A new Web Cache Replacement Approach based on Internal Requests factor

Amany Sarhan, Ahmed M. Elmogy, Sally Mohamed Ali
at Computers and Control engineering department, Tanta University, Egypt
BSc. Of Computers and Control Engineering, Tanta University, Egypt

Summary
The increasing demand for World Wide Web (WWW) services has led to a considerable increase in the amount of Internet traffic. As a result, the network becomes highly prone to congestion which increases the load on servers, resulting in increasing the access times of WWW documents. Thus, web caching is crucial for reducing the load on network, shorten network latency and improve clients' waiting time. Many web cashing systems and policies have been proposed to determine which objects to evict from the cache memory to accommodate new ones. Most of these systems and policies are mainly based on the enhancement of a well-known scheme called the Least Frequently-Used (LFU) scheme. Although most of the proposed schemes could overcome the disadvantages of the LFU, they still have lots of overhead and are difficult to implement. This work proposes a replacement policy with better characteristics. Also, the developed system is easier to be implemented than the previous approaches. The proposed policy considers the internal requests generated in each web site and add this factor to the frequency to select the evicted object. Another scheme which was developed in the literature to improve the LFU called Weighting Replacement Policy (WRP). Our research adds the internal requests factor to this policy to improve its performance and assure the effectiveness of this new factor. The simulation results show the effectiveness of the proposed approach compared with the earlier approaches.

Key words
Web Cache, Replacement Algorithms, LFU schemes, Hit Ratio.

1. Introduction

The rapid growth of the World Wide Web services has caused a tremendous and exponential increase in network traffic and page access latency. Thus a reliable and an efficient cashing mechanism is urgently needed. Caching is an old and well-known performance enhancement technique that is widely used in storage systems, databases, Web servers, middleware, processors, and other applications [1]. In all levels of storage hierarchy, performance of the system is related to the caching mechanism [2]. Web caching is the temporary storage of remote web objects on a local server [3,4]. Web caching can effectively decrease network traffic volume, and reduce the latency problem [5] by bringing documents closer to the clients. As a result, Web cache servers are widely deployed in many places throughout the Web [3]. Three distinct approaches to web caching currently exist, including: client-side caching [6,7], server-side caching [8], and proxy caching [9]. Client-side caching refers to caches that are built into most web browsers, which caches Internet objects for a single user, but from a variety of servers. Server-side caching (also known as reverse caching) refers to caching that is placed in front of a particular server to reduce the number of server requests [10]. Whereas, in proxy caching, proxy servers serve as intermediary between users and central servers. User's request is forwarded to the web server by the proxy server. When the server returns the requested resource to the proxy server, the proxy stores a copy in its cache such that further requests to the same resource by the same user or another user are met at the proxy without contacting the web server again.

A replacement policy is increasingly becoming one of the fundamental components of the caching mechanisms to act as a key technology for high-speed multimedia services delivery. The replacement policy acts as a decision rule for evicting a page currently in the cache to make room for a new page in case of cache saturation. Thus, determining when and what to evict from the cache [11]. Early versions of replacement algorithms depended only on a single factor to decide the priority of the object in cache memory including; the first-in-first-out (FIFO) policy [12], the random replacement (RAND) policy [13], the least recently used (LRU) policy [14], the least frequently used (LFU) policy [15], and the LFU-aging policy [1]. More recent algorithms aim to keep in the cache the most valuable objects according to a cost function which combines multiple parameters to calculate the score of an object.

Towards finding a collection of algorithms that have a profound impact on the performance of the network, many caching and replacement algorithms have been proposed. In [16], a model for adaptive cache size control (MACSC) at runtime is proposed to automatically maintain the prescribed hit ratio. Thus, the minimum expected caching performance is guaranteed. Nimrod Megiddo in [1] proposed ARC adaptive replacement cache algorithm which outperforms the least-recently-used algorithm by dynamically responding to changing access patterns and continually balancing between workload recency and frequency features. C. Umapathi et al. in [9, 17] described a Web caching scheme that capitalizes on Web log

Manuscript received March 5, 2015
Manuscript revised March 20, 2015

methods. The authors reported that the perfected documents are accommodated in a dedicated part of the cache, to avoid the drawback of incorrect replacement of requested documents. Also, in [2], an adaptive replacement policy is proposed . This policy has low overhead on system and is easy to implement. This model is named Weighting Replacement Policy (WRP) which is based on ranking of the pages in the cache according to three factors. Whenever a miss occurs, a page with the lowest rank point is selected to be substituted by the new desired page. The most advantage of this model is its similarity to both LRU and LFU, which means it has the benefits of both.

