Practical LFU Implementation for Web Caching

G. Karakostas
Dept. of Computer Science
Princeton University
35 Olden St.
Princeton, NJ 08544
USA

D.N. Serpanos
Dept. of Computer Science
University of Crete
P.O. Box 1470
GR-71110 Heraklion, Crete
Greece

June 6, 2000

Abstract

Web caches can achieve high hit rates through exploitation of the properties of object access distribution, which is governed by Zipf's law, in general. Web caches need to store the most popular objects and typically employ the Least Frequently Used (LFU) replacement policy, which achieves high cache hit rates, and often the highest hit rate. Correct implementation of LFU requires that replacement decisions are made based on frequency access information (popularity), for all the objects accessed since the beginning of a cache's operation. The immensely large number of such objects renders the implementation of LFU impractical in many environments.

In this paper, we introduce an alternative implementation of LFU, the Window-LFU policy, which makes replacement decisions based on access frequency measurements in a recent past, called time-window. Window-LFU achieves cache hit rates equivalent to those of LFU, but with access information from a shorter history, leading to high performance at a low cost (significantly lower than that of LFU). We provide analytical results which enable one to estimate the appropriate window size, in order to achieve the target cache hit rate of LFU. Furthermore, we present simulation results using actual traces, which indicate that the proposed Window-LFU policy behaves as expected and in some configurations it leads to better results than theoretically expected, due to dependencies between successive Web objects requests in real environments. Our theoretical and simulation results demonstrate that Window-LFU provides an efficient solution for effective Web caches at a low cost, due to its shorter history measurements.

1 Introduction

The World Wide Web constitutes a significant technological advance, but importantly, it provides the means for the development and deployment of a wide range of services to end-users. The number of Web users (clients) is increasing dramatically, with exponential growth, at a rate which is larger than the deployment of bandwidth either in the backbone or to the end-user. This leads to significant congestion in the network, leading to long access delays, absence of Quality-of-Service (QoS), and low penetration of services to the electronic customer base. Caching of Web objects provides a promising solution to the problem of long access delay. Its potential and improved performance has led to the development and deployment of systems, which implement caching mechanisms [1] [3].

Effective and efficient caching requires careful analysis of the access patterns of Web objects, so that caches can store the objects that are most likely to be accessed in the near future. Analyses of traffic (user access patterns) on the Web show that accesses follow a non-uniform distribution. Specifically, they are governed by Zipf's law, following a Zipf or Zipf-like distribution [4] [2]. According to this law, the probability of requesting a particular object (page) is inversely proportional to its popularity. Specifically, considering a set of objects accessed by a set of clients, Zipf's law

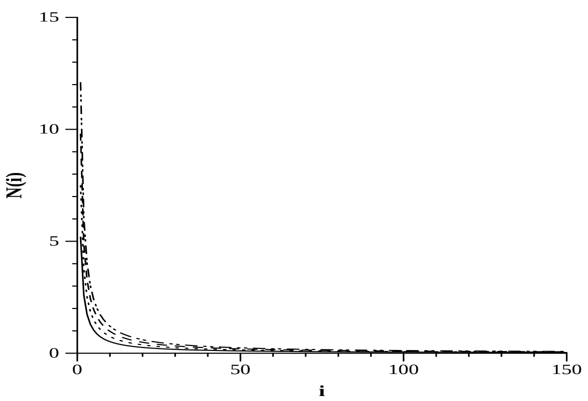

Figure 1: Zipf's function

enables us to calculate the number of accesses to each object based on its popularity. Assuming a set $S = \{O_i | 1 \leq i \leq N \text{ and } O_i \text{ is the } i\text{-th most popular object}\}$ of N objects, Zipf's function quantifies the probability that an access is made to object O_i: $P_i = \frac{a}{i}$ where a is a constant (it is easily calculated that $a = \frac{1}{H_N}$, where H_N is the N-th harmonic number). Figure 1 plots the probabilities for $N = 10^i$, where $2 \leq i \leq 5$.

