Accurate Modeling of Cache Replacement Policies in a Data Grid

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

Abstract

Caching techniques have been used to improve the performance gap of storage hierarchies in computing systems. In data intensive applications that access large data files over wide area network environment, such as a data grid, caching mechanism can significantly improve the data access performance under appropriate workloads. In a data grid, it is envisioned that local disk storage resources retain or cache the data files being used by local application. Under a workload of shared access and high locality of reference, the performance of the caching techniques depends heavily on the replacement policies being used. A replacement policy effectively determines which set of objects must be evicted when space is needed. Unlike cache replacement policies in virtual memory paging or database buffering, developing an optimal replacement policy for data grids is complicated by the fact that the file objects being cached have varying sizes and varying transfer and processing costs that vary with time. We present an accurate model for evaluating various replacement policies and propose a new replacement algorithm referred to as *Least Cost Beneficial based on K backward references (LCB-K)*. Using this modeling technique, we compare LCB-K with various replacement policies such as Least Frequently Used (LFU), Least Recently Used (LRU), Greedy Dual Size (GDS), etc., using synthetic and actual workload of accesses to and from tertiary storage systems. The results obtained show that (LCB-K) and (GDS) are the most cost effective cache replacement policies for storage resource management in data grids.

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

DISCLAIMER

Accurate Modeling of Cache Replacement Policies in a Data Grid

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

Abstract

Caching techniques have been used to improve the performance gap of storage hierarchies in computing systems. In data intensive applications that access large data files over wide area network environment, such as a data grid, caching mechanism can significantly improve the data access performance under appropriate workloads. In a data grid, it is envisioned that local disk storage resources retain or cache the data files being used by local application. Under a workload of shared access and high locality of reference, the performance of the caching techniques depends heavily on the replacement policies being used. A replacement policy effectively determines which set of objects must be evicted when space is needed. Unlike cache replacement policies in virtual memory paging or database buffering, developing an optimal replacement policy for data grids is complicated by the fact that the file objects being cached have varying sizes and varying transfer and processing costs that vary with time. We present an accurate model for evaluating various replacement policies and propose a new replacement algorithm referred to as *Least Cost Beneficial based on K backward references (LCB-K)*. Using this modeling technique, we compare LCB-K with various replacement policies such as Least Frequently Used (LFU), Least Recently Used (LRU), Greedy Dual Size (GDS), etc., using synthetic and actual workload of accesses to and from tertiary storage systems. The results obtained show that (LCB-K) and (GDS) are the most cost effective cache replacement policies for storage resource management in data grids.

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

1 Introduction

We address the problem of cache replacement policies for Storage Resource Managers (SRMs) that are used in data grids, taking into account the latency delays in retrieving, transferring and processing of files. An SRM maintains a large capacity disk for caching file objects of varying sizes that are read from or written to Mass Storage Systems (MSS). An MSS may reside either at the same local site as the client or at some remote site that is accessible over a wide area network. A storage resource manager (SRM) [15], in the context of the data grid infrastructure [5, 8], is essentially a middleware component that facilitates

the sharing of data and storage resources. One key function of its services is the management of the disk cache for which it enforces various policies for its usage. An example of such policies being the cache replacement policy. Its role in the data grid is analogous to that of a proxy server or a reverse proxy server [2, 4], 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. We will use the terms file and object interchangeably.

One difference between caching into an SRM and caching into a web-server is that SRMs are designed to deal with batched jobs that make requests for files or objects of very large sizes and incur significantly long delays in transferring and processing them. Storage resources are accessible to users who interact with them, either directly through client interface or indirectly through application programs or other SRMs, for creating, destroying, reading, writing and manipulating *files.* Another major distinguishing characteristic between SRMs and web-proxy servers is the manner of handling replicas of files or objects. When an object is unavailable in a proxy-server, the proxy-server immediately contacts the source-server of the object. When an object is not found in an SRM's cache, it determines the fastest and the most cost effective means of fetching a copy into its cache. This may involve consulting a replica catalogue service and determining the current state of the available network bandwidth to the replicas' sources to make an intelligent decision as to which replica to fetch. SRMs are also used as front-ends to mass storage systems and hierarchical/tertiary storage systems.

Two significant decisions govern the operation of an SRM. Unlike web proxy severs, each job that arrives at an SRM can request hundreds or thousands of objects at the same time. As a result, an SRM generally queues the jobs and subsequently makes decisions as to which job needs to be serviced next and which file, from the batch of files of the selected job, must be retrieved into or transfered from the disk cache. If a requested file happens to be in the cache, the SRM may choose to *"pin"* it. Files that are in cache but are either in use or have been designated to be held in cache, are said to be *"pinned."*

The decision of selecting the next job to process is governed by a policy termed *"the service policy."* The decision of which file to retrieve into the disk cache is governed by a *"file caching policy."* When a decision is made to cache a file it may have to determine which of the files currently in the cache must be evicted to create space for the incoming one. This latter decision is also governed generally by what is termed a *"cache replacement policy."* The *service policy* and *caching policy* are sometimes combined together and referred to as the *"admission policy."*

