A new Replica Placement Algorithm in Data Grid

Zeinab Fadaie[1], Amir Masoud Rahmani[2]

[1]Department of Computer Engineering, Faculty of Engineering,

Science and Research Branch, Islamic Azad University,Tehran, Iran

[2]Department of Computer Engineering, Faculty of Engineering,

Science and Research Branch, Islamic Azad University,Tehran, Iran

Summary *- Replica placement is one of the important factors to improve performance in data grid systems. A good replica placement algorithm can result in good performance gains. It should be mentioned that, these algorithms or strategies are dependent on architecture of the data grid. By considering different kinds of architecture in data grid systems, a true representation of a grid is a general graph. So we propose a new algorithm for suitable placement of replicas on graph-based data grids. The performance of the proposed algorithm is improved by minimizing the data access time, avoiding unnecessary replications and nice performance in balancing the load of replica servers. The algorithm will be simulated using a data grid simulator Optorsim, developed by European Data Grid projects.*

Keywords: *Grid, Data Grid, graph-based topology, Replica Placement, Data Replication.*

1. Introduction

Grid

In 2002 Kesselman and Foster introduced us to a definition of a grid as follow: "A system that coordinates resources that are not subject to centralized control, using standard, open, general purpose protocols and interfaces to deliver non-trivial qualities of services" [1]. Nowadays, data grids can be seen as frameworks responding to the needs of large scale applications by affording so many resources. These are distributed on different geographically locations, but are organized to provide an integrated service. When we have to perform some complex computational experiments which require high computational resources, we do not need to install that computing infrastructure. Rather we can simply become the part of a grid with high computational powers. The idea of sharing the computing powers of the available resources across the grid environment to perform some experiment, without having to install additional computational resources is called the Computational Grid.

Data Grid

On the other hand, data grid is a type of grid which provides services and infrastructure to assist the widely distributed data intensive applications which require the access of huge amounts of data. The basic services provided by data grid architecture are storage systems, data access, and metadata services [2]. In a data grid system the computers are distributed across several geographical locations. The issue is to provide maximum availability of data to the users which are normally scientists from different universities and research laboratories. The size of data that needs to be accessed is in terabytes and it will soon be measured in total petabytes. The efficient access of such a huge data, which is widely distributed, is slowed down due to network latencies and bandwidth problems. With the growing size of the grid, the complexity of this infrastructure is increasing. High availability of data is a major challenge in the grid environment.

Data Replication

To meet the challenge of high availability, data replication is considered to be the major technique. It promotes high data availability, low bandwidth consumption, increased fault tolerance and improved scalability and response time [3-9]. When data is replicated, copies of data files are created at many different places in the data grid. Replication can save storage resources as compared to the storage occupancy of data present at each site. It also saves a large amount of bandwidth as compared to the storage occupancy of data present at each site. Hence, for the provision of speedy data access all the time, data replication is an excellent tradeoff between storage availability and network bandwidth availability [10]. The data replication algorithm has to answer critical questions such as: 1.how to balance the number of replicas in grid sites. Indeed, increasing number of replicas lead to increase data availability and reliability, however the storage space will be increased as well. Therefore, a good balancing of number of replicas is required; 2.where the replica must be placed. Placing the new replicas in the appropriate location site can promote reducing the network bandwidth

consumption and reduce the turnaround job time. The main idea is to keep the data close to the user in order to make the access efficient and fast. But the dynamic behavior of grid users makes it difficult to make decisions regarding the data replications to attain the target of maximum availability [11-12]. To maximize the potential gain from file replication, a replica placement strategy is so important. A replica placement service is a component of the data grid architecture that decides where a file replica should be placed in the system. In recent years, more and more works focused on the replica management in parallel and distributed systems [13]. But most of them concerned on replica location, replica placement or building infrastructures for replica management [14-15]. In fact, replica placement is one of the important challenges to improve performance and good placement strategies can result in significant performance gains [16-18].

The data grid architecture

The replication technique is highly dependent upon architecture of the grid. A data grid can be supported by different architectures. It can be a multi-tier architecture; a tree like structure in which the nodes are arranged in a tree like hierarchy. For example, the data grid of the GriPhyN project [19] in which tier 0 is the main data source (CERN), tier 1 contains the national centers, tier 2 the regional centers, tier 3 the workgroups and finally, the nodes at tier 4 are desktops. Alternatively, it can be graph like topology, in which any node can be connected to any other node without any restrictions of tree topology. It can be peer to peer topology, or it can be any hybrid model. A replication technique is designed according to the architecture in question. It should be noted that, every node in this structure as a grid site has at least two basic elements: the Storage Element (SE) and the Computing Element (CE). According to these elements, other important issues are considered: storage load and access load. Most of the time, the replicas placed at the parent node of a client that generates the maximum request. So the access load of each node is calculated and ranked according to file access frequency, e.g. 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 the node p has three children, then access load of p will be the cumulative access loads of all three children. On the other hand, the storage load of each node should exceed its capacity [20]. Therefore, in this paper, we address the replica placement problem in graph-based data grid to meet the load balancing of replicas with the objective of minimizing communication cost and responding to user's requirements as fast as possible. Load balancing is managed by workload constraint of replicas, and the main idea is to keep the data close to the user in order to make the access efficient and fast. At the first step, we propose a new architecture to show the communication between our main components which will be used in our proposed algorithm. Then a new replica placement algorithm is proposed to solve our replica placement problem. This algorithm contains three phases: at the first phase, the graph based data grid structure is traversed by the Breadth First Search (BFS) algorithm to determine the level of each node. Additionally, the Depth First Search (DFS) algorithm is used to label the nodes in the depth-first order they are encountered. The result of this phase is a tree structure. The level and depth-first order of each node on the yielded structure is maintained to use in the next phase. At the second phase, replica selection and replica placement is performed on the tree structure that is obtained from previous phase. It should be noted that, this tree structure is used to better managing of replicas; actually in the real environment the nodes are located in graph-based topology. The algorithm considers this graph-based topology to balance the load on the replica servers. This goal is achieved by considering the sibling nodes of replica servers on this topology, and the concept of storage load and access load on them. At the third phase, the storage space on each replica server is considered. In this phase the replacement strategy is performed. The rest of the paper is organized as follows:

Section 2 gives a brief introduction of previous works on data replication and placement. **Section 3** defines the architecture that is being used. **Section 4** introduces our replica placement algorithm. In **section 5** the simulation results will be described. Finally in **section 6** we will present our conclusion and future works.

2. Related Works

Tang et al. [21] in 2005 have presented two dynamic replication algorithms, Simple Bottom up (SBU) and Aggregate Bottom up (ABU) to reduce the average response time of data access. The job of SBU is to create a replica as close as possible to the client which requests for a certain file. It only processes the records individually in the access history and does not know its relationship to other nodes. While the ABU's job is to aggregate the history records to the next upper tiers one by one till it reaches the root node. The results of simulation show that these two algorithms reduce the data access time significantly when compared to the static replication strategies. Tang's model is tree structure. Their assumption is that all requests travel up towards the root to find the desired replica. So this assumption may cause bottleneck problem.

In 2007 Yuan et al. [22] proposed a dynamic data replication strategy based on the local optimization principle. They considered the bottleneck of data grid storage capacity of different nodes and bandwidth available between these nodes. The proposed data replication strategy is based upon two important factors (1) the storage capacity available at different nodes and (2) the bandwidth available between different nodes. The idea is to achieve the global data access optimization, first by achieving the local data access optimization. Yuan's model is again a tree structure because of its simplicity. Tree structures are not very suitable in real grid environments because their infrastructures are very dynamic in nature, and nodes in grid can be added and deleted any time.

Abdullah [23] presented a P2P model in 2008 for higher availability, reliability, and scalability. Then have developed their own data grid simulator to test the proposed replication strategy, taking response time, number of hops and average bandwidth consumption as basic parameters for evaluation. In this research they are studying four replication strategies, out of which two are existing strategies: "requester node placement strategy" and "path node placement strategy", and two are newly proposed in this research: "path and requester node placement strategy", and "N-hop distance node placement". In the "requester node placement strategy", when a required file is found then it is only replicated to the requester node. In the "Path node placement strategy" the file is replicated to all the nodes on the path from the requester node to provider node. The newly proposed strategy "Path and requester node placement strategy" is a combination of the first two strategies. In "N-hop distance node placement" a file is replicated to all neighbors' of provider nodes within an n hop distance. The results of their simulation show that new strategies have shown better performance than existing ones in terms of performance, success rates and response time. However, the proposed strategies use more bandwidth than the existing strategies. The drawback of the research is that the storage loads of replica servers are not considered in their strategies, because the file is replicated to all the nodes on the path from the requester node to provider node.

