CFS: A New Dynamic Replication Strategy for Data Grids

Feras Hanandeh[1], Mutaz Khazaaleh[2], Hamidah Ibrahim[3], and Rohaya Latip[3]
[1]Prince Al- Hussein bin Abdullah II Faculty of Information Technology, Hashemite University, Jordan
[2]Irbid College, Al-Balqa Applied University, Jordan
[3]Faculty of Computer Science and Information Technology, University Putra Malaysia, Malaysia

Abstract: *Data grids are currently proposed solutions to large scale data management problems including efficient file transfer and replication. Large amounts of data and the world-wide distribution of data stores contribute to the complexity of the data management challenge. Recent architecture proposals and prototypes deal with dynamic replication strategies for a high-performance data grid. This paper describes a new dynamic replication strategy called Constrained Fast Spread (CFS). It aims to alleviate the main problems encountered in the current replication strategies like the negligence of the storage capacity of the nodes. The new CFS strategy enhanced the fast spread strategy by concentrating on the feasibility of replicating the requested replica on each node among the network.*

Keywords: *Grid computing, dynamic replication strategies*.

Received March 1, 2010; accepted May 20, 2010

1. Introduction

Data management is one of the key features of a data grid where large amounts of data are distributed and/or replicated to remote sites, potentially all over the world. In general, a data grid needs to provide features of a pure computational grid [7]; (resource discovery, sharing etc.,) as well as more specialized data management features like dynamic replication strategies for a high performance data grid which is the main focus of this article.

With respect to replication, there are two types of files in a data grid: “master” files and “replicas”. A replica is any copy of a file other than the master. The master file is owned and managed by the creator of the file, but the replicas are managed by the grid (middleware). For example, a storage element may delete unused replicas to make space available for new replicas without notifying the owner of the file. The use of replicas is transparent to users; they are created as needed by the grid middleware in order to improve overall performance of jobs. However, sites can explicitly ask for the creation of replicas locally. Initially, replica files are by definition read-only; read-write implies the creation of a new master file. This is to avoid the extremely difficult synchronization problem of allowing users to write to multiple replicas of the same file. Master files would typically be stored on a "reliable" system, (i.e., backed up), whereas a replica does not require backup. A simple example of replica usage is as follows: To improve the performance of a data grid job to be run at site A, data in permanent storage at site B is copied to site A. This data may then be used by subsequent jobs at site A, or by jobs at site C, which has a better network connection to site A than site B. For this reason, the data should be kept at site A as long as possible. However, there is no need to store this file permanently at site A, because the file can always be retrieved from site B. The replica manager keeps track of all replica data so that the replica selection service can select the optimal physical file to use for a given job, or to request the creation of a new replica. Replica usage can be thought of as a type of long-term cache, where the data remains in the cache for use by future jobs until the cache is full, in which case the least recently used files are removed, subject to their "lifetime" attributes.

Both master and replica file include the notion of a "lifetime". Master files may be given a finite lifetime so that they can be deleted automatically by the system. Replicas may always be deleted by the system, but they may also be assigned a lifetime so that they are not deleted too soon. A replica lifetime might be set manually by a user who knows the same file will be used for a series of jobs, or it could be set by the scheduler.

Replicas are currently defined in terms of files and not objects. The initial focus is on the movement of files, without specific regard for what the files contain. We realize that many users are mainly interested in objects. However, we believe that there are well defined mechanisms to map objects to files for both objectivity and root, and that all of this will be completely transparent to the applications. However, achieving this transparency will require close interaction with the applications' data model. In the case of most other commercial database products, it appears that this is difficult to do efficiently, and requires additional study.

The article is organized as follows. In section 2, we outline briefly previous work that influenced our new strategy. The methodology of this article is presented in section 3. In section 4, the proposed algorithm is presented and discussed. We present experimental results and discussions in section 5. Section 6 contains a summary and our conclusions from this effort.

2. Previous Work

The European Data Grid project (EDG), one of the largest data grid projects today, have a main focus on providing and deploying such data replication tools. Although the project officially started in January 2001, prototype implementations started already in early 2000 and first data management architecture was presented in [7]; thus, within the project there is already a well-established experience in providing replication tools and deploying them on a large-scale testbed.

