Journal of Computer Science 3 (3): 186-194, 2007
ISSN 1546-9239

Task Load Balancing Strategy for Grid Computing

[1]B. Yagoubi and [2]Y. Slimani
[1]Department of Computer Science, Faculty of Sciences, University of Oran, 31000 Oran, Algeria
[2]Department of Computer Science, Faculty of Sciences of Tunis, 1060 Tunis, Tunisia

Abstract: Workload and resource management are two essential functions provided at the service level of the Grid software infrastructure. To improve the global throughput of these environments, effective and efficient load balancing algorithms are fundamentally important. Most strategies were developed in mind, assuming homogeneous set of resources linked with homogeneous and fast networks. However for computational Grids we must address main new challenges, like heterogeneity, scalability and adaptability. Our contributions in this perspective are two fold. First we propose a dynamic tree-based model to represent Grid architecture in order to manage workload. This model was characterized by three main features: (i) it was hierarchical; (ii) it supports heterogeneity and scalability; and (iii) it was totally independent from any Grid physical architecture. Second, we develop a hierarchical load balancing strategy and associated algorithms based on neighbourhood propriety. The main benefit of this idea was to decrease the amount of messages exchanged between Grid resources. As consequence, the communication overhead induced by tasks transferring and flow information was reduced. In order to evaluate the practicability and performance of our strategy we have developed a Grid simulator in Java. The first results of our experimentations were very promising. We have realized a significant improvement in mean response time with a reduction of communication cost. It means that the proposed model can lead to a better load balancing between resources without high overhead.

Keywords: Grid computing, load balancing, workload, tree-based model, hierarchical strategy

INTRODUCTION

The rapid development in computing resources has enhanced the performance of computers and reduced their costs. This availability of low cost powerful computers coupled with the popularity of the Internet and high-speed networks has led the computing environment to be mapped from distributed to Grid environments. In fact, recent researches on computing architectures are allowed the emergence of a new computing paradigm known as *Grid computing*[1]. Grid 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[2]. This technology allows the use of geographically widely distributed and multi-owner resources to solve large-scale applications like meteorological simulations, data intensive applications, research of DNA sequences and so on[3].

In order to fulfil the user expectations in terms of performance and efficiency, the Grid system needs efficient load balancing algorithms for the distribution of tasks. A load balancing algorithm attempts to improve the response time of user's submitted applications by ensuring maximal utilization of available resources. The main goal is to prevent, if possible, the condition where some processors are overloaded with a set of tasks while others are lightly loaded or even idle[4].

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 under way. This is due to the characteristics of Grid computing and the complex nature of the problem itself. Load balancing algorithms in classical distributed systems, which usually run on homogeneous and dedicated resources, cannot work well in the Grid architectures[5]. Grids has a lot of specific characteristics, like *heterogeneity*, *autonomy*, *scalability*, *adaptability* and *resources computation-data separation*, which make the load balancing problem more difficult[6].

In this study we proposed a framework consisting of distributed dynamic load balancing algorithm in perspective to minimize the average response time of

Corresponding Author: B. Yagoubi, Department of Computer Science, Faculty of Sciences, University of Oran, 31000 Oran, Algeria

applications submitted to Grid computing. Our main contributions are two fold. First 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 Grid physical architecture. Second, we develop a hierarchical load balancing strategy and associated algorithms based on neighbourhood propriety. The goal of this idea is to decrease the amount of messages exchanged between Grid resources. As consequence, the communication overhead induced by tasks transferring and flow information is reduced.

Load balancing problem

Overview: A typical distributed system will have a number of interconnected resources who can work independently or in cooperation with each other. Each resource has owner workload, which represents an amount of work to be performed and every one may have a different processing capability. To minimize the time needed to perform all tasks, the workload has to be evenly distributed over all resources based on their processing speed.

The essential objective of a load balancing consists primarily in optimizing the average response time of applications, which often means maintaining the workload proportionally equivalent on the whole resources of a system.

Conceptually, load balancing algorithms can be classified into two categories: *static* or *dynamic*[7].

* In static load balancing, a task is assigned to an available resource when it is generated or admitted to the system using a fixed schema.
* In contrast to static load balancing, dynamic load balancing allocate/reallocate tasks to resources at runtime based on no priori task information, which may determine when and whose tasks can be migrated. In this way, imbalances load can be resolved by redistributing tasks in real-time, thus solving the shortcoming of static load balancing. However, network traffic for transmitting load information to the load balancing system would increase too much due to the decision dynamicity.

Load balancing algorithms can be defined by their implementation of the following policies[8]:

* Information policy: specifies what load information to be collected, when it is to be collected and from where.
* Triggering policy: determines the appropriate moment to start a load balancing operation.
* Resource type policy: classifies a resource as server or receiver of tasks according to its availability status and capabilities.
* Location policy: uses results of the resource type policy to find a suitable partner for a server or receiver.
* Selection policy: defines tasks that should be migrated from overloaded resources to idlest ones.

