A Class-Based Least-Recently Used Caching Algorithm for WWW Proxies

Rachid El Abdouni Khayari, Ramin Sadre, Boudewijn R. Haverkort
Laboratory for Performance Evaluation and Distributed Systems
Department of Computer Science, RWTH Aachen, D–52056 Aachen
http://www-lvs.informatik.rwth-aachen.de/

July 12, 2002

Abstract

In this paper we study and analyze the influence of caching stategies on the performance of WWW proxies. We propose a new strategy called **class-based LRU** that works recency-based as well as size-based, with the ultimate aim to obtain a well-balanced mixture between large and small documents in the cache, and hence, good performance for both small and large object requests. We show that for class-based LRU good results are obtained for both the hit rate and the byte hit rate, if the size of the classes and the corresponding document size ranges are well choosen. The latter is achieved by using a Bayesian decision rule and a characterisation of the requested object-size distribution using the EM-algorithm. Furthermore, the overhead to implement class-based LRU is comparable to that of LRU and does not depend on the number of cached objects.

1 Introduction

Today, the largest share of traffic in the internet originates from WWW requests. The increasing use of WWW-based services has not only led to high frequented web servers but also to heavy-used components of the internet. Fortunately, it is well known that there are popular and frequently requested sites, so that object caching can be employed to reduce the internet network traffic [3] and to decrease the perceived response times.

Web caching has some special properties that make it an interesting and new reseach area, separate from traditional approaches towards caching: (i) The sizes of the objects to be cached vary greatly. Thus, one can not assume fixed-sized "pages" as in main-memory caching. Additionally, objects may be of different types, thus influencing caching decisions. (ii) The costs to request particular objects vary largely and are difficult to compute in advance. These costs are not only different per object, even for the same object they depend on the load of the origin server, its opererational state and the distance between client and server. (iii) Objects in the cache are *read only* and hence no write-back mechanism is needed.

There are three possible locations for object caching (see Fig. 1): (i) *Client caching.* The locality in the requests of many clients can be exploited by client-side caching in order to reduce network traffic and response time. (ii) *Proxy server caching.* The proxy server typically resides between the LAN (at which also the clients are connected) and the internet. The clients must configurate their browsers so that all HTTP requests are directed to the proxy. In doing so, external bandwidth will be saved, however, at the risk of the proxy server itself becoming a bottleneck. (iii) *Primary web server caching.* Primary web servers store objects in main memory, in order to reduce disk I/O. This caching problem did not attract much attention, due to the fact that no (internet) bandwidth can be saved with it and the document retrieval time is often dominated by the network latency [2]. Furthermore, the load on the disks seldomly forms a bottleneck in a WWW server. Table 1 gives an overview over the pros and cons of the three different caching locations.

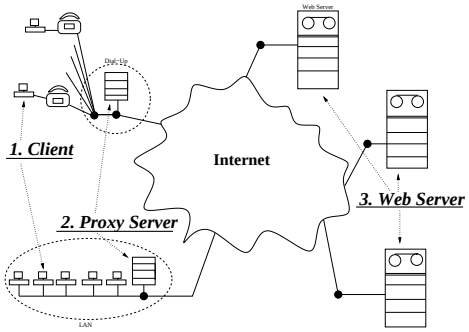

Figure 1: Three cache locations in the WWW

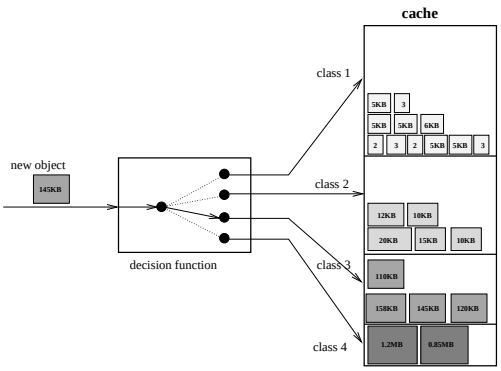

Figure 2: Principle of Class-LRU

at **client** and **proxy**	at **server**
+ reduces network traffic	− does not reduce network traffic
+ reduces response time	− does not (really) reduce response time
+ decreases server workload	− does not decrease server workload
− distorts server access statistics	− higher server load
− danger of data inconsistency	+ decreases response time in LAN
at **proxy**	+ decreases I/O load
− cache miss extends response time	
− possible new bottleneck	

Table 1: Pros and cons of WWW-caching according to its location

Organisation of the paper. We will give a consise overview of existing caching strategies in Section 2, before we present our new strategy in Section 3. The implementation, validation and comparison is presented in Section 4. Conclusions are drawn in Section 5.

