"1st International Symposium on Computing in Informatics and Mathematics (ISCIM 2011)"
in Collabaration between EPOKA University and "Aleksandër Moisiu" University of Durrës
on June 2-4 2011, Tirana-Durres, ALBANIA

Efficiency of cache-replacement algorithms while retrieving data from a relational database and XML files in a web based system

Lule AHMEDI[1], Hilmi HAXHIHAMZA[2]

[1]*Faculty of Contemporary Sciences and Technologies, SEE UNIVERSITY, Tetovo-MACEDONIA*
E-mail: l.ahmedi@seeu.edu.mk
[2]*Faculty of Contemporary Sciences and Technologies, SEE UNIVERSITY, Tetovo-MACEDONIA*
E-mail: hhilmi@gmail.com

ABSTRACT

Caching has been applied in Web based information systems in order to reduce the transmission of redundant network traffic and response latency by saving copies of the content obtained from the Web closer to the end user. The efficiency of caching is influenced to a significant extent by the cache replacement algorithms which are triggered when the cache becomes full and old objects must be evicted to make space for the new ones.

This paper presents a framework that can be used in future work to tune cache-replacement algorithms while data is simultaneously retrieved from a relational database and XML files in a web based environment, by a large number of end-users. Three replacement policies are considered: Least Recently Used (LRU), Least Frequently Used (LFU) and Lowest Latency First (LLF). The experimental results obtained from the framework show that data caching greatly improves the overall performance of web based systems, and the type of the applied cache replacement policy also plays an important role in the performance. In the scenarios considered in this paper, the LLF algorithm produced the best performance when retrieving data from a relational database, while the LFU algorithm was the most efficient algorithm when data was retrieved from an XML file.

INTRODUCTION

With the expansion of the popularity of the Internet in the last decade, the Web based applications became very important in the software industry, mainly because of the flexibilities that they offer while running on a web browser. Today, under normal circumstances, people access the web daily for various purposes: to check the e-mail, read the latest news, for e-commerce, for entertainment etc. But, although most of today's web users have switched to faster Internet connections,

they still do not decrease the transmission of redundant data being sent across the Web. This could be weather related data, news, stock quotes, sport scores, course notes, technical papers, exchange rate information etc. If a large number of users attempt to access a web page at the same time, then there is a high probability that they may experience problems in getting connected to that site because the website is unable to cope with the load, and the responses received from it are either slow or even completely absent. Thus, in order to provide an enjoyable working and surfing experience to the end-users, when developing web applications which are subject to be accessed by a potentially large group of end-users, it is important to pay attention to the time that it takes for a page to be requested from the server and rendered on the user's web browser.

Web caching is a technology aimed at reducing the transmission of redundant network traffic by saving copies of content (obtained from the Web) closer to the end user, in order to enable quicker access to the content. Data-driven applications with large and complex queries that commonly consume the majority of the application's execution time can often be improved by storing the results of expensive database queries in the server or client memory (cache). Caching generally enhances the performance of web applications, but it may also produce limitations by occupying space in the memory where the cache is stored. In this case, a cache-replacement algorithm must be called to purge the cache by removing suitable contents from it. The efficiency of caching is influenced to a significant extent by the cache replacement algorithms which aim to minimize the hit ratio; byte hit ratio, the cost of access and the latency [1]. This paper presents a framework to simulate cache-replacement algorithms while retrieving large amounts of data from a relational database and XML files in a web based environment. The data is retrieved simultaneously by a large group of users. Three replacement algorithms were evaluated in our system:

- Least Recently Used (LRU),
- Least Frequently Used (LFU).
- LLF (Lowest Latency First)

The results of the experiment show that data caching greatly improves the performance of web based systems that frequently retrieve huge amounts of data. The use of a suitable cache replacement strategy can enhance the performance even more.

This research paper contributes towards identification of methods for development of more optimized and faster web applications, by implementing proper caching strategies and cache-replacement policies. It may also serve as a solid foundation for further investigation and improvement of the existing cache replacement algorithms.

Related Work

In this section, we introduce the literature and research papers that are related with our work and which were used as a basis for our research.