A replacement policy involves computing some utility function $\phi_i(t)$ for each of the files *i* that poten-

tially can be replaced, and then replacing the ones that have either the minimum or the maximum utility function value depending on the criterion for replacement. Only files in the cache that are neither being processed nor specifically indicated to be pinned in the cache, are candidates for eviction. Such files are said to be *"unpinned."*

The performance measures of cache replacement policies are typically expressed by two metrics: the ***hit ratio*** and the ***byte hit ratio***. Given a reference stream (or a workload), the hit ratio is defined as the ratio of the number of objects found in the cache to the number of objects referenced in the workload. A byte hit ratio is the ratio of the volume of data (in bytes) found in the cache to the total volume of data referenced. In either case these measures give some indication of the improvement in response times and the savings in bandwidth utilization due to caching. However, none of these measures accounts for the latency incurred, at the data source, during large data transfers and in processing the file after it is cached. 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 take into account such delays. In this paper we adopt a third measure, first introduced in [11], which we call the *"Average Cost Per Reference (ACPR)"* and show that it is a more appropriate measure for the relative comparisons of cache replacement policies when large latencies exist. This is defined as the ratio of the total cost of all retrievals (in time units) into the cache, to the total number of references made in the workload. This gives a better comparison of the relative savings in time to retrieve, transfer and utilize files from the cache.

A considerable number of research studies have addressed the problem of *"cache replacement policies"* both within the realm of computer memory hierarchy and more recently in web-caching. See [17] for some of the survey reports on *"cache replacement policies."* The main objective of a replacement policy is to optimize a particular metric measure. The quest for optimal replacement policies is a long standing problem. The approach that has been taken are either analytical or by simulation modeling. Unfortunately, modeling of cache replacement policies presented in the literature so far, assume instantaneous references and hence do not adequately evaluate cache replacement policies in the data grid environment.

We present, in this paper, a more accurate model and algorithm for evaluating and comparing various replacement policies. The distinction being made between modeling of policies that assume instantaneous references and one that takes into account the long delays at each reference, impacts the size of a cache that can be used to handle a given workload. Under instantaneous references, the minimum size

of cache needed is one that can hold the largest size of the object. When reference delays (i.e., the sum of the delays to locate and retrieve the object or its replica from source, transfer the object and hold the object in cache for processing), are considered, the minimum size of the cache required is considerably larger than the maximum object size; otherwise some file requests in the workload may not be satisfied in which case they may be rejected. It follows intuitively then that there is a minimum size of a cache, much larger than the largest size of the object, required to successfully process the entire references in a workload. This cache size constraint is not reflected in cache models with instantaneous references.

The main contributions in this paper are the introduction of accurate models for evaluating replacement policies and the presentation of comparative performance results of the traditional performance metrics, e.g., hit-ratio, and ACPR for various caching policies. In [11] we introduced a definition of a utility function for ranking file objects that are candidates for replacement and also described briefly an efficient algorithm for evaluating the functions for each file object when one has to be evicted. The cache replacement policy introduced was referred to as the *least cost beneficial based on the K backward references* or *LCB-K policy* for short. We present, in this paper, more extensive results, using a synthetic workload and two real workloads: one from file caching activities of the mass storage system at the Thomas Jefferson National Accelerator Laboratory (JLab), the other from the access logs of the high performance storage system (HPSS) at the National Energy Research Scientific Computing Center (NERSC), at Berkeley. We compare the LCB-K policy with other known replacement policies such as *random (RND), least frequently used (LFU), least recently used (LRU), maximum inter-arrival time based on last k-backward references (LRU-K)* and *Greedy Dual Size (GDS),* under the performance metrics of *"hit ratio"*, *"byte hit ratio"* and *"average cost per reference."* Under the ACPR performance metric, LCB-K and GDS give the best minimum average cost per reference compared with the other replacement policies. We note that these two policies do not necessarily give the maximum values for either the hit ratio or the byte hit ratio compared to the other policies.

2 Configuration and Related Works

Figure 1 shows the schematic diagram of the positional role of an SRM within a data grid. A storage resource manager may be specialized to be either a *disk resource manager (DRM)* or a *hierarchical resource manager (HRM)*. Jobs submitted to a DRM are requests for files that are either in the DRM's disk cache or can be retrieved from another remote SRM into its cache. An HRM acts as a front end to

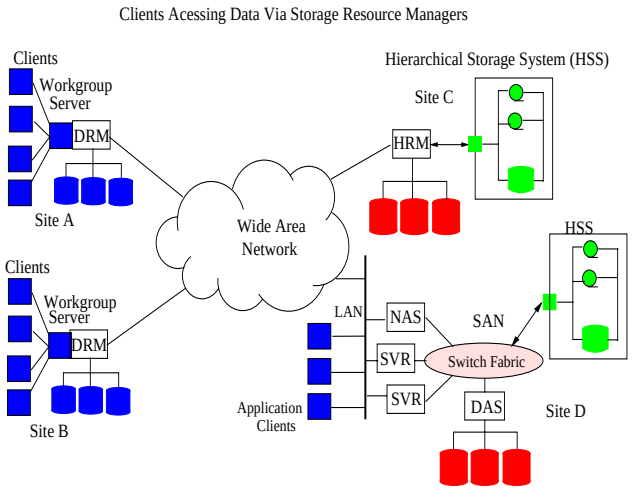


Figure 1: Use of Storage Resource Managers in a Data Grid

a tertiary storage system. It services requests from either its disk cache or from a tertiary storage by first retrieving the file into its disk cache.

