ARIMA Journal

Distributed Load Balancing Model for Grid Computing

Belabbas Yagoubi [*] — Meriem Meddeber [**]

[*] University of Oran
Department of Computer Science,Oran, Algeria
b.yagoubi@univ-oran.dz

[**] University of Mascara
Department of Computer Science,Mascara, Algeria
m.meddeber@yahoo.fr

ABSTRACT. Most of the existing load balancing strategies were interested in distributed systems which were supposed to have homogeneous resources interconnected with homogeneous and fast networks. For Grid computing, these assumptions are not realistic because of *heterogeneity*, *scalability* and *dynamicity* characteristics. For these environments the load balancing problem is then a new challenge presently for which many research projects are under way.

In this perspective, our contributions through this paper are two folds. First, we propose a distributed load balancing model which can represent any Grid topology into a forest structure. After that, we develop on this model, a load balancing strategy at two levels; its principal objectives : the reduction of average response time of tasks and their transferring cost. The proposed strategy is naturally distributed with a local decision, which allows the possibility of avoiding use of wide area communication network.

RÉSUMÉ. La plupart des stratégies d'équilibrage de charge existantes se sont intéressées à des systèmes distribués supposés avoir des ressources homogènes interconnectées à l'aide de réseaux homogènes et à hauts débits. Pour les grilles de calcul, ces hypothèses ne sont pas réalistes à cause des caractéristiques d'*hétérogénéité*, de *passage à l'échelle* et de *dynamicité*. Pour ces environnements, le problème d'équilibrage de charge constitue donc, un nouveau défi pour lequel plusieurs recherches sont actuellement investies.

Notre contribution dans cette perspective à travers ce papier est double: premièrement, nous proposons un modèle distribué d'équilibrage de charge, permettant de représenter n'importe quelle topologie de grille en une structure de forêt. Nous développons ensuite sur ce modèle, une stratégie d'équilibrage à deux niveaux ayant comme principaux objectifs la réduction du temps de réponse moyen et le coût de transfert de tâches. La stratégie proposée est de nature distribuée avec une prise de décision locale, ce qui permettra d'éviter le recours au réseau de communication à large échelle.

KEYWORDS : Load balancing, Grid computing, Distributed model, Transferring cost, Workload.

MOTS-CLÉS : Équilibrage de charge,Grilles de calcul, Modèle distribué, Coût de transfert, Charge de travail.

Reçu le 14 janvier 2010,
révisé le 26 avril 2010,
accepté le 22 septembre 2010..

Revue ARIMA, vol. 12 (2010), pp. 43-60

1. Introduction

Recent researches on computing architectures have allowed the emergence of a new computing paradigm known as *Grid computing*. A computational Grid is a hardware and software infrastructure that provides a dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities [14]. This technology is a type of distributed system which supports the sharing and coordinated use of resources, independently of their physical type and location, in dynamic virtual organizations that share the same goal [5]. The Grid allows the use of geographically widely distributed and multi-owner resources to solve large-scale applications, such as meteorological simulations, data intensive applications, research of DNA sequences, etc.[3].

The two major parties in Grid computing, namely *resource consumers* (users), who submit various applications and *resources providers*, who share their resources, usually have different motivations when they join the Grid. These incentives are presented by objective functions in scheduling. While Grid users basically deal with the performance of their applications, such as the total cost to run a particular application, resource providers usually pay more attention to the performance of their resources, such as the resource utilization in a particular period. Thus, objective functions can be classified into two categories [21]: *application-centric* and *resource-centric*.

– *Application-Centric (application-level).*
Load balancing algorithms adopting an application-centric objective function aim to optimize the performance of each individual application. Most of current Grid applications concerns are about time, such as the *makespan*.

– *Resource-Centric (system-level).*
Load balancing algorithms adopting resource-centric objective functions aim to optimize the performance of resources. Resource-centric objectives are usually related to resource utilization, for example:
(a)*Throughput* which is the ability of a resource to process a certain number of jobs in a given period;
(b) *Utilization*, which is the amount of time, a resource is busy.
For a multiprocessor resource, *utilization* differences, among other processors also describe the load balance of the system and decrease the *throughput*.

Although load balancing problem in conventional parallel and distributed systems has been intensively studied, new challenges in Grid computing still make it an interesting topic, and many research projects are under way. Load balancing algorithms in classical parallel and distributed systems, which usually run on homogeneous and dedicated resources, cannot work properly in the Grid architectures [1] because Grids have a lot of specific characteristics, like *heterogeneity*, *autonomy*, *dynamicity* and *scalability*, which make the load balancing problem more difficult.

Our main contributions deal with the *system level load balancing* and are two folds: First, we propose a distributed load balancing model, transforming any Grid topology into a forest structure. Second, we develop a two level strategy to balance the load among resources of computational Grid. Our strategy privileges local rather than global load balancing in order to achieve two main objectives:
(i) The reduction of the average response time of tasks;
(ii) The reduction of communication cost induced by the task transferring.

The rest of this paper is organized as follows: Some related works are described in Section 2. Section 3, briefly reviews the load balancing problem. The mapping of any Grid architecture into a forest is explained in Section 4. Section 5 describes the main steps of the proposed load balancing strategy. In Section 6, we present and discuss the performance of our strategy through some experimental results. Section 7 compares the proposed strategy with that described in [11]. Finally, Section 8 concludes the paper and provides some future researches.

2. Related works

Most load balancing approaches are orientated towards application partitioning via graph algorithms [8]. However, they do not address the issue of reducing migration cost, that is the cost entailed by load redistribution, which can consume much more time than the actual computation of a new decomposition.

