World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:2, No:7, 2008

Dynamic Load Balancing Strategy for Grid Computing

Belabbas Yagoubi and Yahya Slimani

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 software environments, workloads have to be evenly scheduled among the available resources. To realize this goal several load balancing strategies and algorithms have been proposed. Most strategies were developed in mind, assuming homogeneous set of sites linked with homogeneous and fast networks. However for computational grids we must address main new issues, namely: *heterogeneity*, *scalability* and *adaptability*. In this paper, we propose a layered algorithm which achieve dynamic load balancing in grid computing. Based on a tree model, our algorithm presents the following main features: (i) it is layered; (ii) it supports heterogeneity and scalability; and, (iii) it is totally independent from any physical architecture of a grid.

Keywords— Grid Computing, Load balancing, Workload, Tree-based model.

I. Introduction

The development of computational grids and the associated middleware has been actively pursued in recent years to deal with the emergence of greedy applications of large computing tasks and amounts of data [6], [7]. Managing such applications leads to some complex problems for which traditional architectures are insufficient. There are many potential advantages of use grid architectures, including the ability to solve large scale advanced scientific and engineering applications whose computational requirements exceed local resources, and the reduction of job turnaround time through workload balancing across multiple computing facilities [1]. In his reference book, Foster [7] defined a computational grid as an emerging computing infrastructure that enables effective access to high performance computing resources. An important issue of such systems is the efficient assignment of tasks and utilization of resources, commonly referred to as load balancing problem.
The main contributions of this paper are two fold. First we propose a tree-based model to represent grid architecture with the perspective of managing workload. This model is characterized by three main features: (1) it's hierarchical in order to minimize the load balancing computation overhead; (2) the model is based on a structure using an univocal transformation of any grid architecture into a tree, with at most four levels; (3) it's totally independent from any physical grid architecture. The second contribution , the proposed load balancing algorithm which is layered in accordance with the strategy based on the above model. We seek to achieve load balancing that privilege workload neighborhood, to reduce amount of messages. Our strategy deals with a three layers algorithms (intra-site, intra-cluster and intra-grid).
The remainder of this paper is organized as follows: Section 2 describes the essential aspects of the load balancing problem. The mapping of any grid architecture into a tree-based model is explained in Section 3. Section 4 describes the main steps of our proposed load balancing strategy and outlines the layered algorithm developed over the model. Behavior and performance of our algorithm are evaluated in Section 5. Finally, Section 6 concludes the paper and gives some interesting areas for future researches.

II. Load Balancing Problem

This problem has been discussed in traditional distributed systems literature for more than two decades. Various strategies and algorithms have been proposed, implemented and classified in a number of studies [4], [13], [15]. Load balancing algorithms can be classified into two categories: static or dynamic. In static algorithms, the decisions related to load balance are made at compile time when resource requirements are estimated. A multicomputer with dynamic load balancing allocate/reallocate resources at runtime based on no a priori task information, which may determine when and whose tasks can be migrated. A good description of customized load balancing strategies for a network of workstations can be found in [15]. More recently, Houle and al. [9] consider algorithms for static load balancing on trees, assuming that the total load is fixed. Contrary to the traditional distributed systems for which a plethora of algorithms have been proposed, few of which were focussed on grid computing. This is due to the innovation and the specific characteristics of this infrastructure.
Load balancing algorithms can be defined by their implementation of the following policies [10]:

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

A. Performance parameters

The main objective of load balancing methods is to speed up the execution of applications on resources whose workload varies at run time in unpredictable way [5]. Hence, it is significant to define metrics to measure the resource workload.

Every dynamic load balancing method must estimate the timely workload information of each resource. This is a key information in a load balancing system where responses are given to following questions: (i) how to measure resource workload? (ii) what criteria are retaining to define this workload? (iii) how to avoid the negative effects of resources dynamicity on the workload; and, (iv) how to take into account the resources heterogeneity in order to obtain an instantaneous average workload representative of the system?

Several load indices have been proposed in the literature, like CPU queue length, average CPU queue length, CPU utilization, etc. The success of a load balancing algorithm depends from stability of the number of messages (small overhead), support environment, low cost update of the workload, and short mean response time which is a significant measurement for a user [2]. It is also essential to measure the communication cost induced by a load balancing operation.