An SRM is applicable in environments that deal with file transfers to and from shared disks, mass storage and archival tape systems over wide area networks. Caching strategies provide, in these environments, the benefits of improved data access, improved response times, savings in network bandwidth and decreased server congestion. The performance of storage technologies, used in the delivery of high volume data over wide area networks, is improved with caching techniques. These include systems for *Data Staging* [16], *Web-Caching* [4], *storage brokering* [13], Network Attached Storage (NAS) and file servers (SVR) of Storage Area Networks (SAN) that is configured to include independent hierarchical storage systems and other direct access storage (DAS). In Figure 1, we show how these may be configured as part of a data grid.

Caching techniques in these systems improve the performance when the file reference streams exhibit:

- *Locality of Reference:* A file that is referenced and read into a cache is referenced multiple times by the same user over a short period of time.
- *Shared Access to Files:* The same file after it is read into cache, is also referenced, multiple times, by different users.

Earlier works on file caching in distributed systems and the staging of files from tertiary storage have been presented in [7, 9]. Recent studies on caching have focused more on web-caching [4, 14]. Cao and

Irani [4] present a relative comparison of various cache replacement policies that have been proposed for web-caching. They propose a replacement policy for web-caching called the *Greedy-Dual-Size (GDS)* [4]. Although these works have also presented some relative comparison of caching algorithm, they have not addressed the impact of the delays incurred at the source of the file object, delays in caching the file and the time to hold the file in cache for processing. The impacts of such delays in the caching algorithms can affect the performance metrics thereby making one preferable over the other. A correct model of the caching algorithm is the first step towards making an informed decision. There are some major differences between caching of large data files in SRMs and web-caching. We summarize the differences in the Table 1 below.

Characteristic Property	**Web Caching**	**Disk Caching in SRMs**
File/Object Size	Variable size objects of the order of megabytes	Variable size objects of the order of 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	Almost Instantaneous	In seconds up to a many minutes
Caching Requirement	Optional	Mandatory
Batched Requests	Typically one request references one object but may have a additional references to linked objects.	May involve thousands of files in one submitted job.
Bundle Constraint	Only one object is referenced per request.	May require that multiple files be accessed simultaneously.
Cache Consistency	Cognizant of modified documents	Predominantly Read-Only and ignores consideration of cache coherence
Network bandwidth requirement	Standard Internet	High speed gigabit networks
Replica Access	Considers only the source server of an object if not found in cache	Involves intelligent selection of a site to retrieve a file replica if not found in cache.

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

3 Ranking of Files for Eviction

The basic idea of our file replacement policy is to evaluate the utility function $\phi_i(t)$ for each file i in the disk cache. A file object i of size s_i has a retrieval cost $c_{i,r}(t)$ from a site r at time t. In the data grid environment there could be replicas of the same file at different sites r, each with a different file access

cost. We will denote the cost simply by $c_i(t)$, with the understanding that this is the minimum cost over all replica sites. At each instant in time t, when we need to acquire space of size s_i for a file i, we order all the unpinned files in non-decreasing order of their utility functions and evict the first m files with the lowest values of $\phi_i(t)$ and whose sizes sum up to or just exceed s_j. We always assume that the sizes of the cached files are relatively small compared to the total size S, of the cache.

Since the precise characteristics, such as the cost $c_i(t)$, to retrieve the file at time t, is not known in advance, caching techniques make use of the information accumulated from past references. Examples of such information retained are: the time of the last reference, the cost in time of the last retrieval, the number of accumulated references (also called the frequency count) to the file, etc. The determination of the utility function $\phi_i(t)$ and how it is evaluated distinguish one cache replacement policy from another. For example in the *Least Frequently Used* policy, the ranking is done based on the frequency count of the references to each file. In the *Least Recently Used (LRU)* the ranking is based on the last reference time while *LRU-K* [10] is based on the K^{th} backward reference time. Others such as the *Greedy Dual Size (GDS)* and the *Least Cost Beneficial* cache replacement utilize a more involved utility function that includes the size of the file, the cost of retrieval and the frequency counts. For example, our new LCB-K algorithm utility function is given by:

$$\phi_i(t) = \frac{k_i(t)}{t - t_{(-k_i)}} * \frac{g_i(t) * c_i(t)}{s_i} \tag{1}$$

where, for each file i, s_i is its size, $k_i(t)$ is the number of the most recent references retained, up to a maximum of K, within the time interval $[t - t_{(-k_i)}]$, $t_{(-k_i)}$ is the time of the $k_i(t)$ backward reference, $g_i(t)$ is the cumulative count of references to the file over the active period of references to the file and $c_i(t)$ is the cost of retrieving the file from its source into the cache at the time t. The idea of retaining up to K relevant history of references is borrowed from the development of LRU-K [10]

3.1 Reference Streams in SRMs

Consider an SRM that serves as a front end to a tertiary storage system system that has N distinct files $F = \{f_1, f_2, \ldots f_N\}$. Denote each job for file requests to an SRM by $J_j = \{f_{1,j}, \ldots f_{i,j}, \ldots\ldots f_{m_j,j}\}$, $j = 1, \ldots, q$, when q jobs are in the queue. The combined use of the *service* and *caching policies* generates a schedule of file admissions which in turn derives a file reference stream $\omega = r_1, r_2, \ldots, r_t, \ldots$. Each file

reference is for a file specified in some job, i.e., $r_t = (f_i, j), i = 1, \ldots N, j = 1, \ldots q$. The logged reference stream ω, constitutes a workload.

