LRU-based algorithms for Web Cache Replacement

A. I. Vakali

Department of Informatics
Aristotle University of Thessaloniki, Greece
E-mail: avakali@csd.auth.gr

Abstract. Caching has been introduced and applied in prototype and commercial Web-based information systems in order to reduce the overall bandwidth and increase system's fault tolerance. This paper presents a track of Web cache replacement algorithms based on the Least Recently Used (LRU) idea. We propose an extension to the conventional LRU algorithm by considering the number of references to Web objects as a critical parameter for the cache content replacement. The proposed algorithms are validated and experimented under Web cache traces provided by a major *Squid* proxy cache server installation environment. Cache and bytes hit rates are reported showing that the proposed cache replacement algorithms improve cache content.

Key-Words : Web-based information systems, Web caching and proxies, Cache replacement algorithms, Cache consistency.

1 Introduction

The continously rapid growth and worldwide expansion of the Internet has introduced new issues such as World-Wide Web (WWW) traffic, bandwidth insufficiency and distributed objects exchange. Web caching has presented an effective solution, since it provides mechanisms to faster web access, to improved load balancing and to reduced server load. Cache efficiency depends on its content update frequency as well as on the algorithmic approach used to retain the cache content reliable and consistent. Several approaches have been suggested for more effective cache management and the problem of maintaining an updated cache has gained a lot of attention recently, due to the fact that many web caches often fail to maintain a consistent cache. Several techniques and frameworks have been proposed towards a more reliable and consistent cache infrastructure [5, 7]. In [2] the importance of various workload characteristics for the Web proxy caches replacement is analyzed and trace-driven simulation is used to evaluate the replacement effectiveness. The performance and the homogeneity of Web caching is studied in [1] where a new generalized LRU is presented as an extension to the typical SLRU algorithm. Hit ratios and robustness of the proposed replacement algorithm is compared with other Web replacement policies using both event and trace-driven simulations.

Performance improvements due to Web caching have been investigated in order to estimate the value and importance of Web caching. Research efforts have focused in maintaining Web objects coherency by proposing effective cache replacement policies. A number of Web replacement policies are discussed in [3], and compared on the basis of trace-driven simulations. A web-based evolutionary model has been presented in [12] where cache content is updated by evolving over a number of successive cache objects populations and it is shown by trace-driven simulation that cache content is improved. A Genetic algorithm model is presented in [13] for Web objects replication and caching. Cache replacement is performed by the evolution of Web objects cache population accompanied by replication policies employed to the most recently accessed objects.

This paper presents a new approach to Web Cache replacement by proposing a set of algorithms for cache replacement. The proposed set of algorithms is based on the popular *L*east *R*ecently *U*sed (*LRU*) algorithm which replaces from the Web cache the objects that have not been requested for the longest time. Thus LRU uses only the time of the last request to a Web object and this time is the critical factor for an objects purge from the Web cache. Here, we introduce a variation of LRU which considers a "history" of Web objects requests. A similar approach was presented in [10] for the page replacement process in database disk buffering. The main idea is to keep a record for a number of past references to Web objects, i.e. a history of the times of the last h requests is evaluated for each cached Web object. This approach defines a whole set of cache replacement algorithms called History LRU (notation *HLRU(h)*). A number of HLRU algorithms is experimeneted under Squid proxy cache traces and cache log files.

The remainder of the paper is organized as follows. The next section introduces and formulates the cache replacement problem. Section 3 presents the typical LRU and the proposed HLRU algorithms whereas Web proxies ans their performance issues are presented in Section 4. Section 5 discusses implementation and validation details and results from trace driven experimentation are presented. Section 6 points some conclusions and discusses potential future work.

2 The Cache Replacement Problem

Web proxies define a limited cache area, for storage of a number of Web objects. Once the cache area is almost filled there has to be a decision to replace some of the cached objects with newer ones. Therefore, a cached Web object "freshness" has to be determined by specific rules and in Web caching terminology an object is considered *stale* when the original server must be contacted to validate the existence of the cache copy. Object's staleness results from the cache server's lack of awareness about the original object's changes. Each proxy cache server must be reinforced with specific staleness confrontation. The most important parameters corresponding to attributes of each cached object are summarized in Table 1.

parameter	description
s_i	server on which object resides.
b_i	object's size, in KBytes.
t_i	time the object was logged.
c_i	time the object was cached.
l_i	time of object's last modification.
f_i	number of accesses since last time object i was accessed.
key_i	objects original copy identification (e.g. its URL address).

Table 1. The most useful attributes of each cached object i.