S. V. Nagaraj [1] states that the efficiency of proxy caches is influenced to a significant extent by document replacement algorithms. Cache replacement algorithms aim to minimize various parameters as the hit ratio, the byte hit ratio, the cost of access and the latency. T. Partl [2] analyzed the performance of four common cache clean-up algorithms: LRU, Space Working Set (SWS), Space-Time Working Set (STWS) and Space Time Product (STP). He states that the choice of the clean-up algorithm affects both the hit/access ratio and the actual time the users save by using the cache. The analysis of the algorithms was performed through an experiment where a public caching proxy was set up to collect data for the experiment. He tested the hit to access ratio and time gain of the algorithms on two different cache sizes. The results showed that a high hit/access ratio does not guarantee a high time gain. Similarly, Busari [3] explores the performance of cache clean-up algorithms, but he concentrates exclusively on LFU algorithm and several of its variations (LFU-Aging, LFU* and LFU*-Aging). The experimental part is performed by using a trace-driven simulation to determine the performance of the algorithms. From the results, LFU-Aging and LFU*-Aging provided best performance. Cárdenas, Gil, Sahuquillo and Pont in [4] present a framework to simulate web proxy cache systems which provides an environment to simulate and explore cache management techniques. Zahran [5] states that the efficiency of the replacement policy affects both the hit rate and the access latency of a cache system.

This research paper is particularly related to the articles mentioned in [2-4] which evaluate the performance of cache-cleanup policies by proposing and using various methods. However, our work differs from the above mentioned papers since its main objective is to evaluate cache replacement algorithms while caching data from a database or an XML file in web applications, while the related papers are generally concerned with caching of data in computer hardware components and caching of web documents.

PROPOSED FRAMEWOK AND EXPERIMENTAL DESIGN

The efficiency of cache replacement algorithms was measured in a custom caching framework which was developed in C# programming language specifically for this purpose. The framework can simulate different scenarios where a large number of users simultaneously retrieve data from a relational SQL database and XML files. The amount and structure of the data is same both in the database and XML file.

The caching algorithms are implemented and maintained in a specific caching module which runs as a separate process on the application server. All requests that are directed to the application server are initially received from the caching component. When the server receives a request for data, it firstly checks if it is located in the cache. If the cache contains a copy of the requested object, it is

retrieved from the cache and returned to the user. Otherwise, the object must be requested from the server, and at the same time its copy should be stored in the cache in order to avoid multiple executions of the same queries. The diagram in Figure 1 presents a high-level overview of the system.

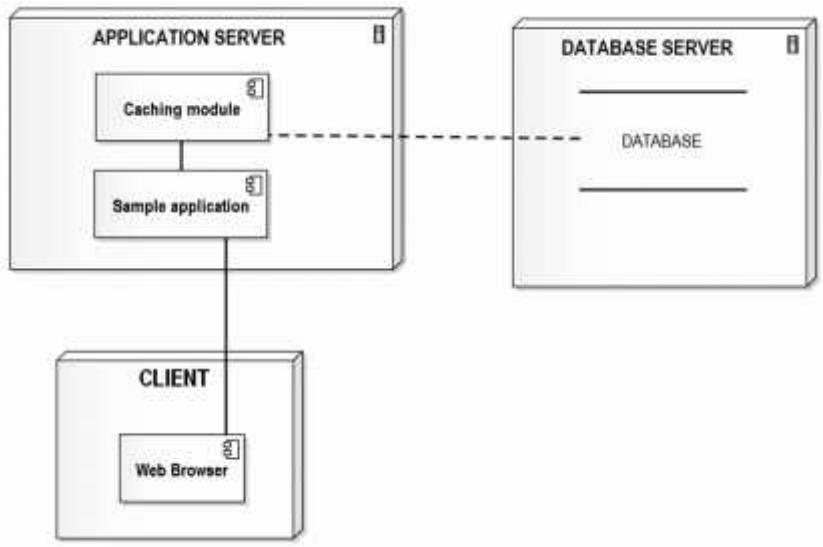

Figure 3 A high level overview of the system

The data retrieved from database and XML files is stored in separate objects (also known as datasets) in the cache. Three factors which can be manually configured in the framework are important in the experimental phase:

- cache size,
- number of simultaneous users that retrieve data,
- cache-replacement algorithm used.

