Dynamic Replication Based on Availability and Popularity in the Presence of Failures

Bakhta Meroufel* and Ghalem Belalem*

Abstract—The data grid provides geographically distributed resources for large-scale applications. It generates a large set of data. The replication of this data in several sites of the grid is an effective solution for achieving good performance. In this paper we propose an approach of dynamic replication in a hierarchical grid that takes into account crash failures in the system. The replication decision is taken based on two parameters: the availability and popularity of the data. The administrator requires a minimum rate of availability for each piece of data according to its access history in previous periods, but this availability may increase if the demand is high on this data. We also proposed a strategy to keep the desired availability respected even in case of a failure or rarity (no-popularity) of the data. The simulation results show the effectiveness of our replication strategy in terms of response time, the unavailability of requests, and availability

Keywords—Data Grid, Dynamic Replication, Availability, Failures, Best Client, Best Responsible, Data Management

1. INTRODUCTION

Grid computing is an important technique to manage computing resources that are distributed in different geographical locations and appear as a single entity [14]. A kind of powerful virtual machine would be so formed by the sharing of many machines from different administrative organizations. The current level of technology and the great interest of researchers in this methodology make the data grid an interesting solution to the problems of computing and storage intensive [5].

The data grid is a type of grid computing, where all of the components (virtual organizations) require access to data from other components [20]. The main task in this type of grid is to manage the large amount of data in a transparent mining to ensure good availability with easy access and the sharing of such data [3, 5]. Replication is a technique used by the data manager to achieve the above objectives. In other words a data grid needs to create and manage replicas of data. Replication can redirect user requests to other sites that have the data. This minimizes the response time and enables load balancing. The main advantages of replication are [9]:

- Improved availability: in case of a failure of a node, the system can replicate the data from this site, which also improves the availability;
- Improved performance: since the data is replicated among several nodes, the user can obtain

Manuscript received June 28, 2011; first revision February 27, 2012; accepted April 12, 2012.
Corresponding Author: Ghalem BELALEM
* Dept. of Computer Science, Faculty of Sciences, University of Oran (Es Senia), Algeria (bakhtasba@gmail.com, ghalem1dz@gmail.com)

data from the node nearest the node or that is the best in terms of workload.

In this paper, the proposed approach of dynamic replication is based on two main parameters of data management: the availability and popularity. The administrator must specify a certain percentage of availability for the data in the system and the number of replicas that satisfy this percentage will be considered as the minimum degree of replication, but this percentage of availability may increase depending on the popularity of the data. This approach provides also a strategy to assure the availability even in the presence of failures (faults).

The remainder of this paper is organized as follows: Section 2 introduces some related work on the different techniques of replication. Section 3 presents our approach of dynamic replication based on availability and popularity in the presence of failures. Section 4 shows some of the results of our simulations and Section 5 presents a summary of this work and a conclusion.

2. Related Work on Replication

Replication is a technique that is widely used in a distributed environment [1], especially in a data grid where literature has several works that adopt this strategy to improve the performance of their systems. To achieve the desired end of the replication process, replication should be conducted in accordance with certain strategies that organize and manage data. These strategies are made based on several factors such as the demand for data, network conditions, and the cost of transfer [17]. The replication strategies can be classified according to whether the strategy is static or dynamic. If resource conditions are relatively stable in the data grid over a long period, the static replication strategy can be applied as it creates and manages the replicas manually [19] (i.e., a replica remains in the system until it is deleted by users or its lifetime has expired [20]). This means that the static replication will not adapt to changing the data access, which is considered an inherent characteristic of the architecture of a data grid when the client changes the access, instead of benefiting the replication strategy. This is unnecessary and will create a burden on the system. Instead, the dynamic replication automatically creates new replicas depending on the situation, in other words, the dynamic replication takes into account changes in the grid environment [3]. Usually, dynamic replication is triggered when the popularity of a file exceeds a certain threshold [13]. The authors of [15, 18] use dynamic replication to conserve bandwidth in a hierarchical environment. The work in [9] studies the replication dynamics in different topologies such as ring, tree, and hybrid. The work in [2] presents a replication based on the economic concept to specify the site of the new replica; a replica of data X is created when the number of requests on data X exceeds the replication threshold. With the same concept the authors of [16] propose an approach of replication based on a compromise between the cost and future gain of the new replication. Works [6, 11] propose a replication based on data popularity. In the article [8] replication is based on the availability of the data; that is to say it replicates a given data to ensure its availability in case of failure, but this type of replication is static.

3. The Proposed Approach

Dynamic replication is an effective strategy that fits with the grid environment. It can be used for many reasons such as: minimization of response time, reduced cost of communication, pres-

ervation of bandwidth, assurance of data availability, fault tolerance, etc. [16]. Although this type of replication ensures good availability for the requested data it can cause a loss of data:

- The loss of data due to rarity (no-popularity): if there is a change in the popularity of a piece of data (data that is not popular at the time t becomes popular at the instant t'), this data will be unavailable or will be very rare. This situation increases the response time and increases the server load.
- The loss of data due to failures of nodes: the failure of a node that contains rare data can cause the unavailability of this data or even a loss of it (data).