In this environment the concept of correlated references is *weak* since each job makes only one request for each file and the references to the same file come from independent jobs. Further, we assume that the files are written once but read many times. Hence the file accesses are predominantly read-only. Correlated references originate frequently from file updates where the file is read into the cache, referenced many times to update subsets of the file and then written back onto the backing store or tertiary storage. Frequent file update is not the mode of operation in our setting.

Although multiple jobs can be serviced when a file is retrieved into cache, we consider each reference in the reference stream ω as being independent. The appearance of a file into the cache, as a result of a reference from some job, impacts the subsequent decision rule for scheduling the next file to be cached. For example, the *service* and *caching policies* may take into account the fact that a file being requested is already in the cache when making a decision as to which file request and from which job should be honored next.

3.2 Metric for Ranking Files in SRMs

Replacement algorithms are key to the implementation of a caching system. Not only should this be evaluated in almost negligible time relative to the time it takes to cache an object, it should optimize in some sense a measure of a performance metric. Cache replacement policies are typically designed to optimize the *hit ratio* usually by retaining in the cache either the most frequently referenced objects or the most recently referenced object. The former effectively evicts the least frequently used object (i.e., the LFU-policy), the latter evicts the least recently used object (i.e., the LRU-policy). Both policies are predicated on the assumption that a reference stream has a high degree of shared and locality of references.

Since the goal of caching is to improve some performance measures such as response time, throughput, network bandwidth usage, etc., we examine the significance of hit ratio, byte-hit ratio in the improvement of such performance measures.

Hit Ratio: This assumes that all files are of the same size and have the same access cost. This assumption is unrealistic in the use of SRMs in data grids. The files have varying sizes and have replicas at different sources with different delays and transfer cost into an SRM's disk cache. It is easy

to envisage a replacement policy that favors only files of small sizes thereby retaining as many files in cache as possible and improving the hit ratio at the expense of high retrieval cost and poor response time whenever large files are referenced. Hit ratio only measures the effectiveness of the use of a cache as the number of hits and does not reflect in any way the effects of source and transfers delays of the objects.

Byte Hit Ratio: The byte hit ratio used in measuring the relative performance of cache replacement algorithms implicitly addresses transfer delays but is based on the assumption that the rate of transfers from every originating source of an object to the cache is constant. This assumption, as in the case of hit ratio, is unrealistic. This performance metric reflects the relative savings in bandwidth usage achieved by caching but does not reflect the delays at the originating sources of the files.

Average Cost Per Reference (ACPR): This metric measures the effectiveness of a caching policy by the average response time per reference. It takes into consideration the total delay in caching files of varying sizes, varying source delays and varying transfer times. Consequently an optimal replacement algorithm based on ACPR, implicitly minimizes the response times of file requests. This is a more practical objective in designing cache replacement algorithm for SRMs on the grid.

Our objective then in the design of a cache replacement policy is to optimize the overall response time of file requests. As such, the performance metric we optimize (i.e., minimize), is the expected access cost of a file per reference.

4 Derivation of the LCB-K Utility Function

In virtual memory paging and file buffering, where objects (i.e., program and data pages) are of fixed sizes and the cost of disk to memory transfer is constant, the accepted *"principle of optimality"* for replacement policies was first proposed by Belady [3]. It states that *"the page y evicted is that which has the furthest time of the next reference."*

Given a reference stream $\omega = r_1, r_2, \ldots, r_t, \ldots$. where each reference is for an object, i.e., $r_t = (f_i), i = 1, \ldots N$, we can now consider the reference stream as random variables with stationary probabilities $p_1(t), p_2(t), p_3(t), \ldots, p_n(t), \ldots$ with $Prob(r_i = j) = p_j(t)$. For a cache of size S such that the size s_i of each object i is very much less than S, the principle of optimality implies that at the next reference instant

