A Load Balancing Strategy for Replica Consistency Maintenance in Data Grid Systems

Senhadji Sarra, Kateb Amar and Belbachir Hafida
LSSD Laboratory, University of science and Technology, Oran, Algeria
E-mail: senhadji.sarah@gmail.com, kateb_amar@yahoo.fr and h_belbach@yahoo.fr

Keywords: consistency, replication, data grid, load balancing

Received: January 28, 2013

Abstract: In data grid environment, the management of shared data is one of the major scientific challenges. Data replication is one of the important techniques used in grid systems to increase the availability, scalability and fault tolerance. However, the update of a replica might bring a critical problem of replica consistency maintenance. Thus, maintaining the consistency of the replicas is not trivial because of the instability of the grid system where the nodes can join and leave at any time. In addition, according to Read/Write access frequency some nodes can be more uploaded than others and become a bottleneck. In order to handle these problems, we propose a model of consistency based on quorum protocol. The replica consistency performances are improved by using a dynamic load balancing strategy in a simulated grid environment.

Povzetek: Članek govori o ohranjanju konsistentnosti kopij podatkov v omrežnem računalništvu in predlaga nov model, ki temelji na protokolu sklepčnosti.

1 Introduction

Grids [7, 25] are a wide area computing system geographically dispersed that involves high computational and storage resources. The grid is considered as one of the promiscuous technologies for scientific applications like astronomy, bioinformatics and earth sciences. In this kind of dynamic and large scale environment, the management of massive data is still being one of the important open problems [14, 6]. Different techniques dedicated for intensive data management are used in several domains. Data replication is one of the most important techniques having attracted the researcher community attention.

The replication technique improves data availability, scalability and fault tolerance. The main research areas dealing with data replications are the replica placement and the replica consistency maintenance.

The replicas placement mechanism determines which data should be replicated? When to create new copy? And where the new replicas should be placed? For replica placement, many works and solutions have been proposed in the literature [17, 18, 22, 23]. The main objective of these solutions is to store copies (or replicas) of a data in different sites, so that data can be easily restored if one copy is lost. Also, by placing replicas closer to grid users, the access performances are significantly improved in term of response time, consumed bandwidth, etc.

However, the update of replica by any grid user might bring a critical problem of maintaining consistency among the other replicas of the grid. In fact, when a replica is modified, a copy must be propagated to the rest of replicas in order to get identical and consistent copies.

Moreover, according to the frequency access to the different grid nodes, some nodes are more uploaded then others. This irregular evolution engenders an unbalance and many resources can be unexploited. With an efficient load balancing strategy, the system can reduce the query response time and avoid failures due to overloaded nodes. That is why we propose a complementary solution between replica consistency and load balancing.

For this, we propose a consistency protocol based on quorum system [5]. The main idea of quorum systems is to involve a large collection of possible sets of nodes in the replica consistency management. Nodes holding replicas of the same data are represented logically into a tree structure, called Coterie. When a Read/ Write request is addressed to a grid node, a path from the root to the leaf tree, called quorum, is selected to achieve the replica consistency. Complementing to replica consistency, we define a load balancing strategy, based on elementary permutation between coterie's nodes in order to reduce the load and communication time of R/W request.

Thus, the main idea of our contribution consists of ameliorating the replica consistency performances through a dynamic load balancing strategy in term of consistency, load balancing and communication cost.

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

2 Related works

Various works have been done on the replica consistency domain in distributed systems, such as cluster, peer to peer and grid. Many consistency models exist in the literature [1] as Strong models and Weak models [9, 24]. Strong consistency models keep data consistent among all replicas simultaneously, which requires more resources and expensive protocols than other models. In weak consistency models the strong consistency protocols are relaxed in order to tolerate inconsistencies among replicas for a while to improve access performances. In consequence, replicas returned to a read request are not perfectly the latest updated value. For this reason, replica divergence must be controlled since maintaining replica freshness becomes more complex as divergence increases.

In [8] the authors address 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 [15]. 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 handle the problem of storage data in grids, the consistency model proposed in [4] 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.

Another similar work to [4] was proposed by [3] by considering the bandwidth consumed until the Read/ Write operations. Most of existing replication works [2, 10] in data grid systems focuses on consistency management without taking care of the load imbalance of the grid nodes which can low significantly the replication performances.