This loss of data causes problems especially for systems that have data that have a dynamic popularity (popular data becomes unpopular and vice-versa). To solve the problem of data loss in both cases (unpopularity or failure), we proposed, with some improvements, an approach of dynamic replication that combines replication that is based on the availability of the work proposed in [11, 15] and replication based on the popularity of the work given in [8, 16]. In this approach the minimum number of replicas for each piece of data is the number of replicas that meets the availability desired by the administrator. We call the replicas that meet the desired availability primary copies. A primary copy is a replica that cannot be deleted from the system even if it is unpopular because this deletion can decrease the desired availability. The number of replicas can increase depending on the number of petitions requesting this information (popularity). The replicas created to ensure the popularity are ordinary replicas. In case of failure, the node is not required to replicate all its data elsewhere, but only the primary replicas. Our replication approach is articulated on a semi-centralized hierarchical topology.

3.1 Description of the topology used

In our work, we used a hierarchical topology. This use is motivated by the following points:

- Minimizing the time of receipt of each message as the management and communication are made in each cluster.
- Minimize the number of messages exchanged through the tree architecture.

Several systems such as Internet and DIET [8, 15] adopted this topology. Fig. 1 shows an example of a hierarchical topology consisting of four clusters.

The root in the topology is used to bind together the different clusters, so the longer distance between two clusters consists of two jumps. This root contains a list of replicas that exists in the system. That is to say on each cluster, if a request comes in and it can not be satisfied within the cluster, the Cluster-Head (CH) leads this request to the root which in turn directs it to a cluster that has a copy of this data. In cases where there are several clusters that have the data, the root chooses the best cluster in terms of response time and communication cost. The cluster-head, which represents members of its cluster, has a routing table that manages the nodes within the cluster. It also contains replicas like the ordinary nodes. The other nodes are storage elements; they contain one replica of one or more data.

In the system, nodes have predictive behavior. We used the technique SMART to predict failures in hard drives where data is stored [7]. Predicting a failure triggers the replication service

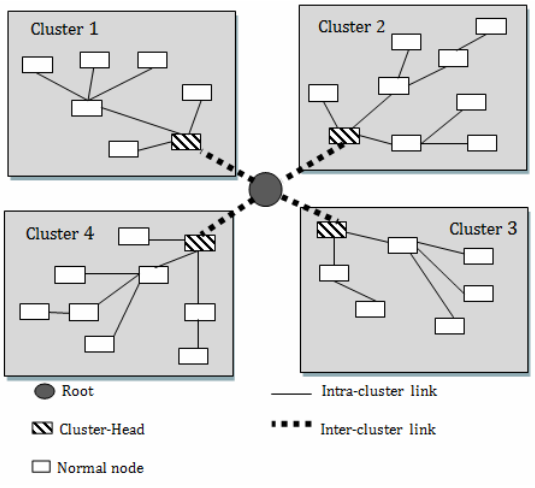

Fig. 1. The used topology

(See § Section 3.2). To avoid false suspicions (SMART reports a failure but the node is correct), especially in the case of expensive resources, we also used a service of fault detection that is based on the life messages [10]. If a failure is detected, the self-stabilization will be triggered to keep the topology connected. In other words, instead of self-stabilizing the system after the prediction, as has been proposed in several other works [4], self-stabilization will start after the failure of the node. This strategy avoids the stabilization phase or eliminating resources because of false suspects. The self-stabilization is a uniform strategy that it is applied by all the nodes of the topology including the root. It is based on the following idea: in the case of a failure of node *u,* one of its sons (sons of node *u*) replaces it (son inherited) and it adopts the rest the sons. The inherited son of the node *u,* which is noted I_u, is the most stable son among the sons of node *u.* The stability of a node is the probability of its availability in the system. The choice of I_u is affected by each change in the list of the sons of *u*, which is caused by the dynamicity of the grid. This paper is a part of our work and it focuses only on the data and replica management in the grid.

3.2 Management of availability and popularity

The definition of availability is the measure of the frequency or duration in which a service or a system is available to the user. To calculate the availability of data we assume the following case: each node (the component that contains the replica) has a certain probability of stability. The availability of a replica is the stability of the node where it is stored. So if *p* is the probability of the availability of data noted *M* in a node, then the probability of its available *No_Avail(M)* is:

$$No_{Avail(M)} = (1 - p) \tag{1}$$

and if α is the number of replicas of data *M*, then the availability *Avail (M)* can be calculated as follows [16]:

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

From this formula we can calculate the number of replicas of α that are needed to have some probability of availability.

At first, the administrator requires a certain probability of availability. The desired availability for each piece of data is specified by some parameters such as the access history in prior periods and the importance of the data. The replicas that assure the availability are primary copies. The primary copy is an ordinary replica, but it cannot be deleted from the system because this deletion decreases the desired availability.

As soon as the number of primary replicas is known (using the formula 2), the replication manager with its centralized management at the Cluster-Head starts creating primary replicas. We associate with each replica of *i* a Boolean variable of *D(i)*.

- *D(i)* = False: indicates that the replica *i* is primary and it is created by the Cluster_Head to meet availability. The node is not allowed to delete this replica even if it is not requested (unpopular replica). Nodes that have this type of replica are the most stable in the system. In case of failure suspicion, the node that contains this data will replicate it in the best responsible node.