This property of Web traffic enables us to develop Web caches which can achieve arbitrarily high cache hit rates, by storing the most popular objects and employing the LFU (Least Frequently Used) cache replacement policy, as has been shown analytically [7] and verified with simulations [2]. However, such caches suffer from two drawbacks: *(a)* the successful implementation of LFU (Perfect-LFU) requires the accumulation of access information from the beginning of a cache's operation, and *(b)* the size of the cache has to be large in order to achieve a high hit rate [7] [2].

In this paper we solve the problem of implementing efficiently the LFU policy for Web caching. Instead of examining *all* past requests in order to determine the popularity of each object, we take into account only the latest few requests to determine the ordering of the objects according to their popularity. Specifically, we introduce the concept of a *time window*, W, a time interval of the recent past, and implement an LFU policy, called *Window-LFU*, which replaces objects based on access measurements only in the window W. We prove analytically that the time window size can be chosen as a function of the cache size, independently of the number of available objects on the Web, and still achieve the same cache hit rate as Perfect-LFU, under certain assumptions. Considering that the number of objects on which we need to keep statistics cannot be larger than the window size, it becomes clear that a small window size leads to a small number of objects on which statistics are collected. In this fashion, we overcome the most significant obstacle to the implementation of LFU policies: the impractically large amount of resources necessary for the calculation and storage of statistics on accessed objects.

Furthermore, we show that our analytical result applies to realistic traffic through simulation based on actual traces. Actually, our simulation results indicate that, for some configurations our caching mechanism achieves in practice better results than expected from the analysis. This counter-intuitive result is due to our assumption (the same assumption has been made by others [2]), that the requests for objects are independent. Our experiments show that, in reality, there are dependencies, due to locality phenomena, and Window-LFU takes advantage of them to achieve higher cache hit rates than those expected theoretically, with even smaller window sizes than those predicted by our analysis. This leads to an efficient caching scheme with a low cost, since all information used is based only on the recent history of requests.

The paper is organized as follows. Section 2 describes the model we use and introduces our notation. Section 3 presents our analysis and its results, while Section 4 presents the simulation and its results.

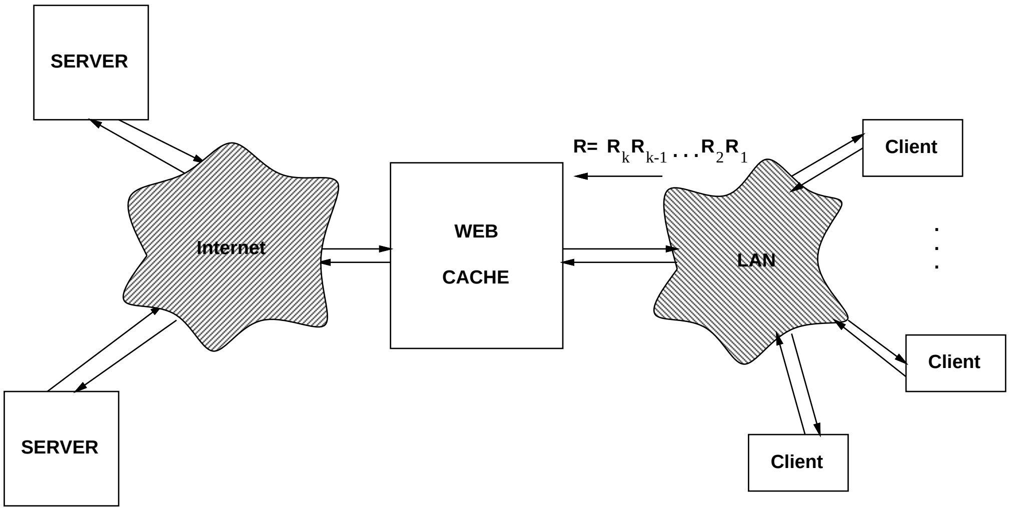

Figure 2: A simple caching environment

2 Model and Notation

We analyze a simple environment, as the one shown in Figure 2. In this environment, an enterprise network (or LAN) is connected to the Internet through a gateway, which also serves as a cache (for example, in a typical environment, the gateway could be a firewall). Users (clients) connect to Web servers through the gateway. So, user requests arrive to the gateway-cache and they are either forwarded to the Internet, or served through the cache, if the data are already cache-resident.

