http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 7 (5): 776-782, 2008
ISSN 1812-5638

Fair-Share Replication in Data Grid

Qaisar Rasool, Jianzhong Li, George S. Oreku and Ehsan Ullah Munir
Harbin Institute of Technology, 6th Dorm, 23 Gongjian Street, Nangang District, Harbin, China

Abstract: We propose a novel approach called Fair-Share Replication (FSR) to balance the work load and storage resource usage of the replica servers in data grid. The approach takes into account the number of requests and the storage load on the candidate nodes before placing replicas in the grid nodes. The data requests are serviced by the siblings as well as the parent node. Experiment results show that FSR performs better than Fast Spread replication technique with respect to effective storage utilization in replication process.

Key words: Replication, data grid, replica placement

INTRODUCTION

Data grid (Chervenak *et al*., 2000; Foster and Kesselman, 2003) is a grid infrastructure of virtual organizations and individuals with specific needs to transfer and manage massive amounts of scientific data for analysis purposes. Replication is an important technique that is used in grid and other distributed systems for the purpose of improving data availability and fault tolerance. In static replication, a replica persists until it is deleted by user or its duration is expired; however, the benefits of replication decrease if access pattern changes randomly. In contrast to static replication, dynamic replication automatically creates and deletes replicas according to changing access patterns and thus ensures that benefits of replication continues even if user behavior changes. In general, a replication mechanism determines which files should be replicated, when to create new replicas and where the new replicas should be placed.

There are many techniques proposed in research for dynamic replication in Grid environment (Ranganathan and Foster, 2001; Abawajy, 2004; Lamehamedi and Szymanski, 2007; Lei *et al*., 2008; Chang and Chang, 2008). These strategies differ by the assumptions made regarding underlying grid topology, user request patterns, dataset sizes and their distribution and storage node capacities. Other distinguishing features include data request path and the manner in which replicas are placed on the grid nodes.

In this research, we use a framework for data grid with specific path for data search and retrieval. It consider that the children under the same parent in the data grid tree are linked in a P2P-like manner. For any client's request, if desired data is not available at client's parent node, the request moves to the sibling nodes one by one until it finds the required data. If none of the siblings can fulfill the request, the request moves to the parent node one level up. Here also all the siblings are probed and if data not found the request moves to next parent and ultimately to root node.

Our proposed replication technique, Fair-Share Replication (FSR), finds the best candidate nodes for placing replicas of popular files as follows. The replicas are always placed at the parent node of client that generates maximum requests. In other words, all the nodes above the client-tier can be potential candidates for hosting replicas. First, the load on each potential candidate is calculated and ranked according to file access frequency. Secondly, any ranked node that has maximum access load and the maximum storage load in the system is removed from the list of candidates. For each file we primarily select the highest ranked node in the list as its best candidate. However if this node has access load greater than that of its immediate sibling then we examine storage load of the node and its sibling. The node which has less storage load is selected as the best candidate to host replica. We studied the fast spread dynamic replication technique for comparison purpose. The results show that FSR performs better than Fast Spread in balancing the storage utilization of Data Grid nodes during the replication process.

SYSTEM MODEL

Several grid activities have been launched since the beginning of 21st century. Large scientific initiatives such as global climate change, high energy physics and computational genomics require collection and management of large amount of data of petabyte scale. To

Corresponding Author: Qaisar Rasool, Harbin Institute of Technology, 6th Dorm, 23 Gongjian Street, Nangang District, Harbin, China

support these scientific applications, data grid technology has been developed. The High Energy Physics (HEP) community, for example, seeks to take advantage of the grid technology to provide physicists with the access to real as well as simulated LHC data from their home institutes. Data replication and management is hence considered to be one the most important aspects of HEP data grids.

The LHC computing grid adopts a hierarchical model. The data generated during an LHC experiment is preprocessed and stored at the Tier-0 facility at the CERN laboratory site. Then, processed data is distributed over high-speed networks to 10-20 national Tier-1 centers in the USA, leading European countries, Japan and elsewhere. The data is there further processed and analyzed and then stored at 60 Tier-2 regional centers, each serving a small to medium-sized country, or one region of a larger country (as in the US, UK and Italy). Data subsets are accessed and further analyzed by physics groups using one of the hundreds of Tier-3 workgroup servers and/or thousands of Tier-4 desktops.