Challenges of load balancing in grid: Although load balancing methods in conventional parallel and distributed systems has been intensively studied[4], they do not work in Grid architectures because these two classes of environments are radically distinct. Indeed, the schedule of tasks on multiprocessors or multi computers suppose that processors are homogeneous and linked with homogeneous and fast networks[9]. The rationale behind this approach is that:

i. The resources have same capabilities;
ii. The interconnection bandwidth between processing elements is high;
iii. Input data is readily available at the processing site;
iv. The overall time spent transferring input and output data is negligible in comparison with the total application duration.

Given the distribution of tremendous resources in a Grid environment and the size of the data to be moved, it becomes clear that this approach is not accurate because following properties[5,6]:

Heterogeneity: Heterogeneity exists in both of computational and networks resources.

* First, networks used in Grids may differ significantly in terms of their bandwidth and communication protocols.
* Second, computational resources are usually heterogeneous (processors, resource capabilities memory size and so on). Also different software's, like operating systems, file systems; cluster management software may be heterogeneous.

Autonomy: Because the multiple administrative domains that share Grid resources, a site is viewed as an autonomous computational entity. It usually has its own scheduling policy, which complicates the task allocation problem. A single overall performance goal is not feasible for a Grid system since each site has its own performance goal and scheduling decision is made independently of other sites according to its own performances.

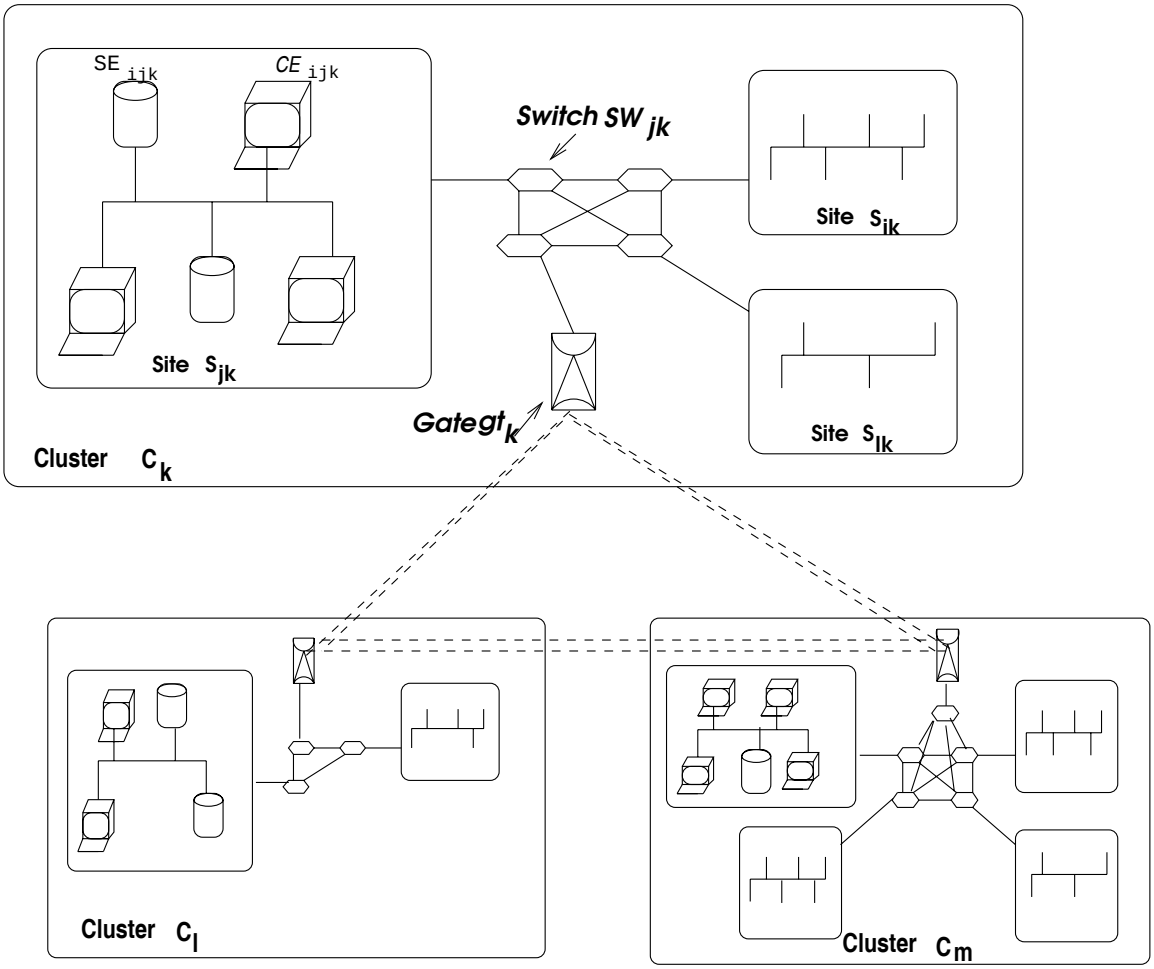

Fig. 1: Example of grid topology

Scalability and adaptability: A Grid might grow from few resources to millions. This raises the problem of potential performance degradation as the size of a Grid increases. If the pool of resources can be assumed fixed or stable in traditional parallel and distributed computing environments, in a Grid dynamicity exists in the networks and computational resources.