B. *Problems Specific to grid computing*

The most proposed load balancing algorithms were developed in mind, assuming homogeneous set of sites linked with homogeneous and fast networks [11]. If this assumption is true in traditional distributed systems, it is not realistic in grid architectures because following properties that characterize them [3]:

- *Heterogeneity*: A Grid involves multiple resources that are heterogeneous in nature and might span numerous administrative domains across a potentially global expanse.
- *Scalability*: A Grid might grow from few resources to millions. This raises the problem of potential performance degradation as the size of a Grid increases.
- *Adaptability*: In a Grid, a resource failure is the rule, not the exception. That means that the probability of some resources fail is naturally high. Resource managers must tailor their behaviour dynamically so that they can extract the maximum performance from the available resources and services.

These properties make the load balancing problem more complex than in traditional parallel and distributed systems, which offer homogeneity and stability of their resources. Also interconnected networks on grids have very disparate performances and tasks submitted to the system can be very diversified and irregular. These various observations show that it is very difficult to define a load balancing system which can integrate all these factors.

III. Tree-based balancing model

In order to well explain our model, we first define the topological structure for a grid computing.

A. *Grid topology*

We suppose that a grid computing (see Fig. 1) is a finite set of G clusters C_k, interconnected by gates gt_k, $k \in \{0, ..., G-1\}$, where each cluster contains one or more sites S_{jk} interconnected by switches SW_{jk} and every site contains some Computing Elements CE_{ijk} and some Storage Elements SE_{ijk}, interconnected by a local area network.

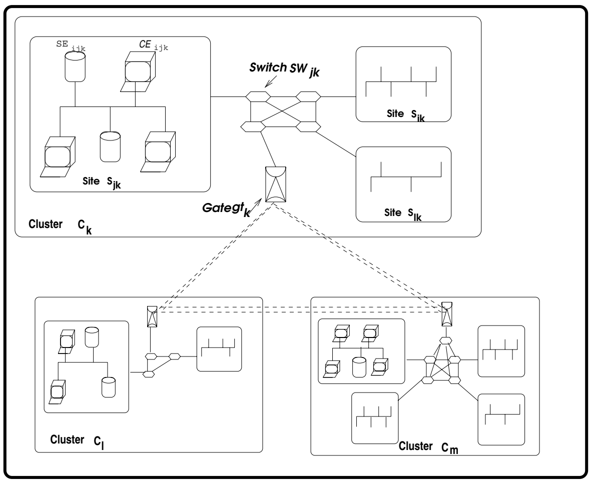

Fig. 1. Grid topology

B. *Load balancing generic model*

Our model is based on an incremental tree. First, for each site we create a two-level subtree. The leaves of this subtree correspond to the computing elements of a site, and the root is a virtual node associated to the site. These subtrees, that correspond to sites of a cluster, are then aggregated to form a three-level sub-tree. Finally, these sub-trees are connected together to generate a four-level tree called *load balancing generic model*. This model is denoted by G/S/M, where G is the number of clusters that compose a grid, S the number of sites in the grid and M the number of Computing Elements. This model can be transformed into three specific models: $G/S/M$, $1/S/M$ and $1/1/M$, depending on the values of G and S. The generic model is a non cyclic connected graph where each level has specific functions.

- ***Level 0***: In this first level (top level of the tree), we have a virtual node that corresponds to the root of the tree. It is associated to the grid and performs two main functions: (i) manage the workload information of the grid; (ii) decides, upon receiving tasks from users, where these tasks can be launched, based on the user requirements and the current load of the 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 workload of 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 the grid linked to their respective sites and clusters.

Figure 2 shows the generic tree model associated to a grid, with its three variants: $1/1/M, 1/S/M$ and $G/S/M$.