Although a great number of caching algorithms have been reported in the literature, important parameters have been ignored or given only sparse attention. In this paper, a new replacement strategy is presented. The new strategy is based on developing LFU algorithm by considering a new factor which is the number of internal requests generated from web sites and pages. This proposed algorithm guarantees to occupy the cache memory with the most benefit web pages and web site; we call this least frequency and internal request (LFIR). After that we use the new proposed factor to improve anther policy called Weighting Replacement Policy (WRP) to assure the effectiveness of the new factor. The rest of this paper is organized as follows. Replacement policies strategies and their review are presented in section 2. Section 3 introduces an overview of LFU and WRP schemes. The details of the proposed approach are presented in section 4. The conducted experimental results are introduced in Section 5. Finally, conclusion and future work are summarized in section 6.

2. Replacement Policy Strategies

When the cache is full, and new objects need to be stored, a replacement policy must be used to determine which objects to evict to make rooms for the new objects [9]. The replacement policy is considered as a decision rule for evicting a page currently in the cache. A poor replacement policy leads to high number of misses in the cache, and also increases the cache miss (hit) penalty. A lot of research, both in academia and industry is geared toward finding the best cache replacement policy [18]. In general, the replacement policies are classified as [19]:

•**Recency-based policies:** in this type of policies, recency is used as the primary decision making factor; most of the policies in this category are LRU variants. LRU policy is one of the most popular policies. It evicts the least recently referenced object first. This is particularly popular because of its simplicity and fairly good performance in many situations. It is designed on the assumption that a recently referenced document will be referenced again in the near future. LRU threshold is needed for estimating the expected time needed to fill or completely replace the cache content. This threshold is dynamically evaluated based on current cache size. One of the disadvantages of the LRU is that it only considers the time of the last reference and it has no indication of the number of references for a certain Web Object [20]. LRU-MIN algorithm [21] is much similar to LRU. It maintains a sorted list of cached documents based on the time document was last used. The difference between LRU and LRU-MIN is the method of selecting the candidate for replacement. When the cache needs to replace a document it searches from the tail of the sorted list. The first document whose size is larger than or equal to the size of the new document is removed.

•**Frequency-based policies:** the object popularity (or frequency count) is considered as the primary factor in this type [1]. As a result, this category of polices is suitable for systems in which the popularity distribution of objects is highly skewed, or in which there are many requests to Web sites having objects with very steady popularity (rarely changing abruptly). Such Web sites include online libraries, distant learning, and online art galleries. LFU is a simple example of this category[1]. LFU-Aging strategy attempts to remove the problem of cache pollution due to objects that become popular in short time by introducing an aging factor [21]. On the other hand, LFU-DA, a variant of LFU, avoids the cache pollution problem by using the dynamic aging technique, which adds a constant value to the frequency count of an object when it is accessed [9].

•**Size-based policies:** the object size is used as the primary factor [22], and this usually remove larger objects first. The size-based policy sorts cached documents by size. Documents with the same size are sorted by recency. When there is insufficient space for caching the most recently requested document, the least recently used document with the largest size is replaced [23].

•**Function-based policies**: each object is generally associated with a utility value [19]. This value is calculated based on a specific function incorporating different factors such as time, frequency, size, cost, latency, and different weighting parameters. GD-Size is the representative policy in this category. GD size deals with variable size documents by setting H to cost/size where cost is the cost of fetching the document while size is the size of the document in bytes, resulting in the Greedy-Dual-Size (GDS) algorithm. If the cost function for each document is set uniformly to one, larger documents have a smaller initial H value than smaller ones, and are likely to be replaced if they are not referenced again in the near future [24]. GDS algorithm was originally developed in the context of disk paging and, later on, was adapted to web caching. In the web caching version, its goal is to reduce the aggregate latency where size and location of data can vary widely [25].

