File Replacement Algorithm for Storage Resource Managers in Data Grids

J.H. Abawajy

Deakin University
School of Information Technology
Geelong, Victoria, Australia.

Abstract. The main problem in data grids is how to provide good and timely access to huge data given the limited number and size of storage devices and high latency of the interconnection network. One approach to address this problem is to cache the files locally such that remote access overheads are avoided. Caching requires a cache-replacement algorithm, which is the focus of this paper. Specifically, we propose a new replacement policy and compare it with an existing policy using simulations. The results of the simulation show that the proposed policy performs better than the baseline policy.

1 Introduction

Grid computing enables sharing, selection, and aggregation of geographically distrusted autonomous resources (e.g., super-computers, cluster-computing farms and workstations). One class of grid computing and the focus of this paper is the data grid systems [1] [7] that facilitate the sharing of data and storage resources (e.g., direct attached storage, a tape storage system, mass storage sys-tem, etc.) for large-scale data-intensive applications. Research in data grid computing is being explored worldwide and data grid technologies are being applied to the management and analysis of large-scale data intensive applications. The main problem in data grids is how to provide good and timely access to such huge data given the limited number and size of storage devices and high latency of the interconnection network. One approach to address this problem is to cache the files locally such that remote access overheads are avoided. The motivation for disk caching in data grid is that it tasks several minutes to load data on to the mass storage system. Also, it takes a very long time (up to a few hours) to complete file transfers for a request over WAN. In addition, the user's personal workstation and in some cases the local computer centers may not be able to store all the dataset for a long time for his needs.

In this paper, we address the problem of disk cache replacement policy in data grids environments. The motivation for studying this problem is that the performance of the caching techniques depends heavily on the cache replacement policy that determines which file(s) are evicted to create space for incoming files. Caching techniques have been used to improve the performance gap of storage hierarchies in computing systems, databases and web-caching. Note that, unlike cache replacement policies in virtual memory paging or database buffering or web-caching, 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. For example, files or objects in web-caching can be of any media type and of varying sizes. However web caching in proxy servers are realistic only for documents, images, video clips and objects of moderate size in the order of a few megabytes. On the other hand, the files in data grids have sizes of the order of hundreds of megabytes to a few giga-bytes. Some of the differences between caching in data grids and web-caching are given in [2].

The rest of the paper is organized as follows. Section 2 describes related work. Section 3 presents the proposed file replacement algorithm. Section 4 discusses the performance evaluation framework and the results of the proposed policy. Conclusion and future directions are presented in Section 5.

2 Related Work and System Model

The problem of disk cache replacement policy is one of the most important problems in data grids. This is because effectively utilizing disk caches is critical for delivering and sharing data in data-grids considering the large sizes of re-quested files and excessively prolonged file transmission time. Ideally, one strives for an optimal replacement policy for a particular metric measure. An optimal replacement policy evicts a file that will not be used for the longest period of time. This requires future knowledge of the file references, which is quite difficult, if not impossible, to determine.

2.1 System Model

The data grid model used in this paper has n sites interconnected by a wide-area network (WAN). Each site has one or more storage subsystems (e.g., a tape drive and disk) and a set of processors. The resources within a site are interconnected through a local area network (LAN). We assume that the disk subsystem is used as data cache while all other storage subsystems are used as archives. Each site maintains a set of files, which have varying sizes. A given file, *fi*, can be in one or more sites in the grid at the same time. We use a nomenclature hard replica (HR) to designate a copy of *fi* file that has not been modified and kept in one or more well-known sites in the system while those file replicas that may have been modified or subject to be purged from a site (i.e., a file copy is not guaranteed to be available all the time) as soft replica (SR). For example, in European Data Grid system [3], the original copy of the files (i.e., FH replica) is always kept in at least the European Organization for Nuclear Research (CERN) site while other sites can keep a copy of the files if need be.

