Exploitation of Different Types of Locality for Web Caches

G. Karakostas*
Applied Research
Telcordia Technologies Inc.
Morristown, NJ 07960
USA
gk@research.telcordia.com

D.N. Serpanos
Dept. of Electrical and Computer Eng.
University of Patras
GR-26500 Rion, Patras
Greece
serpanos@ee.upatras.gr

Abstract

Object access distribution in the Web is governed by Zipf's law, in general. This property leads to effective Web caches, which store the most popular objects and typically employ the LFU replacement policy, which achieves high, and often the highest, cache hit rates. However, Web cache design based only on Zipf's law has two main disadvantages: (i) it does not exploit the temporal and spatial locality of user accesses on a per session basis, and (ii) LFU implementation is costly and impractical in many environments, because it requires statistics on all objects accessed since the beginning of a cache's operation.

We consider all parameters of locality of references in the Web (temporal, spatial and popularity) and draw an analogy with processor caches. Given cache replacement policies that address different locality characteristics, we argue that there exist replacement algorithms that combine these characteristics and achieve high performance at a low cost. We describe Window-LFU (W-LFU), a policy that combines LFU and LRU and achieves better performance than LFU at lower cost. W-LFU exploits both Zipf's law, and temporal locality by using the accesses in a recent time-window. Simulations with actual traces indicate that W-LFU provides better results than theoretically expected.

1 Introduction

The World Wide Web constitutes the basic platform for the development and deployment of a wide range of services to end-users. The number of Web users is increasing exponentially, at a rate larger than the deployment of bandwidth either in the backbone or to the end-user. This leads to significant congestion in the network and results 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 caching requires careful analysis of the access patterns for Web objects, so that caches exploit locality of references, i.e. they should store the objects that are most likely to be accessed in the near future. Analyses of Web traffic (user access patterns) show that accesses are governed by Zipf's law, following a Zipf or Zipf-like distribution [4] [2]. According to Zipf's law, the probability to request a particular object is inversely proportional to its popularity (a formal definition is given in Section 3).

Modelling Web traffic with Zipf-like distributions enables the development of Web caches which can achieve arbitrarily high cache hit rates, by storing the most popular objects and employing the LFU replacement policy, as has been shown analytically [9] and verified with simulations [2]. Existing results show that, for large enough cache sizes, LFU (also called the *Perfect-LFU* or *P-LFU* [2]) is optimal. Furthermore, P-LFU provides improved results even for smaller caches over alternative, widely used policies (e.g. LRU) for some traces used in the simulations of [2]. Thus, P-LFU is the best policy to implement in certain environments (for certain traces and cache sizes), in order to achieve the highest hit rate. However, implementation of P-LFU has high cost, because it requires the accumulation of all access information from the beginning of a cache's operation; in many environments, this cannot be achieved with the limited resources of a cache system.

Although exploitation of Zipf's law leads to effective caching architectures, such caches do not address sufficiently the characteristics of temporal and spatial locality within user sessions. Such locality exists in web accesses, and its exploitation may lead to improved cache performance over existing solutions. Since processor caches have

*This work was done when the author was at the Dept. of Computer Science, Princeton University, with support by NSF CAREER award NSF CCR-9502747, an Alfred Sloan Fellowship, and a Packard Fellowship.

been designed to exploit temporal and spatial locality, we draw an analogy between processor caches and Web caches. In this analogy, program accesses correspond to user session requests, experiencing temporal and spatial locality on a per-session basis; furthermore, accesses from different users to the Web are correlated and, specifically, they obey Zipf's law, in contrast to accesses from different programs, which are not known to correlate in some well-understood or well-defined fashion. This analogy leads to a view where Web accesses are better characterized than processor accesses. As a result, all types of locality, temporal, spatial and popularity, can be viewed to work synergistically and designers can take advantage of all types of locality to provide an effective solution. However, efforts to exploit different types of locality seem to result in the use of contradicting replacement policies: exploitation of temporal locality leads to use of LRU, while exploitation of popularity –and Zipf's law, in general– leads to use of LFU.

