1.2 Server CPU utilization

The following graph shows the server's CPU utilization at the time of the experiment execution. Lower bars represent lower CPU usage, thus better utilization. The best CPU utilization is shown by the All static technique. Since the technique involves only file read and file transfer functions, it is clear why the CPU does not play a major role. The CF=2, CF=5 and CF=10 policies show the worst CPU utilization with the CF=2 being the worst of all. The fragment policy shows excellent CPU utilization since it comes in second place with better CPU utilization than all the CF policies and the All dynamic technique.

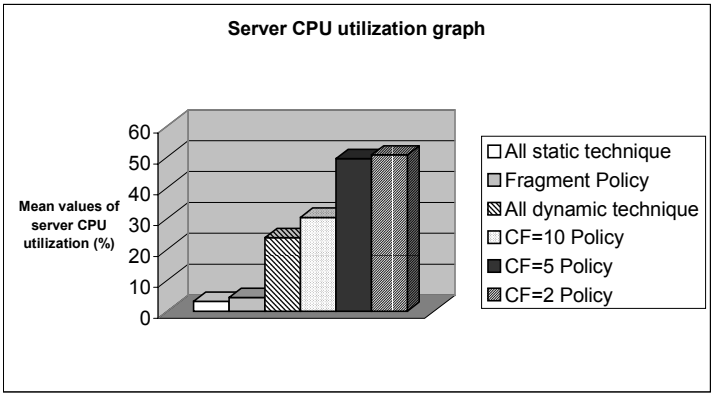


Figure 4 The server CPU utilization graph for all policies

5.1.3 Server throughput

The following graph shows the Web server's throughput by plotting the bytes sent by the Web server/sec. The All static technique shows the best throughput values and the CF=2 policy shows the worst throughput values. The fragment policy shows the second largest value for throughput (after the All static technique).

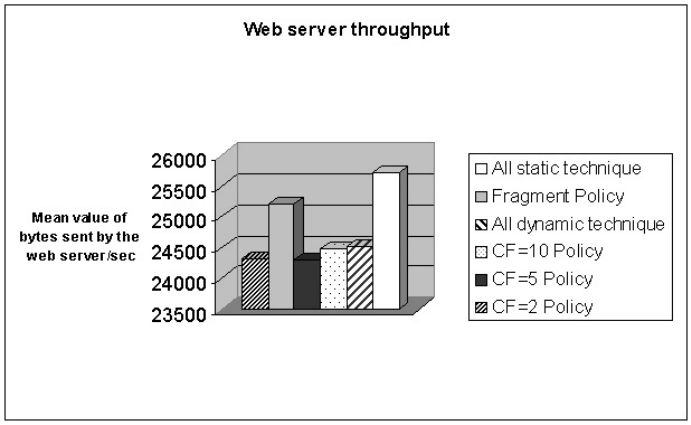


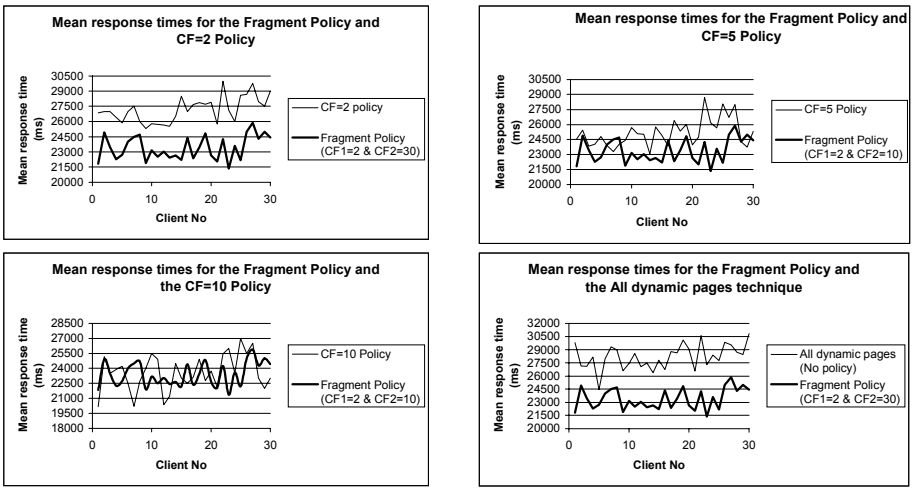
Figure 5 The Web server throughput graph for all the approaches

5.2 Client response times evaluation

In this section we focus on mean response times measured at our experimental clients during the execution of the experiments. First, we compare the fragment policy with all the other policies and techniques and then show some interesting comparisons between the similar approaches.

