Load Balancing Strategy in Grid Environment

Belabbas Yagoubi
Department of Computer Science, Faculty of Sciences, University of Oran, 31000 Oran, Algeria
byagoubi@yahoo.fr

Yahya Slimani
Department of Computer Science, Faculty of Sciences of Tunis, 1060 Tunis, Tunisia
yahya.slimani@fst.rnu.tn

Abstract

In the provision of a Grid service, a provider may have heterogeneous clusters of resources offering a variety of services to widely distributed user communities. Within such a provision of services, it will be desirable that the clusters will be hosted in a cost effective manner. Hence, an efficient structure of the available resources should be decided upon these clusters. A static structure, adopted in classical distributed systems, where a single master node controls all resources and decides where incoming jobs should be executed, is not efficient for Grid computing. For this purpose, we propose a dynamic tree-based model to represent Grid architecture in order to manage workload. This model is characterized by three main features: (i) it is hierarchical; (ii) it supports heterogeneity and scalability; and, (iii) it is totally independent from any physical Grid architecture. Over the proposed model, we develop a load balancing strategy suitable for large scale, dynamic and heterogeneous environments. The proposed strategy is based on a neighbourhood load balancing whose goal is to decrease the amount of messages exchanged between Grid resources. As a consequence, the communication overhead induced by task transfer and workload information flow is reduced, leading to a high improvement in the global throughput of a Grid. The first experiment results of our strategy are very promising. In effect, we have obtained a significant improvement of the mean response time with a reduction of the communication cost.

Keywords: Grid computing, Load balancing, Workload, Tree-based model, Hierarchical strategy, Heuristics.

1. INTRODUCTION

Decrease in powerful computers costs coupled with advances in computer networking technologies have led to increased interest in the use of large-scale parallel and distributed computing systems. Indeed, recent researches on computing architectures have allowed the emergence of a new computing paradigm known as Grid computing [4, 8]. A computational Grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities [9]. This technology is a type of distributed system which supports the sharing and coordinated use of resources, independently from their physical type and location, in dynamic virtual organizations that share the same goal [10].

One of the biggest issues in such systems is the development of effective algorithms for the distribution of the workload of a parallel or distributed program on multiple hosts to achieve some predefined goal(s) such as minimizing response time, reducing communication cost, maximizing resource utilization and throughput.

The development of Grid architectures and the associated middleware has been actively pursued in recent years to deal with the emergence of greedy applications composed of large computing tasks and amounts of data in several areas such as cosmology, molecular biology, nanomaterials, etc. [7, 10]. There are many potential advantages to use Grid architectures, including the ability to simulate applications of which computational requirements exceed local resources, and the reduction of job turnaround time through workload balancing across multiple computing facilities [9]. However, it would be inaccurate to say that the computing power of this architecture increases proportionally with the number of resources involved. Care should be taken so that some resources do not become overloaded and some others remain idle [13]. In the provision of a Grid service, a provider may have heterogeneous clusters of resources offering a variety of services to widely distributed user communities. Within such a provision of services, it will be desirable that the clusters will be hosted in a cost effective manner. Hence, an efficient structure of the available resources should be decided upon these clusters. A static structure, where a single master node controls all the resources and decides where incoming jobs should be executed, is not efficient for Grids. The master node can become easily overloaded when demand is high, leading to poor processor utilization and large response times.

In order to fulfill the user expectations in terms of performance and efficiency, a Grid system needs efficient load balancing algorithms for the distribution of tasks among their available resources. Hence, an important issue of such systems is the
effective and efficient assignment of tasks, commonly referred to as load balancing problem and it is known to be NP-complete [19]. Although load balancing problem in conventional distributed systems has been intensively studied, new challenges in Grid computing still make it an interesting topic, and many research projects are interested in this problem. This is due to the characteristics of Grids and the complex nature of the problem itself.

Our main contributions in this perspective are two folds. First, we propose a tree-based model to represent any Grid architecture into a tree structure. Second, we develop a load balancing strategy which privileges local load balancing than global ones over the whole Grid. The proposed strategy is based on a neighbourhood load balancing whose goal is to reduce the amount of messages exchanged between Grid resources during a load balancing operation. As consequence, the communication overhead induced by tasks transfer and workload information flow is also reduced.