Since interoperability of services and international collaborations on software development are of major importance for EDG as well as other grid projects in Europe, the U.S. etc., the first set of data management tools (i.e. replication tools) provided and presented here, are based on established de-facto standards in the Grid community. In addition, for parts of the software presented here, EDG has development and deployment collaborations with partner projects like the Particle Physics Data Grid collaboratory pilot (PPDG), TransAmerica Grid (TAG) project and LHC Computing Grid (LCG).

Caching frequently accessed data in a mobile client's memory has been proposed to improve the performance of the various systems by increasing the availability of data in the presence of disconnectivity, reducing the data retrieval from the original server, relieving the bandwidth consumption, and reducing the latency in data access [15]; In mobile environment, caching classified into two categories, the Stateless approaches [1, 3, 4, 8, 10, 12, 13, 15, 16, 17, 18] and stateful schemes [9, 15].

In most of the grid computing researches, researchers implemented and evaluated six different strategies. This helped demonstrate what the simulator is capable of doing, as well as helped them understand the dynamics of a grid system better. In this paper, we distinguish between caching and replication. Replication is assumed to be a server side phenomenon. A server decides when and where to create a copy of a file it has. It may do this randomly or by recording client behaviour or by some other means. But the decision to make a copy (replica) and send it to some other node is taken solely by the server. Caching is defined as a client side phenomenon. A client requests for a file and stores a copy of the file locally for future use. Any other nearby node can also request for that cached copy.

The first strategy is no replication or caching. In this strategy no replication takes place. The entire data set is available at the root of the hierarchy. Any request for some files from any node, the file will be called from the root. The second strategy is the best client. The criterion that this strategy adopts is the history of files [6]; where the number of requests for those files is kept. The number of request will be compared with a specified threshold. If it exceeds the threshold it considered as a best client. The best client is the one that has generated the most number of requests for that file. The third strategy is cascading replication. In this strategy, the free space is considered in each node. Once the capacity of a specific node becomes less than the specified threshold then a replica is created and sent to the node at the next level taking into consideration that the replica is created to be placed at the path to the best client. The fourth strategy is plain caching. The client that requests a file stores a copy locally. Since these files are large and a client has enough space to store only one file at a time, the files get replaced quickly. This strategy is straight forward. The fifth strategy is caching plus cascading replication. This combines strategy three and four. The sixth strategy is fast spread. In this method, a replica of the file is stored at each node along its path to the client [5]; That is, when a client requests a file, a copy is stored at each tier on the way. This leads to a faster spread of data. When a node does not have enough space for a new replica it deletes the least popular file that had come in the earliest.

In such strategies, this might delete a relatively new file that has just come in and not yet been requested for. It might hold potential to become popular in the future. Thus there needs to be a measure of time and hence the age of each file in that cache. The replacement strategy we employed takes care of both these aspects and is a combination of least popular and the age of the file. If more than one file are equally unpopular, the oldest file is deleted. One detail to be noted here is that the popularity logs for all the files are cleared periodically. Thus the dynamic aspect of changing user patterns is captured.

3. Preliminaries

This article uses the PARallel Simulation Environment for Complex systems (PARSEC) simulator downloaded from parsec home page, "http://pcl.cs.ucla.edu/projects/parsec". The reason why we use the PARSEC is its ability to execute a discrete-event simulation model using several different asynchronous parallel simulation protocols on a variety of parallel architectures. The simulator was built to formulate the data grid and the transference of the data among it. The data structure adopted is the tree structure were only one shortest path is considered. The data grid consists of three levels as depicted in Figure 1. The responsibility of producing data is the top most level (the root).

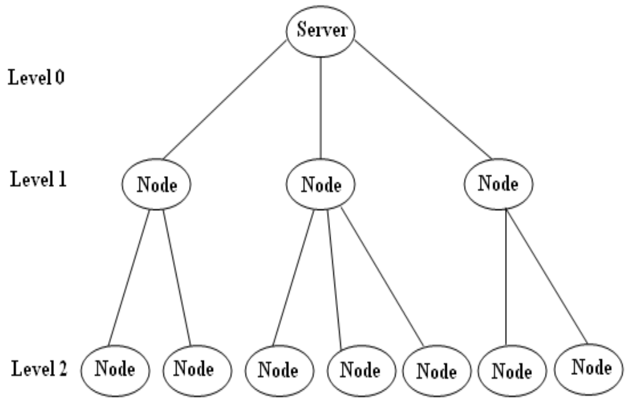


Figure 1. Three level tree structure data grid.