•**Randomized polices:** policies with complex data structures motivate the consideration of randomized policies that require no data structures to support eviction decisions. A particularly simple example is RAND that evicts an object drawn randomly from the cache [26].

The Hit ratio which is defined as the number of requests met in the cache memory to the total number of requests is used to measure the performance of the replacement algorithms [27]. The higher the hit ratio, the better the replacement policy [5]. The Byte hit ratio is the ratio of byte in the total proxy cache size [27].

3. Least Frequency Used (LFU) Overview

Since a network proxy is required to serve thousands of requests per second, the overhead needed to do so should be kept to a minimum. To do so, the network proxy should evict only resources that are not frequently used. Hence, the frequently used resources should be kept at the expense of the not frequently used ones since the former have proved themselves to be useful over a period of time. Static resources of heavily used pages are always requested by every user of that page. Hence, the LFU cache replacement strategy can be employed by these caching proxies to evict the least frequently used items in its cache [25]. The standard characteristics of LFU method involve that the system is keeping track of the number of times a block is referenced in memory. When the cache is full and requires more room, the system will purge the item with the lowest reference frequency. The major disadvantage of the LFU replacement algorithm is that the web site keeps its place in cache memory for a long time even without using it again. This leads to wasting a certain size of cache memory since this element stayed in the memory with no change. Other disadvantages of LFU policies are that they require logarithmic implementation complexity in cache size, and they almost pay no attention to recent history.

Most web browsers are still using traditional replacement policies which are not efficient in web caching [28-30]. In fact, there are few important factors of web objects that can influence the replacement policy[26,30,31]. These factors include but not restricted to recency (i.e., time of the last reference to the object), frequency (i.e., number of the previous requests to the object), size, and access latency of the web object. These factors can be incorporated into the replacement decision. Most of the proposed approaches in the literature use one or more of these factors without paying attention of combining some of these factors. However, combination of these factors is still a challenging task as one factor in a particular environment may be more important than others in other environments [6,19,31].

In order to improve the performance of the LFU algorithm, a replacement based on Weighting-Replacement-Policy (WRP) is proposed in the literature [2]. This algorithm behaves like LFU by replacing pages that were not recently used and pages that are accessed only once. Ranking the pages in cache memory is done by three factors; the counter which shows the recency of block (L), the counter which shows the number of times that block buffer has been referenced (F), and the time difference (ΔT) between the last access time (Tc) and time of penultimate (Tp). Thus, the weighting value of block i can be computed by the following equation:

$$W_i = L_i/(F_i * \Delta T_i)$$

The time between each reference to a block would be at least one in its minimum case. In every access to buffer, if referenced block j is in the buffer then a hit is occurred and this policy will work as follows:

- L_i will be changed to $L_{i+1}$for every $i \neq j$.
- For $i = j$ first we put $\Delta T_i = L_i$, $F_j = F_j + 1$ and then $L_j = 0$

But if referenced block j is not in the buffer, a miss occurs and the algorithm will choose the block in buffer which its weighting function value is greater than the others. This will be done from top to down. In this way, if values of some object are equal to each other, the object which has upper place in the buffer will be chosen to be evicted from buffer. It means that our policy follows FIFO low in its nature. Let assume that a miss has been occurred and block k has the greatest weighting value and then it should be evicted from buffer. First we change Li to Li + 1 for every $i \neq k$ and then replace new referenced block with block k. The final step is to set all weighting factors of block k to their initial values. The weighting value of the blocks that are in buffer will be updated in every access to cache.

4. The Proposed Replacement Algorithm

The main objective of the proposed algorithm is to maximize the hit rate by keeping many web pages and sites in the memory which are carefully chosen. Thus, the developed algorithm gets the largest possible benefit of the cache memory. The proposed algorithm depends on sorting the web sites that should stay in the memory based on the number of internal requests generated by the user through the web site itself. An internal request means a request of another web page originating from this root web page. They may be: a page, an image, a video, a downloaded file or registration page. The internal requests may occur also in the subpages as shown in Figure1.