Each site has a storage resource manager (SRM) that is responsible for the management of the local storage subsystem usage as well as maintaining HR (may be SR as well) file replica meta-data information (e.g., location, cost, size, etc.) in replica table. The storage resources in data grids are accessible to a user, remotely or locally, through a middle-ware service called storage resource manager (SRM). The information on HR and SR file replicas is collected in two ways. First, the sites that are designated as keepers of HR replicas can inform all other sites by sending a notification message when a new file is created. Also, every time these sites send a copy of a given file to a remote SRM upon a request from the latter, a notification message along the identity of the SRM is sent to all sites within certain distance (e.g., region) from SRM. Alternatively, each SRM will send its own notification message to certain sites only when it receives a copy of the file. The notification message includes the cost of preparing the file for transferring (e.g., the time taken to stage a file from a tape to a disk). When a given SRM receives the notification message, it will estimate the cost of a file acquisition based on the time taken for the notification message using the following function:

$$Cost\left(f_i, v, u\right) = T_{process}\left(v\right) + \left(Latency + \frac{size\left(f_i\right)}{bandwidth\left(u, v\right)}\right) \tag{1}$$

Data requests from the local system users are sent to the local SRM for processing. We assume that when a user submits a request for data, the size of storage locally available (i.e., reserved) for the data is also declared. The file request scheduler (FRS) is responsible for the admission of data requests, R_i, from users. All requests that can be satisfied from the local disk are called local request while those that are fetched from remote sites are called remote requests.

Each admitted request, R_i, is stored in the ready request queue and serviced, based on the scheduling approach used (e.g., first come first service), by the file request placement (FRP) component of the SRM. When the file has to be brought in from another site, the SRM searches the replica table for the best site to get the file from. In our system, local requests have priority over remote requests. We refer to files cached locally that are in use as active files while those that are not in use as passive files.

The key function of the SRM is the management of a large capacity disk cache that it maintains for staging files and objects of varying sizes that are read from or written to storage resources that are either at the same local site or some remote site. Two significant decisions govern the operation of an SRM: file request scheduling and file re-placement. Each of file 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 these requests and subsequently makes decisions as to which files are to be retrieved into its disk cache. Such decisions are governed by a policy termed the file admission policy. When a decision is made to cache a file it determines which of the files currently in the cache may have to be removed to create space for the incoming file. The latter decision is generally referred to as a cache replacement policy, which is the subject of this paper.

2.2 Related Work

The Least Recently Used (LRU) and Least Frequently Used (LFU) replacement policies are two extreme replacement policies. The LRU policy gives weight to only one reference for each file, that is, the most recent reference to the file while giving no weight to older ones representing one extreme. In contrast, the LFU gives equal weight to all references representing the other extreme. These extremes imply the existence of a spectrum between them. A number of replacement policies that fall within such a spectrum have been proposed in the literature.

A replacement policy referred to as Least Cost Beneficial based on K backward references (LCB-K) is proposed in [2]. LCB-K is an always cache policy, which means that all files retrieved from remote sites will be cached. It uses a utility function that probabilistically estimates which files would be accessed relatively less soon for ranking the candidate files for eviction. Whenever a file in the cache needs to be evicted, the algorithm orders all the files in non-decreasing order of their utility functions and evict the first files with the lowest values of utility function and whose sizes sum up to or just exceed that of the incoming file. The main problem with this policy is that the utility function used to predict the future references of the file is based on global information such as the file reference counts.

An economic-based cache replacement (EBR) policy is proposed in [5]. EBR uses probability-based utility function technique to measure relative file access locality strength for the files and makes a decision as to cache a file or not. If there enough space to store a file, a newly arrived file is automatically stored on the disk. However, if there is no space left on the disk, EBR evicts the least valuable file from the disk. In order to make this decision, the Replica Optimiser keeps track of the file requests it receives and uses this history as input to a future revenue prediction functions. The prediction function returns the most probable number of times a file will be requested within a time window W in the future based on the requests (for that or similar files) within a time window W' in the past. The algorithm keeps track of the access history and uses a prediction function for estimating the number of future access for a given file in the next *n* requests based on the past *r* request in the history.

In [6], a replacement policy called the Least Value-based on Caching Time (LVCT) that exploits access regularities (i.e., one-timers versus multiple-timers) in references is proposed. A single stack with two fields (i.e., caching time and file size) is used for storing every file request in the site. The size of the stack is adjusted by pruning it when the total size of files represented in the stack is greater than twice the cache size or the number of files represented in the stack is greater than two times of the number of files in the cache. LVCT is shown, through simulation, that it outperforms LRU, GDS and LCB-K policies. However, this performance comes at additional costs for taking replacement decisions as updating the cache state at each reference has overhead. Moreover, there are several shortcomings of this policy. First, it assumes that clients have enough storage to store the file, which is not the case most of the time. Second, a file with multiple accesses must be brought from remote size at least twice. Third, maintaining the stack takes time. Even if there is enough disk space, first time access file is not stored. Decision to evict or not is made after a remote file has been completely received.

