World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:7, No:1, 2013

Increasing Replica Consistency Performances with Load Balancing Strategy in Data Grid Systems

Sarra Senhadji, Amar Kateb, Hafida Belbachir

Abstract**—**Data replication in data grid systems is one of the important solutions that improve availability, scalability, and fault tolerance. However, this technique can also bring some involved issues such as maintaining replica consistency. Moreover, as grid environment are very dynamic some nodes can be more uploaded than the others to become eventually a bottleneck. The main idea of our work is to propose a complementary solution between replica consistency maintenance and dynamic load balancing strategy to improve access performances under a simulated grid environment.

Keywords**—**Consistency, replication, data grid, load balancing.

I. Introduction

GRIDS are considered as one of the promiscuous environment of scientific applications which need an important set of storage and computing resources. In this kind of dynamic and large scale environment, the management of massive data is still being one of the important scientific and industrial research areas. Data replication is one of these important research domains.

The replication technique improves data availability, scalability and fault tolerance, however, the update of replica by any grid user might bring a critical problem of maintaining consistency between the other replicas of the grid. Moreover, according to the frequency access to the different grid nodes, some nodes are more uploaded then the others. This irregular evolution provides an unbalance and many resources can be for example, downloaded and consequently unexploited.

Thus, our main idea is to bring a complementary solution between replica consistency and load balancing. The objective of this work is to improve the consistency performances by taking care of the load of grid nodes. Our contribution consist to increase the replica consistency performances through a dynamic load balancing strategy of the grid nodes in order to guarantee the data availability and reduce the time access with a minimal communication coast.

In the next section we give an overview of some existing replica consistency works. Then we define our approach with the adopted dynamic load balancing strategy. The experimentations of our approach will be discussed in experimentations section. Finally, we close this paper with some conclusions.

Sarra Senhadji, LSSD Laboratory, University of science and Technology, Oran, Algeria (phone: +213-554-985445; e-mail: senhadji.sarah@gmail.com).
Amar Kateb, University of Science and Technology, Oran, Algeria (e-mail: kateb_amar@yahoo.fr).
Hafida Belbachir, Professor at the University of Science and Technology, Oran, Algeria, LSSD Laboratory (e-mail: h_belbach@yahoo.fr).

II. Related Works

Many works have been done on the replica consistency domain in distributed systems, such as cluster, peer to peer and grid. We find many consistency models in the literature [12]: Strong models, Weak models [13], [15]. Strong consistency models keep data consistent among all replicas simultaneously, which requires more resources and expensive protocols than weak models. Contrary to strong consistency, weak consistency can tolerate inconsistencies among replicas for a while to improve access performances.

The authors of [4] addressed the problem of shared data in data grid systems. The consistency of replicated data is introduced by relaxed read which is an extension of the entry Consistency model [5]. Unlike the model of entry consistency, which ensures that data is current as at the acquisition of its lock, this new type of operation can be achieved without locking, in parallel with write operations. However, data freshness constraint is released and older versions, which however still be controlled, are accepted. The grid architecture considered in this work is composed of clients requesting the data, the data providers and two hierarchical levels: LDG (Local Data Grid) and GDG (Global Data Grid).

Two types of copies are considered: local copy, hosted by the LDG and global copy, hosted by the GDG. When a client accesses the data, a request to acquire the synchronization object is addressed to the node hosting the local copy. If the node owns the synchronization object, the client is served. Otherwise, an acquisition request is sent to the node hosting the global copy.

To avoid the problem of storage data in grids, the consistency model proposed in [9] improved the storage space and access time of replicated data. The authors of this work suggest a topology built hierarchically upon three types of nodes: Super Node (SN), Master Node (MN), and Child Node (CN). The source of the replicated data is kept in the SN; this data can then be modified by users of the grid, called "original data". When the original data is added or modified, then it is automatically replicated to the master nodes (MN). The replica of the master node (MN) is called Master Replica. At the node (CN), the data is replicated from the master node (MN) according to two main factors: the file access frequency and the storage space capacity. The replicated data is called (Child Replica). Replicas located at (MN) and (CN) are read only.

