Disk Cache Replacement Algorithm for Storage Resource Managers in Data Grids (Extended Abstract)

Ekow Otoo, Frank Olken and **Arie Shoshani**
Lawrence Berkeley National Laboratory
1 Cyclotron Road, MS: 50B-3238
University of California
Berkeley, CA 94720

Abstract

We address the problem of cache replacement policies for Storage Resource Managers (SRM) used in Data Grids. An SRM has a disk storage of bounded capacity that retains some N objects. A replacement policy is applied to determine which object in the cache needs to be evicted when space is needed. We present an optimal utility function for ranking the candidate objects for eviction and then describe an efficient algorithm for computing the replacement policy based on this function. This computation takes time $O(\log N)$. We compare our policy with traditional replacement policies such as Least Frequently Used (LFU), Least Recently Used (LRU), LRU-K, etc., using simulations of both synthetic and real workloads of file accesses to tertiary storage. Our simulations of replacement policies account for delays in cache space reservation, data transfer and processing. The results obtained show that our proposed method is the most cost effective cache replacement policy for Storage Resource Managers (SRM).

Keywords and Phrases: file caching; cache replacement algorithm; trace-driven simulation; data staging; storage resource management.

Disk Cache Replacement Algorithm for Storage Resource Managers in Data Grids
(Extended Abstract)

Abstract

We address the problem of cache replacement policies for Storage Resource Managers (SRM) used in Data Grids. An SRM has a disk storage of bounded capacity that retains some N objects. A replacement policy is applied to determine which object in the cache needs to be evicted when space is needed. We present an optimal utility function for ranking the candidate objects for eviction and then describe an efficient algorithm for computing the replacement policy based on this function. This computation takes time $O(\log N)$. We compare our policy with traditional replacement policies such as Least Frequently Used (LFU), Least Recently Used (LRU), LRU-K, etc., using simulations of both synthetic and real workloads of file accesses to tertiary storage. Our simulations of replacement policies account for delays in cache space reservation, data transfer and processing. The results obtained show that our proposed method is the most cost effective cache replacement policy for Storage Resource Managers (SRM).

Keywords and Phrases: file caching; cache replacement algorithm; trace-driven simulation; data staging; storage resource management.

1 Introduction

A storage resource manager (SRM) [12], in the context of the data-grid infrastructure [3, 8], is essentially a middle-ware component that facilitates the sharing of data and storage resources. A key component of its functions is the management of a large capacity disk cache that it maintains. We address the problem of cache replacement policies for Storage Resource Managers (SRMs), used in data-grids. An SRM, described subsequently, maintains a large capacity disk for staging files and objects of varying sizes that are read from or written to Mass Storage Systems (MSS) and/or other remote sites. Its role in the data-grid is similar to that of a proxy server or a reverse proxy server in the World Wide Web. Although SRMs differ in many respects from proxy and reverse proxy servers, they share some common service functionalities such as caching of files or objects. One difference between caching in an SRM and caching in a web-server is that SRMs typically deal with batched requests of files or objects that are very large and incur significantly long delays in transferring and processing them. We address file/object replacement policies in SRMs, taking into account the latency delays in retrieving, transferring and processing of the objects.

The *Grid* [5] may be described as a network of geographically distributed platforms of high performance heterogeneous computational nodes and data storage resources. Computational platforms range from super-computers, and large scale cluster-computing farms to desktop workstations. Storage resources range from mass storage systems, e.g., tertiary storage system, high performance storage systems (HPSS), RAID farms and network attached storage(NAS), to workstation group servers. A *data-grid* is a network of geographically dispersed nodes of data and storage resource that is used to efficiently support data intensive applications through middleware services. A storage resource is accessible by a user, either remotely or locally for creating, destroying, reading, writing and manipulating *file or object instances.* For the purposes of our discussion in this paper, we will use the terms *file* and *object* interchangeably.

The dataset derived from some scientific experiments or observations is maintained in multiple storage resources that are distributed over wide area networks. The experiments of a particular project, carried out over a number of years, result in the generation of datasets in the order of hundreds of terabytes to a few petabytes. These datasets are typically maintained in mass storage systems or tertiary storage systems. Client applications that run on workstations or workstations clusters and super-computers make requests for files/objects that reside on direct attached storage (DAS), a tape storage system or even mass storage system at a remote site. We will use the term *mass storage system or MSS* for all large capacity storage systems, except DAS, for the simple reason that they can potentially incur very long latency in retrieving the objects resident on them.

Two significant decisions govern the operation of an SRM. Unlike web proxy severs, each of the requests that arrive at an SRM can be for hundreds or thousands of objects at the same time. As a result, an SRM generally queues the requests and makes decisions as to which objects are to be retrieved into its disk cache. When a decision is made to cache a file it determines which of the objects currently in the cache may have to be evicted to create space for the incoming object. The latter decision is generally referred to as *a cache replacement policy* and it is the subject of this paper in the context of storage resource managers in the data-grid.

