
World Wide Web: Internet and Web Information Systems, 6, 23–47, 2003

Performance Evaluation of a Hybrid Run-Time Management Policy for Data Intensive Web Sites

CHRISTOS BOURAS and AGISILAOS KONIDARIS {bouras, konidari}@cti.gr
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

The issues of performance, response efficiency and data consistency are among the most important for data intensive Web sites. In order to deal with these issues we analyze and evaluate a hybrid run-time management 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 hybrid Web site run-time management policy that may apply to different Web site request patterns and data update frequencies. A run-time management policy is viewed as a Web page materialization policy that can adapt to different conditions at run-time. We define a concept that we have named the Compromise Factor (CF), to achieve the relationship between current server conditions and the materialization policy. 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 first present a general a comparative cost model for the hybrid management policy and three other related and popular Web management policies. We then evaluate the performance of all the approaches. The results of our evaluation show that the concept of the CF may be beneficial to Web servers in terms of performance.

Keywords: Web server performance, content management, Web metrics and measurements, Web practice and experience, Web consistency, views over the Web, querying/searching on the Web, Web latency

1. Introduction

In our days the Web is the most popular application on the Internet. Even though most users do not know how the Web works, almost all of them experience 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 actually reaches his/her computer. It can be traced down to all components of the client-server model [13,18,21]. 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 the help that they provide in managing, upgrading and maintaining Web sites [9,17]. Even though there have been other approaches proposed (such as the integration of Web servers into databases), the independent Web and Database client/server 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 run-time management policies. The issue of implementing run-time management policies for Web sites, has become very important in recent years, due to the explosive growth of the Web. 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) control over those functions. A run-time management policy receives inputs such as user request rates, database update frequencies and server utilization parameters. Some of the outputs produced, are the pages that should be materialized, the pages that should be kept in the server's cache and the pages that should be invalidated from cache. This work proposes a hybrid run-time management policy that can efficiently handle the data demand and request load of a data intensive Web sites.

In order to propose run-time management 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 general cost model for a hybrid run-time management policy that can be tailored to different Web site needs and then present its performance evaluation. The main aim of a run-time management 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.

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. Section 3 presents the basic issues related to dynamic data handling on the Web and how we deal with them in this work. In Section 4 we introduce the CF concept and show how it can be computed in the general case. Section 5 contains the cost models for the most popular approaches for dynamic data handling on the Web today but also of two run-time management policies that we propose. In Section 6 we present our experimental methodology and in Section 7 our experiments. Finally, Section 8 contains the future work that can be carried out in the field and our conclusions.

2. 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. In [5–7,15], the approach is presented 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. A case study for materializing and replicating dynamic data is presented in [22]. The paper presents an interesting materialization architecture that is quite straightforward.

STRUDEL [8,10] 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 [19,24] 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 run-time management policy can then be utilized without any further constraint.

Work related to the optimization of query workloads on DBMS can be found in [2,11,12,23]. 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 in 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 [16] 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 that is based on the same principles as the one presented in this paper. The basic difference is that we present a more analytic cost model that evaluates more complicated policies that those presented in [16]. A similar approach to ours but implemented in the field of data replication is presented in [20]. 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 the approach of [20] is not transparent to users, since they can actually ask for a degree of acceptable staleness along with their queries. This would be very difficult in a Web environment.

Another interesting approach on Web fragmentation and caching is presented in [26]. 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 approach to already implemented Web pages that do not follow the Web fragment approach. We intend to utilize it in our future work.

3. Dynamic Web data and caching

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 has been received. The first approach has the obvious drawback of large response times and the second has the drawback of serving "stale" data to users. In this paper we refer to these approaches even though they do not qualify as run-time management policies, in order to show their extreme results.

The issue of materialization and caching of dynamic Web pages is one that has been thoroughly discussed in the bibliography. The basic idea behind materializing and caching dynamic Web pages is that a Web server response is much quicker when serving static and not dynamic pages. There are several issues related to dynamic Web page caching:

Cache consistency and invalidation policy. The basic problem behind caching dynamic data is consistency maintenance between the data that are stored in the database and those that are stored in server cache. 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 the proxy server approaches can be applied to Web server caches [1,14,25]. The basic tradeoff related to cache consistency and invalidation policies is between server resource utilization and the degree of cache consistency. This is the basic issue that this paper deals with. The balancing problem between consistency and utilization is addressed through the hybrid run-time management policy that we propose.

Object caching. Another interesting issue in Web server caches is the selection of the objects that can be cached. Web pages are not the only objects that can be cached in a Web server's cache. The issue of caching Web page fragments has also been presented in the bibliography [5–7,15,26] and is very interesting in relevance to our approach, since in this work we propose the materialization and caching of Web fragments that can be viewed as Server Side Includes.

Cache size and type. The size of the cache is one of the most important parameters in caching dynamic Web pages. 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 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. In this work we will consider disk caching of Web fragments as a part of our materialization approach.

Caches and Web server dependency. A caching scheme for dynamic Web pages should be applicable to many different Web server implementations. In order to achieve this, caching schemes should be implemented as a standalone application that is able to exchange information with various Web servers. Our approach can be implemented in different Web server environments, as long as Web server utilization parameters can be passed from the Web server to the management policy module.

In this paper we efficiently address all the issues mentioned above, through a run-time management policy. Throughout this paper a run-time management policy will be considered as a materialization policy with a materialization frequency that can adapt to specific server parameters at run-time. The materialization that we will consider in this paper uses

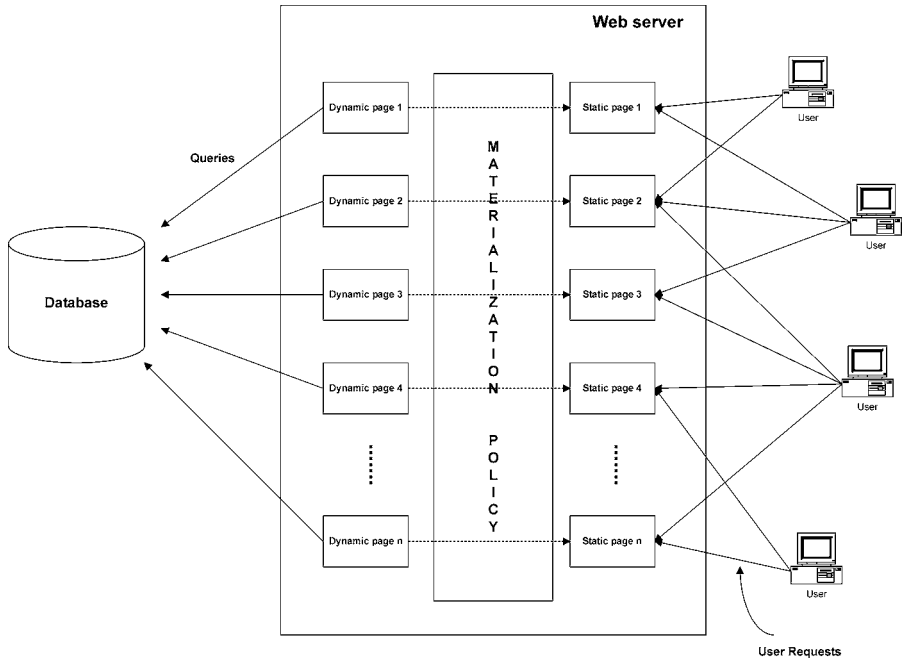

Figure 1. The materialization architecture.