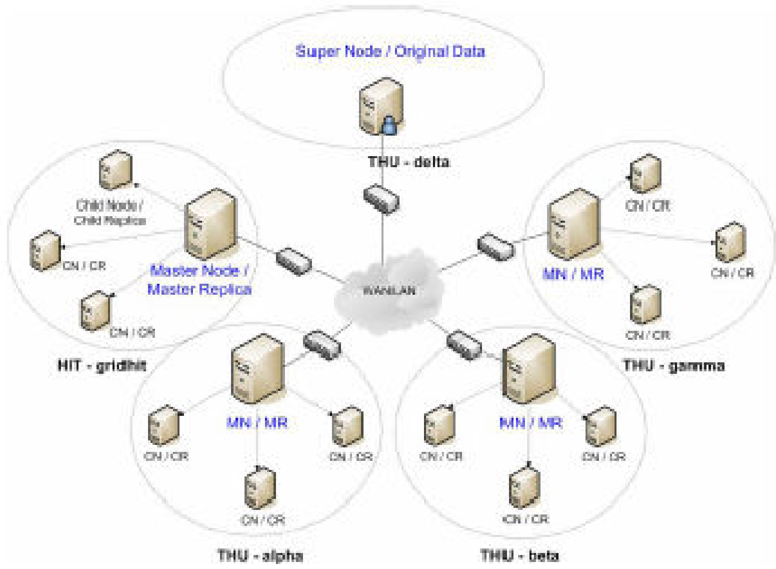


Fig. 1 Replica distribution topology [7]

Another similar work to [7] was proposed by [6] by considering the bandwidth consumed until the read/write operations. Most of existing replication works [9], [10] in data grid systems focuses on consistency management without taking care of the load imbalance of the grid nodes which can degrades significantly the replication performances.

Some of load balancing solution was proposed in the literature [8], [14]. For example in [1] Quorum systems are used. A Quorum is defined as the minimum set of nodes owning a replica. A coterie represents a set of replicas. Quorum protocols are characterized by two main properties which are properties of intersection and minimality [2]. Considering two quorums of a coterie C, the property $Q \cap Q' \neq \emptyset$ is called intersection property and the property $Q \not\subseteq Q'$ is called the minimality property. The authors of [1] studied the load balancing problem, by providing a coterie reconfiguration method, to improve the read/write accesses. The load of a quorum Q is the maximum load of the nodes of this Quorum and the load of a coterie is equivalent to the sum of loads of its quorums. The nodes are tree structured and a Quorum is obtained by taking nodes of any path from the root to a leaf of the tree. Every read (or write) operation is performed on a Quorum of the coterie.

An elementary permutation of the coterie is performed to obtain a new less loaded coterie. For this, two parent's nodes are selected to be swapped in the tree (father and its son) when the son's load is less than the father's load. The main goal of this work is to put nodes with the lightest load above the busiest ones.

An extension of the atomic read/write service [3] is proposed, with multiple readers and multiple writers. Two phases are proposed: query and a propagation phase. During the request phase, a read-quorum is contacted and each node returns the recent version which is consequently propagated to all the nodes of the quorum. This has the advantage that obsoletes copies are updated even during read operations.

As load balancing for replica consistency has not been sufficiently studied in the literature, we propose a strategy, based on quorum for load balancing of nodes to increase performance consistency in terms of availability, read/write accesses and communication cost. In the next section we present our approach.

III. Proposed Approach

In our work, we adopt a strategy of load balancing based on quorum structure, as this provides better representation and management of replicas. For this, we propose to represent the grid into coteries. A coterie contains all nodes owning replicas of the same data, structured in a binary tree. A path from the root to the leaf node of the tree is called quorum.

To improve the data availability, for each coterie node we define n versions. Each version is characterized by three parameters <N, S, V>, representing respectively, the node that creates or modified this version, the stamp which represents the moment of the creation or the update version of the replica and finally the value of the replica. An example is shown in Fig. 2.