The performance measures of replacement policies are typically expressed by two metrics: the *hit ratio* and the *byte hit ratio.* These metrics are briefly defined in sub-section 4.2. We introduce a third measure of goodness which we term the *average cost per reference.* Depending on the context of the usage of a cache and the factors that are of interest to users, an optimal

replacement policy either maximizes the hit ratio, maximizes the byte-hit ratio, minimizes the average cost per cache reference or some combination of these.

We make three main contributions to disk cache replacement policies in this paper. First, we give an algorithm for a cache replacement policy that computes the minimum utility function in time $O(logN)$. The utility function $\phi_i(t)$ is defined as

$$\phi_i(t) = \frac{K_i(t)}{(t - t_{-K_i})} * \frac{c_i(t)}{s_i};$$

where, for each object i, s_i is the size of the object, $K_i(t)$ is the number of references, up to a maximum of K, made to the object within the time interval $[t - t_{-K_i}]$, t_{-K_i} is the time of the K_i backward reference and $c_i(t)$ is the cost of retrieving the object from its source into the cache at the time t. We call the cache replacement policy, that is based on the above utility function, the *least cost beneficial based on the K backward references* or the *LCB-K* for short. Second, we define a new measure for cache replacement policies which we term *"average cost per reference"*. Using both synthetic and real workload, we show that a policy that replaces the object in the cache with the minimum utility function as defined above gives the minimum average cost per reference compared with *random (RND)*, *least frequently used (LFU)* and *least recently used (LRU)* replacement policies. The LCB-K replacement policy does not necessarily maximize either the hit ratio or the byte hit ratio. Third, we present a cache replacement policy simulation model that takes into account, the latency delays at the source, file transfer delays and processing time delays of the objects. An object that is held in the cache during processing and marked as non-evictable is said to be *pinned*. We are unaware of any simulation comparisons of cache replacement policies that consider pinning delays of objects in the cache.

2 Configuration and Related Works

The use of an SRM helps to ameliorate the latency experienced when users request objects with long total retrieval times. Its role is to stage in its cache, a single copy of an object that is requested and then use it to service multiple requests. In some respect, its usage is similar to that of a *proxy server* or a *reverse proxy server*, except that SRMs deal with transfers of objects that are of the order of gigabytes in size. Figure 1 depicts the positional role of an SRM within

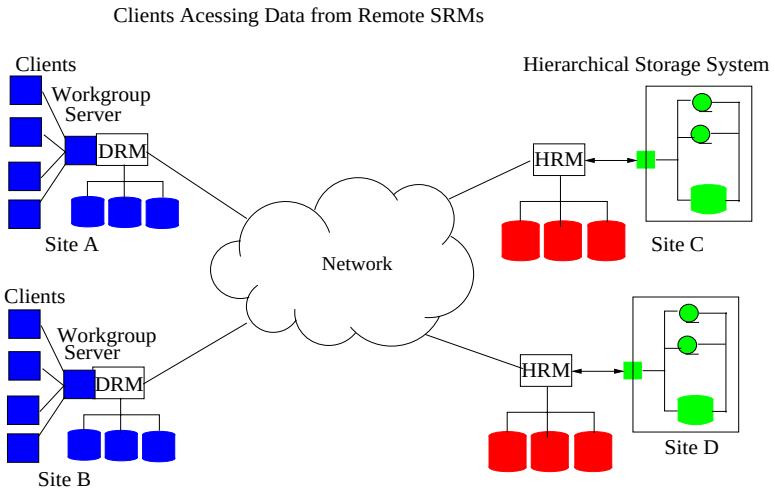


Figure 1: Use of a Storage Resource Manager in a Data-Grid

a data-grid.

In environments that deal with object transfers from archival tape storage to clients at remote sites, the idea of holding files in disk storage at some site along the transfer path and then subsequently forwarding it to the client has been called *Data Staging* [13, 14]. Data staging, in data intensive wide area networks is a form of shared disk caching and a good replacement policy should be applicable to it as well. Another related area where the issues we address in this paper have applicability is in *Web-Caching* [1, 2, 4, 6]. Caching techniques are effective where reference streams to objects observe two basic principles:

- *Locality of Reference:* An object that is referenced and read into a cache is often referenced many times, by the same user, over a very short period.
- *Shared Access to Objects:* The same object, after it is read into a cache, is also referenced by different users.

Some earlier works on file caching in distributed system and the staging of files on tertiary storage onto disk, have been presented in [7, 9, 13, 14, 15, 17]. Recent studies on caching have focused more on web-caching [1, 2, 4, 11, 16]. Cao and Irani [2] present a relative comparison of various cache replacement policies that have been proposed for web-caching. Their work discusses some of the merits and concerns of replacements policies such as Least Frequently Used (LFU), Least Recently Used (LRU), Size, etc. They propose a replacement policy for

web-caching called the *Greedy-Dual-Size (GDS)* [2]. It is a variant of the replacement policy termed *Greedy-Dual (GD)* [17], that was originally proposed for main memory caching of fixed size pages. It is also perceived as a generalization of the LRU when there is some variability in the cost of reading pages from secondary storage.