2 Caching strategies

Over the last few years, many well-known caching strategies have been evaluated [7, 15, 11, 10]. Aim of these strategies has been to improve the cache hit rate (defined as the percentage of requests that could be served from the cache), the cache byte hit rate (defined as the percentage of bytes that could be served from the cache), or, even better, both. At the center of all the approaches is the question which object has to be replaced when a new object has to be stored (and the cached is already completely filled). The algorithms referred to above have been developed for specific contexts and it has been shown that an algorithm that is optimal for one context may fail to provide good results in another context [7, 10]. This is due to the fact that the caching algorithms rate some object characteristics more important than others. Table 2 summarises whether the caching algorithms use certain object characteristics or not.

Some of the mentioned caching strategies have been evaluated for use in proxy and primary web servers. Williams and Abrams [15] have proposed the LFF strategy and have compared it with LRU and FIFO. The LFF did show better results, but only w.r.t. the hit rate. Arlitt and Williamson [6] have analyzed trace-driven cache simulations with different logfiles. LFU-aging has provided the best results w.r.t. both hit rate and byte hit rate. If one considers only the hit rate, then LFF was better. Furthermore, the methods LRU, LFU and FBR (Frequency Based Replacement) have been compared with LFU-aging and LFF. It has been shown that the extension based on the aging approach only delivers a small improvement but produces more CPU load. In comparison to LRU, LFU, LFF and other strategies, GDS has shown the best results for both the hit rate and the byte hit rate. However, a simultaneous optimization of both metrics could not be attained. An extension of GDS to GDSF has been proposed in [8]; the results and conclusions remained the same, with the exception that GDSF generally yields better results than GDS.

strategy	size-based	recency-based	frequency-based	origin context
FIFO	-	-	-	-
LFF	√	-	-	web caching
LRU	-	√	-	memory caching
LFU	-	-	√	memory caching
SLRU	-	√	√	disk caching
LRU-K	-	√	√	database caching
LFU-Aging	-	√	√	web, disk caching
GDS	√	√	-	web caching
GDSF	√	√	√	web caching

Table 2: Classification of caching algorithms according to the employed object characteristics and their origins

Furthermore, it has been shown that LFF and LFU have severe problems with *cache pollution* [7], that is, with cached objects which are not requested any more but stay in the cache due to their popularity in the past.

These results indicate that size-based strategies yield better results for the hit rate, whereas frequency-based strategies improve the byte hit rate. No strategy has been recognized as the ultimate best one, rather the choice of a good strategy depends on the characteristics of the considered workload. These considerations have led us to develop a workload-based caching strategy for WWW proxies, as will be discussed in the next section.

3 Class-based LRU

3.1 Basic idea

The caching strategy *class-based LRU* is a refinement of standard LRU. Its justification lies in the fact that object-size distributions in the WWW are heavy-tailed, that is, although small objects are more popular and are requested more frequently, large objects occur more often than it has been expected in the past, and therefore have a great impact on the perceived performance.

In most caching methods, the object sizes are completely ignored, or either small or large objects are favoured. However, since caching large objects increases the byte hit rate and decreases the hit rate (and vice versa for small objects), both a high byte hit rate and a high hit rate can only be attained by creating a proper balance between large and small objects in the cache. With C-LRU, this is achieved by partioning the cache into portions reserved for objects of a specific size, as follows:

- The available memory for the cache is divided into I partitions where each partition i (for $i = 1, \ldots, I$) takes a specific fraction p_i of the cache ($0 < p_i < 1, \sum_i p_i = 1$).
- Partition i caches objects belonging to class i, where class i is defined to encompass all objects of size s with $r_{i-1} \leq s < r_i$ ($0 = r_0 < r_1 < \ldots < r_{I-1} < r_I = \infty$).
- Each partition in itself is managed with the LRU strategy.