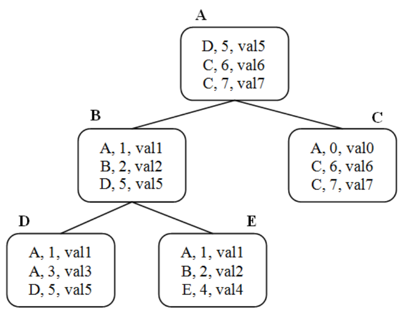


Fig. 2 Example of a coterie with versions

In this example, three versions are defined for each node of the coterie.

A. Replica Consistency Maintenance

The replica consistency is based on the write and read protocol. A replica is updated through a write protocol and requested through a read protocol. We define three states for a node: Free (F), Occupied (O) and Blocked (B) as shown in Fig. 3. A node is free if all its versions are released. A node is occupied if it contains at least one version locked. A node is blocked if all its versions are locked. The possible transitions from a state to another are illustrated in Fig. 3.

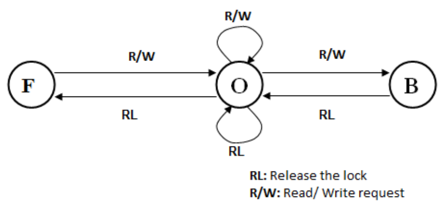


Fig. 3 State of a node in a coterie

Suppose that the initial state of the node is free (F). If a request (read/ write) is addressed to that node, the version chosen to perform the operation is locked and the node passes to occupied state (O). If this node receives another request then it keeps the same state even it still has released versions, else it transits to the blocked state (B). If the node is in a blocked state and a version has been released, then the node returns to an occupied state. The node returns to a free state if all the locks of all versions are released.

Write protocol

*The node N requests the write on the data **D**.*

```
Contact the coterie corresponding to the data D.
If exist a « Free » node then designate a quorum containing this
node, having no blocked nodes, with minimal occupied nodes and
maximal free nodes.
    Else if exist an « Occupied » node then designate a quorum
    containing this node, having no blocked nodes, with minimal
    occupied nodes.
        Else write is hold until the release of a node. // since all nodes
        of the coterie are blocked
        End if
End if
// Write the designated quorum
-   Lock the oldest version in writing (having the smallest
    estampille).
-   Perform the writing operation.
-   Release the write lock.
-   Propagate the written version to the nodes of the quorum.
```

Read protocol

*The node N requests the read on the data **D**.*

```
Contact the coterie corresponding to the data D.
If exist a « Free » node then designate a quorum containing this
node, having no blocked nodes, with minimal occupied nodes and
maximal free nodes.
    Else if exist an « Occupied » node then designate a quorum
    containing this node, having no blocked nodes, with minimal
    occupied nodes.
        Else write is hold until the release of a node. // since all nodes
        of the coterie are blocked
        End if
End if
// Read the designated quorum
-   Contact all nodes of the designed quorum and get the latest
    version of the replica (having the biggest estampille).
-   If there is divergence between replicas then propagate the latest
    version to the nodes of the quorum.
-   Lock the chosen version to be red.
-   Perform the reading operation.
-   Release the read lock.
-   Return the result read to the request node.
```

An example of write algorithm is illustrated in Fig. 4. Suppose that at time t = 8, the Node B requests the write of the data D with the value val8. The coterie corresponding to the data D is contacted and the quorum {A, B, D} is designated because it contains the node B, which is free. The oldest version (A, 1, val1) is locked to be written. The chosen version is locked to perform the writing operation. At the end of the write, the lock is released and it propagates the update to the quorum nodes.