Request path: For a data grid tree, usually clients at the leaf nodes generate data requests. A request travels from client to parent node in search of replica until it reaches at root node. Potentially a request may take various paths in search of data based on the topology of the system (Table 1). Any interior or leaf node can initiate data requests. Initially data is held at the tier-0 and gradually replicated to the lower tiers of the data grid as the frequency of access exceeds a threshold.

If data grid is formed as a tree with no interaction among the nodes at same tier then it means only parent-child relationships exist and thus messages and data transference could take place in upward and downward. However, if data grid hierarchy is structured in such a way to make a plex and/or ring of tier (Lamehamedi *et al.*, 2002) then the request can travel different paths before it can reach the required data. For example, if a client node n has requested for data then we can probe the parent, children and sibling of node n for the requested data. In this research, we assume that data requests are generated only by the clients at the leaves of the data grid tree.

Data grid topology: The data grid topology used in this research is adopted from the study of Lamehamedi *et al.* (2002) with a modification that nodes at client tier are not connected to each other (Fig. 1). Above the client tier, the children of same parent are siblings and can transfer replicas if required. We choose this topology in order to exploit the locality principle. Considering the previous example of tiered data grid, the tier-1 consists of centers

Table 1: Request paths for different grid structures

Grid structure	Connection	Request path
Tree	Parent-child relationships only	Child → parent
Tree+P2P or ring	Parent-child relationship with leaves and/or middle-tier nodes connected with each other	Client → sibling Client → parent
Plex	Parent-child relationship. A child node can have more than one parent node.	Client → parent-1 Client → parent-2 Client → parent-n
Plex+P2P or ring	Parent-child relationship with leaves and/or middle-tier nodes connected with each other. A child node can have more than one parent node	Client → parent-1 Client → parent-2 Client → parent-n Client → sibling
P2P or flat	All nodes equal. No hierarchy and no parent-child relationships.	Client → neighbor-1 Client → neighbor-2 Client → neighbor-n

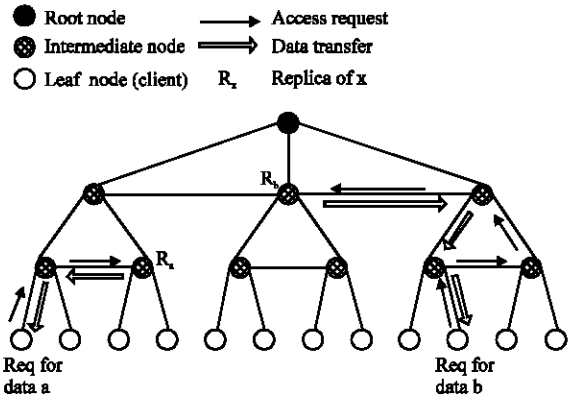

Fig. 1: Data access operation in hierarchical data grid model

which belong to different continents, say, North America, Europe and Asia. The tier-2 is composed of regional centers. For instance, UK and Italy are tier-2 regional centers within the continent Europe. Further, there are workgroups at tier-3, say campuses of Imperial College and Oxford University. Lastly, we have workstations at tier-4. Obviously there is a high correlation among tier-4 and tier-3 nodes which fall under the same tier-2 node. So, in Data Grid we group the related nodes at a tier into siblings. This kind of hierarchical structure has many advantages (Tang *et al.*, 2005). For example, data can be distributed to appropriate resources and accessed by multiple sites. Researchers and scientists can fetch data in a common and efficient way. The network bandwidth can be used efficiently because most of the data transfers only use local or national network resources, hence alleviating the workload of international network links etc. Data requests in this model follow the following pattern. A request moves upward to parent node only after all the sibling nodes have been searched for the required data. The process is as follows:

Step 1: A client c requests a file f. If the file is available in the client's cache then ok. Otherwise step 2.

Step 2: The request is forwarded to the parent of client c. If data is found there, it is transferred to the client. Otherwise request is forwarded to the sibling node.

Step 3: Probe all sibling nodes one after another in search of data. If data is found, it is transferred to the client via shortest path.

Step 4: If data is not found at any sibling node, the request is forwarded to the parent node and the step 3 is repeated.

Step 5: If data is not found in step 4 then step 4 continues until the request reaches at root.

