An improvement in LVCT cache replacement policy for data grid

Jagdish Prasad Achara[1][2], Abhishek Rathore[2], Vijay Kumar Gupta[2] and Arti Kashyap[2]

[2] *LNMIIT, Jaipur, India*

E-mail: `jagdish.06@lnmiit.com, arathore.06@lnmiit.com, vijay.06@lnmiit.com, arti.kashyap@lnmiit.com`

Caching in data grid has great benefits because of faster and nearer data availability of data objects. Caching decreases retrieval time of data objects. One of the challenging tasks in designing a cache is designing its replacement policy. The traditional replacements policies, based on LRU and LFU algorithms are not suitable for the data grid because of large-and-varying size and retrieving cost of data objects in data grid. Replacement policies used in data grid are based on calculating utility function for each file which takes care of size and retrieving cost of the file along with locality strength. Lesser the value of locality strength for a file, it is better to evict that file. One of the potential replacement policies, i.e. Least Value Based on Caching Time (LVCT), estimates the locality strength in the utility function using "Caching Time (CT)" where CT for a file F is defined to be the sum of size of all files accessed after last reference to the file F. Here, we propose that number of files accessed after last reference to the file F should also be considered along with CT in the estimation of locality strength so that a better decision can be made for the eviction of a file.

13th International Workshop on Advanced Computing and Analysis Techniques in Physics Research - ACAT 2010
Jaipur, India
February 22–27 2010

[1] Speaker

1. Introduction

A large amount of data generated from various modern scientific experiments is hard to store at a single place. This led to the concept of data grid which is a network of geographically distributed platforms of high performance heterogeneous computational nodes and data storage resources [1].

For the purpose of integration of data, we need a service to manage and retrieve data efficiently in data grid. Data intensive applications are efficiently supported through middleware services in data grid. Storage Resource Manager (SRM) is a middleware that facilitates the sharing of data and storage resources in data grid [2]. When a request for a particular data is made to SRM by a client, it's the job of SRM to fetch the data efficiently to the client. Sometimes, a client may request the same file after a short period of time, so, disk cache functionality is used in SRM so that it doesn't need to go to the source of data each and every time a request is made by the client. Disk cache mechanism plays an important role in SRM by saving network bandwidth and providing faster availability of data.

Cache replacement policy has a significant role in disk cache because it decides which set of files are to be evicted when space is needed in the disk cache and whether a newly arrived file should be cached or not. We cannot use traditional page replacement policies like LRU, LFU in data grids because they keep track of access pattern only. In data grid, data objects have large-and-varying size and retrieving cost. So, these attributes should also be considered in cache replacement policy along with access pattern.

An effective replacement algorithm works because of existence of locality. In our context, frequently accessed files have more locality strength. It is known that an optimal algorithm chooses the object with the smallest locality strength for eviction with fixed sizes and uniform retrieving cost. For caching files with varied sizes and a non-uniform retrieving cost, locality strength of a file is combined with its size and retrieving cost in a utility function to determine its benefit of being kept in the disk [3].

So, a new type of replacement algorithms were developed based on utility function where

Utility function = Locality Strength Estimator * Retrieving Cost / File Size

It's very easy to know file sizes and to estimate their retrieving costs but the most challenging task in implementing this utility function is to estimate locality strength on-line by correctly interpreting history access patterns. In the three factors of a utility function, locality strength is the most critical one, which determines cache hit ratio, while the other two factors are mainly related to the amount of gain from the hits such as responsive time reduction and network bandwidth savings [3].

2. Existing Replacement policies in Data Grid:

Cache replacement policies have been extensively studied in the past and many policies are devised. These replacement policies can be broadly classified in two categories (i) policies without using utility function and (ii) policies based on utility function.

Some of the major replacement policies, without utility function, are depicted below:
Least-Recently-Used (LRU): evicts the set of file(s) which are not used for the longest period of time.
Least-Frequently-Used (LFU): evicts the set of file(s) which are used least often.
Size [4]: evicts the set of file(s) which has largest size.
Log (Size) + LRU [5]: evicts the set of file(s) which has the largest log (Size) and is the least-recently used file among all files with same Log (Size).
The replacement policies where the concept of Utility function has been used are:
Hybrid Algorithm [6]: uses $(nref_i)^{Wn}$ as the locality strength estimator, where $nref_i$ is the number of references to file i since it last entered in the cache. W_n is a constant.
Greedy-Dual Size (GDS) [7]: It's an extension to LRU, which also incorporates varied retrieving cost and file sizes in it. Its locality strength estimator for a file is essentially the inverse of the number of missed files since its last access.
Least Value based on Caching Time (LVCT) [3]: In this policy, locality strength estimator is the inverse of 'Caching Time (CT)' where CT of a file F is the sum of size of all files accessed after last reference to the file F.