The storage capacity at each level is given in Table 1. All network links have the same bandwidth which is 320 Mbytes/Sec. The files have a uniform size equal to two gigabytes each.

Table 1. The storage capacity at each level.

Level	Storage Capacity (GB)
Level 0	2000
Level 1	1000
Level 2	500

The experiments were run on three different kinds of access patterns:

- P-random: Random access patterns. No locality in patterns.
- P1: Data, which contained a small degree of temporal locality.
- P2: Data containing a small degree of geographical and temporal locality.

Locality is varying from 0 to 1, where 0 indicates that the requests are completely random and there is no locality. At the other end of the spectrum, when locality is 1, it means all the requests are for the same file. Tests were also conducted with data of varying degrees of locality and the results are in section 5.

4. Constrained Fast Spread Algorithm

The main concentration of our approach is the consideration of two criterions, the size of the Requested Replica (RR) and the number of requests of the RR. A ratio is calculated to produce the Partial Number Of Requests (PNOR). PNOR is the ratio of the Number Of Requests (NOR) to the shortage of the requested space to accommodate the RR.

In Constrained Fast Spread method (CFS), a replica is verified to be stored at each node along its path to the client. This leads to a constrained faster spread of data when a node does not have enough space for a new replica; it verifies the number of requests of each file. A comparison process is performed to compare the number of requests of the files in each node. Summations of the sizes of some of replicas are compared with the size of the requested replica. If the space allocated by the replica is sufficient to be allocated by the requested replica then all the replicas will be deleted and the requested replica replaces them. Figure 2 presents the algorithm to check whether the newly RR deserves to be copied to the Requested Node (RN) based on the criterions mentioned above or not*. The definitions of pseudo code variables used in the CFS algorithm listed in Table 2.

Table 2. Definitions of pseudo code variables of CFS algorithm.

Variable	Definition
RR	Requested replica
RN	Requesting node
CNFSS	Checked node's free storage space
NOR	Number of requests of RR
PNOR	Partial number of requests of RR which is the ratio of the NOR to the shortage of the requested space to accommodate the RR
SOS	Sum of sizes, where this variable contains the sum of sizes of a group of replicas on the checked node to be replaced by the RR
NSPList	The list that contains the nodes on the shortest path from RN +1 to the main server, where RN +1 is the parent of RN
ReplicaList	The list that contains the existing replicas on the checked node sorted in non-increasing order based on their sizes. If two or more replicas have the same size, these replicas are sorted in non-decreasing order based on their number of requests. If they also have the same number of requests, they are sorted randomly
SizeList	The list that contains the sizes of the corresponding replicas in ReplicaList
NORList	The list that contains how many times each replica in ReplicaList has been requested by the current node

If the process of verifying the existence of the RR at the RN finds that the RR exists then the replica will be used and no more verification will be performed. Otherwise, the process proceeds to verify whether the checked node's free storage space is enough to accommodate the RR. If the storage space is enough to be replaced by the RR then the replica will be copied and used and no more verification will be performed.

Otherwise, the process proceeds to compute the PNOR and to be compared with the NOR of the accommodated replica(s). Based on the results one can report that the RR will replace the accommodated replica(s) or not.

* Different results has been developed and evaluated in [2]. However, in this paper, a new enhancement of the fast spread algorithm was evaluated under different scenarios which represents a new contribution.

Pseudo code 1: CFS Algorithm

```
Initialize SOS to 0;
If RR exists on RN then
        Use RR;
Else
  For i = 1 to NSPList.size do
     If RR exists on NSPList(i) then
        For j = NSPList(i -1) to 1 do
           If CNFSS ≥ RR.Size then
              Copy RR;
           Else if CNFSS < RR.Size then
              $PNOR = NOR * \frac{RR.Size - CNFSS}{RR.Size}$
              For x = 1 to ReplicaList.size do
                 If SOS < RR.Size - CNFSS then
                    SOS = SOS + SizeList(x);
                 Else
                    Break;
                 End
              End
                 If $\sum_{y=1}^{x-1}$ NORList(y) < PNOR then
                 For y = 1 to x - 1 do
                  Delete ReplicaList(y), SizeList(y), NORList(y);
                 End
                 Copy RR;
              End
           End
        End
     End
  End
End
```

Figure 2. The proposed CFS algorithm.