2.1 Differences between Caching in SRMs and Web-Caching

In principle the objects in web-caching can be of any type and of varying sizes. However web-caching in proxy servers are realistic only for documents, images, video clips and objects of moderate size of less than a few megabytes. On the other hand, the files and objects in SRMs have sizes of the order of hundreds of megabytes to a few gigabytes. Some of the differences between caching in SRMs and web-caching are summarized in Table 1.

3 Optimal Utility Function

The basic idea of our page replacement policy is to define a utility function $\phi_i()$ for each object i in the set C of cached objects. An object i is of size s_i and has retrieval cost $c_{i,r}(t)$ from site r that varies with time according to when and where it is fetched from. Note that in the environment of the data-grid there could be replicas of the same object at different sites r. We will denote cost simply by $c_i(t)$, with the understanding that this would be the minimum cost of retrieving the object into the cache. Let the total number of objects in the cache be N, and let the number of unpinned objects be N'. At each instant in time t, when we need to acquire space of size s_j for an object j that is not in cache, we order all the unpinned objects in non-decreasing order of their utility functions and evict the first m objects with the lowest values of $\phi_i(t)$, whose sizes sum up to or just exceed s_j. We always assume that the sizes of the cached objects are relatively small compared to the total size S, of the cache.

To see why this holds consider a request stream to an SRM denoted by $r_1, r_2, r_3, \ldots, r_i \ldots$, as a sequence of random variables with common stationary probability distribution $p_1(t), p_2(t)$, $p_3(t), \ldots, p_n(t)$, where the probability that r_i references the object j is $p_j(t)$, i.e., $Prob(r_i = j) = p_j(t)$ for all $i \geq 1$. Our objective then is to minimize the total cost of retrieving the objects that are not found in the cache but need to be fetched. Let the set of objects retrieved into the cache be C, where $C \subseteq I = \{1, 2, \ldots, n\}$. Then an optimal cache replacement retains the set C

Characteristic Property	Web Caching	Disk Caching in SRMs
Object Size	Variable size objects of the order of a few megabytes	Variable size objects of the order of a few Gigabytes
Cache Size	In the order of tens to hundreds of gigabyte	In the order of hundreds of gigabyte to tens of terabytes
Source Latency	A few milliseconds to seconds	In milliseconds to minutes
Object Transfer Time	In milliseconds to a few minutes	In seconds up to a few hours
Duration of Object Reference	Instantaneous	In seconds up to a few minutes
Caching Requirement	Optional	Mandatory
Batched requests	Typically one request references one object but may involve a few additional linked objects.	May involve hundreds of objects in one request.
Bundle Constraint	Only one object is referenced per request.	Simultaneousaccesses of multiple objects in the same request.
Cache Consistency	Cognizant of modified documents	Predominantly Read-Only and ignores consideration of cache coherence
Network bandwidth requirement	Standard Internet	High speed gigabit networks

Table 1: Summary of Differences between Caching in SRMs and Web-Caching

such that

$$\sum_{i \in (I-C)} p_i(t) * c_i(t); \tag{1}$$

is minimized subject to

$$\sum_{i \in C} s_i \ \leq S. \tag{2}$$

Since we assume that the sizes of the cached objects are relatively small compared to the total size S of the cache, the amount of space left after caching the maximum number of objects is negligible. The solution space may be restricted to the set C satisfying $\sum_{i \in C} s_i \ = S$. We can

restate the problem then as

$$maximize \sum_{i \in C} p_i(t) * c_i(t); \tag{3}$$

subject to

$$\sum_{i \in C} s_i = S. \tag{4}$$

Under the above assumption the problem statement expressed by 3 and 4 is equivalent to that of the *fractional Knapsack* problem. An optimal solution is given by a simple greedy algorithm as follows. We rank the items $i \in I$ in non-increasing order of $p_i(t) * c_i(t)/s_i$ and then insert into the cache the items beginning from the first to the last until either all items are inserted or the constraint (2) is violated.

3.1 Cost Beneficial Cache Replacement Policy

The cache replacement policy we propose, is based on the optimal utility function defined in the preceeding subsection. Restated differently, we have that whenever some object in the cache needs to be evicted at some instant in time t, the eviction candidate is the one that has the minimal utility function $\phi_i(t)$ given by

$$\phi_i(t) = \frac{p_i(t) * c_i(t)}{s_i} \tag{5}$$

Similar conclusions are given in [9, 11] but under different assumptions. The problems studied for which their results are derived are different. The utility function, as expressed by equation (5), is impossible to apply since we do not know the probabilities and further these probabilities are not stationary. Under the assumption that the references to the objects are independent, the arrival rate of references to an individual object i can be approximated by a Poisson distribution with parameter λ_i and the probability term in equation (5) can be replaced by

$$p_i(t) = \frac{\lambda_i(t)}{\sum\limits_{1 \leq j \leq n} \lambda_j(t)}.$$

Since the replacement decision is based only on the relative rankings of $p_i(t)$, we can rewrite equation (5) as