* First, a network shared by many execution domains cannot provide guaranteed bandwidth. This is particularly true for Wide-Area Networks like Internet.
* Second, both the availability and capability of computational resources will exhibit dynamic behaviour. On one hand new resources may join the Grid and on the other hand, some resources may become unavailable. Resource managers must tailor their behaviour dynamically so that they can extract the maximum performance from the available resources and services.

Resource selection and computation-data separation: In traditional systems, executable codes of applications and input/output data are usually in the same site, or the input sources and output destinations are determined before the submission of an application. Thus the cost for data staging can be neglected or the cost is a constant determined before execution and load balancing algorithms need not consider it. But in a Grid the computation sites of an application are usually selected by the Grid scheduler according to resource status and some performance criterion. Additionally, the communication bandwidth of the underlying network is limited and shared by a host of background loads, so the communication cost cannot be neglected. This situation brings about the computation-data separation problem: the advantage brought by selecting a computational resource that can provide low computational cost may be neutralized by its high access cost to the storage site.

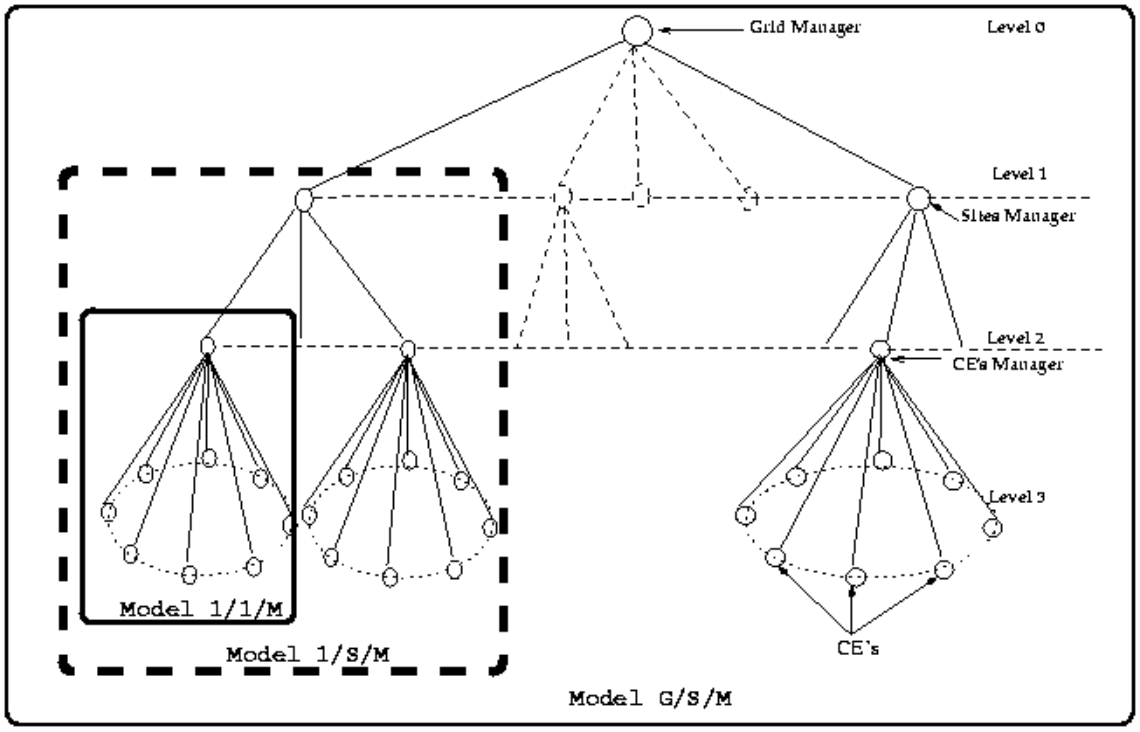

Fig. 2: Tree-based representation of a grid

These challenges pose significant obstacles on the problem of designing an efficient and effective load balancing system for Grid environments. Some problems resulting from the above are not solved successfully yet and still open research issues. Thus it is very difficult to define a load balancing system which can integrate all these factors.