The remainder of this paper is organized as follows. Section 2 reviews the load balancing problem. Some related works are described in Section 3. The mapping of any Grid architecture into a tree-based model is explained in Section 4. Section 5 introduces the main steps of the load balancing strategy developed over the tree model. Load balancing algorithms related to our strategy are depicted in Section 6. Section 7 discusses the performance of the proposed strategy through some experimentation. Finally, the main conclusions of our research work are summarized in Section 8 and previews of future research are also highlighted.

2. LOAD BALANCING PROBLEM

2.1 Background

A typical distributed system involves a large number of geographically distributed worker nodes which can be interconnected and effectively utilized in order to achieve performances not ordinarily attainable on a single node. Each worker node possesses an initial load, which represents an amount of work to be performed, and may have a different processing capacity. To minimize the time needed to perform all tasks, the workload has to be evenly distributed over all nodes which are based on their processing capabilities. This is why load balancing is needed. The load balancing problem is closely related to scheduling and resource allocation. It is concerned with all techniques allowing an evenly distribution of the workload among the available resources in a system [13]. The main objective of a load balancing consists primarily to optimize the average response time of applications, which often means to maintain the workload proportionally equivalent on the whole resources of a system. Load balancing is usually described in the literature as either load balancing or load sharing. These terms are often used interchangeably, but can also attract quite distinct definitions.

In the following we distinguish between three forms of load balancing.

- **Load Sharing**: This is the coarsest form of load distribution. Load may only be placed on idle resources, and can be viewed as a binary problem, where a resource is either idle or busy.

- **Load Balancing**: Where load sharing is the coarsest form of load distribution, load balancing is the finest. Load balancing attempts to ensure that the workload on each resource is within a small degree, or balance criterion, of the workload present on every other resource in the system.

- **Load Levelling**: Load levelling introduces a third category of load balancing to describe the middle ground between the two extremes of load sharing and load balancing. Rather than trying to obtain a strictly even distribution of load across all resources, or simply utilizing idle resources, load levelling seeks to avoid congestion on any resource.

Typically, a load balancing algorithm involves four policies [19]: information, location, selection and transference policies that we describe as follows:

A) **Information policy**: This policy is responsible for defining when and how the information on resources availability is updated on the Grid.

- What information of resource state which should be collected? This determines the type of information that makes a load index (queue CPU length, memory size, etc.).

- When will this information be collected? It defines the method by which information, such as the load on a resource, is communicated between the resources.

- From where will it be collected?

B) **Location policy**: This policy is responsible for finding a suitable transfer partner (server or receiver), once the transference policy has decided that a resource is an overloaded (server) or under loaded (receiver).

C) **Selection policy**: This policy defines the task (or a set of tasks) that should be transferred from the busiest resource to the idler one.

D) **Transfer policy**: This policy classifies a resource as task server or receiver according to its
availability status. It can be simple as a threshold based on the local load, or more complex including for example security issue such as the refusal of an intended migration from untrusted server.

In practice, the effectiveness of a load balancing algorithm depends on several factors:
- Stability of the messages generated by the system.
- Heterogeneity of the resources.
- Processing cost (complexity) of the used load balancing algorithm.
- Communication cost induced by tasks transfer between overloaded and under loaded resources.

2. 2 Taxonomy of load balancing approaches

The load balancing problem is often described in the literature with different terminologies, sometimes even contradictory, making thus the qualitative analysis of the various load balancing approaches difficult. Casavant and Kuhl [6] propose a largely adopted taxonomy, because it is very complete, for scheduling and load balancing algorithms in general purpose distributed computing systems. The organization of the different load balancing schemes is shown in Figure1.

![Hierarchical taxonomy for load balancing approaches](image)

The elements of Figure1 can be explained as follows:

A) Static versus Dynamic
- Static load balancing, also known as deterministic distribution, assigns a given job to a fixed resource. Every time the system is restarted, the same binding task resource (allocation of a task to the same resource) is used without considering changes that may occur during the system lifetime. In this approach, every task comprising the application is assigned once to a resource. Thus, the placement of an application is static, and a firm estimate of the computation cost can be made in advance of the actual execution.