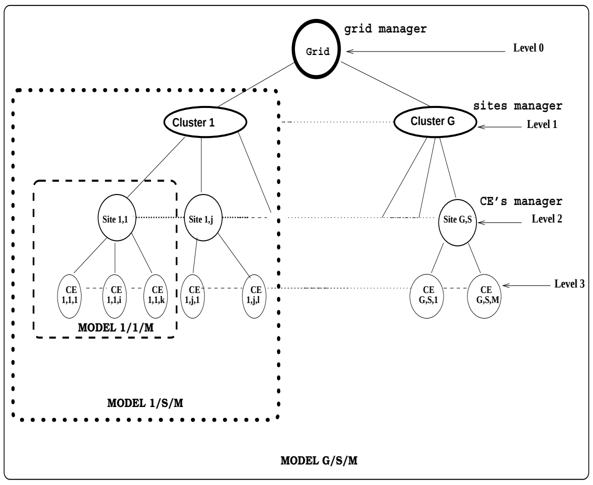

Fig. 2. Load balancing generic model

C. Characteristics of the proposed model

The main features of our proposed load balancing generic model are listed below:

1) it is *hierarchical*: this characteristic facilitate the information flow through the tree and well defines the message traffic in our strategy;
2) it supports *heterogeneity* and *scalability* of grids: adding or removing entities (computing elements, sites or clusters) are very simple operations in our model (adding or removing nodes, subtrees);
3) it is totally *independent* from any physical architecture of a grid: the transformation of a grid into a tree is an univocal transformation. Each grid corresponds to one and only one tree.

IV. Load balancing strategy

A. Principles

In accordance with the structure of proposed model, the load balancing strategy is also hierarchical. Hence, we distinguish between three load balancing levels:

1) ***Intra-site load balancing***: In this first level, depending on its current load, each site decides to start a load balancing operation. In this case, the site tries, in priority, to load balance its workload among its computing elements.
2) ***Intra-cluster load balancing***: In this second level, load balance concerns only one cluster, C_k, among the clusters of a grid. This kind of load balance is achieved only if some sites S_{jk} fail to load balance its workload among their respective computing elements. In this case, they need the assistance of its direct parent, namely cluster C_k.
3) ***Intra-grid load balancing***: The load balance at this level is used only if some clusters C_k's fail to load balance their load among their associated sites.

The main advantage of this strategy is to prioritize local load balancing first (within a site, then within a cluster and finally on the whole grid). The goal of this neighborhood strategy is to decrease the amount of messages between computing elements. As consequence of this goal, the overhead induced by our strategy is reduced. In terms of complexity, we can easily prove that the amount of messages is linear according to the number of computing elements.

B. Algorithms

According to the strategy described above, we define three levels of load balancing algorithms: *intra-site*, *intra-cluster* and *intra-grid*.

1) Notations: Notations used in the following algorithms are defined in [14] as follows:
$C_k = k^{th}$ cluster; $S_{jk} = j^{th}$ site of k^{th} cluster; $CE_{ijk} = i^{th}$ computing element of j^{th} site of k^{th} cluster; T=Tolerance factor expressed in (%); L_{ijk}=actual workload of CE_{ijk}; L_{jk}=actual workload of S_{jk}; L_k=actual workload of C_k; $Avrg$ = grid average load; $sit\text{-}max$ = parameter to start an intra-cluster load balancing; $clu\text{-}max$ = parameter to start an inter-clusters load balancing.