$$\phi_i(t) = \lambda_i(t) * \frac{c_i(t)}{s_i}. \tag{6}$$

To estimate the values of $\lambda_i(t)$ we utilize the concepts used in the development of the *Least Recently Used Based on on the K^{th} backward reference (or LRU-K),* page replacement policy. In the *LRU-K* [10] the times of the last $K_i(t)$ references to the object i are retained. At time t, let $K_i(t)$ denote the count of the last references made to i up to a maximum of K, $1 \leq K_i(t) \leq K$. Let the time of the backward K_i^{th} reference be denoted by t_{-K_i}. Then we can approximate the rate of arrival by

$$\lambda_i(t) \;=\; \frac{K_i(t)}{t - t_{-K_i}}$$

Since the cost of the future retrieval is also not known, we utilize a best effort estimate, denoted by $c_i'(t)$, by deriving it from the last K retrievals. Note that before an object becomes a candidate for eviction, at least one access for the object must have been made to cache it. Our eviction candidate then becomes that object with the minimum value of $\phi_i(t)$ where

$$\phi_i'(t) = \frac{K_i(t)}{t - t_{-K_i}} * \frac{c_i'(t)}{s_i} \tag{7}$$

Our disk cache replacement policy, based on the equation (7) is referred to as a *Least Cost Beneficial* replacement policy based on at most K backward references or *LCB-K* for short. Using equation (7), the problem of implementing an efficient algorithm for quickly ranking the objects in cache is still non-trivial.

4 The LCB-K Replacement Algorithm

The LCB-K algorithm is based on an approximation of the optimal utility function given by equation (7). The algorithm makes use of three principal data structures whose detailed implementations are satisfied with equivalent container types of the Standard Template Library (STL).

Search Tree T_s: A search tree whose nodes hold the identifiers of objects that have been

accessed. The nodes have fields for values of $K_i(j), t_i(j), c_i(k), 1 \leq j \leq K, 1 \leq k \leq K$. A node also maintains a status indicator that specifies whether the object is in cache and pinned, is in cache and not pinned or is considered evicted out of the cache.

A Vector $C_p[]$: This is a vector container whose elements hold information of those objects that are pinned. In particular the elements of the container have fields for the count of the number of pins held on the object.

A Vector $C_u[]$: This is a vector container of m heap-like structures where m is the number of partitions of the values c_i'/s_i. Each element $C_u[k], 1 \leq k \leq m$, is a heap-like data structure. The root node of $C_u[k]$ holds the minimum valued element of all nodes in the heap. The vector forms a tournament of heaps for selecting an actual candidate for removal.

4.1 The Simulation Model

In simulating cache replacement polices in SRMs, one needs to account specifically for delays three types of delays:

i the latency incurred at the originating source of the object;

ii the transfer delay in reading the object into the cache, and

iii the holding or pinning delay incurred while a user processes the object after it has been cached. This may simply involve transfering the object into the users workspace.

Simulations of cache replacement policies that have been studied for virtual memory management, database buffering, tertiary storage file staging and web-caching do not normally address these concerns. We are not aware of any simulations of cache replacement policies that account for the additional delays in the processing objects after they are cached.

The simulation of the replacement policies is done as a discrete event simulation and identifies the occurrences of five distinct events for an object request and acts appropriately. The distinct event times of each requests r_i are *Arrival_Time(r_i)*, *Start_Caching(r_i)*, *End_Caching(r_i)*, *Start_Processing(r_i)* and *End_Processing(r_i)*. An event object *(r_i = evtObj)*, is constructed upon an arrival of a request and inserted into an event queue denoted by *evtQueue*. An *evtObj* has a component *eventType* that identifies which event type it is and the time, denoted by *schdTime*, at which the event is to occur. The *evtQueue* is priority queue.

The actions taken upon the occurrence of these events are implied by their names. We give the semantic action of only one to illustrate the idea. Suppose an event object *evtObj*, is popped up from the top of the event queue. If *eventType(evtObj) = "Start_Caching"* then the *eventType(evtObj)* is set to *"End_Caching"* and the *schdTime(evtObj)* is set to the scheduled completion time for retrieving the object into cache. The event object *evtObj*, is then reinserted into the priority queue *evtQueue*.

The simulator is driven by the arrival of file requests. Let the time of arrival of a request r_i for an object j, be t_0 and let the root object of a non-empty *evtQueue* by denoted by *evtQueue(Root)* If t_0 is greater than or equal to *schdTime(evtQueue(Root))* then *evtQueue(Root)* is removed and assigned to *evtObj*. The simulator acts appropriately according to the event type of *evtObj* that has been scheduled. Otherwise if t_0 is less than *schdTime(evtQueue(Root))* then a series of actions are carried out that eventually ensures that the object is brought into the cache. In particular, if the object of size s_i is not in cache then the simulator runs the cache replacement algorithm on the vector container $C_u[]$ to free enough space for the incoming object.

It is possible to encounter the situation where the simulator is unable to free enough cache space for an incoming object. This can easily occur particularly if either there is not enough cache space allocated to handle the workload or all objects in the cache are pinned. We introduce a new performance metric in evaluating cache replacement policies which we refer to as *"availability"*. We define this formally in the next subsection.