3. LVCT Cache Replacement Policy and scope of improvement:

Estimation of 'Caching Time' in this policy is based on Timescale Relativity Principle. This policy also takes care of whether a newly arrived file should be entered in the cache or not whereas all previous replacement policies [2, 4-8] admit each and every new file in cache without considering its eligibility. This avoids wrong file admission in the cache. Once a wrong file is admitted into disk, it could remain in cache for a long time even if it would not be accessed any longer because of its recent access.

In LVCT, when the value of 'Caching Time' increases, the value of the utility function decreases for this policy if the ratio of two other parameters remains the same. This is because of the accumulated size of the accessed files and the disk cache size to measure the probability of in-cache file re-use. The value of the utility function is directly proportional to 'File Cost' because it's better to evict the file which takes less time to transfer (because of bandwidth, distance etc.) as compared to some other file if the size of both files is the same. However, it's better to keep the file of less size in cache as compared to some other file if both of them have the same cost to retrieve i.e. 'File Cost'. This is why; utility function is inversely proportional to size of the file.

$$\text{Utility function for LVCT} = \frac{1}{\text{Caching Time}} \times \frac{\text{File Cost}}{\text{File Size}}$$

This policy showed better performance compared to previous policies [2, 4-8], but it did not take care of the cases where CT computed in the policy is almost the same but the number of files accessed after last reference to the file F differs largely. For example, two cases where (i) number of files is large but sizes of these files are small and (ii) number of files is small but sizes of these files are big after last reference to a file; can result into nearly same value of CT and hence same value of locality strength. However, locality strength of first case should be low compared to second case because of large number of files accessed after its last reference having the same value of CT for both. For more information on LVCT cache replacement policy, please, refer to [3].

4. Proposed improvement in LVCT policy:

LVCT evicts file based on size of accessed files only because time advances at a slower rate for accessing files with small sizes than accessing files with large sizes [3]. To formulate this fact, locality strength estimator in LVCT is the inverse of 'Caching Time'. We propose that if we take the number of accessed files after last reference to a file in consideration along with 'Caching Time', then, locality strength will also incorporate frequency of access. Incorporating frequency of access with 'Caching Time' in locality strength estimation would result in better performance.

We are illustrating one of such cases where a large slot of small files with initial file, say F1, and after that, a small slot of large files with initial file, say F2, are accessed as shown below in Fig. 1.

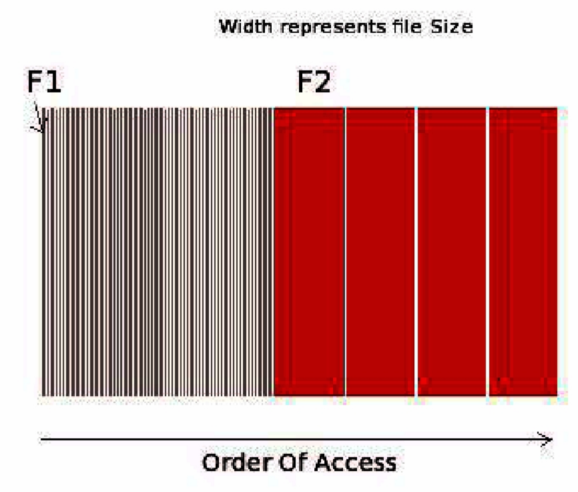


Fig.1

In this case, F1 and F2 will have comparable 'Caching Time' in LVCT although a large number of files have been accessed between F1 and F2. Since the files accessed in between F1 and F2 are of comparably smaller size than files accessed after F2, they will not have significant effect on 'Caching Time'. In such cases, utility function of F1 will become more than F2 because of large file size difference in F1 and F2. This will hide the effect of 'Caching Time' in estimation of utility function.