The best responsible node is the node that has the smallest degree of responsibility and a good stability. It will always be the destination of the primary replica created by the CH or will be replicated by another node in case of failure suspicion. The degree of responsibility is the sum of the sizes of primary replicas in the node in MB (Mega Byte) or GB (Gega Byte). But what is missing in this strategy is that the data does not have the same popularity. The popularity of the data *M* is calculated by the following formula:

$$Pop(M) = \frac{Number\_of\_requests\_demanding\_M}{Number\_of\_all\_requests} \tag{3}$$

So it is not useful to assure the same degree of availability for all data. For this reason we add another type of replication based on popularity. This replication is a non-centralized replication (unlike the case of replication based on availability) because it is triggered by the local replication manager of each node.

Each node has a history table that stores the number of accesses to its data. If the number of total access to data (primary or not) exceeds a certain threshold, the node replicates the data in the best client. The best client is the node that has the greatest number of access to a given piece of data. The replica *j* created in this case is an ordinary copy (not primary) that has *D(j)* = True. Let us consider the data shown in Fig. 2 as an example of access history.

Node 1 has two pieces of data, ***X*** and ***Y***, each of these data has its own history table (Access-history of X, Access-history of Y) stored also in node 1. The data X in node 1 was accessed 21 times by the nodes 2, 3, and 4 and the best client is node 3. But data Y was accessed only 15 times by the nodes 3, and 4 (according to the access-history of Y) and the best client is node 4.

- *D(j)*=True: indicates that the replica *j* is ordinary (not primary) and it is created by a node to

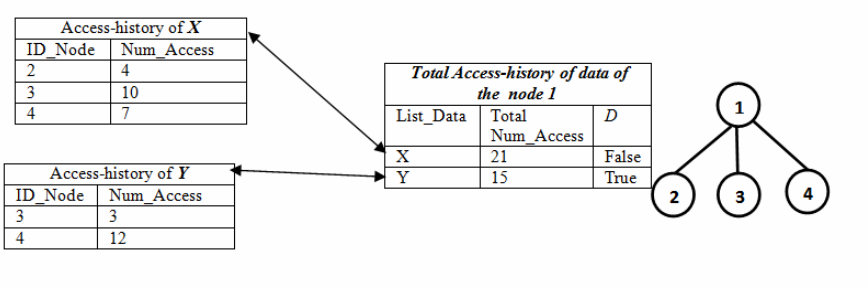

Fig. 2. Example of the history access

meet the demands on the data. The node that contains the replica *j* can remove it to store more popular data in a local disk. In case of failure suspicion, it is not necessary to replicate this data elsewhere.

In Fig. 2, if the Boolean variable *D* of the data *X* in the node 1 is noted *D(X)*=False, then data *X* is a primary copy and it was created to satisfy the desired availability. In this case node 1 is not allowed to delete the data *X* to store other data *Z* even if the data *Z* is more popular then *X*. In case of data *Y*, *D(Y)* = True, so *Y* is an ordinary replica and it was created to satisfy its popularity in the users queries (popularity of *Y*). In this case node 1 can delete this replica to store more important data. In case of failure suspicion, node 1 replicates only data *X* in the best responsible node as the primary copy.

The threshold of replication plays an important role in the dynamic replication. It controls the number of replicas that can be had in the system to meet the demands on the data concerned. As the ratio between the threshold of replication and the total number of requests increases, the maximum number of replicas of the data involved gets bigger. This report is called $\boldsymbol{\beta}$ and is expressed by formula 4.

$$\beta = \frac{replication\_threshold}{Total\_Number\_of\_requests} \tag{4}$$

The diagram in Fig. 3 shows the steps of replication in our approach.

The first step is comparing the number of replicas that really exists in the system ($\boldsymbol{e}$) with the number of replicas that meets the desired availability ($\boldsymbol{\alpha}$).

- If $\boldsymbol{e} < \boldsymbol{\alpha}$: then the system starts the replication based on availability. It creates primary copies and stores them in the best responsible node.
- If $\boldsymbol{e} \geq \boldsymbol{\alpha}$: then the system starts the replication based on popularity and it creates ordinary copies and stores them in the best clients.

The discontinuous lines in the diagram indicate that the system must wait some time before executing the next step (For example: between step 6 and step 1 in Fig. 3).

Returning to Fig. 2: if $\alpha(X) = 2$ and $e(X) = 1$ then node 1 replicates *X* as a primary replica in the best responsible node (node 4). But in the case where $\alpha(X) \leq e(X)$, the node starts to satisfy the popularity of the data *X*, so if the threshold of replication for data *X* is 20, then this data will be