Some works [9] have proposed a latency - tolerant algorithm that takes advantage of overlapping the computation of internal data and the communication of incoming data to reduce data migration cost. Unfortunately, it requires applications to provide such a parallelism between data processing and migration, which restricts its applicability.

Genaud et al. [6] enhanced 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.

In [7], Hu et al. 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 homogenous environments, but it does not consider the network heterogeneity.

As mentioned above, a large number of load balancing techniques and heuristics, presented in literature, 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 these characteristics on the design of load balancing techniques.

The traditional objective, when balancing sets of computational tasks, is to minimize the overall execution time called *makespan*. However, in the context of heterogeneous distributed platforms, makespan minimization problems are in most cases *NP-complete*. In addition, when dealing with large scale systems, an absolute minimization of the total execution time is not the only objective of a load balancing strategy. The communication cost, induced by load redistribution, is also a critical issue. For this purpose, Yagoubi proposes in [11], a hierarchical load balancing model as a new framework to balance computing load in a Grid. Unfortunately, the root of the proposed model can become a bottleneck.

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 are:

(i) It is a *fully distributed* strategy, which allows to solve bottleneck of the model proposed in[11];

(ii) It uses a *system-level* load balancing;

(iii) It privileges local tasks transfer than global ones, for the purpose of *reducing communication costs* induced by the task migration.

3. Load Balancing Problem

3.1. Overview

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; this often means the maintenance the workload proportionally equivalent on the whole system resources.

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.

A Grid load balancer receives applications from Grid users, selects feasible resources for these applications according to acquired information, and finally generates application-to-resource mappings , on the basis of objective functions and predicted resource performance.
Basically , a load balancing system can be generalized into four basic steps [20]:
(1) Monitoring resource load and state;
(2) Exchanging load and state information between resources;
(3) Calculating the new load distribution;
(4) Updating data movement.

3.2. Classifica ion of load balancing mechanisms

Load balancing mechanisms can be broadly categorized as centralized or decentralized, dynamic or static, and periodic or non-periodic [12].
In a centralized algorithm, there is a central scheduler which gathers all load information from the nodes and makes appropriate decisions. However, this approach is not scalable for a vast environment like the Grid. In decentralized models, there is not usually a specific node known as a server or collector. Instead, all nodes have information about some or all other nodes. This leads to a huge overhead in communication. Furthermore, this information is not very reliable because of the drastic load variation in the Grid and the need for frequent updating .

Static algorithms are not affected by the system state, as their behavior is predetermined. On the other hand, dynamic algorithms make decisions according to the system state. The state refers to certain types of information, such as the number of jobs waiting in the ready queue, the current job arrival rate, etc. Dynamic algorithms tend to have better performance than static ones. Some dynamic load balancing algorithms are adaptive; in other words, dynamic policies are modifiable as the system state changes.

3.3. Load balancing in Grid computing environments

Load balancing systems for traditional distributed environments do not work in Grid environments because the two classes of environments are radically distinct. Load balancing in Grid environments is significantly complicated by the unique characteristics of Grids:

– *Heterogeneity of the Grid resources.*
Heterogeneity exists in two categories of resources. First, networks used to interconnect these computational resources may differ significantly in terms of their bandwidth and communication protocols.
Second, computational resources may have different hardware, computer architectures, physical memory size, CPU speed and also different software, such as different operating systems, cluster management software and so on. This characteristic complicates the system workload estimation because the heterogeneous resources could not be considered uniformly.

– *Dynamicity of Grid resources.*
In traditional distributed systems, such as a cluster, the pool of resources is assumed to be fixed or stable. In the Grid, this character is not verified because of computational resources and communication networks dynamicity.
Both computational resources availability and capability will exhibit dynamic behaviour. On the one hand, new resources may join the Grid and on the other hand, some resources may become unavailable due to problems such as network failure. This causes constraints on applications such as fault tolerance. A resource that connects or disconnects must be detected and taken into account by the system.

4. Mapping a Grid into a forest-based model

4.1. Grid topology

From the topological point of view, a Grid computing is a set of clusters in a multi-nodes platform. Each cluster owns a set of worker nodes (storage and computing elements) and belongs to a local domain, i.e. a LAN (Local Area Network). Every cluster is connected to the global network or WAN (World Area Network) by a switch. Figure1 describes this topology.

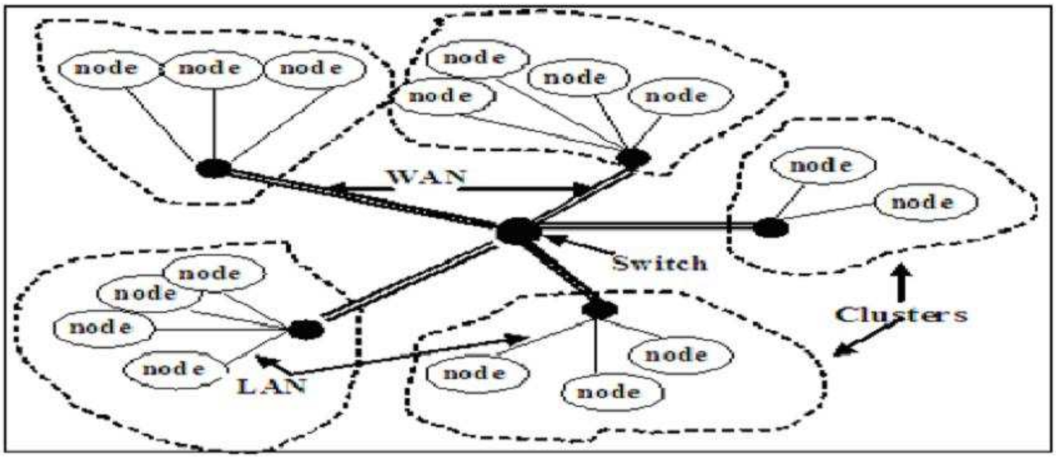

Figure 1. *Example of Grid Topology*

4.2. Mapping a Grid into a forest-based model

The load balancing strategy proposed in this paper is based on mapping of any Grid architecture into a forest structure. This forest is built by aggregation as follows:

4.3. basic model

The basic model (Figure 2) represents a logical view of the smallest possible Grid, namely only one cluster. This model, is defined by a two levels tree. The leaves of this tree correspond to the nodes of the cluster. Its root, called cluster manager, represents a virtual tree node associated with the cluster, of which role is to manage the cluster workload.