Ding et al. proposed Data Placement algorithm and self tuning data replication algorithm in 2009 for general grid topology in [24] for improved load balancing, reduced response time and conserved network bandwidth. In proposed model, grid is composed of clusters, with each cluster having different storage and computational capabilities. As the resources in the cluster sites and data access patterns keeps on changing. So a self tuning data replication algorithm is proposed to automatically adjust such changes. The new replication algorithm outperforms the general threshold based algorithms in terms of efficiency and load balancing.

In 2007 Nehra et al. [25] presented architecture for load balancing with parallel resource allocation. Performance measures such as the average queue length at each server and the average throughput are used for the evaluation. The experimental results show that execution time is reduced in parallel algorithm compared to serial one. Throughput is also measured with and without load balancing. Load is balanced using mobile agent (MA) approach which provides a new solution to support load balancing with resource management. In the preliminary simulation, for simplicity, workload at a server is defined as the length of the job queue, which represents the number of jobs in the queue. The storage load is another factor that should be concerned in balancing the load of servers. This factor was not considered in Nehra's experimental results.

In 2008 Horri et al. [26] proposed a three level hierarchical structure for dynamic replicating file in data grids. In contrast to Bandwidth Hierarchy Replication (BHR) algorithm [27] which considers 2-level, the 3-level proposed performs better and it is more realistic (BHR was presented in 2004 by park et al.). From job scheduling point of view, the proposed algorithm, first selects the appropriate region (i.e. available maximum requested files), next selects the appropriate LAN in that region and finally selects the appropriate site in that LAN, therefore job execution time Decreases since we have minimum data transfer time.

In 2011 Sashi et al. [28] presented a modified form of BHR to overcome its limitations. In the modified BHR model a network region is defined as a network topological space where sites are located closely. Whenever the required replica is present in the same region, the job completion will be fast. Again, the Modified BHR model is based on tree structure which is not very suitable in real data grid environment.

In 2009 Rasool et al. [29] proposed a two way replication strategy. The multi-tier sibling tree architecture is used which a mixture of the architectures is presented by Ranghatan and Lin. It's a hierarchical model in which all the siblings are connected to each other as well. In this two way replication (TWR) scheme the most popular data is identified and placed to its proper host in a bottom up manner in this they are closer to the clients. In top down manner the less popular files are identified and are placed to one tier below the root node, in this way they are close to the roots. In this approach, replica selection is done by using the closest policy which tries to provide the data from the nearest site. The drawback of the research is that it only considers the homogeneous data grid nodes and cannot be applied to heterogeneous nodes while the nodes in a data grid are normally heterogeneous.

In 2008 Lin et al. [30] have addressed the problem of placement of a new replica in a proper place by considering a priority list. The proposed replica placement algorithm finds out the minimum number of replicas when the maximum workload capacity of each replica is given. The hierarchal model is different from other related works that done, because in this model they assume a logical connection between all siblings of a parent and a request can be served from a node present in sibling ring. If requested data is not present in sibling ring then parent ring is searched. This architecture is called a Sibling Tree model, which is an extension of a normal tree structure. The presented hierarchal model assumes a logical connection between the siblings and actually all connections from on sibling to another physically involves the parent i.e. at most two hops. This means the actual time taken to serve a request is infected more than it is presented, as this logical connection is assumed physical and already the time complexity is too high. The problem of network congestion or bandwidth consumption is not mentioned in proposed model.

By considering different kinds of architecture in data grid, a true representation of a grid is a general graph in which there is no central node designated as a root node, and each node can be connected

with any number of nodes. In literature we can see the work done on a graph as the grid architecture is much less. Most researchers have worked on hierarchical structure and have mentioned extending their work to general graphs in the future. Therefore in this paper, we consider a grid with graph-based topology. The main idea is to keep the data close to the user in order to make the efficient and fast accesses. As mentioned before, replica placement is one of the important challenges to improve performance and good placement strategies can result in significant performance gains. This goal is achieved, when every replica server in the data grid is aware of the location of its related nodes. Thus, at the first step, the graph structure is converted to tree structure due to better managing of replica servers and their related nodes. Then, their sibling nodes in real topology as graph topology are considered to balance the load on our structure. So our proposed algorithm which has not been used in other existing placement strategies not only considers access load of replica servers but also reduces the mean job execution time.

1. RPGBA: A new Architecture for Replica Placement

In this section we will describe our proposed architecture, Replica Placement on Graph-Based data grid Architecture. As mentioned before, a true representation of a grid is a general graph in which there is no central node designated as a root node, and each node can be connected to any number of nodes. Therefore in this paper, we consider a grid with graph-based topology. At the first step of our proposed algorithm, this structure is converted to hierarchal structure due to better managing of replica servers and their related nodes. The multi tier data grid as shown in Fig.1 has many advantages: First, it allows hundreds or even thousands of scientists everywhere to access the resources in a common and efficient way. Second, the datasets can be distributed to appropriate resources and accessed by multiple sites. The network bandwidth will be used efficiently because most of the data transfers only use local network resources. Furthermore, the multi tier structure enables the flexible and scalable management for datasets and users. In this figure, the data grid is modeled to have three tiers: The machine of Tier 0 is connected to machines of Tier1. The Tier0 machines provide abundant storage capacity. The Tier1 machines provide computing and storage resources, each Tier1 machine that we called the Regional Server has a number of related Tier 2 machine each of which has the computing resources as well as the storage capabilities.

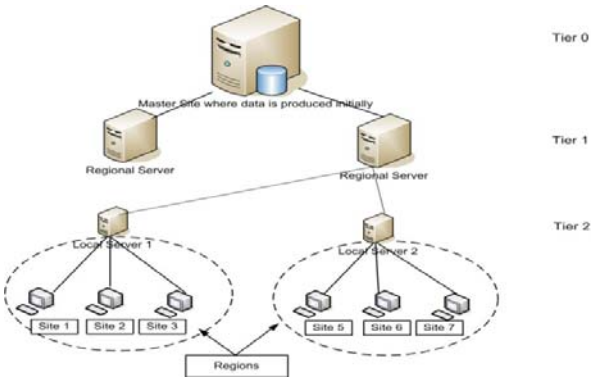

Figure 1: The multi tier data grid

The Tier2 machines called Local Servers. Tier3 machines are workstations and they are implicit in this model. According to the modified BHR proposed model [28] which is based on network level locality, the enhanced algorithm tries to replicate files within a region. A network region is a network topological space where sites are closely located. If the required replica is found within the region the job completion will be fast. Regional Servers should have huge storage capacity because they play the role of intermediate replica servers and interact with two important components: Replica Manager (RM) and Replica Catalog (RC) which will be explained later. In order to facilitate dynamic file replication in the multi-tier data grid, following services are available in the system [31]: Local Replica Catalog (LRC), Local Replica Manager (LRM), Replica Catalog (RC) and Replica Manager (RM). The LRC and LRM are local services which are distributed on every machine in the system, where RC and RM are centralized services, these two services located at the Region Servers. The RM and RC at the Regional Server manage LRM and LRC of sites which are connected to them. In addition to these file services, some more services are assumed to be running in the data grid [32]. As shown in Fig. 2, RC and RM are two main components and in our model consist of some more detailed services which will be described. File replication is the process of storing multiple copies of the same file at different physical locations. The gained redundancy improves reliability, fault tolerance and accessibility. The copies are called replicas. A URL pointing to a physical copy of the file is called a physical file name (PFN) of the file. The set of PFNs are mapped to a system-wide unique identifier, called the logical file names (LFN), so in order to relate the LFNs and PFNs another component which called RC is introduced.

1) Replica Catalog (RC): Through the Replica Catalog, the physical locations of data files are recognized. The replica catalog consists of some detailed services such as:

- Database which stores mapping between LFN and PFN: This database is a registry that keeps track of where the files are stored in the grid. It stores mapping between LFN and PFN of each file. All files that have been placed in the grid

using data placement service are registered in this database.

- Database which stores level and depth-first order of each node: As it we will be discussed in the next section, all of the entire nodes in data grid environment are labeled by our proposed algorithm. As a result of this algorithm, each node can be aware of the nodes which are located at its subtree. Additionally, the "level" indicates the distance of studied node from the root node.
- Replica Location Service (RLS): The RLS invokes two above databases to store mapping between LFN and PFN or the information that were mentioned before like the order and level of each node.