2) Intra-site load balancing algorithm: This algorithm is considered as the kernel of our load balancing strategy. It is executed when CE's managers find that there exists an imbalance between computer elements under their control. To make this report, the managers receive periodically local workload information from computing elements. Based on these information and on the Avrg of average grid workload, the managers analyze the workload of sites periodically. Depending on the result of this analysis, either they decide to start a local load balancing between CE's of the same site, or either they inform their manager site (cluster) that are currently overloaded.

```
Case of a site $S_{jk}$

BEGIN
For Every $CE_{ijk}$ AND Periodically do
    -Update actual workload $L_{ijk}$ of $CE_{ijk}$
    -Send load information to CE's manager $S_{jk}$
end For
- Update current load $L_{jk}$ of site $S_{jk}$
- Send load information to sites manager $C_k$
- Receive grid average load from $C_k$
If ($\frac{|L_{jk} - Avrg|}{L_{jk}} > T$) then
    imbalance state
else
    return
end If
```

```
[Partition CE's of $S_jk$ into overloaded, underloaded and balanced]
CES $\leftarrow \emptyset$; CER$\leftarrow \emptyset$; CEN $\leftarrow \emptyset$
For Every $CE_{ijk}$ of $S_{jk}$ do
    Switch
        - $L_{ijk} > Avrg + T$ :
        $CES \leftarrow CES \cup \{CE_{ijk}\}$
        - $L_{ijk} < Avrg$:
        $CER \leftarrow CER \cup \{CE_{ijk}\}$
        - $Avrg \leq L_{ijk} \leq Avrg + T$:
        $CEN \leftarrow CEN \cup \{EC_{ijk}\}$
    end Switch
end For
While (CES $\neq \emptyset$  AND CER $\neq \emptyset$) do
    - Sort CES by descending order of $L_{ijk}$
    - Sort CER by ascending order of $L_{ijk}$
    - $CE_{sjk} \leftarrow$ CE most overloaded;
    - $EC_{rjk} \leftarrow$ CE most lightly overloaded
    - Load offered by $EC_{rjk}$ = $Avrg - L_{rjk}$
    - Tasks migration stage from $CE_{sjk}$ to $CE_{rjk}$
    - Update current workload of $CE_{sjk}, CE_{rjk}$
    - Update sets CES , CER and CEN
done
If ($card(CES) \geq sit\text{-}max$) then
    -Intra-Site load balancing fail
    -Call intra cluster load balancing algorithm
else
    success load balancing
end If
END.
```

3) *Intra-cluster load balancing algorithm:* This algorithm is executed only when some CE's managers fail to balance locally the overload of their CE's. Knowing the global state of each site, the sites manager can evenly distribute the global overload between its sites.

```
Case of a cluster $C_k$

BEGIN
For Every site $S_{jk}$ of $C_k$  AND Periodically do
    Update current workload $L_{jk}$ and send it to sites manager $C_k$
end For
- Update actual load $L_k$ of $C_k$
- Send it to grid manager
- Receive grid average load from grid manager
If (Overloaded sites number $\geq sit\text{-}max$) then
    $C_k$ is in imbalance state
else
    Return
end If
[Partition sites of $C_k$ into overloaded, underloaded and balanced sites]
SS $\leftarrow \emptyset$; SR $\leftarrow \emptyset$; SN $\leftarrow \emptyset$
For every $S_{jk}$ of $C_k$  do
    Switch
        -$L_{jk} > Avrg + T$:
        $SS \leftarrow SS \cup \{S_{jk}\}$
        -$L_{jk} < Avrg$ :
        $SR \leftarrow SR \cup \{S_{jk}\}$
        -$Avrg \leq L_{jk} \leq Avrg + T$:
        $SN \leftarrow SN \cup \{S_{jk}\}$
    end Switch
end For
While (SS $\neq \emptyset$  AND SR $\neq \emptyset$) do
    - Sort SS by descending order of workload $L_{jk}$
    - Sort SR by ascending order of $L_{jk}$
    - $S_{lk} \leftarrow$ most overloaded site ;
    -$S_{rk} \leftarrow$ most underloaded site
    - Load offered by $S_{rk} = Avrg - L_{rk}$
    - Tasks migration decision from $S_{lk}$ to $S_{rk}$
    - Start Intra site load balancing algorithm
done
If ($card(SS) \geq clu\text{-}max$) then
    -Intra-Cluster Load balancing fail
    -Call intra-grid load balancing algorithm
else
    success load balancing
end If
END.
```

4) *Intra-grid load balancing algorithm:* This third algorithm performs a global load balancing among all clusters of a grid. It is executed only if the other two levels are failed to achieve a complete load balance.
It is significant to remark that at this level, the algorithm always succeeds load balancing all these clusters. It is thus useless to test if some clusters are still imbalanced.

```
Case of the global grid

BEGIN
For Every $C_k$  AND Periodically do
    - Update current workload $L_k$
    - Send it to grid manager
end For
- Update global grid workload
- Compute grid average load
- Send it to all clusters
If (overloaded clusters number $\geq clu\text{-}max$) then
    Grid is imbalanced
else
    Return
end If
[Grid Partition into overloaded, underloaded and balanced clusters]
CS $\leftarrow \emptyset$; CR $\leftarrow \emptyset$; CN $\leftarrow \emptyset$
```

```
For Every $C_k$ do
    Switch
        $L_k > Avrg + T : CS \leftarrow CS \cup \{C_k\}$
        $L_k < Avrg : CR \leftarrow CR \cup \{C_k\}$
        $Avrg \leq L_k \leq Avrg+T$: $CN \leftarrow CN \cup \{C_k\}$
    end Switch
end For
While (CS $\neq \emptyset$ AND CR $\neq \emptyset$) do
    - Sort CS in decreasing order of workload $L_k$
    - Sort CR in increasing order of $L_k$
    - $C_s \leftarrow$ most overloaded cluster
    - $C_r \leftarrow$ most underloaded
    - Tasks migration stage from $C_s$ to $C_r$
    - Start intra-Cluster load balancing algorithm
done
END.
```