FAIR-SHARE REPLICATION STRATEGY (FSR)

Here, we describe our proposed replication scheme, Fair-Share Replication (FSR). To satisfy the latency constraints in data grid, there are two ways: one is to vary the speed of data transfer and other is to shorten the transfer distance. Since bandwidth and CPU speed are usually expensive to change, shortening transfer distance by placing replicas of data objects closer to requesting clients is the cheapest and essential way to ensure faster response time.

The main idea of FSR to identify best candidate nodes for replica placement primarily based on access load and if access load of the best candidate is equal to its sibling node then select candidate node based on storage load. For convenience, we label the client tier nodes as c and nodes above the client tier as p. The access load of a node p is equivalent to the workload of node p incurred due the number of requests contributed by its children. For instance, if node p has three children then access load of p will be the cumulative access loads of all three children nodes.

The Grid Replication Scheduler (GRS) is the central managing entity for FSR scheme. At each cache node and the root, a Replica Manager is held that stores information about requested files and the time when requests were made. This information is accumulated and communicated to GRS into a global workload table, G (arrival time, clientid, fileid). An entry in table G depicts that at time arrivaltime, a client clientid has requested a file fileid. The GRS holds a Replica Catalog in the system that is used to register replicas when they are created and placed at the selected Grid nodes. The Replica Catalog stores the mapping from the logical file name to the physical file name.

Replica creation: A replica management system should be able to handle a large number of replicas and their creation and placement. Like previous dynamic replication strategies, we base the decision of replica creation on the data access frequency. Over the time, the GRS accumulates access request history in the global workload table G. At interval, the table G is processed to get a cumulative workload table W(fileid, clientid, freq) where freq stands for number of accesses. This table W is used to trigger replication of requested files. First, GRS calculates the average access frequency, $freq_{avg}$, from the table W. Then, the files which have access frequency greater than or equal to $freq_{avg}$ are marked for replication. If n is the number of entries in the table W then average access frequency is:

$$Freq_{avg} = \frac{\sum freq}{n}$$

The GRS maintains all the necessary information about the replicas in Replica Catalog. Whenever the decision is made to initiate replication, the GRS registered the newly created replicas into the replica catalog along with the information of their creation time and the hosting nodes.

Replica placement: Replica placement is an important phenomenon in any replication technique. Deciding on the right number of replicas and their locations to meet some performance goals in dynamic large-scale systems with different network characteristics and resources and changing user behaviors is challenging. It has been shown that determining how many replicas to create and where to place them in a distributed system in order to meet a performance goal is an NP-hard problem, therefore all the replica placement approaches proposed in the literature are heuristics that are designed for certain systems and workloads (Loukopoulos and Ahmad, 2000). Replication is performed periodically and at times when the system is idle or at its low peak. Experiments from HEP have shown that request for the execution of jobs and thus need for data increase at certain times around 3 pm European time (Elghirani *et al.*, 2007). Therefore it is quite natural to use heuristics and meta-heuristics for replication when the system is either idle or not overloaded.

From the access statistics, the GRS decides which replicas need to be created and where to place them. The decision of replica placement is made in the following way (Fig. 2). At interval, for each file in the table W, the GRS finds the client nodes who have requested the files. Assume that any three clients c_1, c_2 and c_3 have accessed the file f_i 10, 12 and 6 times, respectively. Also suppose that the parent nodes of c_1, c_2 and c_3 have access load 35, 48 and 17, respectively. Since the client c_2 has greater access frequency for file i than other clients, therefore GRS will rank its parent as p_1. Likewise, parents of the clients c_1 and c_3 will be ranked as p_2 and p_3. Note that,

```
Input:  (1) Static: Data Grid topology, link bandwidth, nodes' capacity,
            data set size
        (2) Dynamic: workload in terms of number of requests, current
            time
Output: Number of replicas and optimal locations for placing replicas in
        such a way to balance the replica servers' workload and storage
        usage

/* Deciding which files to replicate */
For each file fi in the access load table, W(clientid, fileid, freq)
    If freq(fi) > = freqavg
      Mark the file fi as to be replicated
    End-if
End-for

/* Selecting the best candidate nodes for placing file replicas */
For each file fi marked to be replicated
  Find p nodes who have received requests for file fi
  Rank p nodes in descending order of freq(fi), i.e., p1, p2, p3, .., pk
  Select p1, p2, p3, .., pk as potential candidate replica servers for file fi
  If replica of file i already exists at p1
        Update CT (i, t)
        Skip to end
  End-if
  Deselect nodes among p1, p2,.., pk having access load = almax and storage
  load = slmax
  Let pk be the selected node of highest rank
  If sibling(pk) does not exists
     bci = pk
  Else
     If al(pk) > al(sibling(pk)) and sl(bci) > sl(sibling(pk))
        bci = sibling(pk)
     Else
        bci = pk
     End-if
End-if
If Available-Space(bci) < Size(fi)
      Evacuate
  End-if
  Replicate(fi, bci, t)
End-for
```

