Run-time Management Policies for Data Intensive Web sites

Christos Bouras[1,2] , Agisilaos Konidaris[1,2]

[1] *Computer Technology Institute-CTI, Kolokotroni 3, 26221 Patras, Greece*

[2] *Computer Engineering and Informatics Department, University of Patras, 26500 Rion, Patras, Greece*
e-mails : {bouras, konidari}@cti.gr

Abstract

Web developers have been concerned with the issues of Web latency and Web data consistency for many years. These issues have become more important in our days since the accurate and imminent dissemination of information is vital to businesses and individuals that rely on the Web. In this paper, we evaluate different run-time management policies against real Web site data. We first define the meaning of data intensive Web sites and categorize them according to their hit patterns. Our research relies on real world Web data collected from various popular Web sites and proxy log files. We propose a Web site run-time management policy that may apply to various real Web site hit patterns and Web data update frequencies.

1. Introduction

In our days the WWW is the most popular application on the Internet because it is easy to use and has the ability to keep all network functions that are executed throughout a browsing session, transparent to the user. The user has the notion that he is requesting information and this information is somehow brought to his/her computer. He/she does not have to know how this happens. Even though most users don't know how the WWW works, almost all of them experience a phenomenon that is formally known as Web Latency [5]. Web Latency is nothing more than the delay between the time that a user requests (by clicking or typing a URL) a Web page and the time that the Web page actually reaches his computer. This intuitive and informal definition of Web latency urged Web site developers and Web equipment developers to try and reduce the waiting time of users in order to make their browsing sessions as productive as possible.

A solution that has been proposed for reducing web latency, and at the same time keeping web data consistent with database changes, is run-time management policies [12, 13]. The issue of implementing run-time management policies for data intensive web sites is as old as data intensive web sites themselves. In order to keep up with user demand and show efficient response times, web sites have resorted to run-time management policies. A run-time management policy may be viewed as a web server component that is executed in parallel with the web server's functions and has total (or partial in some cases) control over those functions. A run-time management policy receives inputs such as user request rates, database update frequencies and page popularity values. The outputs produced may be the pages that should be pre-computed, the pages that should be kept in the server's cache and the pages that should be invalidated from cache.

A run-time management policy for web servers is a dynamic element that can be handled in different ways. Not all policies are suitable for all web sites. A lot of work has been carried out in the fields of specifying, executing and optimizing run-time management policies for data intensive web sites [11, 14].

2. A definition of Data intensive Web sites

In order to propose run-time management policies for data intensive Web sites we must first determine the meaning of Data-intensive Web sites. The term is used for quite a lot of different

"types" of web sites that share a common characteristic. The common characteristic is that a lot of data is demanded of these web sites at a certain point in time. The term "Data intensive web site" differs from site to site. We argue that Data intensive Web sites must be further analyzed in order to determine suitable run-time management policies for each one.

In order to break down Data intensive web sites into categories, we consider two criteria. The first criterion is user demand (meaning number of requests) in time, and the second is the number and change rate of the dynamic elements of a Web page in these web sites[6]. At this point we must clarify that, for simplicity reasons, we will refer to the default pages[1] of sites from now on in our analysis. Our research is focused on the default page because it is the page that most of the sites' requests are directed to (according to the Zipfian distribution of requests). Having this fact in mind we analyze data intensive web site default pages according to the following criteria:

1. **The number of requests in time**. This criterion is closely related to the geographical topology of every web site and its target group. An obvious example is that of a Greek portal. Many Greek portals can be considered Data intensive web sites but only for a limited (and in general well defined) time slot, every day. The peak web access time in Greece is considered to be 09:00 to 14:00 Central European Time (CET). The Greek portals' peak request times are the same. The requests for pages decline after 14:00 CET and come to a minimum overnight. As opposed to Greek portals, that show a peak request time of about 5 hours, and then a substantial reduction in requests, major US portals such as cnn.com show significant demand all through the day. The request threshold of major US portals is much higher than that of Greek portals. Even though they experience similar request curves, as those shown in Figure 1, due to the fact that US users outnumber users from all other countries, their curves do not decline as much as those of Figure 1. In simple terms this means that US portals experience more intense prime-time periods and show much higher request thresholds while not at prime time. In coming years, with the expected expansion of the Internet in other countries (in Europe, Asia and Africa), the curves of Figure 1 (especially for portals of Global interest), will tend to become straight lines. The deference between Greek and US portals has to do with language (Greek versus English) and information importance. It is obvious at this point that a run-time management policy for a Greek portal should be very different from a policy for a major US portal. The problem becomes even more complex when we consider web sites such as the site of the Olympics or web sites set-up for election days, that show peak request demand, all through the day, but only for a limited number of days (or hours). These sites should be treated as an entirely different category in relevance to their run-time management policies.
2. **The number and rate of change of dynamic elements in the default page**. This criterion is directly related to the database involvement in the construction of the Web page. More dynamic web page elements (also referred to as dynamic components in this paper), demand more interaction between the web page and the database. The rate of change of dynamic elements in a default web page, is mainly related to the nature of the information that they contain. For example a dynamic element that contains stock market information must be updated frequently. The in.gr portal has a maximum of 4 dynamic elements in its default page, of which only 2 are updated frequently (in October 2000). The cnn.com site contains 6 dynamic elements in its default page of which 3 are updated frequently (in October 2000).

