Characterization of Web Reference Behavior Revisited: Evidence for Dichotomized Cache Management

Hyokyung Bahn[1] and Sam H. Noh[2]

[1] Department of Computer Science and Engineering, Ewha Womans University
bahn@ewha.ac.kr
[2] School of Information and Computer Engineering, Hong-Ik University
samhnoh@hongik.ac.kr

Abstract. In this paper, we present the Dichotomized Cache Management (DCM) scheme for Web caches. The motivation of the DCM scheme is discovered by observing the Web reference behavior from the viewpoint of Belady's optimal replacement algorithm. The observation shows that 1) separate allocation of cache space for temporal locality and reference popularity better approximates the optimal algorithm, and 2) the contribution of temporal locality and reference popularity on the performance of caching is dependent on the cache size. With these observations, we devise the DCM scheme that provides a robust framework for on-line detection and allocation of cache space based on the marginal contribution of temporal locality and reference popularity. Trace-driven simulations with actual Web cache logs show that DCM outperforms existing schemes for various performance measures for a wide range of cache configurations.

1 Introduction

Caching mechanisms have been studied extensively to alleviate the speed gap of hierarchical storages in computer systems. More recently, due to the increase in popularity of the Internet, Web caching is becoming increasingly important. To improve the performance of caching, modeling of reference characteristics is essential. However, reference characteristics of Web environments are quite different from those of traditional caching environment because there is a human factor involved in the generation of Web references. That is, it is human that generates Web reference streams, not computer programs as in conventional paging systems. This makes management of a caching system more challenging compared to those of traditional caching systems. In traditional paging systems, *temporal locality* is dominant and hence, page reference streams can be modeled well by the LRU-stack model [1]. In Web caching environment, however, both *temporal locality* and *reference popularity* influence the re-reference likelihood of object references. Hence, an efficient cache management system must take into account both properties.

H.-K. Kahng (Ed.): ICOIN 2003, LNCS 2662, pp. 1018–1027, 2003.

Another important property that must be considered in Web caching is that the retrieval *cost* and *size* of an object to be cached are not necessarily uniform. This property makes the Web cache replacement problem much more complicated. Determining even the off-line optimal cache replacement algorithm is known to be NP-hard [2]. Hence, little effort has been made to examine how an optimal algorithm works and use it as a model for the design of a replacement algorithm. Specifically, even the influence of temporal locality and reference popularity on the re-reference likelihood of a Web object has not been explored quantitatively from the viewpoint of an optimal algorithm. This paper makes an attempt to do so.

To this end, we take the following approach. First, we simplify the problem by assuming that all objects considered by the Web cache are uniform (i.e., identical cost and size), and focus on the analysis of the re-reference likelihood of cached objects. With this assumption, we can readily observe the reference behavior of Web objects (based on a particular replacement algorithm) in contrast to the Belady's optimal algorithm (OPT) [1].

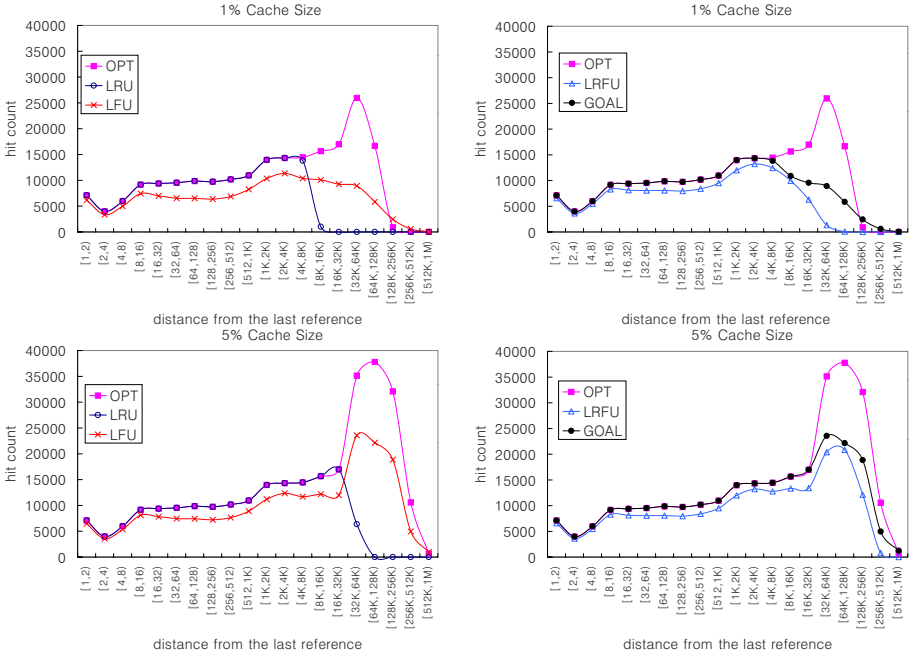
We make the following two observations from this first phase. One, the contribution of temporal locality and reference popularity on the re-reference likelihood of a Web object is dependent on the cache size, and two, partitioning of cache space for management based on temporal locality and reference popularity better approximates the optimal algorithm. Based on these observations, we devise the Dichotomized Cache Management (DCM) scheme that divides the cache space into two to separately exploit temporal locality and reference popularity based on their contribution to the reference probability.