Fig. 2: FSR replication algorithm

rank-wise, p_1 is greater than p_2 and so on. Now suppose that 55 is the maximum access load, al_{max}, (of any parent node) in the system. Since none of p_1, p_2 and p_3 has the access load equal to al_{max} therefore p_1, p_2 and p_3 are the potential candidates for holding the replica of file f_i. The GRS shall make a tentative decision that p_1 having the highest rank, is the best candidate node for holding the replica of file f_i. This decision is committed if the node p_1 has no sibling. If an immediate sibling of p_1 exists then GRS will declare it as the best candidate node for file f_i if its access load and the storage load both are less than that of p_1. The node p_1 may have a left sibling and a right sibling. If so, both siblings will be examined and best candidate be selected based on less access and storage load. In order to place the replica at the selected best candidate, the GRS fetches the replica from a server node which is at the nearest distance from the destination node. Table 2 introduces the terminology used in the FSR algorithm.

Table 2: Terminology used

Freq (F_i)	Access frequency of a file i
$Freq_{avg}$	Mean of the access frequencies of all files
$al(c_j)$	Access load contributed by the client j in terms of number of requests
p_k	Node having rank k and having client-tier nodes as its children c_j (j = 1, 2,. ., n)
al(p)	Access load of node p i.e., sum of the workload of children of node p; that is equal to $\Sigma al(c_j)$ where j = 1, 2,. .,n
al_{max}	Maximum access load (among all of the p nodes) in the system
sc(p)	Storage capacity of node p
sl(p)	Storage load on node p in terms of number of replicas
sl_{min}	Minimum storage load
bc_i	Best candidate node (i.e., replica server) for placing replica of file f_i
Sibling(p)	Immediate sibling of node p

Having a replica at a node has an associated storage cost that depends on the replica and that node. We assume that each client request that is serviced by a replica server imposes a certain integer load on that node and that the total load at each of replica servers should not exceed a certain integer upper bound for that node, for example, the node's capacity, i.e., sl(p) = sc(p). In other words, for a replica server to be eligible to hold replica, its available storage space must be greater than or equal to the size of the replica i. If storage load exceeds upper bound, we move the old replicas to the upper tier of the Data Grid so that if in future these replicas become popular then it would be convenient to replicate them back to the past server. The movement of old replicas to the upper tier is handled by the function Evacuate (Fig. 2).

After replication, the GRS will flush out the workload tables in order to calculate the access statistics afresh for the next interval. While placing a replica if GRS finds that desired replica is already present at the selected node, it just updates the replica creation time in the catalog so that the replica is treated as a newly created one.

Replica selection: In this research we use the closest policy (Benoit *et al*., 2007) for replica selection. In the closest policy, the needed data is always located near to the requesting client so that it is transferred to the client in minimum time. In other words, a replica of needed file is selected which is least number of hops away from the requesting client. Since multiple replicas for a data file may coexist in the Data Grid therefore a service should be provided to choose from these available replicas and select one which is the closest to the client. Replica Selector, a component of GRS, is responsible to implement this service (Tang *et al*., 2005). On receiving a request for a specific data file, the replica selector query replica catalog to get information about all the available replicas and return the location of the replica that offers the highest transfer speed.

Replica replacement policy: A replica replacement policy is essential to make decision which of the stored replicas should be replaced with the new replica in case there is a shortage of storage space at the selected node. The function Evacuate is used to get space for the new replicas and works as follows. For a given selected node it checks the creation times of all present replicas. It considers redundant replicas to get removed which were created earlier than the current time session and currently not being active or referenced. These redundant replicas are moved to the upper tier node of the data grid, i.e., to the parent of the current replica server. This is done to ensure that these replicas would be replicated conveniently if become popular in future. The GRS continues to remove each redundant replica from the selected node until the storage space is sufficient to host the new replica. A replica will be deleted permanently from the system if it has not been referenced since last two sessions, thus having minimum usability.

