Available online at www.sciencedirect.com

ScienceDirect

AASRI Procedia 5 (2013) 147 – 155

AASRI Procedia

www.elsevier.com/locate/procedia

2013 AASRI Conference on Parallel and Distributed Computing Systems

Managing Data Replication and Placement Based on Availability

Bakhta Meroufel[a]*, Ghalem Belalem[b]

[a]bakhtasba@gmail.com, [b]ghalem1dz@univ-oran.dz
Department of Computer Science
Faculty of Faculty of Exact Sciences and Applied
University of Oran (Es Senia), Algeria

Abstract

The replication of data across multiple sites of data grid is an effective solution to achieve good performance in terms of load balancing, response time, and improving data availability. To get the maximum gain that can make the data replication, their placement strategy in the system is critical. This paper proposes a replication strategy based on availability. It proposes also a placement and replacement strategies of replicas that ensures the desired availability with the minimum replicas despite the presence of nodes failures and without overloading the system. The results of our experimentations confirm that the proposed approach reaches its objectives.

Keywords: replication; availability; data grid; similarity; placement; clusters.

1. Introduction

A data grid is a promising technology for distributed systems in general. It offers the availability of a large amount of data [16], the problem in this type of grid environment is to ensure continuous availability of data and respond to user requests as soon as possible, considering the geographical distribution of nodes and the

* Bakhta Meroufel.
E-mail address: bakhtasba@gmail.com.

popularity of data. The most often solution used to solve this problem is the replication [15]. Data replication is a technique of creating identical copies of data (files, databases, etc.) in geographically distributed sites. Each copy is called a replica [7]. The aim of our work is to ensure the desired availability with minimum replicas without degrading system performances. This goal is possible with a placement strategy that takes into account: the desired availability, the stability of nodes in the system and the failures.

We present the following contributions: Section 2 introduces some related researches on the replication and placement of replicas in distributed systems and data grids. In Section 3, we define the used topology. Section 4 presents our contribution, namely a proposal for an efficient dynamic replication approach which takes into account the placement of replicas and failures in the system. Section 5 presents the experimental results of our different simulations. The last section summarizes the paper and gives a short overview of future works.

2. Related works

Various placement strategies have been proposed for replicas in grid that have a hierarchical topology. The proposed model in [3] minimizes both the cost of communication (data transfer) and storage (cost of placing replicas). The paper [5] uses two replica placement models. In the first, the authors use the reading/writing costs as placement parameters. In the second, they also take into consideration the burden of storage required by each node. In [11], the authors address the placement problem by ensuring load balancing between nodes and number of replicas at the same time. In [5, 11], the number of nodes of the grid is fixed, which is not an easy condition to meet in the grids where the nodes usually connect and disconnect unpredictably. In [8], the replicas are created on the nodes that receive a large number of requests. They will then be deleted at the request of the user or when the data are no longer used, or where storage space is full and replicas highest priority must be created. The replica placement decision is based on a cost model. The authors of [6] implement a replication approach in distributed networks. The network is modeled as colored nodes structured in bi-directional graph. When creating replicas, nodes containing identical replicas have the same color. The principle of the protocol [6] is that each node chooses a color graph to minimize the distance between nodes of different colors and maximize the distance between nodes of the same color. The problem with this approach that it considers only the case of one data for each node that is to say that the node can have only one color.

3. System Model

Our used system model is a cluster federation with a single root to link between thus clusters. Several systems have this topology such as Internet [10] and DIET [1]. Figure 1 shows an example of the system with its components.

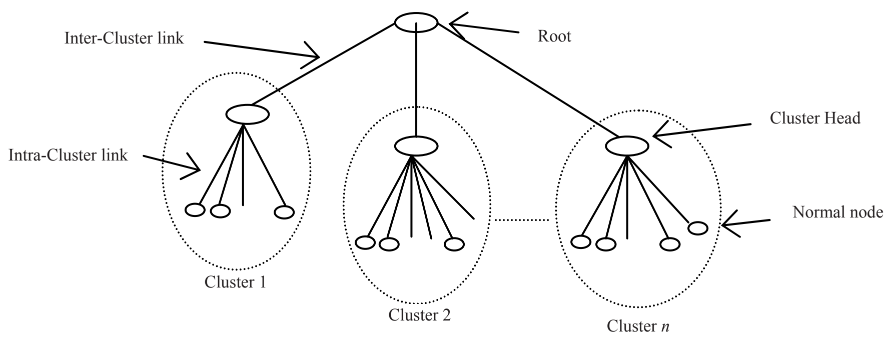

Fig. 1. Used topology.