5. Experimental Results and Discussions

We compare the results of four of the experiments, no replication, caching plus cascading, fast spread and CFS strategy. The experiments were run on the three access patterns P-random, P1 and P2.

5.1. P_Random

P-random: Is random access pattern, in order to evaluate the effect of replication strategies, the three replication strategies using P-random pattern listed in Table 3 were compared to the base case of no replication. Table 3 lists the improvement ratio in response time and bandwidth savings.

Table 3. Improvement in response time and bandwidth savings for random data as compared to the base case of no replication.

Replication Strategy	Improvement in Response Time	Bandwidth Savings
Cascading Plus Caching	34%	30%
Fast Spread	31%	29%
CFS	29%	27%

From the results above, it is clear that cascading plus caching is the best among other strategies including CFS using P-random pattern. The reason of that is the random frequent propagation of the popular files to the clients which improves the response time and bandwidth because of the availability of such files regardless the consideration of the space. This consider as drawback of such strategy although the much improvement of response time and bandwidth savings. There is neglect in considering the space which is considered in this article.

5.2. P1

P1: Is temporal locality patterns, the three replication strategies using P1 pattern listed in Table 4 were compared to the base case of no replication. Table 4 lists the improvement ratio in response time and bandwidth savings.

Table 4. Improvement in response time and bandwidth savings for temporal locality data as compared to the base case of no replication.

Replication Strategy	Improvement in Response Time	Bandwidth Savings
Cascading Plus Caching	37%	41%
Fast Spread	34%	41%
CFS	34%	40%

From the results above, it is clear that the improvement of bandwidth is approximately close. At the same time cascading plus caching is still the best in improving the response time among other strategies including CFS using P1 pattern. The reason of that is the temporal frequent propagation of the popular files to the clients concentrating on the time of the request over the available space which improves the response time and bandwidth. This consider as drawback of such strategy although the much improvement of Response time savings. There is neglect in considering the space which is considered in this article.

5.3. P2

P2: Is geographical locality patterns, the three replication strategies using P2 pattern listed in Table 5 were compared to the base case of no replication. Table 5 lists the improvement ratio in response time and bandwidth savings.

Table 5. Improvement in response time and bandwidth savings for geographical locality data as compared to the base case of no replication.

Replication Strategy	Improvement in Response Time	Bandwidth Savings
Cascading Plus Caching	59%	58%
Fast Spread	65%	62%
CFS	66.5%	60%

From the results above, it is clear that the improvement of bandwidth using CFS is approximately close and is more efficient than cascading plus caching. At the same time CFS is the best in improving the response time among other strategies using P2 pattern. The reason of that is the planned availability of the requested files nearby the requested node. Although the CFS concentrates on the spreading the files among the network, it takes into consideration the PNOR to decide whether replacing the requested file or not.

Followings are Figure 3 and Figure 4 which show via graphs the percentage savings in response time and bandwidth, respectively, using the three patterns for the three strategies, cascading plus caching, fast Spread and CFS. Figure 5 shows via a graph the comparison of the response time as a function of q in terms of geographical locality index for Fast Spread and CFS. q is the index used to measure the amount of locality in the patterns, where 0<q<1. If q=0, it means there is no locality, when q=1 it means all the requests are for the same file.

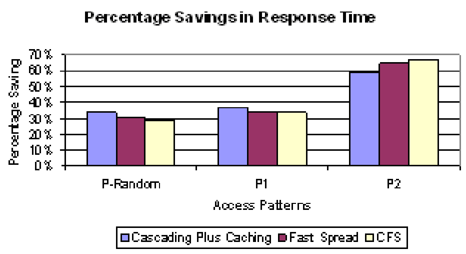


Figure 3. The graph compares the percentage savings in response time for P-random, P1 and P2 for three strategies (cascading plus caching, fast spread and CFS).