Thus, when an object has to be cached, its class has to be determined before it is passed to the corresponding partition (see Figure 2). For this strategy to work, we need an approach to determine the values $p_1, \ldots, p_I$ and $r_1, \ldots, r_I$. This will be addressed in the next section.

3.2 Determining the fractions p_i and the boundaries r_i

Object-size characterisation. As has recently been shown, the object-size distribution of objects requested at proxy servers, can very well be described as a hyper-exponential distribution; the parameters of such a hyperexponential distribution can be estimated easily with the EM-algorithm [12]. This implies that the object-sizes density $f(x)$ takes the form of a probabilistic mixture of exponential terms:

$$f(x) = \sum_{i=1}^{I} c_i \lambda_i e^{-\lambda_i x}, \qquad 0 \leq c_i \leq 1, \qquad \sum_{i=1}^{I} c_i = 1, \qquad \text{for } i = 1, \ldots, I. \tag{1}$$

In [12] it is shown that I normally is relatively small, say in the range of 4 to 8.

Cache fractions p_i. For the fraction p_i, we propose two possible values:

(a) to optimize the hit rate, we take the partition size p_i proportional to the probability that a request refers to an object from class i, that is, we set: $p_i = c_i$;

(b) to optimize the byte hit rate, we take into account the expected amount of bytes "encompassed by" class i in relation to the overall expected amount of bytes. Since the average object size in class i is $1/\lambda_i$, we set: $p_i = \frac{c_i/\lambda_i}{\sum_{j=1}^{I} c_j/\lambda_j}$.

Cache boundaries r_i. The range boundaries r_i are computed using a Bayesian decision rule (see [13]). For an object of size s, the appropriate class $C(s)$ is taken such that the decision-error is minimised. To achieve this aim, we set [13]:

$$C(s) = argmax_i(c_i \lambda_i e^{-\lambda_i s}), \qquad i = 1, \ldots, I. \tag{2}$$

Note that the range boundaries need to be determined only once. Upon the arrival of a request for an object with size s, a simple (binary) search in the ranges $\{[0, r_1], [r_1, r_2], \ldots, [r_{I-1}, r_I]\}$ will yield the appropriate class $C(s)$.

4 Application, evaluation and comparison

Cache performance heavily depends on the size of the provided cache and the employed replacement strategy. To evaluate and compare the performance of C-LRU, we performed trace-driven simulations. We used two traces: the **RWTH trace** has been collected in early 2000 and consist of the logged requests to the proxy-server of the RWTH of Aachen, and the **DEC trace** of the web proxy of Digital Equipment Company [1]. Note that simulation time passes much faster than real time; in our studies, a trace comprising 54 days could be simulated in only a few minutes.

4.1 Trace analysis

In our study, we only considered static (cacheable) objects, requests to dynamic objects were removed as far as identified. Table 3 presents some important statistics for both traces. The heavy-tailedness of the object-size distribution is clearly visible: high squared coefficients of variation and very small medians (compared to the means). The maximum reachable hit rate (denoted as HR_∞) and the maximum reachable byte hit rate (BHR_∞) have been computed using a trace-based simulation with infinite cache. Below, we address the object size distribution, the recency and the frequency of object requests. We focus on the RWTH trace (the corresponding, and similar, results for the DEC trace are given in [14].

Distribution of the object sizes. It has been found that the object-size distribution shows the property of heavy-tailedness (see [14]); the distribution decays more slowly than an exponential distribution. This becomes even more clear from the histogram of object sizes in Figure 3. The heavy-tailedness is also present when looking at the request frequency as a function of the object size (see Figure 4). It shows that small objects are not only more numerous but also that they are requested more often than large objects (this inverse correlation between file size and file popularity has also been observed in [4]). Thus, caching strategies which favour small objects are expected to perform better. However, the figure also shows that large objects cannot be neglected.

Recency of reference (temporal locality). Another way to determine the popularity of objects is the temporal locality of their references [5]. However, recent tests have pointed out that temporal locality decreases [9], possibly due to client caching. We performed the common LRU stack-depth [5] method to analyse the temporal locality of references. The results are given in Figure 5 (left). The positions of the requested objects within the LRU stack are combined in 5000 blocks. The figure shows that about 20% of all requests have a strong temporal locality, thus suggesting that a recency-based caching strategy should be used.