The root in the topology is used to bind the various clusters with each other. The Cluster-Head (CH), which represents members of the cluster, has a routing table that manages the nodes within the cluster. It also contains metadata and information about the replicas existing in the cluster. The other nodes are storage elements; they contain one or more replicas of various data. In the system, nodes have predictive behavior [4] and the fault detection is based on the messages of life [9]. If a failure is detected, the auto stabilization [2] will be triggered to keep the topology connected, in other words.

4. Proposed approach

The present paper uses a new model of replication and placement of data. The principal objective of this model is to minimize the number of replicas that ensures certain availability degree without degrading the performance of the system. Our strategy takes into account the different and independent stability nodes in contrast to most work in the literature [10] [13] [14] [17]. Each Cluster_Head contains a replication controller to manage the replication and placement of replicas in the cluster.

4.1.Number of necessary replicas

To calculate the availability Availj of a data j in the system, we can use the formula proposed in [12]:

$$Avail_j = 1 - \prod_{i=1}^{\alpha}(1 - p_i) \tag{1}$$

Where

- j: data;
- Availj: the availability of the data j. $0 \leq Availj \leq 1$;
- α: the number of replicas of the data j. $\alpha \in N$;
- pi: the stability of the node i where the replica of data j is stored. $0 \leq pi \leq 1$. In the rest of the paper, we will make the difference between the stability of the node and the stability of its data, we note the first STAB and the second p. So if data j is stored in node i then: $\mathrm{STAB_i} = \mathrm{p_j}$.

In the case of a system where nodes have the same independent stability, then the formula (1) will be [9]:

$$Availj = 1 - (1 - p)^{\alpha} \tag{2}$$

Formula (2) allows us to estimate the value of α that satisfies the availability Avail for data j.

Many of works that exist in the literature [10], [14], [17] assume that the nodes have the same degree of stability in order to use the formula 2. In the work [13], the author used nodes of different degrees of stability, but he proposes to replicate the data in the nodes of the same class of stability. This proposal allows him to use the formula 2 to calculate the number of replicas necessary to meet the desired availability. In our system, nodes have different and independent stabilities; this is the case of existing systems. To calculate the number of necessary replicas α that ensure the desired degree of availability Avail, we have three possibilities:

- Optimistic: is the case where all the replicas of the data i are stored in the nodes of a good stability. So the availability Availi will be assured with the minimum number of replicas. The αOp is the number of replicas necessary to assure the desired availability in the optimist case. It is calculated by the formula 2 where p is the best stability in the system.
- Pessimistic: is the case where all replicas of the data i are stored in the nodes of poor stability. So the availability Availi will be assured with the maximum number of replicas. The αPes is the number of replicas necessary to assure the desired availability in the pessimist case, it is calculated by the formula 2 where p is the minimum stability in the system.
- Hybrid is the case where the replicas are stored in nodes of different degrees of stability. So the availability Availi will be assured by crating αHyp replicas in the system. So αHyp will be in the range [αOp, αPes]. The number αHyp can not be calculated after specifying the placement of replicas because you have to select the participating pi in the formula 1.

To ensure the degree of availability Availj to data j in a system that contains different nodes of different stabilities, several solutions can be proposed:

- Create αPes replicas, that way we will be sure that availability is respected. But in an environment where the creation and management of replicas are expensive, this solution will not be effective.
- Create αOpt replicas, so the number of replicas is the minimum, but this solution requires that the number of nodes that have the best stability is greater than or equal to αOpt. This solution overloaded the most stable nodes which increases the number of lost files in case of failure and also increases the recovery time.