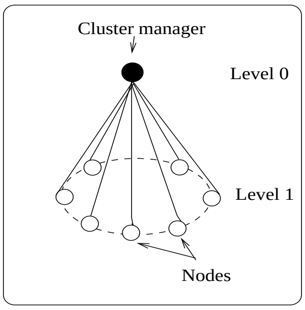

Figure 2. *Basic Model Associated to Cluster*

4.4. Generic model

As illustrated by Figure 3, the load balancing algorithm proposed in this paper is based on the mapping model of any Grid topology into a forest structure by aggregating several basic models.

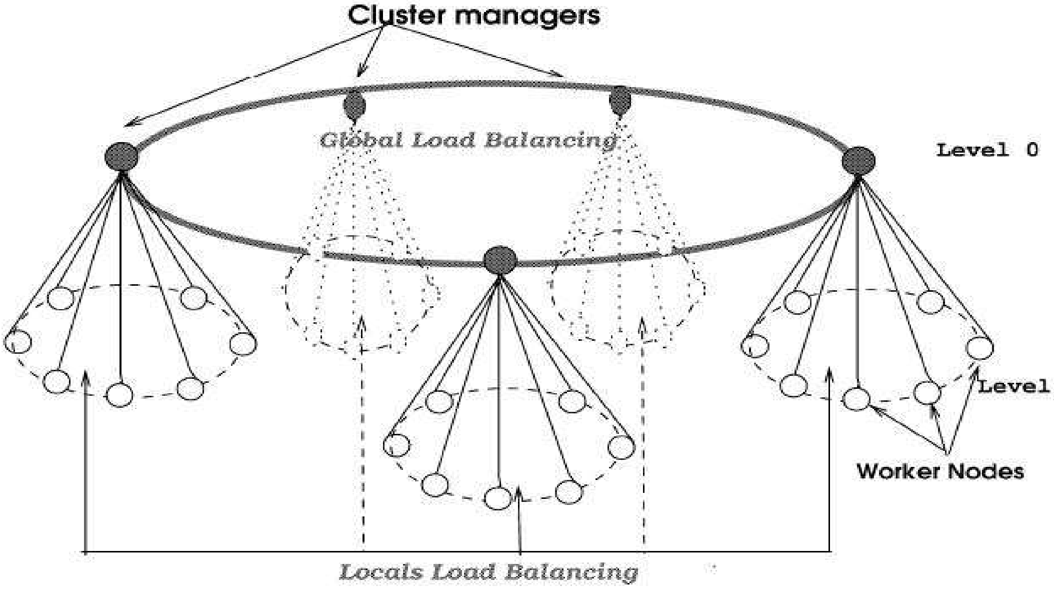

Figure 3. *Two Level Representation Model of Grid*

Level 0: Each *cluster manager* of this level is associated with a physical cluster of the Grid. In our load balancing strategy, this manager is responsible for:

– maintaining the workload information related to each one of its worker nodes;

– estimating the workload of associated cluster and diffusing this information to other cluster managers;

– deciding to start a local load balancing, which we will call ***intra-cluster*** load balancing;

– sending the load balancing decisions to the worker nodes which they manage for execution;

– initiating a global load balancing, which we will call ***inter-clusters*** load balancing.

Level 1: At this level, we find the *worker nodes* of a Grid linked to their respective clusters. Each node at this level is responsible for:

– maintaining its workload information;

– sending this information to its cluster manager;

– performing the load balancing decided by its cluster manager.

Remark.

Although we have presented the nodes the forest interconnection as virtual, each virtual node corresponds to a physical worker node of which role is to manage the load of the cluster or Grid. For example, if a cluster contains ***N*** worker nodes then (***N-1***) nodes will be used to perform the tasks and the **N***th* node (considered as a virtual node in the model) manages the load of the cluster in question. This manager can either be dedicated or have a dual role (performing tasks and managing cluster).

4.5. Model Characteristics

The proposed model is characterized by the following:

– It facilitates information flow through the Grid. We distinguish three types of information flows:

1) *Ascending flow* This flow is related to the workload information flow in order to get current workload status of Grid resources.

2) *Horizontal flow* It concerns the inputs used to perform load balancing operations.

3) *Descending flow* This flow conveys the load balancing decisions made by cluster managers to their corresponding resources.

– It supports heterogeneity and scalability of Grids: Connecting/disconnecting Grid resources correspond to adding/removing leaves or sub trees from the tree model.

– It is totally independent of any physical Grid architecture.

5. Load Balancing Strategy

In accordance with the structure of the proposed model, we distinguish between two load balancing levels: *Intra-cluster* (Inter-worker nodes) and *Inter-clusters* (Intra-Grid) *load balancing*.
We suppose that:

1) Computing elements have different characteristics (speed, period for sending load information),

2) Tasks are independent,

3) It is natural to have different communication costs between clusters, because of the heterogeneity of WANs. However, in intra-cluster communication costs are the same, since each cluster has a network (LAN) providing similar bandwidths for all its worker nodes.