Considering the first criteria that we have mentioned above, we may break down data intensive web sites to the following general categories:

- Web sites that show limited peak demand (e.g. Greek portals)
- Web sites that experience peak and heavily tailed demand (e.g. US portals)
- Web sites that experience continuos peak demand for a limited number of days (e.g. election sites and sites of sporting events)

All three categories of web sites may contain various numbers of dynamic elements in their default pages, and these dynamic elements may be updated in various time intervals. By combining the two criteria we end-up with virtually unlimited categories of web sites. All of

[1] The default page in this study is the page that is transferred to the user when he/she requests a base URL such as http://www.cnn.com. It is also referred to as the initial web site page.

these web site categories must be efficiently serviced by run-time management policies. It is obvious that each web site needs a self-tailored run-time management policy. In Figure 1 we have included four different request distributions in time. These distributions come from diverse Web sites. It is obvious that the peak request period is much more intense and demanding in the case of the Greek portal in.gr. The curves in Figure 1 reveal that almost all Web servers show peak response times (limited or extended less or more demanding).

In this paper we propose a general cost estimation for a hybrid run-time management policy that can be tailored to different web site needs and we then present its performance evaluation. According to the categorization made in this paragraph, we determine that the main aim of a run-time management policy is to efficiently handle the different request demand of web sites, by servicing different numbers of dynamic elements that have different change rates.

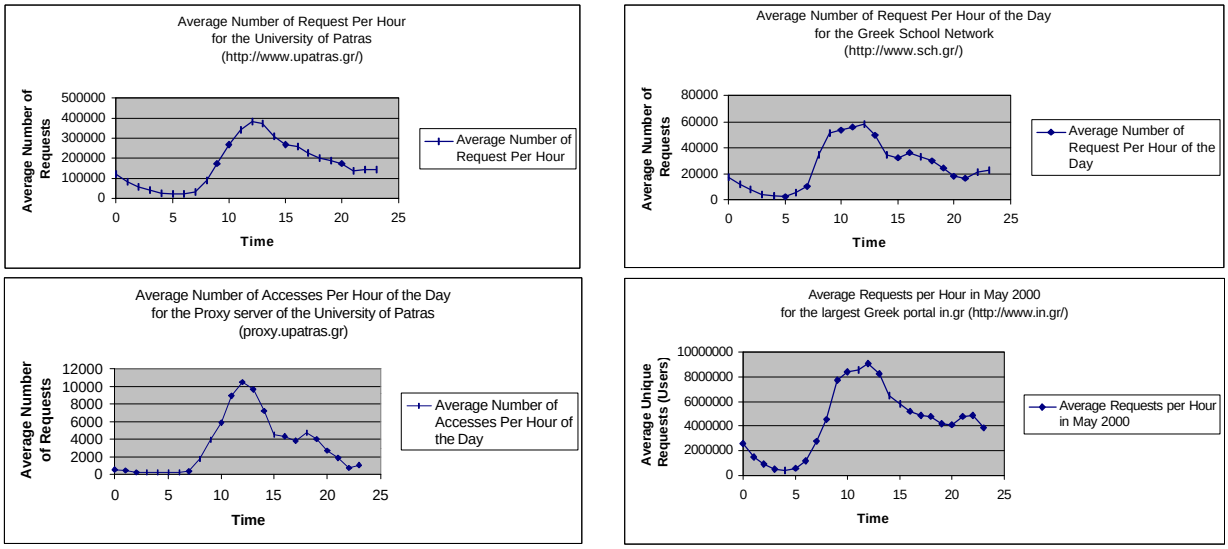

Figure 1 Request distribution for 2 popular Greek sites, 1 University proxy server and the most popular Greek portal

3. Run time management policies

There are two extreme approaches for handling dynamic data on the web. The first is the on the fly creation of dynamic web pages on every user request and the second is the materialization and caching of all dynamic web pages before any request (or in frequent time intervals). The first approach has the obvious drawback of large response times and the second approach has the drawback of possibly serving "stale" data to users. In this paper we will not refer to these approaches since, in our opinion, they do not qualify as run-time management policies. We will only refer to run-time management policies that follow a hybrid approach. These policies implement two basic functions at run-time:

- **Adapt to differentiating request demand.** This means that they can adapt to match the changes of user request in time.
- **Satisfy differentiating numbers of default web page dynamic elements with differentiating update times.** This means that they can perform well, when handling pages that contain different numbers of dynamic elements, which must be updated in different time intervals.