Some of load balancing solutions have been proposed in the literature [13, 20].For example in [12] Quorum systems are used under a simulated environment [5]. A coterie represents a set of copies of the replicated data. A Quorum is defined as the minimum set of nodes owning a replica. Quorum protocols are characterized by two main properties which are properties of intersection and minimality [11]. Considering two quorums Q and Q' 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 [12] treated 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 all 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. This has the effect of positioning the node with the lightest load above the busiest node. An extension of the atomic read / write service [16] 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 obsolete copies are updated even during read operations.

As few works attempted to resolve the load balancing problem with replica consistency, our contribution is to propose a dynamic load balancing strategy to increase replica consistency performances in terms of load and communication cost.

3 Proposed approach

In the literature, few studies addressed the problem of load balancing to increase the replica access performances. In [12] load balancing is adopted by introducing dynamic node permutations, but regardless of the problem of communication cost generated during Read /Write access. In our work, nodes hosting replicas of the same data are represented into a binary tree, called Coterie. Indeed, in a R /W access, a Quorum designed from the root to a leaf node is selected to be red or written. An intermediate node can participate in different possible Quorums. These intermediate nodes can represent a critical point where the cost of communication may be degraded during exchange accesses. To handle this problem, when a Read/ write request is addressed, our load balancing strategy is invoked and the Coterie is restructured in order to reduce the load of the coterie and the communication cost of the quorum when the Read/Write request is achieved.

3.1 Grid and replicas model

The nodes of the grid are represented into coteries. A coterie that owns all replicas of the same data is structured in a binary tree. In order to improve 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 modifies 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 figure 1.

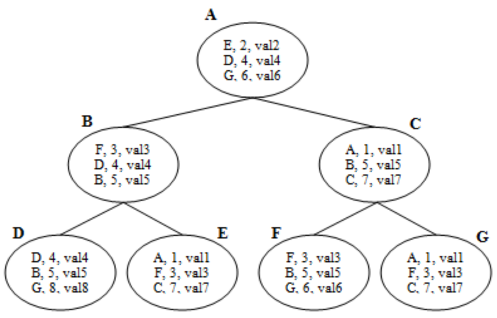


Figure 1: Example of a coterie with versions.

In figure 1, three versions are defined for each node of the coterie. For example, the node A owns the following versions **<E, 2, val2>, < D, 4, val4>, < G, 6, val6>**

3.2 Replica consistency

In order to achieve the consistency protocol, a replica is updated through a write protocol and requested through a read protocol. Before presenting the read/write protocol, we assume that a version can be locked or released. In addition, a node can take one of these three states: Free (F), Occupied (O) and Blocked (B). A node is free if all its versions are released. A node is occupied if it contains at least one released version. A node is blocked if all its versions are locked. The possible transitions from a state to another are illustrated in figure2.

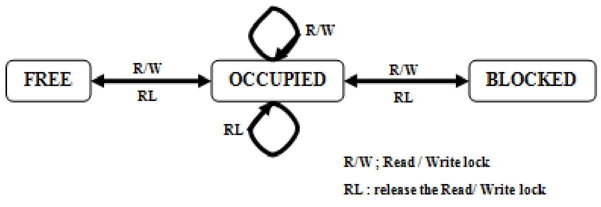


Figure 2: State of a node in a coterie.

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, otherwise 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.

3.2.1 Write protocol

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

```
Choose the coterie corresponding to the data D.
If exist free nodes in the chosen coterie
Then
 Begin
    Select one of the free nodes that is near root node
    If exist quorums containing the selected free node with
     no blocked nodes
    Then
    Begin
      - Choose one of the existing quorums getting minimal
           occupied nodes and maximal free nodes.
      - Write on the selected free node of the chosen quorum.
           (See Write quorum algorithm)
    End
    End If
  End
Else If exist occupied nodes in the chosen coterie // an
occupied node has got at least one released version
Then
 Begin
   Select one of the occupied nodes that is near root node
   If exist quorums containing the selected occupied node with
   no blocked node
   Then
     Begin
       - Choose one of the existing quorums getting minimal
            occupied nodes.
         - Write on the selected occupied node of the chosen
            quorum. (See Write quorum algorithm)
     End
    End If
End
Else Write operation aborted.//since all nodes of the chosen
coterie are blocked
End If
End If
```