Tree-based balancing model: In order to well explain the proposal model, we must define the topological structure of a Grid computing.

Grid topology: As topological point of view, we regard a Grid as a collection of G clusters C_k, connected by WAN links through gates gt_k, k ∈ {0... G-1}. Each cluster contains S sites S_{jk} interconnected via switches SWjk, j∈{0,...,S-1}. Every site involves M resources (Computing and Storage Elements) denoted CEijk and SE_{ijk}, i∈{0,...,M-1}. Resources within a site are interconnected together by a local area network.
An example of such topology is shown in Fig. 1.

Mapping a Grid into a tree-based model: The load balancing strategy proposed in this is based on a mapping of any Grid into a tree-based model. It is build as follows:

* First, for each site we create a two levels subtree. The leaves of this subtree correspond to the Computing Elements of a site and the root of this subtree is a virtual node associated to the site. The role of this virtual node is to manage the workload of a site. In practice, this management function is processed by a computing element within the site.
* Second, the subtrees corresponding to all sites of a cluster are aggregated to generate a three levels subtree.
* Third, these subtrees are connected together to build a four levels tree.

The final tree is denoted by *G/S/M*, where *G* is the number of Clusters that compose the Grid, *S* the number of Sites and *M* the number of CE's. As illustrated by Fig. 2 this generic tree can be transformed in turn into three specific trees: *G/S/M*, *1/S/M* and *1/1/M*, depending on the values of G, *S* and *M*. The mapping function generates a non cyclic connected graph where each level has specific functions.

Level 0: In this first level (top level), we have a virtual node that corresponds to the root of the tree. It is associated to the Grid and it manages the workload on the whole Grid.

Level 1: This level contains G virtual nodes, each one associated to a physical cluster of the Grid. In our load balancing strategy, this virtual node is responsible to manage its sites.

Level 2: In this third level, we find S nodes associated to physical sites of all clusters of the Grid. The main function of these nodes is to manage the workload of their physical Computing Elements.

Level 3: At this last level (leaves of the tree), we find the M Computing Elements of a Grid linked to their respective sites and clusters.

Regardless the tree model, we use the term of virtual node, but in practice each virtual node corresponds to a physical Computing Element. For example, if a site S_{jk} contains *M* Computing Elements, *M-1* CE's will be used to run tasks and the M^{th} is considered as a virtual node whose role is to manage the workload within the site S_{jk}.

Proposed model has some characteristics that we can resume as follows:

i. It is *hierarchical*: this characteristic will facilitate the workflow information flow through the tree.
ii. It supports *heterogeneity*, *autonomy* and *scalability*: adding or removing entities like (CE's, sites or clusters) correspond to simple operations (adding/removing nodes or sub trees) in our model.
iii. It is totally *independent* from any physical architecture of a Grid, because the mapping of a Grid into a tree is univocal. For each Grid corresponds one and only one tree.

Load balancing strategy

Principles: In accordance with the structure of proposed model, we develop a hierarchical load balancing strategy. We distinguish between three load balancing levels: *Intra-site* (Inter-CE's), *Intra-cluster* (Inter-sites) and *Intra-Grid* (Inter-clusters).

1. Intra-site load balancing: In this first level, depending on its current load, each CE's manager decides to start a load balancing operation. In this case, the *CE's manager* tries in priority, to load balance its workload among its computing elements. Hence, we can proceed *S* local load balancing in parallel, where *S* is the number of sites.

2. Intra-cluster load balancing: In this second level, load balance concerns clusters C_k, for which some owner *CE's managers* fail to achieve a local load balance. In this case, the *sites manager* transfers tasks from overloaded sites to under loaded ones.

Intra-Site Load Balancing Algorithm: Case of a site S_{jk}

Inputs

a- Balance threshold ε, saturation threshold δ and expectation threshold ρ.
b- Workload informations about each computing element CE_{ijk} of S_{jk}.
For Every CE_{ijk} **AND** according to its specific period PER_{ijk} **do**
  sends its current workload LOD_{ijk} to its manager S_{jk}
end For

