MANAGEMENT AND PLACEMENT OF REPLICAS IN A HIERARCHICAL DATA GRID

Ghalem Belalem[1] and Bakhta Meroufel[2]

[1]Department of Computer Science, Faculty of Sciences, University of Oran (Es Senia), Algeria

ghalem1dz@gmail.com

[2]Department of Computer Science, Faculty of Sciences, University of Oran (Es Senia), Algeria

bakhtasba@gmail.com

ABSTRACT

In large systems, data management is only possible through the use of replication techniques. In order to improve availability and ensure fast and efficient access, we propose in this paper a dynamic replication algorithm which takes into account also the placement of data in a hierarchical data grid. Experimental results show the effectiveness of our approach in terms of response time and availability of data in the system.

KEYWORDS

Data Grid, Replication, Data management, Availability.

1. INTRODUCTION

Grid computing is an emerging technology that presents new challenges. The management aspect of large scale, heterogeneity and dynamicity of the sites are the most important [1].

In large-scale systems such as data grids, data access, pose problems of performance and quality of service. The use of replication techniques given can implement solutions with more or less effective in some of these problems [2]. For this we proposed an approach of dynamic replication in a hierarchical data grid[11]. This approach can create or delete the data according to their popularity. In our case, we have attempted, through this paper to propose a dynamic replication approach is to:

- Ensure the desired availability for the data and improve this availability according to the popularity of this data [12].
- Improve system performance in terms of response time, number of replicas.
- Load balancing between the sites of the grid.

The remainder of this paper is organized as follows: in Section 2, we present a state of the art of replication. Section 3 studies the topology of work on how to articulate our approach. In Section 4 we present our approach to dynamic replication. Section 5 shows the different results obtained by the simulations of our approach. The paper ends with a conclusion and a set of perspectives.

2. RELATED WORKS

The literature provides much of the work in the field of management and placement of replicas. In the work [3], the authors compare several dynamic replication strategies in a hierarchical grid. The strategies are compared by measuring (by simulation) the average response time and total bandwidth used. To minimize the cost of communication between the replicas, the authors of [4] used two topologies, hierarchical and ring. The authors tested three scenarios: there is no replica; replicas are placed at the second level of the tree: the first intermediate nodes and the last scenario when the replicas are placed at the lowest intermediate nodes. An economic model based on auctions management (creation and destruction) of replicas is proposed in [5]. In this system, although this model has been successful, due to its ability to determine the most records accessed through the history to access and replicate them accordingly, but did not take into consideration the cost of storage. Other types of algorithms take into account data locality [6]. The servers are grouped into different regions according to the network topology. Communications between nodes within a region should be quite fast. When the data is needed on a server and there is more room to store the algorithm BHR (Bandwidth Hierarchy based Replication) would seek to recover the data in question only if it is not already on one of the nodes in the same region.

3. USED TOPOLOGY

We used a hierarchical grid in this work because the architecture is similar to the existing grid managers [7] [8] [9]. For example, in the LCG project (World-Wide Large Hadron Collider Computing Grid) which consists of 27 countries [10], the system is hierarchical when CERN (the European Organization for Nuclear Research) is the root of this topology, the root is the 0-level. There are 11 sites that are direct son of the root and which have the 1-level. The project EGEE/LCG-2 contains 36 countries and presents an extension of the LCG (provide a diagram of the grid). The main features of this topology is used [10]:

- A Multi-tier hierarchical topology consists of a root, servers and clients;
- The components of the topology are organized as a tree which minimizes the number of forwarded messages;
- The data in the first time is only in the root and it can not be removed wherever is its popularity ;
- Data can be replicated in the servers;
- Clients are the leaves of this topology, they can run read queries with different frequencies;
- Clients can not store the replicas;
- Servers can store or delete the data as required.

The multilevel structure allows the flexible and scalable data sets and users. Figure 1 shows an example of hierarchical model used in our work.