In the second phase of our approach, we extend the DCM, which was devised under the assumption that the cache objects are uniform, to non-uniform objects. This is done via a normalization method that evaluates a cached object based on its estimated reference probability multiplied by the cost of the object per unit size. This results in a normalized assessment of the contribution to the performance measure for non-uniform objects, leading to a fair replacement algorithm.

2 Motivation

The success of any cache replacement algorithm depends heavily on how well the algorithm estimates the re-reference likelihood of cached objects. Hence, we wanted to quantify the impact of temporal locality and reference popularity on the re-reference of Web objects. To do this, we compared the hit distributions of a particular Web reference trace when using the Least Recently Used (LRU) and the Least Frequently Used (LFU)[1] replacement algorithms, and compared them with that when using Belady's off-line optimal (OPT) algorithm, assuming that all objects are uniform.

[1] When not explicitly mentioned otherwise, the LFU we refer to here is the perfect LFU, which does not lose its frequency count even when it is evicted from the cache.



(a) OPT, LRU, and LFU (b) OPT, LRFU, and GOAL

Fig. 1. Hit count distributions for various replacement algorithms

Figure 1(a) shows the comparison results. The x-axis in the figure represents the logical time since last reference, that is, the inter-reference gap, and the y-axis is the total hit count with the given range of inter-reference gap for the OPT, LRU, and LFU algorithms. (Note that the x-axis is in log-scale.) This particular figure is obtained with the public proxy cache logs of Digital Equipment Corporation [6] for cache sizes that are 1% and 5% of the infinite cache size. (Note that "infinite cache size" refers to the total number of distinct objects in the trace as objects are assumed to be of uniform size.)

The characteristics observable from Figure 1(a) is quite different from traditional caches such as the file system buffer cache [12], where repetitious regular reference patterns are generally observed. Observe here that for the OPT algorithm a large number of hits occur for references with large inter-reference gaps (64K-256K). Another observation, though not immediately discernable in this figure due to the log-scale in the x-axis, is that on account of temporal locality, the hit count decreases as the distance from the last reference increases. These same kind of behaviors were observable for not only this DEC trace, but for other traces as well.

Now, let us take a look at how the LRU and LFU algorithms behave. Since the LRU maintains objects based on the ordering of the distance from the last reference, the LRU follows almost exactly the hit counts of the OPT from left

(smaller inter-reference gap) to right (larger inter-reference gap). This shows that temporal locality is an important factor in determining the replacement object, especially when the inter-reference gap is small. Note also that after some point in the x-axis, the number of hits drops rather quickly compared to the OPT. This implies that temporal locality alone is not sufficient to estimate the re-reference likelihood of Web objects.

Taking a look at the LFU algorithm, it follows, but with much less accuracy, the hit counts of the OPT for a wider range of inter-reference gap values. Specifically, the LFU sustains the hit counts of the OPT for large inter-reference gaps fairly well, which was not possible for the LRU. This implies that hits with large inter-reference gaps can be more effectively covered by exploiting reference popularity rather than temporal locality.

Previous research in cache replacement have pointed out that considering both the temporal locality and reference popularity at the same time can help in estimating the re-reference likelihood of objects. For example, the LRFU (Least Recently/Frequently Used) algorithm, which is one of the representative algorithms in this class, exploits temporal locality and reference popularity by a combined recency and frequency value (CRF value) [5]. This and other algorithms [3, 9] all tried to somehow unify the two properties together.