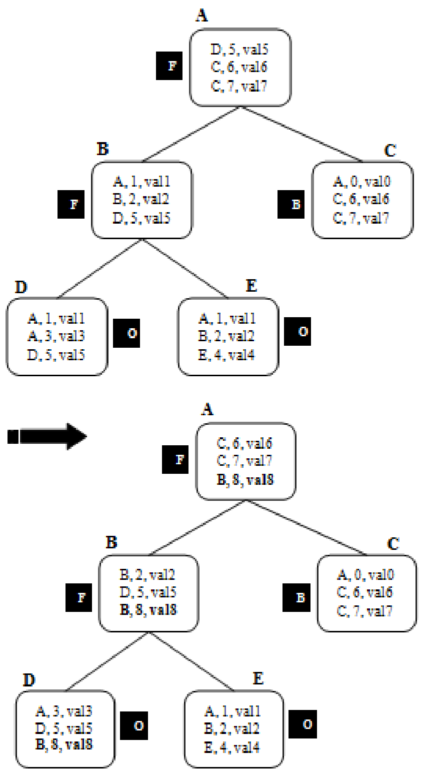


Fig. 4 Write example

An example of the read algorithm is illustrated in Fig. 5.

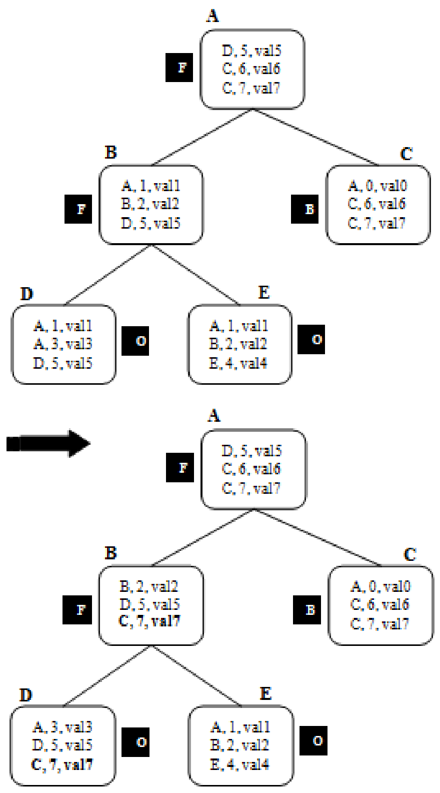


Fig. 5 Read example

At time t=8, the Node B requests the read of the data D. The coterie corresponding to the D is contacted and the quorum {A, B, D} is designed because it contains the node B which is free. After, the latest version is chosen in the designed quorum. Among the nodes of the chosen quorum, the latest version is located in the node A (C, 7, val7). As there is a divergence between the versions, the latest version must be propagated to the nodes that do not contain it. The latest version is locked to perform the reading operation. At the end of the read, the lock is released and the latest value is return to the request node.

B. Load Balancing Strategy

Before presenting our load balancing strategy, we precise how the load of a coterie is estimated.

First we define the load of each node of the coterie. In our approach, we define three load states: **under loaded, medium loaded** and **up loaded**, having respectively the values 1, 2 and 3.

The load of a node, noted $\mathbf{L_{node}}$ is calculated by following the read/write access frequency. The node access frequency, noted $\mathbf{fa_{node}}$, is incremented at each read operation and reinitialized periodically. For this, two thresholds: $\mathbf{fa_{min}}$ and $\mathbf{fa_{max}}$.are defined arbitrary.

$$\textit{if } (fa_{node} < fa_{min}) \textit{ then } L_{node} = 1$$
$$\textit{if } (fa_{min} \leq fa_{node} < fa_{max}) \textit{then } L_{node} = 2$$
$$\textit{if } (fa_{node} \geq fa_{max}) \textit{then } L_{node} = 3$$

The load of a Quorum, noted $\mathbf{L_{quorum}}$, represent the maximum load of the nodes of this quorum.

Finally, the load of a Coterie, noted $\mathbf{L_{coterie}}$, represents the sum of all quorums load of this coterie.

In the following example, we assume that the nodes {A, B, C, D and E} have got respectively the following load values {2, 2, 3, 3, 1}.