We assume that the set of all available objects, denoted $O = \{O_1, O_2, \ldots,\ O_N$, has size $\mid O \mid = N$. Also, we assume that client requests follow Zipf's distribution. Specifically, we assume that the stream of client requests, R, is a series of *independent* trials drawn from a Zipf (or Zipf-like) distribution over the set of N possible objects (e.g., web pages or sites). This means that the next request in R will be for the i-th most popular of the N items with probability

$$P_N(i) = \frac{\Omega}{i}$$

where

$$\Omega = \frac{1}{H_N} \approx \frac{1}{\ln N}$$

H_N is the N-th harmonic number, which we approximate with $\ln N$. Furthermore, we assume that the system is *closed*, i.e., that N, the total number of objects, and their nature do not change (no objects "die" and no new ones are "born"). This assumption is realistic for time intervals of the order of weeks or months, when we observe no dramatic changes in the population of requested objects.

In any caching scheme, a cache stores the items that have been accessed in some recent past, which we refer to as *time window* W (or simply *window*). We denote as $\mid W \mid$ the length of the window, measured in number of requests. The window W always contains the last $\mid W \mid$ requests, which are denoted as $W_1,\ W_2,\ \ldots,\ W_{|W|}$; for example, in Figure 2, window W contains requests $(W_1,\ W_2,\ \ldots,\ W_{|W|}) =\ (R_k,\ \ldots,\ R_{k-|W|+1})$. The existent analytical results have been drawn for the case $\mid W \mid = \mid R \mid$, where R contains all requests received by the cache since the beginning of its operation [7].

Considering the definition of W, as the $\mid W \mid$ most recent requests in R, we define $n_W(i)$ as the number of appearances of the i-th most popular object, object O_i, in W; the definition of the i-th most popular object is based on the number of requests in R. The expected value of $n_W(i)$ is easily calculated:

$$E[n_W(i)] = \frac{\mid W \mid}{i \ln N}$$

We denote this value as $E(i)$.

3 Estimation of Window Size

The goal of our analysis is to estimate the length of W, so that, if the cache measures access frequencies using the information in the last $\mid W \mid$ accesses, then the achieved cache hit rate approximates the one achieved with Perfect-LFU. We formalize this, through the following definition:

Definition 1 (Good estimator) *Let C be the number of objects that are kept in the cache. Then the window W will be a* **good estimator** *of the C most popular objects in R, if two conditions are met:*

- *the number of appearances of the C most popular objects is greater than $E(C+1)$;*
- *the number of appearances of the remaining $N-C$ objects is smaller than $E(C+1)$, i.e. the remaining objects do not interfere with the ordering of the C most popular ones.*

The definition indicates that, while we ensure a separation between the C most popular objects and the $(N-C)$ less frequent, the conditions of the definition are too weak to ensure the correct ordering of the objects according to their access frequencies in the complete history (for a stronger condition, the reader is referred to [6]). However, the critical observation is that for the implementation of Perfect-LFU, it is sufficient to have the C most popular objects in the cache, without a need for knowledge of the specific order of their frequencies (popularities) in the window. The weakness of the conditions in the definition are the key of the improvements we achieve.

If both of the conditions are met, then we designate the window as *good*. In this case, our replacement algorithm, Window-LFU, will provide exactly the same performance (hit rate) as Perfect-LFU. So, the goal of our analysis is to choose W in such a way, so that it will ensure the "goodness" of the window with very high probability. Then our hit-rate will be very close to the one achieved with Perfect-LFU.