Definition 1 : The cached object's *staleness ratio* is defined by,

$$StalRatio_i = \frac{c_i - l_i}{now - c_i}$$

where the numerator corresponds to the time interval between the time of object being cached and the time of the object's last modification and the denominator is the cache "age" of the object i.e. it determines the time that the object has remained in cache. It is always true that $StalRatio_i \geq 0$ since $c_i - l_i \geq 0$ and $now - c_i > 0$ (now is the current time). The lower the value of $StalRatio_i$ the more stale the object i is, since that indicates that it has remained in cache for longer period.

Definition 2 : The cached object's *dynamic frequency* is defined by

$$DynFreq_i \ = \ \frac{1}{f_i}$$

since f_i is the metric for estimating an object's access frequency (Table 1). It is true that the higher the values of $DynFreq_i$, the most recently was accessed.

Definition 3 : The cached object's *retrieval rate* is defined by

$$RetRate_i \ = \ \frac{lat_s}{band_s}$$

where lat_s is the latency to open a connection to the server s and $band_s$ is the bandwidth of the connection to server s. $RetRate_i$ represents the cost involved when retrieving an object from its original storage server.

Definition 4 : The cached object's *action function* is defined by

$$act_i = \begin{cases} 0 \text{ if object } \; i \; \text{ will be purged from cache} \\ 1 \text{ otherwise} \end{cases}$$

Problem Statement : If N is the number of objects in cache and C is the total capacity of the cache area, then the cache replacement problem is to :

$$\text{MAXIMIZE} \quad \sum_{i=1}^{N} act_i \times StalRatio_i \times \frac{DynFreq_i}{RetRate_i}$$

$$\text{subject to} \qquad \sum_{i=1}^{N} act_i \times b_i \leq C$$

where the fraction $\frac{DynFreq_i}{RetRate_i}$ is used as a weight factor associated with each cached object, since it relates the objects access frequency with its retrieval rate.

3 LRU and HLRU Cache Replacement

LRU cache replacement is based on the Temporal Locality Rule which states that "The Web objects which were not referenced in the recent past, are not expected to be referenced again in the near future". LRU is widely used in database and Web-based applications. For example, in Squid, the LRU is used along with certain parameters such as a *low watermark* and a *high watermark* to control the usage of the cache. Once the cache disk usage is closer to the low watermark (usually considered to be 90%) fewer cached Web objects are purged from cache, whereas when disk usage is closer to the high watermark (usually considered to be 95%) the cache replacement is more severe i.e. more cached Web objects are purged from cache. There are several factors as of which objects should be purged from cache.

Definition 5 [LRU Threshold] : A value identified as *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 and on the low and high watermarks. When current cache size is closer to low watermark the threshold gets a higher value, otherwise when current cache size is closer to high watermark the threshold value is smaller.

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. Here we introduce a scheme to support a "history" of the number of references to a specific Web object.

```
for (i=left boundary; i<=right boundary; i++)
{       if (hashTable[i]. OldtimeOfAccess = = 0)
        {       ++counterOfObjectswithOneAccess;
                if (counterOfObjectswithOneAccess > 1)
                        break;
        }
}
if (there is >1  object with only 1 access)
        for (i=left boundary; i<=right boundary; i++)
                if (the object i had more than one accesses)
                        age[i] = CurrentTime – hashTable[i]. OldtimeOfAccess;
                else
                        age[i] = CurrentTime – hashTable[i]. NewtimeOfAccess;
else
        for (i=left boundary; i<=right boundary; i++)
                if (the object i had more than one accesses)
                        age[i] = CurrentTime – hashTable[i]. OldtimeOfAccess;
                else
                        hashTable[i].empty = true;

qsort(hashTable);

for (i=left boundary; i<=right boundary; i++)
{       if (not an emergency purge)
        {       if (position is full)
                        if (the age of object i > LRU threshold)
                        {
                                object is purged;
                                time the object stayed in cache =
                                        CurrentTime - hashTable.timeOfFirstAccess;
                                current cache swap - = object's size;
                        }
        }
        else    // the case of an emergency purge
        {       if ( i <= left boundary of the bucket + 8 )
                        if ( position i is not empty )
                        {
                                purge the object;
                                time the object stayed in cache =
                                CurrentTime – hashTable.timeOfFirstAccess;
                                current cache swap - = object's size;
                        }
        }
}
```

Fig. 1. The History LRU cache replacement algorithm.