5.1. Intra-cluster load balancing

Depending on its current load, each cluster manager decides to start a load balancing operation.
In this case, the cluster manager tries, in priority, to balance its workload among its worker nodes. 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. To implement this local load balancing, we propose the following three steps strategy:

Step 1: *Estimation of the current cluster workload*
Each cluster manager estimates its associated cluster capability by performing the following actions:
(i) it estimates the current workload of the cluster based on workload information received from its nodes;
(ii) it computes the standard deviation over the workload index[1] in order to measure the deviation between its involved nodes;
(iii) it sends workload information to the other cluster managers.

```
Step 1: Workload Estimation of Cluster C

1. For Every node $E_i$ of cluster C  AND according to its specific period do
       Sends its workload $LOD_i$ to its cluster manager
   end For
2. Upon receiving all node workloads and according to its period the cluster manager performs:
   a- Computes speed $SPD_C$ and capacity $SAT_C$ of cluster C
   b- Evaluates load $LOD_C$ and processing time $TEX_C$ of C
   c- Computes the standard deviation $\sigma_C$ over processing times
   d- Sends workload information of C to other cluster managers
```

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

1. Defining the imbalance/saturation state of cluster. If we consider that the standard deviation σ measures the average deviation between the processing times of worker nodes and the processing time of their cluster, we can say that this cluster is in balance state when σ is small. Then, we define the balance and saturation states as follows:

Balance state: We define a *balance threshold* ε, from which we can say that the σ tends to zero and hence the group is balanced: **If** $(\sigma \leq \varepsilon)$ **Then** the cluster is balanced, **Else** It is imbalanced.
Saturation state: A cluster can be balanced while being saturated. In this particular case, it is not useful to start an intra cluster load balancing since its nodes will remain overloaded. To measure saturation, we introduce another threshold called *saturation threshold*, denoted as δ.

2. Cluster partitioning. For an imbalance case, we determine the overloaded and the under-loaded nodes, depending on processing time of every node relatively to their associated cluster.

[1] To consider the heterogeneity between worker nodes capabilities, we propose to take as workload index the processing time denoted (TEX). We define the processing time of an entity (node or cluster) as ratio between its workload (LOD) and its capability (SPD): $TEX = \frac{LOD}{SPD}$.

Step 2: Decision Making

```
3. Balance criteria
If ($\sigma_C < \varepsilon$ ) then
    Cluster is balanced
    Return
end If
4. Saturation criteria
If ($\frac{LOD_C}{SAT_C} > \delta$) then
    cluster G is saturated
    Return
end If
5. Partitioning cluster G into overloaded (GES), under-loaded (GER) and balanced (GEN) worker nodes $GES \leftarrow \phi$; $GER \leftarrow \phi$ ; $GEN \leftarrow \phi$
For every node $E_i$ of cluster G do
    If ( $E_i$ is imbalanced) then
        $GES \leftarrow GES \cup E_i$
    else
        Switch
            $TEX_i > TEX_C + \sigma_C$: $GES \leftarrow GES \cup E_i$ /* Source */
            $TEX_i < TEX_C - \sigma_C$: $GER \leftarrow GER \cup E_i$ /* Receiver */
            $TEX_C - \sigma_C \leq TEX_i \leq TEX_C + \sigma_C$: $GEN \leftarrow GEN \cup E_i$
        end Switch
    end If
end For
```

Step 3: *Task transferring.*
To transfer tasks from overloaded to under-loaded nodes, we propose the following heuristic:
a- Evaluate the total amount of load "**Supply**", available on receiver nodes;
b- Compute the total amount of load "**Demand**", required by source nodes;
c- If the supply is much lower than the demand, it is not recommended to start local load balancing. We introduce a third threshold, called expectation threshold denoted as ρ , to measure relative deviation between supply and demand. We can then write **If** (Supply/Demand > ρ) **Then** perform Local load balancing **Else** perform Global load balancing;
d- Perform tasks transfer according to selection criteria[2].

[2] As criterion selection we propose to transfer in first the task with *longest remaining processing time.*

Intra-Cluster tasks transfer

```
6. If (Supply/Demand > $\rho$ ) then
       a- Sort GES by descending order of their nodes processing times.
       b- Sort GER by ascending order of their nodes processing times.
       c- While (GES $\neq \phi$ .AND. $GER \neq \phi$ ) do
              For i = 1 To SizeOf(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 $i^{th}$receiver
                       node of GER
                  (iii) Update the current workloads of receiver and source nodes,
                  (iv) Update sets GES, GER and GEN,
                  (v) If (GES = $\phi$ .OR. GER = $\phi$) then
                          Return
                      end If
                  (vi) Sort GES by descending order of their processing times.
              end For
          done
   else
       perform inter-cluster load balancing.
   end If
```

Load Supply and Demand estimation

The supply of a receiver node E_r corresponds to the amount of load X_r that it agrees to receive so that its processing time: $TEX_r \epsilon [TEX_C$ - σ_C ; TEX_C + $\sigma_C]$, where σ_C corresponds to the standard deviation over the processing times of cluster associated to node E_r, and TEX_C represents the processing time of this cluster. In practice, we must reach the convergence:

$TEX_r \rightarrow TEX_C$

$TEX_r = \frac{LOD_r+X_r}{SPD_r} = \frac{LOD_C}{SPD_C}$

With LOD represents the workload and SPD the capability.

$X_r = \frac{LOD_C \cdot SPD_r}{SPD_C} - LOD_r$

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

Supply= $\sum_{Er \in GER} \frac{LOD_C \cdot SPD_r}{SPD_C} - LOD_r$

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: $TEX_s \to TEX_C$.