EXPERIMENT SETUP AND RESULTS

The study of FSR replication scheme was carried out using the data grid model shown in Fig. 3. We also study fast spread replication technique proposed by Ranganathan and Foster (2001) for comparison purposes. The fast spread gives the best performance when the access pattern is random. In our simulation, we also used the random access patterns. The fast spread scheme works as follows: for each data file request from the client, the replica of the data file is created in each cache-tier node along the transmission path.

With respect to data access operations, Data Grids are read-only environments in which either data is introduced or existing data is replicated (Hoschek *et al*., 2000). In this research, we consider the data read-only and hence there are no consistency issues involved.

The simulation was done using GridNet simulator (Lamehamedi *et al*., 2003) and runs in sessions. Each session have a random set of requests generated by various clients in the system. The data access requests from the clients follow Poisson arrival. On average each client sends 1 request per 150 sec. According to the properties of Poisson process, the merging of 12 Poisson streams results in a Poisson with about 0.2 requests per sec for the whole system. For each simulation, there are 150 data file accesses requested by clients. Each file is of 1 GB size and scaled to 3.2 MB in experiments. Table 3 shows the bandwidth between Grid tiers and scaling values.

We use geometric distribution for file popularity. In geometric distribution, the probability of requests for the

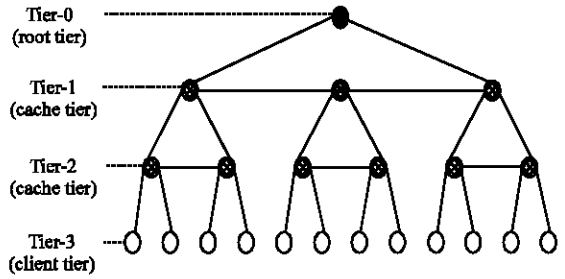

Fig. 3: Simulated data grid topology

Table 3: Grid tiers link bandwidth

Data grid tiers	Band width	Scaling
Tier 0-1	2.5 Gbps (320 MBps)	1.0 MB
Among tier 1	7.0 Gpbs (896 MBps)	2.8 MB
Tier 1-2	2.5 Gpbs (320 MBps)	1.0 MB
Among tier 2	7.0 Gpbs (896 MBps)	2.8 MB
Tier 2-3	622 Mbps (77.75 MBps)	0.24 MB

nth most popular file is defined as $P(n) = p(1-p)^{n-1}$, where, $n = 1, 2, \ldots$ and $0<p<1$. The geometric distribution is used to model the scenario that some data files are requested more times than others. A larger value of p means more requests for a smaller proportion of data files. In this research, the value of $p = 0.01$ is used to model the data request distribution.

When a client generates a request for a file, the replica of that file is fetched from the nearest replica server and transported and saved to the local cache of the client. The client storage is cache enabled, in other words, if the client requests to access a data file that was accessed in the last time; it can get the file from the local storage directly. Initially all the data was held at root node with a small proportion of data distributed at cache tiers in a random fashion. As the time progresses, the access statistics are gathered and are used for the decision of replica creation and placement. When a replica is being transferred from one node to another, the link is considered busy for the duration of transfer and cannot be used for any other transfer simultaneously. For each client node, we keep a record of how much time it took for each file that it requested to be transported to it. The average of this time for various simulation runs was calculated.

The response time for a data file access is the interval between the beginning of the data request sent by the client and the end of the data transmission. The average response time is the mean value of the response times for all data accesses requested by the clients in a simulation session. In Fig. 4, the replication schemes are compared on the basis of mean response time from the simulation. The graph shows that the performance of Fast Spread is better than FSR; however, it is observed that Fast Spread