Begin

1. CE's manager computes the speed SPD_{jk} and the capacity SAT_{jk} of S_{jk}.
2. According to its own period PER_{jk} the manager :
- Evaluates current load LOD_{jk} and processing time TEX_{jk} of S_{jk}
- Sends workload information of S_{jk} to its associated manager
3. In saturation case , it is useless to seek a local load balancing
If $(\frac{LOD_{jk}}{SAT_{jk}} > \delta)$ **then** Site saturated; Perform intra-cluster load balancing **end If**
4. Compute the standard deviation $\sigma_{jk} = \sqrt{\frac{1}{N_{jk}} \cdot \sum_{i=1}^{N_{jk}} (TEX_{ijk} - TEX_{jk})^2}$
5. Test of balance state
If $(\sigma_{jk} \leq \varepsilon)$ **then** Site in balance state; Return **end If**
6. *Partitioning the CE's into overloaded, under loaded and balanced*
CES← Φ; CER← Φ; CEN ← Φ
For Every CE_{ijk} de S_{jk} **do**
  If $(\frac{LOD_{ijk}}{SAT_{ijk}} > \delta)$ **then**
    $CES \leftarrow CES \cup \{CE_{ijk}\}$ /*saturation ⇒ overload*/
  else
    Switch
      $TEX_{ijk} > TEX_{jk} + \sigma_{jk}$:CES $\leftarrow CES \cup \{EC_{ijk}\}$ /*source*/
      $TEX_{ijk} < TEX_{jk} - \sigma_{jk}$:CER $\leftarrow CER \cup \{CE_{ijk}\}$ /* receiver*/
      $TEX_{jk} - \sigma_{jk} \leq TEX_{ijk} \leq TEX_{jk} + \sigma_{jk}$: $CEN \leftarrow CEN \cup \{CE_{ijk}\}$
    end Switch
  end If
end For
7. Supply and demand estimation
$Supply = \sum_{CE_{rjk} \in CER} \frac{LOD_{jk} \cdot SPD_{rjk}}{SPD_{jk}} - LOD_{rjk}$
$Demand = \sum_{CE_{sjk} \in CES} LOD_{sjk} - \frac{LOD_{jk} \cdot SPD_{sjk}}{SPD_{jk}}$
8. Test of supply and demand
If $((\frac{Supply}{Demand} \leq \rho))$ **then** Perform intra-cluster load balancing algorithm **end If**
9. **Heuristic 1**: Intra-site tasks transfer
- Sort CES by descending order of their processing times TEX_{ijk}
- Sort CER by ascending order of their processing times
While $((CES \neq \Phi$ **And** $CER \neq \Phi))$ **do**
  For i = 1 **To** # (CER) **do**
    (i)After having sorted the tasks of first CE belonging to CES by priority (selection criterion), Transfer the higher priority task from first CE of CES (CE_{sjk}) to $i^{th}CE$ of CER (CE_{rjk})
    (ii) update the current workloads of CE_{sjk} and CE_{rjk}.
    (iii) update sets CES, CER and CEN: according to new workloads of CE_{rjk} and CE_{sjk}
    (iv) **If** $(CES = \Phi$ **OR** $CER = \Phi)$ **Then** Return **Endif**
    (v) Sort CES by descending order of their processing times
  end For
done
10. **If** $((\#(CES) \neq 0))$ **then** Load balancing Fail **else** Success **end If**
End.

3. Intra-Grid load balancing: The load balance at this level is used only if some sites managers fail to load balance their workload among their associated sites. If we have such as case, tasks of overloaded clusters are transferred to under loaded clusters by the *Grid manager*.

The main advantage of this strategy is to privilege local load balancing in first (within a site, then within a cluster and finally on the whole Grid). The goal of this neighbourhood strategy is to decrease the amount of messages between sites and clusters. As consequence of this goal, the communication overhead induced by tasks transfer is reduced.