In the analysis, we use the following Chernoff bounds:

Lemma 1 (Chernoff bounds) *Let $X_1, X_2, \ldots, X_n$ be mutually independent random variables such that*

$$Pr[X_i = 1] = p$$

$$Pr[X_i = 0] = 1 - p$$

for some $p \in [0, 1]$.
Let $X = X_1 + X_2 + \ldots + X_n$ and $E[X] = pn$. Then

$$Pr[X > (1+\theta)pn] \leq e^{-\frac{\theta^2}{3}pn} \tag{1}$$

$$Pr[X - pn < -\alpha] < e^{-\alpha^2/2pn} \tag{2}$$

$$Pr[X - pn > \beta] < e^{-2pn/27} \tag{3}$$

$$Pr[X - pn > \gamma] < e^{\gamma - (\gamma + pn)\ln(1+\gamma/pn)} \tag{4}$$

where $0 < \theta \leq 1,\ \alpha > 0,\ \beta > 2pn/3$ and $\gamma > 0$.

We use these bounds, because they describe quantitatively the following simple fact: a series of independent trials is concentrated very heavily around its expected value. We use this fact to prove that, one does not need many trials, i.e. past requests, in order to get a very good estimate of the expected value, i.e. the frequency.

3.1 Theoretical upper bounds for window size

Assume that, the N objects are ordered according to their popularity in R (O_1 is the most popular, O_2 the second most popular, etc.). We define the following sequence of random variables for each

O_i:

$$w_j(i) = \begin{cases} 1, & \text{if } W_j \text{ (the } j\text{-th request in } W\text{) is for } O_i \\ 0, & \text{otherwise} \end{cases} \qquad j = 1, 2, \ \ldots, \ | W |$$

Then, by hypothesis, the $w_j(i)$'s are mutually independent, $n_W(i) = \sum_{j=1}^{|W|} w_j(i)$ and $Pr[w_j(i) = 1] \approx \frac{1}{i \ln N}$ from Zipf's distribution.

We distinguish the following cases:

Case 1: $1 \leq i \leq (C+1)$

¿From inequality (2) with $\alpha = E(i) - E(C+1)$ we obtain:

$$Pr[n_w(i) < E(C+1)] = Pr[n_w(i) - E(i) < -(E(i) - E(C+1))] < e^{-\frac{|W|(C+1-i)^2}{2i(C+1)^2 \ln N}} \leq e^{-\frac{|W|}{2C \ln N}} \quad (5)$$

Case 2: $C+1 < i < 2(C+1)$

¿From inequality (1) with $\theta = \frac{i-C-1}{C+1}$ we obtain:

$$Pr[n_w(i) > E(C+1)] < e^{-\frac{(i-C-1)^2}{3(C+1)^2} \frac{|W|}{i \ln N}} \leq e^{-\frac{|W|}{3C(C+1)^2 \ln N}} \quad (6)$$

Case 3: $2(C+1) \leq i \leq 3(C+1)$

¿From inequality (3) with $\beta = E(C+1) - E(i)$ we obtain:

$$Pr[n_w(i) > E(C+1)] < e^{-\frac{2|W|}{27 i \ln N}} \leq e^{-\frac{|W|}{40.5(C+1) \ln N}} \quad (7)$$

Case 4: $3(C+1) < i \leq N$

From inequality (4) with $\gamma = E(C+1) - E(i)$ we obtain:

$$Pr[n_w(i) > E(C+1)] < e^{E(C+1) - E(i) - E(C+1) \ln \frac{i}{C+1}} \leq e^{-\frac{2|W|}{5(C+1) \ln N}} \quad (8)$$

The probability that a window is not a good estimator is evaluated as:

$$Pr[\text{the window is } \textit{not} \text{ a good estimator}] = Pr[\text{Case 1 holds} \vee \ldots \vee \text{Case 4 holds, for some } i]$$

$$\leq Ce^{-\frac{|W|}{2C \ln N}} + Ce^{-\frac{|W|}{3C(C+1)^2 \ln N}} + (C+2)e^{-\frac{|W|}{40.5(C+1) \ln N}} + (N - 3(C+1))e^{-\frac{2|W|}{5(C+1) \ln N}} \quad (9)$$

If we choose $|W| = \max\{\Theta(C^3 \ln C \ln N), \Theta(C \ln^2 N)\}$, then we can force the probability in Equation (9) to be smaller than any constant $\epsilon > 0$.