#CE's	Response time(sec)			Cost
	Before	After	Gain (%)	(Sec)
50	817	730	10.65	41
100	409	353	13.69	50
150	273	230	15.75	49
200	126	99	21.43	47
250	164	139	15.24	49

TABLE I
RESPONSE TIME VS NUMBER OF CE'S (NUMBER OF TASKS=2000)

#Tasks	Response time(sec)			Cost
	Before	After	Gain (%)	(Sec)
1000	113	95	15.93	22
1250	134	114	14.93	33
1500	145	122	15.86	37
1750	156	132	15.38	47
2000	164	139	15.24	49

TABLE II
RESPONSE TIME VS NUMBER OF TASKS (NUMBER OF CE'S=250)

V. EXPERIMENTAL STUDY

A. Modelling parameters

In order to evaluate the practicability and the performance of our model we have developed a grid simulator. This simulator was built in Java and uses the following parameters:

1) ***CE's parameters***: these parameters give information about available CE's during load balancing period such as: (i) number of sites; (ii) number of CE's in each site; (iii) CE's speeds; (iv) date to send workload information from CE's; and, (v) tolerance factor.
2) ***Tasks parameters***: these parameters include: (i) number of tasks queued at every CE; (ii) task submission date; (iii) number of instructions per task; (iv) task size; and, (v) priority.
3) ***Network parameter***: bandwidth size.
4) ***Workload index***: as workload for Computing Elements, we have used their *occupation ratio*: workload=$\frac{inst}{speed}$, where $inst$ denotes the total number of instructions queued on a given CE and $speed$ is its speed.
5) ***Performance parameters***: in our experimentations, we focused on two performance parameters: tasks average response time and cost communication.

B. Experimental results

All the experiments were performed on PC Pentium IV of 2.8 GHz, with a 256 MB RAM and running under Windows XP. In order to obtain reliable results, we reiterated the same experiments more than ten (10) times.
In the sequel, we will give the experimental results relating to the response time according to the number of tasks and according to the number of computing elements. The following tables (see Tables I and II) show the variation of the average response time before and after execution of the intra-site load balancing algorithm.
In Tables I and II, $Before$ and $After$ represent mean response time before and after load balancing is performed and $cost$ defines the communication cost expressed in seconds.
From these tables, we remark that our strategy leads to a good load balancing:

1) For a number of tasks fixed at 2000 and for a number of CE's varying from 50 to 250 by step of 50, we obtain a gain varying between 10.65% and 21.43%, with negligible cost of communication.
2) For a number of CE's equal to 250 and for a number of tasks varying from 1000 to 2000 by step of 250, the gain varies from 14.93% to 15.93%.
3) During our experiments, we have remarked that the best gains are obtained when the grid is in a stable state (neither overloaded nor completely idle).

Besides the decrease in the average response times, we notice a stability on the time communication and, therefore, a small amount of messages induced by neighborhood strategy. Figures 3illustrates the variation of time communication according to the number of tasks and according to the number of computing elements.