Definition 6 [History Function] : Suppose that $r_1, r_2, \ldots, r_n$ are the requests for cached Web objects at the times $t_1, t_2, \ldots, t_n$, respectively. A history function is defined as follows :

$$hist(x,h) = \begin{cases} t_i & \text{if there are exactly } \; h-1 \text{ references} \\ & \text{between times } t_i \text{ and } \; t_n \\ 0 & \text{otherwise} \end{cases}$$

The above function $hist(x, h)$ defines the time of the past h-th reference to a specific cached object x.

Therefore, the proposed HLRU algorithm will replace the cached objects with the maximum $hist$ value. In case there are many cached objects with $hist = 0$, the typical LRU is considered to decide on which object will be purged from cache. The same idea of the threshold value (to decide when the cache replacement will occur) still holds. In Table 2 the main structure of the cache hash

LRU	HLRU
struct $HashTable$	struct $HashTable$
{ long LRU_age	{ int $OldTimeOfAccess$
long $positionInFile$	long $positionInFile$
boolean $empty$	boolean $empty$
long $timeOfFirstAccess$	long $timeOfFirstAccess$
} $hashTable[50000]$	} $hashTable[50000]$

Table 2. The main LRU and HLRU data structure

table is presented. For LRU each cached object is assigned an LRU_age to indicate the time since its last reference. Variable $positionInFile$ declares the position the specific object has in the file, whereas the boolean type variable *empty* indicate whether the specific cache location is empty or not. Finally, variable $timeOfFirstAccess$ is used for the time the specific object was cached. The HLRU data structure is quite similar, there are two different times kept for each cached object. $OldTimeOfAccess$ is the time the cached object was first referenced whereas $NewTimeOfAccess$ is the time of the last reference to the cached object. Similarly, Figure 1 presents the implemented HLRU algorithm in a pseudocode format, for the case of two ($h = 2$) past references. Under HLRU a linked list has been used for each cached object in order to keep track of the times of past references.

4 Web Proxies - Performance Issues

The performance metrics used in the presented approach focus on the cached objects cache-hit ratio and byte-hit ratio :

- **Cache hit ratio :** represents the percentage of all requests being serviced by a cache copy of the requested object, instead of contacting the original object's server.
- **Byte hit ratio :** represents the percentage of all data transferred from cache, i.e. corresponds to ratio of the size of objects retrieved from the cache server. Byte hit ratio provides an indication of the network bandwidth.

The above metrics are considered to be the most typical ones in order to capture and analyze the cache replacement policies (e.g. [3, 1, 2]).

Furthermore, the performance of the proposed cache replacement algorithms is studied by estimating the strength of the cache content. This strength is evaluated by the consideration of the cached objects retrieval rates as well as their frequency of access and their "freshness". In order to evaluate the HLU algorithms we have devised a function in order to have a performance metric for assessing the utilization and strength of the cache content. The following formula $F(x)$ considers the main web cache factors as identified in the Cache replacement problem statement in Secion 2. Here, we consider a cache content x of N individual cached objects :

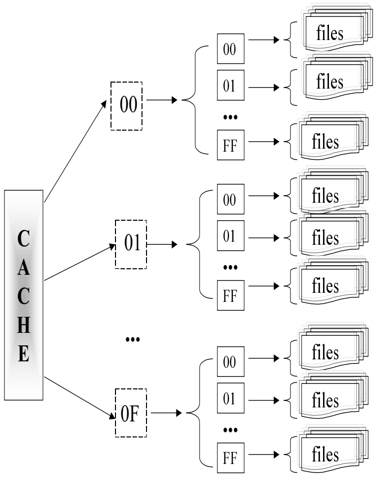


Fig. 2. Structure of the Squid proxy cache area.

$$F(x) \;=\; \sum_{i=1}^{N} act_i \times StalRatio_i \times \frac{DynFreq_i}{RetRate_i} \qquad (1)$$

The above function has been introduced in the present research effort in order to consider the effect of staleness, access frequency and retrieval cost in the overall cache replacement process.

5 Experimentation - Results

Aristotle University has installed Squid proxy cache for main and sibling caches and supports a Squid mirror site. The present paper uses traced information provided by this cache installation for experimentation. A simulation model was developed and tested by Squid cache traces and their corresponding log files. Traces refer to the period from May to August 1999, regarding a total of almost 70,000,000 requests, of more that 900 GB content. A compact log was created for the support of an effective caching simulator, due to extremely large access logs created by the proxy. The reduced simulation log was constructed by the original Squid log fields needed for the overall simulation runs.