In this paper, we argue that it is feasible to develop cache replacement methods that address the requirements of all types of locality. Specifically, we describe Window-LFU, W-LFU, a replacement method that combines characteristics of both LFU and LRU, and leads to caches that provide improved performance over existing solutions at a lower cost. W-LFU solves two problems: (a) efficient implemention of LFU in Web caches, and (b) exploitation of *all* types of locality that characterize user sessions.

The development of W-LFU is based on a proof that, under the assumption of statistical independence for the requests, the hit rate achieved by LFU, when only a small subset is used, is equivalent to the hit rate achieved by LFU, when the *whole* history is used. Based on the property that this subset can be from any portion of past history, W-LFU uses a subset which is composed of the latest few requests. These requests are sufficient to order the cached objects according to their popularity; this is the necessary information to make replacement decisions, rather than the exact number of past object requests used by P-LFU. The motivation to use the most recent requests is actually twofold:

1. W-LFU avoids using the whole history, since it may not even be available (traces are recorded only for short periods of time, e.g. for a week);

2. despite the assumption of statistical independence in the proof, in practice there exist locality phenomena and dependencies, which are exploited by W-LFU and lead to performance that is higher than the theoretical limits of P-LFU, as simulations show.

In this paper we study the connection between the different types of locality that appear in the behaviour of a Web client and the design of efficient Web caching policies. First, we identify the types of locality one should expect in a stream of Web accesses, and compare them to the types of locality observed in the memory accesses of a computer program. Then, we discuss the effects of locality on Web cache design and we demonstrate this relation by showing how a particular policy, W-LFU [5], can be designed to take advantage of all locality types *simultaneously*.

The paper is organized as follows. Section 2 describes the analogies of processor caches and Web caches, and puts in perspective all different types of locality and their influence on cache design. Section 3 presents Window-LFU and its analysis, while Section 4 describes our simulations.

2 Locality of References and Cache Design

The design of processor caches is based on the exploitation of temporal and spatial locality. Temporal locality captures the characteristic that a program which makes a memory reference will use the same memory address in the near future with high probability. Spatial locality captures the characteristic that, when a program makes a memory reference, then it will access its neighboring addresses with high probability. These two types of locality of reference are the basis of cache design and have been analyzed extensively for various programs. Although all programs experience both types of locality, there does not seem to exist a uniform model to characterize locality in all programs; e.g., there does not seem to exist a well-understood distribution of memory references that characterizes all (or the majority of) programs. This lack of a *common* distribution results to the absence of a model for memory accesses which would enable analysis of cache architectures and leads to analysis through traces from benchmark (or typical) applications.

Temporal and spatial locality characterize Web accesses as well. In order to analyze Web accesses analogously to memory references, we consider Web requests for objects (pages, or items in pages) analogous to memory accesses in processors; furthermore, we consider user sessions analogous to program executions on a processor system. Similarly to multiple programs that may run on a processing system, several users may establish sessions simultaneously on the Web. As programs request accesses to memory locations, Web users request objects from the Web (which is considered as a memory). Simple analysis of user behavior indicates that temporal and spatial locality characterize users' Web accesses: a page (or set of items in a page) may be re-visited in the near future with high probability (temporal locality) and pages (or items) that are linked to a viewed page will be accessed with high probability (spatial locality). However, in contrast to the case of processors executing multiple programs, Web clients (users) make requests which follow a well-understood distribution (Zipf's function), which is derived from Zipf's law. Zipf's law specifies that accessed objects are characterized by different popularity and quantifies the number of accesses to each object exactly. Thus, Web accesses can be modeled easier than memory references of programs and this enables one to develop Web cache architectures that are at least as effective as processor caches. This view of locality of Web accesses is the key to our approach for the design of Web caches.