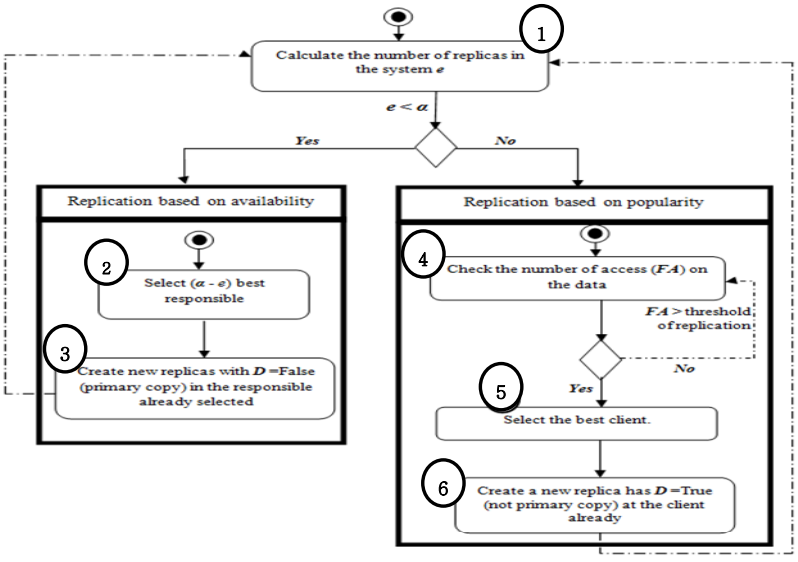

Fig. 3. Replication steps in our approach

replicated as an ordinary copy in the best client, which is node 3.

If a node receives a request to store the new replica home, it will react according to the type of replication such as replication based on availability (best responsible node) or replication based on popularity (best client node). Fig. 4 presents the algorithm used by the local replication manager of each node to store a new replica of data ***M*** in both cases.

The inputs of the program are the type of data *M* that we want to store and its size. *Memory-pro* is a procedure that calculates the available space in the storage space of the node in question. If a node receives a request for the storage of data *M*, it first checks whether it has enough space to store this data (ligne01). If there is no space (from line 02 to 32), the node begins to build the list *ListSup* (03 to 22). *ListSup* is a list of data that can be removed to store data *M*:

- If the node that wants to store *M* is the best client node (*M* is not a primary copy: lines 05 to 14), the list *ListSup* contains data that has *D* = True and an access frequency *FA* that is strictly less than that of *M*. The *Compt* is the sum of the frequency of access of data in the list *ListSup* (line 09), this is in order to avoid deleting data that has a lower *FA* than *M*, but the sum of their *FA* exceeds the frequency of the access of *M*. The list *SizeList* is the sum of the sizes of data in the list *ListSup* (line 19 to 18).
- If the node that wants to store *M* is the best responsible node (*M* is primary copy: lines 15 to 22), the list *ListSup* contains only the data with *D* = True (not the primary copy). In other words to satisfy the availability we can delete some ordinary replicas (not primary replicas) regardless of their popularity (replicas popularity).

After the construction of the list *ListSup*, the node check if it is possible to provide sufficient space to store *M* before starting to delete data from the list *ListSup* and from the node (line 23). In line 24, the program sorts the list *ListSup* in the ascending order frequency of access. It then

```
Algorithm Storage Data M
01: If (Memory_disp ≥ Size(M)) Then Store M
02: Else
03:         Compt ← 0
04:       ListSup ← ∅ ; SizeList ← ∅
05: If (D(M) = True) Then                                   // ****** The node is the Best client
06:         For all Data M' in the node do
07:                 If (D(M') = True) Then
08                         If (FA(M') < FA(M)) and (Compt < FA(M)) Then
09:                                                                 ListSup ← ListSup +
M'
10:                                                                 Compt ← Compt +
FA(M')
11:                                                                 SizeList ← SizeList +
Size(M')
12:                         End If
13:                 End If
14:           End For
15: Else
16:     If (D(M) = True) Then                                 // ******* The node is the Best responsible
17:             For all Data M' in the node do
18:                     If (D(M') = True) Then
19:                                                 ListSup ← ListSup + M'
20:                                                 SizeList ← SizeList + Size(M')
21:                     End If
22:             End For
23:     End If
24:     If (SizeList ≥ Size(M)) Then                          // ****** Release space to store the data M
25:                     Store ListSup in ascending order of FA
26:                     While (Memory_disp < Size(M)) or (ListSup ≠∅) do
27:                             M" ← ListSup(1)
28:                             ListSup← ListSup - ListSup(1)
29:                 End While
30:               Store    M
31:     Else
32:     Refuse to store M
33:     End If
34:EndIf
```

Fig. 4. Algorithm for the storage of data M in the best client node and best responsible node

(the node) starts to remove the first items (the data) in the list and from the node itself until the sufficient space is provided for the data *M* (line 25 to 29).

3.3 Positioning of our approach compared to related works

To evaluate our approach theoretically, we have compared it with other replication approaches proposed in the literature. We focus the comparaison on six options:

- Popularity: this option specifies whether the replication approach takes into account the number of access to data (number of queries asking for the data) to create a new replica or not.
- Availability: this option specifies whether the approach ensures a certain degree of availability of various data whatever their rarity or popularity in the system is or is not.

- Placement of replicas: shows the candidate nodes to store the new replicas.
- Replication type: there are two types of replication. One type is static replication where the number of replicas of each data is specified at the beginning and it does not change. The second type of replication is the dynamic replication where the number of replicas of each data item may change depending on some parameters.
- The type of management: this is centralized if a special element in the system is responsible for managing the replication, and it is distributed if the management of the replication can be applied based on local information in the node itself.
- Fault tolerance: if the replication approach takes into account a failure in the system and tolerates it, we say that this strategy is "fault tolerant."