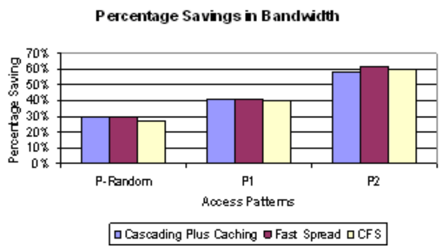


Figure 4. The graph compares the percentage savings in bandwidth for P-random, P1 and P2 for three strategies (cascading plus caching, fast spread and CFS).

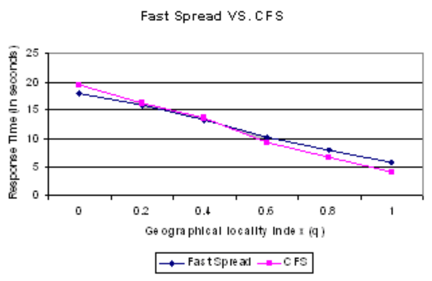


Figure 5. The graph of comparing the response time as a function of q in terms of geographical locality index for fast spread and CFS.

6. Conclusions

This article defined a new dynamic replication strategy that supports, via a network, the use, copy, replacement, and management of requested replicas with dynamic, managed lifetime. The article discussed the need for improving the dynamic replication strategies to manage large data sets in a high performance data grid. Dynamic replication enables faster access to files, decreases the bandwidth consumption and distributes server load. The above can be said about static replication too. The advantage of dynamic replication is that it automatically creates and deletes replicas according to changes in the access patterns. This ensures that the benefits of replication do continue even if user behaviour changes frequently.

We also presented various dynamic replication strategies for this scenario and tested them using the simulator. We generated three different kinds of access patterns, random, temporal, and geographical and showed how the bandwidth savings and latency differ with each kind of access pattern. Three strategies performed the best in our tests: Cascading plus caching, fast spread and CFS. While cascading plus caching worked well for cases when the request patterns were random, fast spread worked better when there was a small amount of locality in the file usage patterns. On the other hand, CSF worked better when there was a geographical locality patterns. We analyzed why we thought these were the best strategies and the constraints of each method.

References

[1] Barbara D. and Imielinski T., "Sleepers and Workaholics: Caching Strategies for Mobile Environments," *in Proceedings of the ACM SIGMOD International Conference on Management of Data SIGMOD*, USA, pp. 1-12, 1994.

[2] Bsoul M., Al-Khasawneh A., Kilani Y., and Obeidat I., "A threshold-based dynamic data replication strategy," *J Supercomput*, Published online: 13 AUGUST 2010. DOI 10.1007/s11227-010-0466-3.

[3] Cao G., "On Improving the Performance of Cache Invalidation in Mobile Environments," *Computer Journal of Mobile Networks and Applications*, vol. 7, no. 4, pp. 291-303, 2002.

[4] Chuang P. and Chiu Y., "Constructing Efficient Cache Invalidation Schemes in Mobile Environments," *in Proceedings of 3rd International IEEE Conference on Signal-Image Technologies and Internet-Based System*, Shanghai, pp. 281-288, 2007.

[5] Fan L., Cao P., Almeida J., and Broder Z., "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol," *IEEE/ACM Transactions on Networking*, vol. 8, no. 3, pp. 281-293, 2000.

[6] Gwertzman J. and Seltzer M., "The Case for Geographical Push-Caching," *in Proceedings of Presented at 5th Annual Workshop on Hot Operating Systems*, USA, pp. 51-55, 1995.

[7] Hoschek W., Jean-Martinez J., Samar A., Stockinger H., and Stockinger K., "Data

Management in an International Data Grid Project," *in Proceedings of 1st IEEE/ACM International Workshop on Grid Computing*, Bangalore, India, pp. 17-20, 2000.

[8] Hou W., Su M., Zhang H., and Wang H., "An Optimal Construction of Invalidation Reports for Mobile Databases," *in Proceedings of the 10th International Conference on Information and Knowledge Management*, USA, pp. 259-269, 2001.