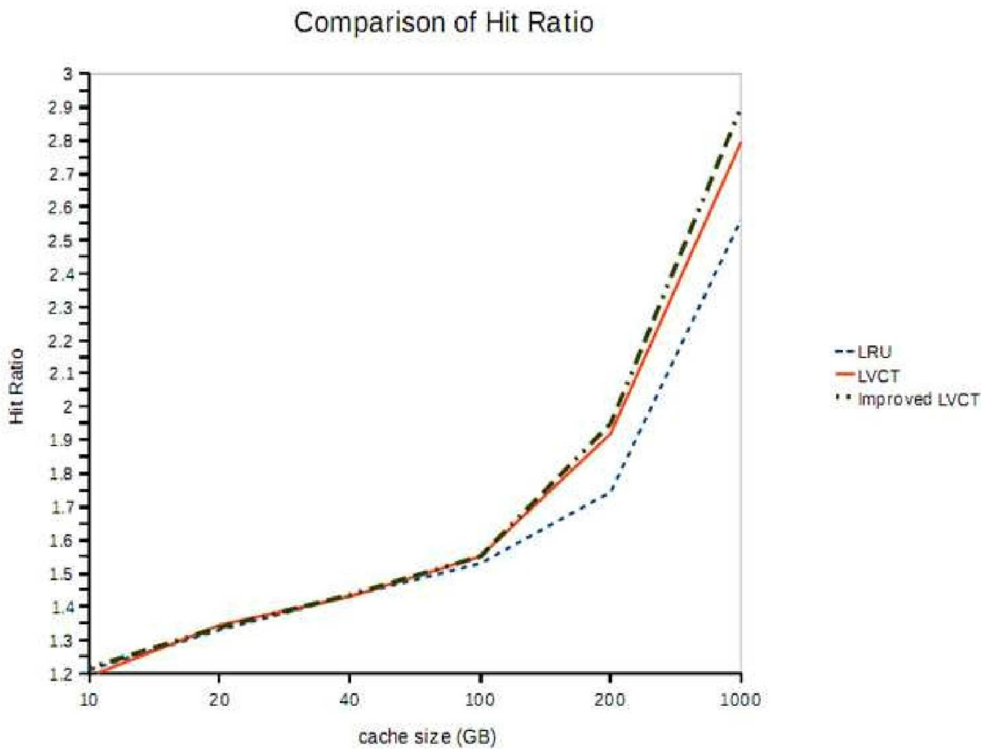
Here, we propose to incorporate how recently a file is accessed in addition to 'Caching Time' estimation. To formulate this, we define a parameter N_f as number of accessed files after last reference to a file f. In our policy **ILVCT** (Improved-Least Value based on Caching Time), locality strength estimator is the inverse of product of N_f and 'Caching Time'. So, utility function for our policy ILVCT is as shown below:

$$\text{Utility function} = \frac{1}{N_f \times \text{Caching Time}} \times \frac{\text{File Cost}}{\text{File Size}}$$

Because of the better estimation of the locality strength, we expect ILVCT to perform better than LVCT especially when there is a large variation in the size of files accessed.

5. Results and Discussion:

We use hit ratio and byte-hit ratio for comparison between LVCT and ILVCT. **Hit ratio** is the ratio of number of cache hits (when a request is satisfied by the cache, it's called a cache hit; and when a request is not satisfied by the cache, it's called cache miss) to number of total requests made to the cache (cache hits + cache misses). Similarly, **byte-hit ratio** is the ratio of the size of files of cache hits to the total size of all files whose requests are made to the cache. Sometimes, hit ratio and byte hit ratio are also defined in percentage, but, here, they are not in percentage.



