Publication of Little Lion Scientific R&D, Islamabad PAKISTAN
Journal of Theoretical and Applied Information Technology
15th April 2011. Vol. 26 No.1
© 2005 - 2011 JATIT & LLS. All rights reserved·
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

PERFORMANCE EVALUATION OF PAGE REMOVAL POLICIES

DR. K A PARTHASARATHY

PROFESSOR & HOD - DEPT OF INFORMATION TECHNOLOGY
APOLLO ENGINEERING COLLEGE, CHENNAI

ABSTRACT

Web caching is an important technique to scale the Internet. One important performance factor of Web caches is the page replacement policies. Due to specific characteristics of the World Wide Web, there exist a huge number of proposals for cache replacement. Furthermore, the article discusses the importance of cache replacement strategies in modern proxy caches and outlines potential future research topics. In this paper we analyze effectiveness of LFU-K replacement policy for the purposes of caching on proxy servers and give the results of traces analysis taken from real proxy servers to reveal a set of properties of network traffic. On the basis of the analysis we have drawn a conclusion about expediency of usage of LFU-K policy which uses information about dynamic change of document popularity, for Web caching. The scheme of LFU-K policy is given as well as results of experiments aimed to compare its effectiveness with the most popular replacement algorithms.

Keywords : *Web Caching, Proxy, Client, Server, Replacement Policy, LRU, LFU and GDS*

1 INTRODUCTION

The recent increase in popularity of the World Wide Web has led to a considerable increase in the amount of traffic over the Internet. As a result, the Web has now become one of the primary bottlenecks to network performance. When objects are requested by a user who is connected to a server on a slow network link, there is generally considerable latency noticeable at the client end. Further, transferring the object over the network leads to an increase in the level of traffic. This has the effect of reducing the bandwidth available for competing requests, and thus increasing latencies for other users. In order to reduce access latencies, it is desirable to store copies of popular objects closer to the user.

Consequently, Web caching has become an increasingly important topic [1]. Caching can be implemented at various points in the network. On one end of the spectrum, there is typically a cache in the Web *server* itself. Further, it is increasingly common for a university or corporation to implement specialized servers in the network called caching *proxies*. Such proxies act as agents on behalf of the client in order to locate a cached copy of a object if possible. More information on caching proxies may be found in [17]. Usually caching proxies and Web servers behave as secondary or higher level caches, because they are concerned only with misses left over from *client* caches. Such client caches are built into the Web browsers themselves. They may store only those accesses from the current invocation.

In this paper, we shall discuss general main memory cache replacement policies designed specifically for use by Web caches. The results are applicable to Web server, proxy and client caches. One of the key complications in implementing cache replacement policies for Web objects is that the objects to be cached are not necessarily of homogeneous size. For example, if two objects are accessed with equal frequency, the hit ratio is maximized when the replacement policy is biased towards the smaller object. This is because it is possible to store a larger number of objects of smaller size. In the standard *least recently used* (LRU) caching algorithm for equal sized objects we maintain a list of the objects in the cache which is ordered based on the time of last access. In particular, the

Publication of Little Lion Scientific R&D, Islamabad PAKISTAN
Journal of Theoretical and Applied Information Technology
15th April 2011. Vol. 26 No.1
© 2005 - 2011 JATIT & LLS. All rights reserved
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

most recently accessed object is at the top of the list, while the least recently accessed object is at the bottom. When a new object comes in and the cache is full, one object in the cache must be pruned in order to make room for the newly accessed object. The object chosen is the one which was least recently used. Clearly the LRU policy needs to be extended to handle objects of varying sizes.

A key component of a cache is its replacement policy, which is a decision rule for evicting a page currently in the cache to make room for a new page. The rule that replaces the least recently used (LRU) page from the cache, is the most popular replacement policy. This is due to a number of reasons: LRU is an optimal online algorithm in the competitve ratio sense 1, it only requires a linked list to be efficiently implemented as opposed to more complicated data structures required for other schemes, and takes advantage of temporal locality in the request sequence. Suppose that we associate with any replacement scheme a *utility function*, which sorts pages according to their suitability for eviction. For example, the utility function for LRU assigns to each page a value which is the time since the page’s last use. The replacement scheme would then replace that page which is most suitable for eviction.