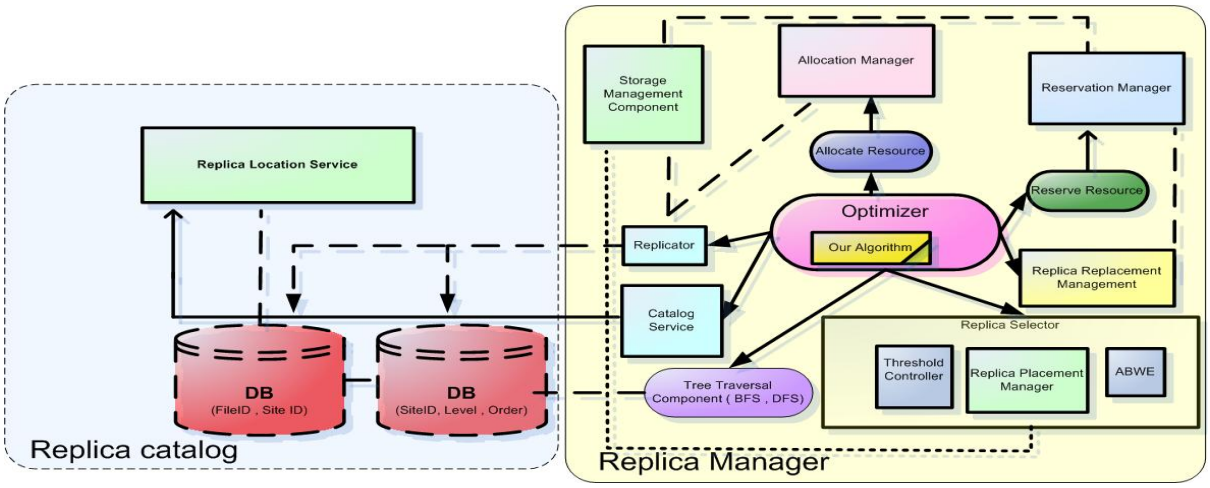

Figure 2: the RPGBA components

2) Replica Manager (RM): Its major duty is to perform replication and creating replicas. This component consists of following components:

- Reservation Manager: validates the reservation requests in terms of the resource usage policies and performs the admission control for the resource availability.
- Allocation Manager: Communicates with the resource manager of the computing, networking and storage systems for the allocation of these resources with respect to the reservations.
- Replica Selector: The replica selector selects the replica with the minimum communication cost. The communication cost is denoted as follows:

$$\textit{Communication cost} = \frac{Size_i}{BandWidth_{ab}} \qquad (1)$$

$Size_i$= size of replica 'i'

$BandWidth_{ab}$= *available bandwidth between grid site 'a' and grid site 'b'*

- Catalog Service: When the optimizer invokes the Catalog Service, it interacts with RLS to get the information from Database that saves mapping between LFNs and PFNs. This database connects to another database that stores the order and the level of each node.
- Replicator: The replicator replicates the selected replica on the best place which will be found by our proposed algorithm.
- Replica Selection component: This component is composed of some more detailed services such as:

  ✓ ABWE [33]
  The ABWE monitoring tool can be used to estimate RTT and available bandwidth between host pairs. ABWE is a low network intrusive monitoring application, based on packet pair techniques and designed to work in continuous mode. The source node (requester node) in grid is configured to use ABWE for sending continuously probe packets to all other target nodes (contained desired replica), and reporting some information. The information like delay and available bandwidth are returned back to the source node. The source node collects this kind of end to end metrics for source and target nodes pair and performs some process to select the target node with minimum communication cost.

  ✓ Replica Placement Management

  Our proposed algorithm uses this component to find the minimum distance between source and destination. Some important factors are considered to achieve this goal. These factors will be discussed later in next section. Additionally, this component considers the access rate of each replica server and finds the replica server with minimum access load. It should be noted that every time the access load of target nodes is compared with the threshold value. The target nodes which their access rates are lesser than the threshold value are selected for replication.

  ✓ Threshold Controller [34]

  This controller checks the access request rate and available server capacity to determine the threshold value. The threshold value is set based on the average aggregated access counts at the replica servers. The value of the average aggregated access counts is calculated by dividing the total number of aggregated access counts for a file at the replica servers at each

level to the number of the replica servers at that level. A high access request rate corresponds to frequent accesses by the users. A file with frequent access is called most "Popular" file whose access count exceeds the threshold. Increasing the threshold value lessens the number of replicas created. On the other hand, when the request rate drops, the fewer replicas are created, Even though the system might be capable (in terms of bandwidth and storage) of supporting more replicas to improve access latency.

If Reservation manager succeed in making reservations, RM calls for allocation manager. Once Allocation Manager Finishes allocating the reserved resources, RM starts the file transfer from a source machine to the destination machine. It should be noted that, every request passes three parameters to Catalog Service, F: indicates the file which requested, M: indicates the machine that requests the file, D: indicates the request deadline .

In Fig. 3 the message passing structure between the components that were discussed above is illustrated. As it can be seen in this figure, first of all the grid site sends its request to the RM. Every request contains the information like: F: indicates the file which requested, M: the machine that requests the file, D: indicates the request deadline. The catalog service at the RM passes user's request to the RC to find the grid sites which contain the desired file F. The Replica Location Service at the RC invokes two databases:

1. A database which stores (File ID , Site ID)
2. A database which stores (Level , Order , Site ID)

The RLS searches in DBs to find the information like: The exact physical location of target sites and their level. This information is returned to RM for further investigations. The replica selector at the RM plays a key role in data placement. So through replica placement component, the minimum distance between Machine M and the target sites is computed. The sites with minimum hope are chosen for next step. The ABWE component estimates RTT between source and target sites, then chooses those nodes which can answer to our requests before their deadlines *d*. The ABWE checks the available space on the storage element of grid sites, through storage management component. In the next step the replica placement component contacts with the Threshold Controller to find the site with minimum access load among the remaining sites. As mentioned before, we know the level of each site in our structure. Additionally, the threshold value of each level is stored in threshold controller. So the access load of selected sites at each level is compared to the threshold value of that level.

Finally the site which its access load is lesser than the threshold is chosen as a destination site. The request is sent to replicator to perform the replication process and submit jobs. This component contacts to the storage management and allocation manager component. If Reservation manager succeeds in making reservations, RM calls allocation manager. Once Allocation Manager Finishes allocating the reserved resources, RM starts the file transfer from a source machine to the destination machine.

4. RPGB: A new algorithm for Replica Placement

In this section we will describe our proposed Replication algorithm, Replica Placement on Graph-Based data grid. Our proposed algorithm consists of three phases:

Phase1, traversing the data grid structure: in this phase, the data grid structure is traversed by the Breadth First Search (BFS) algorithm to determine the level of each node. Additionally, the Depth First Search (DFS) algorithm is used to label the nodes in the depth-first order they are encountered.

Phase 2, requesting a file and performing the replica selection and replica placement

Phase 3, Replacement, if there was enough space in storage element for storing a new replica, it will be stored; otherwise an existing file should be selected for replacement.

Phase1: Traversing the hierarchical structure

In this phase, our graph structure is traversed by the BFS algorithm. The BFS begins at the root node and explores all the neighbor nodes. By this algorithm the level of each node is determined. In the first stage of this algorithm, the "Level 0" is assigned to the root node. In the second stage, the vertices adjacent to the root node are visited. These vertices placed into the "Level 1". In the third stage, the new vertices that are at the distance of two edges away from the root node are reached. These nodes are placed into the "Level 2" and so on. The BFS traversal terminates when every node has been visited. As a result of this algorithm, we assume that, the Regional Servers are located at "Level 1" and the Local Servers are located at "Level 2". Consider the graph structure which is shown in Fig.4. This structure is traversed by the BFS algorithm. It should be noted that, every node in this graph structure is a site in real data grid structure, and every edge demonstrates the relations between sites. In the traversal tree the adjacent of each node should be maintained (Fig. 5).

In addition to the BFS algorithm, another traversal algorithm that is called the DFS is used to find the nodes in the depth-first order they are encountered. Our proposed algorithm is constructed on the bases of following assumptions:

- Let '*T*' denotes a tree that is obtained after traversing our graph structure by the BFS algorithm.