4.2 Performance Metrics

The traditional performance metrics that have been used in general to measure the effectiveness of a cache replacement policy are the *hit ratio* and the *byte hit ratio*. Given a request stream (or workload), the hit ratio is defined as the ratio of the requests that find their objects in cache (i.e., hits) to the total number of requests. A byte hit ratio is the ratio of the volume of data (in bytes) hit to the volume of data requested.

In any case these measures provide some insight into the improvement in response times and savings in bandwidth utilization due to caching. Hit ratio gives the relative savings as a count of the number of objects hit, while the byte hit ratio measures the relative savings in the volume of data prevented from being transferred. The measure of byte hit ratio translates to savings in bandwidth usage and consequently improved response time. None of these measures accounts

for the latency at the data source. For example when the data source is from a robotic tape device where the delay can sometimes be comparable to the data transfer time, the measure of byte hit ratio does not reflect it. We introduce a third measure which we call the *"Cost Per Reference"*. This is defined as the ratio of the total cost of retrievals to the total number of references. This measures the relative savings as the average cost avoided per reference in retrieving and transferring the object into cache. A lower value of this measure indicates an effective replacement policy. A fourth measure that is relevant to object caching on disk as opposed to main memory caching is *"availability"*. Given a workload with defined arrival times of each request, we say a request is satisfied if either the object is already in cache or enough disk space can be freed to allow the object to be cached. We define *availability* as the ratio of the number of requests that were immediately satisfied to the total number of requests made in a workload. The minimum cache size for which availability becomes 100% indicates the minimum cache required to satisfy the given workload. We measure all four performance metrics in our experimental studies using synthetic and real workloads.

5 Performance Comparison of Some Replacement Policies

We compared the performance of a number of replacement policies, namely LFU, RND, LRU, LRU-K, MIT-K (i.e., variant of LRU-K) and LCB-K, to evaluate their relative performance under the different metrics of *hit ratio*, *byte hit ratio*, *average cost per reference* and *availability*. We also compared the average times taken by each policy to free enough space for an incoming object. The simulations used both synthetic workload and a real workload of a three month file caching activities of the mass storage system, JasMINE, at Jefferson's National Accelerator Facility (JLab).

The file sizes ranged from about 1.0 to 2.1 gigabyte and the time scales are in the order of minutes. Arrivals in these workload are batched, in the sense that the same request, with the same submit time can be associated with multiple file identifiers. However the workload has very low locality of reference.

Using the time scale and file sizes of the workload from JLab as a guide, we generated a synthetic workload based on a Poisson inter-arrival time with mean 90 seconds. The file sizes, in bytes, are uniformly distributed between 500,000,000 and 2,147,000,000. The entire period

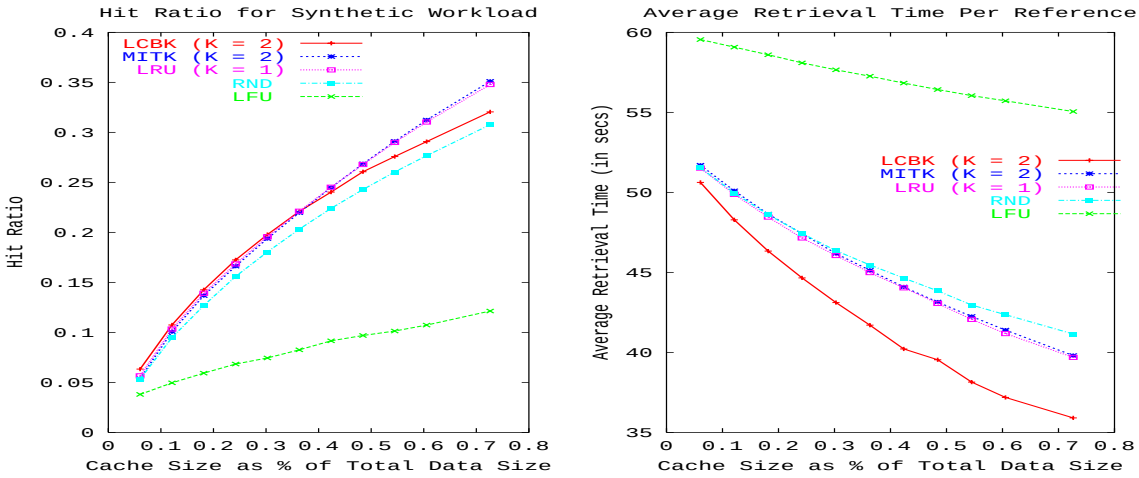
of request generation is broken into random intervals and we inject locality of reference using the 80-20 rule, i.e., within each interval of the request stream 80% of the requests are made to 20% of the files. The length of an interval is uniformly distributed between 1% and 5% of the generated workload.