If we denote with $H_{\text{W-LFU}}(C, W)$ and $H_{\text{P-LFU}}(C)$ the cache hit rates for the Window-LFU and Perfect-LFU cases, respectively (with the window size $|W|$, as specified above), the following relations hold:

$$H_{\text{P-LFU}}(C) = \sum_{i=1}^{C} \frac{1}{i \ln N} \quad (10)$$

$$
\begin{aligned}
H_{\text{W-LFU}}(C,W) &= Pr[\text{next requested item } r \text{ is in the cache}] && (11)\\
&\geq \sum_{i=1}^{C} Pr[r=i|W \text{ is good estimator}] \times Pr[W \text{ is good estimator}] && (12)\\
&= \sum_{i=1}^{C} \frac{1}{i \ln N} \times (1 - Pr[W \text{ is } \textit{not} \text{ a good estimator}]) && (13)\\
&\overset{(9)}{\geq} (1-\epsilon) \sum_{i=1}^{C} \frac{1}{i \ln N} && (14)\\
&= (1-\epsilon) H_{\text{P-LFU}}(C) && (15)
\end{aligned}
$$

where $\epsilon > 0$ is the accuracy constant we have chosen.

3.2 The importance of the theoretical results

From the analysis, it becomes clear that the required window size depends only on the logarithm of the total number of objects N that can be accessed. Thus, we succeed to reduce exponentially the effect of parameter N on our cache replacement policy, which is an advantage because N is not a parameter of the cache system itself, and we cannot control it. control. As the number of objects for which one keeps statistics cannot be larger than the window size, a smaller time window results in a smaller set of such objects. Unfortunately, Case 2 shows a dependency on the cache size C, which is impractical for big enough caches. However, the result is very strong (approximation of Perfect-LFU performance within any constant factor), which means that, in practice, smaller window sizes should perform quite well, e.g., $|W| = O(C \ln N)$ or $O(C^2 \ln N)$. This is supported by the results of simulations with traces of real traffic, as described below.

4 Simulation Results

We have performed several simulations of a cache employing the Window-LFU policy using traces from actual traffic patterns. Specifically, we have used two traces from NLANR [5]. The first of the traces is short, it includes the object requests of one day, while the second one is longer, including the requests of a week. The traces are continuously updated; the ones we used are from the last week of January 2000.

Our simulator simulates a cache that uses Window-LFU replacement policy for variable window sizes W. When a replacement of an object is due, the object with the smallest frequency is replaced. If more than one objects in the cache have the same (smallest) frequency, then we replace the one which was used *least recently*; i.e., we use an LRU (Least Recently Used) rule in order to "break ties" among the least popular window objects.