Cache Size

The cache size represents the maximum amount of data (in Megabytes) that can be stored in the cache. When the maximum limit for the data that can be stored in the cache is reached, a cache-replacement policy is initiated. In our experiments, 3 cases were explored:

(1) Cache size = 50 MB

(2) Cache size = 100 MB

(3) Cache size = 200 MB

Number of Users

The number of users represents the number of simultaneous requests for the same data from a database or XML file. The data are stored in objects known as datasets. The size of a dataset is same in all cases. The following scenarios were considered:

(1) Number of users = 100

(2) Number of users = 500

(3) Number of users = 1000

Cache-replacement Algorithm

When the cache memory becomes full and there is no place for a new object to be stored, then the caching component must trigger a replacement policy. The replacement policy is based on a cache-replacement algorithm which must free space from the cache according to a certain criteria. In our experiment, three well known replacement algorithms were evaluated:

- LRU (Least Recently Used) - the objects with the oldest requests are deleted first. This algorithm uses a structure that stores the time when the object was lastly used.
- LFU (Least Frequently Used) - the objects which are used less are deleted from the cache first. If all objects are used in same frequency, the objects that should be deleted are selected in a random order.
- LLF (Lowest Latency First) - this algorithm keeps the average latency to a minimum by first expelling the object with the lowest download latency (the smallest object).

Table 1 The list of scenarios considered in the experimental part

Cache size (MB)	Number of users	Cache-replacement policy
50	100	LRU, LFU, LLF
500	100	
1000	100	
50	500	
500	500	
1000	500	
50	1000	
500	1000	
1000	1000	

RESULTS ANALYSIS

All planned scenarios were performed in five consecutive series and the data retrieval time was measured for each scenario. Then, from the measured times, an average retrieval time was calculated. Finally, after the comparison of the calculated average retrieval times, the most efficient algorithm was chosen.

Most of the requests at the beginning of the simulation were not located in the cache ("cold misses"). In order to obtain more accurate results, we tried to avoid them as much as possible by firstly "warming-up" (initializing) the cache with random objects before starting to measure the actual performance of the algorithms.

The average retrieval times (in seconds) obtained from the experiments which were performed while reading data from a relational database are showed in Table 2. As it can be seen, the performance of the web application is improved significantly when data caching is applied. LLF (Lowest Latency First) replacement algorithm produced the best results and its performance was faster than the LRU algorithms for approximately 15 seconds. A graphical representation of the results can be observed in Figure 1.

Table 2 Results obtained from experiments where data was retrieved from a database in five consecutive series with and without caching of the data

Algorithm	Without caching (seconds)	Series of retrieved data from a relational database by using caching (Measured in seconds)					
		1	2	3	4	5	Average
LRU	152.219	22.87	21.98	21.544	22.262	21.887	21.898
LFU	146.231	80.387	17.55	17.519	17.816	17.144	33.217
LLF	158.83	24.196	17.207	17.035	16.771	17.005	18.443

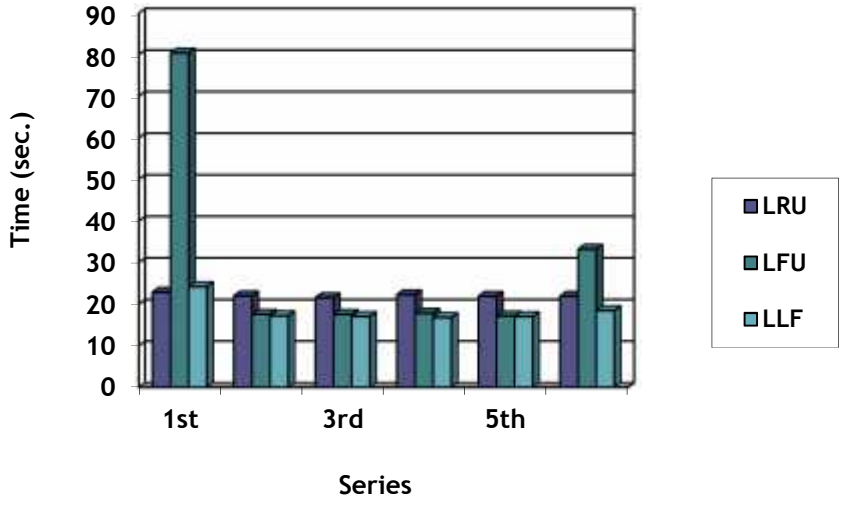

Figure 1 Graphical representation of average data retrieval times while applying LRU, LFU and LLF cache-replacement algorithms in scenarios where data is retrieved from a relational database, in five consecutive series

The results obtained from the scenarios in which data was retrieved from XML files are presented in Table 3. They prove that in these cases too, caching significantly improves the performance of the web applications, but give slightly different results when cache replacement strategies are applied. Namely, in the XML-based scenarios, LFU (Least Frequently Used) algorithm provided the best average performance in all cases. The graph in Figure 2 represents the results obtained from these scenarios.