2. NEED FOR WEB CACHING

Web caching is the temporary storage of Web objects (such as HTML documents) for later retrieval. There are three significant advantages to Web caching: reduced bandwidth consumption (fewer requests and responses that need to go over the network), reduced server load (fewer requests for a server to handle), and reduced latency (since responses for cached requests are available immediately, and closer to the client being served). Together, they make the Web less expensive and better performing.
Caching can be performed by the client application, and is built in to most Web browsers. There are a number of products that extend or replace the built-in caches with systems that contain larger storage, more features, or better performance. In any case, these systems cache net objects from many servers but all for a single user.
Caching can also be utilized in the middle, between the client and the server as part of a proxy. Proxy caches are often located near network gateways to reduce the bandwidth required over expensive dedicated Internet connections. These systems serve many users (clients) with cached objects from many servers. In fact, much of the usefulness is in caching objects requested by one client for later retrieval by another client. For even greater performance, many proxy caches are part of cache hierarchies, in which a cache can inquire of neighboring caches for a requested document to reduce the need to fetch the object directly.

Finally, caches can be placed directly in front of a particular server, to reduce the number of requests that the server must handle. Most proxy caches can be used in this fashion, but this form has a different name (reverse cache, inverse cache, or sometimes httpd accelerator) to reflect the fact that it caches objects for many clients but from only one server.

Several metrics are commonly used when evaluating replacement policies. These include the following:

a) Hit rate The hit rate is generally a percentage ratio of documents obtained through using the caching mechanism versus the total documents requested. In addition, if measurement focuses on byte transfer efficiency, weighted hit rate is a better performance measurement.
b) Bandwidth Utilization An efficiency metric. A reduction in the amount of bandwidth consumed shows the cache is better.
c) Response time/access time The response time is the time it takes for a user to get a document.

3 . CLASSIFICATION OF REPLACEMENT STRATEGIES

To present the different proposals in a structured way, we want to give a Classification of replacement strategies. Such classifications were also used by other authors. Before we describe the used classification, we summarize the important factors (characteristics) of Web objects that can influence the replacement process (most of these factors are described in Krishnamurthy and Rexford [2001]):

—recency: time of (since) the last reference to the object;
—frequency: number of requests to an object;
—size: size of the Web object;

—cost of fetching the object: cost to fetch an object from its origin server;
—modification time: time of (since) last modification;
—(heuristic) expiration time: time when an object gets stale and can be replaced immediately.

These factors can be incorporated into the replacement decision. Most of the proposals in the literature use the first four factors. A first classification of replacement strategies was given in Aggarwal et al. [1999]. They proposed three categories:

—*Direct extensions of traditional strategies.* This category subsumes traditional strategies known from other areas (data base buffer management, paging) and extensions thereof.
—*Key-based replacement strategies.* Replacement strategies in this category sort objects upon a primary key (factor). Ties are broken based on secondary key, tertiary key, etc.
—*Function-based replacement strategies.* The idea in function-based replacement is to use a potentially general function derived from the different factors described above. A similar classification was given in Krishnamurthy and Rexford [2001]: one level strategies that use one factor, two level strategies that use a primary and secondary factor, and combination strategies that use a weighted approach for the combination of factors.

There are two major problems with these proposals. First, the first two classes could be merged as every traditional algorithm can be regarded as a key-based strategy using one key (factor). A number of strategies apply various additional techniques (more lists, etc.), that is, they are not only key-based. Second, randomized strategies cannot be classified according to the above described classification. The pure random strategy cannot be classified into any of these categories. It uses no key and no function. Some sophisticated random strategies can be combined with key-based decisions or function-based decisions. Therefore, they can be classified into more than one category.