The issue of materialization and caching of dynamic web pages, is one that has been thoroughly discussed in the bibliography [4, 14]. The basic idea behind materializing and caching dynamic web pages, is that a web server response is much quicker when serving static, rather than dynamic pages. This is easily understood, since in the case of static pages, the web server does not have to query the database for data contained in the web pages. A run-time management policy is basically about dynamic web page materialization and caching. There are several issues related to these functions of the policies. These are:

- **Cache consistency and invalidation policy**. The basic problem behind caching dynamic data is the consistency between the data that are stored in the database and the data that are contained in the cached web pages. The problem of keeping caches consistent is a very "popular" problem that has been addressed not only in the context of Web server caches, but mostly in the context of proxy servers. Some of these policies can be applied to the Web server caches [10, 8].
- **Object caching**. Another interesting issue (directly related to the previous) in Web server caches, is the selection of objects that must be cached. Throughout this paper we have considered that the objects that can be kept in a Web server's cache are only "whole" Web pages. This is not true. The issue of caching fragments of Web pages has also been presented in the bibliography [1, 2, 3, 7, 9] and is very interesting, in relevance to our approach that is presented in the following section.
- **Cache size and type**. The size of the cache is one of the most important parameters in caching of dynamic web pages on server. In the extreme case that an infinite (or a very large) cache was available, many of the problems related to caching would be eliminated. Since, in our days, large disks are available and are fairly inexpensive, the issue of cache size should not be considered very important when referring to disk caches. In the case of memory caches the issue of capacity is still considered very important.
- **Caches and web server dependency**. A caching scheme for dynamic web pages should be applicable to many different web server implementations. To make this possible, caching schemes should be implemented as a standalone application that should be able to exchange information with various web servers.
- **Methods of Materialization triggering**. A materialization policy may be implemented with the use of various techniques. Many frequently updated portals update their data through web forms. A news site for example must rely on Web forms, because it relies on journalists to update the site and thus requires a user friendly update environment. The use of web forms causes the triggering of the materialization policy and the consequent update of Web site pages.

3.1. Data intensive Web site study results

In order to record the "behavior" of web sites, in relevance to their default web page update policies, we performed a study that included two well known Greek portals (www.in.gr and www.naftemporiki.gr) and one "global" portal (europe.cnn.com). The first Greek portal is mainly a news and search engine portal and the second is a stock market web site that includes a stock market quote on its default page. In order to determine the default web page update policy in relevance to their request demand we performed an analysis by issuing default web page requests every 3 minutes. From these requests we were able to measure their response times and whether their default pages had changed from our previous request.

The results show a high rate of change in all three sites. The www.in.gr site shows a slight decline in update frequency during early morning hours, but no substantial changes occur during peak times. The www.naftemporiki.gr site shows a decline in update frequency during peak times but on average the update frequency remains the same. The europe.cnn.com site shows a standard update frequency at all times.
The update frequency results may be interpreted as run-time management policies for the default web pages of the sites. The advertisement policies of the sites may have played a substantial role in these results. A more frequent request policy on our part may have shown different results (since the update frequency of a site may be less than 1 change per three minutes) but we did not want to load sites with our requests. The first column of Figure 2 shows the response time of the three sites after issuing requests to the sites every 3 minutes through a web client that we implemented in Java. The second column contains the corresponding number of changes that occurred to the default page, every ten requests to a web site. The change of the web page was computed through its checksum.