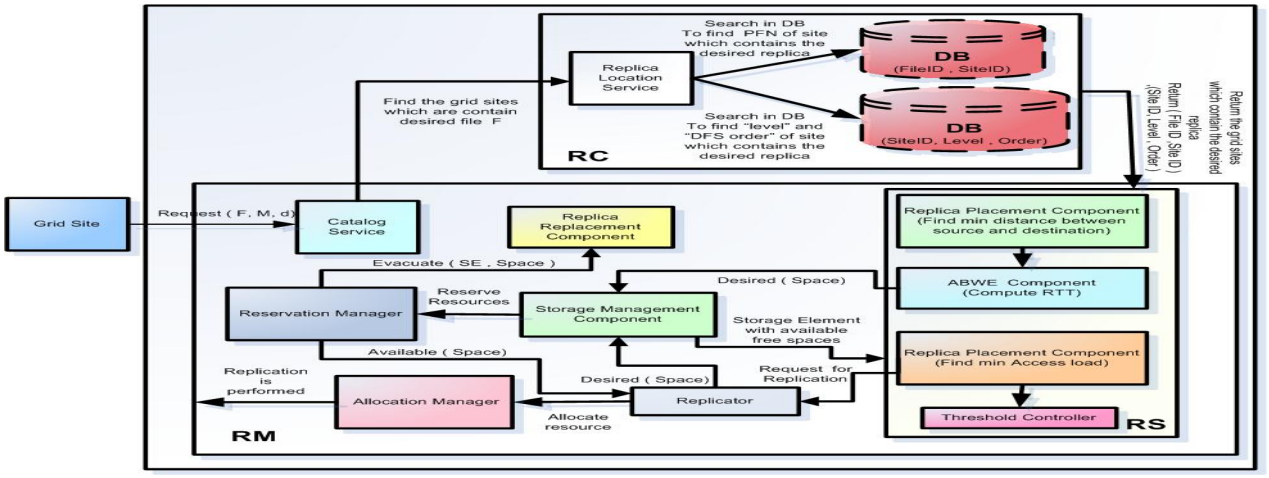

Figure 3: The message passing between components

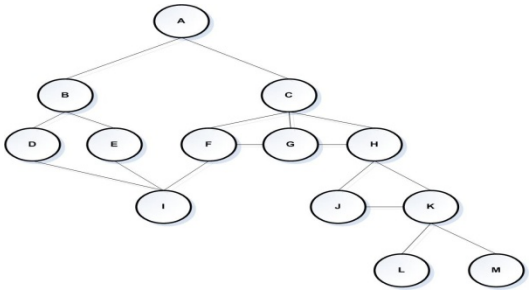

Figure 4: The graph structure

- Let *lca(u,v)* denotes the lowest common ancestor of nodes *u* and *v* in tree '*T*'.

As shown in Fig. 6, first, the depth-first traversal is executed on tree 'T' to label the nodes in the depth-first order they are encountered. Then, in that same traversal we maintain a list '*L*'. The list 'L' demonstrates the order of nodes of 'T'. These nodes were visited by DFS algorithm. It should be mentioned that the number given to any node is smaller than the number which had given to any of its descendents.

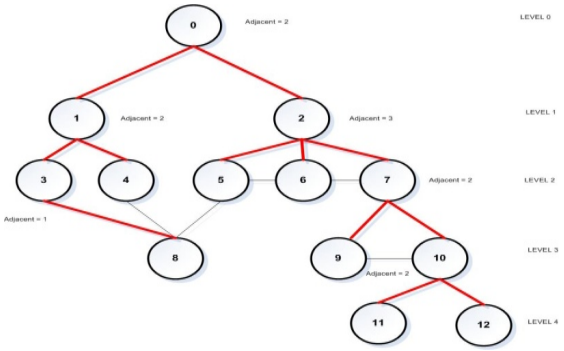

Figure 5: The Tree 'T' structure is obtained, after traversing the graph by the BFS algorithm

It is assumed that these numbers which are assigned to the nodes of the tree T are the grid sites IDs.

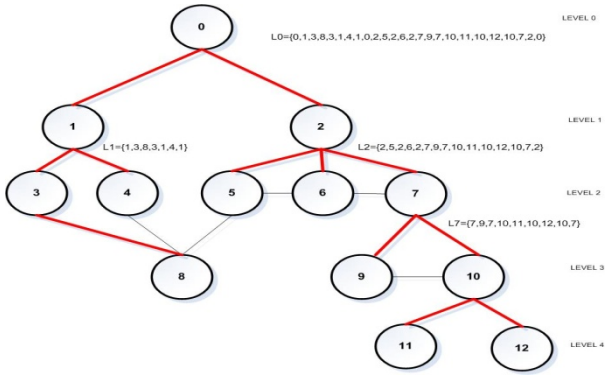

Figure 6: The tree 'T' that is traversed by the DFS algorithm

Now if we want to find *lca (u,v)*, we find the first occurrence of the two nodes in *L*, this defines an interval *I* in *L*. Suppose *u* occurs in *L* before *v*. Now, *I* describes the part of the traversal, from the point we first discovered *u* to the point we first discovered *v*. *Lca (u,v)* can be retrieved by finding the minimum number in I. This is due to the following two simple facts:

- If *u* is an ancestor of *v* then all those nodes visited between *u* and *v* are in *u*'s subtree, and thus the depth-number assigned to *u* is minimal in I.
- If *I* is not an ancestor of *v*, then all those nodes visited between u and v are in *lca (u,v)*'s subtree, and the traversal must visit *lca (u,v)*. Thus the minimum of *I* is the depth-number assigned to *lca(u,v)*.

Therefore, the BFS algorithm determines the level of each node in tree structure, and by the DFS algorithm the depth-first order of labeled nodes will be obtained. The proposed algorithm stores this

information in databases that are located in Replica catalog which is described at previous section. As shown in Fig. 7, the depth-first traversal creates these depth numbers and the following list *L7*: *{7,9,7,10,11,10,12,0,7}*. Now if we want to find *lca (9,12)*, we find the first occurrences of the two nodes *(9,12)* in *L*, this defines an interval *I* in *L*. The *lca* can be retrieved by finding the minimum number in *I*. Here the minimum number in *interval I* is 7.

$$L7: \{7,\underbrace{9,7,10,11,10,12}_{\text{Interval } I},0,7\}$$

$$Lca\ (9,12) = 7$$

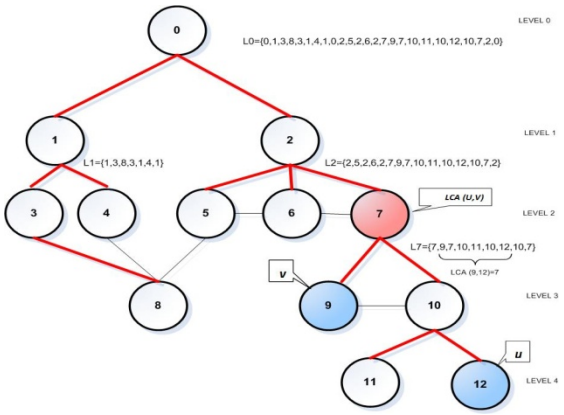

Figure 7: The Lowest Common Ancestor (*LCA (u,v)*)

As mentioned before the numbers which are assigned to the nodes of the tree 'T' are the grid sites IDs. After traversing the tree by the BFS and the DFS algorithms, the triple (SiteID, level, depth order) could be registered into target database. According to our proposed architecture that is illustrated in previous section, the Local Server and the Regional Servers which are located at upper level in our structure can be aware of the location and the level of their children and descendent nodes. So every Regional server is aware of the grid sites which are placed on its subtree. This phase is illustrated in Fig.8.

Phase 2: Replica Selection and Replica Placement

In the first step, the Replica Manager nodes which contain our proposed algorithm aggregate the access records of each file from lower to upper level to determine the threshold value. A high access request rate corresponds to frequent accesses by clients which results in more "popular" files whose access count exceed the threshold. The initial threshold value is set based on the average aggregated access count at the replica servers in each level. The value of the average aggregated access cost is calculated by dividing the total number of aggregated access count for a file at the replica servers in the second to lowest tier of the hierarchy by the number of replica servers at that tier.

Phase 1

T: Tree , ST: SubTree

1. Run the BFS algorithm to traverse the graph structure
2. Let *T* denotes a tree which traversed by the Step 1.
3. Maintain the adjacent of each node on the tree *T*.
4. Run DFS algorithm to traverse the tree *T*.
5. Maintain the List *L* of nodes in the same order that they are visited.
6. Keep the subtree of every node.
   This step is based on the number of occurrences of each node in list L
   The interval *ST* of node 's' describes the part of the traversal from the point we first discovered node 's' to the point we discovered the 's' for the (adjacent(s) + 1) time.

Figure 8: Traversing the hierarchical structure (Phase 1)

The initial value is the adjusted dynamically based on available storage and user request arrival rates. (Fig. 9). Whenever a grid site needs a popular file that is not stored locally, the request will be sent to the Local Server.