Our approach consists of creating the minimum possible replicas (minimum then αPes) without overloading the nodes of good stability and without degrading system performance. The principle of our proposal is as follows:

Each Cluster-Head (CH) specifies a certain degree of availability AvailD to ensure in its cluster. The AvailD is estimated using the history of data itself and its importance (popularity) in previous periods. After the calculation of AvailD , the CH compare the actual availability (real availability) of the data AvailR in the cluster with the desired availability AvailD .

- If AvailD $\geq$ AvailR: in this case the availability is satisfied. So the CH does nothing, it waits a while to recheck the constraint.
- If AvailR < AvailD: in this case, the CH starts to create replicas and stored in a good placement as far as availability desired AvailD will be satisfied.

We call the node that stores the replica: the best Target (see § Section 4.2).

4.2 Placement of replicas

The placement of replicas in the system plays an important role. In [17], the author shows that the placement of several replicas of same data in the same node does not improve the availability or fault tolerance. For this reason, it will be useful to store a single replica of the same data in a node. But what are the good candidates to store the data?. In our system, nodes can predict failures. In case of suspecting the failure, the node places all its data in other node to assure the desired availability.

We used a new parameter called ***D**egree of **R**esponsibility* (*DR)* that present the amount of bytes must be replaced elsewhere in case of suspected failure of the node.

$$DR(Nj) = \sum_{i=1}^{k} Size\ (D\,ij) \qquad (3)$$

Where

- N_j: Node;
- k: the number of data in the node N_j;
- D_{ij}: the data i stored in the node j.

A high degree of responsibility indicates that the node has a lot of bytes to place (move) in case of suspected failure. In this case the recovery time of that node will be high and the fault will be accelerated. The recovery time presents the needed time for moving data from the suspected node to other nodes. We suppose that in the beginning the DR is 1 in all the nodes.

The candidate nodes also called best Targets are characterized by:

- Does not already have the data.
- A sufficient available storage space to store data.
- A good *Availability Factor (AF).*

The *AF* is calculated as follows:

$$AF\,(Ni) = \frac{STAB(N_i)}{DR(N_i)} \qquad (4)$$

That is to say that the node with a good *AF* is the node that has good stability *STAB*, and low degree of responsibility *DR*. The placement of replicas according to the strategy of availability factor *AF* guarantees a good distribution of replicas on cluster's nodes, but does not guarantee a good distribution on the network. There may be cases where many replicas of the same data are stored in neighbors which increase the response time for the other nodes. To avoid this problem, we added the last condition:

To increase the distance between identical replicas (replicas of the same data) [6], we proposed to use the parameter of no-similarity $\bar{\beta}$ with the following definition:

$$\bar{\beta}\ (n) = ||(LLDn \cup LDNn\,) - (LLDu \cap LDNn\,)|| \qquad (5)$$

- n: node;
- D_i: data;
- LLD_n: **L**ist **L**ocal of **D**ata in the node *n*;
- LDN_n: **L**ist of **D**ata of all **N**eighbors of the node *n*.

So $\bar{\beta}(n)$ is the size of the list of different data between data list of the node *n* and the data list of its neighbors. For example, if $\bar{\beta}(n) = 0$ then all the data in the node *n* also exist in the 1-neighborhood. Our goal is to distribute the load in the network. In a system where we do not know the frequency of access to the data because:

- The system is new or is in a new period.
- The popularity of data is often changeable.

It more useful to maximize $\bar{\beta}$ of each node without overloading it. For thus reasons we propose the following algorithm that combines between the factor of availability *AF* and the no-similarity parameter $\bar{\beta}$: In the first step of the algorithm 1(See Figure 2), the CH creates the list of candidates nodes that can store replicas of that data, this list is noted *Liste_cand*. This candidates are nodes that do not have the data and have a sufficient storage space. The list *Liste_cand* will be ranked in descending order of $AF * (1/\bar{\beta})$ (see Algorithm 1: Line 01 and 02) that is to say that we support the node of good *AF* and lower $\bar{\beta}$ (see Algorithm 1: Line 01 and 02). Just to avoid non-deterministic case if $\bar{\beta} = 0$, we classify the nodes in the list according to the following parameter:

$$\text{Classification criterion} = AF * (1/(1+\bar{\beta})) \tag{6}$$

As long as the CH has not replicated the data, it verifies for each node in the list *Liste_cand* if adding the replica increases the no-similarity of the node, if so then it stores this data in the node else it tests the next node in the list *Liste_cand* (see Algorithm 1: line 04 to 11). In cases where the CH goes through a whole list *Liste_cand* without replicating the data, so it chooses the best node in terms of *AF* (see Algorithm 1 line 12 and 15).

```
01: Create a list of candidate nodes Liste_cand
02: Sort the list Liste_cand in descending order of $AF * (1/(1+\bar{\beta}))$.
03: Rep ← False
04: While (Liste_cand ≠∅  OU Rep=False)
// $\bar{\beta}'(n)$ non similarity in cases where the node n stores the data
05:     if $\bar{\beta}'(n) > \bar{\beta}(n)$ then // the replica does not exist in 1-neighboring
06:         Store replica of the data in the node n
07:         $\bar{\beta}(n) \leftarrow \bar{\beta}'(n)$
08:         Rep ← True
09:     Else go to the next node in the Liste_cand.
10:     End If
11: End While
12: If Rep = false then
13:     Store the replica in the first node n in Liste_cand
14:     $\bar{\beta}(n) \leftarrow \bar{\beta}'(n)$
15: End If.
```

Fig. 2. Placement algorithm

Figure 3 shows an example of a cluster consisting of five nodes. Assuming that CH wants to store a new replica of the data *M* and that all nodes have the same *AF*. If *Liste_Cand* = {1, 4}, creating a replica of data M in the node 1 does not increase its no-similarity $((\bar{\beta}'(1)) = (\bar{\beta}(1)))$ because this data already exists in its 1-neighborhood. But the addition of this data in the node 4 increases its no-similarity ($\bar{\beta}'(4) = 2 > \bar{\beta}(4) = 1$). So the CH replicates this data in the node 4. In case where the CH add a new data *S* in the cluster, and it has as candidates *Liste_Cand* = {1, 4}, so it choose the node 4 to store a replicas of the data because this node has the minimum $\bar{\beta}$ ($\frac{1}{1+\bar{\beta}(4)} = 1/2 > \frac{1}{1+\bar{\beta}(1)} = 1/6$).

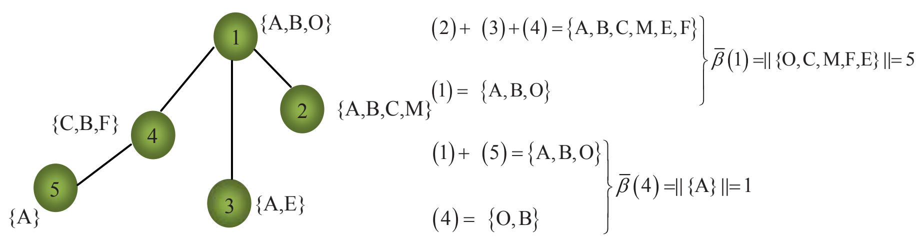
{A,B,O}
1
2
{A,B,C,M}
{C,B,F}
4
5
{A}
3
{A,E}
(2)+ (3)+(4)={A,B,C,M,E,F}
(1)= {A,B,O}
$\overline{\beta}(1)$=‖{O,C,M,F,E}‖=5
(1)+ (5)={A,B,O}
(4)= {O,B}
$\overline{\beta}(4)$=‖{A}‖=1

Fig.3. Example of no-similarity

Each node in the system stores the list of nodes requesting access to its data. If there is a failure suspicion, the node moves its data to other nodes to keep the availability in the cluster. For each data, the node selects from the list of nodes requesting this data the nearest node with a good *AF*. In this way the node minimizes recovery time and keeps the distance between the different data replication.