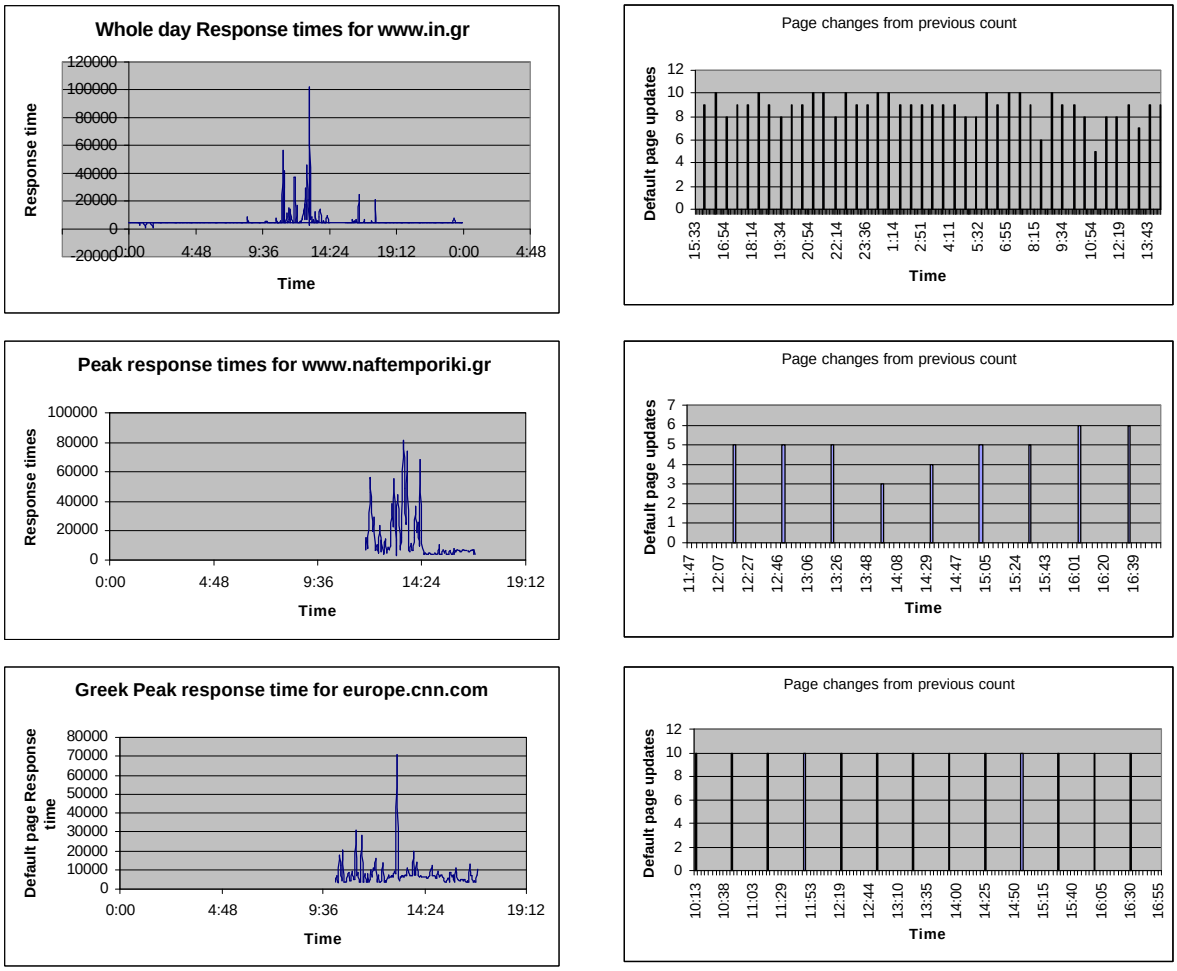

Figure 2 Request times of three popular sites and the correspondent default web page update frequencies

The main result of the statistics in Figure 2 is that web sites follow update policies that rely on standard time intervals. A web site is updated every *n* minutes at all times (peak or not). This gave us another motive to construct an efficient run-time management policy.

4. The proposed Run-time management policy

In this section we will propose a run-time management policy based on characteristics that we have observed in real world data intensive web sites. We will propose a general model for constructing and validating a run-time management policy. Our run-time management policy is based on the assumption that a policy must be based on the current web server conditions and the current web site request demand.

4.1. Proposed policies and cost estimation

The cost estimation of a run-time management policy is the basis of its performance evaluation. In order to have a theoretical view of the efficiency of a policy, one must perform a cost estimation corresponding to the policy. In this paragraph we perform a cost estimation for a default web page of a data intensive web site. First we will refer to a policy, that we will call ***on-data-change-materialize*** policy, that materializes every dynamic component of a web page, every time it changes. This policy has the advantage that web pages are always consistent with database changes but imposes a substantial load to servers in the case of frequently changing web pages.