$t+1$ we should always retain in the cache the I objects such that

$$\sum_{i \in I} p_i(t); \tag{2}$$

is maximized subject to

$$\sum_{i \in I} s_i \ \leq S. \tag{3}$$

The above equations maximizes the hit ratio. Now if we assume the cost of retrieving a file object f_i, of size s_i into the cache is $c_i(t)$, with the proviso that this cost varies from a reference instant to a reference instant then from the reference instant t to the next instant $t+1$, we need to retain I objects in cache such that

$$\sum_{i \in I} p_i(t) * c_i(t); \tag{4}$$

is maximized subject to

$$\sum_{i \in I} s_i \ \leq S. \tag{5}$$

Maximizing the objective function 4 implies a minimization of the response time per reference. The task of an SRM is to solve, at each replacement instant, the above classical *Knapsack Problem* which is known to be NP-hard [6]. By considering 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 I satisfying $\sum_{i \in I} s_i \ = S$ and thus we can restate the problem as

$$\textit{maximize} \ \sum_{i \in I} p_i(t) * c_i(t); \tag{6}$$

subject to

$$\sum_{i \in I} s_i \ = S. \tag{7}$$

The above problem can now be considered as equivalent to the *fractional knapsack* problem for

which an optimal solution is given by a simple greedy algorithm as follows: The items $i \in I$ are ranked in non-increasing order of $\phi_i(t) = p_i(t) * c_i(t)/s_i$ and then the first I items are retained in the cache. The solution implies that whenever some object in the cache needs to be evicted at some instant in time t, the eviction candidate is one that has the minimal utility function $\phi(t)$ given by

$$\phi_i(t) = \frac{p_i(t) * c_i(t)}{s_i} \tag{8}$$

Similar conclusion are reached in [9, 14] but under different assumptions.

The utility function, as expressed by equation (8), is impossible to apply since we do not know the probabilities, and further these probabilities are not stationary. We utilize the history of accesses to estimate the probability of a future access of a file. 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 (8) may be replaced by

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

Since the replacement decision is based only on the relative rankings of $p_i(t)$, we can rewrite equation (8) as

$$\phi_i(t) = \lambda_i(t) * \frac{c_i(t)}{s_i}. \tag{9}$$

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, $1 \le k_i(t) \le K$. 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 $k_i(t)$ retrievals. Note that before an object becomes a candidate for eviction, at least one access for the object must have been made in order to cache it. Further, the observed locality of

reference suggests that the most frequently accessed object is most likely to be referenced in the future, we factor in the accumulated number of references made to a file. Let this be denoted by $g(t)$, then our eviction candidate is the object with the minimum value of $\phi_i(t)$ where

$$\phi_i'(t) = \frac{k_i(t)}{t - t_{(-k_i)}} * \frac{g_i(t) * c_i'(t)}{s_i} \tag{10}$$

Empirical evidence derived from plotting the accumulated frequency and the probability of access in the next time steps suggests that $g_i(t)$ should be applied as a decaying function of the accumulated frequency [12]. The derivation of the parameters of an appropriate decaying function given a cumulative frequency counts, is left for future work.

Our disk cache replacement policy, based on the equation (10), with the superscript dropped from the retrieval cost $c'(t)$, is that given in equation 1. We refer to this as a *Least Cost Beneficial* replacement policy based on at most *K* backward references or *LCB-K* for short. Using equation (1), the problem of implementing an efficient algorithm for quickly ranking the objects in cache is still non-trivial.

5 The Cache Replacement Algorithm

A direct application of the LCB-K utility function given by equation 10, for selecting a candidate for replacement is computationally expensive. Whenever an object needs to be evicted at time *t*, the utility function must be evaluated for every object in the cache and this takes time $O(I)$ when I objects are in cache. This is so since the function $\phi_i'(t)$ is a non-stationary ranking function in the sense defined in [1].

We utilize then a tournament of heaps to achieve $O(\log I)$ selection in the following manner. We divide the number of *unpinned* object in cache into groups *m* groups according to the $(k_i(t) * g_i(t) * c_i(t))/s_i$ and maintain each group as a priority queue with the object having the minimum utility function value at the root. Only the root node element of each priority queue can be evicted. Each time a replacement needs to be made, the root element with the lowest utility function value is evicted from the cache.

In simulating cache replacement polices in SRMs, or any disk caching environments such as Web-Caching, one needs to account specifically for three types of delays:

i the latency incurred at the source of the file;

ii the transfer delay in reading the file into the disk cache; and

iii the holding or pinning delay incurred while a user processes the file after it has been cached. This may simply involve transferring the file into the user's workspace.

The simulations of the replacement policies are done as discrete event simulations. Each reference r_i provides five distinct event times. These 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 created upon an arrival of a request. This is then inserted into an event queue denoted by *EvtQueue*. An event object *evtObj* has two fields, *eventType(evtObj)* and *schdTime(evtObj)*, that identify respectively the type of event and the scheduled time at which that event should occur. The *EvtQueue* is implemented as a priority queue.

The action taken upon the occurrence of an event is implied by its type. We describe the semantic actions in handling one type of event, i.e., "Start_Caching," to illustrate the idea. Suppose an event object *evtObj*, is removed from the top of the event queue. If the event type given by *eventType(evtObj)* is a "Start_Caching" event, then the *eventType(evtObj)* is set to *"End_Caching"* and the *schdTime(evtObj)* is set to the scheduled completion time for retrieving the file into the cache. The event object *evtObj*, is then reinserted into the priority queue *EvtQueue*.