Write quorum algorithm

```
Write on the oldest version of the selected node (having the
smallest stamp).
// propagate the written version to all nodes of the quorum
For the other nodes of the chosen quorum do
  If the latest version is locked in writing
  Then abort the propagation
  Else Write on the oldest version (having the smallest
  stamp).
  End If
End For
```

3.2.2 Read protocol

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

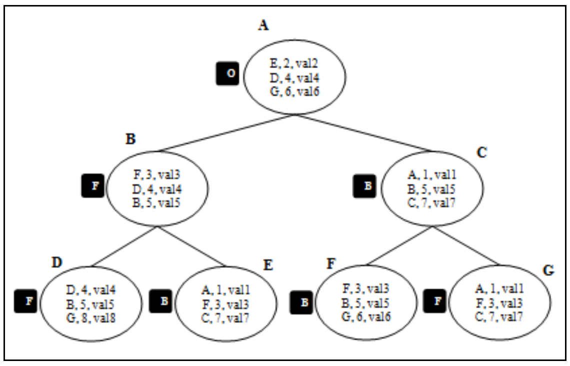
```
Choose the coterie corresponding to the data D.
If exist free nodes in the chosen coterie
Then
 Begin
   Select one of the free nodes that is near root node
  If exist quorums containing the selected free node with no
  Blocked node
  Then Begin
        -Choose one of the existing quorums getting minimal
        occupied nodes and maximal free nodes
```

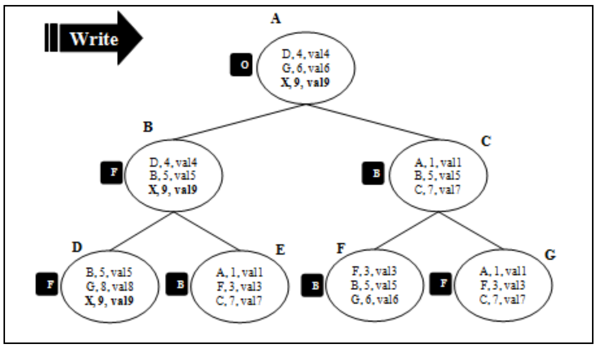
```
            -Read on the selected free node of the chosen quorum.
            (See Read quorum algorithm)
           End
    End If
   End
Else
If exist occupied nodes in the chosen coterie
Then Begin
        Select one of the occupied nodes that is near root node
        If exist quorums containing the selected occupied node
        with no blocked node
        Then Begin
                - Choose one of the existing quorums getting
                  minimal occupied nodes.
                - Read on the occupied node of the chosen
                  quorum. (See Read quorum algorithm)
             End
        End If
       End
Else Read request aborted.// since all nodes of the chosen
coterie are blocked
End If
End If
```

Read quorum algorithm

```
Select the latest version of replicas (having the biggest stamp)
of each node of the chosen quorum.
Read the selected last version.
If there is divergence between replicas of each node of the
quorum
Then //propagate the selected last version to the quorum nodes.
   For each node of the chosen quorum do
   If the latest version is locked in writing
   Then abort the propagation
   Else Write on the oldest version of the selected node (having
            the smallest stamp).
   End If
   End For
End If
```

In the following examples, a coterie of a data D with 3 versions is represented with 7 nodes A, B, C, D, E, F and G having respectively the state occupied, free, blocked, free, blocked, blocked and free. An example of write and read algorithm on the coterie of data D is illustrated in the figure 3.





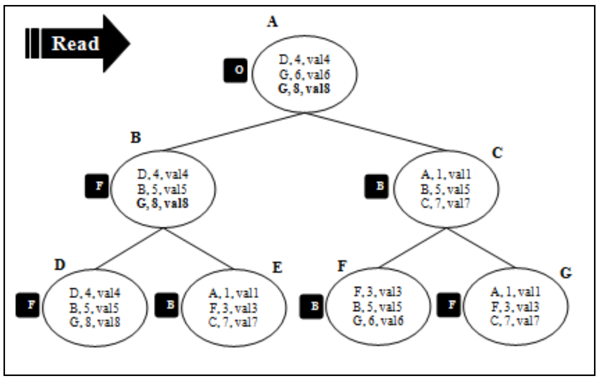


Figure 3: Write/ Read example.