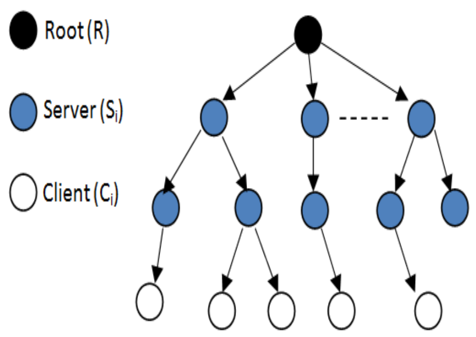


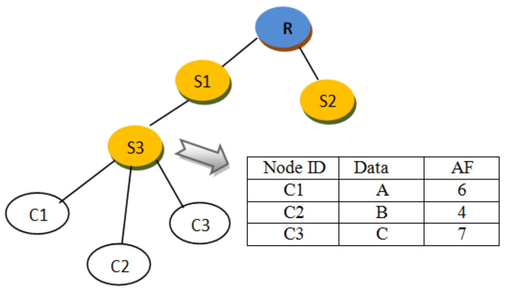
Figure 1. Used topology

4. ROPOSED APPROACH

The principle of our approach to create replicas based on the popularity of such data. In our system. The root is the only component that contains the data, but it applies the replication algorithm to create replicas on servers. The leaves of the tree are the client nodes that can run the query, all nodes except the root and the clients are servers.

Each server stores the number of hits on each of his sons. In Figure 2, we have a system consisting of a root *R*, three servers {*S1, S2, S3*} and three clients {*C1, C2, C3*}. According to the history table stored in the server *S3* (Father clients {*C1, C2, C3*}), the client *C1* has consulted the file "*A*" 6 times (*FA (A, C1)* = 6), the client *C2* accessed file "*B*" 4 times (*FA (B, C2)* = 4) and *C3* consulted the file "*F"* 7 times (*AF (F, C3)* = 7). So if the request of a client on a data exceeds the threshold of replication, then the root replicates this data in the father of this client. If the father can not store the data (for reasons of insufficient storage space) then the root try with the father of the father and so on until it ends all the great fathers of the client.

In the example in Figure 2, if the threshold is 7, then the data *F* will be replicated in the server *S3*. If *S3* can not store this data, then the root is trying to replicate the data *F* in the grandfather *S1*.



Node ID	Data	AF
C1	A	6
C2	B	4
C3	C	7

Figure 2. Exemple 1 of access frequence

Our algorithm is a dynamic replication, a server can delete data with access frequency null or less then the data that we want to store[13].

To detail the replication process applied by the root, we proposed an algorithm (see Figure 3) which consists of two steps:

- Line 01-11: apply for each server in the system.
- Line from 12 to 15, apply by the root.

```
01: For each server $S_i$
// List_Son ($S_i$): list of the sons of the server $S_i$
// List_Data ($S_i$): list of all data requested by the sons of $S_i$
// List_Rep ($S_i$): list of data to be replicated in Si
02: For each node $Son_j \in$ List_Son of the server $S_i$
03:    For each Data $D_i \in$ List_Data($S_i$)
04:        If AF($D_i$, $Son_j$) $\geq$ Threshold ($D_i$) Then
05:            List_Rep($S_i$) = List_Rep($S_i$) + ($D_i$)
06:        End If
07:    End For
08:  End For
09: Scheduling List_Rep ($S_i$) in ascending order of $AF$.
10: Sends List_Rep ($S_i$) to the root $R$.
11:  End For
The Root :
12: For each data $D_i \in$ List_Rep($S_i$)
13:        Rep_Procedure ($D_i$, $S_i$)
14:  End For
```

Figure 3. Algorithm of replication

Each server S_i in the system has a view of his sons (List_Son (Si)), their requests and their access frequency *AF* to each data. In each period, the server builds a list List_Rep (Si) that contains data that has a access frequency (AF) that exceeds the replication threshold of the subject data (see algorithm of Figure 3: line 01 to 11). The AF (D_i, Son_j) is the access frequency of node Son_j on the data D_i. The list List_Rep (S_i) will be classified in ascending order of access frequency and will be sent to the root.