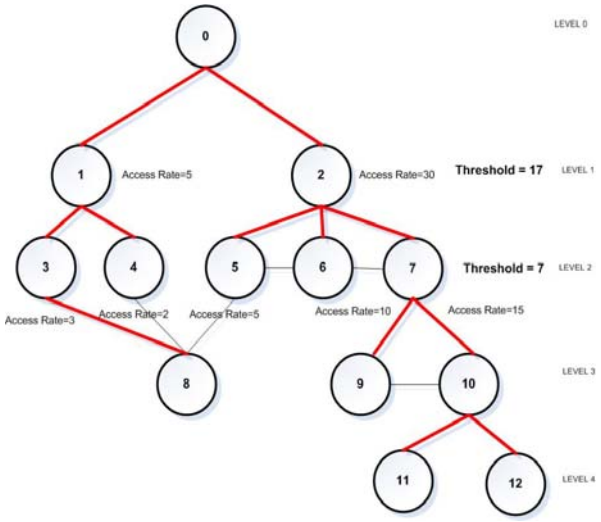

Figure 9: The threshold value is calculated in each level

The Local Server queries it's Replica Catalog through its Replica Manager for determining which grid sites have the requested replica. Note that every time the

Local Server receives a file request, it stores "LFN of that file, PFN of the requester site". So each request will be stored as follows in Replica Catalog: (File ID, Site ID). As mentioned before, another database is introduced which registered the information like "Level ID "and "Depth-first order" of each node. By the depth-first order, each node is aware of all of its descendents which are located at its subtree. So the Local Server is aware of all nodes that are located at its subtree.

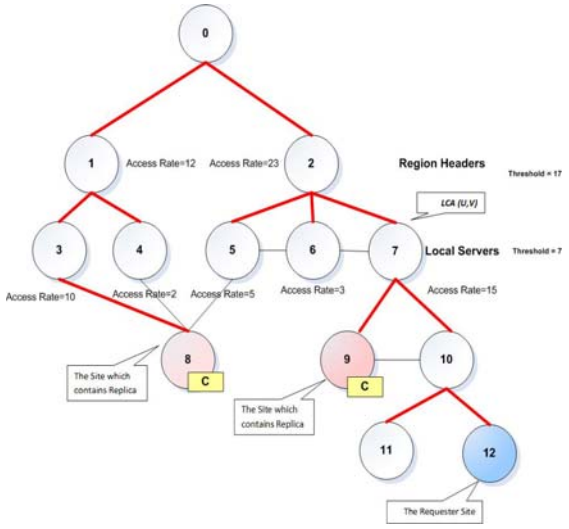

Figure 10: The requester site and the target sites

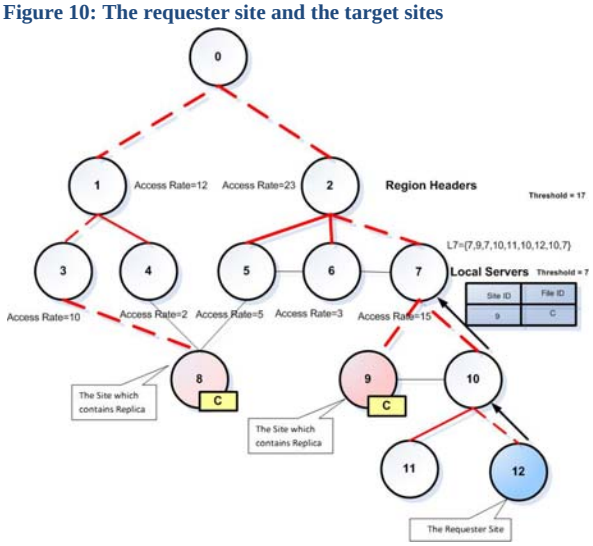

Figure 11: The request is sent to the Local Server

Among the grid sites which are selected as the candidate sites, the sites which have minimum distance from the requester site are chosen. Note that the grid sites that contain the desired replica are called the target sites. As shown in Fig. 10, assume that the grid sites with Site ID=8 and Site ID =9 contain the requested replica. First the proposed algorithm maintains the depth-first order of the Local Server. If the target site is involved in this order then they are selected for next step considerations. Otherwise, the proposed algorithm maintains the order of the Regional servers for further investigation. As shown in Fig. 11, the paths between the requester site and the target sites are demonstrated by dashed lines. The request that came from Site 12 at the second hop through its path, meets grid site 7 as local server at its region. So the grid site 7 checks its depth-first order and finds the target site in its area. As mentioned before, if the required replica is found within the region the job completion will be fast. Therefore, the proposed algorithm selects Site 9 as the target site. In the next step of our algorithm the Replica Selector is called by the RM to compute the Round Trip Time (RTT) and communication cost between the requester site and the target site. As mentioned before each request has specific deadline, so our proposed algorithm estimates the ability of responding the job before its deadline. If (request's deadline > RTT) then the user can access the file remotely. Otherwise the replication will be performed. Now the RM invokes Reservation Manager. If it doesn't succeed in making reservations on requester site, the proposed algorithm finds *lca* of the requester and the target site. As shown in Fig. 12, by considering the depth-first order of grid site 7, the lca (9,12) is the grid site with SiteID =7.

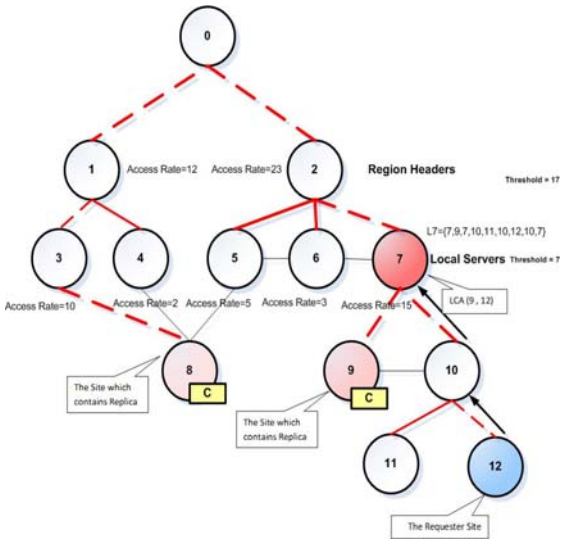

Figure 12: The LCA (the requester site, the target site)

The access load of the target node which is located on LCA is maintained. This access load is compared with the level's threshold value. If its access load exceeds the threshold, then one of its sibling nodes whose load is lesser than the threshold value is selected as the best candidate. (Fig. 13)

It should be considered that, if none of the grid sites, which are located at Local Server and Regional Server's zone do not have the desired replica, the

request will be sent upward the tree. The proposed algorithm is shown in Fig. 15.

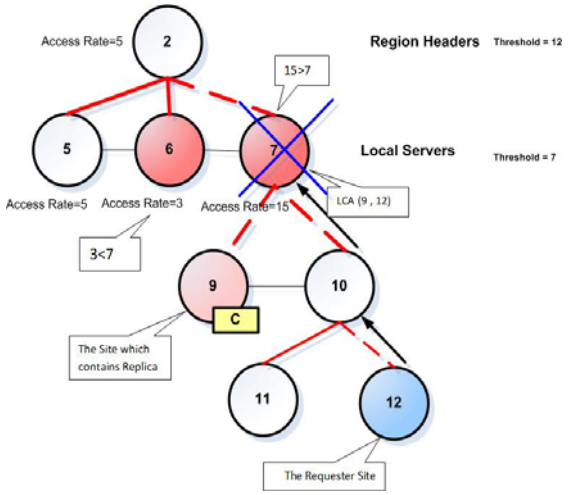

Figure 13: The sibling node is selected

Phase 3: Replacement

As mentioned before the threshold controller component controls the threshold value at each level of our structure. The threshold value is decreased or increased by the difference between the current and previous average aggregated access costs of replica servers. Once the threshold value is updated, the available storage capacities of the replica servers are checked. Further adjustment of the threshold value is done based on the available capacities at the replica servers. Every time the threshold controller checks the available free space of replica servers. Every node is located in specific level. If the storage load of each node exceeds the threshold value of its level, then the threshold controller invokes the proposed algorithm to evacuate that node according to Least Recently Used (LRU) replacement policy. Furthermore, replacement is performed, when a remote replicas has been selected for replication to the target site's storage element. The storage element might not have sufficient spare capacity. In this case, one or more replicas must be deleted by LRU algorithm. (Fig 14.)