Table 1 presents the results of the comparative study between our approach and other approaches cited in the paper. The sign (+) indicates that the option is met. The sign (-) indicates that the replication approach does not consider the concerned option. The PNOR in [6] is the ratio of the number of requests to the shortage of the requested space to accommodate the replica.

So for example, if we compare between our approach and the work [16] the strategy proposed in [16] does not take into consideration the popularity and fault tolerance, but it provides a certain degree of availability for the data system. Replication used in [16] is static and distributed because it can be executed by any peer of the P2P system (*peer to peer system*). This replication stores the replicas in nodes at a reasonable cost transfer.

Our approach takes into account the majority of options such as popularity, availability, and fault tolerance. The replication used a combination between centralized static replication to ensure availability (the number of primary copies is fixed from the beginning by the Cluster-Head) and a distributed dynamic replication to ensure the popularity (ability to remove/add non-primary copies by the node). We consider that the combination of static and dynamic replication is a hybrid replication because of the transparency of the approach. Both types of destinations for new replicas allow some load balancing for nodes in the system. Each type of destination (best client/responsible) is responsible for storing a certain type of replica (copies primary/non-primary).

Table 1. The results of the comparative study

	[8]	[11]	[18]	[16]	[6]	Our approach
Popularity	-	+	+	-	+	+
Availability	+	+	-	+	-	+
Replica placement	Nodes belonging to the same class of stability	Future best client	Best client	Customer with a reasonable transfer cost	Each node requesting the data if PNOR* is sufficient	Best client + Best responsible
Replication type	Static	Dynamic	Dynamic	static	Dynamic	Static+Dynamic = Hybrid
Type of management	Central (super-peer) (P2P)	Central (manager of the site)	Central (manager of the site)	Distributed (each peer in P2P)	Central (Root server)	Central(CH) + Distributed (Node)= Semi-centralized
Fault tolerance	-	-	-	-	-	+

4. EXPERIMENTAL RESULTS

To validate the proposed approach we have developed a simulator called *FTSim* (***F**ault **T**olerance **Sim**ulator*) using Java language [12]. With *FTSim*, users can easily create a Data Grid with desired parameters such as: the number of clusters, number of nodes in each cluster, number of data and their distribution in nodes, bandwidth intra and inter cluster, etc (Table 2).

After the creation of the grid, the users specify the simulation parameters, which include the number of jobs and their distributions, the threshold of replication, desired availability, etc.

This simulator has three main objectives:

- Measure the performance of our replication approach.
- Study the impact of certain parameters on the proposed approach.
- Compare this approach (our approach) with the classical approach of dynamic replication proposed in [10], which we call, “Dynamic."

So to study our approach using the simulator, we proposed a simulation environment that has the following parameters: the number of nodes varies between 100 and 500 nodes (depending on the objective of the simulation) with a size of 20 MB and an 80% stability, the bandwidth is equal to 100 Mbit/sec, and the system contains several pieces of data with a size 5 MB. The early availability of the data that is required by the administrator is 90%, which is to say the system must have two replicas to meet the desired availability (see Table 2). The number of requests is between 100 and 500. The frequency of failures presents the ratio between the number of failures and the total number of nodes in the system. This ratio varies between 0% (no failures) and 50 % (half of the nodes in the system have failed).

We studied the following three parameters: the response time, the unavailability of requests (*System file messing rate*), and the availability and the time needed to assure the service of replication.

Table 2. Parameters of simulations

Parameter	Value
Number of nodes	100 to 500
Number of clusters	2 to 30
Size of nodes	20MB
Stability of nodes	80%
Bandwidth inter-cluster	100 to 200 MB/second
Bandwidth intra-cluster	10 to 50 MB/ second
Availability desired	90% (α=2)
Number of requests	100 to 500
Number of data	10 to 120
Frequence of failures	0% to 50%

4.1 Response time

In this experimentation, we studied the impact of number of nodes on the response time. We used the same parameters cited in Table 1, but we fixed the number of requests as 300 and the number of failures is 0%. In the first scenario we studied the impact of the number of nodes re-

sponse time so we varied this number if the nodes are between 100 and 500.

The response time in our approach is better than the response time in the case of classic dynamic replication whatever the number of nodes. This improvement can be estimated at 18.6%. The result is logical because:

- Increasing the number of nodes in the system increases the response time in both types of replication (our replication and the classical dynamic replication) because the distance between the data and the user becomes longer.
- Classical dynamic replication guarantees the existence of the data in the system if it is popular or else this data will be lost or at least rare (because of unpopularity). In the event of a change in the frequency of access to data (unpopular data becomes popular), some nodes may overload in access before they begin to create new replicas, which increases the time response. However, our approach guarantees the existence of data regardless of its popularity, which minimizes the response time.