The following graphs (Figure 6) compare the fragment policy with the CF=2, the CF=5, the CF=10 policies and the All static and All dynamic techniques. The basic conclusions that may be extracted from the first set of graphs are the following:

- The fragment policy is clearly better than the CF=2, the CF=5 policies and the All dynamic pages technique
- The fragment policy can be compared (in terms of performance) with the CF=10 policy
- The fragment policy is clearly worst than the All static technique



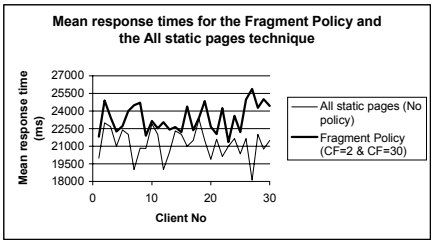


Figure 6 A comparison of the fragment policy with all other approaches

The following set of graphs (Figure 7) compare the different CF policies.

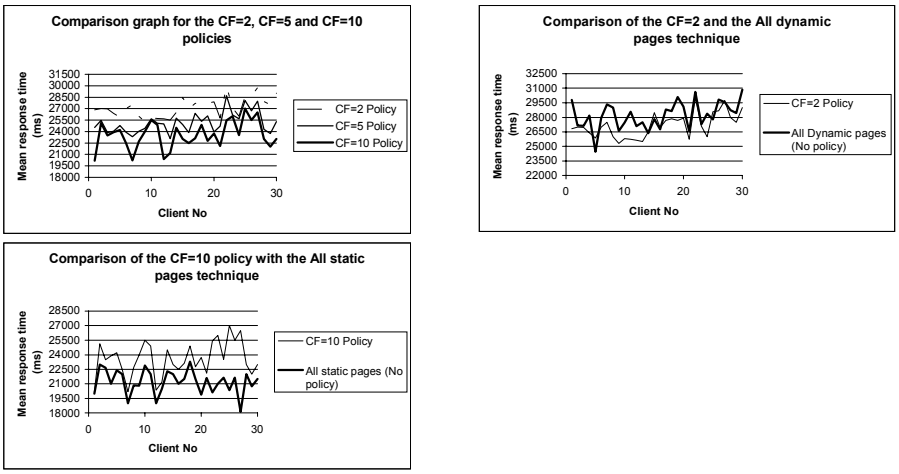


Figure 7 A comparison of all the similar materialization approaches

The purpose of Figure 7 is to compare similar approaches. The first graph compares all the CF policies. As expected the CF=2 policy shows worst response time and the CF=10 policy shows the best response times. The second graph compares the CF=2 and the All dynamic pages technique. Although not very clear, the All dynamic pages technique shows worst response times. The third graph compares the CF=10 policy with the All static technique. The CF=10, as expected, shows worst response times.

Overall, the proposed fragment policy shows excellent results both at the server and at the clients. The policy was tested under real Web server conditions and in most of the experiments was outscored only be the All static approach, that can not be considered an actual alternative for the management of a data-intensive Web site.

The experiments show that by following a fragment approach based on Server Side Includes when constructing and materializing Web pages, the Web server performance can improve, compared to other approaches. As the Web server load grows the materialization policy may use the “Acceptable inconsistency” concept to compute materialization frequencies for all web site pages. It is clear that the “Acceptable inconsistency” concept can be used to keep Web data as fresh as a Web server administrator wants or needs.

6.0 Future Work

Our future work will aim at improving the materialization policy described in this paper and also integrating the policy into a "general purpose" run-time management policy. We believe that the

policy that has been presented here can be improved much further. First we aim at adding a fragment prioritization module before materialization is executed. This module will be responsible of ranking fragments according to several parameters. In this work we have implicitly prioritized fragment materialization by materializing fragments that show higher request rates, with a higher frequency. There are other parameters that should play a role in materialization selection, such as server materialization costs and fragment usability or even administrator prediction. We are currently assembling an ISAPI filter for MS IIS that implements our policy based on a specific algorithm [19]. This filter will be able to receive specific server utilization parameters and analyze Web server log files at run-time. According to some initialization parameters, it will be able to "intelligently" adapt to server conditions and request rates and execute a Web site's materialization policy. Finally, we aim at implementing transparent Web fragment caching on the server's memory, in order to abandon the requirement of creating Web sites with the use of Server Side Includes, that our approach imposes.