5. Performance Evaluation

1) Simulation tool

OptorSim is used as the simulator tool to evaluate the performance of our proposed algorithm. OptorSim [35] is a simulation package written in Java. It was developed to study the effectiveness of replica optimization algorithms within a Data Grid environment [36] and to represent the structure of a real European Data Grid [37]. The structure [38] of OptorSim is illustrated in Fig. 16. The simulation was constructed assuming that the Grid contains several sites; each consists of zero or more Computing Elements (CEs) and zero or more Storage Elements (SEs). CEs run jobs by processing data files, which are stored in the SEs. A Resource Broker (RB) controls the scheduling of jobs to Grid Sites, and schedules jobs to CEs according to scheduling algorithm. Each site handles its file content with Replica Manager (RM), within which a Replica Optimizer (RO) contains the replication algorithm which drives automatic creation and deletion of replicas [33].

Phase 3

sl is defined as : (the desired percentage of storage use / the actual percentage of storage used) at each replica server

T: Threshold value

1. The threshold controller controls the threshold value at each level *i*
2. foreach (node *j* in level *i*)
   checks the sl(j)
3. The threshold value on level *i* is updated

   $T = \frac{\sum_{j=1}^{n} sl\ (j)}{n}$ *(where n is number of node at level i)*
4. foreach (node j in level i)
   if (sl (j) > T) Then Do Replacement
5. If (target site doesn't have enough free space)
   Then Do Replacement
6. Replacement:
   Sort Files in SE using LRU
   Foreach (file f_i in SE)
   {if (file duplicated in other site within Region)

   Then Delete f_i

   If (Enough Space to share new Replica) Break}

Figure 14: Replacement (Phase 3)

Each job has a set of files it may request. The order in which those files are requested is determined by the access pattern. The following access patterns were considered in OptorSim [36]:

Sequential: the set is ordered, forming a list of successive requests.

Random: files are selected randomly from a set with a flat distribution.

Unitary random walk: set is ordered and successive files are exactly one element away from the previous file, direction is random.

Gaussian random walk: similar to unitary random walk, but files are selected from a Gaussian distribution centered on the previous file.

RPGB Algorithm

Inputs: Grid Topology, Coding this topology with Graph traversal algorithms, Bandwidth and Storage Space, d as file's deadline)

Outputs: Find Best Candidate for replication, Load Balancing, Job execution time, number of replica, Remote file access, local file access.

Method:

1. Submit jobs to grid
2. Every request Sends to Replica Manager of Masters : Master of Site, Regional servers
3. Replica manager query Replica Catalog to determine which grid site contains the desired replica
4. If the file not found in lower level its Manager Send Request to upper level
5. When we want to replicate, according to Site Storage Space use Replication or Remote access.
6. Determining the path between source and destination
7. Compute the Lowest Common Ancestor between source and destination.
8. Considering the target site's access load to balance the load on grid environment. If the access load exceeds from threshold then compare the access load of sibling nodes
9. Replicate on the node with minimum access load
10. Execute the jobs

Replica Optimizer

1. Compute the threshold value on each level:

The initial threshold value is set based on the average aggregated access count at the replica servers in each level

Threshold = $\frac{Total\ number\ of\ aggregated\ access\ count\ on\ level_i}{number\ of\ replica\ servers\ at\ level_i}$

2. for each file f_i

{if (Freq(f_i)) >= Threshold of node's level {

Mark the file f_i as to be replicated } FileID = f_i }

3. The information like (FileID,SiteID) & (SiteID,level,depth-first order) is retrieved from the Replica Catalog

4. Each Master node searches for the target nodes in its subtree

5. The sites which have minimum distance from the requester site are chosen.

6. Foreach (grid site 's' in list of target sites)

Replica Selector compute the RTT through ABWE

7. If (request's deadline > RTT) then access remotely & terminate optimizer

8. If (Req file size > SE of storage site) then *lca* = LCA (source,target site)

9 . Compare the access load of *lca* with the threshold value

if (al (*lca*) > threshold then (if al (*lca*'s sibling node) < threshold then select *lca*'s sibling as the Best Candidate)

else select ***lca*** as the Best Candidate

10. Perform replication

Figure 15: The proposed RPGB algorithm (Phase 2)

There are two types of algorithms in OptorSim: the scheduling algorithm used by the RB to schedule jobs to CEs and the replication algorithm used by RM at each site to manage replication. Each scheduling and replication algorithm is implemented as a separate Resource Broker and Replica Optimizer class respectively. We have made changes only in Replica Optimizer Class and the default Resource Broker class is used.

There are three options for Replication Algorithms in OptorSim. First, one can choose No Replication which never replicates a file and all replicas are taken from the master site where the data were produced at the beginning of the simulation and the distribution of files does not change during simulation. Second, one can use LRU or LFU algorithm that always tries to replicate and, if necessary, deletes Least Recently Used files or Least Frequently Used files. Third, one can use an economic model in which algorithm only deletes files if they are less valuable than a new file. There are currently two types of the economic model: the binomial economic model, where file values are predicted by ranking the files in a binomial distribution according to their popularity in the recent past, and the Zipf economic model, where a Zipf-like distribution is used instead [38].We have compared our proposed algorithm with all of these algorithms.

2) Configuration Files

There are four configuration files used to control various inputs to OptorSim. These are as follows [38, 39]:

2.1 Simulation parameter file

It contains various simulation parameters which the user can modify like the names of the grid configuration file and the job configuration, number of jobs, the scheduling strategy for the RB, the optimization algorithm, the files access pattern, a GUI and statistics parameters, and some other important parameters.

2.2) Grid configuration file

It describes the Grid topology and the content of each site; that is, the resource available and the network connections to other sites.

The grid configuration that we have used in our simulation is the CMS Data Challenge 2002 test bed [40] (Fig. 17). For the CMS test bed, CERN and FNAL were given SEs of 100 GB and no CEs. All master files were stored at one of these sites. Every other site was given 70 GB and 50 GB of storage and a CE with one worker node.

2.3)Job configuration file

It contains information on the simulated files like size of each file and its identifier, information on jobs like list of files needed for each job, the probability each job runs and the site policies for each site. In our simulation there are six job types.

2.4) Bandwidth configuration file

The bandwidth configuration file is used to describe the background network traffic. It is a site by site matrix which gives, for each pair of sites, the name of the data file containing the relevant bandwidth information and also the time difference between the reference time zone and the source site.

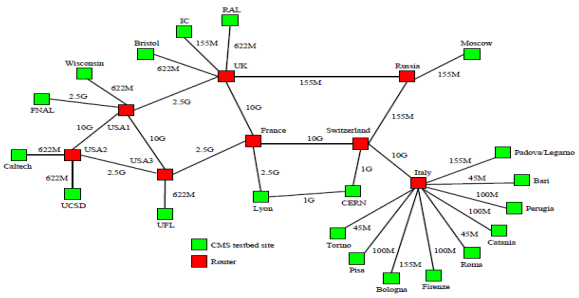

Figure 17. CMS Data Challenge 2002 grid topology [40]

3) Simulation results

The RPGB algorithm was compared with No Replication, LRU, LFU, and Modified BHR algorithms. We have introduced these algorithms in the last section.

3.1) Final results and discussion

As mentioned before the CMS Data Challenge 2002 test bed has been used in our simulation. The simulated grid used in our experiments has 20 sites, 18 of them have Storage Element (SE) and Computing Element (CE) and 2 of them have only SE. The capacity of sites 14 (CERN) and 19 (FNAL) that only have SE's are 100 GB (all master files are stored in these two sites at the beginning of simulation) and the other ones are 70 GB and 50 GB. The SE's of Regional Server are 70 GB. Also there are 8 routers that do not have SEs and CEs. The general simulation parameters are shown in Table 1. We have compared our proposed algorithm with 4 of existing algorithms: No replication, LRU, LFU, and Modified BHR [28]. The first three algorithms are implemented in optorsim. The Modified BHR [28] and RPGB algorithms are implemented by us. The simulation results for the different access patterns are shown in Figures 18, 19 and 20.We ran six jobs totally 100 times and evaluated the impacts of file access pattern. We tested RPGB and the other algorithms in 2 types of access pattern: 1.Random Access, 2.Random Zipf Access. The performance evaluation metrics that we used in our simulation are: Mean Job Execution Time, Effective Network Usage (ENU) and Average Storage Usage.

Table 1.General Simulation Parameters

Parameter	Value
Number of sites	20
Number of Storage Elements (SEs)	20

Number of Computing Elements (CEs)	18
Number of routers	8
Storage capacity at each site (GB)	50, 70, 100
Number of jobs	100
Number of jobs types	6
Number of experiments	10
Job delay (ms)[a]	2500
Size of single file (GB)	1
Total size of files (GB)	97
Access history length (ms)[b]	10^6
Minimum bandwidth between sites (Mbit/s)	45
Maximum bandwidth between sites (Mbit/s)	10000

a. The job delay is the interval in ms between the RB submitting each job.
b. Determines the time period over which the past file access history is considered.