disk caching, but it can very be modified to use memory caching as well. Figure 1 shows the materialization architecture.

Materialization is a term used to represent the transformation of dynamic Web data into 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.

When materialization policies are applied to Web sites, users request the static equivalents of dynamic Web pages. It is clear that if the materialization frequency is not selected according to database updates, users will view stale data on some of their requests.

4. The Compromise Factor (CF) concept

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 is to apply an efficient materialization policy to a Web site that will have two main characteristics:

- 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.

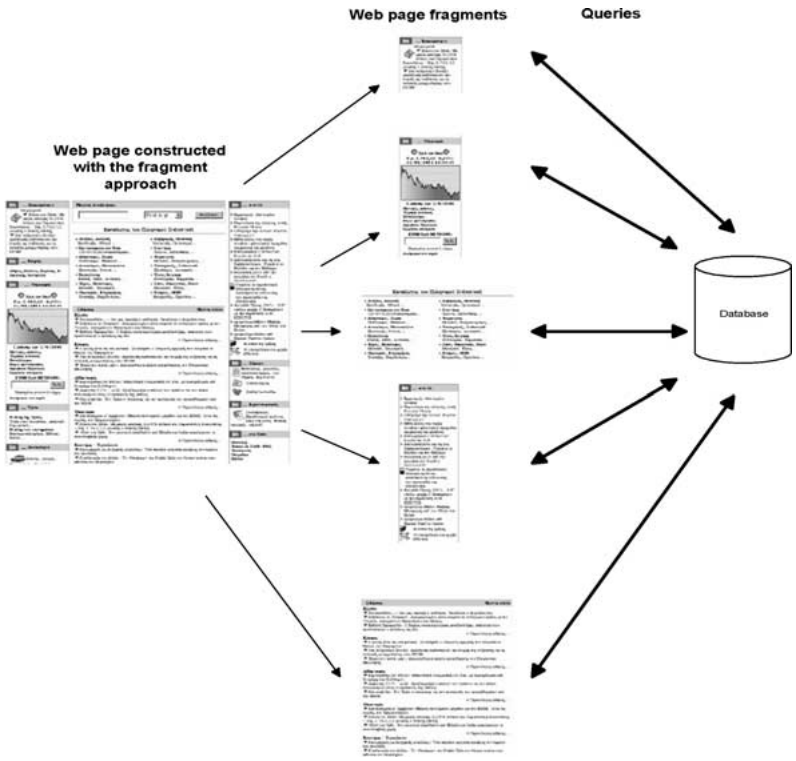

Figure 2. A Web page constructed by independent fragments.

It is clear that the two main materialization policy characteristics imply a balance (or tradeoff) between data freshness and response performance. This work deals with this tradeoff by introducing a balancing technique called the Compromise Factor.

Initially we assume that Web pages are constructed by 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 2. 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. Let us 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 on average, over a period of time. The CF is a metric that computes the materialization frequency in relevance to server conditions, fragment update frequencies and Web page request frequencies.

4.1. The problem

Let us assume that a Web server hosts a data intensive Web site constructed with the fragment approach (e.g., [15]). As the site is data intensive, the demand is high and the back-end database is frequently updated. The server follows a run-time management policy that requires the materialization of changed Web pages in frequent time intervals. The Web server has two main responsibilities:

1. To efficiently respond to user requests for pages.
2. To efficiently materialize changed Web pages in order to keep the data included in them as fresh as possible.

The CF concept is a metric for keeping the balance between efficient response times and efficient materialization. In most Web sites the periods of peak request coincide with the periods of frequent data updates. The run-time management policies that are used in most cases only consider the parameter of page freshness in order to go ahead with materialization. In other words, the run-time management policy does not rely on current Web server conditions and current request demand, in order to decide on a suitable materialization policy. The CF concept enables Web servers to switch between materialization frequencies according to certain conditions. In order to provide a step by step approach to the CF concept, we present the following examples.

Example 1 (Fragment based CF selection for one page). Let us assume that a Web page consists of six fragments. Each fragment has a different mean update frequency. We intend to find a metric that would help us decide on a suitable materialization frequency. The empirical methodology that we use is the following. First we plot Figure 3. The vertical axis represents the fragment number and the horizontal axis represents the corresponding mean update frequency. The small squares correspond to the update frequencies of each fragment. The thick vertical line represents the CF. It is clear that the concept of the CF in this example represents the value of the materialization frequency that must be applied to the specific Web page. The selected value of the CF depends on the update frequencies of the fragments contained in the Web page. The position of the vertical line on the graph may change according to server conditions and move to the left or to the right. The graph area on the left of the CF line corresponds to the fragments of which the updates frequencies are "satisfied" by the current value of CF. Since the materialization frequency is greater than the update frequency of these fragments, they will be kept "fresh," from a user's point of view, over a period of time. Of course, since user requests may occur after a fragment has been updated and before it has been materialized some user requests will possibly receive "stale" data. The graph area on the right corresponds to the fragments of which the update frequencies are not "satisfied" by the current value of the CF. This means that there will be materialized pages in the context of which these fragments will be "stale." In the case where the line representing the CF would be over Freq9, and thus equal to the update frequency of the fastest changing Web page fragment, the materialized Web page would mostly hold "fresh" data.