Write: Suppose that at time t = 9, a Node X requests the write of the data D with the value val9. The corresponding coterie of the data D is contacted. There are three free nodes in the corresponding coterie: B, D and G but the node B is chosen because it is near the root node. Thus, two possible quorums can be designed from the node B: {A, B, D} and {A, B, E} and the quorum {A, B, D} is chosen to be written because it doesn't get blocked nodes. The oldest version (E, 2, val2) is locked to be written. At the end of the write, the lock is released and the new version (X, 9, val9) is propagated to the quorum nodes. In the propagation phase, the oldest version of each node is replaced by the new written version, for example in the node F the oldest version (D, 4, val4) is replaced by the new version (X, 9, val9).

Read: At time t = 9, a Node X requests the read of the data D. The coterie corresponding to the data D is contacted and the quorum {A, B, D} is designed. 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 D (G, 8, val8). 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. As there is a divergence between the latest versions of each node of the quorum, the latest version (G, 8, val8) must be propagated to the nodes that do not contain it. In the propagation phase, the oldest version of each node is consequently replaced by the latest version (G, 8, val8).

3.3 Load balancing strategy

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

In our approach, we define three load states of a node: **under loaded, medium loaded** and **up loaded**. In addition, we assign a numeric value to each state as follows:

Under loaded (1): the node is inactive or downloaded.
Medium loaded (2): the node is midway loaded.
Up loaded (3): the node is overloaded.

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/ write operation and reinitialized periodically. For this, two thresholds: $\mathbf{fa_{min}}$ and $\mathbf{fa_{max}}$ are defined (The choice of the values of these thresholds will be defined in experimentations section).

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

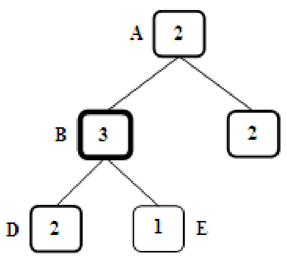
The load of a Quorum, noted $\mathbf{L_{quorum}}$, represent the maximum load of the nodes appertaining to this quorum.

$$L_{quorum} = Max_{node \in quorum}\{L_{node}\}$$

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

$$L_{coterie} = \sum\nolimits_{quorum \in coterie} L_{quorum}$$

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



Load of nodes:

Nodes under loaded= {E}
Nodes medium loaded = {A, C, D}
Nodes up loaded = {B}

Load of quorums:

$L_{\{A, B, D\}} = \max\{2, 3, 2\} = 3$
$L_{\{A, B, E\}} = \max\{2, 3, 1\} = 3$
$L_{\{A, C\}} = \max\{2, 2\} = 2$

Load of coterie:

$L_{\{A, B, D\}} + L_{\{A, B, E\}} + L_{\{A, C\}} = 3+3+2= 8$

Figure 4: Example of coterie load.

The load balancing is performed by following a dynamic reconfiguration of nodes of a coterie. The purpose of this reconfiguration is to reduce the load of quorums and the communication cost of Read / Write queries.

The communication cost represents the exchanged messages between two nodes of the grid. This cost can represent the bandwidth, the debit, the latency, etc. In our work we assume that the value of communication cost between any two nodes is provided with the file configuration of the system and it doesn't need to be calculated. As we suppose that the cost communication between grid nodes is brought by the configuration system file, the consumed communication cost of a Read/Write request can be evaluated as follow:

Write: When a node **i** writes a new value of the data D, a quorum Q of the corresponding coterie is contacted. We assume that the write cost and the size of the new value are negligible. The nodes appertaining to this quorum communicate with each other following a propagation phase then an acknowledgement phase.

$$com\_cost_{Write\_Qi} = (N-1) \times \left(\sum_{\substack{i \neq j \\ j \in Q}} prop\_cost_{i,j} + \sum_{\substack{i \neq j \\ j \in Q}} ack\_cost_{j,i} \right)$$

Where:
N: number of nodes appertaining to the quorum Q
Com_cost $_{Write\_Qi}$***:*** communication cost of the write quorum initiated by the node i
Prop_cost $_{i,j}$: propagation cost of the new value from the node i to the node j
Ack_cost $_{j,i}$***:*** acknowledgment cost from the node j to the node i