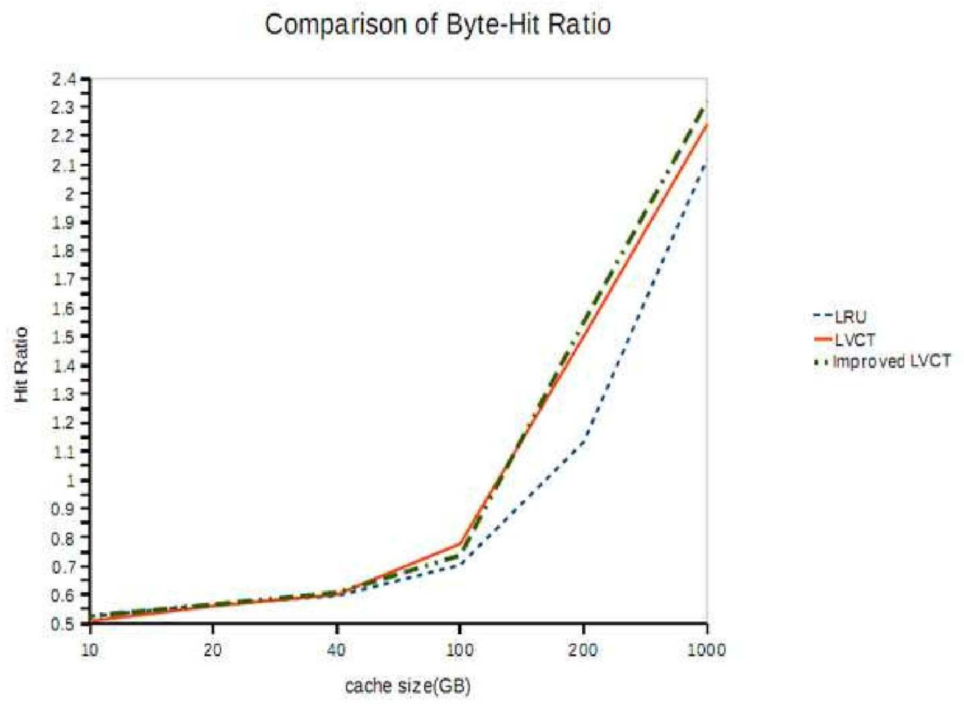


Fig. 2: Hit Ratio and Byte Hit Ratio curves of simulation on data from JLab workload trace for LRU, LVCT and our Improved LVCT

For this purpose, we have made a simulator to test the cache replacement policy. The real workload trace was taken from Jefferson's National Accelerator Facility (JLab), which reflects representative data grid access activities for a period of 4 months (October 2009-January 2010). The size of files accessed for this period of 4 months was more than 1000TB. We have taken the cache size of 10, 20, 40, 100, 200 and 1000GB for replacement policy simulation.

Both hit ratio and byte hit ratio curves for ILVCT show slightly better performance than LVCT. This proves that ILVCT is better than LVCT cache replacement policy.

6. Conclusion:

As depicted by simulation results, the values of hit ratio and byte-hit ratio for ILVCT is more than LVCT cache replacement policy for almost all disk cache sizes. The magnitude of improvement of these ratios for ILVCT with LVCT increases as the size of disk cache increases because of the increase of cases where there is a large variation in the size of files. At the same time, it's clearly visible from the Fig. 2 that this improvement is not so huge. This improvement is not as pronounced as expected due to very low occurrence of cases where subsequently accessed files have large difference in size in this real workload trace from JLab, which was basis for our proposed improvement.

Acknowledgement: We thank Christopher Larrieu and David Lawrence from JLab for providing us real workload trace. We are very grateful to Stephane Genaud from Strasburg University for guiding us throughout this work.

References

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

[2] Ekow Otoo, Frank Olken and Arie Shoshani, " *Disk Cache Replacement Algorithm for Storage Resource Managers in Data Grids*", Proceeding of IEEE Conference on Super-Computing , 2002

[3] Song Jiang and Xiaodong Zhang, " *Efficient distributed disk caching in data grid management*", Proceedings of IEEE International Conference on Cluster Computing, 2003.

[4] S. Williams, M. Abrams, C.R. Standbridge, G.Abdulla and E.A. Fox. *Removal Policies in Network Caches for World-Wide Web Documents.* In Proceedings of the ACM Sigcomm96, August, 1996, Stanford University.

[5] M. Abrams, C.R. Standbridge, G.Abdulla, S. Williams and E.A. Fox. *Caching Proxies: Limitations and Potentials*. WWW-4, Boston Conference, December, 1995.

[6] R. Wooster and M. Abrams, " *Proxy caching that estimates page load delays*", Proceedings of 6th international World Wide Web Conference, April 1997

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

[8] P. Lorensetti, L. Rizzo and L. Vicisano, " *Replacement Policies for a Proxy Cache*", http://www.iet.unipi.it/luigi/research.html