In data intensive web sites, and especially during the period of intensity the policy described above might not be a very good proposal. Consider an example of a web page that contains three

dynamic components. The data contained in the database, and concerns the first component, changes very frequently (e.g. as in the case of stock quotes). The data of the other two dynamic components change periodically but with a much smaller frequency. According to this cost estimation, the default page should be re-computed and materialized very frequently because of the frequent changes in data of the first component, even though the other two components did not need refreshing. The cost of materialization in this case, equals to the cost of materializing the web page every time a dynamic component that is included needs refreshing. As one can conclude, this is not a very good proposal especially at peak request times.

The other proposal consists of a hybrid model. Our basic concern is peak request demand periods. Those are the periods when a web site should perform best. Our hybrid model combines two run-time management policies. The ***on-data-change-materialize*** policy, that has been already presented, and an ***on-data-change-batch-compromise*** policy that we will explain in the following paragraphs. The basic idea is that the ***on-data-change-materialize*** policy should be executed until the Web server load reaches a certain threshold after which the ***on-data-change-batch-compromise*** policy is executed.

The ***on-data-change-batch-compromise*** policy attempts to estimate the optimal wait time between default web page materializations. We will attempt to clarify this by presenting the three web page dynamic elements example, the first of which is a stock quote. The stock quote element has a change frequency f_1=6 changes per minute. The second element has a change frequency f_2=2 changes per minute and the third has a change frequency f_3=0.2 changes per minute. The ***on-data-change-materialize*** policy considers these three frequencies independent variables. In simple terms this means that the default page would be materialized 8.2 times per minute, on average. This is very costly and can not be considered a good practice. In our ***on-data-change-batch-compromise*** policy, we relate these three change frequencies by attempting to implement a batch materialization model that relies on accepted compromises in data "freshness". In our example the stock quote dynamic element should be refreshed every 10 seconds, the second element every 30 seconds and the third every 5 minutes (300 seconds). The problem here, is to combine the web page materializations that are caused by one dynamic element change frequency with those caused by another. The obvious solution is to divide all dynamic element change periods with the smallest period. In our case, since the smallest change period is 10 seconds we would divide 10 seconds/10 seconds = 1, 30seconds/10seconds=3 and 300 seconds/10 seconds=30. We name the smallest value (e.g. 1) the Minimum Compromise Factor-CF$_{MIN}$, the second result (e.g. 3) the First Compromise Factor-FCF and the third (e.g. 30) as the Maximum Compromise Factor-CF$_{MAX}$. The ***on-data-change-batch-compromise*** policy may be implemented by choosing any of the compromise factors that were derived. The data on the web page are "more" consistent to database changes, as the compromise factor becomes smaller. The compromise factor is a meter of the web page developer's or administrator's willingness to compromise in data "freshness", in order to achieve better Web server performance. After a compromise factor has been chosen, the materialization of the web page is executed in batches. Let’s consider the case that the FCF has been chosen. This means that a web page materialization takes place every 3 changes of the stock quote dynamic element value in the database, which will include a change in the second element value and so on.

The concept of the CF (Compromise Factor) can also be very beneficial to a run-time management policy. Its value does not have to be the result of the division that we mentioned above. The value of the CF may significantly differ from this result. This may happen in cases where the administrator believes that the result of the division is unacceptable because the services demanded by users (web site Quality of Service) require a different approach. Our proposal aims at relating the value of the CF with the Maximum Request Handling value (MRH) on the server, which is the maximum requests that the web server can fulfill at any point in time, and the Current Request Rate (CRR) which is the current amount of requests to the web server. It is obvious that the value of the CF should be greater on a server with a MRH of 100000 that has a CRR of 90000 than the value of the CF on a server with an MRH of 100000 and a CRR 20000. In our proposal the CF value is computed at run-time in frequent time intervals.

5. Performance Evaluation

5.1. Experimental set-up and methodology

The basic requirement that we wanted to ensure in the topology that we used to evaluate the hybrid run-time management policy, was the non-intervention of conditions not related with the policies. In order to ensure this as much as possible we used the topology shown in Figure 3.

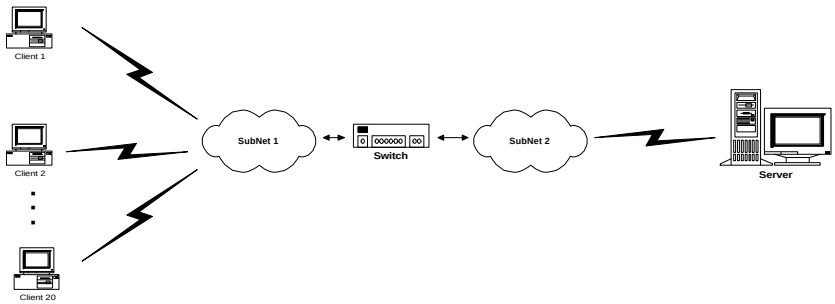

Figure 3 The experimental set-up

We tested our policies by implementing the client-server model on campus at the University of Patras. We used a Pentium III Computer with 256MB of RAM and 30GB Hard disk running Windows NT as a server and an Ultra Sparc 30 with 128MB of RAM and 16GB Hard Disk running Solaris 2.6 to simulate our clients. Our client program was executed on the Ultra Sparc and was able to simulate simultaneous client requests to the server running as independent UNIX processes.