$TEX_s = \frac{LOD_s - Y_s}{SPD_s} = \frac{LOD_C}{SPD_C}$

$Y_s = LOD_s - \frac{LOD_C \cdot SPD_s}{SPD_C}$

The total demand of source set GES is obtained by:
Demand= $\sum_{Es \in GES} LOD_s - \frac{LOD_C \cdot SPD_s}{SPD_C}$

5.2. Inter-clusters load balancing

The load balancing at this level is used if a cluster manager fails to balance its workload among its associated nodes. If we have a such case, each overloaded cluster manager transfers tasks from its overloaded worker nodes to under loaded clusters. In contrast to the intra-cluster level, we should consider the communication cost among clusters. Knowing the global state of each cluster, the overloaded cluster manager can distribute its overloaded tasks between under-loaded clusters. The chosen under-loaded clusters are those that need minimal communication cost for transferring tasks from overloaded 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.
We associate a period to each cluster manager, during which each cluster manager sends its current workload information to the other clusters.So, a cluster manager CM_i can receive new workload information about another one at any time. This updated information will be considered at the next period of CM_i.

Inter-clusters load balancing algorithm

```
1. For Each cluster manager $CM_i$ AND according to its specific period do
      a- Computes speed $SPD_i$ , capacity $SAT_i$ and processing time $TEX_i$ of $G_i$.
      b - Sends its current workload information to all other cluster managers.
   end For
2. $CM_i$ calculates the load LOD, the execution time TEX and the standard deviation $\sigma_G$ over processing times of all clusters.
3. If ($TEX_i$ > TEX + $\sigma_G$) then  Cluster is overloaded
   a- $CM_i$ determines the set of the under loaded clusters $GER$;
   b- $CM_i$ Compute its "Demand";
   end If
4. While (($demand \neq 0$ And $GER \neq \Phi$)) do
      (a) Sort the clusters $C_r$ of $GER$ by ascending order of inter clusters ($C_i$-$C_r$) WAN bandwidth sizes.
      (b) Sort the nodes of $C_i$ by descending order of their processing times.
      (c) Sort tasks of first node of $C_i$ by selection criterion and communication cost,
      (d) Transfer the higher priority task from the first node of $C_i$ to $j^{th}$ cluster of GER,
      (e) Update locally the current workloads of receiver cluster,
      (f) Update locally set GER, and the $C_i$ demand
   done
```

5.3. Strategy characteristics

The proposed strategy presents the following features:

1) *It is an* **asynchronous** *strategy:* every worker node has its *specific* period during which it sends its information workload to the other nodes.

2) *It is a* ***distributed*** *load balancing strategy:* insofar as several operations of load balancing can be started by using the local load information of Grid clusters.

3) *It is a* ***source initiative*** *load balancing strategy:* inter-clusters load balancing is started by the overloaded clusters (source).

4) *It* ***reduces the communication cost****:* since it gives the privilege, when it is possible, to a local load balancing to avoid the use of large scale communication network. In addition, the proposed strategy starts a load balancing operation only if it is beneficial.

6. Experimental study

6.1. Simulation tools

There are some available tools for application scheduling simulation in Grid computing environments, such as:

– Bricks [15]: focuses on simulating scheduling policies over the Computational Grid. It is a multiple clients and multiple servers scenarios simulator that returns average overall service rates. The network is structured as stand alone simulating entity between a Grid Computing Client and a Grid Server, without complex topology implementation.

– OptorSim [16]: has explicitly accounts for both dynamic provisioning and resource scheduling policies, obtaining resulting performance. Dynamic provisioning is performed in the context of replication in data Grid. Various economic market model replication strategies can be evaluated with the canonical reference.

– GangSim [17]: developed to support studies of scheduling strategies in Grid environments, with a particular focus on interactions of local and global resource allocation policies. It is derived as part of the Ganglia monitoring framework, an implementation that mix simulated and real Grid components. It is the first simulator that model not only sites but also Virtual Organizations(VO) users and planners.

– SimGrid [18]: focuses on scheduling to model the Grid network topology, and simulate the data flow over the available network bandwidth. The mayor drawback of this tool is that is not modeling job decomposition and task Parallelization characteristics, or resource availability is also not modeled.

– GridSim [2]: toolkit analyzing and comparing the performance of resource scheduling algorithms in Grids. There is not Data Grid simulation, only processing is implemented on variants of the Nimrod-G [19] resource broker based on the market based economic model. GridSim is the shape of heterogeneous, multitasking Grid resources, calendar based on deadline and budget based constraints. It also allows to evaluate various scheduling policies in a single simulation. Each Grid task is implemented as a separate thread in the Java Virtual Machine.

In order to evaluate the practicability and performance of our proposed strategy, we have implemented it on the **GridSim V4.0** simulator [2], which we have extended to support simulation of varying Grid load balancing problems. GridSim is a Java-based discrete-event Grid simulation toolkit. It provides a comprehensive facility for simulation of different classes of heterogeneous resources, users, applications, resource brokers, and schedulers. In GridSim, application tasks/jobs are modeled as Gridlet objects that contain all the information related to the job and the execution management details such as:

1) *Resources parameters:* it gives information about the worker nodes, clusters and networks. A node is characterized by its capacity, speed and networks bandwidth sizes.

2) *Tasks parameters:* it includes the number of tasks queued at every node, task submission date, number of instructions per task, cumulative processing time, cumulative waiting time and so on.