3.1.1)Mean Job Time of all Jobs on Grid

The mean job time of all jobs on grid is defined as the combined total time in milliseconds of all the jobs run divided by the number of jobs completed. (Eq. 2)

$$MJET = \frac{\sum_{i=1}^{n} Job\ Arrival\ Time - Job\ Departure\ Time}{Number\ of\ Jobs\ Completed} \quad (2)$$

Note that for all the components, total job time is defined as the sum of the entire individual job times, including their queuing times [39]. We have compared Mean Job Time of our proposed algorithm with other existing ones. The comparison results are shown in fig 18. The simulation results show that RPGB has the lowest value of Mean Job Execution Time in both Random and zipf access patterns. The reason is because of the nodes at upper level like: the Local Server and the Regional Servers can be aware of the location and the level of their children and descendent nodes. So every Regional server aware of the grid sites which are placed in its Region. According to phase 2 of our proposed algorithm, the most popular file is chosen to replicate. On the other hand the minimum distance between the requester site and target sites is computed. Then the proposed algorithm computes the RTT between these sites, and estimates the ability of responding the job before its deadline. By considering these factors, the algorithm tries to replicate the desired file on site which is located nearby the client, and could respond the job before its deadline. So at the time of execution, jobs will have their required files locally. One of the important factors that decrease the grid site's job execution time is having their required files locally stored on their storage element. It should be noted that, according to zipf access pattern, a few files are requested many times. So, as mentioned before, in our proposed architecture the physical location of studied sites and the files that requested by them are registered in specific databases. Additionally, the proposed algorithm selects the most popular file. By this features the proposed algorithm has the lowest value of Mean Job Execution Time in comparison with LFU, LRU, No replication and Modified BHR. If Random zipf access pattern is used, the Modified BHR works better than LFU, LRU and No Replication. But when Random access pattern is used, LFU and LRU have shorter mean job time and work better than Modified BHR. The Modified BHR stores access history of files, so if files are selected randomly, mean job execution time will not be improved. As Mean Job Execution Time is the most important evaluation metric, RPGB can be considered as the superior strategy.

3.1.2)Effective Network Usage (ENU)

This is effectively the ratio of files transferred to files requested, so a low value indicates that the optimization strategy used is better at putting files in the right places [39]. It ranges from 0 to 1. It can be measured by using equation (3).

$$ENU = \frac{N_{remote\ file\ accesses} + N_{file\ replications}}{N_{remote\ file\ accesses} + N_{local\ file\ accesses}} \quad (3)$$

Through the graph search algorithms like: BFS and DFS, which traverse our graph structure (Phase 1), every node knows its location among its sibling and child nodes, so the required file could be replicated from the sites which are nearest to it. By this assumption the bandwidth consumption is minimized and used effectively. The No Replication strategy performs the worst because it always accesses files remotely. LRU and LFU are better than Modified BHR because the replica is present in the entire site if there is free storage space. (Fig. 19)

3.1.3) Average storage Usage

As shown in Fig. 20, the average storage usage in modified BHR is lesser than RPGB because in Modified BHR files can be stored in a particular site instead of storing them in several sites. Therefore the storage usage can be reduced.The Modified BHR checks access history of files before replicating them and find the storage element which has accessed the files at most. In our proposed algoritm files can be stored in several sites. It should be considered that, if files are selected randomly then the Modified BHR would not be the best solution. If we use zip-f distribution, the Modified BHR performs better than the others, because few files being requested many times by this algorithm. On the other hand, our proposed algorithm checks the storage load of replica servers and every time compares their load with threshold value to evacuate servers from least recently used files. So it uses the storage elements lesser than other three replication algorithm.

6. Conclusion and Future works

In this paper we described our proposed architecture, Replica Placement on Graph-Based data grid Architecture. As a true representation of a grid is a general graph in which there is no central node designated as a root node, and each node can be connected with any number of nodes. Therefore in this paper, we consider a grid with graph-based topology. But at the first step of our proposed algorithm, this structure is converted to hierarchal

tree structure due to better managing of replica servers and their related nodes. In the second phase, we determine a suitable selection and placement of replicas on this hierarchical structure. The proposed algorithm takes into account both the access load and the storage load of the replica servers and their sibling nodes before placing the replicas. So the workload among these nodes is balanced and data availability is increased. It also reduces unnecessary replication and improves our performance. The comparison of our algorithm with LRU, LFU and Modified BHR demonstrates that our algorithm has better performance in average mean job time and effective network usage but on the other hand the Modified BHR acts better in average storage usage when zipf access pattern is used. Finally, the important challenge for the replica placement problem is how can we place the replicas in such a way that service quality can be guaranteed for all requests, which will be considered in our future work.

Acknowledgement:

The author would like to thank Iran Telecommunication Research Center (ITRC) for their financial support.

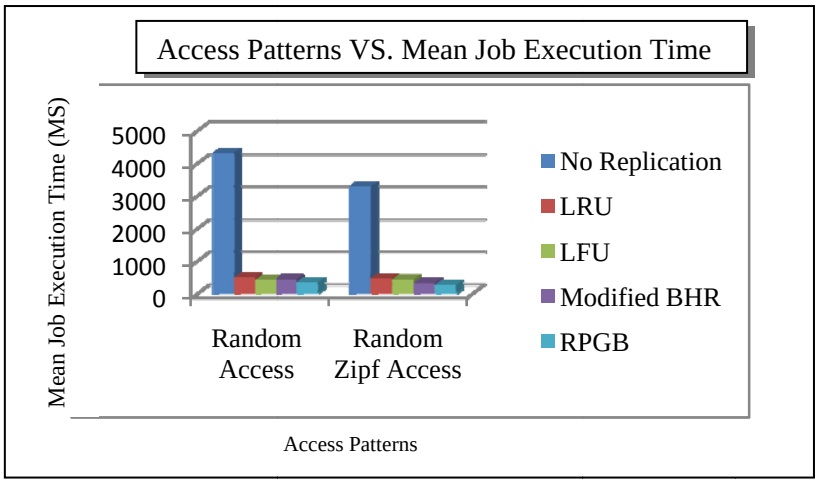

Figure 18: Access Patterns VS. Mean Job Execution Time

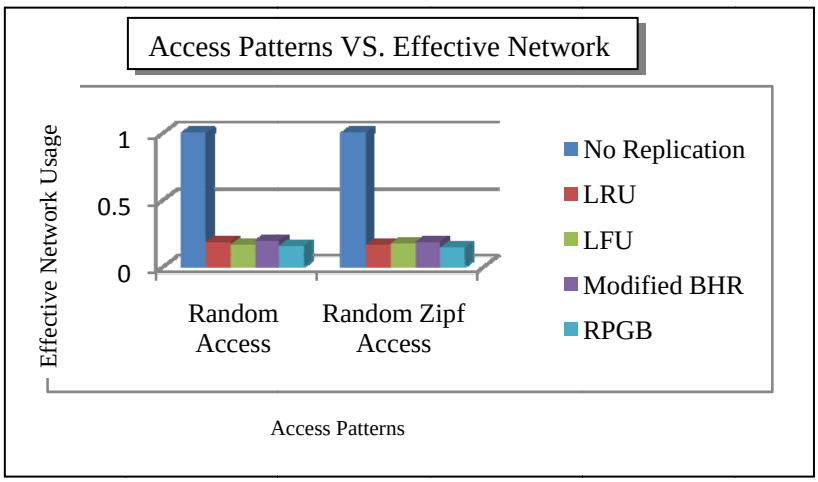

Figure 19: Access Patterns VS. Effective Network Usage

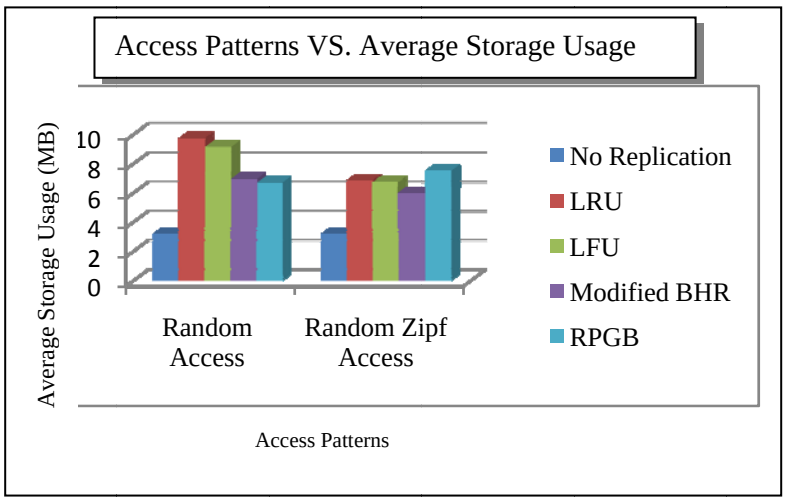

Figure 20: Access Patterns VS. Average Storage Usage

References

[1] I. Foster, C. Kesselman, S. Tuecke, The anatomy of the grid, (2001).