- Dynamic load balancing takes into account the fact that the system parameters may not be known beforehand and therefore using a fixed or static scheme will eventually produce poor results. A dynamic strategy is usually executed several times and may reassign a previously scheduled task to a new resource based on the current state of system environment [17]. The advantage of dynamic over static load balancing is that the system needs not be aware of the run-time behaviour of the application before execution. For dynamic strategies, the main problem is how to characterize the exact workload of a system, while it changes in a continuous way (adding and removing computing resources, heterogeneity of the computing resources, bandwidth variations, etc.). Dynamic strategies can be applied both for homogeneous or heterogeneous platforms with different degree of performances.

B) Distributed vs. Centralized

In dynamic load balancing, the responsibility for making global decisions may lie with one centralized location, or be shared by multiple distributed locations. The centralized strategy has the advantage of ease of implementation, but suffers from the lack of scalability, fault tolerance and the possibility of becoming a performance bottleneck. In distributed strategy, the state of resources is distributed among the nodes that are responsible for managing their own resources or allocating tasks residing in their queues to other nodes.

C) Local vs. Global

In a local load balancing, each resource polls other resources in its neighbourhood and uses local informations to decide upon a load transfer. However, in a global balancing scheme, global informations of all or part of the system are used to initiate the load balancing. This scheme requires an exchange of workload informations between system elements.

D) Cooperative vs. Non-cooperative

If a distributed load balancing mode is adopted, the next issue that should be considered is whether the nodes involved in job balancing are working cooperatively or independently. In the non-cooperative case, an individual system load balancing acts alone as autonomous entities and make the decisions regarding their own objectives independently of these decisions effects about the rest of the system.

E) Adaptive vs. Non adaptive

In an adaptive scheme, task migration decisions take into account past and current system performance and are affected by previous decisions or changes in the environment.
In the non-adaptive scheme, parameters used in balancing remain the same regardless the past behaviour of the system. Confusion may arise between dynamic and adaptive balancing. Whereas a dynamic solution takes environmental inputs into account when making its decision, an adaptive solution (which is also dynamic) takes environmental stimuli into account to modify the load balancing policy itself [3].

F) Sender/Receiver/Symmetric Initiated Balancing
Techniques of load balancing tasks in distributed systems have been divided mainly into sender-initiated, receiver-initiated, and symmetrically-initiated [1]. In sender-initiated models, the overloaded resources transfer one or more of their tasks to more under loaded resources. In receiver initiated models, under loaded resources request tasks to be sent to them from resources with higher loads. Finally, in symmetric models, both the under loaded as well as the overloaded resources can initiate load transfer (task migration).

3. RELATED WORKS

Research works on load balancing has been focused on system-level load balancing or task scheduling [2, 15]. Their main objective is to maximize the overall throughput or average response time. Most application-level load balancing approaches are oriented on application partitioning via graph algorithms. However, it does not address the issue of reducing migration cost. That is the cost entailed by load redistribution, which can consume order of magnitude more time than the actual computation of a new decomposition. Some works [18] have proposed a latency-tolerant algorithm that takes advantage of overlapping the internal data computation and incoming data communication to reduce data migration cost. Unfortunately, it requires applications to provide such a parallelism between data processing and migration, which restricts its applicability. Payli et al. [16] propose a dynamic load balancing approach to provide application level load balancing for individual parallel jobs in Grid computing environment. Agent-based approaches have been tried to provide load balancing in cluster of machines [5, 14]. In [11] Genaud et al. enhance the MPI Scatterv primitive to support master-slave load balancing by taking into consideration the optimization of computation and data distribution using a linear programming algorithm. However, this solution is limited to static load balancing. Hu et al. [12] propose an optimal data migration algorithm in diffusive dynamic load balancing through the calculation of Lagrange multiplier of the Euclidean form of transferred weight. This work can effectively minimize the data movement in homogeneous environments, but it does not consider the network heterogeneity. In particular, workload migration is critical to be considered because the wide area network performance is dynamic, changing throughout execution, instable, etc., in addition to considering the resource heterogeneity. This communication aspect is neglected in traditional application-level load balancing strategies.