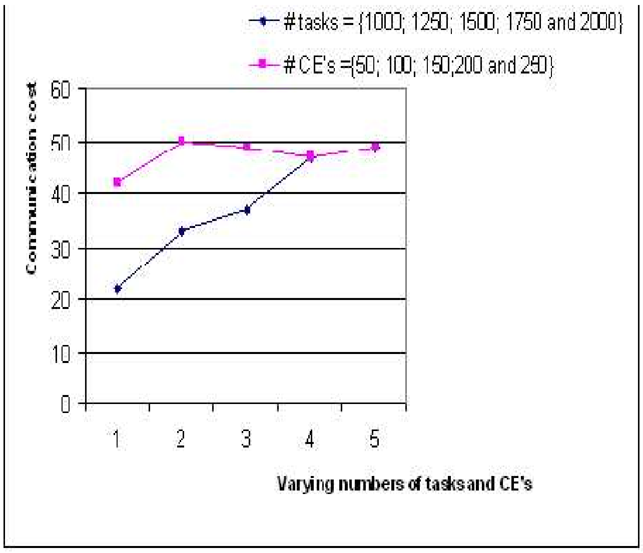

Fig. 3. Communication time Vs Number of CE's and tasks

VI. Conclusion and future works

In this paper, we addressed the problem of load balancing in grid computing. We proposed a load balancing strategy based on a tree model representation of a grid architecture. The model allows the transformation of any grid architecture into an 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: one site, one cluster, or in the general case multiple clusters. Using this model, we defined a hierarchical load balancing strategy that gives priority to local load balancing within sites. The proposed strategy leads to a layered algorithm which an prototype was implemented and evaluated on a grid simulator developed for the circumstance. The first results of our experimentations show that the proposed model can lead to a better load balancing between CE's of a grid without high overhead. We have observed that significant benefit in mean response time was realized with a reduction of communication cost between clusters.

The model presented in this paper raises a number of challenges for further researches. First, we plan to test our model on others grid simulators [8] . Second, we plan to experiment our model on real grid environments like Globus [6] and XtremWeb [12], using a realistic grid application, in order to validate the practicality of the model.

References

[1] E. Deelman A.Chervenak and al. High performance remote access to climate simulation data: a challenge problem for data grid technologies. In *Proceeding. of 22th parallel computing*, volume 29(10), pages 13–35, 1997.

[2] E. Badidi. *Architecture and services for load balancing in object distributed systems*. PhD thesis, Faculty of High Studies, University of Montreal, Mai 2000.

[3] F. Berman, G. Fox, and Y. Hey. *Grid Computing: Making the Global Infrastructure a Reality*. Wiley Series in Communications Networking & Distributed Systems, 2003.

[4] T.L. Casavant and J.G. Kuhl. A taxonomy of scheduling in general purpose distributed computing systems. *IEEE Transactions on Software Engineering*, 14(2):141–153, 1994.

[5] D.L. Eager, E.D. Lazowska, and J. Zahorjan. Adaptive load sharing in homogeneous distributed systems. In *IEEE Trans. on Soft. Eng.*, volume 12(5), pages 662–675, 1986.

[6] I. Foster and C. Kesselman. Globus: a metacomputing infrastructure toolkit. *Int. Jour. of Super-Computer and High Performance Computing Applications*, 11(2):115–128, 1997.

[7] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.

[8] GridSim. A grid simulation toolkit for resource modelling and application scheduling for parallel and distributed computing. www.buyya.com/gridsim/.

[9] M. Houle, A. Symnovis, and D. Wood. Dimension-exchange algorithms for load balancing on trees. In *Proc. of 9th Int. Colloquium on Structural Information and Communication Complexity*, pages 181–196, Andros, Greece, June 2002.

[10] H.D. Karatza. Job scheduling in heterogeneous distributed systems. *Journal. of Systems and Software*, 56:203–212, 1994.

[11] W. Leinberger, G. Karypis, V. Kumar, and R. Biswas. Load balancing across near-homogeneous multi-resource servers. In *9th Heterogeneous Computing Workshop*, pages 60–71, 2000.

[12] XtremWeb. A global computing experimental platform. http://www.lri.fr/fedak/XtremWeb/introduction.php3.

[13] C.Z. Xu and F.C.M. Lau. *Load Balancing in Parallel Computers: Theory and Practice*. Kluwer, Boston, MA, 1997.

[14] B. Yagoubi. Dynamic load balancing for beowulf clusters. In *Proceeding of the 2005 International Arab Conference On information Technology*, pages 394–401, Israa University, Jordan, December 6th 8th 2005.

[15] M.J. Zaki, W. Li, and S. Parthasarathy. Customized dynamic load balancing for a network of workstations. In *Proc. of the 5th IEEE Int. Symp. HDPC*, pages 282–291, 1996.