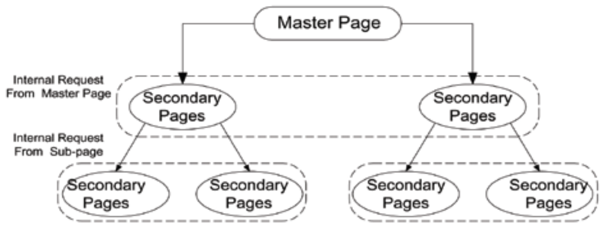
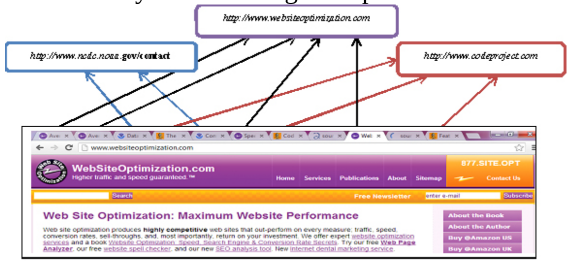


Figure 1. The internal requests generated by the site

To explain the counting methods of internal requests, assume we have three web sites, each site was used to access some other pages which we will call “internal requests”. These pages are stored in the cache with their frequency. Then we increment the root web site (i.e. which was used to produce this page) by one for each internal request generated from it (in addition to its frequency). In this way, each page will have a priority according to two factors: the number of times it was used (LFU based) and the number of internal requests produced from it. That is why we called it Least Frequently Used with Internal Requests algorithm (LFUIR). In this way, the sites that produce more internal requests will have higher priority and should be kept in cache as they are the main source of other pages. This will help the proxy to keep the root pages for longer time especially with the new technology of dynamic pages where the root page always guides to dynamic links leading to changing objects in time. There is no benefit of keeping each possible object that will change in time and will be of no use. The concept of the proposed algorithm can be demonstrated by the following example:



Web Site Optimization: Maximum Website Performance

Web site optimization produces **highly competitive** web sites that out-perform on every measure: traffic, speed, conversion rates, sell-throughs, and, most importantly, return on your investment. We offer expert website optimization services and a book Website Optimization: Speed, Search Engine & Conversion Rate Secrets. Try our free **Web Page Analyzer**, our free website spell checker, and our new SEO analysis tool. New internet dental marketing service.

Figure 2. The counting method of internal requests

Given that we have three web sites:

1- www.websiteobtimization.com
2- www.codeproject.com
3- www.ncdc.noaa.com

These sites were visited in the following order and were used to produce requests of other pages as follows.

http://www.websiteobtimization.com
http://www.websiteoptimization.com/speed/tweak/average-web-page/
http://www.websiteoptimization.com/sitemap/
http://www.websiteoptimization.com/publications/
http://www.websiteoptimization.com/services
http://www.codeproject.com
http://www.codeproject.com/Lounge.aspx?msg=4557115#xx4557115xx
http://www.codeproject.com/Lounge.aspxc#
http://www.codeproject.com/Lounge.aspx.java
http://www.ncdc.noaa.gov
http://www.ncdc.noaa.gov/cdo-web/webservice
http://www.ncdc.noaa.gov/cdo-web/#t=secondTabLink

The first site (http://www.websiteobtimization.com) was used to produce four pages; thus performing 4 internal requests. The second site (http://www.codeproject.com) was used to produce four pages performing 3 internal requests. Finally, the third site (http://www.ncdc.noaa.gov) was used to produce four pages performing 2 internal requests. Hence, the priorities of these 12 pages are: 5, 1, 1, 1, 1, 4, 1, 1, 3, 1, 1 respectively

5. Simulation Results

The proposed Least frequency and internal requests (LFIR) algorithm works through three steps. The first step is to calculate the frequency of all visited pages. The second step is to calculate the number of internal requests generated from master web page and subpages. Finally, it adds the internal requests for number of frequency for each page to decide the priority of data to select the item with the lowest priority to be replaced with the requested new item

In our experiment, about two hundred requests were taken as a sample of requests. The percentage of hit ratio is estimated for LFU and the proposed LFIR algorithms considering different sizes of proxy as shown in Table 1. As shown in Table 1, the developed LFIR algorithm improves the performance of web cache by increasing the hit ratio in different size of proxy (2MB, 5MB, 8MB, 10MB). The average enhancement compared with LFU is about 1.7%. This leads to improving the process of prefetching data from cache memory.