Scheduling sets of computational tasks on distributed platforms is a key issue but a difficult problem. Although, as mentioned above, a large number of scheduling techniques and heuristics have been presented in the literature, most of them target only homogeneous resources. However, modern computing systems, such as the computational Grid, are most likely to be widely distributed and strongly heterogeneous. Therefore, it is essential to consider the impact of heterogeneity on the design and analysis of scheduling techniques. The traditional objective, when scheduling sets of computational tasks, is to minimize the overall execution time called makespan. However, in the context of heterogeneous distributed platforms, makespan minimization problem is in most cases NP-complete, sometimes even APX-complete [19]. When dealing with large scale systems, an absolute minimization of the total execution time is not the only objective of a load balancing strategy. We think that the communication cost, induced by load redistribution, is also a critical issue. For this purpose, we propose, in this paper, a novel load balancing strategy to address the new challenges in Grid computing. Comparatively to the existing works, the main characteristics of our strategy can be summarized as follows:

- It uses a task-level load balancing;
- It privileges a local task transfer (within a cluster) more than global transfer (task transfer between clusters);
- It reduces tasks moving across the heterogeneous resources of a Grid architecture;
- It allows performing more than one load balancing operation at the same time (load balancing operation in each cluster).

4. LOAD BALANCING MODEL

4.1 Grid topology

As a topological point of view, we regard a Grid computing as a set of clusters present in a multi-node platform. Each cluster owns a set of worker nodes and belongs to a LAN local domain (Local Area Network). Every cluster is connected to the WAN global network (World Area Network) by a switch. Figure 2 describes this topology.
4.2 Mapping a Grid into a tree-based model

The load balancing strategy proposed in this paper is based on mapping of any Grid architecture into a tree-based model. This tree, illustrated in Figure 3, is built by aggregation as follows:

- First, for each cluster we create a two level sub-tree. The leaves of this sub-tree correspond to the cluster nodes, and its root, called cluster manager, represents a virtual node associated with the cluster. Its role is to manage the cluster workload.

- Second, sub-trees corresponding to all clusters are aggregated to generate a three level sub-tree of which root is a virtual node designated as Grid manager. The final tree is denoted by C/N, where C is the number of clusters that compose the Grid and N the number of worker nodes. As illustrated in Figure 3, this tree can be instantiated into two specific trees: C/N and 1/N, depending on the values of C and N.

The mapping function generates a non cyclic connected graph where each level has specific functions.

- **Level 0**: In this first level, we have a virtual node called Grid manager, which corresponds to the tree root. This node realizes the following functions:
  
  (i) It maintains the workload information of the entire Grid.
  (ii) It decides to start a global load balancing between the Grid clusters, which we will call an Intra-Grid load balancing.
  (iii) It sends the load balancing decisions to the nodes of level 1 for execution.

- **Level 1**: Each virtual node of this level, called cluster manager, is associated with a physical Grid cluster. In our load balancing strategy, this virtual node is responsible for:
  
  (i) Maintaining the workload information relating to each one of its worker nodes.
  (ii) Estimating its associated cluster workload.
  (iii) Deciding to start a local load balancing, which we will call an Intra-Cluster load balancing.
  (iv) Sending the load balancing decisions to the worker nodes which it manages.

- **Level 2**: At this last level, we find the worker nodes of the Grid linked to their respective clusters. Each node is responsible for:
  
  (i) Maintaining its workload information.
  (ii) Sending, periodically, this information to its cluster manager.
  (iii) Performing the load balancing operations decided by its manager.

4.3 Characteristics of the proposed model

The proposed model described here is characterized by the following properties:

A) It is hierarchical: This propriety will facilitate the flow of workload information through the tree. We distinguish three types of workload information flows:

(i) Ascending flow: This flow relates to the workload information flow, to get current workload state.
use the concept of group and element. Depending on cases, a group designs either a cluster or the Grid (level 1 or level 0 in the tree). An element is a group component (worker node of level 2 or cluster of level 1). The main steps of our strategy can be summarized as follows:

A) Step 1: Estimation of the group workload
Here we are interested by the information policy to define what information reflects the workload status of the group? When is it to be collected and from where? Knowing the number of available elements under its control and their computing capabilities, each group manager estimates its own capability by performing the following actions:

- Estimates its current workload based on workload informations received periodically from its component elements.
- Computes the standard deviation over the workload index, in order to measure the deviation between its involved elements. To consider the heterogeneity between nodes capabilities, we propose to take as workload index the processing time\(^1\) denoted \(TEX = \frac{LOD}{SPD}\).
- Sends workload information to its manager.