The simulation is driven by a workload of file requests. Suppose the time of arrival of a request r_i is t_0 and assuming the root node of a non-empty *EvtQueue* is denoted by *EvtQueue(Root)*. If $t_0 \geq$ *schdTime(EvtQueue(Root))* then *EvtQueue(Root)* is removed and assigned to *evtObj*. The simulation then executes the actions corresponding to the *eventType(evtObj)*. If $t_0 <$ *schdTime(EvtQueue(Root))* then a new event object is created using the information of the arriving request. This is then inserted into the *EvtQueue*. We should also remark that of the five event times of a request, the simulation only checks if the requested file is in cache at the time when a *Start_Caching* event occurs. If the file is in cache, the request is immediately scheduled for processing and this instant corresponds to a cache hit. However if the file is not in cache then a cache replacement algorithm is executed to free enough space to retrieve the file.

6 Performance Comparison of Some Replacement Policies

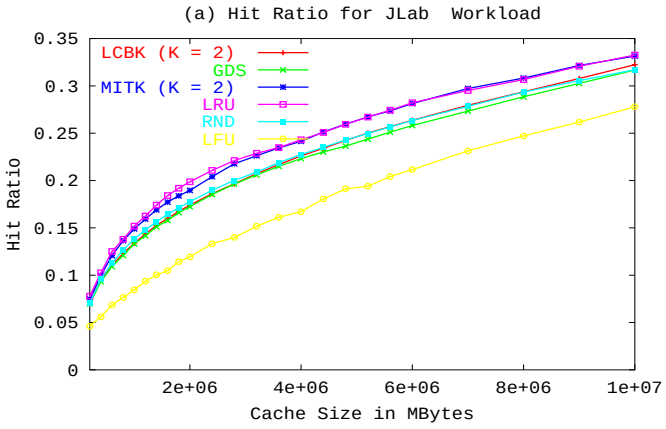
We compared the performance metrics of *hit ratio*, *byte hit ratio* and *average cost per reference* for a number of cache replacement policies, namely random (RND), least frequently used (LFU), least recently used (LRU), maximum inter-arrival time based on at most K backward references (i.e., a variant of LRU-K), greedy dual size (GDS) and least cost beneficial based on at most K backward references (LCB-K)

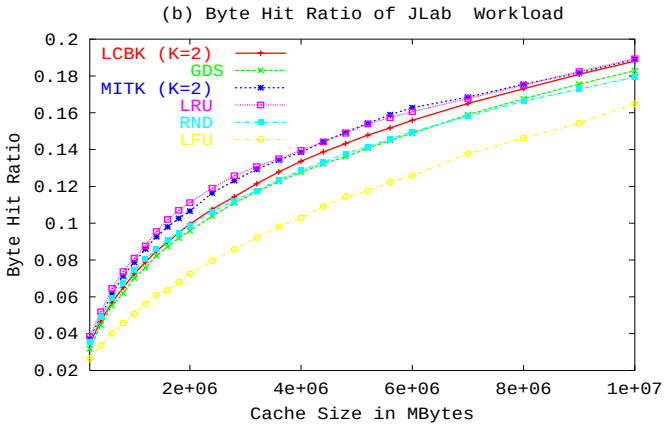
under the modeling strategy with delays. We set the active lifetime T to be five days. That is, if a file has not been accessed in the last five days, it is purged from memory. The simulations were conducted for both synthetic and real workloads of a mass storage system. Two real system workloads are used in our experiments. The first is a log of file access activities, for about a six months period, of the mass storage system, JasMINE, at JLab. The second workload is derived from file access to an HPSS at NERSC. The JLab workload has a medium locality of reference and uses the LRU caching policy. The NERSC workload has very little locality of references and the synthetic work is instrumented to have a reasonable high degree of locality of reference. The graphs are given respectively in Figures 2, 3 and 4.

6.1 Some Experimental Results

Figures 2a, 2b and 2c shows the graphs of the performance metrics of hit ratio, byte hit ratio and average cost per reference for the workload from JLab. The Figure 2a shows the hit ratios for RND, LFU, LRU, GDS, MIT-K and LCB-K for K = 2. MIT-K and LRU give the best performance results under the hit-ratio performance metric. Even then the graphs are not significantly different from those of GDS and LCB-K. On the other hand, examination of the graphs of the average retrieval cost per reference, depicted in Figure 2b, shows that LCB-K and GDS give the best performance measures under this metric. In all three performance measures LFU shows the worst performance.

Figures 3a, 3b and 3c show the corresponding graphs for the workload from NERSC. The graphs of the hit ratios for GDS, LCB-K, MIT-K and LRU all show comparable behavior. RND and LFU replacement policies behave poorly. The graphs of Figure 3c depict some interesting behavior with varying cache sizes. For a cache size between 200 GBytes and 500 GBytes, the average cost per reference increases with increasing cache size. In this phase the cache size is insufficient to handle the request arrivals. This causes some files to be rejected and not cached.