	RWTH	DEC
total #requests	32,341,063	3,763,710
total #bytes	353.27 GB	31.93 GB
#*cacheable* request	26,329,276	3,571,761
#*cacheable* bytes	277.25 GB	30.14 GB
fraction #*cacheable* requests	81.4 %	94.9 %
total #*cacheable* bytes	78.5 %	94.4 %
average object size	10,529 Bytes	10,959 Bytes
squared coeff. of variation	373.54	90.92
median	3,761 Bytes	3,696 Bytes
smallest object	118 Bytes	14 Bytes
largest object	228.9 MB	132.7 MB
unique objects	8,398,821	1,379,865
total size of unique objects	157.31 GB	17.08 GB
HR∞	30.46 %	47.34 %
BHR∞	16.01 %	39.32 %
original size of trace file	2 GB	800 MB
size after preprocessing	340 MB	47 MB

Table 3: Statistics for the RWTH and the DEC trace

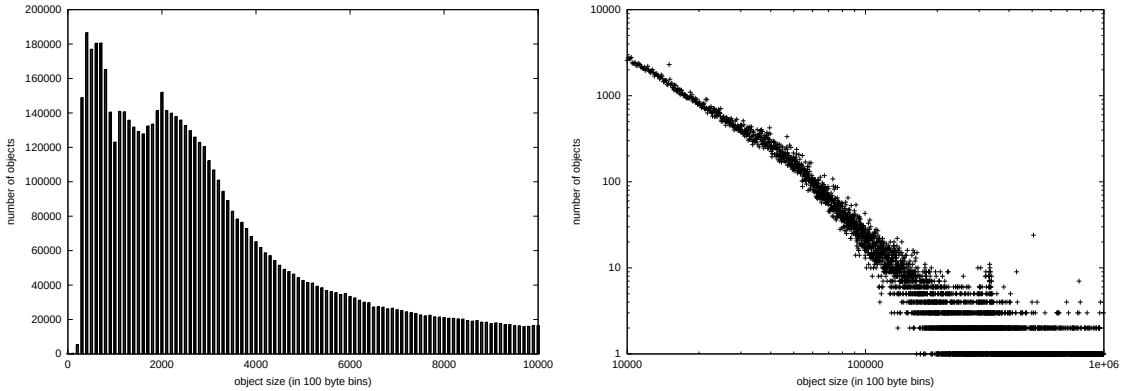

Figure 3: Number of objects as function of objects size for the RWTH trace: (left) linear scale for objects smaller than 10 KB; (right) log-log scale for objects larger than 10 KB

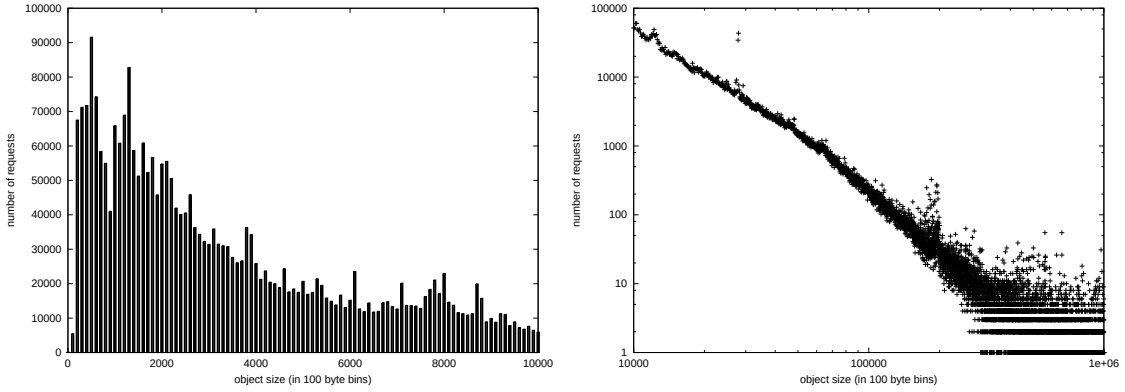

Figure 4: Number of requests by object size for the RWTH trace: (left) linear scale for objects smaller than 10 KB; (right) log-log scale for objects larger than 10 KB