B) Step 2: Decision making
In this step, the manager decides whether it is necessary to perform a load balancing operation or not. For this purpose it executes the two following actions:

(i) Defining the imbalance/saturation state of the group. If we consider that the standard deviation \(\sigma\) measures the average deviation between the processing times of elements and the processing time of their group, we can say that this group is in balance state when this deviation is small. Indeed, this implies that processing time of each element converges to the processing time of its group. Then, we define the imbalance and saturation states.

- **Imbalance state**: In practice, we define a balance threshold, denoted as \(\varepsilon\), from which we can say that the standard deviation tends to zero and hence the group is balanced. We propose for this purpose to define a threshold \(\varepsilon\in[0-1]\). Then, we can write the following expression: If \((\sigma \leq \varepsilon)\) Then the group is Balanced Else it is Imbalanced.

- **Saturation state**: An element can be balanced while being saturated. In this

\(^1\) We define processing time of an entity (element or group) as the ratio between workload (LOD) and capability (SPD) of this entity.
particular case, it is not useful to start an intra-group load balancing since its elements
will remain overloaded. To measure saturation, we introduce another threshold called saturation threshold, noted as $\delta$. When the current workload of a group borders its capacity, it is obvious that it is useless to balance it since all belonging components are saturated.

(ii) Group partitioning. For an imbalance state, we determine the overloaded elements (sources) and the under loaded ones (receivers), depending on the processing time of every element and relatively to processing time of the associated group.

C) Step 3: Tasks transfer
In order to transfer tasks from overloaded elements to under loaded ones, we propose the following heuristic:

(i) Evaluation of the total amount of workload "Supply", available on the receiver elements.
(ii) Computation of the total amount of workload "Demand", required by source elements.
(iii) If the supply is much lower than the demand (supply is far to satisfying the request) it is not recommended to start local load balancing. Here, we introduce a third threshold, called expectation threshold denoted as $\rho$, to measure relative deviation between supply and demand. We can then write the following expression:

\[ \text{If (Supply/Demand > } \rho) \text{ Then perform a Local load balancing Else perform a Higher level load balancing.} \]

(iv) Perform the tasks transfer regarding communication cost induced by this transfer and according to the criteria selection of tasks.

As criterion selection we propose the following:

- **Shortest process time** which transfer in first the task with shortest remaining processing time.
- **Longest process time**: transfer the task with longest remaining processing time.
- **FIFO**: transfer the first submitted task.
- **LIFO**: transfer the last submitted task.
- **Random**: choosing a task randomly.

5.3 Load Supply and demand estimation
The supply of a receiver element $E_r$ corresponds to the amount of load $X_r$ that it agrees to receive so that its processing time:

\[ \text{TEX}_r \rightarrow \text{TEX}_G, \]

\[ \text{TEX}_r = \frac{LOD_r + X_r}{SPD_r} \leq \frac{LOD_G}{SPD_G} \]

With $LOD$ represents the workload and $SPD$ the capability.

\[ X_r = \frac{LOD_G \cdot SPD_r}{SPD_G} - LOD_r \]

Thus, we estimate the total supply of receiver set $GER$ by:

\[ \text{Supply} = \sum_{E_r \in GER} X_r \]

By similar reasoning we determine the demand of a source element $E_s$ which corresponds to the load $Y_s$ that it requests to transfer so that:

\[ \text{TEX}_s \rightarrow \text{TEX}_G, \]

\[ \text{TEX}_s = \frac{LOD_s - Y_s}{SPD_s} \leq \frac{LOD_G}{SPD_G} \]

\[ Y_s = LOD_s - \frac{LOD_G \cdot SPD_s}{SPD_G} \]

The total demand of source set $GES$ is obtained by:

\[ \text{Demand} = \sum_{E_s \in GES} Y_s \]

6. LOAD BALANCING ALGORITHM
We define two levels of load balancing algorithms: *intra-cluster* and *intra-Grid* load balancing algorithm.

6.1 Intra cluster load balancing algorithm
This algorithm is considered as the kernel of our load balancing strategy. The neighbourhood load balancing used by our strategy makes us think that the imbalance situations can be resolved within a cluster. It is triggered when any cluster manager finds that there is a load imbalance between the nodes which are under its control. To do this, the cluster manager receives periodically workload information from each worker node. On the basis of these informations and the estimated balance threshold $\varepsilon$, it analyzes the current workload of the cluster. According to the result of this analysis, it decides whether to start a local balancing in the case of imbalance state, or eventually just to inform its Grid manager about its current workload.