A track of the proposed HLRU algorithms has been tested. More specifically the notations $HLRU(2)$, $HLRU(4)$ and $HLRU(6)$ refer to the HLRU implementations for 2, 4 and 6 past histories respectively. Furthermore, the typical LRU cache replacement policy applied in most proxies (e.g. Squid), has been simulated in order to serve as a basis for comparisons and discussion.

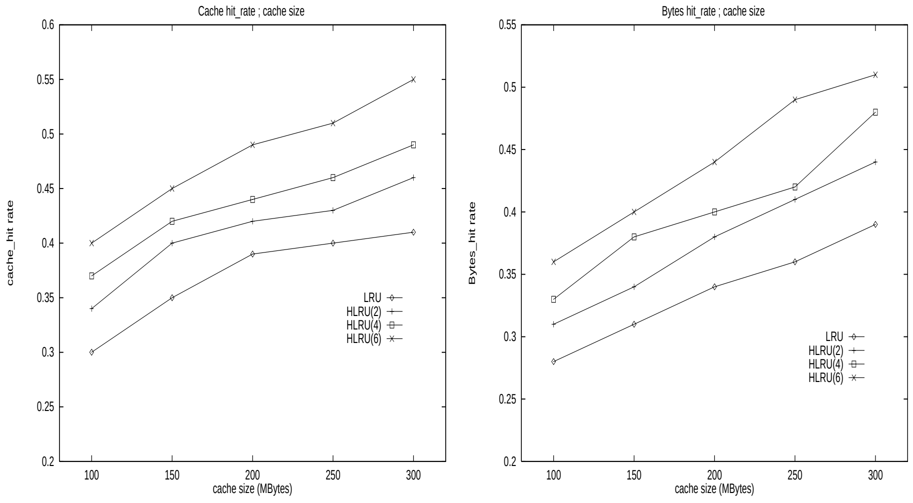
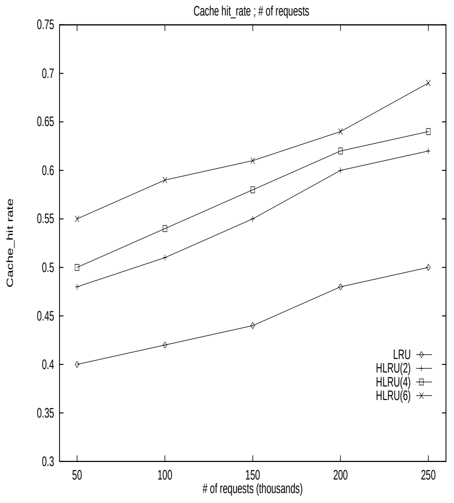


Fig. 3. Cache and Bytes hit; cache size

The performance metrics used in this simulation model focus on the cached objects cache-hit ratio, byte-hit ratio and the strenth function $F(x)$ (equation 1) normalized to the interval [0,1]. Figure 3 depicts the cache hit and the bytes hit ratio for all four algorithms with respect to cache size. More specifically, the left part of Figure 3 presents the cache hit ratio for a cache size of $100, 150, \cdots, 300$ MBytes. The cache hit under $HLRU(6)$ policy outperforms the corresponding metric of all other policies whereas all $HLRU$ algorithms have better cache hits than the corresponding typical LRU approach. This was expected due to the ability of the $HLRU$ approach to keep track of a history of past references. It should be noted that the cache hit ratios seem to get to a peak and remain closer to this peak value as the cache size increases. This is explained due to the fact that the larger the cache size, the less replacement actions need to be taken since there is more space to store the cached objects and the cache server can "afford" to accommodate them there with less replacement actions. Figure 3 (right part) depicts the byte hit ratio for the four different cache replacement policies with respect to the cache size. The byte hit ratios follow a similar skew as the corresponding cache hit ratios, but they never get in as hit high values as cache hit ratios. a smoother curve as the cache size increases. Again $HLRU(6)$ algorithm is the best of all and as the byte hit rates decrease as the number of past references (history set) decreases.