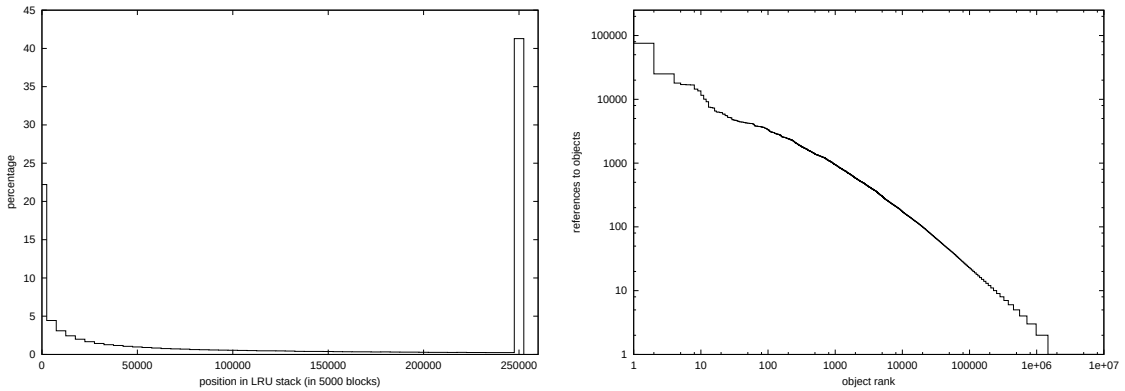

Figure 5: Analysis of the RWTH trace: (left) temporal locality characteristics (LRU stack-depth); (right) frequency of reference as a function of object rank (Zipf's law)

	64 MB	256 MB	1 GB	4 GB	16 GB	64 GB	256 GB
RWTH	0,04%	0,16%	0,64%	2,54%	10,17%	40,69%	162,7%
DEC	0,37%	1,49%	5,86%	23,43%	93,7%	374,8%	—

Table 4: Cache sizes as percentage of total amount of objects requested

Frequency of reference. Object which have often been requested in the past, are probably popular for the future too. This is explained by Zipf's law: if one ranks the popularity of words in a given text (denoted ρ) by their frequency of use (denoted P), then it holds $P \sim 1/\rho$. Studies have shown that Zipf's law also holds for WWW objects. Figure 5 (right) shows a log-log plot of all 8.3 million requested objects of the RWTH trace. As can be seen, the slope of the log-log plot is nearly -1, as predicted by Zipf's law, suggesting that a frequency-based strategies should be used. It should be mentioned that there are many objects that have been requested only once, namely 72.64% of all objects in the DEC trace and 67.5% in the RWTH trace. Frequency-based strategies have the advantage that "one timers" are poorly valued, so that frequently requested objects stay longer in the cache and cache pollution can be avoided.

4.2 Performance comparison

In this section, we only consider the RWTH trace; similar experiments were performed for the DEC trace (and reported in [14]). We performed the trace-driven simulations using our own simulator, written in C++. To obtain reasonable results for the hit rate and the byte hit rate, the simulator has to run for a certain amount of time without hits or misses being counted. The so-called *warm-up* phase was set to 8% of all requests, which corresponds to two million requests and a time periode of approximately four days.

The cache size is a decisive factor for the performance of the cache, hence, we want to choose the caching strategy that provides the best result for a given cache size. To compare the caching strategies, we have performed the evaluation with different cache sizes, as shown in Table 4.

First, we have to specify the parameters for the C-LRU strategy, as described in Section 3. Using the EM-algorithm with $I = 4$, the corresponding values for p_i and r_i are listed in Table 5 (with the cases (a)–(b) as presented in Section 3.2).

In Figure 6, we show the simulation results of the RWTH trace for the different caching strategies with respect to the hit rate and the byte hit rate, as a function of the cache size; notice that the cache size is expressed as percentage of the trace size. For the C-LRU strategy, we have included the results for the cases (a) and (b) as "C-LRU(a)" and "C-LRU(b)", respectively.