Table 3 Results obtained from the experiments where data was retrieved from an XML file in five consecutive series with and without caching of the data

Algorithm	Without caching (seconds)	Series of retrieved data from XML file by using caching (Measured in seconds)					
		1	2	3	4	5	Average
LRU	124.532	43.914	40.857	40.045	39.453	39.39	41.067
LFU	126.213	53.134	37.487	25.568	25.054	25.132	33.275
LLF	114.241	82.977	22.339	22.386	22.714	22.62	37.604

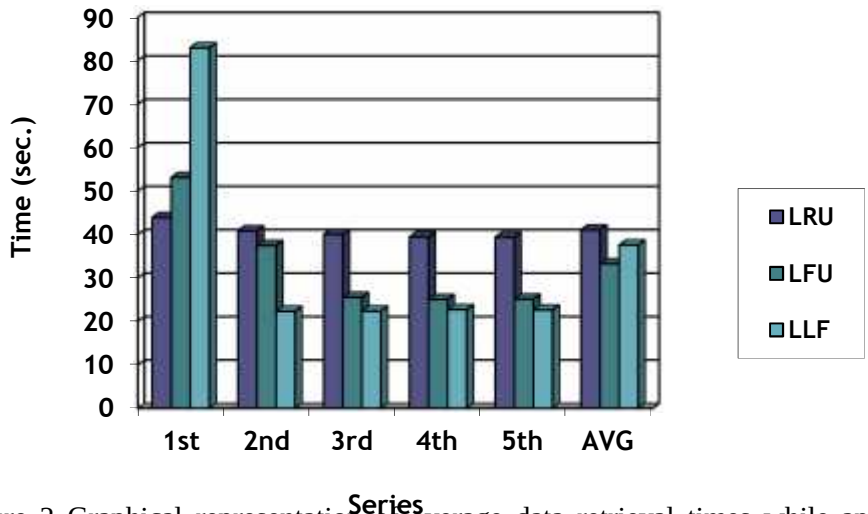

Figure 2 Graphical representation of average data retrieval times while applying LRU, LFU and LLF cache-replacement algorithms in scenarios where data is retrieved from an XML file, in five consecutive series

CONCLUSION

The results obtained from our simulation framework prove that web-caching techniques can significantly improve the performance of web based applications which frequently read data from databases or XML files. They are convenient especially in scenarios where huge amount of static data should be retrieved multiple times.

The use of a proper cache replacement strategy in certain scenarios is an important factor which can considerably enhance the overall efficiency of web caching. LLF (Lowest Latency First) replacement algorithm, which keeps the average latency of the objects in cache to a minimum by first removing the object with the lowest download latency, gives the best results in cases where the data is retrieved by executing a query against a relational database. On the other hand, LFU (Least Frequently Used) algorithm which firstly deletes objects with the oldest requests provides the best performance in scenarios where the data is retrieved from an XML file.

REFERENCES

[1] Nagaraj S. V. (2004), Web Caching and its Applications, *Kluwer Academic Publishers*

[2] Partl T. (1996), A Comparison of WWW Caching Algorithm Efficiency, *ICM Workshop on Web Caching, Warsaw, Poland*

[3] Busari M. (1999), Comparison of Web Caching Algorithms in Web Proxies, *University of Saskatchewan in Canada*

[4] Cárdenas L.G., Gil J.A., Sahuquillo J, Pont A. (2005), Emulating Web Cache Replacement Algorithms versus a Real System, *Computers and Communications, 2005. ISCC 2005. Proceedings. 10th IEEE Symposium*

[5] Zahran M. (2007), Cache Replacement Policy Revisited, *Proceedings of the 6th Workshop on Duplicating, Deconstructing, and Debunking, San Diego, CA, USA*