The root is trying to replicate each data D_i of the list List_Rep (S_i) of the server S_i by running the procedure *Rep_Procedure* (D_i, S_i) (see algorithm of Figure 3: line 12 to 14). If the server S_i can not store data at home (insufficient memory space), the S_i verifies the ability to free up space by deleting data (replica) unsolicited or rarely (their *AF* is zero or smaller the *AF* of the new data), this verification is performed by the procedure *Possible_Space* (S_i).

If the root is unable to replicate the data D_i in the server S_i, it repeats the same procedure of replication *Rep_Procedure* but with the father of S_i (*Father* (S_i)) and then with *Father (Father (Si))* and so on until reaching the root (see Algorithm 2).

```
Rep_Procedure ($D_i$, $S_i$)
01: If $S_i \neq$ Root then
02:    If Space_Available($S_i$) $\geq$ Size ($D_i$) Then
03:            store $D_i$ in $S_i$
04:      Else If Possible_Space ($S_i$) $\geq$ Size ($D_i$)  Then
05:            store $D_i$ in $S_i$
06:          Else  Rep_Procedure ($D_i$, Father($S_i$))
07:      End If
08:    End If
09 : End If
```

Figure 4. Rep_Procedure Algorithm

To improve the performance of our system, we used a different strategy to calculate the frequency of a given access.

$$AF(D_i, N_j) = \sum_{j=1}^{m} AF(D_i, Son_j)$$

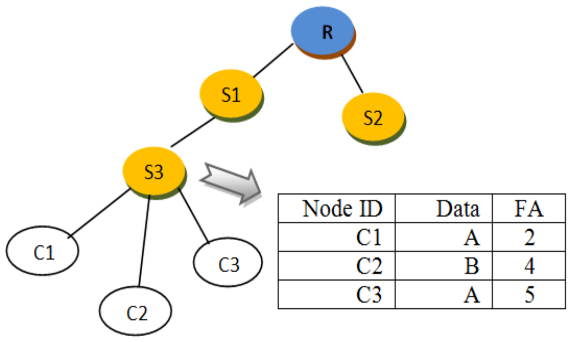
Where

- m: is the number of the sons of the server N_i.
- Son_j: the son of the affected server N_i.
- D_i: the data identified by i.

The access frequency of the data is defined as the sum of the access frequency of all sons who ask the same data. To fix ideas on this definition, we have the example of Figure 5, the access frequency of the data X is the sum of access frequencies of C1 and C3.

FA (A, S3) = FA (A, C1) + FA (A, C3)

So if the threshold for replication of the data X is 7, the root replicates this data in the father S3.



Node ID	Data	FA
C1	A	2
C2	B	4
C3	A	5

Figure 5. Exemple 2 of access frequency

5. EXPERIMENTAL RESULTS

To validate our approach of dynamic replication and measure its performance under different constraints, we conducted a simulator that creates a hierarchical topology and apply our strategy of replication. Our simulation compares the performance of:

- Approach of non-replication: where all data are in the root and it will never be replicated elsewhere (No_Rep).
- Our approach of replication in which the access frequency is usually calculated using the technique as shown in Figure 2 (RD_O).
- Our approach of replication in which the access frequency is calculated using the technique of sum frequency of son as shown in Figure 3 (RD_S).

5.1 Impact of number of queries

The number of requests plays an important role on the performance of different approaches. We realized our simulations in a system with: number of levels is 6, number maximum of sons is 3, time of simulation is 200 seconds, threshold of replication is 3 and the number of data is two.

Figure 6 illustrates the impact of the number of queries on the response time of the system. We note that the response time is fixed in No_rep but other approaches minimize the response time.