With respect to the hit rate, the simulations show that GDS-Hit provides the best performance for smaller cache sizes. However, for larger cache sizes, it is outperformed by C-LRU(a). The weak performance of C-LRU(a) for small absolute cache sizes can be understood when looking at the partition sizes: the assigned cache size of only 0.2% for class 4 ($i = 4$) is too small considering the fact that partition 4 is responsible for all objects larger than 367 kBytes. In practical use, this

			p_i			
i	c_i	λ_i	(a)	(b)	r_{i-1}	r_i
1	0.65	0.0003858	65%	16%	0	7455
2	0.321	0.0000798	32.1%	38.2%	7455	63985
3	0.027	0.000015633	2.7%	16.4%	63985	386270
4	0.002	0.000000646	0.2%	29.4%	386270	∞

Table 5: RWTH trace: parameters for C-LRU

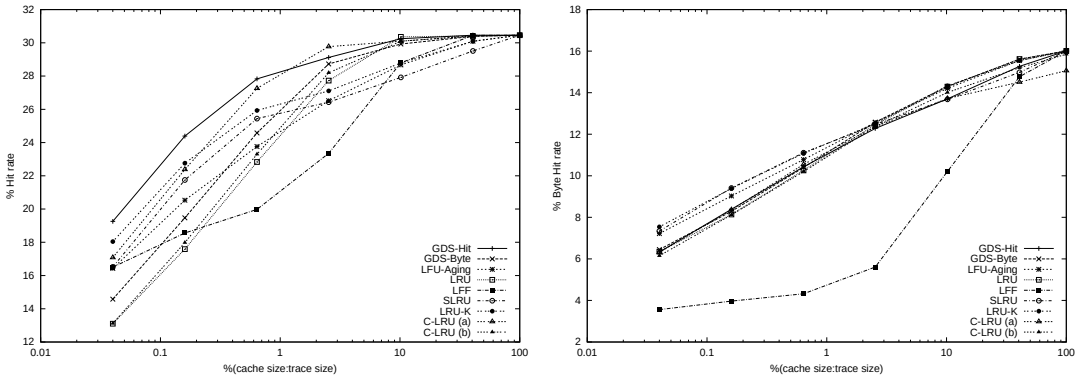

Figure 6: Hit rate (left) and Byte hit rate (right) comparision of the caching strategies for the RWTH trace as a function of the cache size

fact does not pose a problem since typical caches nowadays are larger than 1 GBytes and, indeed, C-LRU performs well for those cache sizes.

For the byte hit rate, one observes that the performance of all strategies is nearly equal, except for LFF which yields the worst results. C-LRU(a) shows a small performance decrease of about 1% for very large cache sizes. However, C-LRU(b) performs as good as the other strategies. Note that this behaviour is not surprising since C-LRU(a) has been chosen to optimise the hit rate. The reverse can be observed for C-LRU(b): optimised for the byte hit rate, its performance is quite low when considering the hit rate.

4.3 Time complexity

When choosing a caching strategy for practical use, the incurred CPU overhead for managing the cache is of utmost importance. Table 6 shows the time complexity of the typical operations performed by the cache, being (i) the identification of a cache hit or miss, (ii) the insertion of an object into the cache, (iii) the deletion of an object from the cache and (iv) the update of the specific data structures when an access to a cache entry has taken place. Note that N is the number of objects in the cache, and I is the number of C-LRU classes.

As can be seen, LRU, SLRU have the smallest time complexity. The performance of C-LRU depends on the number of classes I which, with values of I around 6, implies that the complexity of C-LRU is nearly equal to the complexity of LRU. In contrast, GDS and other methods require $O(\log N)$, where N is very large.