A number of document replacement polices have been proposed in the literature or deployed in implemented proxy caches.These include: Least Recently Used (LRU), First In First Out (FIFO), Least Frequently Used (LFU), Least Frequently Used with Document Aging (LFU-Aging), the largest of log (base 2) of the document size rounded down to the nearest whole number with LRU. Figure 1 represents HR (HitRate) and BHR (ByteHitRate) and corresponding policy removal algorithms.

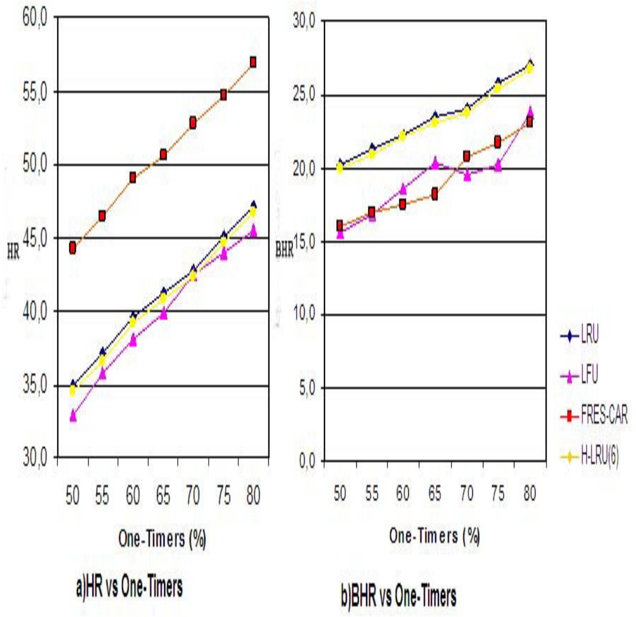

Figure.1

4. PAGE REMOVAL POLICIES

The document replacement algorithms used in this study are FIFO, LRU, LRU-MIN, LFU, LFU-Aging, Size, Log2(Size) and LRU*, random and infinite. The details of implementing these algorithms in the simulator are described below. Each algorithm is based on sorting the documents stored in the cache into an abstract list of documents.

FIFO: The FIFO algorithm maintains a sorted list of cached documents based on document entry times. When there is insufficient space in the cache, the document(s) located at the tail of the sorted list is replaced. More than one document will be removed if the new document is larger than the document at the tail of the list.

LRU: The LRU algorithm sorts cached documents by the latest access time. When a cache hit occurs the access time of the requested document is updated and it is moved to the head of the list. The least recently used document (located at the tail of the list) is the next to be replaced.

LRU-MIN: The LRU-MIN algorithm is similar to LRU. Like LRU, LRU-MIN maintains a sorted list of cached documents based on the time the document was last used. The difference between LRU and LRUMIN 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. If all cached documents are smaller than the new document, the search is repeated looking for the first two documents greater than half the size of the new document. This process (of halving the size and doubling the number of documents to be removed) is repeated if large enough documents can still not be found for replacement.

LFU: This algorithm maintains a reference count for each cached document. All cached documents are sorted by reference count. Documents with the same reference count are sorted by recency. When a cache hit occurs, the reference count of the hit document is incremented by one and the documents are sorted using theupdated reference count. When document replacement is needed, the document located at the tail of thesorted list (i.e. the document that has the smallest reference count and least recency) is removed.

LFU-Aging: The LFU-Aging algorithm sorts cached documents in the same way as LFU. The difference is that LFU-Aging additionally monitors the average of the reference counts of all cached documents. When the average is given over a given maximum number, the cache starts again. In our simulations, the maximum is set to 10 as suggested in [4].

Size: The SIZE algorithm 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 of the document with the largest size is replaced. More than one document will be replaced if necessary.

LRU*: Our algorithm, LRU*, is a mixture of LRU and LFU. In LRU*, cached documents are maintained in a sorted list based on document recency. When a cached document is hit, it is moved to the start of the list and its hit count is incremented by one.