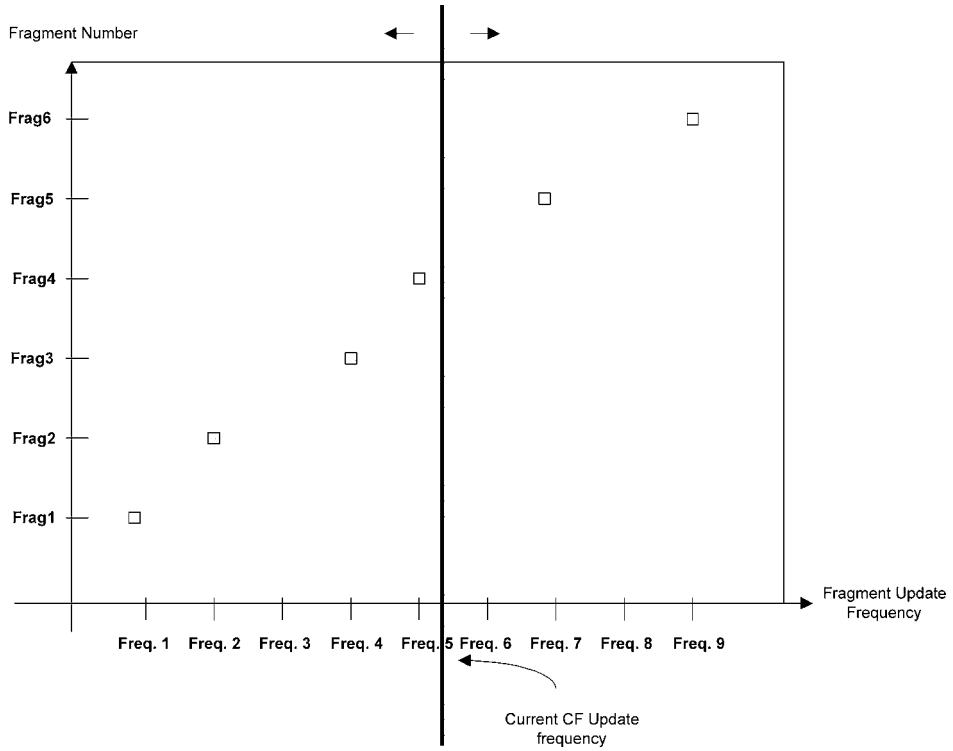

Figure 3. The fragment based CF approach.

Example 2 (Web page based selection of the CF for multiple pages). In this example we work on a Web page basis and attempt to compute the CF for multiple pages. Let us assume that a Web site contains six Web pages that are constructed with the use of fragments. We intend to find a metric that would help us batch the materialization of many pages at once. What we actually want to achieve is the definition of a materialization frequency that would be suitable for as many pages as possible. In order to do this we plot the graph of Figure 4. The vertical axis represents the Web page numbers appointed to each page. The horizontal axis represents the update frequencies of the Web pages. Each page has a series of update frequencies associated with it. These frequencies are those of the fragments contained in every page. The fragment frequencies form a frequency window for every Web page as shown in Figure 4. The thick vertical line represents the current CF for the set of Web pages contained in the graph. The CF here, defines the batch materialization frequency for the set of pages.

Figure 4 shows that, according to the value of the Compromise Factor, each Web page is compromised at a different extent. Page1 is not compromised at all, Page2 is 75% compromised since 3 out of 4 of its fragments have update frequencies greater than the current value of CF, and Page3 is 33% compromised. This approach is similar to [20] since a Web server administrator can use it to identify specific precision-performance tradeoffs.

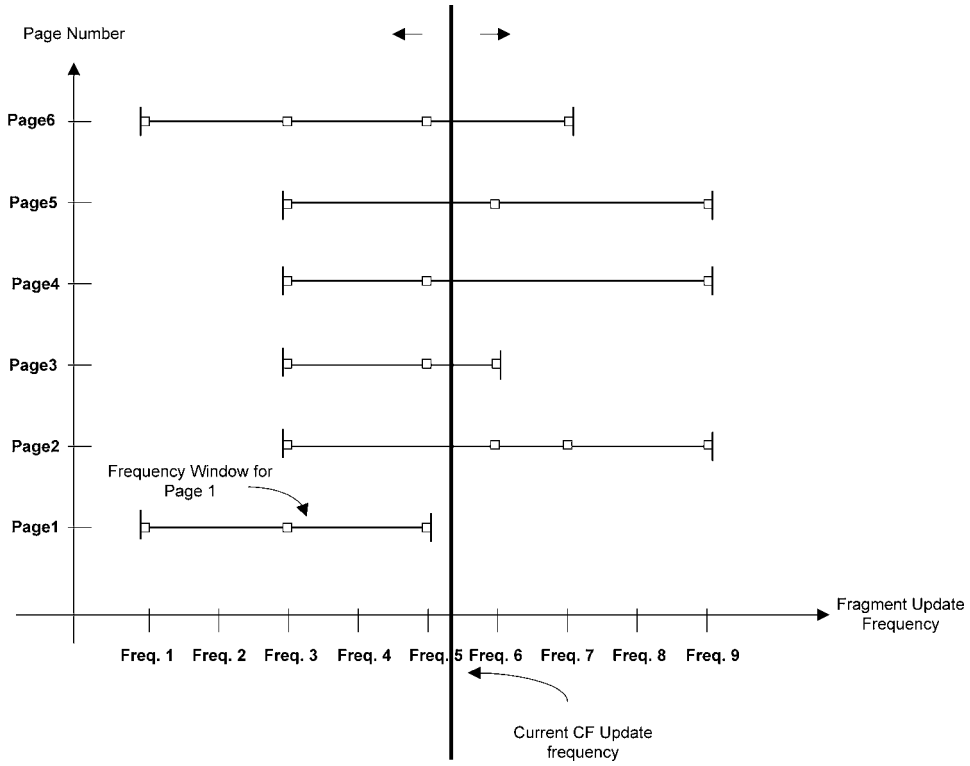

Figure 4. The Web page based CF approach.

From what we have presented until now, the CF always causes compromises to Web page "freshness." This is not true. The concept of the CF can facilitate a Web server without introducing compromises to Web page freshness.

As presented in this paper, a Web page "freshness" compromise is actually the inclusion of portions of stale fragments in materialized pages. Looking at the issue from the users' point of view, it does not really matter if a Web page contains stale data in the period of time that they are not visiting it. Their only concern is whether a Web page contains "fresh" data when they request it. An extreme example will prove this. Let us assume that a Web page contains a stock-quote. A stock-quote must be, at least, updated every one or two seconds in order for it to be useful. One user visits the specific Web page every minute. Users will always be getting an updated version of the stock quote if the Web page is materialized with a frequency of at least 1 materialization every 60 s, even though the stock quote changes with a much greater frequency. It is clear that a Web page materialization policy should take into consideration, not only the update frequencies of the page but also the request frequency of the page. In order to approach the CF concept under this consideration we present the following example.