Intra-Cluster Load Balancing Algorithm: Case of a cluster C_k

Inputs

a- Balance threshold ε, saturation threshold δ and expectation threshold ρ.
b- Proportion $Site\text{-}Max$ of overloaded sites.
c- Workload informations about each site S_{jk} belonging to C_k.
For Every S_{jk} **AND** according to its specific period PER_{jk} **do** sends its current workload LOD_{jk} to its manager C_k **end For**
Begin
1.According to its own period PER_k the sites manager perform:
- Computes the speed SPD_k and the capacity SAT_k of cluster C_k.
- Evaluates current load LOD_k and processing time TEX_k of C_k
- Sends workload information of C_k to its correspondant manager
2. Test the load balancing triggering factor
If (proportion of overloaded sites $\leq Site\text{-}Max$) **then** Cluster C_k is balanced ; Return **end If**
3. In imbalance case , it is useful to test the saturation state of C_k
If ($\frac{LOD_k}{SAT_k} > \delta$) **then** Cluster saturated; Perform intra-grid balancing **end If**
4. *Partitioning the sites into overloaded, under loaded and balanced sites*
- SS $\leftarrow \Phi$; SR $\leftarrow \Phi$; SN $\leftarrow \Phi$
- Compute the standard deviation $\sigma_k = \sqrt{\frac{1}{N_k} \cdot \sum_{i=1}^{N_k} (TEX_{jk} - TEX_k)^2}$
For Every site S_{jk} of C_k **do**
  If ($\frac{LOD_{jk}}{SAT_{jk}} > \delta$) **then**
    $SS \leftarrow SS \cup \{S_{jk}\}$ /*satured site $\Rightarrow$ overload*/
  else
    Switch
      $TEX_{jk} > TEX_k + \sigma_k$: SS $\leftarrow SS \cup \{S_{jk}\}$ /*site source*/
      $TEX_{jk} < TEX_k - \sigma_k$:SR $\leftarrow SR \cup \{S_{jk}\}$ /*site receiver*/
      $TEX_k - \sigma_k \leq TEX_{jk} \leq TEX_k + \sigma_k$: SN $\leftarrow SN \cup \{S_{jk}\}$
    end Switch
  end If
end For
5. Supply and demand estimation
$Supply = \sum_{S_{rk} \in SR} \frac{LOD_k . SPD_{rk}}{SPD_k} - LOD_{rk}$
$Demand = \sum_{S_{sk} \in SS} LOD_{sk} - \frac{LOD_k . SPD_{sk}}{SPD_k}$ 6. Test of supply and demand
If ($(\frac{Supply}{Demand} \leq \rho)$) **then** Perform intra-grid load balancing ; Return **end If**
7. **Heuristic 2**: Inter-sites tasks transfer
- Sort items of SS by descending order of their processing times TEX_{jk}
For Every site S_{jk} of set SS **do**
  (i) Sort the sites S_{rk} of SR by ascending order of inter sites.
  (ii) Sort the EC's of S_{jk} by descending order of their processing times.
  (iii) **While** ($(SS \neq \Phi$ **And** $SR \neq \Phi)$) **do**
    For i = 1 **To** # (SR) **do**
      (a)After having sorted the tasks of first CE belonging to S_{jk} by selection criterion and communication cost, Transfer the higher priority task from first CE of S_{jk} to i^{th} site of SR (S_{rk})
      (b) update the current workloads of S_{jk} and S_{rk}.
      (c) update sets SS, SR and SN
      (d) **If** ($SS = \Phi$ **OR** $SR = \Phi$) **Then Return Endif**
      (e) Sort SS by descending order of their processing times
    end For
  done
end For
8. **If** ($(\#(SS) \neq 0)$) **then** Load balancing Fail **else** Success **end If**
End.

Generic strategy: At any load balancing level, we propose the following strategy:

1. *Estimate the current workload of a site, a cluster or a Grid:* Here we are interested by the information policy to define what information reflects the workload status of site/cluster/Grid, when it is to be collected and from where. Knowing the number of available elements under his control and their computing capabilities, each group manager estimates its own capability and performs the following actions:

i. Estimates current group workload based on workload information received periodically from its elements.
ii. Computes the standard deviation over the workload index in order to measure the deviations between its involved elements.
iii. Sends workload information to its manager.

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. Determines the imbalance/saturation state.

If we consider that the standard deviation measures the average deviation between the *processing time* of an element and the processing time of its group (Site/Cluster/Grid), we can say that this element 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.

In practice, we define a *balance threshold*, noted ε, from which we can say that the standard deviation tends to zero and hence the element is balanced.

An element can be balanced while being saturated. In this particular case, it is not useful to start an intra Site/Cluster/Grid load balancing since CE's / Sites / Clusters will remain overloaded. To measure saturation we introduce another threshold called *saturation threshold* noted by δ. When the current workload of the element borders its capacity, it is obvious that it is useless to balance since all belonging components are saturated.

ii. Partitioning. For an imbalance case, we determine the overloaded elements (*sources*) and the under-loaded ones (*receivers*), depending on processing time of every element relatively to average processing time of the associated group.

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

a. Evaluate the total amount of load: "*Supply*", available on receiver elements.
b. Compute the total amount of load: "*Demand*", required by source elements
c. 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. We introduce a third threshold, called *expectation threshold* ρ , to measure relative deviation between supply and demand.
d. Otherwise performs tasks transfer regarding communication cost induced by this transfer and according to criteria selection.

Load balancing algorithm: We define three levels of load balancing algorithms: *intra-site*, *intra-cluster* and *intra-Grid* load balancing algorithm.

Each algorithm is triggered when the group manager notes that there is a load imbalance between the elements which are under its control. To do this report, the group manager receives, in a periodic way, workload information from each element. Based on this information and on estimated *balance threshold* □, it analyzes the current load of the group. According to the result of this analysis, it decides whether to start a local balancing in the case of imbalance state, or just to inform its manager of the higher level about his current load.

Notations: Given a Grid computing whose topology is illustrated by Fig. 1, we use the following notations in the description of various load balancing algorithms:

1. Thresholds: Balance threshold ε, Saturation threshold δ and Expectation threshold ρ, defined above.
2. Computing element parameters:

* CE_{ijk}: i^{th} computing element of j^{th} site S_{jk} of k^{th} cluster C_k;
* SPD_{ijk}: computing capability of CE_{ijk} expressed in number of computational unities executed per time unity;
* SAT_{ijk}: Capacity of CE_{ijk}, it is the maximum number of computational unities which CE_{ijk} can queued ;
* PER_{ijk}: specific period of CE_{ijk}, during which it evaluates and sends its workload information to its CE's manager;
* LOD_{ijk}: Current workload of CE_{ijk} expressed in number of computational unities waiting to be executed on CE_{ijk}.