6.2. Simulation parameters

All the experiments have been performed on 3Ghz P4 Intel Pentium with 2 GB main memory, running on Linux Redhat 9.0. In order to obtain reliable results, the same experiments have been run several (more than ten) times.
The experiments were performed, on the basis of variation of several performance parameters in a Grid, namely the number of clusters, their worker nodes and the number of tasks.
We have focused on the following objectives:

– *Gain on the average waiting time:* this gain is obtained by a report from the average waiting time after balancing on the average waiting time before balancing;

– *Gain on the average execution time:* this gain is expressed as a ratio of the average execution time after balancing on the average execution time before balancing;

– *Gain on the average response time:* this gain is calculated by dividing the average response time after balancing on the average response time before balancing;

– *Number of tasks transferred between Grid nodes during the load balancing:* this is an important result in order to determine the communication costs induced by the tasks transfer.

For the needs for our experiments, we have considered that the tasks distribution is done in a periodic and random way according to a uniform law. We have randomly generated nodes capabilities and tasks parameters To evaluate the benefit of our strategy, we compute the above metrics before (denoted *Bef*) and after (*Aft*) execution of our load balancing algorithm.
After many evaluation tests, various thresholds were set to:

– *Balance threshold* ($\varepsilon = 0.5$),

– *Saturation threshold* ($\delta = 0.8$),

– *Expectation threshold* ($\rho = 0.75$).

The bandwidth between LAN clusters was defined to be 500 MIPS (Million Instructions Per Second), and the bandwidth between WAN nodes was set to be 30 MIPS to 400 MIPS.

Experiments 1: Intra-cluster load balancing
In the first set of experiments we focused results related to objective parameters, according to various numbers of tasks and worker nodes. We have taken the nodes number from 100 to 400 by step of 100. For each node we generate randomly associated speed varying between 5 and 40 MIPS. The number of tasks has been varied from 6000 to 10000 by step of 2000, with sizes generated randomly between 1000 and 200000 MI (Million of Instructions).

Table 1. *Improvement realized by intra-cluster strategy*

# Tasks	# Nodes	**100**	**200**	**300**	**400**
6000	*Bef*	6.00E+05	3.03E+04	6.44E+03	2.28E+03
	Aft	4.21E+05	1.49E+04	3.81E+03	2.11E+03
	Gain	**30%**	**51%**	**41%**	**7%**
8000	*Bef*	9.06E+05	4.86E+04	1.05E+04	3.63E+03
	Aft	6.46E+05	2.43E+04	5.76E+03	3.04E+03
	Gain	**29%**	**50%**	**45%**	**16%**
10000	*Bef*	1.30E+06	7.42E+04	1.63E+04	5.63E+03
	Aft	9.39E+05	3.93E+04	7.03E+03	4.30E+03
	Gain	**28%**	**47%**	**57%**	**24%**

Table 1 shows the variation of the average response time (in seconds) before and after execution. We can note that:

- The proposed strategy has allowed to reduce in a very clear way the mean response time of the tasks. We obtain a gain varying between 7% and 57%.
- In more than 95% of cases, this improvement is greater than 15%.
- The lower improvements were obtained when the number of nodes exceed 350. We can justify this by the Grid instability state (nodes are largely under-loaded).
- The best gains were realized when the number of nodes was between 200 and 300. In this case, we can say that our strategy is optimal because of the stability of the Grid.
- For a given tasks amount, the gain increases with the nodes number until it reaches a certain threshold (200-300 nodes) and then deteriorates. This behavior is similar to a Gaussian distribution. The threshold is the average attached to a stable state (balanced state). On both sides, we have less or more imbalanced states (underload and overload states).

Experiments 2: Inter-clusters load balancing

During these experiments , we were interested in the inter-clusters load balancing behaviors. We have considered different numbers of clusters and supposed that each cluster involves 30 worker nodes.

Figure 4 illustrates the improvement of the mean response time, obtained by our load balancing strategy.

- Except the case of 16 clusters, all the profits are higher than 10%. We consider this important behavior as being successful.
- Best improvements are obtained when the Grid is in a stable state: (For # clusters$\in \{4, 6, 8\}$).
- The lower benefits were obtained when the number of clusters were set to 16. We can justify this by the instability of the Grid state (Most nodes are underloaded or even idle).

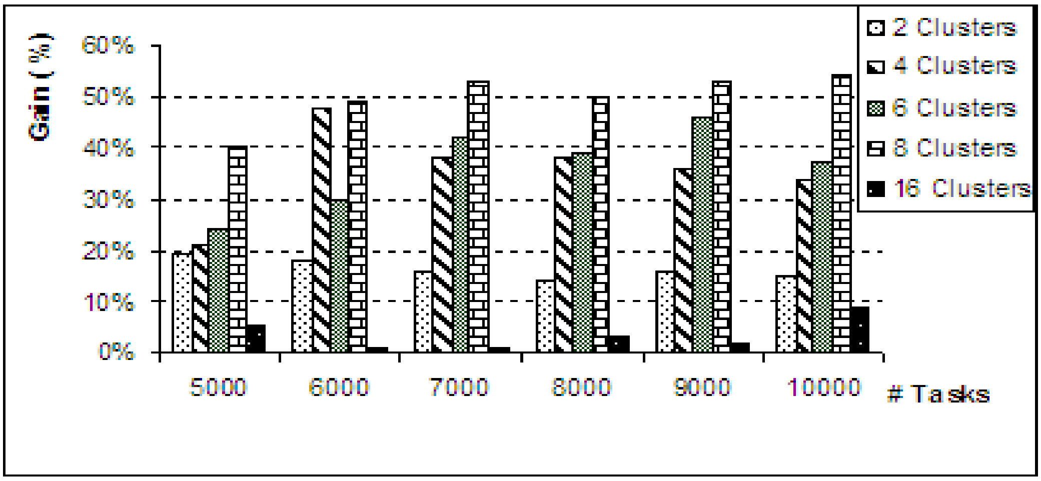

Figure 4. *Gain According to Various Number of Clusters*

Experiments 3: Number of tasks transferred
In order to show that our strategy gives the priority to a local load balancing, we are interested in this experiment to the number of tasks transferred at each load balancing level.