5.1 Experimental Results

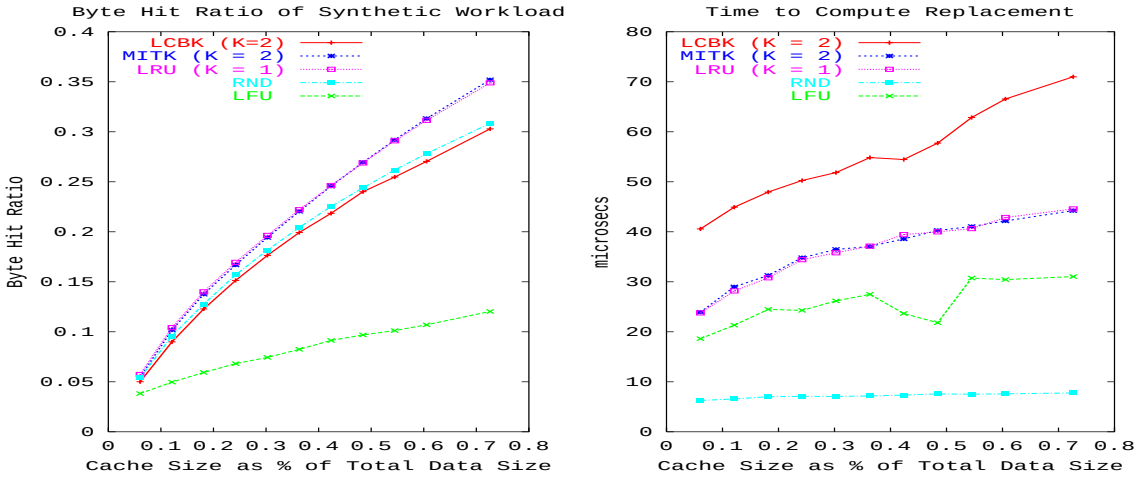
Figure 2a shows the graphs of the hit ratio versus the cache size, expressed as a percentage of size of the total workload, for the respective policies of LCB-K, MIT-K, LRU, RND and LFU. The corresponding graphs of the average cost per reference and byte hit ratio are shown in Figure 2b and Figure 2c respectively. The average time taken to free sufficient space to fetch a requested file is shown in Figure 2d. There is no significant difference between the policies in the values of the hit ratio except for LFU which is clearly the worst. However in Figure 2d LCB-K gives the least average cost per reference with LFU showing the worst performance.

The performance graphs using the workload from JLab are shown in Figure 3. Figures 3a, 3b and 3c follow the same trends as in the synthetic workload except that the average cost per reference, increases with increasing cache size between 1% to 4.2% of the workload. This is due to the fact that the simulator discards requests that can not be satisfied in the sense described in sub-section 4.2. This is the range of cache sizes for which the availability is less than 100%. The availability graph is shown in Figure 3d. A key observation here is that the availability is not dependent on replacement policy used but rather it is dependent on the amount of cache size allocated.

In the development of LRU-K, the authors suggested that values of K = 2 or 3 is sufficient and recommended the use of K = 2. We verified this fact in our simulation to see if the performance measures for LCB-K and MIT-K would be radically different for $K = 3$ and $K = 5$ than for $K = 2$. The graphs of Figure 4, using the synthetic workload, support the suggestion made that the use of $K \leq 3$ is sufficient. Note that even for the measure of average cost per reference shown in Figure 4b, the graphs separate out clearly into two groups; the group for LCB-K and MIT-K for $K = 2, 3, 5$. The group for LCB-K clearly gives the better performance measures. Within each group, there is very little difference in the plots.



(a) (b)



(c) (d)

Figure 2: Graphs of Performance Metrics for Synthetic Workload

6 Conclusion and Future Work

Disk file caching, in storage resource managers, has some characteristic features that make it different from caching in other domains such as virtual memory, database page buffering and web-caching. In particular, disk file caching in SRMs involve variable size objects that are very