We define the processing time of CE_{ijk} by: $TEX_{ijk} = LOD_{ijk} / SPD_{ijk}$

3. Site parameters:

* $S_{jk:}$ j^{th} site S_{jk} of k^{th} cluster C_k;
* N_{jk}: number of available CE's of S_{jk};
* PER_{jk}: specific period of S_{jk}, during which it evaluates and sends its workload information to its sites manager;
* LB_{jk}: bandwidth of LAN connection between the CE's within S_{jk};

From this information, we define by aggregation the following parameters:

* $SPD_{jk} = \sum_{i=1}^{N_{jk}} SPD_{ijk}$ = Speed of site S_{jk};
* $SAT_{jk} = \sum_{i=1}^{N_{jk}} SAT_{ijk}$ = Capacity of S_{jk};
* $LOD_{jk} = \sum_{i=1}^{N_{jk}} LOD_{ijk}$ = Current load of S_{jk};
* $TEX_{jk} = \frac{LOD_{jk}}{SPD_{jk}}$ = Site processing time;
* $\sigma_{jk} = \sqrt{\frac{1}{N_{jk}} \cdot \sum_{i=1}^{N_{jk}} \left(TEX_{ijk} - TEX_{jk}\right)^2}$ = Deviation over the processing times of S_{jk};

4. Cluster parameters

* $C_{k:}$ k^{th} cluster of the Grid;
* N_k: number of available sites of C_k;
* PER_k: specific period of Grid manager, during which it estimates its workload information;
* LB: Various bandwidth connections inter-clusters;
* $SPD = \sum_{k=1}^{N} SPD_k$ = Speed of the Grid;
* $SAT = \sum_{k=1}^{N} SAT_k$ = Capacity of the Grid;
* $LOD = \sum_{k=1}^{N} LOD_k$ = Grid workload;
* $TEX = \frac{LOD}{SPD}$ = Grid processing time;
* $\sigma = \sqrt{\frac{1}{N} \cdot \sum_{k=1}^{N} \left(TEX_k - TEX\right)^2}$ = Deviation over the Grid processing times.

Intra site load balancing algorithm: This algorithm is considered as the kernel of our load balancing strategy. The neighbourhood idea which privilege local load balancing in first, lets us think that it is the most frequently requested level. It is executed when CE's managers find that there exists an imbalance between computer elements under their control. At this level, communication costs are not taken into account in the tasks transfer since the CE's of the same site are inter-connected by a LAN network, whose communication cost is constant. Indeed, for any destination of the migration task, we will have the same transfer cost.

Intra cluster load balancing algorithm: This algorithm, source -initiated, is executed only when some CE's managers fail to balance locally the overload computing. Knowing the global state of each own site, the sites manager can evenly distribute the global overload between its sites (tasks migration between sites of the same cluster). Contrary to the intra-site level, in this level algorithm we must take account of the communication cost between sites. A task can be transferred only if, the sum of its latency in the site source and its cost transferring, is lower than its latency on the site receiver. This precaution will avoid making useless tasks migration.

Intra Grid load balancing algorithm: This third level algorithm performs a global load balancing among all clusters of the Grid. It is started in the extreme case

where majority of the sites managers fail to locally balance their overload. This load balancing level must, as far as possible, be avoided because inter clusters communication costs are very significant. This over cost is due primarily to the strong heterogeneity of the Grid resources. This algorithm proceeds in the same way as the intra-cluster algorithm with two major differences:

1. The Grid manager does not have a higher level; thus it does not send any information about its workload.
2. It is useless to test at the end of the algorithm if load balancing is successful or not, because there are not other possible alternatives.

Experimental study: All the experiments were performed on PC Pentium IV of 3 GHz, with 1 GB RAM and running under Windows XP. In order to obtain reliable results, we reiterated the same experimentations more than ten (10) times. For every CE we generate an random speed varying between 10 and 30 computing units per time unity. Tasks were randomly submitted during a fixed period between [0 - 100sec]. Number of instructions per task varied between 300 and 1500 computing units.

Experiments 1: In this first set of experiments, we focused to the response time according to the number of tasks in one hand and then to the number of CE's in other hand. We have remarked the following:

1. In practically all cases we obtained an improvement in response time greater than 10%.
2. For a number of CE's fixed at 80 and for a number of tasks varying from 5000 to 25000 by step of 5000, the improvement varies from 10.18% to 17.41%. These results shows in a very clear way that the proposed strategy allowed to reduce in a very significant way the means response time of the tasks. The best results are obtained for a number of tasks equal to 25000, which leads us to think that our load balancing strategy is interesting if we have a large number of tasks.
3. For a number of tasks fixed at 20000 and for a number of CE's varying from 20 to 100 by step of 20, we obtain a gain between 10.39% and 27.42%. The best improvements are reached when CE's number is 60. In this context the Grid is a stable state (neither overloaded nor completely idle).