[9] Huang Y., Cao J., Wang Z., Jin B., and Feng Y., "Achieving Flexible Cache Consistency for Pervasive Internet Access," *in Proceedings of the 5th Annual IEEE International Conference Pervasive Computing and Communications*, NY, pp. 239-250, 2007.

[10] Jing J., Elmagarmid A., Helal A., and Alonso R., "Bit-Sequences: An Adaptive Cache Invalidation Method in Mobile Client/Server Environments," *Mobile Networks and Applications*, vol. 12, no.7, pp. 115-127, 1997.

[11] Kahol A., Khurana S., Gupta S., and Srimani P., "A Strategy to Manage Cache Consistency in a Distributed Mobile Wireless Environment," *Computer Journal of IEEE Transaction on Parallel and Distributed Systems*, vol. 12, no. 7, 2001.

[12] Lam K., Chan E., Leung H., and Au M., "Concurrency Control Strategies for Ordered Data Broadcast in Mobile Computing Systems," *Computer Journal of Information Systems*, vol. 29, no. 3, pp. 207-234, 2004.

[13] Lee S., "System and Method for Maintaining Cache Consistency in a Wireless Communication System," *United States Patent*, no. 14, 2006.

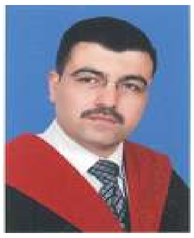
[14] Madhukar A. and Alhajj R., "An Adaptive Energy Efficient Cache Invalidation Scheme for Mobile Databases," *in Proceedings of the ACM Symposium on Applied Computing*, Dijon France, pp. 23-27, 2006.

[15] Shao X. and Shanglu Y., "Maintain Cache Consistency of Mobile Database using Dynamical Periodical Broadcast Strategy," *in Proceedings of the Second International Conference on Machine Learning and Cybernetics*, China, pp. 2389-2393, 2003.

[16] Yi S., Shin H., and Jung S., "Enhanced Cost Effective Cache Invalidation for Mobile Clients in Stateless Server Environments," *Lecture Notes in Computer Science*, vol. 3207, pp. 387-397, 2004.

[17] Yi S., Song W., Jung S., and Park S. "A Cost Effective Cache Consistency Method for Mobile Clients in Wireless Environment," *DASFAA, Lecture Notes in Computer Science*, vol. 2973, pp. 908-915, 2004.

[18] Yuen J., Chan E., Lam K., and Leung H., "Cache Invalidation Scheme for Mobile Computing Systems with Real-time Data," *Computer Journal of ACM SIGMOD Record*, vol. 29, no. 4, pp. 34-39, 2000.



Feras Hanandeh is currently an assistant professor at the Prince Al Hussein Bin Abdullah II Faculty of Information Technology, Al-Hashemite University, Jordan. He obtained his PhD in computer science from University Putra Malaysia, Malaysia in 2006. His current research interests include distributed databases, parallel databases focusing on issues related to integrity maintenance, transaction processing, query processing and optimization; grid computing, artificial intelligence and geographical information systems.

Mutaz Khazaaleh is a lecturer in computer science and information technology at Al-Balqa Applied University, Irbid, Jordan. He holds Master degree in computer science and information from the Yarmouk University, Jordan in 2005. He has more than 8 years of teaching experience.

Hamidah Ibrahim is currently an associate professor at the Faculty of Computer Science and Information Technology, Universiti Putra Malaysia. She obtained her PhD in computer science from the University of Wales Cardiff, UK in 1998. Her current research interests include databases (distributed, parallel, mobile, bio-medical, XML) focusing on issues related to integrity constraints checking, cache strategies, integration, access control, transaction processing, and query processing and optimization; data management in grid and knowledge-based systems.

Rohaya Latip is a senior lecturer at the Technology Communication and Network Department, Faculty of Computer Science and Information Technology. She received her BSc degree in computer science from University Technology Malaysia in 1999. Her MS degree in distributed system in 2001, and her PhD in distributed database in 2009 from University Putra Malaysia. She is a member of IEEE computer society and also an associate researcher at the Laboratory of Computational Science and Informatics, Institute of Mathematical Science (INSPEM), University Putra Malaysia. Her main research interest includes data grid, distributed database, grid computing, and network management.