5.2. Evaluation of different types of dynamic web components

In our performance evaluation we tested six different categories of dynamic web components. The categories are shown in Table 1. Only three of the categories are dynamic components. The other three are the corresponding static components. We included them in the evaluation in order to show the big difference between static and dynamic components of web pages.

Category ID	Category Name	Description
1	Simple Query Static component **(SQS)**	A static component that is a result of a simple query
2	Simple Query Dynamic component **(SQD)**	A dynamic component that includes the same simple query of 1
3	Multiple Queries Static component **(MQS)**	A static component that is the result of multiple queries to the database
4	Multiple Queries Dynamic component **(MQD)**	A dynamic component that contains the same multiple queries as 3
5	Simple Query Static component with large result size **(LDS)**	A static component containing the data of a simple query that returned a substantial amount of data
6	Simple Query with large result size Dynamic component **(LDD)**	A dynamic component that contains the same simple query as 5 that returns a substantial amount of data

Table 1 The categories of Web components used in our performance evaluation

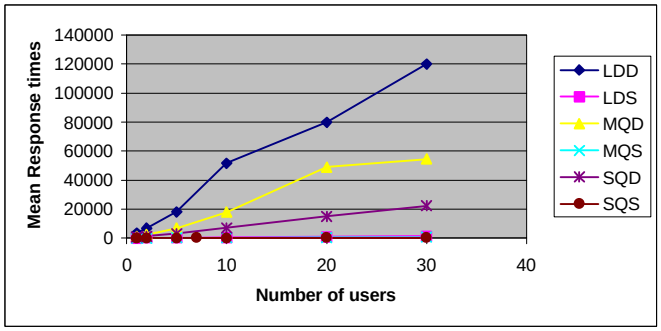

Figure 4 The Mean Response times of all categories of components (MQS and LDS static components are not shown clearly because they show response times very similar to SQS)

In is obvious from Figure 4 that the dynamic component that contains a lot of data has the largest response times. The second worse response times are shown for the dynamic component that contains multiple queries and the third worse from the dynamic component that contains a simple query. The static components follow with smaller response times.