Experiments 2: During this experiment, we interested to communication time. We fixed at each time the number of CE's and we varied the number of tasks. Results of the experiments are gathered in Fig. 3.

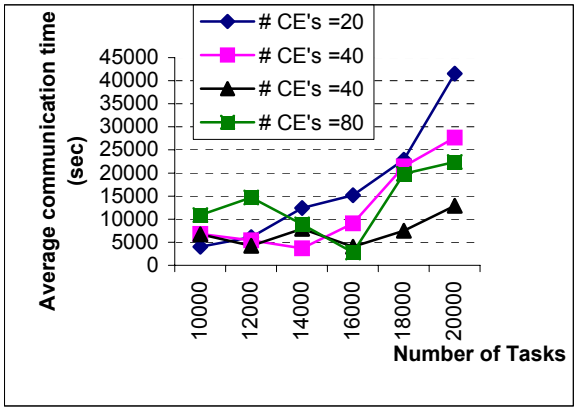

Fig. 3: Variation of communication time

The variation of this metric is very sensitive to the initial distribution of the tasks and to the number of CE's having a high speed.

For a number of tasks equal to 20000, the curve associated with a number of CE's equal to 20 reached its maximum which is normal. Indeed, for this value and to reach a balancing, it is necessary to transfer more tasks because of the randomly distribution of the tasks.

However, for the number of tasks equal to 16000, the curve associated with a number of CE's equal to 80 reached its minimum in time of communication. This result is explained by the fact that the randomly distribution of the tasks was rather equitable.

CONCLUSION

In this study, we addressed the problem of load balancing in large scale distributed systems. We proposed a load balancing strategy based on a tree representation of a Grid. The model allows transforming any Grid architecture into a unique tree with at most four levels. From this generic tree, we can derive three sub-models depending on the elements that compose a Grid. Using this model, we defined a hierarchical load balancing strategy that privileges local balancing in first (load balance within sites without communication between sites). The first results of our experimentations are very promising and lead to a better load balancing between CE's of a Grid without high computing overhead. We have appreciably improved the metrics defined, in particular average response time. In the future, we plan to integrate our load balancing strategy on known simulators in the field of the Grids, like GridSim[10]. This will allow us to measure the effectiveness of our strategy in existing simulators. We also envisage to develop our strategy as a service of GLOBUS middleware[11].

REFERENCES

1. Buyya, R., D. Abramson, J. Giddy and H. Stockinger, 2002. Economic models for resource management and scheduling in grid computing. J. Concurrency and Computation: Practice and Experience, 14: 1507-1542.
2. Foster, I., C. Kesselman and S. Tuecke, 2002. The anatomy of the Grid: Enabling scalable virtual organizations. Intl. J. High Performance Computing Applications, 15: 3.
3. Chervenak, A., I. Fosterand C. Kesselman, C. Salisbury and S. Tuecke, 2000. The data Grid: towards an architecture for the distributed management and analysis of large scientific datasets. J. Network and Computer Applications, 23: 187-200.
4. Xu, C.Z. and F.C.M. Lau, 1997. Load Balancing in Parallel Computers: Theory and Practice. Kluwer, Boston, MA.
5. Berman, F., G. Fox and Y. Hey, 2003. Grid Computing: Making the Global Infrastructure a Reality. Wiley Series in Comm. Networking & Distributed System.
6. Zhu, Y., 2003. A survey on grid scheduling systems. Technical report, Department of Computer Science, Hong Kong University of science and Technology.
7. Houle, M., A. Symnovis and D. Wood, 2002. Dimension-exchange algorithms for load balancing on trees. In Proc. of 9th Int. Colloquium on Structural Information and Communication Complexity, pp: 181-196.
8. Kabalan, K.Y., W.W. Smar and J.Y. Hakimian, 2002. Adaptive load sharing in heterogeneous systems: Policies, modifications and simulation. Intl. J. Simulation, 3: 89-100.
9. Leinberger, W., G. Karypis, V. Kumar and R. Biswas, 2000. Load balancing across nearhomogeneous multi-resource servers. In 9th Heterogeneous Computing Workshop, pp: 60-71.
10. Buyya, R., A Grid simulation toolkit for resource modelling and application scheduling for parallel and distributed computing. www.buyya.com/gridsim/.
11. Foster, I. and C. Kesselman, 1997. Globus: a metacomputing infrastructure toolkit. Intl. J. Super-Computer and High Performance Computing Applications, 11: 115-128.