5. Evaluation

In order to estimate the behavior of our approach called *PD* (*Placement Dynamic*), we used our developed simulator *FTSim* [13]. The first experiment evaluates the response time using different number of nodes in the system. The results are shown in Figure 4. We note that the response time in the proposed model becomes smaller compared to the random approach (the replicas are placed randomly) if the number of nodes increases in the system. In our approach *PD* we ensure a good distribution of replicas on the cluster which minimizes the distance between the node and the target data. The second experiment (see Figure 5) calculates the number of replicas in the system for the two placement approaches (*PD/Random*). We note that the number of replicas of our approach is less than the random approach; despite the availability desired $Avail_D$ is the same in both cases because our approach chooses as the placement of replicas the most stable nodes which minimize the number of replicas necessary to ensure $Avail_D$ (see § Section 4.1).

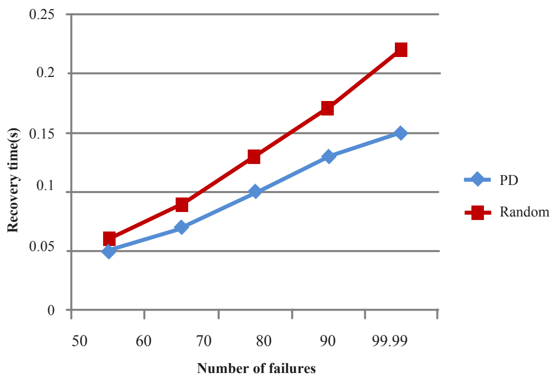

Fig. 4. Number of nodes vs Response time.

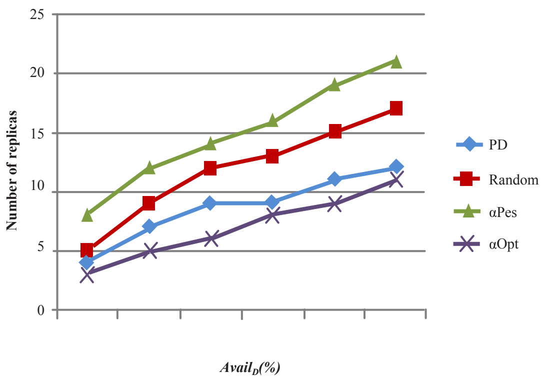

Fig. 5. $Avail_D$ vs Number of replicas.

The recovery time is a very important parameter in fault-tolerant systems. The recovery time of the system is the sum of the recovery time of each node suspect to crash. The results obtained (see Figure 6) shows that this recovery time increases with the increase in the number of failures in the system, but our approach *PD* minimizes this time because the replacement of data is done by the node itself without the intervention CH nodes and also chooses the closest nodes for storing data. Failures can cause loss of data or minimizing its availability. The results of our experiment shown in figure 7 prove that our approach *PD* ensures the availability $Avail_D$ as possible ($Avail_R >= Avail_D$). But the random approach increase the recovery time because the node suspect to crash has no strategy to choose the new placement of its data.