Table 1. Comparison between hit ratio in LFU and LFIR simulation algorithms.

Cache Size	LFIR hit	LFU hit	Improvement%
2MB	19.5%	17.5%	2.0%
5MB	36.5%	34%	2.5%
8MB	46%	44.5%	1.5%
10MB	47%	46%	1.0%

In order to maximize the hit ratio and improve the performance of WRP algorithm, we used Least Internal request (LIR) by adding it to WRP to create a new policy called WRPIR which based on WRP factors and the internal requests number of web pages. This can be described by the following equation:

$$W_i = \frac{L_i}{(F_i + IR_i) * \Delta T_i}$$

In our experiment, about two hundred requests have been taken as a sample to estimate the percentage between number of hit page and the number of total requests on different size of cache memory (2MB, 5MB, 8MB and 10MB). Table 2 shows the comparison between WRP and WRPIR hit ratio with different sizes of cache memory.

Table 2. The comparison between WRP and WRPIR

Cache Size	WRPIR hit ratio	WRP hit ratio	Improvement (%)
2MB	42.6%	42.6%	0%
5MB	54%	45.6%	8.4%
8MB	88.6%	79.3%	9.3%
10MB	79%	82.3%	-3.3%

As seen in the previous table, the average enhancement compared with the (WRP) is 3.6%. This leads to improve the process of prefetching data from cache memory.

6. Conclusion

Although many web caching policies have been proposed in the literature, they still have lots of overheads and are difficult to implement. In this paper, a new replacement policy is developed in order to overcome some of the problems found in the literature. The proposed strategy was able to evict the large size objects from the cache memory with low overhead on the network. This was seen in the simulation results through calculating the hit ratio. The simulation results showed that proposed algorithms which were called LFIR and LFIRS occupied the cache memory with most benefit pages, and with most benefit pages plus smaller pages size respectively. Other factors that would help to make the proposed schemes more efficient will be taken into consideration in the future work. Among these factors are the size of objects, and the time of the previous accesses.

References

[1] Nimrod Megiddo and Dharmendra S. Modha, “Outperforming LRU with an Adaptive Replacement Cache Algorithm,” IEEE Computer Society, 2004.
[2] KavehSamiee,“A Replacement Algorithm Based on Weighting and Ranking Cache Objects,” International Journal of Hybrid Information Technology Vol.2, No.2, April, 2009.
[3] G. Barish and K. Obraczka,“World wide web caching: Trends and Techniques,” May 2000.
[4] GeetikaTewari and Kim Hazelwood, “Adaptive Web Proxy Caching Algorithms,” Harvard University,2004.
[5] Dr T R Gopalakrishnan Nair1, P Jayarekha2, “A Rank Based Replacement Policy for Multimedia Server Cache Using Zipf-Like Law”journal of computing 2010.
[6] Waleed Ali andSitiMariyamShamsuddin, “Integration of Least Recently Used Algorithm and Neuro-FuzzySystem into Client-side Web Caching,” International Journal of Computer Science and Security (IJCSS),2009.
[7] Teng, Wei-Guang, Cheng-Yue Chang, and Ming-Syan Chen. "Integrating web caching and web prefetching in client-side proxies." Parallel and Distributed Systems, IEEE Transactions on 16.5 (2005): 444-455.
[8] Kim, Kyungbaek, and Daeyeon Park. "Reducing outgoing traffic of proxy cache by using client-cluster." Communications and Networks, Journal of 8.3 (2006): 330-338.
[9] K. Ramu and R.Sugumar, “Design and Implementation of Server Side Web Proxy Caching Algorithm,” International Journal of Advanced Research in Computer and Communication Engineering, March 2012.
[10] Brian D. Davison, “A Web Caching Primer,” IEEE InternetComputing, 2001..
[11] Brian D. Davison, “A Web Caching Primer,” IEEE InternetComputing, 2001..
[12] Aguilar, Jose, and Ernst Leiss. "A Web proxy cache coherency and replacement approach." Web Intelligence: Research and Development. Springer Berlin Heidelberg, 2001. 75-84.
[13] González-Cañete, F. J., J. Sanz-Bustamante, E. Casilari, and A. Triviño-Cabrera. "EVALUATION OF RANDOMIZED REPLACEMENT POLICIES FOR WEB CACHES." (2007).
[14] Romano, Sam, and Hala ElAarag. "A quantitative study of recency and frequency based web cache replacement strategies." In Proceedings of the 11th communications and networking simulation symposium, pp. 70-78. ACM, 2008.
[15] Prof. Ketan Shah, Anirban Mitra, Dhruv Matani, algorithm for implementing the LFU cache eviction scheme, August 16, 2010.
[16] AKY Wong, “A Novel Dynamic Cache Size Adjustment Approach for Better Data Retrieval Performance over The Internet,”ScienceDirect,2003.
[17] C. Umapathi and J. Raja, “A Prefetching Algorithm for Improving Web Cache Performance, Journal of Applied Sciences, 2006.
[18] Mohamed Zahran, “Cache Replacement Policy Revisited,” The Annual Workshop on Duplicating, Deconstructing, and Debunking (WDDD) Held in Conjunction with the International Symposium on Computer Architecture (ISCA), 2007.
[19] A.K.Y. Wong. “Web Cache Replacement Policies: A Pragmatic Approach,” IEEE Network
[20] Magazine, 2006.
[21] I. Vakali,“LRU-based algorithms for Web Cache Replacement”In proceeding of: Electronic Commerce and Web Technologies, First International Conference, EC-Web 2000, London, UK, September 4-6,
[22] Hala-elarag,”web proxy cache replacement strategies: simulation, implementation and Performance Evaluation”SpringerBriefs in Computer Science