[2] J. Zhang, B.S. Lee, X. Tang, C.K. Yeo, A model to predict the optimal performance of the hierarchical data grid, Future Generation Computer Systems 26 (2010), pp. 1–11.

[3] I. Foster, The grid: A new infrastructure for 21st century science, (2002).

[4] K. Ranganathan, I. Foster, Design and evaluation of dynamic replication strategies for a high performance data grid, in: International Conference on Computing in High Energy and Nuclear Physics, (2001).

[5] H. Lamehamedi, B. Szymanski, Data replication strategies in grid environments, in: ICA3PP, (2002), p. 0378.

[6] K. Ranganathan, A. Iamnitchi, I. Foster, Improving data availability through dynamic model-driven replication in large peer-to-peer communities, in:CCGrid, (2002), p. 376.

[7] R.M. Rahman, K. Barker, R. Alhajj, Replica placement in data grid: Considering utility and risk, (2005).

[8] S. Vazhkudai, S. Tuecke, I. Foster, Replica selection in the globus data grid, in: CCGrid, (2001), p. 106.

[9] H. Stockinger, A. Samar, K. Holtman, B. Allcock, I. Foster, B. Tierney, File and object replication in data grids, Cluster Computing 5 (3) (2002), pp. 305–314.

[10] Y. Yuan, Y. Wu, G. Yang, F. Yu, Dynamic data replication based on local optimization principlein data grid, (2007).

[11] F. Schintke, A. Reinefeld, Modeling replica availability in large data grids, Journal of Grid Computing 1 (2) (2003), pp. 219–227.

[12] H. E. Al Mistarihi1, C. H. Yong, Replica Management in Data Grid, IJCSNS International Journal of Computer Science and Network Security, June (2008), pp. 22-32.

[13] A. Benoit, V. Rehn-Sonigo, and Y. Robert, Replica placement and access policies in tree networks, IEEE *Transactions On Parallel and Distributed Systems*, 12(19), (2008), pp.1614–1627.

[14] T. Hara, Data replication issues in mobile ad hoc networks, In *Proceedings of the 16th International Workshop on Database and Expert Systems Applications (DEXA'05)*, IEEE Computer Society, (2005).

[15] A. Litke, D. Skoutas, and T. Varvarigou, Mobile grid computing: Changes and challenges of resource management in a mobile grid environment, In *Proceedings of PAKM 2004, Lecture Notes in Computer Science(LNCS), Vol.3336*, Vienna, Austria, Springer-Verlag, (2004).

[16] A. Benoit, V. Rehn-Sonigo, and Y. Robert, Replica placement and access policies in tree networks, *IEEE Transactions on Parallel and Distributed Systems*, 12(19), 2008, pp. 1614–1627.

[17] M. Rashedur, B. Ken, and A. Reda, Replica placement strategies in data grid, Journal *of Grid Computing*, (2008), pp. 103–123.

[18] X. Tang and J. Xu, Qos-aware replica placement for content distribution, *IEEE Transactions on Parallel and distributed Systems*, 10(16), October (2005), pp. 921-932.

[19] GriPhyN project, http://www.usatlas.bnl.gov/computing/grid/griphyn/

[20] Q. rasool, J. Li, G. S. Oreku, Sh. Zhang, D. Yang, "A Load Balancing Replica Placement Strategy in Data Grid", Digital Information Management, ICDIM 2008. Third International Conference, Nov. (2008), pp. 751-756

[21] M. Tang, B.S. Lee, C.K. Yeo, X. Tang, Dynamic replication algorithms for the multi-tier data grid, Future Generation Computer Systems 21 (5) (2005), pp. 775–790.

[22] Y. Yuan, Y. Wu, G. Yang, F. Yu, Dynamic data replication based on local optimization principle in data grid, (2007).

[23] A. Abdullah, M. Othman, H. Ibrahim, M.N. Sulaiman, A.T. Othman, Decentralized replication strategies for P2P based scientific data grid, in: Information Technology, ITSim, International Symposium on, (2008), pp. 1–8.

[24] Y. Ding, Y. Lu, Automatic data placement and replication in grids, in: High Performance Computing, HiPC, International Conference on, (2009), pp. 30–39.

[25] Neeraj Nehra, R.B.Patel, V.K.Bhat, Distributed Parallel Resource Co-Allocation with Load Balancing in Grid Computing, IJCSNS International Journal of Computer Science and Network Security, January (2007), pp. 282-291.

[26] A. Horri, R. Sepahvand, Gh. Dastghaibyfard, A Hierarchical Scheduling and Replication Strategy, IJCSNS International Journal of Computer Science and Network 30 Security, August (2008), pp. 30-35.

[27] S. M. Park, J. H. Kim, Y. B. Ko, W. S. Yoon, "Dynamic Data Replication Strategy Based on Internet Hierarchy BHR", in: Lecture notes in Computer Science Publisher, 2004, pp. 838-846.

[28] K. Sashi, A.S. Thanamani, Dynamic replication in a data grid using a modified BHR region based algorithm, Future Generation Computer Systems 27 (2), (2011), pp. 202–210.

[29] Q. Rasool, J. Li, S. Zhang, Replica placement in multi-tier data grid, in: 2009 Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing, (2009), pp. 103–108.

[30] Y.F. Lin, J.J. Wu, P. Liu, A list-based strategy for optimal replica placement in data grid systems, in: 37th International Conference on Parallel Processing, (2008), pp. 198–205.

[31] A. Dogan, "A study on performance of dynamic file replication algorithms for real-time file access in Data Grids", Future

Generation Computer Systems, September (2009), pp. 829-839.

[32] M. Hofling, On Data Placement Issues in Grid Computing Environments, Master's Thesis in Computing Science, November (2008).

[33] R. M. Vozmediano, Application Layer Multicast for Efficient Grid File transfer, International Journal of Computer Science and Applications, Technomathematics Research Foundation, (2009), pp.70-84.

[34] M. Shorfuzzaman, P. Graham, R. Eskicioglu, Adaptive Replica Placement in Hierarchical Data Grids, Journal of Physics: Conference Series **256** (2010).

[35] OptorSim – A Replica Optimiser Simulation, http://grid-data-management.web.cern.ch/grid-data-management/optimization/optor.

[36] W.H. Bell, D.G. Cameron, L. Capozza, A.P. Millar, K. Stockinger, F. Zini, Simulation of Dynamic Grid Replication Strategies in OptorSim, Int. Journal of High performance Computing Applications, 17(4), (2003).

[37] The European DataGrid Project, http://www.edg.org.

[38] D.G. Cameron, A.P. Millar, C. Nicholson, OptorSim: a simulation tool for scheduling and replica optimization in data grids, Proc. Computing in High Energy and Nuclear Physics (CHEP), (2004).

[39] D.G. Cameron, R. Schiaffino, J. Ferguson, A.P. Millar, C. Nicholson, K. Stockinger, F. Zini, OptorSim v2.1 Installation and User Guide, October (2006).

[40] CMS Data Challenge, (2004), http://www.uscms.org/s&c/dc04.

Zeinab Fadaie[1] received her B.S. in computer engineering from center branch of IAU University, Tehran, in 2008; she is currently pursuing her M.S. degree in the Department of Computer and Mechatronics Engineering at the IAU University. Her area of interests includes distributed computing, computer networks and communications, heterogeneous system and grid computing.

Amir Masoud RAHMANI received his B.S. in computer engineering from Amir Kabir University, Tehran, in 1996, the M.S. in computer engineering from Sharif University of technology, Tehran, in 1998 and the PhD degree in computer engineering from IAU University, Tehran, in 2005. He is assistant professor in the Department of Computer and Mechatronics Engineering at the IAU University. He is the author/co-author of more than 80 publications in technical journals and conferences. He served on the program committees of several national and international conferences. His research interests are in the areas of distributed systems, scheduling algorithms and evolutionary computing.

[1] Corresponding author

Science and Research University, Simon Blvd., Ashrafi Esfahani Ave., Tehran, Iran, phone number: +98 (21) 44869730, fax number: +98 (21) 44869744.