Read: When a read request is invoked by a node **i**, a quorum Q of the corresponding coterie is selected. Nodes appertaining to this quorum communicate to each other following a selection of latest version then the propagation of the selected last version then the acknowledgement phase.

$$com\_cost_{Read\_Qi} = (N-1) \times \left(\begin{array}{c} \sum_{\substack{i \neq j \\ j \in Q}} slc_{cost_{j,i}} + \sum_{\substack{i \neq j \\ j \in Q}} prop_{cost_{i,j}} \\ + \sum_{\substack{i \neq \\ j \in Q}} ack\_cost_{j,i} \end{array} \right)$$

Where:
N: number of node appertaining to the quorum Q
Com_cost $_{Write\_Qi}$ ***:*** communication cost of the read quorum initiated by the node i
Slc_cost $_{j,i}$ ***:*** selection cost of the latest version to be red from the node j to the node i
Prop_cost $_{i,j}$: propagation cost of the new value from the node i to the node j
Ack_cost $_{j,i}$ ***:*** acknowledgment cost from the node j to the node i

Our load balancing strategy is achieved with a dynamic reconfiguration of nodes of the coterie from the root node to the leaf nodes. This reconfiguration is 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 main objective of this reconfiguration is to involve at least possible the overloaded nodes in the construction of quorums. In this way, we get overloaded nodes at the lowest level of the coterie (leaves level) and this without degrading the performance of consistency in terms of load and communication cost. This reconfiguration is triggered at each Read/ Write request.

```
*Load balancing strategy*
**Input**: structured coterie, load nodes of each coterie.
**Output:** restructured coterie.

**For** each coterie **do**
 **// seek the tree from the root to the leaf nodes**
 **For** each parent node **do**
 // $N_1$, $N_2$ {$child_1$, $child_2$} and $N_1 \neq N_2$
 **If** (load$_{parent} \geq$ **max** (load$_{N1}$, load$_{N2}$)
 **Then If** ((cost$_{(N1, N2)} \leq$ cost$_{(parent, N2)}$) and
        (cost$_{(N1, N2)} >$ cost$_{(parent, N1)}$)
     Then Swap (Parent, $N_1$)
     **Else If** ((cost$_{(N1, N2)} \leq$ cost$_{(parent, N1)}$) and
        (cost$_{(N1, N2)} >$ cost$_{(parent, N2)}$)
       Then Swap (parent, $N_2$)
      **End if**
   **End if**
  **Else If** (load$_{parent} \geq$ load$_{N1}$) and
      (cost$_{(N1, N2)} \geq$ cost$_{(parent, N2)}$)
    Then Swap (Parent, $N_1$)
     **Else if** ((load$_{parent} >$ load$_{N2}$) and
        (cost$_{(N1, N2)} \geq$ cost$_{(parent, N1)}$)
       Then Swap (parent, $N_2$)
       **End if**
    **End if**
  **End if**
 **End For**
**End For**
```

The following example illustrates a coterie composed of nine nodes (A, B, C, D, E, F, G, H, I). Each node is represented by its load. The load of the coterie is calculated from the sum of the quorum's load. Thus, the load of the coterie before the reconfiguration is equal to 15.

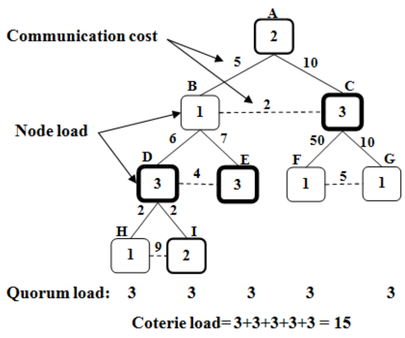


Figure 5: Load of the coterie before reconfiguration.