The second experiment explains the results obtained in the first experiment (see Figure 7). We measured the number of replicas in the system with different numbers of queries. The results show that the number of replicas in the approach RD_S is high compared to the approach RD_O because in the first approach, the root call the procedure of replication more than the second approach.

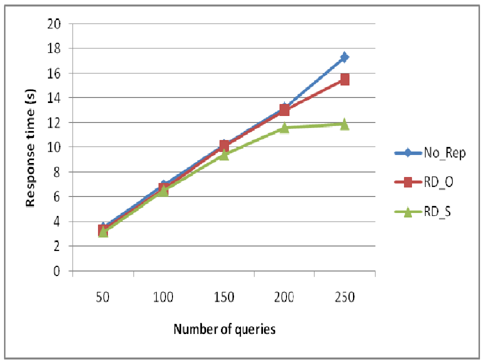


Figure 6. Number of queries vs Response time

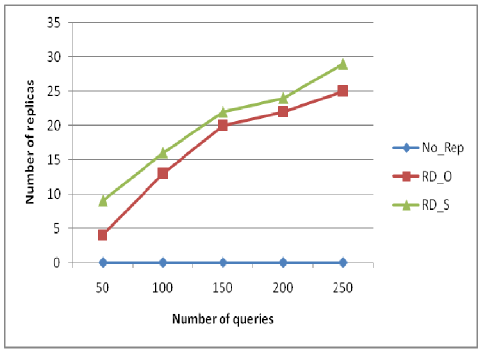


Figure 7. Number of queries vs Number of replicas

5.2 Impact of number of levels

We have also studied the impact of number of levels (the height of the tree) on system performance. The system is characterized by: number maximum of sons is 3, there is only one data and the threshold if replication is 3.

According to experiments, the height of the tree affects the response time. The results in Figure 8 indicate that if the number of levels increases, the response time increases for all approaches because the distance between clients and servers that provide the data increases.

We noticed that the number of replicas varies if the number of levels varies. In Figure 9 the number of replicas in the approach No_Rep remain zero. For both approaches RD_O RD_S and, as the number of levels increases, the number of replicas increases to some level where the number of replicas will decrease because it will be a lot of clients requesting the same data and it not exceed the predefined threshold.

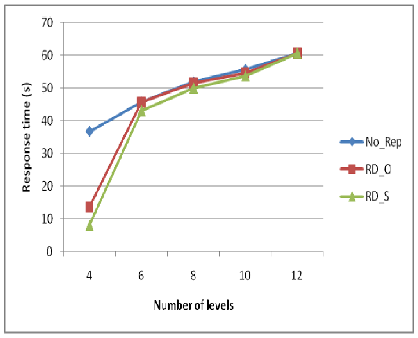


Figure 8. Number of levels vs Response time

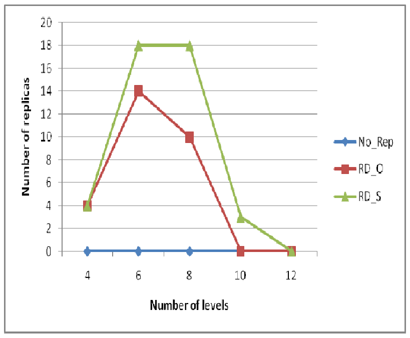


Figure 9. Number of levels vs Number of replicas

5.3 Impact of the threshold of replication

In the last series of experiments, we studied the impact of the threshold of replication on the performance. The system has 5 levels, 5 sons at maximum, the number of queries is 100 and there only one data in the system.

According to Figure 10 which presents the results, we studied the influence of the threshold of replication on the response time, we noticed that increasing the threshold increases the response time in both approaches and RD_O RD_S. a small threshold of replication mean the root will rapidly execute the replication procedure to create a new replicas. In case of a big threshold, the server will take a long time to pass this threshold witch increase the response time, and this why the approach (RD_S) gives better results.

To understand the results obtained in the last experiment (impact threshold of response time), we studied the number of replicas with different threshold of replication and the results are shown in Figure 11. A high level of replication delays the triggering of replication procedure which minimizes the number of replicas and therefore increases the response time.