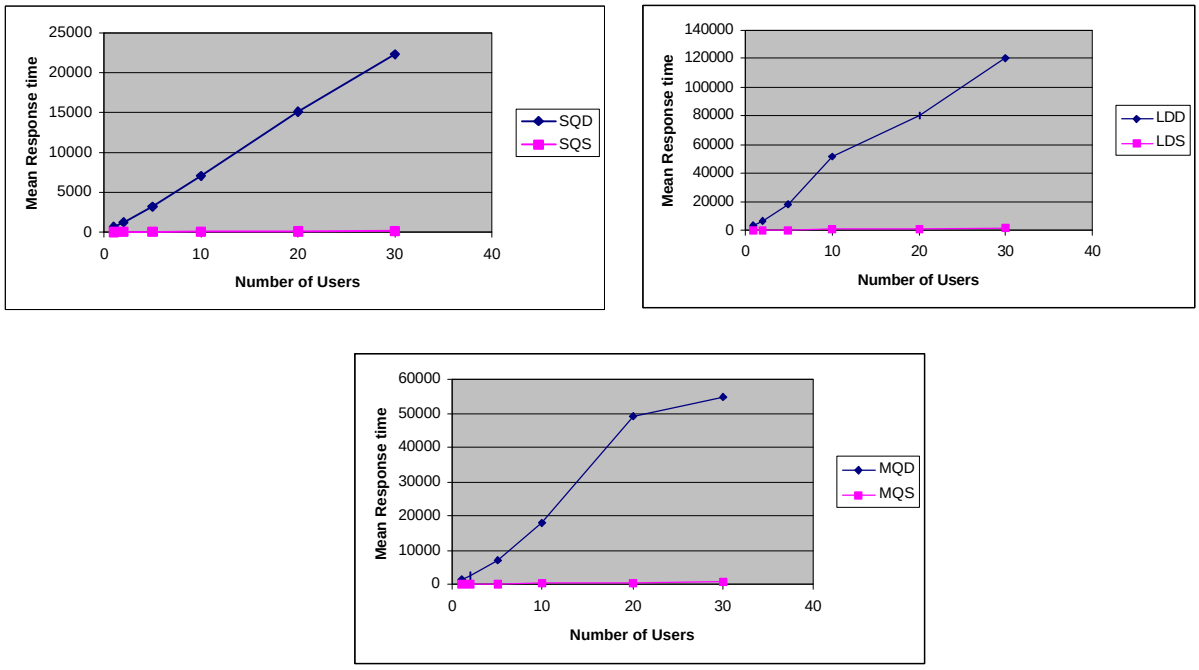

Figure 5 Mean response times for corresponding static and Dynamic pages

Figure 5 illustrates the large differences between dynamic web page components and static web page components. The difference becomes more obvious as the clients become more, and consequently response time rises.

The basic conclusion that may be extracted from this first step of evaluation is that web developers should implement dynamic web components that do not contain a lot of data and are not the result of complex database queries. Thus the use of views and indexes in web database design can be very useful in relevance to dynamic web page elements.

5.3. Evaluation of the hybrid run-time management policy

In this section we evaluate our hybrid run-time management policy. We constructed a default web page that consisted of 3 dynamic elements that needed refreshing every 5, 10 and 20 seconds. We issued a number of requests equal to 11 requests per second for that web page, through client processes. Twenty clients were monitored and their responses were recorded. We also issued a number of requests to the web server through other client processes in order to achieve a background load on the web server. The evaluation Figures contain equalities such as

CF=10sec which must be interpreted as *CF corresponding to 10 second default web page update frequencies.*

The rationale behind the experiments was to show that different CF values may improve performance under specific server loads.

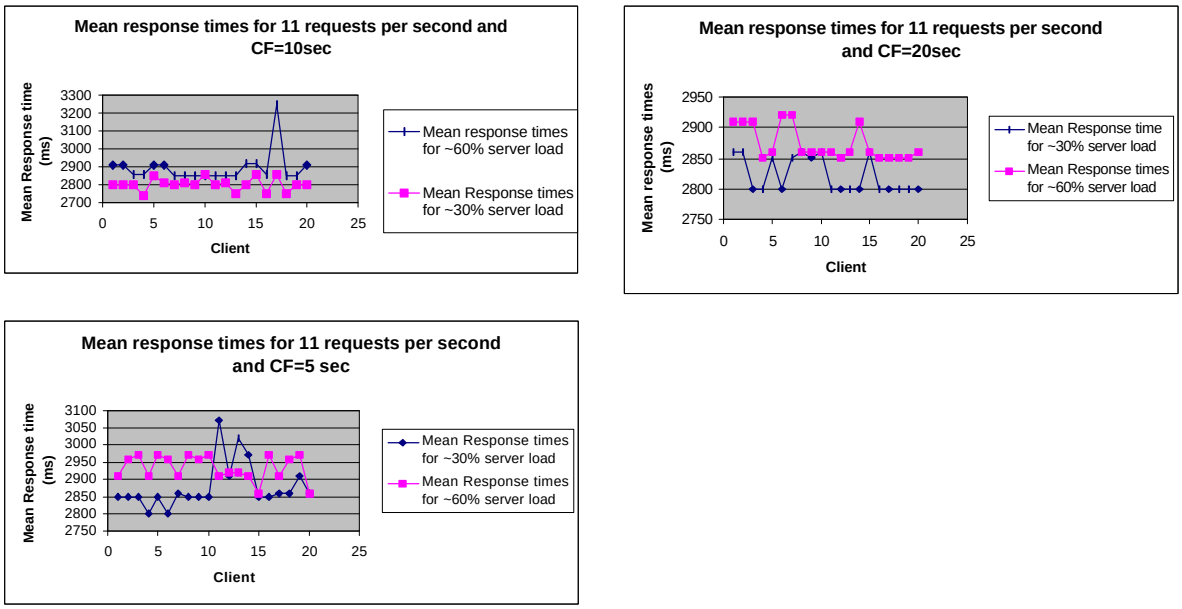

Figure 6 Mean response times of the default page with different values of CF and different server loads

Figure 6 shows the mean response times of the default page in 20 clients, calculated for different values of CF and different server loads. In the case of CF=10, meaning that the default web page was materialized every 10 seconds, we see a clear improvement in the default web page response times when the server load is close to 30% of the maximum. This improvement is even clearer in the case that CF=20. We would expect such a result since what it means is that a web page is transferred faster to a client when it is materialized every 20 seconds than when it is materialized every 10 seconds, under specific server loads.