Table 2. *Number of tasks transferred*

# Tasks	# clusters	**2**	**4**	**8**	**16**
6000	*Global load balancing*	29	68	44	31
	Local load balancing	56	122	71	124
	Ratio	**51%**	**55%**	**61%**	**25%**
8000	*Global load balancing*	61	50	13	21
	Local load balancing	209	86	87	90
	Ratio	**29%**	**58%**	**14%**	**23%**
10000	*Global load balancing*	44	135	152	32
	Local load balancing	157	280	262	198
	Ratio	**28%**	**48%**	**58%**	**16%**

Table 2 shows that:

– The number of tasks transferred in inter-clusters algorithm is always lower than that in intra-cluster algorithm. Indeed it represents $\frac{1}{3}$ of the number of tasks transferred. This result proves that the intra-cluster transfer is more important than the inter-clusters transfer. In other words, our strategy is a local load balancing strategy which seeks to reduce the maximum communication costs,

– For a small number of clusters, the global number of tasks transferred is high. This is due to overloading clusters which cause the call of the inter-clusters load balancing requiring a task transfer between clusters. By increasing the number of clusters, the global number of tasks transferred decreases significantly, encouraging a local transfer (intra-cluster), which reinforces our goal.

7. Comparison with the hierarchical approach

The hierarchical model proposed in [11] represents a Grid computing (Figure5). In this structure, the Grid manager centralizes the global load information on all the Grid. This node, though not frequently asked, can be a bottleneck and a point of fragility in the model.

To solve this problem, we have proposed a completely distributed model by removing the root of the tree. This model aims to reduce the average response time of tasks and to minimize the communication costs.

However, the removal of the tree root creates an increase in the message number interchanged between cluster managers. Indeed, instead of sending one load informational message to the Grid manager, each cluster manager must send a message to all Grid cluster managers.

This theoretical comparison has enabled us to position the advantages and disadvantages of each approach. We will compare the results of the experiments according to the response time of each approach.

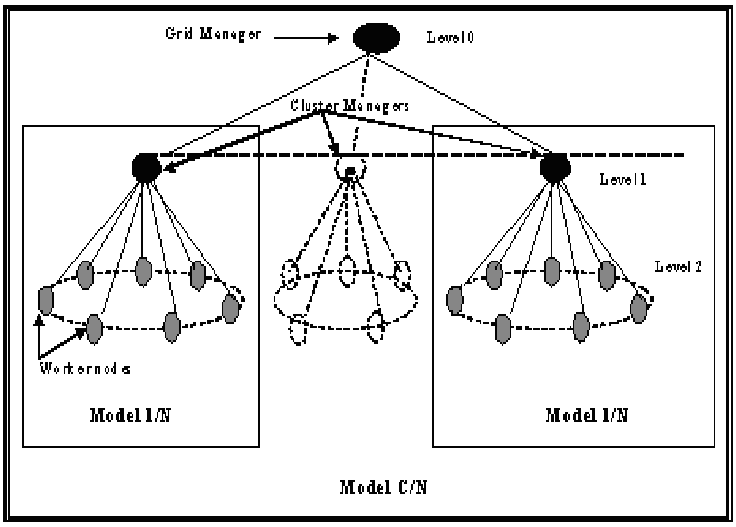

Figure 5. *Hierarchical Model*

Figure 6 shows the variation of proposed approach gains and those obtained by the hierarchical approach for 8 clusters as well as the varying number of tasks.

We note that:

– The distributed approach gains are always better than those achieved by the hierarchical approach. These results confirm our initial goals. The improvement of the gains is due to the parallel global load balancing.

– The hierarchical strategy gains increase until 6000 tasks then they decrease by increasing the tasks number. This is not the case of the proposed approach. We can say that from 6000 tasks the tree root (the Grid manager) is a bottleneck which reduces gains after that threshold.

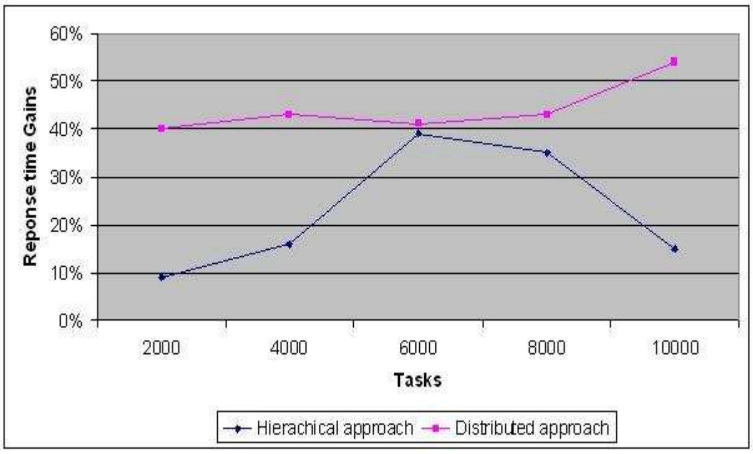

Figure 6. *Comparison with the Hierarchical Approach*

8. Conclusion

This paper addressed the problem of load balancing in Grid computing. We proposed a distributed load balancing model which takes into account the heterogeneity of the resources that is completely independent from any physical architecture Grid.
On the basis of this model, we defined a distributed load balancing strategy with two main objectives:
(i) The reduction of the mean response time of tasks submitted to a Grid computing;
(ii) The reduction of the communication costs during tasks transferring.
Relatively to the existing works, our strategy is fully distributed, uses a task-level load balancing and privileges, as much as possible, a local load balancing to avoid the use of WAN communication.
To implement the proposed strategy, we used the GridSim simulator.
The fundamental advantage of simulators is their independence from the execution platform. We can simulate a distributed system mechanism of 10000 nodes on a single PC. This advantage is made possible because the simulator does not run in a real distributed system, but in a model of this matter.
In order to validate the proposed load balancing strategy, we developed two algorithms: intra-cluster and inter-clusters.
The first results obtained were successful. We have appreciably improved the average response time with a low communication cost.