The design of a cache, whether for a processor or for an Internet gateway allowing Web access, has a relatively small number of parameters: total number of entries, size of each entry, and replacement policy (the method that replaces entries, when a new entry is placed in a full cache).

In processor caches, the size of the cache –in terms of number of entries– is directed by the characteristics of the programs that are typically executed by a processor; specifically, the number of entries is such that a high hit ratio is achieved for a number of programs which are considered the typical load of the designed processor. The size of each entry –the number of words (block) that are fetched from main memory for each memory access– is again tailored to lead to high cache hit rates, based on the analyzed traces. Typically, the number of cache entries and the size of each entry are decided under the implementation cost and target performance of the system, since there is a tradeoff between number of entries and entry size for a specific memory size used for the cache. In typical processors and caches, the replacement policy of choice is LRU, because it is considered optimal in this environment (due to the high degree of temporal locality in most programs).

In Web accesses, most efforts up to date have focused on the property that Web accesses by a large population of users are governed by Zipf's law and specifically, from a Zipf or Zipf-like distribution [9] [2] [4]. Special effort is made to explain the differences of real traces, which follow a Zipf-like distribution, from the exact Zipf function. In our work, we argue that these differences, which are noted mainly for the most popular sites, are due to the convolution of Zipf's function with the functions that characterize temporal and spatial locality; it should be expected that the effect of such locality is more obvious in the most popular sites rather than in the less popular ones, as appears in real traces (the deviation from the expected numbers for non-popular objects disappears as the number of accesses increases). Although not directly addressed, temporal and spatial locality in user sessions explains higher cache hit ratios in systems where the management of cache space uses locality information, as, for example, in a system which optimizes disk storage use associating URLs which are served from the same server [6]; in that system, URLs served from the same server are effectively treated as if they had spatial locality dependencies.

Analyses of Web caches using Zipf's law have led, in general, to designs with caches of large size [9] [2], which use LFU as the prefered replacement policy, due to Zipf's function. The use of LFU in these environments however, is difficult to implement, because it requires collection of statistics from the beginning of a cache's operation, in order to make the appropriate replacement decisions.

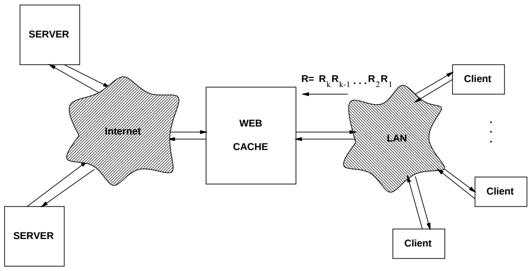

Figure 1. A simple caching environment

From the above, it seems that Web caches and processor caches differ significantly in their design decisions, not only in size (Web caches must be significantly larger than processor caches in terms of number of entries –objects), but in their replacement policy as well. LRU addresses temporal locality well, while LFU addresses popularity well. We argue that these two policies are not necessarily contradicting, but they can operate synergistically and can be combined in new policies that address all types of locality: temporal, spatial and popularity. Furthermore, combination of these policies enables the development of efficient and cost-effective caches (in terms of fast decision making for entry replacement and storage of access history), which achieve improved performance, over alternatives, at a lower cost. W-LFU is a first such algorithm, which achieves improved performance over P-LFU at a lower cost.

3 Window-LFU: Description and Analysis

Window-LFU operates essentially in the same fashion as Perfect-LFU, with the only difference that replacement decisions are made based on statistics for a *subset* of all objects that have been accessed in the past (whereas Perfect LFU uses statistics for *all* of them). In this section, we introduce the model and notation we use, and we provide the results on which the development of Window-LFU is based.

3.1 Model and Notation

We analyze a Web environment, as shown in Figure 1, where 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 by 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 $| O |= N$. 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

$$P_N(i) = \frac{\Omega}{i} \text{ where } \Omega = \frac{1}{H_N} \approx \frac{1}{\ln N}$$

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 $| W |$ the length of the window, measured in number of requests. The window W always contains the last $| W |$ requests, which are denoted as $W_1, W_2, \ldots, W_{|W|}$; for example, in Figure 1, 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 $| W |=| R |$, where R contains all requests received by the cache since the beginning of its operation [9].