4.2. *Selection of the CF*

The selection of the CF for a set of Web pages requires specific steps. The selection of an appropriate CF initially involves two parameters. These are:

- the update frequency of every page,
- the request frequency of every page.

The parameter that must be specified is the materialization frequency for every fragment. In the following paragraphs we present a technique for the appropriate selection of the CF (= materialization frequency) that requires a set-up period at the server.

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 (f_{req}) frequencies of every page in the Web page set under consideration. During T the total requests for fragment k are:

$$\text{TotalRequests}_{F_k} = \sum_{i=1}^{N} \text{Occur}_{F_k}(i) \cdot \text{TotalRequests}_{P_i}. \tag{1}$$

In Equation (1), N is the total number of Web pages in the Web set under consideration and $\text{Occur}_{F_k}(i)$ is the number of occurrences of fragment F_k in Web page 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 inside 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:

$$\text{TotalRequests}_{P_i} = f_{\text{req}(i)} \cdot T. \tag{2}$$

Thus, Equation (1) becomes

$$\begin{aligned} \text{TotalRequests}_{F_k} &= \sum_{i=1}^{N} \text{Occur}_{F_k}(i) \cdot f_{\text{req}(i)} \cdot T \\ \Leftrightarrow \quad \frac{\text{TotalRequests}_{F_k}}{T} &= \sum_{i=1}^{N} \text{Occur}_{F_k}(i) \cdot f_{\text{req}(i)} \\ \Leftrightarrow \quad f_{\text{req}(k)} &= \sum_{i=1}^{N} \text{Occur}_{F_k}(i) \cdot f_{\text{req}(i)}. \end{aligned} \tag{3}$$

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):

$$f_{\text{up}(k)} = \sum_{i=1}^{N} \text{Occur}_{F_k}(i) \cdot f_{\text{up}(i)}. \tag{4}$$

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

$$f_{\mathrm{mat}(k)} = \max(f_{\mathrm{req}(k)}, f_{\mathrm{up}(k)}).$$

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 the CF is equal to the request frequency of the fragment and the compromise is shown in Equation (5):

$$\mathrm{Compromise}_{F_k} = f_{\mathrm{up}(k)} - f_{\mathrm{req}(k)}. \tag{5}$$

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 CF is equal to the update frequency and the compromise 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:

$$f_{\mathrm{mat}(k)} = \sum_{i=1}^{N} \mathrm{Occur}_{F_k}(i) \cdot f_{\mathrm{req}(i)} \quad \text{when } f_{\mathrm{req}(k)} < f_{\mathrm{up}(k)} \tag{6}$$

and

$$f_{\mathrm{mat}(k)} = \sum_{i=1}^{N} \mathrm{Occur}_{F_k}(i) \cdot f_{\mathrm{up}(i)} \quad \text{when } f_{\mathrm{req}(k)} \geqslant f_{\mathrm{up}(k)}. \tag{7}$$

It is clear that the CF in both cases does not cause any staleness (on average) in the actual viewed data. A compromise is introduced only to the materialization frequency of the fragments (in the case where $f_{\mathrm{req}(k)} < f_{\mathrm{up}(k)}$) 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 and memory utilization, total requests) pass a specific threshold then the materialization frequencies must be reduced. This procedure must also be reflected in the technique that we have presented since the Web server's utilization parameters are directly reflected in their total request and update frequencies. Thus, if we select appropriate threshold values for the total request and update frequencies for the Web server, we can reduce all the fragment materialization frequencies proportionally when these values are passed at some point in time. A run-time materialization must always keep track of server utilization parameters through the server's monitoring functions and adjust materialization accordingly.

5. Cost models

The cost model of a run-time management policy is the basis of its performance evaluation. In order to have a theoretical view on the efficiency of a policy, one must construct a cost model corresponding to the policy. In this paragraph we construct four cost models that correspond to the four experimental configurations that we will use in our performance evaluation in the next paragraph. The four cost models correspond to the following configurations:

- A configuration where all pages in a Web site are static and consequently no policy is implemented.
- A configuration where all pages in a Web site are dynamic and no policy is implemented.
- A configuration where a Web page-centric policy is implemented. This approach considers that all Web pages in the Web site are materialized in frequent time intervals.
- A configuration where a Hybrid run-time management policy is implemented. This approach considers that all fragments of the Web pages in a Web site are materialized in frequent time intervals and it actually implements the proposed hybrid policy.

The parameters that we will use to construct the four cost models are the following:

- N: the number of pages in the Web site.
- L: the number of fragments contained in all Web pages.
- $F_{\text{p-up}_i}$ and $F_{\text{f-up}_i}$: the update frequencies of Web page i and fragment i.
- $C_{\text{p-gen}_i}$ and $C_{\text{f-gen}_i}$: the cost of generating Web page i and fragment i.
- $F_{\text{p-ac}_i}$: the access frequency of Web page.
- $C_{\text{p-dread}_i}$ and $C_{\text{f-dread}_i}$: the cost of reading page i and fragment i from the disk cache.
- $C_{\text{p-mread}}$ and $C_{\text{f-mread}}$: the cost of reading page i and fragment i from the memory.
- $C_{\text{p-dwrite}}$ and $C_{\text{f-dwrite}}$: the cost of writing page i and fragment i to the disk cache.

The cost models presented in this section will evaluate three types of server costs. These are:

The Access cost. This cost represents the resources that the server must allocate to serve Web pages to users. The notation used for this cost is C_{A}.

The Construction cost. This cost represents resources that the server must allocate in order to construct pages before serving them to users. Construction costs may apply when considering dynamic pages that must be constructed with the inclusion of data stored in databases or when fragments must be included in Web pages before they are sent to users (e.g., a Server Side Includes approach). The notation used for this cost is C_{C}.

The Materialization cost. This cost represents the resources allocated by the server to the materialization process. It is equivalent to the cost of constructing a page and saving it to disk or memory cache. The notation used for this cost is C_{M}.

5.1. *All-static pages cost model*

The cost model presented in this paragraph evaluates the server cost when hosting a Web site that consists of N static pages. The total server cost during a specific time interval is the following:

$$C_{\text{total}} = C_{\text{A}} = \sum_{i=1}^{N} f_{\text{p-ac}_i} \cdot C_{\text{d-pread}_i}. \tag{8}$$

In this approach the server uses resources only when users access the static ages already stored on disk. Thus, the total cost is equal to the sum of the access costs to every static page. This is simplest case and the obvious drawback is the staleness of the static Web pages that are served to users.