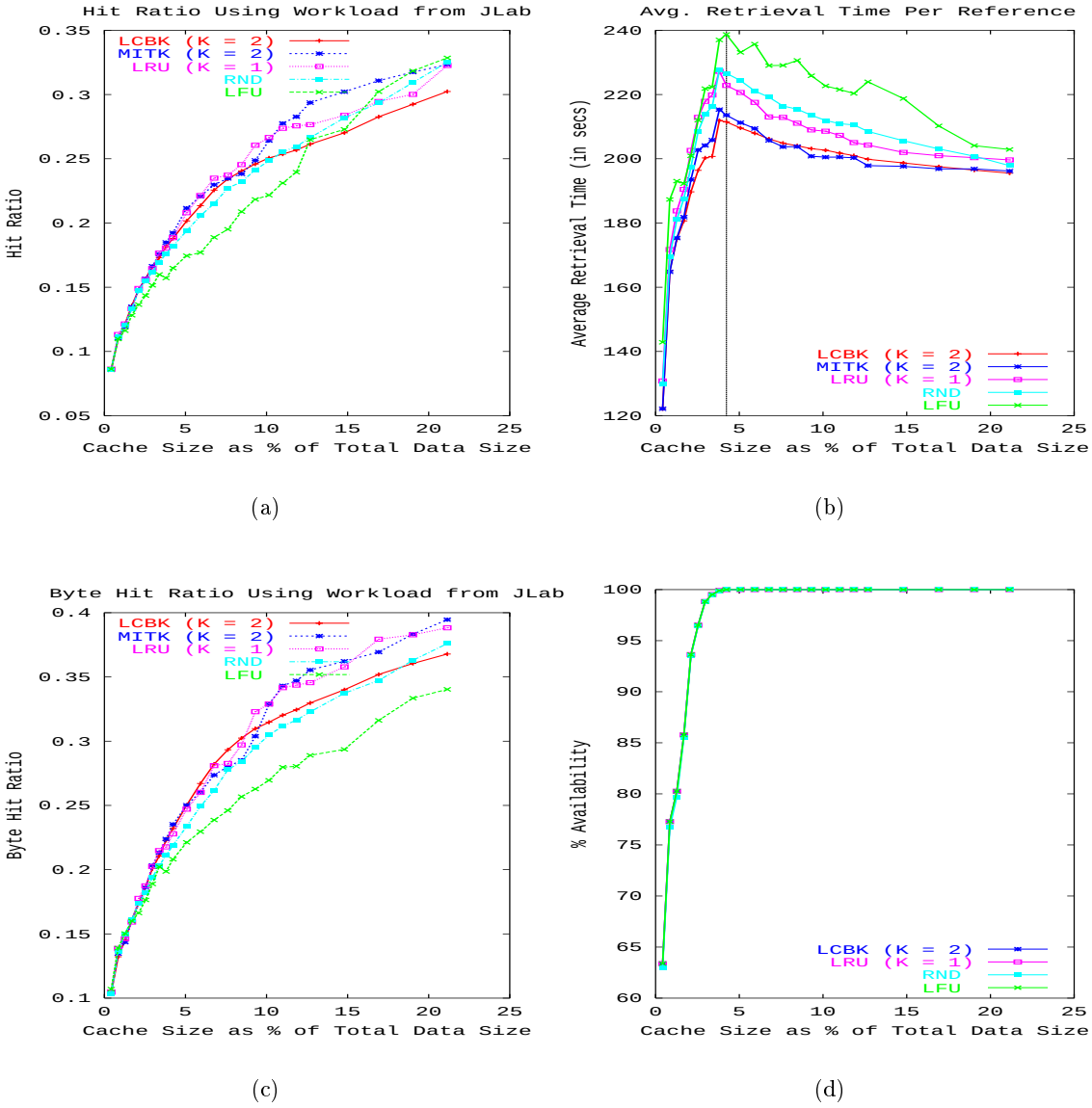
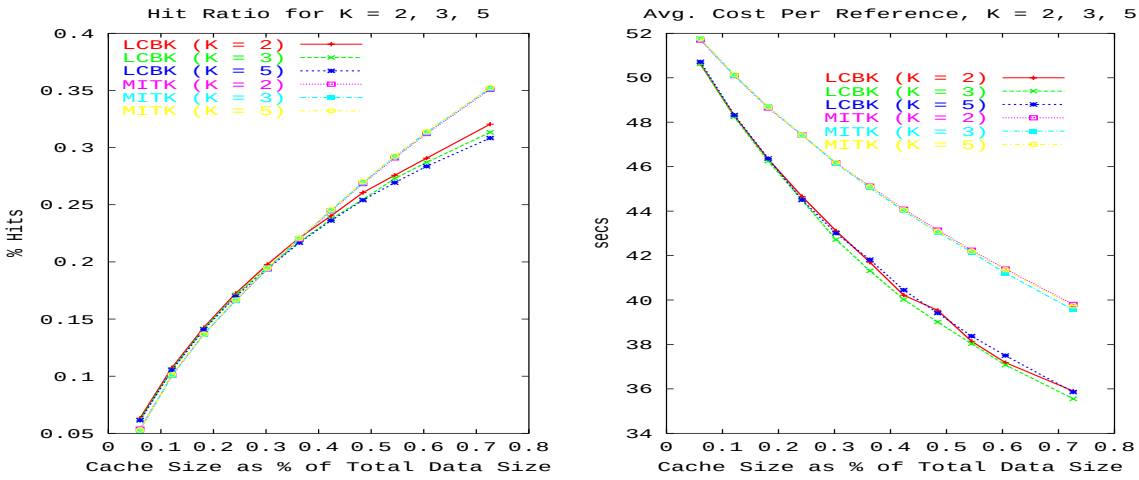
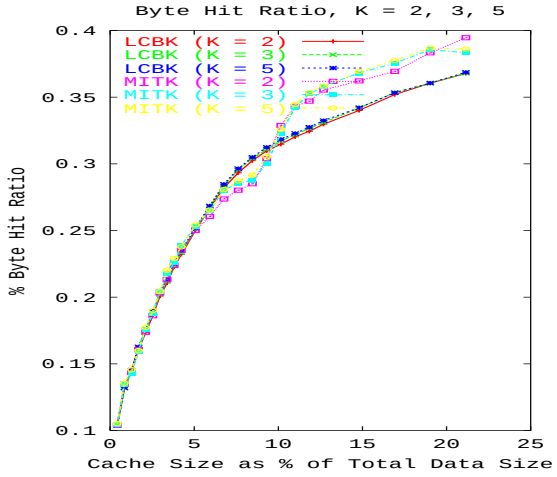


Figure 3: Graphs of Performance Metrics of Real Workload from Jefferson's National Accelerator Facility (JLab)



(a) (b)



(c)

Figure 4: Graphs of Performance Metrics of Synthetic Workload with K = 2, 3, and 5

large. The delays caused by source latency, object transfers and processing of the objects once in the cache are all significant.