[23] Ali, Waleed, Siti Mariyam Shamsuddin, and Abdul Samad Ismail. "A survey of Web caching and prefetching." Int. J. Advance. Soft Comput. Appl 3, no. 1 (2011): 18-44
[24] Chung-yi Chang,Tony McGregor,Geoffrey Holmes,“The LRU* WWW proxy cache document replacement algorithm”Sep 17, 2010.
[25] Cherkasova, Ludmila, and Gianfranco Ciardo. "Role of aging, frequency, and size in web cache replacement policies." In High-Performance Computing and Networking, pp. 114-123. Springer Berlin Heidelberg, 2001
[26] Chung-yi Chang,Tony McGregor,Geoffrey Holmes,“The LRU* WWW proxy cache document replacement algorithm”Sep 17, 2010
[27] S. Podlipnig and L. Boszormenyi, “A Survey of Web Cache Replacement Strategies,” ACM Comp. Surveys, vol. 35, no. 4, Dec. 2003.
[28] Wessels, Duane. Web caching. O'Reilly Media, Inc., 2001.
[29] V. S. Mookerjee, and Y. Tan. “Analysis of a Least Recently Used Cache Management Policy For Web Browsers,” Operations Research, Linthicum, Mar/Apr 2002.
[30] Y. Tan, Y. Ji, and V.S Mookerjee. “Analyzing Document-Duplication Effects on Policies for Browser and Proxy Caching”. INFORMS Journal on Computing. 18(4), 506-522. 2006.
[31] T.Koskela, J.Heikkonen, and K.Kaski. ”Web Cache Optimization with Nonlinear Model Using Object Feature,” Computer Networks Journal, Elsevier, 20 Dec. 2003..
[32] H.T. Chen, “Pre-fetching and Re-fetching in Web Caching systems: Algorithms and Simulation,” Master Thesis, Trent University, Peterborough, Ontario, Canada, 2008
[33] Siegel, Jon. CORBA 3 fundamentals and programming. Vol. 2. Chichester: John Wiley & Sons, 2000.

Amany Sarhan Computers and Control engineering department, Tanta University, Egypt.

Ahmed M. Elmogy Computers and Control engineering department, Tanta University, Egypt.

Sally Mohamed Ali, BSc. Of Computers and Control Engineering, Tanta University, Egypt.