5.2. *All-dynamic pages cost model*

The cost model presented in this paragraph evaluates the server cost when hosting a Web site that consists of N dynamic Web pages and no materialization policy is applied. The total server cost in this case is equal to the page construction cost plus the page access cost.

Thus,

$$C_{\text{total}} = C_{\text{C}} + C_{\text{A}} = \sum_{i=1}^{N} (f_{\text{p-ac}_i} \cdot C_{\text{p-gen}_i}) + \sum_{i=1}^{N} (f_{\text{p-ac}_i} \cdot C_{\text{p-mread}_i}). \tag{9}$$

This approach introduces a page construction cost. This cost represents the server resources that must be allocated in order to execute the database queries in a page and then construct the corresponding HTML page that must be sent to the user. The constructed HTML page usually resides in memory after its construction, thus the access cost of the page is less than having the page on disk ($C_{\text{p-mread}_i}$ instead of $C_{\text{p-dread}_i}$).

5.3. *The Web page based policy*

The cost model presented in this paragraph evaluates the server cost in the case of hosting a Web site that consists of N frequently updated Web pages that are materialized with a frequency equal to CF. The materialization policy follows the architecture presented in Figure 1 and is executed with a frequency represented by CF (Compromise Factor):

$$C_{\text{M}} = \sum_{i=1}^{N} \big(\text{CF} \cdot (C_{\text{p-gen}_i} + C_{\text{p-dwrite}_i})\big). \tag{10}$$

The cost of accessing a Web page C_{A} is given by the following formula:

$$C_{\text{A}} = \sum_{i=1}^{N} f_{\text{p-ac}_i} \cdot C_{\text{p-dread}_i}. \tag{11}$$

The total cost for materialization and accessing the Web page over a period of time is

$$C_{\text{total}} = C_{\text{M}} + C_{\text{A}} = \sum_{i=1}^{N} \big(\text{CF} \cdot (C_{\text{p-gen}_i} + C_{\text{p-dwrite}_i}\big) + \sum_{i=1}^{N} f_{\text{p-ac}_i} \cdot C_{\text{p-dread}_i}. \tag{12}$$

This policy can keep all Web pages consistent to database changes when

$$\text{CF} = \max(F_{\text{p-up}_1}, \ldots, F_{\text{p-up}_N}). \tag{13}$$

When looking at Equation (13) on a page to page basis it implies that when the update frequency of a page equals to its materialization frequency, the data contained in the page is completely consistent to database changes. But can the value of CF be less than $F_{\text{p-up}}$ and at the same time keep Web pages consistent to database changes? We argue that this can appear to be happening, from a user's point of view. Let us consider a Web page whose data is updated with a frequency of 9 changes/min ($F_{\text{p-up}} = 9$ changes/min). The Web page is materialized with CF $= 6$ materializations/min. We consider the following cases:

- The request frequency of the page $F_{\text{p-ac}}$ is equal to 11 requests/min (CF $<$ $F_{\text{p-up}}$ $<$ $F_{\text{p-ac}}$). In this case the selected value of CF produces stale responses to user requests. In order to keep the consistency between database changes and corresponding Web pages we must increase the value of CF to be at least equal to the update frequency of the pages.
- The request frequency of the page f_{r_i} is equal to 5 requests/min ($F_{\text{p-ac}} <$ CF $< F_{\text{p-up}}$). In this case, even though the page's data changes very frequently, users access the page at a lower request rate on average. It is clear that the value of the CF does not have to be equal to the frequency of data change since users can be satisfied by a value just bigger than the request rate. Thus, when $F_{\text{p-ac}} < F_{\text{p-up}}$ the value of CF can be a value between $F_{\text{p-ac}}$ and $F_{\text{p-up}}$, preferably (in terms of server cost reduction) closer to $F_{\text{p-ac}}$.

The example shows that the Web page materialization frequency does not have to be equal to the Web page update frequency for all pages in a Web site. Some pages can be materialized with a rate lower than the actual change rate. This is possible through the CF concept. In following paragraphs we will present an algorithm that utilizes the CF concept and relates the update, materialization and request frequencies of Web pages to current server conditions. The outcome is a CF value for all Web pages in a Web site.

5.4. The Hybrid run-time management policy

In this paragraph we present the cost model of a Hybrid run-time management policy based on Web page fragments. This policy does not consider Web pages at all. Its implementation lies on the concept that Web pages are constructed by a number of Web fragments. Thus, materialization is carried out on a fragment basis. The total server cost of this approach is equal to the fragment materialization cost and the Web page access cost. The construction cost is embedded in the access cost and is equal to the page generation cost at every access.

The fragment materialization cost is

$$C_{\mathrm{M}} = \sum_{k=1}^{L} \big(\mathrm{CF} \cdot (C_{\text{f-gen}_k} + C_{\text{f-dwrite}_k})\big). \tag{14}$$

The access cost is equal to

$$C_{\mathrm{A}} = \sum_{i=1}^{N} (f_{\text{p-ac}_i} \cdot C_{\text{p-gen}_i}). \tag{15}$$

The construction cost is

$$C_{\text{p-gen}_i} = \sum_{k=1}^{j} C_{\text{f-mread}_k} \quad \text{with } 1 < j < k. \tag{16}$$

The total cost for the implementation of the Hybrid policy is

$$\begin{aligned} C_{\text{total}} = C_{\mathrm{M}} + C_{\mathrm{A}} = {} & \sum_{k=1}^{L} \big(\mathrm{CF} \cdot (C_{\text{f-gen}_k} + C_{\text{f-dwrite}_k})\big) \\ & + \sum_{i=1}^{N} \left(f_{\text{p-ac}_i} \cdot \sum_{k=1}^{j} C_{\text{f-read}_k} \right). \end{aligned} \tag{17}$$

The fragment based approach relies on the grouping of fragments with similar update frequencies. It provides some basic advantages over the previous policies. The basic advantage is that a Web page is not either fresh or stale when requested. It can be partially fresh. This means that some fragments can be fresh and some may be stale.

5.5. Cost models' evaluation

The cost models that we have described in this section should be evaluated in terms of performance. It is clear that the All-static model shows the minimum server costs. This is expected since this model does not require any materialization at run-time.

The All-dynamic model incldes a page construction cost at every request. In order to evaluate the All-dynamic model against the Web page based policy we must compare the materialization frequency (= CF) to the page access frequency ($f_{\text{p-ac}}$). Usually the access frequency is much larger than the materialization frequency, and thus, the All-dynamic policy is expected to show worse results. One must note that if the value of CF approaches the access frequency these two approaches will show similar results.