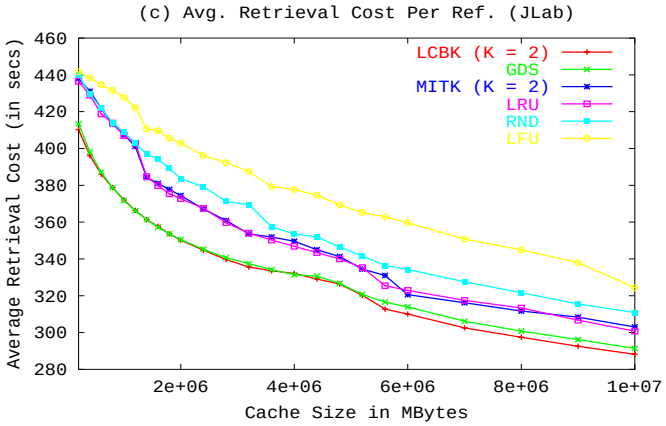
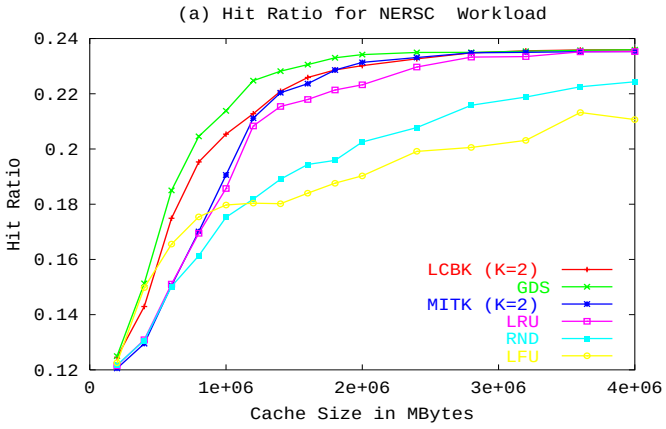
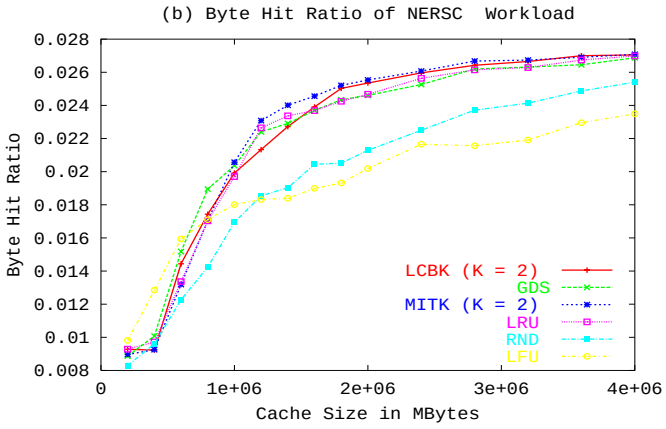


Figure 2: Graphs of Real Workload from Jefferson's National Accelerator Facility (JLab)





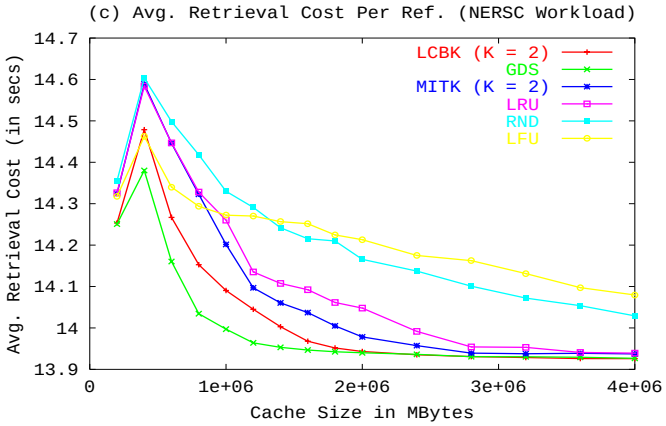
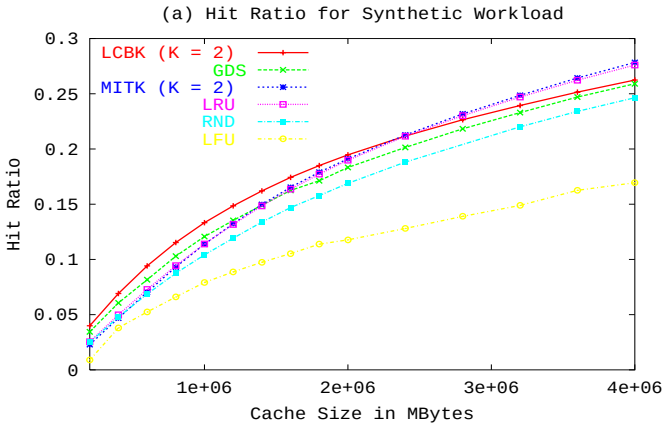
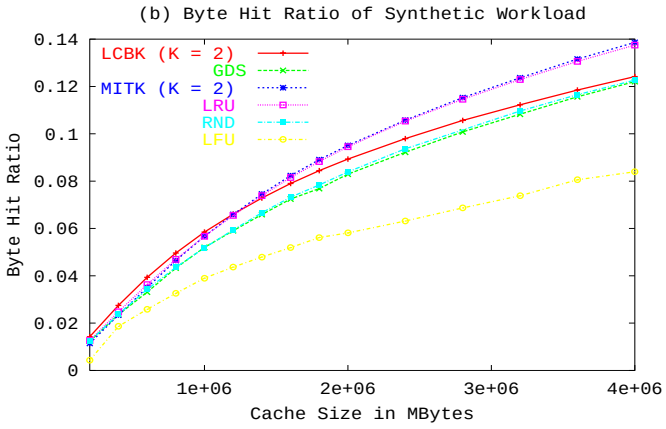