Figure 4 depicts the cache hit and the bytes hit ratio for all four algorithms with respect to the number of requests. More specifically, the left part of Figure 4 presents the cache hit ratio for a cache of $50, 150, \cdots, 250$ thousands of requests. Again, the cache hit under $HLRU(6)$ policy outperforms the corresponding metric of all other policies whereas all $HLRU$ algorithms have better cache hits than the corresponding typical LRU approach. It is important to note



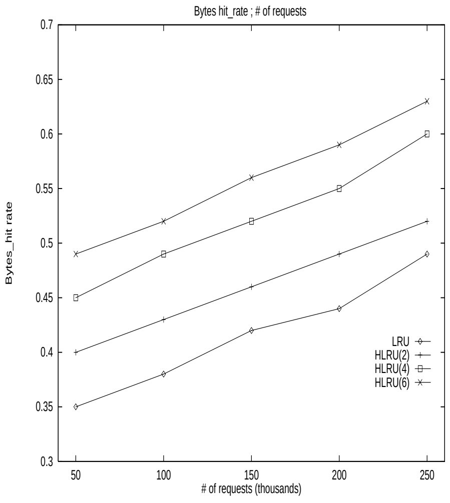


Fig. 4. Cache and Bytes hit; # of requests

that all approaches show better results as the number of requests increases and the cache hit rates do get to considerable hit rates of almost 70%.

6 Conclusions - Future Work

This paper has presented a study of applying a history based approach to the Web-based proxy cache replacement process. A history of past references is associated to each cached object and a track of algorithms has been implemented based on a number of past histories. Trace-driven simulation was used in order to evaluate the performance of the proposed cache replacement techniques and the simulation model was based on the Squid proxy cache server. The experimentation indicated that all of the proposed HLRU approaches outperform the conventional Least-Recently-Used (LRU) policy adopted by most currently available proxies. Results have shown that the HLRU approach significantly improves cache hit and byte hit ratios.

Further research should extend current experimentation and the present scheme in order to integrate most popular cache replacement algorithms. More specifically, different schemes such as SLRU, MFU, RR algorithms could be introduced in the proposed cache replacement in order to study their impact and effectiveness on the Web cache content replacement.

Acknowledgments

The author thanks Panayotis Junakis (System administrator) and Savvas Anastasiades (technical staff) of the Network Operation Center at the Aristotle University, for providing access to the Squid cache traces and trace log files.

References

1. C. Aggarwal, J. Wolf and P.S.Yu: Caching on the World Wide Web,*IEEE Transactions on Knowledge and Data Engineering*, Vol.11,No.1,pp.94-107,Jan-Feb 1999.
2. M. Arlitt, R. Friedrich and T. Jin: Performance Evaluation of Web Proxy Cache Replacement Policies, *Hewlett-Packard Technical Report HPL 98-97, to appear : Performance Evaluation Journal*, May 98.
3. A. Belloum and L.O. Hertzberger : Document Replacement Policies dedicated to Web Caching , *Proceedings ISIC/CIRA/ISAS'98 Conference* , Maryland, USA, Sep. 1998.
4. R. Caceres, F. Douglis, A. Feldmann, C. Glass, M. Rabinovich : Web Proxy Caching : The Devil is in the Details, *Proceedings of the SIGMETRICS Workshop on Internet Server Performance*, Jun 1998.
5. P. Cao, J. Zhang and K. Beach: Active Cache : Caching Dynamic Contents on the Web, *Proceedings of the IFIP International Conference on Distributed Platforms and Open Distributed Processing* , pp. 373-388, Middleware 1998.
6. A. Chankhunthod, P. Danzig and C. Neerdaels: A Hierarchical Internet Object Cache, *Proceedings of the USENIX 1996 Annual Technical Conference*, pp.153-163,San Diego,California,Jan 1996.
7. S. Michel, K. Nguyen, A. Rosenstein and L. Zhang: Adaptive Web Caching : Towards a New Global Caching Architecture, *Proceedings of the 3rd International WWW Caching Workshop*, Manchester, England, Jun 1998.
8. A Distributed Testbed for National Information Provisioning, http://ircache.nlanr.net/, 1998.
9. M. Nottingham: Web Caching Documentation, http://mnot.cbd.net.au/cache_docs/, Nov 1998.
10. E. J. O'Neil, P. E. O'Neil, and G. Weikum : The LRU-K Page Replacement Algorithm For Database Disk Buffering, *Proceedings of the ACM SIGMOD Conference*, pp.297-306, Washington DC, USA, 1993.
11. Squid: Squid Internet Object Cache, mirror site, Aristotle UNiversity, *http://www.auth.gr/Squid/*, 1999.
12. A. Vakali: A Web-based evolutionary model for Internet Data Caching, *Proceedings of the 2nd International Workshop on Network-Based Information Systems, NBIS'99*,IEEE Computer Society Press, Florence,Italy, Aug 1999.
13. A. Vakali: A Genetic Algorithm scheme for Web Replication and Caching, *Proceedings of the 3rd IMACS/IEEE International Conference on Circuits, Systems, Communications and Computers, CSCC'99*, World Scientific and Engineering Society Press, Athens, Greece, Jul. 1999.