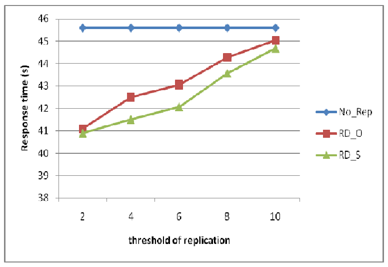


Figure 10. Threshold of replication vs response time

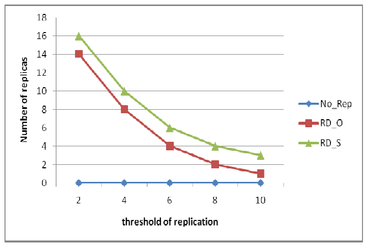


Figure 11. Threshold of replication vs Number of replicas

6. CONCLUSION

In this paper, we proposed a dynamic replication strategy that takes into account the access frequency of the data, our approach can also choose the placement of new replicas in an efficient manner. Experimental results show that the proposed approach improves system performance. In our perspective, we propose the improvement of our manager dynamic replication of data by taking into account the aspect of consistency of the replicas for writing queries. We propose in the near future to implement our proposal on a real grid-based middleware Globus.

REFERENCES

[1] P. Asadzadeh, R. Buyya, and C. Ling Kei. Global Grids and Software Toolkits : A Study of Four Grid Middleware Technologies. Wiley Press, 2005.

[2] C. Baru, R. Moore, and A. Rajasekar. The SDSC storage resource broker. In CASCON'98 conference, volume 38, pages 5–10, Toronto,Canada, 1998.

[3] K. Ranganathan, I. Foster: Identifying Dynamic Replication Strategies for a High Performance Data Grid. Dans Proc. of the Second International Workshop on Grid Computing (2001).

[4] H. Lamehamedi, B. Szymanski, Z. Shentu, E. Deelman: Data replication strategies in Grid environments. In Proceedings of the 5th International Conference (ICA3PP'02). IEEE Press, Los Alamitos, CA, 2002.

[5] W. Bell, D. G. Cameron, R. Carvajal-Schiaffino, A. P. Millar, K. Stockinger, F. Zini : Evaluation of an Economy- Based File Replication Strategy in Data-Grids. Dans Third International Symposium on Cluster Computing and the Grid (CC-GRID) (2003).

[6] S. M. Park, J. H. Kim, Y. B. Ko, W. S. Yoon: Dynamic Data Grid Replication Strategy Based on Internet Hierarchy. Dans GCC (2) (2003), pp. 838–846.

[7] W. B. David. Evaluation of an economy-based file replication strategy for a data grid. In *International Workshop on Agent based Cluster and Grid Computing*, pages 120–126, 2003.

[8] W. Hoschek, F. J. Janez, A. Samar, H. Stockinger, and K. Stockinger. Data management in an international data grid project. In *In Proceedings of GRID Workshop*, pages 77–90, 2000.

[9] K. Ranganathan, A. Iamnitchi, and I.T. Foste. Improving data availability through dynamic modeldriven replication in large peer-to-peer communities. In *In 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 376–381, 2002.

[10] Worldwide LHC Computing Grid. http://lcg.web.cern.ch/lcg/.

[11] M. Lei, S. Vrbsky, *An on-line replication strategy to increase availability in Data Grids.*Future Generation Computer Systems, 24(2): 85-98, 2008.

[12] M. Lei, S. Vrbsky, *A data replication strategy to increase availability in Data Grids, in: Grid Computing and Applications*, Las Vegas, NV, 2006, pp. 221–227.

[13] K. Mohammed Madi and S. Hassan, *Dynamic Replication Algorithm in Data Grid: Survey*, International Conference on Network Applications, Protocols and Services 2008 (NetApps2008), ISBN 978-983-2078-33-3, on 21 - 22 Nov. 2008.