The situation becomes more complex when CF=5 since the response times are not clearly better for a lower server load. This is logical because the rate of default page materialization itself, imposes a load to the server. The overall conclusion of Figure 6 is that the CF is directly related to the response times of the server, under specific server loads. Thus, the CF should be calculated in a way that it would result in better server response times. It is obvious that the value of CF should increase as the load of the Web server (which is the result of greater request demand) rises.

6. Future Work and conclusions

Our future work will aim at improving the hybrid run-time management policy described in this paper. We believe that the policy that is presented in this paper can be improved much further. It can be improved by using what we call Web page component caching. We would like to evaluate our policy together with schemes presented in [1, 2, 3, 7, 9]. These schemes present the idea of fragment caching instead of whole web page caching. This way we would like to evaluate schemes that store fragments or materialized web page elements in cache and integrate them on user request.

We would also like to further enhance the computation of CF with other parameters. In this paper we have only related CF to the Current Request Rate, the Maximum Request Handling value and the update frequencies of components. We would like to be able to relate it with other parameters such as general network state and web page popularity in the future.

The results are interesting, since it was shown that by adopting a compromise policy, the performance of the web server may improve even in peak conditions. The basic problem is to define the compromise factor that we named CF. We believe that our hybrid policy may be enhanced much further in order to reduce web latency that is related to web servers.

7. References

[1] Jim Challenger, Paul Dantzig, Daniel Dias, and Nathaniel Mills, "Engineering Highly Accessed Web Sites for Performance ", Web Engineering, Y. Deshpande and S. Murugesan editors, Springer-Verlag.

[2] J. Challenger, A. Iyengar, K. Witting., "A Publishing System for Efficiently Creating Dynamic Web Content", In INFOCOM 2000, March 26-30, 2000 Tel Aviv, Israel

[3] Jim Challenger, Arun Iyengar and Paul Dantzig, "A Scalable System for Consistently Caching Dynamic Web Data ", In Proceedings of IEEE INFOCOM'99, New York, New York, March 1999.

[4] G. Mecca, P. Atzeni, A. Masci, P. Merialdo, G. Sindoni "The Araneus Web-Based, Management System" - In Exhibits Program of ACM SIGMOD '98, 1998

[5] Md Ahsan Habib, Marc Abrams "Analysis of Sources of Latency in Downloading Web Pages", WebNet 2000, October 30 - November 4, 2000 San Antonio, Texas, USA

[6] Brian E. Brewington, George Cybenko. "Keeping Up with the Changing Web". IEEE Computer Magazine May 2000

[7] Craig E. Wills, Mikhail Mikhailov, "Studying the impact of more complete server information on web caching", 5th International Web caching and Content delivery Workshop, Lisbon, Portugal, 22-24 May 2000

[8] Khaled Yagoub, Daniela Florescu, Valérie Issarny, Patrick Valduriez, "Caching Strategies for Data-Intensive Web Sites", Proc. of the Int. Conf. on Very Large Data Bases (VLDB) Cairo, Egypt , 10-14 september, 2000

[9] A. Iyengar, J. Challenger. "Improving Web Server Performance by Caching Dynamic Data", In Proceedings of the USENIX Symposium on Internet Technologies and Systems, Monterey, California, December 1997

[10] Arlitt, Martin F., Friedrich, Richard J., Jin, Tai Y., "Performance Evaluation of Web Proxy Cache Replacement Policies", HP Technical Report HPL-98-97, 980601 Available at http://www.hpl.hp.com/techreports/98/HPL-98-97.html

[11] Alexandros Lambrinidis and Nick Roussopoulos "On the Materialization of WebViews", CSHCN Technical Report 99-14 (ISR T.R. 99-26) In proceedings of the ACM SIGMOD Workshop on the Web and Databases (WebDB '99), Philadelphia, Pensylvania, USA, June 1999

[12] Yi Li and Kevin Lu, "Performance Issues of a Web Database" In Proc. of Eleventh International Workshop on Database and Expert Systems Applications 4-8 September, 2000, Greenwich, London, UK

[13] Daniela Florescu, Alon Levy, Alberto Mendelzon, "Database Techniques for the World-Wide Web" A Survey. SIGMOD Record, 27(3):59-74, 1998

[14] Birgit Proll, Heinrich Starck, Werner Retschitzegger, Harald Sighart, "Ready for Prime Time Pre-Generation of Web Pages" in TIScover, WebDB '99, Philadelphia, Pennsylvania, 3-4 June, 1999.