We propose a new replacement policy that takes into account factors such as the latency delays in retrieving, transferring and processing of the files. We compare the proposed replacement policy with several existing policies through simulation under varying system and workload parameters (e.g., account for delays in cache space reservation, data transfer and processing). The results of the experiment shows that the proposed policy performs substantially better than the baseline policies used in the experiments.

3 New Disk Cache Replacement Policy

The proposed policy is called New Cache Replacement (NRP) policy. It combines locality, size and cost of files when making a replacement decision. Also, only events that matter for replacement decisions are taken into account. Note that this is not the case in the other policies discussed in the previous section. For each file *f*, we keep its size S and the cost of fetching it from remote site C. Note that C can be the actual cost incurred of a file or estimated as per Equation (1). This information is used in determining the set of files to be evicted if this need be. In the proposed algorithm, only passive files in the cache are potential candidates for eviction. Also, we assume that SRM serves user requests based on FCFS with local requests having priority over remote requests.

The proposed policy has three phases: request admission phase (RAP), candidate selection phase (CSP) and file eviction phase (FEP). The RAP decides if the request would be a remote-fetch-only (i.e., no caching of the file occurs) or remote-fetch-store (i.e., fetch the file from remote site and then store it locally). If there is enough disk space, the request automatically admitted. When there is a miss, the algorithm invokes the CSP to select possible files to be replaced. That is, for each remote-fetch-store request i, CSP selects a set of candidate files that are passive with their size greater than or equal to $S_i / 2^K$ where $K >= 0$. Initially, we set k=0 and if no file is found, we increment k by one and this process is continued until a set of files that meet the criterion is located. The FEB uses the list of candidate files returned by the CSP to replace appropriate files. The algorithm evicts a set of files from the list (i.e., L) whose sizes sum up to or just exceed that of the incoming file.

The advantage of the NRP is that it uses local information as opposed to global information when making the replacement decisions. It also incorporates locality, size and cost considerations effectively to achieve the best performance possible. As it selects the file that have size equal to or greater than the documents to be replaced; this technique of evicting the files saves many misses by not evicting the small files that are least recently used. In addition, it tries to minimize the number of files replaced by evicting files that have not been used recently. The main and more important thing is that the above algorithm also takes in to account the cost of fetching the file to cache from its original severs. By doing so, this algorithm also considers the cost of the file before it replaces a particular page. The above algorithm has some drawbacks when it takes into the account the cost of fetching the document. Because the above algorithm considers the cost of the document before it replaces the page, it sometimes replaces a page that has been accessed recently but doing so it also saves the cost for the new document.

4 Performance Analysis

As in [2] [6] [5], we used the discrete event simulation to compare the performance of the proposed replacement policy against the economic-based policy [5]. As in [2][6], we used the hit ratio and the byte hit ratio to measure the effectiveness of the cache replacement policies.

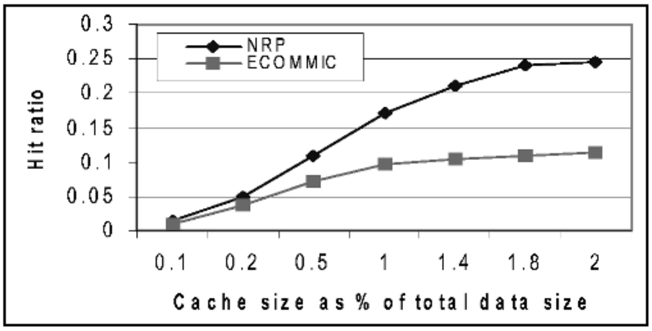

Fig. 1. An overview of the data grid model used in the paper.

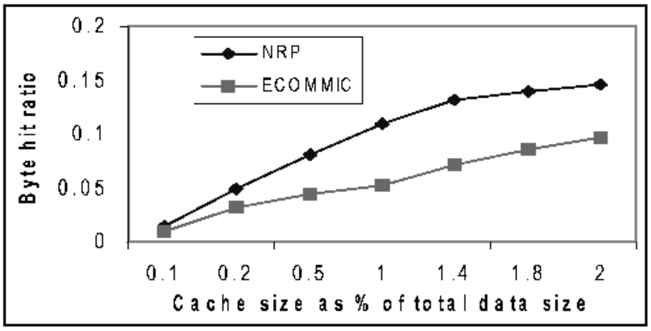

Fig. 2. An overview of the data grid model used in the paper.