Figure 1(b) shows the hit count distributions of the LRFU. Though not directly comparable in the figure, the hit curve for the LRFU is generally located in between the curves for the LRU and LFU. Even though the LRFU generally estimates reference behavior better than the LRU and LFU at some point or another, it does not fully utilize the strong points observed for the LRU and LFU, that is, for small inter-reference gap values and large inter-reference gap values, respectively. For example, the LRU algorithm with the 1% size cache obtains more hit counts than the LRFU algorithm with the 5% size cache when the inter-reference gap values are smaller than 8K. Hence, when we have a 5% size cache, we can obtain better performance than the LRFU algorithm just by allocating the LRU 1% size and the LRFU the remaining part.

Our goal is then to come up with a scheme that will gain the most from exploiting the two properties, that is, temporal locality and reference popularity. This will result in a hit count distribution represented by GOAL in Figure 1(b), which is the hit counts taken from the higher values of the LRU and LFU algorithms of Figure 1(a). We show in Section 3 how this may be achieved.

3 The Dichotomized Cache Management (DCM) Scheme

In this section, we present the Dichotomized Cache Management (DCM) scheme. Subsection 3.1 presents DCM assuming that objects are uniform. This sets the groundwork for DCM for Web environments. The uniform object assumption is then relaxed in Subsection 3.2 via normalization. In Subsection 3.3, we show how DCM may be efficiently implemented using various data structures.

3.1 Cache Partitioning based on Re-reference Likelihood

In this subsection, we assume that all objects are of the same size and cost. As stated previously in Section 1 our goal is to manage the cache so that the performance of the management scheme reflects the best of the LRU and the LFU. This is to be done by exploiting the temporal locality and the reference popularity characteristics of the references in the cache. Note that the LRU and LFU algorithms are known to be optimal for references that show the temporal locality and reference popularity property, respectively [1].

Based on the observations made previously, we would like to employ the LRU algorithm for those objects that have small inter-reference gaps, which reflect temporal locality, while for those with large inter-reference gaps, we would like to employ the LFU algorithm.

The question then is how to divide up the cache such that objects with small inter-reference gaps retain a certain portion of the cache, while objects with large inter-reference gaps retain the remaining cache space. To achieve this we devise the Dichotomized Cache Management (DCM) scheme that divides the cache space into two based on the marginal contribution of temporal locality and reference popularity to the re-reference likelihood. Marginal contribution refers to the additional hit count that is obtained when one more unit of cache space is allocated, and is calculated as follows.

Temporal Locality: As we apply the LRU algorithm, the expected hit ratio of streams with temporal locality is $HR = \sum_{i=1}^{n} a_i$, where n is the cache size and a_i is the reference probability of position i in the LRU stack. It is possible to measure a_i by where cache hits occur in a ghost LRU stack buffer [11]. The ghost LRU stack buffer is simply a list of dataless object headers where all objects are ordered by their backward distance. Specifically, a_i is measured by the hit counts at stack position i divided by the length of reference streams.

Reference Popularity: As we apply the LFU algorithm for the streams with reference popularity, the expected hit ratio is $HR = \sum_{i=1}^{n} r_i$, where n is the cache size and r_i is the reference probability of an object i (assuming $r_i \geq r_j$, if $i \leq j$). This is reasonable because reference popularity of Web objects is known to be modeled as a Zipf-like distribution, which is an independent reference model [4]. Similarly to the temporal locality case, r_i can be measured by observing where buffer hits occur in the ghost LFU buffer.

3.2 Normalization of the Non-uniformity Factor

Now we extend the DCM algorithm for objects with arbitrary cost and size, which is a more realistic scenario for the Web caching environment. The algorithm first partitions the cache space according to the marginal contributions as explained in Subsection 3.1. (Refer to Subsection 3.3 for implementation detail.) Then, both the temporal locality cache space and reference popularity cache space are managed by the normalization method given in Definition 1.

Definition 1. Cache replacement algorithm A for non-uniform objects *normalizes* the cost and size of an object if it selects as its replacement victim the object i that has the smallest $Value(i)$, which is defined as

$$Value(i) = p(i) \cdot Weight(i)$$

where $p(i)$ denotes the reference probability of object i estimated by algorithm A and $Weight(i) = c_i/s_i$ where c_i and s_i denote the cost and size of object i, respectively.