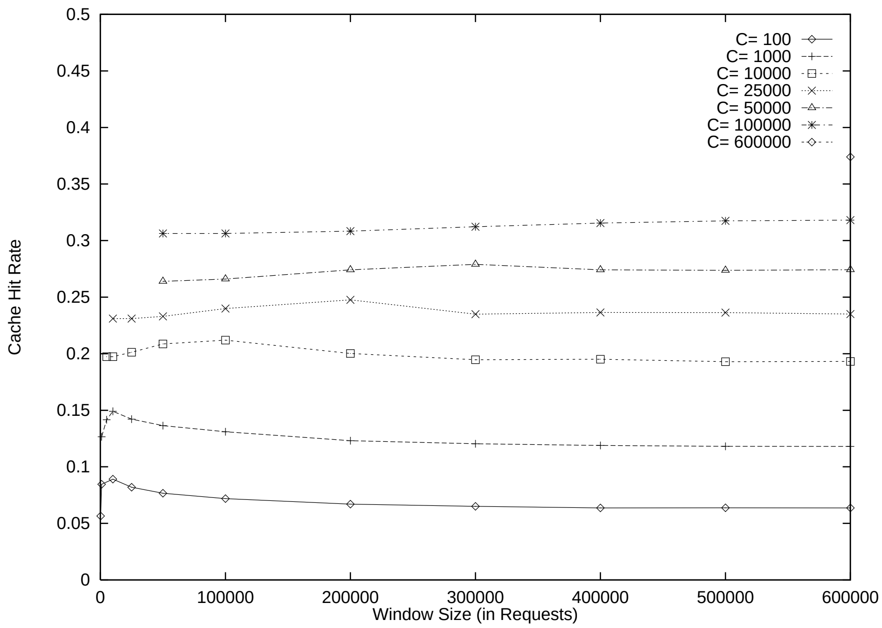

Figure 3: Cache hit rate for variable window sizes (short trace)

Figure 3 shows the results of the simulations, using the short trace with a total of $375,000$ objects and $600,000$ requests. The plot shows the cache hit rate of a cache with Window-LFU as a function of the size of window W, and for various cache sizes C (measured in objects). As the results indicate, for all cache sizes, the effect of the window size is insignificant after a "threshold" value. This verifies our first result that, a small window size is sufficient to achieve the highest possible cache hit rate (per cache size). Interestingly, with small cache sizes and small window lengths, it appears that the cache hit rate improves. This seemingly surprising result can be explained easily: the source is the dependency among successive requests. In our analyses, we have assumed that the object requests R_1, R_2, $\ldots$, are independent. However, in real traces there is a dependency among them, which actually leads to higher locality, and thus improves the hit rate. As the length of the window increases, the cache hit rate, which is $h = \frac{Number\ of\ cache\ hits}{Number\ of\ Requests}$, decreases, because the "longer" history (due to the longer window size) tends to influence the replacement decision using popularities from a distant past, which do not apply to the recent past (due to the dependency of requests). In simpler terms, this means that, if an object was accessed heavily in the distant past, but is not accessed any more, Perfect-LFU will not replace this object from the cache, unless a new object is accessed at least as many times as the previous one; in the (possibly very long) meantime, the object will reside in the cache, although it is not accessed at all. So, with the longer window size, it takes longer for the cache to store the more recently accessed objects, which are more likely to be accessed in the near future due to the aforementioned locality. On the other hand, a small window will not allow accesses made in the distant past to be counted against the calculation of object frequencies. Thus, considering locality, the cache will store objects more likely to be accessed in the future.