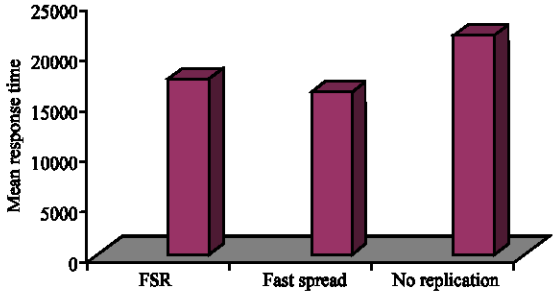

Fig. 4: Mean response time (msec) in replication techniques

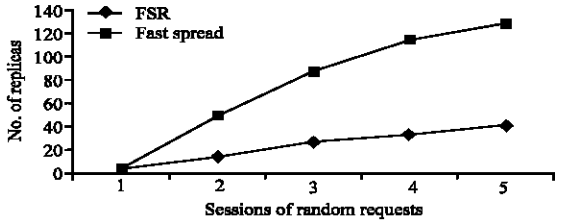

Fig. 5: Storage utilization in two techniques

causes high frequency replication which inflicts heavy load on replica servers. The overhead incurred is shown in the Fig. 5.

In case of fast spread, the placement of replicas occurs with the average of 31 per session. In contrast, we have 10 replicas per session placement in FSR. Since we transfer a small number of replicas using FSR therefore it does not affect network communication cost greatly. So FSR achieves the performance almost near as Fast Spread and at the same time makes a more effective usage of data grid storage. The capacity of the replica server has a major impact on the performance of a dynamic replication algorithm. With FSR we can get favorable mean response time even when the storage size of the replica server is very small.

RELATED WORK

An initial work on dynamic data replication in Grid environment was done by Ranganathan and Foster (2001) proposing six replication strategies: No replication or caching, best client, cascading, plain caching, cascading plus caching and fast spread. The analysis reveals that among all these top-down schemes, Fast Spread shows a relatively consistent performance through various access patterns; it gives best performance when access pattern are random. However, when locality is introduced, Cascading offers good results. In order to find the nearest replica, each technique selects the replica server site that is at the least number of hops from the requesting client.

Abawajy (2004) proposed an improvement in cascading technique namely Proportional Share Replication policy. The method is heuristic one that places replicas on the optimal locations when the number of sites and the total replicas to be distributed is known. The study on dynamic replication algorithms is presented by Tang *et al.* (2005) in which they have shown improvement over fast spread strategy while keeping the size of data files uniform. The technique places replicas on data grid tree nodes in a bottom up fashion with no data movement considered among nodes on the same tier. We in our study used the topology in which data transference along a tier is permitted and is thus more flexible and general. Moreover, in our strategy, there is no need to calculate the access load for the whole path from client to root.

Fan *et al.* (2006) and Yuan *et al.* (2007) have formulated the replica placement problem mathematically followed by theoretical proofs of the solution methods.

A hybrid topology is used by Lamehamedi *et al.* (2002) where ring and fat-tree replica organizations are combined into multi-level hierarchies. Replication of a dataset is triggered when requests for it at a site exceed some threshold. A cost model evaluates the data access costs and performance gains for creating replicas. The replication strategy places a replica at a site that minimizes the total access costs including both read and write costs for the datasets. The experiments were done to reflect the impact of size of different data files and the storage capacities of replica servers. In later study, Lamehamedi and Szymanski (2007) proposed a decentralized data management middleware for Data Grid. Among various components of the proposed middleware, the replica management layer is made responsible for the creation of new replicas and their transfer between Grid nodes. The experiments were done considering both the top down and bottom up methods separately with data repository located at the root of Data Grid and at the bottom (client-tier), respectively.

Assuming the storage system has a limited capacity, two new metrices to evaluate the reliability of the system and an online optimizer algorithm that can minimize the data missing rate is proposed by Lei *et al.* (2008). Chang and Chang (2008) has proposed a technique named Latest Access Largest Weight, LALW, focusing on recent file accesses as high priority events and balancing the system load globally. In present study, fair-share replication, we emphasize on balancing the load in data grid on the basis of individual server.

CONCLUSION

The management of huge amounts of data generated in scientific applications is a challenge. By replicating the frequent data to the strategic locations, we can enhance the data availability and reduce access latency. In this study, we proposed a load balancing replication strategy

for data grid environment. The most frequent files are placed very close to the users and the decision of replica placement is made based on the access load and the storage load of the candidate replica servers. The experiment results show the performance of our proposed scheme. The grids with same application domain are now in the process of internetworking. In our future study we intend to focus on replica placement in InterGrid environment.