Since the DCM uses ghost LRU and LFU buffers, $p(i)$ value can be assigned either the a_i or r_i value depending on whether it resides as part of the temporal locality cache space or the reference popularity cache space, where a_i and r_i are the probabilities of position i in the ghost LRU and LFU buffers, respectively. $Weight(i)$ can be defined differently for the performance measure of interest that reflects the primary goal of the given environment. For example, if the goal of caching is in minimizing the network traffic, the corresponding measure would be the *Byte Hit Ratio* and c_i can be defined as the retrieved size for object i when a miss to the object occurs. Similarly, if the goal is in minimizing latency perceived by Web users, the corresponding measure would be the *Delay-Savings Ratio* and c_i can be defined as the retrieval latency for object i.

3.3 Implementation of the Algorithm

To reduce the measurement overhead of ghost buffers, in our implementation of the DCM, the hit counts of the ghost buffers are observed not by individual stack positions but by groups, that is, disjoint intervals of stack positions. In evaluating each object in the cache, the DCM needs $p(i)$, that is the estimated reference probability of object i. As explained in Subsection 3.2, this value should be assigned the a_i and r_i values for the temporal locality area and reference popularity area, respectively. However, since we are grouping for the sake of efficiency, we cannot obtain all a_i and r_i values. Hence, we use an approximation method to obtain the $p(i)$ values for both areas based on previous characterization studies on Web workloads.

A number of characterization studies have shown that the reference probability of a Web object whose inter-reference time equals t is roughly proportional to $1/t$ [4, 8, 9]. Hence, we can assign the $p(i)$ of object i in the temporal locality area $p(i) = b/t_i$, where t_i is the time since last reference to object i and b is a constant. Constant b need not be considered in this case because it is a common multiple of $Value(i)$ for all objects i. As for the reference popularity area, we can use the reference count of object i divided by the length of reference streams as $p(i)$ values because long-term popularity can be modeled as a Zipf-like distribution (which assumes independent reference), and hence this value converges to the stationary probability r_i.

Another issue that should be clarified for the DCM algorithm is the overlapping problem between the temporal locality area and reference popularity area.

Even if the cache space is separately assigned, some objects may be the target of caching in both the temporal locality area and reference popularity area. In this case, only one copy is preserved in the physical cache and the remaining cache area is repeatedly allocated to either of the areas by the order of hit counts of the successive ghost buffer groups.

For caching algorithms to be practical, it is important that the time complexity of the algorithm not be excessive. When a replacement algorithm satisfies the following order preservation property, it can be implemented with a heap data structure whose time complexity for insertion and deletion operations is $O(\log_2 n)$.

Definition 2. *Order preservation property*

If $Value(a) > Value(b)$ holds at time t and neither a nor b have been referenced after t, a replacement algorithm A with *order preservation property* satisfies $Value(a) > Value(b)$ for any $t'(t' > t)$.

In the DCM algorithm, ordering in the reference popularity area satisfies the order preservation property and hence, it can be effectively implemented in $O(\log_2 n)$. However, ordering in the temporal locality area does not satisfy this property because the value of an object depends on the current time. For example, suppose there exists objects x and y such that $Weight(x) = 10$, $p(x) = 1/10$, $Weight(y) = 10000$, and $p(y) = 1/10001$, respectively, at time t. Then, $Value(x) > Value(y)$ holds at time t. However, if both x and y have not been referenced after t, then say, at time $t + 10$, $Value(y)$ becomes larger than $Value(x)$.

Hence, for the temporal locality area, we approximate the original algorithm using multiple list structures as is done by Aggarwal et al. [10]. The algorithm uses separate list structures, such that the i-th list maintains all objects whose cost per unit size value range $[2^{i-1}, 2^i)$ by the LRU order. Whenever free space is needed in the temporal locality area, the DCM compares the values of the least recently used objects in each list and replaces the object with the smallest value among them.