The reconfiguration of this coterie is performed from the root to a leaf node of the coterie, by considering three nodes: parent node and its two child nodes. Four cases are distinguished as follow:

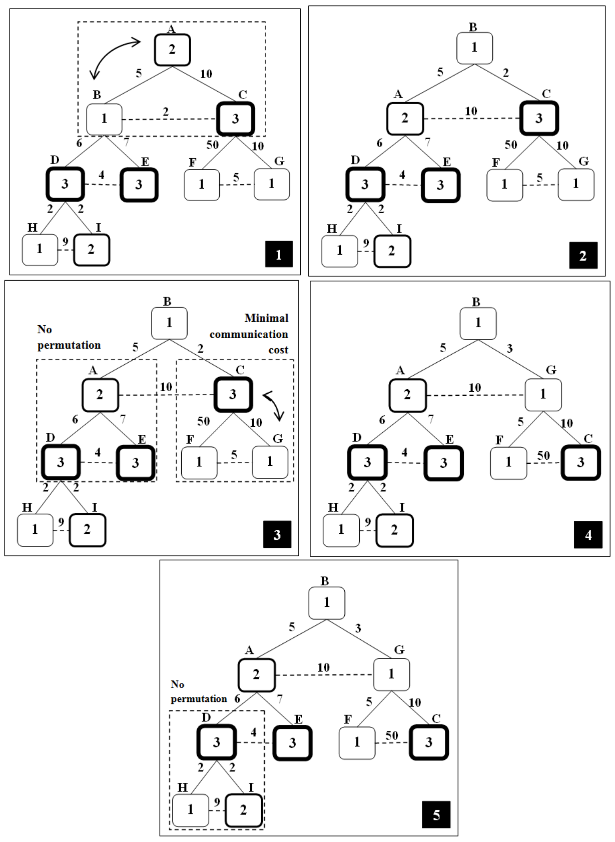


Figure 6: Reconfiguration of a coterie.

In Figure 6.1 as the parent node load (node A) is greater than the child node load (node B) the permutation between Node A (father) and Node B (son) which brings a minimal communication cost is illustrated in Figure 6.2.

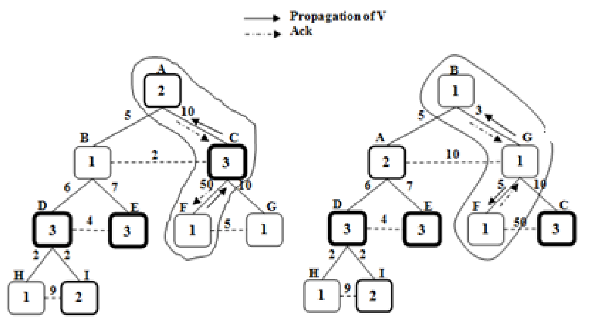
In Figure 6.3 as the parent load (node A) is less than its two child loads (node D and E) then no permutation will be made. In Figure 6.3 the parent load (node C) is greater than its two child loads (F and G) then the child node G is permuted with the parent node C which brings a minimal communication cost, as shown in Figure 6.4.

In Figure 6.5 in spite the parent load (node D) is greater than the load of its two Childs (node H and I); no permutation is possible because this will increase the cost of communication instead of reducing it. The load of the coterie after reconfiguration became equal to 13.

In addition, to show how the cost of communication is optimized we suppose that a write request is addressed to the node F with the value V. So the communication cost of a write request is estimated as defined in section 3.3.

We conclude that the communication cost is reduced from 180 to 26 by using our load balancing strategy with taking care of a minimal communication cost.

We present the simulation results of our proposed approach in the next section.



before reconfiguration **com_cost_{write} =180**

after reconfiguration **com_cost_{write} =26**

Figure 7: Communication cost before and after reconfiguration.

4 Experiments

We evaluate the performances of our consistency approach by using the grid simulator ***Gridsim toolkit 4.2*** [21] under the operating system ***Windows XP*** *7*.We defined different number of grid nodes. We replicate arbitrary **50 data** into **5 versions** over each node of the grid.

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 the number of transactions (Read / Write requests). After many experiments, we conclude that the approximation of the $\mathbf{fa_{min}}$ that gives best results is calculated as follow:

If (**number_transactions >= number_nodes)**
Then $\mathbf{fa_{min}}$ =0
Else $\mathbf{fa_{min}}$= **number_transactions / number_nodes**.

To consider the uploaded state of a node, we approximate the $\mathbf{fa_{max}}$ to $\mathbf{fa_{min}}$**+3**.