When there is insufficient space for the most recently requested document, the hit count of the least recently used document is checked. If the value of hit count is zero, the document is discarded. Otherwise, the hit count is decreased by one, and the checked document is moved to the start of the sorted list. In other words, in LRU*, the least recently used document is not removed from the cache unless its hit count is zero. The cache will keep checking the hit count of the document(s) at the tail of the sorted list and removing the one(s) with hit count equal to zero until there is sufficient space for the most recently requested document.
In order to prevent a document accumulating too large a hit count, the maximum hit count of cached documents is limited. In this simulation, the maximum hit count is 5 because with a hit count of 5, a cached document can be placed at the start of the sorted list 5 times and this is considered large enough for a cached document. LRU* is intended to keep the hit document(s) in the cache longer (like LFU) and also to age cached documents more dynamically than LFU-Aging.

Infinite: Typical WWW cache hit rates are quite low, less than 50% in most cases, which is much lower than hit rates for most other type of caches found in computer systems. However even this low hit rate results in significant savings because there are large costs involved. We establish an upper bound on the performance of a proxy cache by simulating an infinite sized cache.

Random: Similarly, a working lower bound is determined by simulating a cache with a random document replacement algorithm. The boundaries give performance envelope within which the performance of a document replacement algorithm can be judged.

5. PROXY TRACES ANALYSIS

We have thoroughly analysed traces and found out a set of regularities. Most of the requested objects are of a small size up to 5 kilobytes (see Figure 1). This property of traces serves a basis of most of the up-to-date re-placement policies for the proxy server (GD-Size [3], GDSF [4], LRV, SIZE and others). In these policies the most significant parameter to calculate the rating of objects is the document size that is the smaller is the size of a document the longer it is stored in the cache and vice versa, the bigger is its size, the sooner it is removed. But such an approach has its disadvantages. Firstly, the Figure 2 shows that percentage of requests for the most popular documents depending on their size practically does not change in comparison with all the documents. That is a conclusion can be drawn that documents of different sizes can be popular with the same probability. This conclusion corresponds with research showing the lack of correlation between the size of a document and its popularity [2].

Secondly, another disadvantage is that traffic generated by small-size documents (up to 5 Kb) is small in comparison with one generated by big-size documents (more than 5 Kb). Such a correlation concerns both the whole trace and the most popular objects. Thus, small-size objects caching leads to high effectiveness in Hit Rate metric and not high one in Byte Hit Rate. But as it has been pointed out high indices in Byte Hit Rate exactly imply policy effectiveness from the point of view of decrease in network traffic volume. Figure 2 represents Hit Ratio and cache capacity required. From this figure 2 LRUMIN algorithm performs better than LRU and Size based policy removal algorithms**6**

6. CONCLUSIONS

This article has given an exhaustive survey of cache replacement strategies proposed for Web caches. We concentrated on proposals for proxy caches that manage the cache replacement process at one specific proxy. A simple classification scheme for these replacement strategies was given and used for the description and general critique of the described replacement strategies. Although cache replacement is considered as a solved problem, we showed that there are still numerous areas for interesting research. In this paper we have researched into effectiveness of LFU-*K* policy for the purposes of caching on a proxy server. The analysis of the traces demonstrates disadvantages of basic replacement policies in real systems. In comparison with popular replacement policies LFU-1 algorithm shows higher efficiency. The results of the experiments allow to make a conclusion about expediency of LFU-*K* algorithm on proxy servers in the Web. Evaluation of LFU-2 algorithm effectiveness represents a special interest in future. We plan to try to raise effectiveness of caching mechanism by means of combined replacement policies.

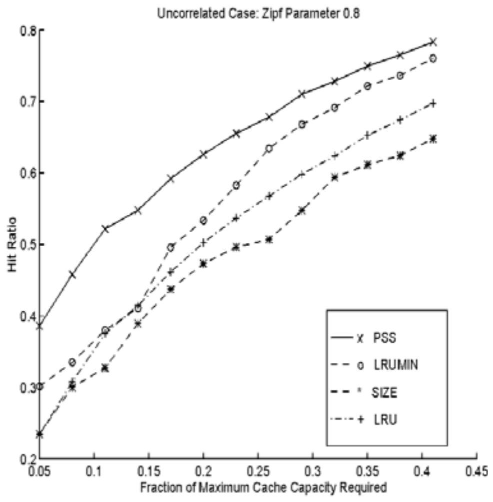

Figure 2

REFRENCES:

[1] Abrams, M., Standridge, C. R., Abdulla, G., Williams, S., And Fox, E. 1995. Caching proxies: Limitations and potentials. In Proceedings of the 4th International World Wide Web Conference.