3.2 Description and analysis of Window-LFU

Window-LFU (W-LFU) is an LFU variation that differs as follows:

- it caches the $|W|$ *most recent* requests;
- cache object replacement is performed with LFU, using the object popularities in window W;
- if two or more objects have the same (minimum) popularity in a replacement decision, LRU is used to identify the object to replace.

In the following, we analyze W-LFU and prove that it achieves equivalent performance to P-LFU, under the assumptions of a Zipf distribution of user accesses, and statistical independence among requests. The issue of dependencies is considered in Section 4, where simulations show that W-LFU exploits them and achieves improved performance over what predicted by the independence-based analysis.

The goal of our analysis is to estimate $| W |$, so that, if the cache measures popularities using the last $| W |$ accesses, then the cache hit rate approximates the one achieved with P-LFU. First, we introduce a definition:

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

- *the number of appearances in W 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 $E(C+1)$.*

The definition ensures a separation between the C most popular and the $(N - C)$ less popular objects using conditions that are too weak to ensure the correct ordering of the objects according to their popularities in the complete history (a stronger condition is considered in [8]). This weakness of the conditions is the key of the improvements we achieve. The critical observation is that, for the correct implementation of Perfect-LFU, it is sufficient to store the C most popular objects in the cache, without a need for knowledge of the specific order of their popularities in W.

A window is designated as *good*, when both conditions are met, and then, W-LFU will provide exactly the same performance as Perfect-LFU. Thus, our goal is to choose W in such a way, so that it is *good* with very high probability, and then W-LFU's hit rate will be very close to the hit rate of Perfect-LFU. We prove the following theorem:

Theorem 1 *Let $\epsilon > 0$ be any parameter, and C the cache size. Under the assumptions of statistical independence and Zipf's distribution for the requests*

$$H_{\text{W-LFU}}(C) \geq (1 - \epsilon) \cdot H_{\text{P-LFU}}(C)$$

where $H_{\text{W-LFU}}(C)$ is the hit rate of W-LFU with $| W |$:

$$|W| = \max\{\Theta(C^3 \ln C \ln N \ln \frac{1}{\epsilon}), \Theta(C \ln^2 N \ln \frac{1}{\epsilon})\}$$

and $H_{\text{P-LFU}}(C)$ is the hit rate of Perfect-LFU.

Proof: We exploit the statistical independence assumption using Chernoff bounds. These bounds are suitable for our analysis because they describe quantitatively a 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 (popularity). The full proof appears in [5]. □

Note that, the proof of Theorem 1 indicates that W-LFU approximates Perfect-LFU for the appropriate size of W, but any set of $| W |$ past requests can be used; it is not necessary to choose W so that it includes the most recent requests. Our preference for the most recent requests is due to our effort to capture temporal locality, as described below.

Although Theorem 1 is stated for Web accesses that follow Zipf's distribution, an analogous theorem can be proved for Zif-like distributions (e.g., the distributions described in [2]). In this case, the bound of Theorem 1 becomes $|W| = \max\{\Theta(N^{1-\alpha}C^3 \ln C \ln \frac{1}{\epsilon}), \Theta(N^{2-2\alpha}C \ln \frac{1}{\epsilon})\}$. In practice α is a number between 0.6 and 0.9 (cf. [2]), so the window size depends *sublinearly* on N.

W-LFU is a generalization of a well known cache replacement scheme used in the Operating Systems community: instead of examining the whole past in order to make cache replacement decisions, the policy considers only the few *most recent* requests and throws out of the cache the least frequently used object in this near past. Since few past requests means a small number of objects one needs to keep statistics for, W-LFU can be implemented efficiently, in contrast to P-LFU that needs statistics for *all* the objects accessed in the past. The intuition behind the decision to use the most recent requests is based on the temporal locality exhibited by computer programs: if a memory page is accessed, then with high probability it will be accessed again in the very near future. Similarly, multiple accesses to a Web page tend to cluster into small time windows.