In the future, we will intent to :

– Integrate our load balancing algorithm into other known Grid simulators, such as SimGrid. This will allow us to measure the effectiveness of our algorithm in existing simulators.

– Give the place taken by the middleware GLOBUS [4] in the Grids, we think of integrating our strategy to it, in the form of services.

– Finally,it is significant to take account, in a balancing algorithm of application characteristics. More these characteristics are known, more the algorithm will be adapted and will suggest to adapt a balancing to a class of applications.

9. References

[1] BERMAN F., FOX G. , HEY Y., *"Grid Computing: Making the Global Infrastructure a Reality"*, Wiley Series in Communications Networking & Distributed Systems, 2003.

[2] BUYYA R., "*A Grid simulation toolkit for resource modelling and application scheduling for parallel and distributed computing*", www.buyya.com/gridsim/.

[3] CHERVENAK A., FOSTER I., KESSELMAN C., SALISBURY C. , TUECKE S., "The data grid: towards an architecture for the distributed management and analysis of large scientific datasets", *Jour. of Network and Computer Applications*, vol. 23, num. 3, Pages:187–200, 2000.

[4] FOSTER I., "Globus toolkit version 4: Software for service oriented systems", *IFIP: International Conference on Network and Parallel Computing*, pages 2–13, Beijing, China, November 2005.

[5] FOSTER I. , KESSELMAN C. (EDITORS), "*The Grid2: Blueprint for a New Computing Infrastructure*", Morgan Kaufmann (second edition), USA, 2004.

[6] GENAUD S., GIERSCH A. , VIVIEN F., "Load-balancing scatter operations for grid computing", *12th Heterogeneous Computing Workshop (HCW 2003)*, IEEE CS Press, 2003.

[7] HU Y.F., BLAKE R.J. , EMERSON D.R., "An optimal migration algorithm for dynamic load balancing", *Concurrency: Practice and Experience*, vol. 10, Pages 467–483, 1998.

[8] JOHANSSON H., STEENSLAND J. , "A performance characterization of load balancing algorithms for parallel SAMR applications", Tech. Report 2006-047,Department of Information Technology, Uppsala University, 2006.

[9] SHAN H., OLIKER L., BISWAS R., , SMITH W., "Scheduling in heterogeneous grid environments: The effects of data migration", *In Proc. of International Conference on Advanced Computing and Communication*, India, 2004.

[10] YAJUN L., YUHANG Y., MAODE M. AND LIANG Z."A hybrid load balancing strategy of sequential tasks for grid computing environments ", *Future Generation Computer Systems*, vol. 25, Pages 819–828, 2009.

[11] YAGOUBI B., "Modèle d'équilibrage de charge pour les grilles de calcul", *Revue Africaine de la Recherche en Informatique et Mathématiques Appliquées: ARIMA*, vol. 7, Pages 1–19, 2007.

[12] MOHSEN AMINI SALEH, HOSSEIN DELDARI AND BAHARE MOKARRAM DORRI, " Balancing Load in a Computational Grid Applying Adaptive, Intelligent Colonies of Ants,".*Informatica*, vol. 32, Pages 327–335, 2008.

[13] K.Y. KABALAN, W.W. SMAR AND J.Y. HAKIMIAN. "Adaptive load sharing in heterogeneous systems: policies, modifications and simulation", *Int. Journ. of SIMULATION*, vol. 3, num. 12, pages 89-100, 2002.

[14] I. FOSTER, C. KESSELMAN, J.M. NICK, AND S. TUECKE. "Grid services for distributed system integration". *IEEE Computer*, vol. 35 , num. 6, pages 37–46, 2002.

[15] A. TAKEFUSA AND S. MATSUOKA AND K. AIDA AND H. NAKADA AND U. NAGASHIMA. " Overview of a performance evaluation system for global computing scheduling algorithms ". *In Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing*, page 11, Washington, DC, USA, 1999.

[16] WILLIAM H. BELL, DAVID G. CAMERON, LUIGI CAPOZZA, A. PAUL MILLAR, KURT STOCKINGER, AND FLORIANO ZINI."Optorsim - a grid simulator for studying dynamic data replication strategies ". *International Journal of High Performance Computing Applications*,vol. 17, num. 4, 2003.

[17] CATALIN L. DUMITRESCU AND IAN FOSTER, "Gangsim: A simulator for grid scheduling studies,". *in In Cluster Computing and Grid*, 2005.

[18] LEGRAND, MARCHAL, AND CASANOVA, "Scheduling distributed applicaitons: The simgrid simulation framework,". *in Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid*,(CCGRID.03), 2003.

[19] BUYYA, ABRAMSON, AND GIDDY, "Nimrod/g: An architecture for a resourcemanagement and scheduling system in a global computational grid,". *in Proceedings 4th International Conference and Exhibition on High Performance Computing in Asia Pacific Region*,(HPC ASIA 2000), 2000.

[20] ZHU Y., " A survey on grid scheduling systems,". *Technical report, Department of Computer Science, Hong Kong University of science and Technology,* 2003.

[21] XU C. , LAU F.C, "*Load Balancing in Parallel Computers: Theory and Practice* ", Kluwer, Boston, MA, 1997.