[2] Arlitt M.,Cherkasova L., Dilley J., Friedrich R., Jin T. Evaluating Content Management Techniques for Web Proxy Caches // SIGMETRICS '99, Inter-national Conference on Measurement and Model-ing of Computer Systems, May 1-4, 1999, Atlanta, Georgia, USA, Proceedings. Performance Evalua-tion Review 27(1), June 1999. P. 3-11.

[3] A. Silberschatz and P. Galvin, *Operating System Concepts*, Fifth Edition, Addison Wesley Longman, 1997.

[4] A McGregor, The NLANR AMP active measurement program, http://amp.nlanr.net/active

[5] Balafoutis, E., Panagakis, A., Laoutaris, N., And Stavakakis, I. 2002. The impact of

replacement granularity on video caching. In *IFIP Networking 2002*. Lecture Notes in Computer Science, vol. 2345. Springer-Verlag, Berlin, Germany, 214–225.

[6] Breslau L., Cao P., Fan L., Philips G., Shenker S.Web caching and Zipf-like distribution: evidence and implications // IEEE Infocom XX (V). 1999. P 1-9.

[7] Cao P., Irani S. Cost-aware www proxy caching algorithms // In Proceeding of the 1997 USENIX Symposium of Internet Technology and Systems. 1997. P 193-206.

[8]Cherkasova L. Improving WWW Proxies Perform-ance with Greedy-Dual-Size-Frequency
Caching Policy // Technical Report HPL-98-69R1, Hewlett-Packard Laboratories, Nov. 1998.

[9] Chris R (1998) Designing for delay in interactive information retrieval. Interacting with Computers, 10:87-104.

[10] C.R. Cunba, A. Bestavros, M.E. Crovella, “Characteristics of WWW Client-based Traces”, BU-CS-96-010, Boston University.

[11] C. Aggarwal, J. Wolf, P. Yu, “Caching on the World Wide Web”, IEEE Transactions on Knowledge and Data Engineering, 11(1), 1999, pp. 94-107.

[12] Davison, B. D. 2001. A Web caching primer. IEEE Internet Comput. 5, 4 (July), 38–45.

[13] Fonseca, R., Almeida, V., Crovella, M., And Abrahao, B. 2003. On the intrinsic locality properties of Web reference streams. In Proceedings of the IEEE INFOCOM. IEEE Computer Society, Piscataway, NJ.

[14] James TB (1986) A theory of productivity in the creative process. IEEE Computer Graphics and Applications, 6(5):25-34, May 1986.

[15] M. Abrams, C.R. Standbridge, G. Abdulla, S. Williams and E.A. Fox, “Caching Proxies: Limitations and Potentials”, *WWW-4*, Boston, December, 1995.

[16] M. Busari, C. Williamson, “ProWGen: A Synthetic Workload Generation Tool for the Simulation Evaluation of Web Proxy Caches”, Computer Networks, 38(6), Jun. 2002, pp. 779-794.

[17] P. Cao and S. Irani, “Cost-Aware WWW Proxy Caching Algorithms”, Proceedings of the 1st USENIX Symposium on Internet Technologies and Systems, Monterey, California, USA, Dec. 1997, pp. 193–206.

[18] R. Durrett, Probability: Theory and Examples, Duxbury Press, 2nd edition, 1996.

[19] S. Jin and A. Bestavros, “GreedyDual* Web Caching Algorithm: Exploiting the Two Sources of Temporal Locality in Web Request Streams”, In Proceedings of the 5th International Web Caching and Content Delivery Workshop, Lisbon, Portugal, May 2000.

[20] Walter JD, Ahrvind JT (1982) The economic value of rapid response time.Technical Report GE20-0752-0, IBM, White Plains, NY, November 1982.