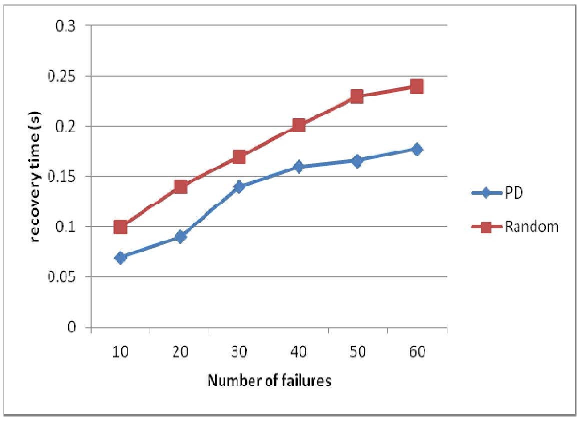

Fig. 6. Number of failures vs Recovery time

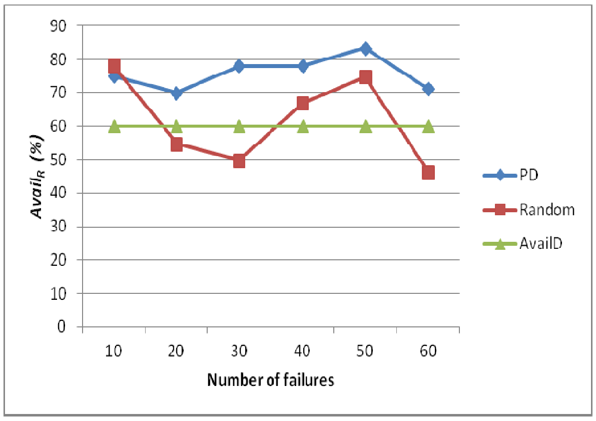

Fig. 7. Number of failures vs $Avail_R$

6. Conclusion & perspectives

In this paper, we proposed a replication strategy based on availability, we also proposed the placement of replicas in the system in an efficient manner that improves system performance without overloading the system nodes. The inconvenient of this approach it is semi-centralized, that is to say that the decision of replication and placement of replicas is achieved by each cluster-head in each cluster. But since the number of nodes in cluster is limited, the constraint of scalability can be met. In the following phase of the research, we will use Globus to study the comportment of our placement model in real grid. We will also extend this work by considering also the task replication and placement in order to ensure a fast and fault tolerant execution of system jobs.

References

[1] E. Caron and Frédéric Desprez. DIET A Scalable Toolbox to Build Network Enabled Servers on the Grid. International Journal of High Performance Computing Applications, 20(3):335–352, 2006.
[2] E.W. Dijstra. Self stabilizing systems in spite of distributed control. Communications of the ACM, 17(11):643–644, 1974.
[3] M. Garmehi and Y. Mansouri. Optimal placement replication on data grid environments. In 10th International Conference on Information Technology (ICIT 2007), pages 190–195, 2007.
[4] G-F. Hughes, J-F. Murray, and K-K. Delgoda. Improved disk drive failure warnings. In IEEE Transaction on reliability, 51(3): 350–357, 2002.

[5] K. Kalpakis, K. Dasgupta, and O.Wolfson. Optimal placement of replicas in trees with read, write, and storage costs. IEEE Trans. Parallel Distributed System, 12(6): 628-637, 2001.
[6] B-J. Ko and D. Rubenstein. Distributed self-stabilizing placement of replicated resources in emerging networks. IEEE/ACM Trans. Netw, 13(3) :476–487, 2005.
[7] H. Lamehamedi. Decentralized data management framework for data grids. PhD thesis, Faculty of Rensselaer Polytechnic Institute, New- York, November 2005.
[8] H. Lamehamedi, B. Szymanski, Z. Shentu, and E. Deelman. Data replication strategies in grid environments. In Proceedings of the Fifth International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'02), pages 378–383, 2002.
[9] M. Larrea, S. Arevalo, and A. Fernandez. Efficient algorithms to implement unreliable failure detectors in partially synchronous system. Distributed computing, 6(9):34–48, 2001.
[10] M. Lei, S. Vrbsky, An on-line replication strategy to increase availability in Data Grids. Future Generation Computer Systems, 24(2): 85-98, 2008.
[11] P. Liu and J-J. Wu. Optimal replica placement strategy for hierarchical data grid systems. In International Symposium on Cluster, Cloud and Grid Computing (CCGRID 2006), pages 417–420, 2006.
[12] E.Marcus,H. Stern, Blueprints for High Availability, Second Edition, Wiley Publishing,2003.
[13] B. Meroufel, G. Belalem : Availability Management in Data Grid, Lecture Notes in Electrical Engineering, 1, Volume 107, IT Convergence and Services, Part 1, Pages 43-53,2011.
[14] K. Ranganathan, A. Iamnitchi Improving data availability through dynamic model-driven replication in large P2P communities. Proceeding CCGRID '02 Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid IEEE Computer Society Washington, DC, USA, 2002.
[15] H. Stockinger, A. Samar, and B. Allcock. File and object replication in data grids. In IEEE Symposium on High Performance and Distributed Computing (HPDC-10), 5(3): 305–314, 2001.
[16] S. Venugopal, R. Buyya, and R. Kotagiri. A taxonomy of data grids for distributed data sharing,management and processing. In ACM computing survey, 38(1): 1–53, 2007.
[17] M. Zhong, K. Shen, J. Seiferas Replication Degree Customization for High Availability, Newsletter Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems, USA. May 2008.