4.1 Experimental Setup

The simulator is a modified version of [8] that also accounts for the latency incurred at the source of the file; the transfer delay in reading the file into the disk cache; and the holding or pinning delay incurred while a user processes the file after it has been cached. The simulator takes a stream of requests as an input, calculates the hit ratio and byte hit ratio for each algorithm, under cache sizes being various percentages of the total data set size. The data for the simulation is based on [2]. The sizes of files ranged from about 1.0 to 2.1 gigabytes and the time scales are in the order of minutes. We use a Poisson inter-arrival time with mean 90 seconds for the requests. The file sizes are uniformly distributed between 500,000 bytes and 2,147,000,000 bytes. The entire period of request generation is broken into random intervals and we inject locality of reference using the 80-20 rule, which within each interval of the request stream, 80% of the requests are directed to 20% of the files. The length of an interval is uniformly distributed between 1% and 5% of the generated workload.

As in [5], we used the European DataGrid testbed sites and the associated network performance. In the simulation 20 sites were used with the total disk capacity of 1.1 terabits distributed in such a way two sites have a disk of 100 GB each while the remaining 18 sites have 50 GB each. The two sites with the 100 terabit disk are used to store the original copy of the files in the system. A single copy of each data file was initially placed at CERN. The storage facilities of each site were set at a level to prevent any site (except CERN) holding all of the files, but allowing the site room for the preferred job files.

4.2 Results and Discussions

Figure 1 show the results of the experiments. The graphs are generated from runs using variance reduction technique. Each workload has just over 500,000 entries. For each specified value of the cache size and for each workload, a run generates 5 values of the required performance metric, at intervals of about 100,000 requests. Each point in the graphs, for any particular measure, is the average of the 5 recorded measures in a run. Figure 1 shows the performance of the two policies as a function of the hit ratios.

Figure 2 shows the performance of the two policies as a function of the byte hit ratios. From the data on these figures, it can be observed that the proposed policy substantially outperforms than the economic-based policy. This can be explained by the fact that eviction in the economic-based policy is driven by probability that a file can be used many times. Also, there is a tendency in the economic-policy for bringing in a file even if there is no space to accommodate it while our policy do not. Moreover, our policy was able to simply transfer one-timer requests without storing them in the cache while the economic-based policy did not.

5 Conclusions and Future Direction

A replacement policy is applied to determine which object in the cache needs to be evicted when space is needed. As a replacement policy is an essential component of the disk cache management system, we have presented a new policy and shown through simulation that it performs substantially better than an existing one. Future work in this area would involve more extensive testing with real workloads from other mass storage systems. Also, we will include the policies proposed in [2][66]. In this paper, we assumed that SRM serves user requests based on FCFS with local requests having priority over remote requests. We are currently studying the impact of different request scheduling on the performance on the cache replication policies.

References

1. Chervenak, A., Foster, I., Kesselman, C., Salisbury, C., Tuecke, S.: The Data Grid: Towards an architecture for the distributed management and analysis of large scientific data-sets. Journal of Network and Computer Applications (2000) 187-200
2. Otoo, E.J., Olken, F., Shoshani, A.: Disk Cache Replacement Algorithm for Storage Resource Managers in Data Grids. In Proceedings of the SC (2002) 1-15
3. EU Data Grid Project. http://www.eu-datagrid.org.
4. Abawajy, J. H.: Placement of File Replicas in Data Grid Environments. In Proceedings of PParGMS Workshop (2004)

5. Carman, M., Zini, F., Serafini, L., Stockinger, K.: Towards an Economy-Based Optimisation of File Access and Replication on a Data Grid. In Proceedings of 2nd CCGRID (2002) 120-126
6. Jiang, S., Zhang, X.: Efficient Distributed Disk Caching in Data Grid Management. In Proceedings of Cluster Computing (2003) , Hong Kong, China.
7. Hoschek, W., Jaén-Martínez, F. J., Samar, A., Stockinger H., Stockinger, K.: Data Management in an International Data Grid Project. In Proceedings of the 1st Workshop on Grid Computing (2000) 77-90
8. P. Cao and S. Irani, "Cost-aware WWW proxy caching algorithms," In USENIX Symposium on Internet Technologies and Systems, 1997.