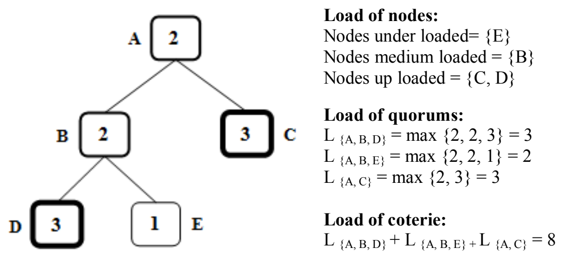

A 2
B 2
3 C
D 3
1 E
Load of nodes:
Nodes under loaded= {E}
Nodes medium loaded = {B}
Nodes up loaded = {C, D}
Load of quorums:
L {A, B, D} = max {2, 2, 3} = 3
L {A, B, E} = max {2, 2, 1} = 2
L {A, C} = max {2, 3} = 3
Load of coterie:
L {A, B, D} + L {A, B, E} + L {A, C} = 8

Fig. 6 Example load

Load balancing nodes of each coterie, is performed by following a dynamic reconfiguration of nodes of the coterie from the root nodes to the leaf nodes *(see load balancing strategy)*. The purpose of this reconfiguration is to reduce the load of quorums based on an elementary permutation between a parent node and its two son nodes so that the load of the father is greater than the load of its two nodes son getting a minimal communication cost. The communication cost represents the time of exchanges messages between two nodes of the grid.

Thus the main objective of this reconfiguration is to involve the least possible overloaded nodes in the construction of quorums. In this way, we get the overloaded nodes at the lowest level (leaf level) without degrading the performance of consistency in terms of communication cost. This reconfiguration is invoked periodically.

Load balancing strategy

```
Input: structured coterie, load nodes of each coterie.
Output: restructured coterie.

For each coterie do
If exist node ≠ leaf and load node=3
Then
For each parent node do
If (load_parent > load_child1) and (load_parent > load_child2)
Then Swap (parent, child_j) with minimal communication cost
Else if (load_parent > load_child1)
        Then  swap (parent, child_1)
        Else if (load_parent > load_child2)
                Then swap (parent, child_2)
                End if
            End if
          End if
     End if
End if
```

IV. Experimentations

To evaluate the performances of our proposed approach, we use the grid simulator (Gridsim toolkit 4.2) [11] under the operating system (Windows XP 7). We defined different number of grid nodes. We replicate arbitrary 10 data into 5 versions over the nodes of the grid. We fixed the number of R/W transactions to 10000.

In order to estimate the load of a node, we define two thresholds $\mathbf{fa_{min}}$ and $\mathbf{fa_{max}}$ by following the number of nodes and transactions. We approximate the $\mathbf{fa_{min}}$ to (number transactions/ number of nodes). To consider the upload state of a node, we approximate the $\mathbf{fa_{max}}$ to $\mathbf{fa_{min}+5}$ as shown in Table I.

TABLE I
SIMULATION PARAMETERS

# Data= 10, # Version=5, #Transactions=10000									
# Nodes	50	100	400	800	1000	2000	3000	5000	10000
Fa_{min}	200	100	25	13	10	5	4	2	1
Fa_{max}	205	105	30	18	15	10	9	7	6

In order to study the consistency results, we assume that a replica is consistent if its latest version corresponds to the latest written value. Thus, the consistency of a data D represents the rate of the consistent replicas of the data D. the consistency of a data D is calculated as below:

$$Consistency_D = \frac{\text{number of nodes hosting consistent replicas of the data « D »}}{\text{number of nodes hosting replicas of the data « D »}} * 100$$

We note that the consistency reduces when the number of nodes increases, this is explained by the fact that when there is a lot of write operations (in our experimentations we fix it to 10000 R/W transactions) the updates of replicas becomes more difficult and consequently inconsistencies occur. That is why we study the freshness of replicas.