When comparing the Web page based policy to the Hybrid run-time management policy we find that their efficiency is based on the appropriate selection of the CF. It is not clear which will be better directly from the cost models. Their efficiency is shown in the following paragraphs through experimental results. Their cost models, on the other hand, can be very useful to evaluate their efficiency under specific circumstances.

6. Methodology

In order to evaluate the Compromise Factor concept in the context of a hybrid run-time management policy, we implemented four different experimental scenarios. Our primary goal was to simulate real Web site conditions. We wanted to evaluate our proposal against real Web conditions. We gave special attention to Web page construction techniques and Web page request patterns. In the following paragraphs we describe our four different experimental scenarios in detail. The scenarios that will be described below are experimentally evaluated in the following section. They are directly related to the cost models of the previous section and deal with the same Web site with the use of four management approaches.

6.1. Scenario 1: Static Web pages

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 [4,27]. 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.

6.2. Scenario 2: Dynamic Web pages

In this scenario we evaluated a Web site that consisted of dynamic 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, we constructed the dynamic pages in a way that they all issued 10 specific queries to the same database tables and also 10 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.

6.3. Scenario 3: Use of the Compromise Factor

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 run-time management policy was applied. The run-time 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 s. We chose to evaluate CF values less than 10 s because we wanted to simulate the materialization needs of popular applications such as stock quotes and online athletic results.

6.4. Scenario 4: The Hybrid run-time management policy

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 2 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 of the standard fragments every 10 s and the materialization of the differentiating fragments every 2 s.

7. Performance evaluation of scenarios

The performance evaluation had three 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.

The evaluation of Web page freshness at the client. This evaluation step aimed at measuring the degree of Web page freshness at the clients. We aimed at evaluating the percentage of fresh (and stale) data that reached the clients during different experiments.

7.1. Server performance

In this section of the performance evaluation, 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.

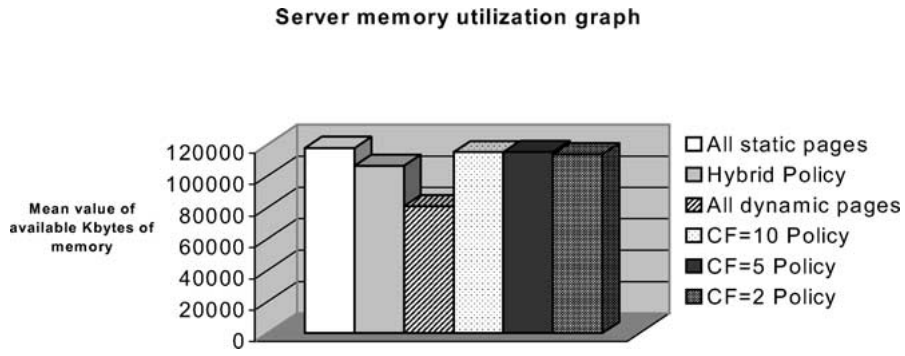

Figure 5. The server memory utilization graph for all approaches.

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

7.1.1. Memory utilization. Figure 5 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 policy. 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 Hybrid 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.

7.1.2. Server CPU utilization. Figure 6 shows the server's CPU utilization percentage 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 Hybrid 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.

7.1.3. Server throughput. Figure 7 shows the Web server's throughput by plotting the bytes sent by the Web server/s. The All-static policy shows the smallest throughput values and the CF $= 2$ policy shows the larger throughput values. The Hybrid policy shows the second largest value for throughput.

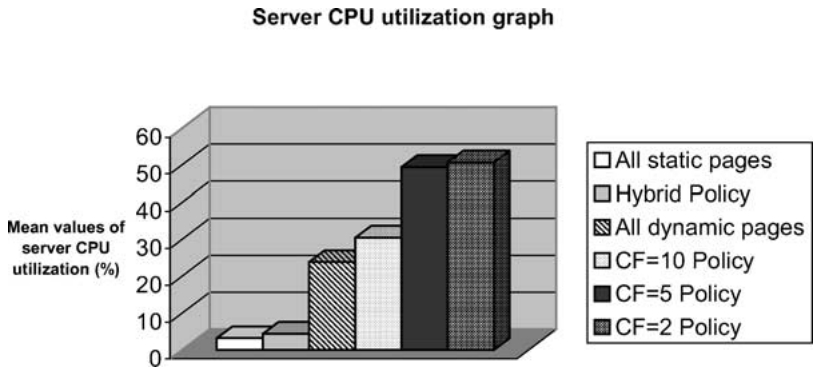

Figure 6. The server CPU utilization graph for all policies.

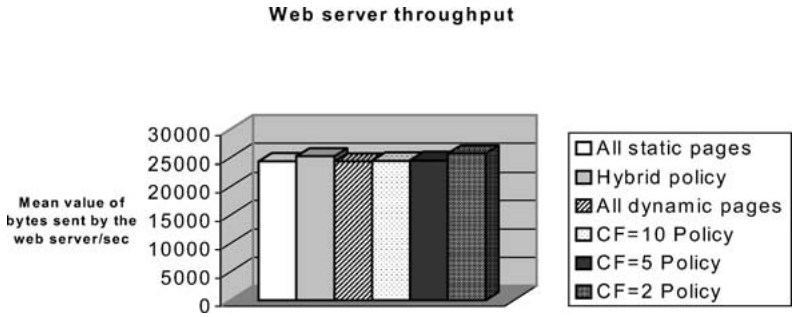

Figure 7. The Web server throughput graph for all the approaches.

7.2. 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 Hybrid policy with all the other policies and techniques and then show some interesting comparisons between the other policies.

Figure 8 compares the Hybrid policy with the CF = 2, the CF = 5, the CF = 10 policies and the All-static pages and All-dynamic pages techniques. The basic conclusions that may be extracted from the first set of graphs are the following:

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

The set of graphs in Figure 9 compare the different CF policies, the CF = 2 policy with the All-dynamic pages technique and the CF = 10 policy with the All-static pages technique.