5 Conclusions

In this paper, we have proposed a new caching strategy which bases its replacement decisions both on the size of the requested objects as well as on the recency of the requests. We have shown that these characteristics are important for WWW proxy-server caching, making our strategy interesting for use in this area. For the performance of C-LRU, we can make two statements: considering the byte hit rate, its performance is comparable to existing strategies, but when looking at the hit rate, C-LRU is clearly better than most other strategies, sharing the first place with GDS-Hit depending on cache size. This is important since the response time of web servers,

	Hit/Miss	Insert	Delete	Update
LRU	$O(1)$	$O(1)$	$O(1)$	$O(1)$
SLRU	$O(1)$	$O(1)$	$O(1)$	$O(1)$
LRU-K	$O(1)$	$O(\log N)$	$O(1)$	$O(\log N)$
LFU	$O(1)$	$O(\log N)$	$O(1)$	$O(\log N)$
LFF	$O(1)$	$O(\log N)$	$O(1)$	$O(\log N)$
GDS	$O(1)$	$O(\log N)$	$O(1)$	$O(\log N)$
C-LRU	$O(1)$	$O(\log I)$	$O(\log I)$	$O(1)$

Table 6: Complexity for various cache operations

as perceived by the end users, is mainly determined by the hit rate [10]. The time complexity of C-LRU is nearly equal to LRU, that is, it is not dependent on the number of cached objects (as is the case for GDS-Hit).

The C-LRU caching approach naturally allows for an adaptive caching strategy if the parameters are recomputed at appropriate times. A thorough investigation of this aspect, however, will be presented in the near future. In a companion study, we have shown that a similar object-size classification can also be exploited for scheduling WWW server; we have reported about that in a separate paper [13].

References

[1] Digital Equipment Cooperation. Digital's Web Proxy Traces. ftp://ftp.digital.com/pub/DEC/traces/proxy.

[2] I. Tatarinov, A. Rousskov, and V. Soloviev. Static caching in web servers. In *Proc. 6th IEEE Int'l Conf. on Computer Communication and Networks*, pages 410–417, Las Vegas, September 1997.

[3] J. Gettys, T. Berners-Lee, and H. F. Nielsen. Replication and Caching Position Statement. http://www.w3.org/Propagation/activity.html, August 1997.

[4] J. Robinson and M. Devrakonda. Data cache management using frequency-based replacement. In *Proc. ACM SIGMETRICS '90*, pages 134–142, 1990.

[5] M. F. Arlitt and C. L. Williamson. Internet web servers: Workload chracterization and performance implications. *IEEE/ACM Transactions on Networking*, 5(5):631–645, 1997.

[6] M. F. Arlitt and C. L. Williamson. Trace-driven simulation of document caching strategies for internet web servers. *Simulation Journal*, 68(1):23–33, 1997.

[7] M. F. Arlitt, R. Friedrich, and T. Jin. Workload characterization of a web proxy in a cable modem. In *Proc. ACM SIGMETRICS '99*, pages 25–36, 1999.

[8] E. J. O'Neil, P. E. O'Neil, and G. Weikum. The LRU-K page replacement algorithm for database disk buffering. In *Proc. ACM SIGMOD '93*, pages 297–306, 1993.

[9] P. Barford, A. Bestavaros, A. Bradley, and M. Crovella. Changes in Web Client Access Patterns. *WWW Journal*, 2(1):3–16, 1999.

[10] P. Cao and S. Irani. Cost-aware WWW Proxy Caching Algorithms. In *Proc. USENIX*, pages 193–206, Monterey, CA, December 1997.

[11] P. Lorenzetti and L. Rizzo. Replacement Policies for a Proxy Cache. Technical report, Universita di Pisa, December 1996.

[12] R. El Abdouni Khayari, R. Sadre, and B.R. Haverkort. Fitting world-wide web request traces with the EM-Algorithm. In R. van der Mei and F. Huebner-Szabo de Bucs, editors, *Proceedings of SPIE*, volume 4523, pages 211–220, August 2001.

[13] R. El Abdouni Khayari, R. Sadre, and B.R. Haverkort. Weighted Fair Queueing Algorithm for WWW Proxies. In R. van der Mei and F. Huebner-Szabo de Bucs, editors, *Proceedings of SPIE*, volume 4865, August 2002.

[14] R. El Abdouni Khayari, R. Sadre, B.R. Haverkort, and M. Pistorius. A Class-Based Least-Recently Used Caching Algorithm for WWW Proxies. Technical report, Laboratory for Performance Evaluation and Distributed Systems, RWTH Aachen, 2002.

[15] S. Williams, M. Abrams, C. R. Standridge, G. Abdulla, and E. A. Fox. Removal policies in network caches for world-wide web documents. In *Proc. ACM SIGCOMM '96*, pages 293–305, 1996.