7.0 Conclusions

In this paper we have proposed a materialization policy for data intensive Web sites. The necessity of such a policy is inherent in every data intensive Web site. We then evaluated our approach through experimental client/server architectures. Our evaluation consisted of two steps. The evaluation of server utilization parameters and the evaluation of server response times (measured at the clients). The results are very interesting since, it was shown, that by adopting a materialization policy based on the “Acceptable inconsistency” concept, the performance of a Web server may improve even in peak request conditions, without affecting Web data freshness. We believe that our materialization policy may be enhanced much further in order to reduce Web latency, caused on Web servers.

References

[1] J. Challenger, P. Dantzig, D. Dias, and N. Mills, *Engineering Highly Accessed Web Sites for Performance*, Web Engineering, pp. 247-265, 2001

[2] J. Challenger, A. Iyengar and K. Witting, *A Publishing System for Efficiently Creating Dynamic Web Content*, in IEEE INFOCOM, 2000, pp. 844-853

[3] J. Challenger, A. Iyengar and P. Dantzig, *A Scalable System for Consistently Caching Dynamic Web Data*, in IEEE INFOCOM, 1999, 294-303

[4] G. Mecca, P. Atzeni, A. Masci, P. Merialdo and G. Sindoni, *The Araneus Web-Based, Management System*, in Exhibits Program of ACM SIGMOD, 1998

[5] M. A. Habib and M. Abrams, *Analysis of Sources of Latency in Downloading Web Pages*, in WebNet, 2000, pp.227-232

[6] G. K. Zipf, *Human Behavior and the Principle of Least Effort*, Addison-Wesley, 1949

[7] C. E. Wills and M. Mikhailov, *Studying the impact of more complete server information on Web caching*, in 5th International Web caching and Content delivery Workshop, 2000, pp. 184-190

[8] E. Baralis, S Paraboschi and E. Teniente, *Materialized views selection in a multidimensional database*, in VLDB, 1997, pp. 156-165

[9] A. Iyengar and J. Challenger, *Improving Web Server Performance by Caching Dynamic Data*, in USENIX Symposium on Internet Technologies and Systems, 1997

[10] H. Gupta, *Selection of Views to materialize in a data warehouse*, in ICDT, 1997, pp. 98-112

[11] A. Lambrinidis and N. Roussopoulos, *On the Materialization of WebViews*, in ACM SIGMOD Workshop on the Web and Databases (WebDB '99), 1999, pp. 79-84

[12] Y. Li and K. Lu, *Performance Issues of a Web Database*, in Eleventh International Workshop on Database and Expert Systems Applications, 2000, pp. 825-834

[13] D. Florescu, A. Levy and A. Mendelzon, *Database Techniques for the World-Wide Web: A Survey,* SIGMOD Record, vol. 27(3), pp. 59-74, 1998
[14] B. Proll, H. Starck, W. Retschitzegger and H. Sighart, *Ready for Prime Time Pre-Generation of Web Pages in TIScover*, in Eighth International ACM Conference on Information and Knowledge Management, 1999, pp. 63-68
[15] T. Palpanas and B. Krishnamurthy, *Reducing Retrieval Latencies in the Web: the Past, the Present, and the Future*, Graduate Department of Computer Science, University of Toronto, Technical Report CSRG-378
[16] D. Florescu, A. Levy, D. Saciu and K. Yakoub, *Optimization of Run-Time Management of Data Intensive Web Sites*, in 25th VLDB Conference, 1999, pp. 627-638
[17] H. Gupta and I.S. Mumick, *Selection of views to materialize under a maintenance cost constraint*, in ICDT, 1999, pp. 453-470
[18] L. Breslau, P. Cao, L. Fan, G. Phillips and S. Shenker, *Web caching and Zipf-like Distributions: Evidence and Implications*, in IEEE INFOCOM, 1999, pp. 126-134
[19] C. Bouras and A. Konidaris, *An algorithm for handling hybrid run-time management policies in data intensive Web sites*, Work in Progress
[20] E. A. Rundensteiner, A. Koeller, and X. Zhang, *Maintaining Data Warehouses over changing Information Sources,* Communication of the ACM, vol. 43. No 6, 2000, pp. 57-62
[21] The ARANEUS project Home page, http://www.dia.uniroma3.it/Araneus/
[22] D. Florescu, M. Fernandez, J. Kang, A. Levy and D. Suciu, *Catching the Boat with Strudel: Experience with a Web-Site Management System*, in ACM SIGMOD Conference. on Management of Data, 1998, pp. 414-425
[23] C. Olston and J. Widom, *Offering a Precision-Performance Tradeoff for Aggregation Queries over Replicated Data*, in 26th International Conference on Very Large Data Bases, 2000, pp. 144-155
[24] B. Liu, *Characterizing Web Response Time*, M.S. thesis, Virginia Polytechnic Institute and State University, 1998