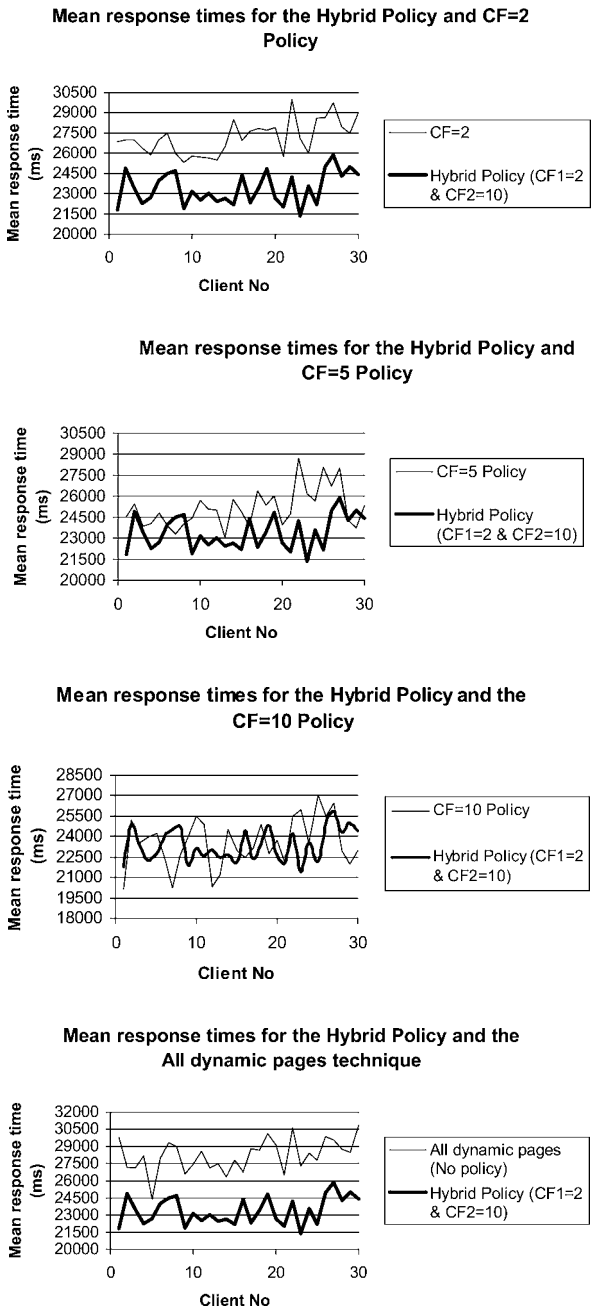

Figure 8. A comparison of the Hybrid run-time management policy with all other approaches.

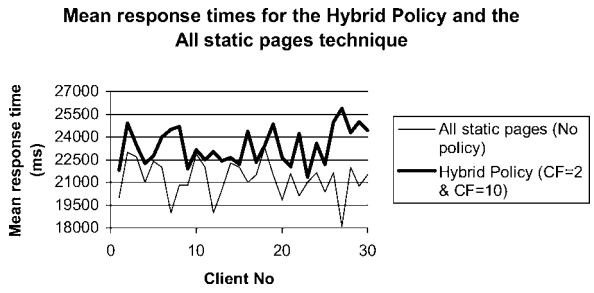

Figure 8. (Continued)

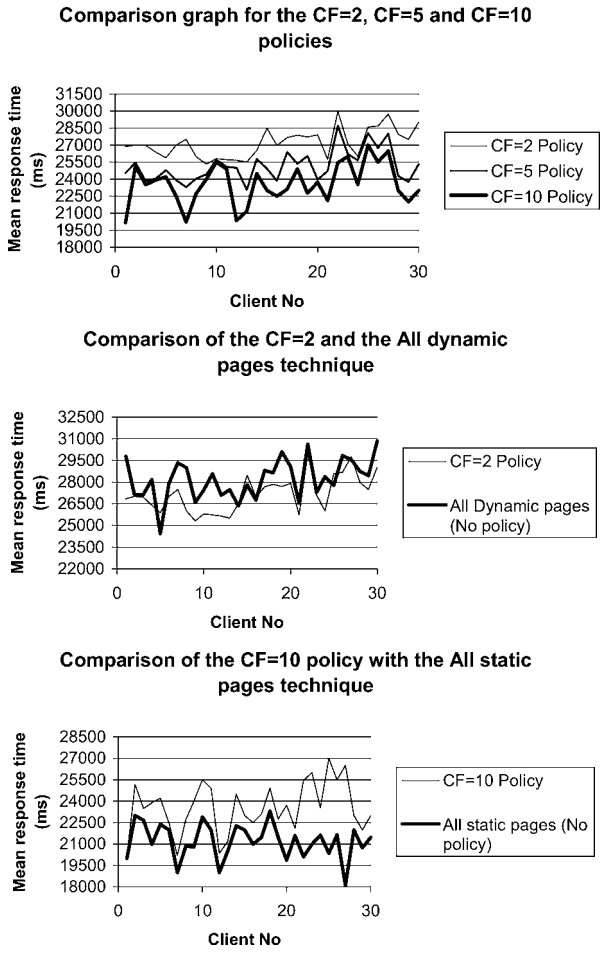

Figure 9. A comparison of all the similar management approaches.

The purpose of these graphs 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 hybrid 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 by the All-static approach, that cannot be considered an actual alternative for the management of a data-intensive Web site.

7.3. Web page freshness evaluation

The hybrid fragment based policy performs well in terms of client response times and server resource utilization as shown in the previous paragraphs. Another important parameter though, in order for the policy to be valuable, is related to its data consistency capabilities. In other words, how fresh can we keep our Web pages when implementing the proposed policy. The answer to this question is "As fresh as we want them to be." In the case of our experiments the change model was exactly the same as the update model. This means that the standard components changed every 10 s and the differentiating components changed every 2 s. Thus, the data sent to users were always updated when implementing the hybrid policy.

A freshness study must always be carried out before the implementation and the set-up of a policy. The CF concept enables this in a very easy way. The freshness study involves three basic parameters. These parameters are:

- the update frequency of every fragment,
- the materialization frequency of every fragment,
- the request frequency of every fragment.

A freshness study must be carried out on a fragment basis. The goal is to come up with a freshness metric in order to evaluate the CF s that we select in the hybrid policy. Let us consider the following example.

Example 3. A Web site that consists of 5 pages and a total of 20 fragments (4 fragments per page). The fragments all have different update frequencies (all values in the example are random except the requests compromised that were calculated). The 5 pages also have different request frequencies. The question is: "When selecting a CF for a fragment what is the freshness compromise that we make on average?".

In our evaluation:

- Total requests $= 121$.
- Total fragment compromise fraction $=$ Total fragment requests compromised/total fragment requests $= 39/121$ requests compromized $(= 32\%)$.