We have derived an optimal utility function for determining the objects to be evicted from the cache of an SRM when space is needed. We have presented an efficient method for evaluating the replacement policy based on the optimality condition using a tournament of heaps. Unlike traditional simulations of cache replacement policies we also presented a realistic simulation model that accounts for the delays in processing objects in the cache. Using the simulation model, we compared the replacement policies of RND, LFU, LRU and MIT-K under the performance metrics of hit ratio, byte hit ratio and average cost per reference for both synthetic and actual workloads.

We conclude that average cost per reference is the most realistic performance metrics for evaluating disk cache replacement policies for storage resource managers. Under this measure of performance comparison, the least cost beneficial replacement policy gives the best result of the policies compared.

Acknowledgment

We would like to express our sincere gratitude to Andy Kowalski of Jefferson's National Accelerator Facility and Don Petravick of Fermi National Laboratory for providing us the workloads used in our simulation runs. This work is supported by the Director, Office of Laboratory Policy and Infrastructure Management of the U. S. Department of Energy under Contract No. DE-AC03-76SF00098. This research used resources of the National Energy Research Scientific Computing (NERSC), which is supported by the Office of Science of the U.S. Department of Energy.

References

[1] C. C. Aggarwal and P. S. Yu. On disk caching of web objects in proxy servers. In *Proc. Int'l. Conf. Info and Knowledge Management, CIKM'97*, pages 238 – 245, Las Vegas, Nevada, 1997.

[2] Pei Cao and Sandy Irani. Cost-aware WWW proxy caching algorithms. In *USENIX Symposium on Internet Technologies and Systems*, 1997.

[3] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. *J. Network and Computer Applications*, 23(3):187 – 200, 2000.

[4] S. Dar, M. Franklin, B. Jonsson, D. Srivastava, and M. Tan. Semantic data caching and replacement. In *Proc. VLDB Conf.*, pages 330 – 341, Bombay, India, Sept. 1996.

[5] I. Foster and C. Kesselman, editors. *The GRID: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publ., San Fracisco, 1999.

[6] D. Guerrero. Caching the web, part 1. *Linux Journal*, 57, Jan. 1999.

[7] U. Hahn, W. Dilling, and D. Kaletta. Adaptive replacement algorithm for disk caches in hsm systems. In *16 Int'l. Symp on Mass Storage Syst.*, pages 128 – 140, San Diego, California, Mar. 15-18 1999.

[8] W. Hoschek, J. Jaen-Martinez, A. Samar, H. Stockinger, and K. Stockinger. Data management in an international data grid project. In *Proc. 1st IEEE/ACM Int'l. Workshop on Grid Computing*, India, 2000.

[9] F. A Olken. Hopt: A myopic version of the stochopt automatic file migration. In *Proc. ACM SIGMETRIC Conf. on Mesurement and Modelling of Comput. Syst.*, pages 39 – 43, Minneapolis, Minnesota, Aug. 1983.

[10] E. J. O'Neil, P. E. O'Neil, and G. weikum. The lru-k page replacement algorithm for database buffering. In *Proc. ACM SIGMOD'93: Int'l. Conf. on Mgmnt. of Data*, pages 297 – 306, Washington, DC, May. 1993.

[11] P. Scheuermann, J. Shim, and R. Vingralek. Watchman: A data warehouse intelligent cache manager. In *Proc. 22nd VLDB Conference*, pages 51 – 62, Bombay, India, Sept. 1996.

[12] A. Shoshani, A. Sim, and J. Gu. Storage resource managers: Middleware components for grid storage, Apr. 15 - 18 2002.

[13] A. J. Smith. Analysis of long term file reference patterns for application to file migration algorithms. *IEEE Trans. on Soft. Eng.*, SE-7(4):403–417, Jul. 1981.

[14] M. Tan, M.D. Theys, H.J. Siegel, N.B. Beck, and M. Jurczyk. A mathematical model, heuristic, and simulation study for a basic data staging problem in a heterogeneous networking environment. In *Proc. of the 7th Hetero. Comput. Workshop*, pages 115–129, Orlando, Florida, Mar. 1998.

[15] T. Theodore Johnson and E. L. Miller. Performance measurements of tertiary storage devices. In A. Gupta, O. Shmueli, and J. Widom, editors, *VLDB'98, Proc. 24th Int'l. Conf. on Very Large Data Bases*, pages 50 – 61. Morgan Kaufmann, Aug. 24-27 1998.

[16] S. Williams, M Abrams, C. Stanbridge, G. Abdulla, and E. Fox. Removal policies in network caches for world-wide-web documents. In *Proc. of ACM SigComm Conf.*, 1999.

[17] N. Young. On-line file caching. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1998.