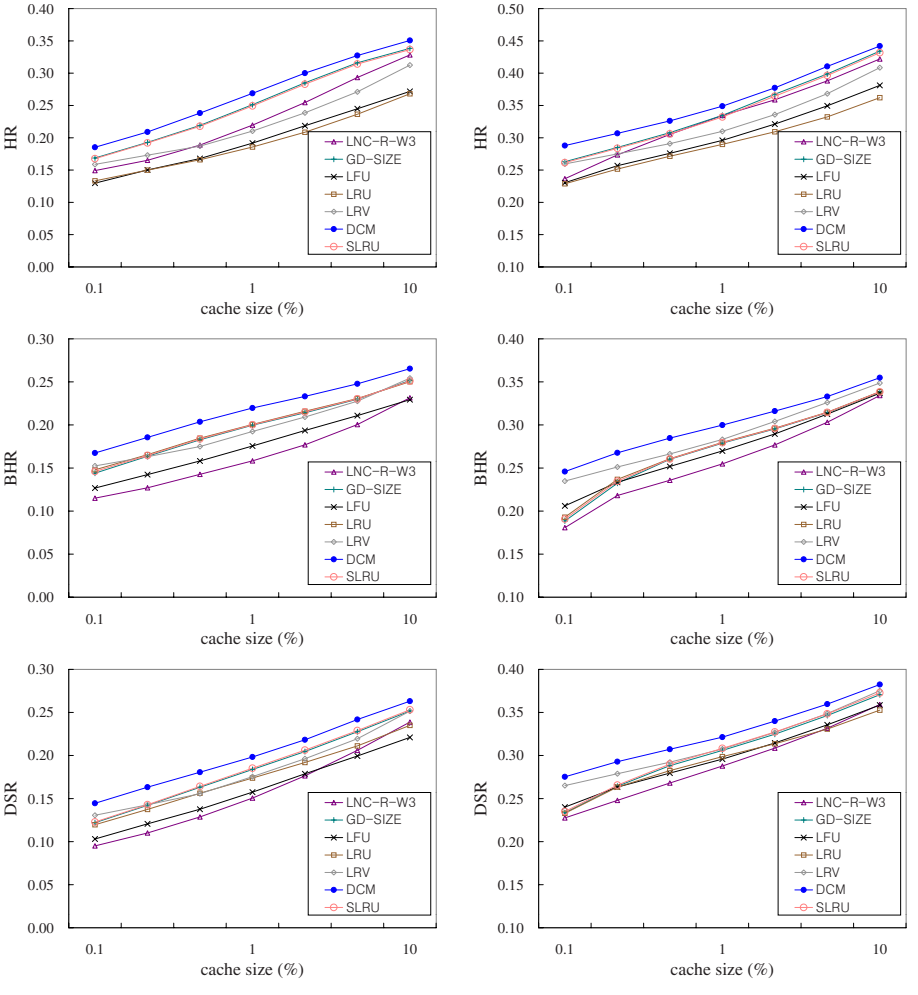
4 Experimental Results

In this section, we discuss the results from trace-driven simulations performed to assess the effectiveness of the DCM scheme. We used two public Web proxy cache traces from National Lab for Applied Network Research (NLANR) [7]. NLANR provides several sanitized proxy cache traces. Among them, we presented the results obtained from the SD and UC traces. (Note that results from other traces are similar to those from the SD and UC traces.) Table 4 shows the characteristics of each trace. In the experiments, we filtered out some requests such as UDP requests, "cgi_bin" requests, and requests whose size is larger than the cache size used in the simulation.

The algorithms used in the simulation are LRU, LFU, LRV [9], GD-SIZE [8], LNC-R-W3 [3], SLRU [10], and DCM. We used three performance measures,

Table 1. Characteristics of traces used in the simulations

Traces	Period	Total Requests	Unique Requests	Total Mbytes	Unique Mbytes
SD	2000.6.28 - 7.4	6913228	3695365	96153.3	56552.3
UC	2000.6.28 - 7.4	1695561	1095161	38241.9	26784.4



(a) NLANR (UC) trace (b)NLANR (SD) trace

Fig. 2. Comparison of the DCM with other algorithms using the NLANR traces

that is the *Hit Ratio* (HR), the *Byte Hit Ratio* (BHR), and the *Delays-Savings Ratio* (DSR). Figures 2 shows the HR, BHR, and DSR of each algorithm as a function of the cache size. The x-axis, which is the cache size in logarithmic

scale, is the size relative to the infinite cache size that is equal to "Total Unique Mbytes" in Table 4.

For all cases, the DCM algorithm performs the best irrespective of the cache size for all performance measures, while the other algorithms give and take amongst each other at particular cache sizes and measures. The performance gain of the DCM is as much as 47.4% (with an average of 21.3%) compared to the LRU algorithm and 30.6% (with an average of 11.9%) compared to the GD-SIZE algorithm for the traces and performance measures we considered. Specifically, the DCM algorithm offers the performance of caches more than twice its size compared with other algorithms for most cases. (Note again, that the x-axis is log-scale.)