Table 1. A CF freshness evaluation table

Update frequency	Request frequency	Materialization frequency = CF	Requests compromised
4	6	4	0
10	2	7	0
12	3	22	0
22	22	8	14
3	7	9	0
4	6	24	0
7	2	5	0
25	3	30	0
20	22	9	13
11	7	5	2
3	6	7	0
30	2	14	0
42	3	18	0
6	22	11	0
27	7	7	0
23	6	25	0
5	2	20	0
26	3	13	0
22	22	12	10
17	8	8	0

In our example we found that 32% of the requests to the Web page fragments were compromised (stale). This is not such a good result in terms of fragments. But this is not necessarily so, when we transfer our study from the fragment level to the Web page level. The best case scenario would be that all compromised fragments would be part of the same Web page. In this case we would have to take into consideration the Web page request frequency instead of the fragment request frequency. For example, if the page that contained all the compromised fragments was requested with a request rate of 22 requests/min then the compromise frequency of the Web site would be $22/121 \cdot 100 = 18\%$. This would mean that $18/100$ requests to the Web site would return stale data.

From the example it is obvious that we can compute the freshness to requests by using the concept of the CF. By changing the value of CF, one can change the freshness metric values and come up with acceptable values of the CF. It is obvious that this freshness metric must be recomputed at frequent time intervals. That is why it must be an integral procedure of a run-time management policy.

8. 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 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. There are several parameters that should play a role in materialization frequency 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 [3]. 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.

In this paper we have proposed a hybrid run-time management policy for data intensive Web sites. The necessity of such a policy is inherent in every data intensive Web site. We then described the cost model of the hybrid policy and evaluated it. Our evaluation consisted of three steps: the evaluation of server utilization parameters, the evaluation of server response times (measured at the clients) and the evaluation of Web page freshness. The results are very interesting, since it was shown that by adopting a hybrid policy, the performance of the Web server can improve. 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, caused on Web servers.

Acknowledgements

We would like to thank our colleagues, Afodite Sevasti, Apostolos Gkamas and Vaggelis Kapoulas, for their help, suggestions and review efforts.

References

[1] M. Arlitt, R. Friedrich, and T. Jin, "Performance evaluation of Web proxy cache replacement policies," in *Proceedings of Performance Tools'98*, Lecture Notes in Computer Science, Vol. 1469, 1998, pp. 193–206.

[2] E. Baralis, S Paraboschi, and E. Teniente, "Materialized views selection in a multidimensional database," in *VLDB*, 1997.

[3] C. Bouras and A. Konidaris, "An algorithm for handling hybrid run-time management policies in data intensive Web sites," work in progress.

[4] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "On the implications of Zipf's law for Web caching," in *Proceedings of IEEE INFOCOM'99*, New York, March 1999.

[5] J. Challenger, P. Dantzig, D. Dias, and N. Mills, "Engineering highly accessed Web sites for performance," *Web Engineering*, Y. Deshpande and S. Murugesan, Eds., Springer-Verlag.

[6] J. Challenger, A. Iyengar, and P. Dantzig, "A scalable system for consistently caching dynamic Web data," in *Proceedings of IEEE INFOCOM'99*, New York, NY, March 1999.

[7] J. Challenger, A. Iyengar, and K. Witting, "A publishing system for efficiently creating dynamic Web content," in *Proceedings of INFOCOM 2000*, Tel Aviv, Israel, March 26–30, 2000.

[8] D. Florescu, M. Fernandez, J. Kang, A. Levy, and D. Suciu, "Catching the boat with Strudel: Experience with a Web-site management system," in *Proceedings of ACM SIGMOD Conference on Management of Data*, Seattle, WA, 1998.

[9] D. Florescu, A. Levy, and A. Mendelzon, "Database techniques for the World-Wide Web: A survey," *SIGMOD Record* 27(3), 1998, 59–74.

[10] D. Florescu, A. Levy, D. Saciu, and K. Yakoub, "Optimization of run-time management of data intensive Web sites," in *Proceedings of the 25th VLDB Conference*, Edinburgh, Scotland, 1999.

[11] H. Gupta, "Selection of views to materialize in a data warehouse" in *ICDT*, 1997.

[12] H. Gupta and I. S. Mumick, "Selection of views to materialize under a maintenance cost constraint," in *ICDT*, 1999.
[13] M. A. Habib and M. Abrams, "Analysis of sources of latency in downloading Web pages," in *WebNet 2000*, San Antonio, TX, October 30–November 4, 2000.
[14] K. Yagoub, D. Florescu, V. Issarny, and P. Valduriez, "Caching strategies for data-intensive Web sites," in *Proceedings of the International Conference on Very Large Data Bases (VLDB 2000)*, Cairo, Egypt, September 10–14, 2000.
[15] A. Iyengar and J. Challenger, "Improving Web server performance by caching dynamic data," in *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, Monterey, CA, December 1997.
[16] A. Lambrinidis and N. Roussopoulos, "WebView Materialization," in *Proceedings of the ACM SIGMOD 2000*, Dallas, TX, May 2000.
[17] Y. Li and K. Lu, "Performance Issues of a Web Database," in *Proceedings of Eleventh International Workshop on Database and Expert Systems Applications*, Greenwich, London, UK, 4–8 September 2000.
[18] B. Liu, "Characterizing Web response time," Master of Science in Computer Science Thesis, Virginia Polytechnic Institute and State University, 1998.
[19] G. Mecca, P. Atzeni, A. Masci, P. Merialdo, and G. Sindoni, "The Araneus Web-based, management system," in *Exhibits Program of ACM SIGMOD'98*, 1998.
[20] C. Olston and J. Widom, "Offering a precision–performance tradeoff for aggregation queries over replicated data," in *Proceedings of the 26th International Conference on Very Large Data Bases*, Cairo, Egypt, September 10–14, 2000.
[21] T. Palpanas and B. Krishnamurthy, "Reducing retrieval latencies in the Web: the past, the present, and the future," Technical Report CSRG-378, Graduate Department of Computer Science, University of Toronto.
[22] B. Proll, H. Starck, W. Retschitzegger, and H. Sighart, "Ready for prime time pre-generation of Web pages in TIScover," in *WebDB'99*, Philadelphia, PA, June 3–4, 1999.
[23] E. A. Rundensteiner, A. Koeller, and X. Zhang, "Maintaining data warehouses over changing information sources," *Communications of the ACM* 43(6), June 2000.
[24] The ARANEUS project home page, `http://www.dia.uniroma3.it/Araneus/`
[25] J. Wang, "A survey of Web caching schemes for the Internet," *ACM Computer Communication Review* 29(5), October 1999, 36–46.
[26] C. E. Wills and M. Mikhailov, "Studying the impact of more complete server information on Web caching," *5th International Web Caching and Content Delivery Workshop*, Lisbon, Portugal, May 22–24, 2000.
[27] G. K. Zipf, *Human Behavior and the Principle of Least Effort*, Addison-Wesley, 1949.