At this level, communication costs are not taken into account in the task transfer since the worker
nodes of the same cluster are interconnected by a LAN network, of which communication cost is constant.

### 6.2 Intra Grid load balancing algorithm

This algorithm, that uses a source-initiated approach, performs a global load balancing among all clusters of the Grid. It is started in the extreme case where some cluster managers fail to locally balance their overload. Knowing the global state of each cluster, the Grid manager can evenly distribute the global overload between its clusters. Contrary to the intra-cluster level, we should consider the communication cost between clusters. A task can be transferred only if the sum of its latency in the source cluster and cost transfer is lower than its latency on the receiver cluster. This assumption will avoid making useless task migration.

### 6.3 Generic intra-group algorithm (Case of a group G)

#### Step 1: Workload Estimation

1. For Every element $E_i$ of G and according to its specific period Do
   - Sends its workload $LOD_i$ to its group manager
   Endfor

2. Upon receiving all elements workloads and according to its period the group manager performs:
   a- Computes speed $SPD_G$ and capacity $SAT_G$ of G
   b- Evaluates current load $LOD_G$ and processing time $TEX_G$ of G
   c- Computes the standard deviation $\sigma_G$ over processing times
   d- Sends workload information of G to its associated manager: in case where G is a cluster

#### Step 2: Decision Making

3. 
   - **Balance criteria**
     a. Cluster: If $(\sigma_G \leq \varepsilon)$ Then Cluster is balanced; Return EndIf
     b. Grid: If ($\#$(overloaded clusters)) $\leq$ given threshold Then Grid is in balance state; Return EndIf

4. 
   - **Saturation criteria**
     If $\left(\frac{LOD_G}{SAT_G} > \delta\right)$ Then Group G is saturated; Return EndIf

5. 
   - **Partitioning group G into overloaded (GES), under-loaded (GER) and balanced (GEN)**
     GES$\leftarrow\Phi$; GER$\leftarrow\Phi$; GEN$\leftarrow\Phi$
     For Every element $E_i$ of G Do
     - If ($E_i$ is saturated) Then GES$\leftarrow$GES$\cup$E_i /* Saturated $\Rightarrow$ Overloaded */
       Else
       Switch
       - $TEX_i > TEX_G + \sigma_G$: GES$\leftarrow$GES$\cup$E_i /* Source */
       - $TEX_i < TEX_G - \sigma_G$: GER$\leftarrow$GER$\cup$E_i /* Receiver */
       - $TEX_G - \sigma_G \leq TEX_i \leq TEX_G + \sigma_G$: GEN$\leftarrow$GEN$\cup$E_i /* Balanced */
     EndSwitch
     EndIf
     EndFor.

#### Step 3: Tasks Transfer

6. Test on Supply and Demand

\[
Supply = \sum_{E_i \in GER} \frac{LOD_G \cdot SPD_G}{SPD_G} - LOD_r
\]
\[
Demand = \sum_{E_i \in GES} LOD_G \cdot \frac{LOD_G \cdot SPD_G}{SPD_G}
\]

If $\left(\frac{Supply}{Demand} \leq \rho\right)$ Then local load balancing Fail; Return EndIf

7. Perform intra-group task transferring:
   - If (G = Cluster) then Perform Heuristic1 else Perform Heuristic2 EndIf
Heuristic 1: Intra-Cluster tasks transfer
a- Sort GES by descending order of their elements processing times.
b- Sort GER by ascending order of their elements processing times.
c- While (GES ≠ Φ AND GER ≠ Φ) Do
   For i = 1 To # (GER) Do
      (i) Sort tasks of first node belonging to GES by selection criterion,
      (ii) Transfer the higher priority task from first source node of GES to i
      (iii) Update the current workloads of receiver and source nodes,
      (iv) Update sets GES, GER and GEN,
      (v) If (GES = Φ OR GER = Φ) then Return Endif,
      (vi) Sort GES by descending order of their processing times.
   Endfor
Done.