# Data= 50, # Version=5		
# Experiment	**# Nodes**	**# Read/ Write Transactions**
experiment 1	**500**	**200, 500, 1000, 5000, 10000**
experiment 2	**1000**	**500, 1000, 5000, 10000, 50000**
experiment 3	**5000**	**1000, 5000, 10000, 50000, 100000**

Table 1: Simulation parameters.

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 is calculated as below:

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

In figure 8, we note that the consistency is variable, this is explained by the fact that when there is a lot of write operations, the updates of replicas becomes more difficult and consequently inconsistencies occur. That is why we study the freshness of replicas.

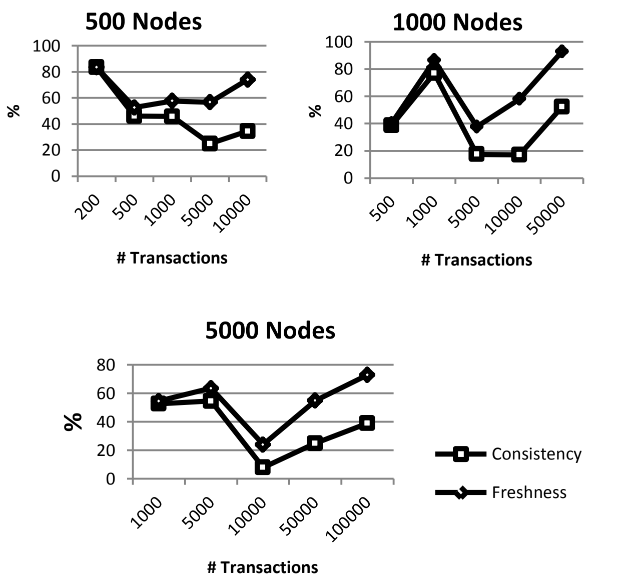


Figure 8: Consistency& Freshness.

Considering a set of replicas of the data **D** = {R_0, R_1… Rn} where: R_0 represents the first written replica and Rn the latest written replica. The freshness margin of a replica Ri 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:

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

In fact, the experimentation results show that with the presence of inconsistent copies, the system still hold fresh replicas, as shown in figure 8.

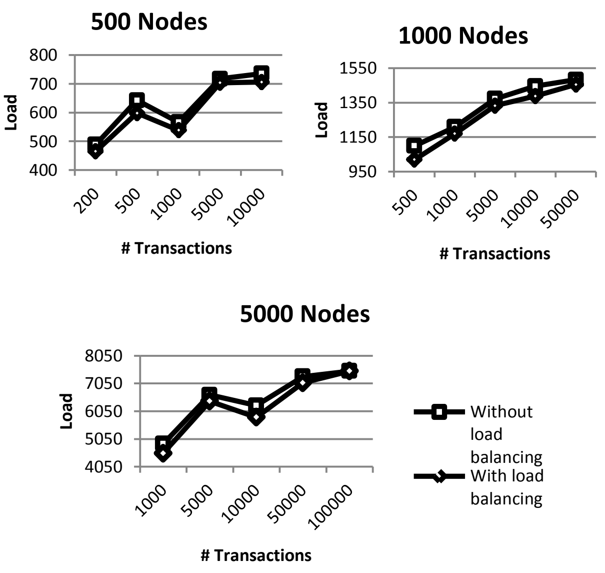


Figure 9: Load balancing results 1.

In figure 9, we note that when the number of transactions is significantly higher than the number of nodes, the average load of coteries increase, otherwise the load is variable. Also, the load of coteries is reduced when our load balancing strategy is used.

In the same time, the communication cost consumed for R/W accesses is significantly reduced as shown in figure 10.