Clearly, we can take advantage of both temporal locality and popularity, if we choose W to be the set of most recent requests. As temporal locality invalidates the statistical independence assumption of Theorem 1, the theoretical results do not apply. Instead, we provide simulation results that show the double influence of both kinds of locality. There is a window size (much smaller than predicted by Theorem 1) for which the hit rate of W-LFU not only approximates but actually *surpasses* the hit rate of P-LFU. It seems that W-LFU has the advantages of both P-LFU (which is optimal for large enough cache sizes [9] [2]) and temporal locality sensitive policies like LRU or in-cache LFU (cf. [2] for a comparison of these policies).

W-LFU exploits spatial locality by choosing W to be a *contiguous* subset of accesses. Very often the request for a Web page is followed immediately by requests for the objects linked with this page (pictures, sound and video clips, etc.). Since the popularities of these objects are the same as the popularity of the page itself, it follows that W will contain nearby not only the most popular pages but also the objects linked to it. This observation seems trivial, but it is not hard to see that a non-contiguous window from the recent past (e.g. every other of the most recent $2|W|$ requests) would still take advantage of the temporal locality and popularity, but it wouldn't exploit spatial locality as effectively.

4 Simulation Results

We have performed several simulations of a cache with Window-LFU using real traffic traces. The simulator implements a cache with variable window size W and variable size C.The implementation of Window-LFU follows closely the description in Section 3.2. The simulator keeps the objects in the cache ordered (sorted) according to their popularity. Maintenance of the order is simple: as the arrival of a request at the cache changes the popularity of one object, the only operations required for the ordering are (a) an update of the object's popularity, and (b) interchange (if necessary) of the order of this object with the one with higher popularity.

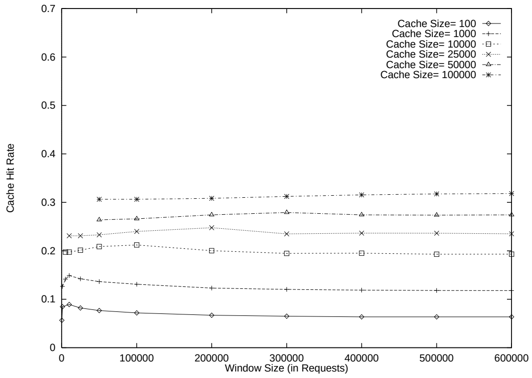

Figure 2. Cache hit rate for variable W

In order to evaluate the performance of the cache in a real environment, we have used actual traffic patterns from NLANR [7], which are either short (object requests of one day), or long (requests of a week). Figure 2 plots the cache hit rate achieved for a trace, as a parameter of the window size W; different curves are provided for variable cache size C. It plots the simulation results for a daily trace that contains $600,000$ requests for a total of $375,000$ different objects.

The simulation plots indicate that as the window size increases, Window-LFU tends to achieving the hit rate of Perfect-LFU. When the window size $|W|$ becomes equal to the trace size, Window-LFU becomes Perfect-LFU (the Perfect-LFU hit rate is the rightmost point of each plot). But for small cache sizes and small window sizes, the plots reveal that Window-LFU achieves significantly *better* hit rate than Perfect-LFU. This effect becomes less significant as the cache size increases, and can be explained through temporal locality as follows:

As the length of the window increases, the cache hit rate, which is $h = \frac{Number\ of\ cache\ hits}{Number\ of\ Requests}$, decreases; this occurs, because the "longer" history (due to the longer window size) influences the replacement decision using popularities from a distant past, which do not apply to the recent past (due to temporal locality). Effectively, this means that, an object which was popular in the distant past (i.e., it was accessed heavily in the distant past), but which is not popular any more, will reside in the cache (and thus play a role in replacement decisions) for a long time, although it is not accessed at all any more. It will be replaced in the cache only when it becomes the least frequently used object and when a new object is accessed at least as many times as this object was accessed in the past. This implies that, a cache with a longer window size, i.e. a cache that takes advantage of popularity mainly, is "inert" and takes a long time to change its state and store the more recently accessed objects, which are more likely to be accessed in the near future due to temporal locality. Alternatively, a small window discards past popular objects fast and thus makes replacement decisions with more recently popular objects. In this case Window-LFU behaves in almost the same way as LRU behaves in a processor system, and captures temporal locality. If the cache size C increases, this effect is diminished since a window cannot have size smaller than C (otherwise the cache would not be full).

5 Conclusions

In this work we presented Window-LFU, a simple generalization of LFU. We studied its performance under two sets of assumptions: (i) under the assumption of statistical independence among web requests we proved *analyticaly* that W-LFU can approximate arbitrarily well the performance of Perfect-LFU, and (ii) under the assumption of dependencies among requests, we argued that these dependencies are due to temporal and spatial locality phenomena, by drawing an analogy with processor caches which are designed to take advantage of temporal and spatial locality; under this assumption we showed through *simulation* that W-LFU can perform even better than Perfect-LFU. It is important to note that arguing which of these two (obviously contradictory) assumptions is true is beyond the scope of this work. We do show though that W-LFU is a practical alternative to Perfect-LFU under any of these assumptions, and its performance should be superior to other cache replacement algorithms in cases where Perfect-LFU is superior (for such cases see [2]).

Overall it becomes clear that two parameters of the cache, C and W must be chosen wisely, so that both types of locality (temporal and popularity) are addressed. Perfect-LFU and simple LRU are limited, because each one addresses only one type of locality, while Window-LFU addresses both. In particular, the window size influences performance significantly and should be chosen large **enough**, so that popularity benefits are not lost, and small enough to take advantage of temporal locality. It is also the one cache parameter a cache designer has the most control of. Trace analyses in specific environments are necessary to identify the appropriate window sizes for these environments, in order to achieve the desired cache hit rate.

W-LFU is quite simple. Although this simplicity has the advantage of easy implementation, it leaves great margins for improvement. For example, the window size need not be constant, but may vary according to the changing characteristics of the request stream and one could study how to compute the window size *dynamically*. Another interesting topic for future work are the assumptions (statistical or otherwise) one can make about Web traffic. The assumptions in this or previous work tend to be very strong and, while some of them (e.g., the Zipf-like distribution of requests) have been well established by measurements, others (e.g., the statistical independence between requests) have not been characterized well. Work in this direction, using weaker assumptions where applicable, could lead to better understanding of the Web and subsequently to the design of better caching policies.

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. Cunha, A. Bestavros, and M. Crovella. Characteristics of WWW Client-based Traces. Technical Report BU-CS-95-010, Computer Science Dept., Boston University, July 1995.

[5] G. Karakostas and D. Serpanos. Practical LFU implementation for web caching. Technical Report TR-622-00, Computer Science Department, Princeton University, 2000.

[6] E. Markatos, M. Katevenis, D. Pnevmatikatos, and M. Flouris. Secondary Storage Management for Web Proxies. In *Proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems*, pages 93–104, 1999.

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

[8] D. Serpanos, G. Karakostas, and W. Wolf. Effective Caching of Web Objects Using Zipf's Law. In *Proceedings of IEEE International Conference on Multimedia and Expo (ICME 2000), Vol. 2*, pages 727–730, July, 30 - August, 2 2000.

[9] D. Serpanos and W. Wolf. Caching Web Objects Using Zipf's Law. In *Proceedings of SPIE, Vol. 3527, Technical Conference 3527: Multimedia Storage and Archiving Systems III, Boston, MA, USA*, pages 320–326, November 2-4 1998.