Heuristic 2: Intra-Grid tasks transfer
1- Sort items of set GES by descending order of their processing times.
2- For Every cluster Cj of GES Do
   (i) Sort clusters Cj of GER by ascending order of inter-Clusters (Cj-Cr) WAN bandwidth sizes.
   (ii) Sort nodes of Cj by descending order of their processing times.
   (iii) While (GES ≠ Φ AND GER ≠ Φ) Do
      For i = 1 To # (GER) Do
         (a) Sort tasks of first node belonging to Cj by selection criterion and communication cost,
         (b) Transfer the higher priority task from first node of Cj to i
         (c) Update the current workloads of Cj and Cr.
         (d) Update sets GES, GER and GEN,
         (e) If (GES = Φ OR GER = Φ) Then Return Endif,
         (f) Sort GES by descending order of their processing times.
      Endfor
   Endfor
Done

7. EXPERIMENTAL STUDY
7.1 Simulation parameters
In order to evaluate the practicability and performance of our proposed strategy, we have developed a Grid simulator in Java. The main parameters used in our experimentations are summarized in the sequel:

1. Cluster parameters: These parameters give information about worker nodes and clusters such as: Number of nodes and clusters, node capabilities like speeds, capacities and so on.
2. Tasks parameters: These parameters include:
   (i) Number of tasks queued in each node;
   (ii) Task submission date;
   (iii) Number of instructions per task;
   (iv) Cumulated processing time;
   (v) Cumulated waiting time.
3. Network parameters: LAN and various WAN bandwidth sizes.
4. Performance parameters: In our experimentations we have used three parameters in order to analyze the performance of the proposed strategy: the number of clusters, their worker nodes and the number of tasks. We have focused on the following objectives relating to a set of tasks submitted during a given period: Average waiting time, Average processing time, Average response time, Average time of communication and Execution time of the load balancing Algorithm.

To evaluate the gain obtained by proposed load balancing strategy, we compute the first three metrics in two cases: before (denoted Bef in Table1) and after (Aft) execution of our load balancing algorithms.

7.2 Results
All the experiments have been performed on a 3 GHz P4 Intel Pentium with 1 GB main memory, running under Windows XP.
In order to obtain reliable results, we have reiterated the same experimentations more than ten (10) times and we have computed the average time. In our experimentations, we consider that the task distribution is done in a periodic and random way according to the uniform law. We randomly generate node capabilities and task parameters. We have set the different thresholds to follows: $\varepsilon = 0.5$;
Load Balancing Strategy in Grid Environment

$\delta = 0.8$ and $\rho = 0.75$. For the selection criterion, we have adopted the *shortest process time* which transfers the task with shortest remaining execution time. This choice is made after the experimentation of several criteria such as first task submitted, last task submitted, random task, task with shortest remaining execution time and task with longest remaining execution time.

We highlight in the sequel, only results related to response and communication times, whose we consider as main objective in a load balancing strategy.

A) *Experiments 1*

In the first set of experimentations, we have focused on the response time, the waiting time and the processing time, according to various numbers of tasks and clusters.

We have considered different numbers of clusters and we suppose that each cluster contains 50 worker nodes. For every node, we generate a random speed varying between 10 and 30 computing units per time unit. The number of instructions per task has varied between 300 and 1500 computing units.