To study the impact of the frequency of failures (the ratio between the number of failed nodes and the total nodes in the system) on the response time, we used the same parameters of simulation proposed for the first experimentation, but we fixed the number of nodes to 300 and we varied the frequency of failures from 0% (no node will fail) to 50% (50% of the nodes in the system will fail). The results are presented in Fig. 6. The number of failures increases the response time because there is data (files) lost, but our approach gives good results compared to that of a typical replication because of the following reasons:

- In classical dynamic replication: the failure of a node that contains popular data shall not cause the loss of this data but it minimizes its availability. But in the case of rare data, the failure can cause a loss of this data.
- In our approach, whatever the number of failures or rarity of the data, the system always serves the desired availability, which improves the response time. The strategy of the best responsible node offers a good distribution of primary copies.
- The average gain in our approach is 19%.

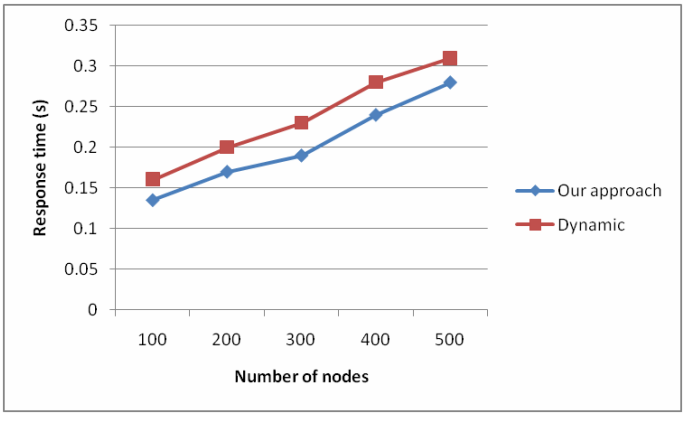

Fig. 5. Impact of the number of nodes in both approaches

4.2 SFMR (System File Missing Rate)

The second metric in our simulations is *SFMR,* which we also call, "Unavailability of requests". The *SFMR* is the ratio between the number of unavailable data and the number of data requested by queries. This parameter is proposed in work [11] where the authors proved that the minimization of this parameter indicates a good availability for the data system. According to [11] *SFMR* is calculated by the following function:

$$SFMR = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m}(1 - P_j)}{\sum_{i=1}^{n} m_i} \tag{5}$$

Where n is the total number of jobs, each job requests to access m data. P_j is the availability of the data. In our case the job is a request and each request requires access to a single given time (m = 1) and then the above formula is formula 6:

$$SFMR = \frac{\sum_{i=1}^{n}(1 - P_j)}{n} \tag{6}$$

Where P_j is the availability of the data j requested by the query i.

We studied the impact of the number of requests on the unavailability of requests, so we fixed the number of nodes to 300 and the number of failures to 0%. However, the number of requests increased from 100 to 500. The result of this scenario is schematized in Fig. 6.

The results in Fig. 7 show that the SFMR decreases if the number of requests increases in the strategy of replication because if the number of requests augments, the data concerned will be replicated and its availability will increase. We also point out that our dynamic replication decreases the SFMR parameter with a gain of 16% of the request's availability and this consequently improves the availability of data in the system, especially in the case where the fre-