REFERENCES

Abawajy, J.H., 2004. Placement of file replicas in data grid environments. Proceedings of 4th International Conference on Computational Science (ICCS 2004), Workshop on Programming Grids and Metasystems. LNCS 3038. June 6-9, Krakow, Poland, Springer-Verlag, pp: 66-73.

Benoit, A., V. Rehn and Y. Robert, 2007. Strategies for replica placement in tree networks. Proceedings of Parallel and Distributed Processing Symposium (IPDPS'07), Heterogeneous Computing Workshop. March 26-30, IEEE, pp: 1-15.

Chang, R.S. and H.P. Chang, 2008. A dynamic data replication strategy using access weights in data grids. J. Supercomputing, 10.1007/s11227-008-0172-6.

Chervenak, A., I. Foster, C. Kesselman, C. Salisbury and S. Tuecke, 2000. The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. Special Issue on network-based storage services. J. Network Comput. Appl., 23: 187-200.

Elghirani, A., A.Y. Zomaya and R. Subrata, 2007. An intelligent replication framework for data grids. Proceedings of IEEE/ACS International Conference on Computer Systems and Applications (AICCSA'07). May 13-16, Amman, Jordan, IEEE, pp: 351-358.

Fan, Q., Q. Wu, Y. He and J. Huang, 2006. Transportation strategies of the data grid. Proceedings of the 1st International Conference on Semantics, Knowledge and Grid (SKG'05). Nov. 2005, Guilin, Guangxi, China, IEEE, pp: 108-110.

Foster, I. and C. Kesselman, 2003. The Grid: Blueprint for a New Computing Infrastructure. 2nd Edn., Morgan Kaufmann Publishers Inc., San Francisco, CA 94104, USA, ISBN: 978-1-55860-933-4, pp: 37-63.

Hoschek, W., J. Jaen-Martinez, A. Samar, H. Stockinger and K. Stockinger, 2000. Data management in an international data grid project. Proceedings of the 1st IEEE/ACM international Workshop on Grid Computin, Bangalore, India, LNCS 1971, Dec. 17, pp: 333-361.

Lamehamedi, H., B.K. Szymanski, Z. Shentu and E. Deelman, 2002. Data replication strategies in grid environments. Proceedings of 5th International Conference on Algorithms and Architectures for Parallel Processing. Oct. 23-25, IEEE, pp: 378-383.

Lamehamedi, H., B.K. Szymanski, Z. Shentu and E. Deelman, 2003. Simulation of dynamic data replication strategies in data grids. Proceedings of International Parallel and Distributed Processing Symposium (IPDPS'03), April 22-26, Homogeneous Computing Workshop, Nice, France, IEEE, pp: 10-10.

Lamehamedi, H. and B.K. Szymanski, 2007. Decentralized data management framework for data grids. Future Generat. Comput. Syst. Elsevier, 23: 109-115.

Lei, M., S.V. Vrbsky and X. Hong, 2008. An on-line replication strategy to increase availability in data grids. Future Generation Comput. Syst. Elsevier, 24: 85-98.

Lin, Y.F., P. Liu and J.J. Wu, 2006. Optimal placement of replicas in data grid environment with locality assurance. Proceedings of 12th International Conference on Parallel and Distributed Systems (ICPADS'06), July 12-15, IEEE, pp: 465-472.

Loukopoulos, T. and I. Ahmad, 2000. Static and adaptive data replication algorithms for fast information access in large distributed systems. Proceedings of 20th IEEE International Conference on Distributed Computing Systems, April 4-13, Taipei, Taiwan, pp: 385-392.

Ranganathan, K. and I. Foster, 2001. Identifying dynamic replication strategies for a high-performance data grid. Proceedings of International Grid Computing Workshop (GRID 2001), LNCS 2242. Nov. 12, Springer-Verlag, Denver, USA., pp: 75-86.

Tang, M., B.S. Lee, C.K. Yeo and Y. Tang, 2005. Dynamic replication algorithms for the multi-tier data grid. Future Generation Comput. Syst. Elsevier, 21: 775-790.

Yuan, Y., Y. Wu, G. Yang and F. Yu, 2007. Dynamic data replication based on local optimization in data grid. Proceedings of 6th International Conference on Grid and Cooperative Computing (GCC'07). Aug. 16-18, IEEE Computer Society, USA., pp: 815-822.