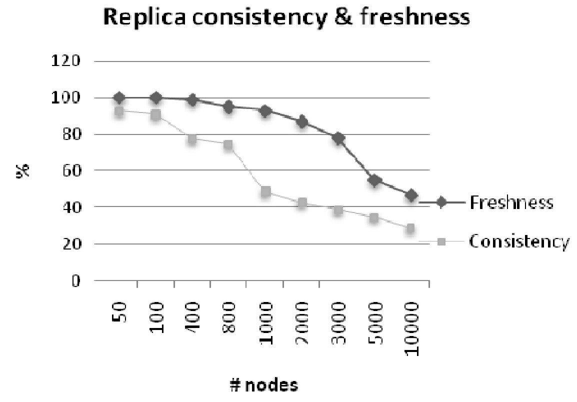


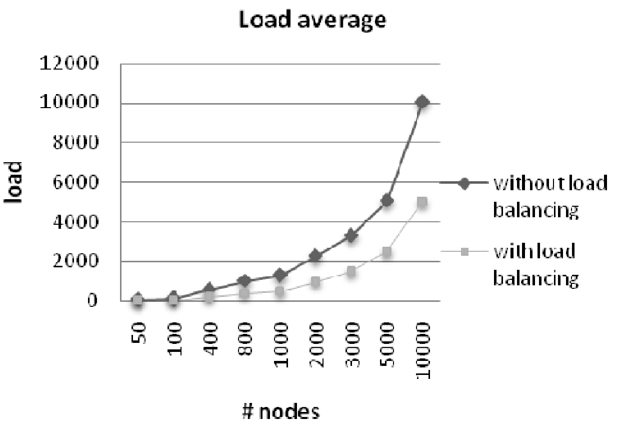
Fig. 7 Consistency& Freshness

Considering a set of replicas of the data « D » = {R_0, R_1… R_n} where: R_0 represents the first written replica and R_n the latest written replica. The freshness margin of a replica R_i is equal to n-i. A replica is assumed to be « fresh » if its freshness margin is lower than n/2. The freshness of a data D is calculated as below:

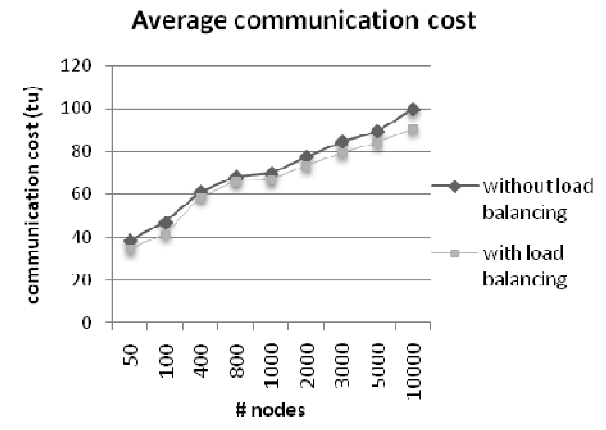
$$Freshness_D = \frac{\text{number of nodes hosting fresh replicas of the data « D »}}{\text{number of nodes hosting the replicas of the data « D »}} * 100$$

In fact, the experimentation results show that the system holds fresh replicas.

Moreover, the average load of the coteries of replicated data (see Fig. 8 (a)) is balanced by using our reconfiguration strategy with taking care of reducing the communication cost between nodes (see Fig. 8 (b)).



(a)



(b)

Fig. 8 Load balancing results

V. CONCLUSION

In this paper we presented our contribution of replica consistency through a dynamic load balancing strategy. The use of a structured tree (coterie) allows a better logical organization of the grid nodes hosting a set of replicas. The definition of multiple versions of a replica can serve as many grid users as available versions and with a certain degree of similarity with the last update of the replicated data. Quorum structure ensures the existence at any time of the latest version of the replicated data. This is explained by the fact that the root node always has the latest version. Indeed, during a read / write operation, the latest version is always propagated to the quorum nodes designated for reading / writing. As a quorum is built from the root to a leaf of the tree, then the root node participates in any designed quorum.