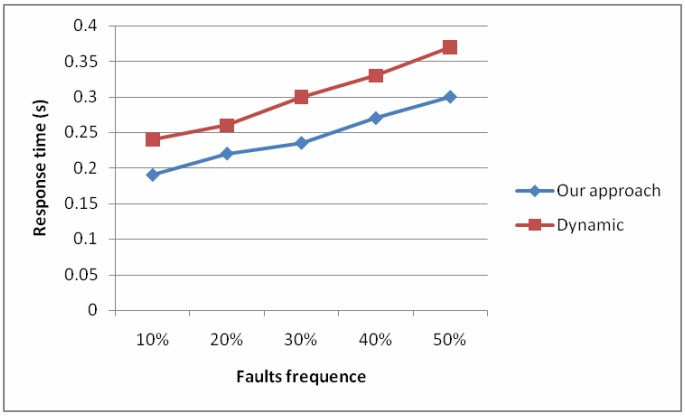

Fig. 6. Impact of the number failures in both approaches

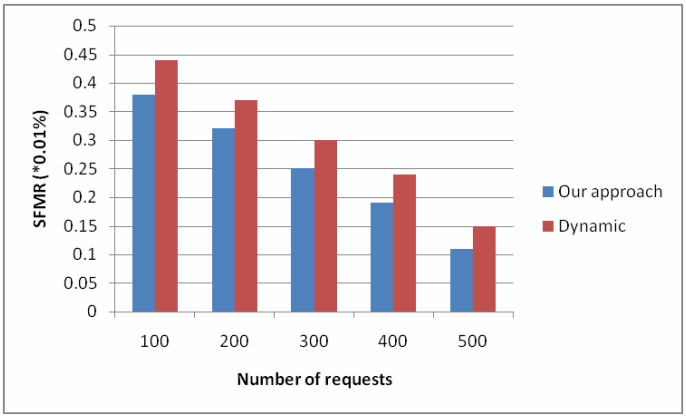

Fig. 7. Impact of the number of requests in SFMR

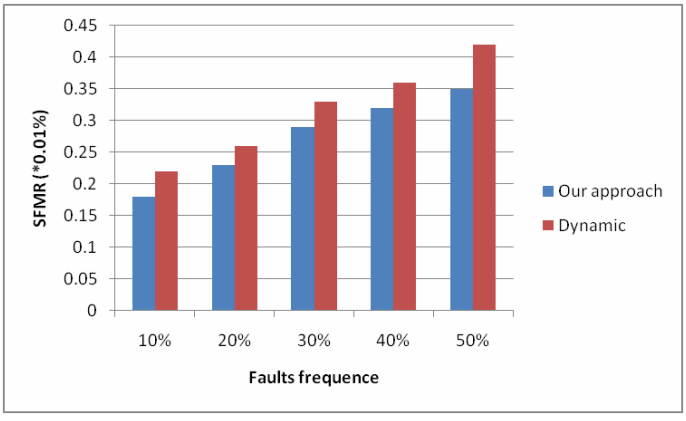

Fig. 8. Impact of the number of failures in SFMR

quency of access to the requested data is changed.

The number of failures also infects the unavailability of requests. The simulation of a system confirms this result with the number of requests at 300 and where the number of nodes is 300. But the frequency of failures was between 0% and 50% (see Fig. 8). Queries laced in a system that uses classical dynamic replication are less satisfied compared to queries made in the system that adopts our approach, which ensures the availability of data. The average gain is estimated to be 14%.

4.3 Availability and time needed to ensure the service

In these experimentations, system availability is the average availability of all data. We measured the availability of our approach and that of the classical dynamic replication. The number of nodes is 400 and the number of requests is 400, but the frequency of failures varies between 0 and 50%. The results illustrated in Fig. 9 show that the average availability of data in a system that uses our approach is better than the availability assured by the approach classical dynamic replication. Our approach provides at least the desired availability, but it increases the availabil-

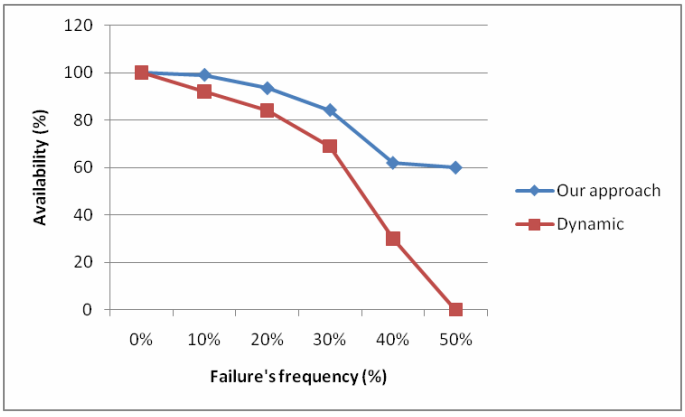

Fig. 9. Impact of failure frequency on availability

ity by the popularity of the data. In the case of classical dynamic replication, data are available only if it is (the data) requested by the system (users queries), otherwise it may be lost because of failures or unpopularity. The average gain in our approach is 32%, which means 112 supplementary days of work in a year.

Although the number of clusters is an important factor in assessing an approach, in our case this factor does not affect the performance of our replication approach because of two reasons:

- The nature of the topology: the topology consists of a root that connects the clusters, which are managed by their cluster-heads. In this case the longest path between two different clusters is two jumps.
- Intra-cluster management: in our approach each cluster-head is responsible for meeting the availability and popularity of its data in its cluster without taking into account the other clusters.

The only problem that may be posed by the number of clusters is the overloading of the root, because it is responsible for managing the inter-cluster communication and routing.

We also tested the impact of tree height in each cluster in the time needed by the system to ensure the availability and popularity data in our approach (this amount of time is different from the response time). In this experimentation the number of nodes is 100 in each cluster, the number of requests is 400, and the number of data is 100, but the levels of the tree varies between 2 and 12. According to the results obtained (see Fig. 10), the time to ensure data availability increases if the tree height increases (we are considering that in each cluster the cluster-head are at level 1 and the leaves are at level $n / n \in N$). This increase is due to the centralized management of availability because the cluster-head is responsible for ensuring, controlling, and monitoring the availability of the data. In this case if the distance between the node and the cluster-head increases (the height of the tree) the time to ensure proper management increases. But management of replication-based popularity is independent of the height of the tree because it is decentralized (managed by each node).

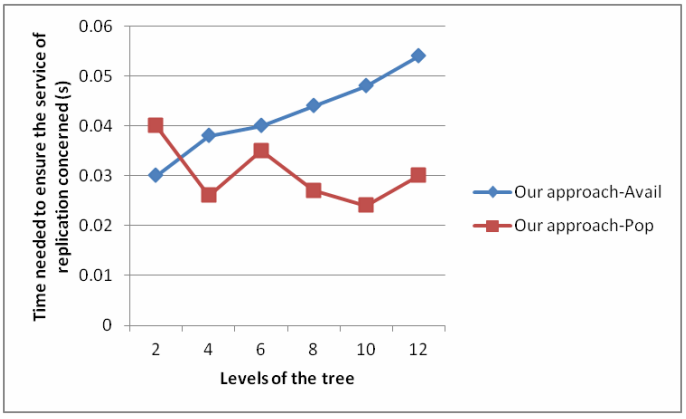

Fig. 10. Impact of the levels of the tree on the time of the service

5. CONCLUSION

In this report we presented our approach of dynamic replication, which is based on popularity and availability. The system must guarantee a minimum level of data availability and it improves the availability depending on the demand for this data. We compared this approach with a classical dynamic replication. Experimental results show that the proposed approach is better than the dynamic replication in terms of response time, the availability of requests, and the availability of data. Looking ahead, we believe that this approach has improved so that it tolerates failures. From long-term perspective, we propose to extend the approach proposed by a smart appearance in decision-making and for the integration of agents within each cluster in order to implement a cooperative distributed intelligence among agents.