Among the algorithms other than DCM, the GD-SIZE and SLRU algorithms consistently show good performance for all performance measures, though the performance gap between the two algorithms and DCM is somewhat wider for the BHR. The performances of GD-SIZE and SLRU are so similar that it is difficult to distinguish between them. Note that the two algorithms are similarly cost-aware and temporal locality based algorithms.

LRV shows good performance for the BHR since it predicts the re-reference likelihood of objects with the prior knowledge of workload characteristics in terms of temporal locality and reference popularity. However, the performance of LRV is inferior when the performance measure is not BHR. The reason is that the LRV algorithm cannot incorporate the cost factor when *cost* in LRV is not proportional to the *size* [9].

The performance of the LNC-R-W3 algorithm in our experiments is not so good as in [3]. We think that the parameters used in the LNC-R-W3 algorithm are perhaps dependent on the traces and performance measures. (Note that we use the same parameter values as in [3].)

5 Conclusion

In this paper, we presented the Dichotomized Cache Management (DCM) scheme that provides a robust framework for on-line detection and allocation of cache space based on the marginal contribution of temporal locality and reference popularity for a given proxy environment and cache size.

To accommodate objects with arbitrary cost and size, the DCM scheme normalizes the reference probability of an object by the cost of the object per unit size. This results in a normalized assessment of the contribution to the performance measure, leading to fair a replacement algorithm. Trace-driven simulations with actual proxy cache logs show that the DCM algorithm outperforms existing algorithms for various performance measures for a wide range of cache configurations. Specifically, DCM offers the performance of caches more than twice its size compared with other algorithms for most cases. We also showed how the algorithm can be effectively implemented as a proxy cache replacement module.

Acknowledgement

This Work was Supported by the Intramural Research Grant of Ewha Womans University.

References

[1] E. G. Coffman and P. J. Denning, *Operating Systems Theory*, Prentice-Hall, Englewood Cliffs, New Jersey, Ch. 6, pp. 241-283, 1973. 1018, 1019, 1022
[2] S. Hosseini-Khayat, "On Optimal Replacement of Non-uniform Cache Objects," *IEEE Trans. Computers*, vol. 49, no. 8, pp. 769-778, 2000. 1019
[3] J. Shim, P. Scheuermann, and R. Vingralek, "Proxy Cache Design: Algorithms, Implementation and Performance," *IEEE Trans. Knowledge and Data Eng.*, vol. 11, no. 4, pp. 549-562, 1999. 1021, 1024, 1026
[4] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web Caching and Zipf-like Distributions: Evidence and Implications," *Proc. IEEE INFOCOM*, pp. 126-134, 1999. 1022, 1023
[5] D. Lee, J. Choi, J. Kim, S. H. Noh, S. L. Min, Y. Cho, and C. Kim, "On the Existence of a Spectrum of Policies that Subsumes the LRU and LFU Policies," *Proc. 1999 ACM SIGMETRICS Conf.*, pp.134-143, 1999. 1021
[6] DEC Proxy Cache Traces, ftp://ftp.digital.com/pub/DEC/traces/. 1020
[7] NLANR Proxy Cache Traces, ftp://ircache.nlanr.net/Traces. 1024
[8] P. Cao and S. Irani, "Cost-Aware WWW Proxy Caching Algorithms," *Proc. 1st USENIX Symp. Internet Technology and Systems*, pp. 193-206, 1997. 1023, 1024
[9] L. Rizzo and L. Vicisano, "Replacement Policies for a Proxy Cache," *IEEE/ACM Trans. Networking*, vol. 8, no. 2, pp. 158-170, 2000. 1021, 1023, 1024, 1026
[10] C. Aggarwal, J. Wolf, and P. Yu, "Caching on the World Wide Web," *IEEE Trans. Knowledge and Data Eng.*, vol. 11, no. 1, pp.94-107, 1999. 1024
[11] J. Choi, S. H. Noh, S. L. Min and Y. Cho, "Towards Application/File-Level Characterization of Block References: A Case for Fine-Grained Buffer Management," *Proc. 2000 ACM SIGMETRICS Conf.*, pp. 286-295, 2000. 1022
[12] J. M. Kim, J. Choi, J. Kim, S. H. Noh, S. L. Min, Y. Cho and C. S. Kim, "A Low-Overhead, High-Performance Unified Buffer Management Scheme that Exploits Sequential and Looping References," *Proc. 4th Symp. Operating Systems Design and Implementation (OSDI 2000)*, San Diego, CA, pp. 119-134, 2000. 1020