The consistency between replicas is not strong in our work in order to serve a maximum of read request. Despite this divergence between copies, the freshness of replicas is assured in our work.

Reconfiguration of the coterie provides better load balancing to increase access performance. The obtained results of our approach reveal that the consistency management of replicas is balanced dynamically following the state of each node of the coterie.

REFERENCES

[1] Ivan Frain, Jean-Paul Bahsoun, Abdelaziz M'zoughi , *« Reconfiguration Dynamique de Coterie Structurée en arbre »,* Institut de Recherche en Informatique de Toulouse, projet RNTL ViSaGe, acte CDUR 2005

[2] Hector Garcia-Molina and Daniel Barabara. «*How to assign votes in a distributed system*». Journal of the ACM, 32(4):841–860, October 1985.

[3] N. A. Lynch and A. A. Shvartsman. «Robust emulation of shared memory using dynamic quorum-acknowledged broadcasts ». In FTCS '97: Proceedings of the 27th International Symposium on Fault-Tolerant Computing (FTCS '97). IEEE Computer Society, 1997.

[4] Sébastien Monnet, « Gestion des données dans les grilles de calcul : support pour la tolérance aux fautes et la cohérence des données », l'université de rennes 1, Thèse de doctorat 2006.

[5] Liviu Iftode, Jaswinder Pal Singh, and Kai Li. «Scope consistency: A bridge between release consistency and entry consistency». In *Proceedings of the 8th ACM Annual Symposium on Parallel Algorithms and Architectures (SPAA '96)*, pages 277.287, Padova, Italy, June 1996.

[6] Changqin Huang, Fuyin Xu, and Xiaoyong Hu, «*Massive Data Oriented Replication Algorithms for Consistency Maintenance in Data Grids*», Part I, LNCS 3991, pp. 838 – 841, 2006.

[7] Chao-Tung Yang Wen-Chi Tsai Tsui-Ting Chen Ching-Hsien Hsu. *«A One-way File Replica Consistency Model in Data Grids»*, Tunghai University, Taiwan. IEEE Asia-Pacific Services Computing Conference 2007.

[8] «*Load Balancing and Replication*», Chapter in Springer-Verlag Berlin Heidelberg 2010.

[9] Cécile Le Pape and Stéphane Gançarski, *« Replica Refresh Strategies in a Database Cluster »*. LIP6, LNCS 4395, pp. 679–691, 2007.

[10] Hartmut Kaiser, Kathrin Kirsch, and Andre erzky, «*Versioning and Consistency in Replica Systems*», LNCS 4331, pp. 618–627, 2006.

[11] R. Buyya and M. Murshed,«GridSim: a toolkit for the modeling and simulation of Distributed resource management and scheduling for grid computing».. Concurrency computat: pract. Exper., 2002

[12] Distributed Computation Laboratory, « replication consistency models», 8 april 2008.

[13] H. Guo and al. «Relaxed currency and consistency: How to say good enough in sql». In ACM SIGMOD int. conf., 2004.

[14] James J. (Jong Hyuk) Park et al. «Data Consistency for Self-acting Load Balancing of Parallel File System». ITCS & STA 2012, LNEE 180, pp. 135–143, DOI: 10.1007/978-94-007-5082-1_18.

[15] Saito, Y., Shapiro, M. «Optimistic replication. Comput. Surveys» 37(1), 42–81. 2005

Sarra SENHADJI, 18- 06- 1987, Algeria. Master in computer science in 2009. Phd student since 2009 at the University of Science and Technology of Oran, Algeria. Interest in data mining, data replication and data grid.

Amar KATEB, Magister in computer science in 2013 at the University of Science and Technology of Oran, Algeria.

Professor Hafida BELBACHIR, department of computing, USTO, BP 1505 El M'Nouer, Oran, Algeria. E-mail: h_balbach@yahoo.fr. PhD in computer science in 1990. Interest in advanced databases, data mining and data grid. Prof. BELBACHIR is the head of the database System Group of Signal, Systems and Data Laboratory.