REFERENCES

[1] W. H. Bell, D. G. Cameron, L. Capozza, A. P. Millar, K. Stockinger, and F. Zini, "*Simulation of Dynamic Grid Replication Strategies in OptorSim,*" Int. Journal of High Performance Computing Applications, Vol.17, 2003, pp.46-57.

[2] W.H. Bell, D.G. Cameron and R.C. Schiaffino "*Evaluation of an Economy-Based File Replication Strategy for a Data Grid".* In Proceedings of the 3rd I*EEE/ACM International Symposium on Cluster Computing and the Grid*, 2003 (CCGrid 2003) (Tokyo, Japan). IEEE CS Press, Los Alamitos, CA, USA.

[3] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke, *The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets, Network and Computer Application*, Journal of Network and Computer Applications, 23(3):187-200, July 2000,

[4] E.W. Dijstra. *Self stabilizing systems in spite of distributed control.* Communications of the ACM, 17(11):643-644, 1974.

[5] I. Foster. *The grid: A new infrastructure for 21st century science*. Physics Today, Vol.55, No.2, 2002, pp.42-47.

[6] F. Hanandeh, M. Khazaaleh, H. Ibrahim and R. Latip: *CFS: a new dynamic replication strategy for data grids*. Int. Arab J. Inf. Technol. 9(1): 94-99, 2012.

[7] 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.

[8] T.Huu, M-T.Segarra, J-M.Gilliot*Un système adaptatif de placement de données,* In CFSE'6 : Conference Français sur les Systemes d'Exploitation, Fribourg, Suisse, du 11 au 13 février 2008

[9] H. Lamhamedi, B. Szymanski, Z. Shentu, E. Deelman. Data replication strategies in grid *environments.* In *Proceedings of the 5th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'02)*. IEEE CS Press, Los Alamitos, CA, USA.

[10] M. Larrea, S. Arevalo, and A. Fernandez. *Efficient algorithms to implement unreliable failure detectors in partially synchronous system*. Distributed computing, Lecture Notes in Computer Science, Vol.1693, 1999.

[11] 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.

[12] B. Meroufel, and G. Belalem; *Availability Management in Data Grid.* Lecture Notes in Electrical Engineering, 1, Volume 107, IT Convergence and Services, Part 1, 2011, pp.43-53.

[13] K. Mohammed 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 November, 2008.

[14] B. Naddaf and J. Habibi, *Bidding Strategically for Scheduling in Grid Systems*. Journal of Information Processing Systems JIPS,5(2): 87-96, 2009.

[15] S.M. Park, J.H. Kim, Y.B. Ko, and W.S. Yoon: *Dynamic Data Grid Replication Strategy based on Internet Hierarchy.* Second International Workshop on Grid and Cooperative computing (GCC'2003) in Shanghai, China, December, 2003.

[16] K. Ranganathan, A. Iamnitchi and I. Foster. 2002. *Improving data availability through dynamic model-driven replication in large peer-to-peer communities.* In Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02) (Berlin, Germany). IEEE CS Press, 21-24 may 2002, Los Alamitos, CA, USA, pp.376-381.

[17] K. Ranganathan and I. Foster, *Design and Evaluation of Dynamic Replication Strategies for a High Performance Data Grid,* International Conference on Computing in High Energy and Nuclear Physics, Beijing, September, 2001.

[18] K. Sashi, Antony Selvadoss Thanamani: *Dynamic replication in a data grid using a Modified BHR Region Based Algorithm.*Future Generation Computer Systems. 27(2): 202-210, 2011.

[19] M. Tang, B. S. Lee, X. Tang, and C. K. Yeo, *The impact of data replication on job scheduling performance in the Data Grid,* Future Generation Computer Systems, 22(3): 254-268, 2006.

[20] S. Venugopal, R. Buyya, and K. Ramamohanarao, *A Taxonomy of Data Grids for Distributed Data Sharing, Management, and Processing,* ACM Computing Surveys. 38(1): 1-53,2006.

Bakhta Meroufel

She is a PhD candidate in the Department of Computer Science in the Faculty of Sciences at the University of Oran in Algeria. She received her M.S. degree in 2011 from the University of Oran. Her research interests are: distributed system, grid computing, cloud computing, fault tolerance, replication strategies, mobile environment and multi-agents systems.

Ghalem Belalem

Ghalem Belalem graduated from the Department of Computer Science in the Faculty of Sciences at the University of Oran in Algeria, where he received his PhD degree in Computer Science in 2007. He is now a research fellow for the management of replicas in data replicas in data grids. His current research interests are distributed systems, grid computing, cloud computing, data grid placement of replicas, and consistency in large scale systems and mobile environments.