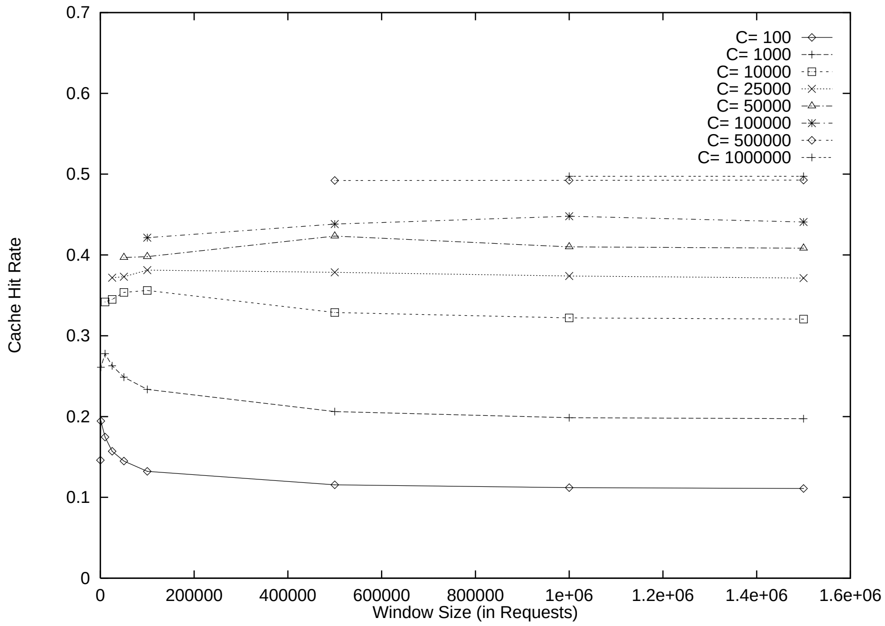

Figure 4: Cache hit rate for variable window sizes (long trace)

Figure 4 shows the results of the simulations, using the long trace with a total of $751,000$ objects and 1.5 million requests. As the results show, the behavior of the cache is similar to the one with the shorter trace, but with increased cache hit rates; this is due to the higher average number of accesses per object in this trace. As in the previous results, the size of the window plays has insignificant effect on the cache hit rate over a "threshold" value and the locality of accesses leads to higher hit rates for small caches and small window sizes, similarly to the short trace. Importantly though, the effect of the dependency among requests is more dramatic in this trace, leading to the significantly higher hit rates observed for small caches and window sizes.

Overall, the simulation results verify the analytical results of Section 3. Importantly, the simulations indicate that Window-LFU performs better than expected from the analytical results. This phenomenon is due to that, the assumption of request independence made for the analysis does not hold; there are dependencies in a real trace of requests, which actually render the Window-LFU more effective than analysis indicates.

5 Conclusions

We introduced Window-LFU, a novel, LFU-based replacement policy for effective and efficient Web caches. Window-LFU makes replacement decisions based on access frequency measurements of objects in a recent past, called time-window, in contrast to Perfect-LFU, which measures object access frequencies for all objects and from the beginning of the cache's operation.

Window-LFU achieves cache hit rates equivalent to those of LFU, but with access information from a shorter history, leading to high performance at a low cost (significantly lower than that of LFU). We have provided analytical results which enable one to estimate the appropriate window size, in order to achieve the target cache hit rate of Perfect-LFU. Furthermore, we presented simulation results using actual traces, which indicate that the proposed Window-LFU policy behaves as expected. Importantly, in some configurations Window-LFU leads to better results than theoretically expected, due to locality phenomena, i.e. dependencies between successive Web objects requests in real environments. Our results demonstrate that Window-LFU provides an efficient, practical solution for effective Web caches at a low cost, due to its shorter history measurements.

References

[1] Akamai Technologies, Inc. http://www.akamai.com.

[2] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. In *Proceedings of Infocom'99*, 1999.

[3] CacheFlow, Inc. http://www.cacheflow.com.

[4] C.R. Cunha, A. Bestavros, and M.E. Crovella. Characteristics of WWW Client-based Traces. Technical Report BU-CS-95-010, Computer Science Department, Boston University, July 1995.

[5] National Laboratory for Applied Network Research. http://www.nlanr.net (traces at: ftp://ircache.nlanr.net/traces/.

[6] D.N. Serpanos, G. Karakostas, and W.H. Wolf. Effective Caching of Web Objects Using Zipf's Law. In *Proceedings of IEEE International Conference on Multimedia and Expo (ICME 2000)*, page (To appear), July, 30 - August, 2 2000.

[7] D.N. Serpanos and W.H. Wolf. Caching Web Objects Using Zipf's Law. In *Proceedings of SPIE, Vol. 3527, Photonics East, Technical Conference 3527: Multimedia Storage and Archiving Systems III, Boston, MA, USA, November 2-4, 1998, pp. (not available yet). See "http://www.spie.org/web/meetings/programs/pe98/confs/3527.html*, 1998.