Table 1: Gain realized by proposed strategy

<table>
<thead>
<tr>
<th># Clusters</th>
<th># Tasks</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>Bef</td>
<td>630</td>
<td>387</td>
<td>340</td>
<td>327</td>
</tr>
<tr>
<td></td>
<td>Aft</td>
<td>495</td>
<td>359</td>
<td>319</td>
<td>311</td>
</tr>
<tr>
<td></td>
<td>Gain</td>
<td>21.43%</td>
<td>07.24%</td>
<td>6.18%</td>
<td>4.89%</td>
</tr>
<tr>
<td>12000</td>
<td>Bef</td>
<td>578</td>
<td>471</td>
<td>431</td>
<td>388</td>
</tr>
<tr>
<td></td>
<td>Aft</td>
<td>509</td>
<td>413</td>
<td>380</td>
<td>376</td>
</tr>
<tr>
<td></td>
<td>Gain</td>
<td>11.94%</td>
<td>12.31%</td>
<td>11.83%</td>
<td>3.09%</td>
</tr>
<tr>
<td>14000</td>
<td>Bef</td>
<td>2099</td>
<td>1653</td>
<td>1211</td>
<td>913</td>
</tr>
<tr>
<td></td>
<td>Aft</td>
<td>1776</td>
<td>1249</td>
<td>987</td>
<td>809</td>
</tr>
<tr>
<td></td>
<td>Gain</td>
<td>15.39%</td>
<td>24.44%</td>
<td>18.50%</td>
<td>11.39%</td>
</tr>
<tr>
<td>16000</td>
<td>Bef</td>
<td>1848</td>
<td>1105</td>
<td>658</td>
<td>387</td>
</tr>
<tr>
<td></td>
<td>Aft</td>
<td>1523</td>
<td>911</td>
<td>502</td>
<td>310</td>
</tr>
<tr>
<td></td>
<td>Gain</td>
<td>17.59%</td>
<td>17.56%</td>
<td>23.71%</td>
<td>19.90%</td>
</tr>
<tr>
<td>18000</td>
<td>Bef</td>
<td>1987</td>
<td>1003</td>
<td>984</td>
<td>891</td>
</tr>
<tr>
<td></td>
<td>Aft</td>
<td>1764</td>
<td>932</td>
<td>831</td>
<td>823</td>
</tr>
<tr>
<td></td>
<td>Gain</td>
<td>11.22%</td>
<td>7.08%</td>
<td>15.55%</td>
<td>7.63%</td>
</tr>
<tr>
<td>20000</td>
<td>Bef</td>
<td>1708</td>
<td>1589</td>
<td>1211</td>
<td>1050</td>
</tr>
<tr>
<td></td>
<td>Aft</td>
<td>1597</td>
<td>1456</td>
<td>1077</td>
<td>976</td>
</tr>
<tr>
<td></td>
<td>Gain</td>
<td>6.50%</td>
<td>8.37%</td>
<td>11.07%</td>
<td>7.05%</td>
</tr>
</tbody>
</table>

Table 1 shows the variation of the average response time (in seconds) before and after execution of our load balancing strategy. We can notice the following:

- Our strategy has allowed to reduce in a very clear way the mean response time of the tasks. We obtain a gain in 100% of cases, varying between 3.09% and 24.44%. In more than 60% of cases, this gain is greater than 11%.
- The lower gains have been obtained when the number of clusters was fixed at 32 on the one hand and when the number of tasks was 10000 and 20000 on the other hand. We can justify this by the instability of the Grid state (either overloaded or idle).

Figure 4 illustrates the gain obtained by the proposed load balancing strategy for various numbers of clusters and tasks.

Figure 4: Gain according to various numbers of clusters by varying the number of tasks

- Best improvements were obtained when the Grid were in a stable state: (for $\#\text{Clusters} \in \{8, 16\}$ and for $\#\text{Tasks} \in \{14000, 16000\}$).
- In some infrequent cases, we have noted that the variation of the gain changes abruptly. We believe that this situation comes from the fact that the number of tasks and/or the number of clusters varies suddenly and generates instability in the Grid.

B) *Experiments 2*

For this second set of experimentations we have been interested in the average communication time. Results of the experiments related to this metric are gathered in Figures 5 and 6.
The variation of this metric is very sensitive to the initial distribution of the tasks and to the disparity between node capabilities (when some nodes have a high speed while others have very low capabilities).

In Figure 5, we remark that the minimum is reached for 16000 tasks and 32 clusters. This result justifies the 19.9% of gain obtained for the response time. It is explained by the fact that the random distribution of tasks has been rather equitable.

For a number of tasks equal to 20000, the curve associated with a number of clusters equal to 4 has reached its maximum. Indeed, for this value and to reach a load balancing, it is necessary to transfer more tasks because of their random distribution.

Figure 6 lets us think that the average communication time is linearly proportional to the number of tasks.

7 CONCLUSION
This paper has addressed the problem of load balancing in Grid computing. We have proposed a load balancing strategy based on a tree representation of a Grid. The model takes into account the heterogeneity of resources and it is completely independent from any physical Grid architecture. Our proposal load balancing strategy privileges local load balancing in first (load balance within clusters without WAN communication). Considering the overhead generated by the execution of load balancing system, our strategy starts a balancing operation only when it ensures that it is efficiency. To validate the proposed strategy, we have developed a Grid simulator in order to measure its performance. The first experimentation results are very promising and lead to a better load balancing between nodes of a Grid without high computing overhead. We have obtained good results especially for the average response time of tasks. In the future, we plan to integrate our load balancing strategy on known simulators in the field of Grids, such as GridSim. This will allow us to measure the effectiveness of our strategy in the existing simulators. We also intend to integrate our strategy as a service of GLOBUS middleware.

REFERENCES