Figure 3: Graphs of Real Workload from National Energy Research Scientific Computing (NERSC)





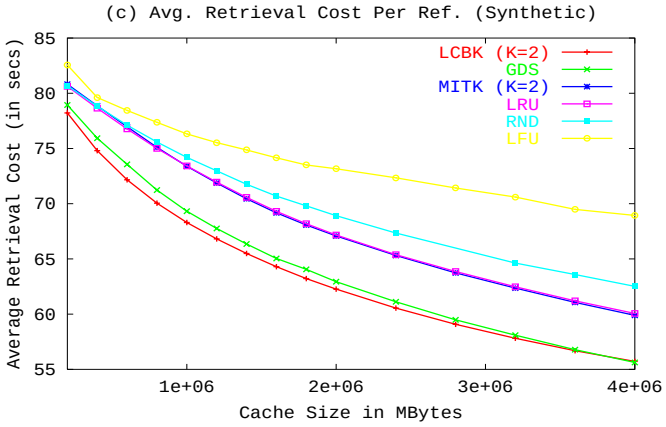


Figure 4: Graphs for Synthetic Workload

At a cache size of about 500 Gbytes the cache is sufficient large to handle the workload. Subsequent increases in the cache size after this point naturally causes a steady decrease in the average cost of retrieval per reference. The GDS and LCB-K replacement policy quickly reach a steady state at cache size of about 2.0 Terabytes. This also suggests that further increases in the cache size have little or no effect in improving (i.e., decreasing) the response times of accessing files.

The corresponding graphs for the synthetic workload are shown in Figures 4a, 4b and 4c. These illustrate the same relative performance measures of the various policies as indicated by a real workload. The major difference here is that the differences between policies in the average cost per reference measure, are more pronounced. This is due to the fact that this workload has a higher degree of locality of reference than the previous two real workloads. 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 confirmed this fact in our simulations and therefore present results for only $K = 2$ in this paper.

7 Conclusion and Future Work

Caching in storage resource managers, has some characteristic features that distinguish it from caching in other domains such as virtual memory, database page buffering and web-caching. In particular, file caching in SRMs involves variable size files or objects that are very large and the delays caused by source latency, file transfers and processing of the files significantly affect its performance. Unlike traditional simulations of cache replacement policies, we have implemented a realistic simulation model that accounts for the delays in processing objects in the cache.

We also defined a utility function for determining which files need to be evicted from the cache. The utility function was used to develop a new cache replacement policy referred to as the LCB-K. Using the simulation model, we compared and showed the results of the replacement policies of RND, LFU, LRU, GDS, MIT-K and LCB-K under the performance metrics of hit ratio and average cost per reference for both synthetic and actual workloads. The average cost per reference is the most realistic performance metrics for evaluating cache replacement policies for storage resource managers and under this performance measure, LCB-K and GDS replacement policies give the best results of all the policies compared.

Acknowledgment

We would like to express our sincere gratitude to Andy Kowalski of Jefferson's National Accelerator Facility and to Havard Holmes of the National Energy Research and Scientific Computing for providing us with the respective real 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] A. V. Aho, P. J. Denning, and J. J. Ullman. Principles of optimal page replacement. *J. ACM*, 18:80 – 93, Jan. 1971.

[2] M. ARLITT, L. CHERKASOVA, J. DILLEY, R. FRIEDRICH, and T. JIN. Evaluating content management techniques for web proxy caches. *Performance Evaluation Review*, 27(4):3 – 11, Mar. 2000.

[3] L. A. Belady. A study of replacment algorithms for virtual-storage computer. *IBM Systems Journal*, 5(3):78 – 101, 1966.

[4] P. Cao and S. Irani. Cost-aware WWW proxy caching algorithms. In *USENIX Symposium on Internet Technologies and Systems*, 1997.

[5] 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.

[6] M. Garey and D. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. W. H. Freeman, 1979.

[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] E. J. Otoo, F. Olken, and A. Shoshani. Disk cache replacement algorithm for storage resource managers in data grids. In *The 15th Annual Supper Computer Conf.*, Baltimore, Maryland, Nov. 2002.

[12] J. Pitkow and M. Recker. A simple yet robust caching algorithm based on dynamic access patterns. In *Proc. 2nd Int'l. WWW Conf.*, pages 1039 – 1046, Chicago, USA, 1994.

[13] A. Rajasekar, M. Wan, and R. Moore. MySRB & srb - components of a data grid. In *The 11th Int'l. Symp. on High Perf. Distrib. Comput. (HPDC-11)*, Edinburgh, Scotland, Jul. 24 - 26 2002.

[14] 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.

[15] A. Shoshani, A. Sim, and J. Gu. Storage resource managers: Middleware components for grid storage. In *10th NASA Goddard Conference on Mass Storage Syst. and Tech.*, Apr. 15 - 18 2002.

[16] M. Tan, M. Theys, H. Siegel, N. 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.

[17] J. Wang. A survey of web caching schemes for the internet. In *ACM SIGCOMM'99*, Cambridge, Massachusetts, Aug. 1999.