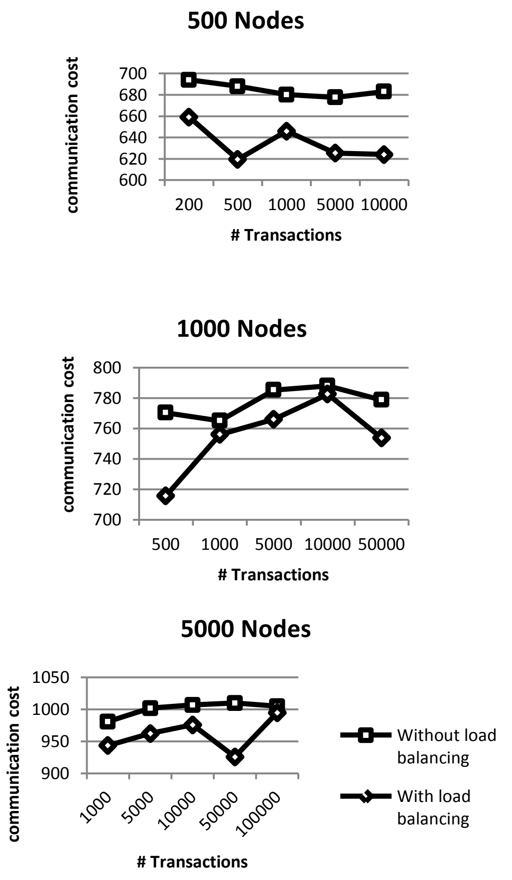


Figure 10: Load balancing results 2.

We conclude the main objectives of our work were gained in term of consistency, load balancing and communication cost.

The experiments results demonstrate that our approach guarantee the freshness when replicas are not strongly consistent. Also, when the load balancing results are studied, we note that the load of coteries and the communication cost of R/W queries are significantly reduced when the load balancing strategy is applied.

5 Conclusion

In this paper we presented our contribution to improve replica consistency trough 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 grid still holds fresh replicas.

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. The obtained results demonstrated the efficiency of our work in term of consistency, load balancing and communication cost of Read/ Write queries.

Finally, our work can present many perspectives; we cite the most interesting ones:

- Study the impact of different replica placement on the consistency performances
- Assure the fault tolerance of the root node of each coterie

References

[1] Alan D. Fekete and Krithi Ramamritham, « Consistency Models for Replicated Data», Replication Lecture Notes in Computer Science Volume 5959, 2010, pp 1-17

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

[3] 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.

[4] 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.

[5] Christian Storm, « Specification of Quorum Systems », Specification and Analytical Evaluation of Heterogeneous Dynamic Quorum-Based Data Replication Schemes, 2012, pp 81-153

[6] Esther Pacitti, Patrick Valduriez, Marta Mattoso, Grid Data Management: Open Problems and New Issues, Journal of Grid Computing, September 2007, Volume 5, Issue 3, pp 273-281

[7] Foster, I.: « What is the Grid ? A Three Point Checklist », Grid Today, 1(6), (2002).

[8] Gabriel Antoniu, Jean François Deverge and Sébastien Monnet,« How to bring together fault tolerance and data»,INRIA research report 2005.

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

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

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

[12] Ivan Frain, Jean-Paul Bahsoun, Abdelaziz M'zoughi,« dynamic reconfiguration of a coterie tree-structured »,RNTL ViSaGe project, CDUR 2005

[13] 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.

[14] Jinchuan Chen, Yueguo Chen, Xiaoyong Du, Cuiping Li, Jiaheng Lu, Suyun Zhao, Xuan Zhou, « Big data challenge: a data management perspective », Frontiers of Computer Science, April 2013, Volume 7, Issue 2, pp 157-164

[15] 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.

[16] 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.

[17] N. Tziritas et al. «Implementing Replica Placements: Feasibility and Cost Minimization»,in IPDPS 2007

[18] N. Tziritas et al. «Using Multicast Transfers in the Replica Migration Problem: Formulation and Scheduling Heuristics», Europar 2009.

[19] Peter Urban, Xavier Défago, and André Schiper. «Neko: A single environment to simulate and prototype distributed algorithms». In 15th Int'l Conference on Information Networking (ICOIN-15), pages 503–511, 2001.

[20] Quang Hieu Vu, Mihai Lupu, Beng Chin Ooi, «Load Balancing and Replication», Peer-to-Peer Computing, 2010, pp 127-156

[21] 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

[22] S. Khan et al. « Robust CDN Replica Placement Techniques », IPDPS 2009

[23] S. Khan et al. «A Pure Nash Equilibrium-Based Game Theoretical Method for Data Replication across Multiple Servers», in TKDE 2009.

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

[25] Uroš Čibej, Anthony Sulistio, Rajkumar Buyya, « Grid Computing», Parallel Computing, 2009, pp 117-145