Copyright © 2020 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0)

Job Migration for Load Balancing Algorithm in Grid Computing Using Queue Length parameter

Ali Wided
Department of Mathematics and Computer Science
University of tebessa
Tebessa, Algeria
wided.ali@univ-tebessa.dz
aliwided1984@gmail.com

Bouakkaz Fatima
Department of Mathematics and Computer Science
University of tebessa
Tebessa, Algeria
f.bouakkaz@univ-tebessa.dz
f_bouakkez@esi.dz

CEUR-WS.org/Vol-2589/Paper10.pdf

***Abstract*— based on the previous works, we used the job migration technique for hierarchical load balancing. In this paper the authors propose a novel job migration algorithm for dynamic load balancing (JMADLB), in which parameters(such as CPU queue length) have been considered which is used for the selection of overloaded resources (or underloaded ones) in grid. . In dynamic load balancing state, the system will change dynamically until it reaches a balance. In a grid environment, efficiency of resources varies with time. Thus, the allocation of jobs must be adjusted dynamically in according with the variation of the resources status. The proposed algorithm has been verified through Alea2 simulator and the simulation results validate that the proposed algorithm allow us to reach our objectives.**

Keywords— grid computing, load Balancing, job Migration, workload, information policy, location policy, selection policy, resources Allocation.component;

I. Introduction

A computational Grid is a hardware and software infrastructure that gives dependable, consistent, pervasive, and cheap access to high-end computational capabilities [1]. The challenges in grid computing lie in load balancing. Load balancing is an important issue for the problem of utilization. It is those techniques which are designed to equally distribute the load on resources and maximize their utilization. These techniques can be approximately categorized as centralized or decentralized, dynamic or static, periodic or non-periodic [2]. Main purpose of load balancing is to enhance the response time of the application by which workload would be saved according to resources. There are causes which are the major raisons of load balancing, resubmission of jobs and job migration; heterogeneity of resources, dynamic nature of resource's performance and diversity of applications in case of Grids[3]. This is even more vital in computational Grid where the main concern is to equally allocate jobs to resources and to minimize the difference between the overloaded and the underloaded resource load [4]. Efficient load balancing through the Grid is required for improving performance of the system. The overloaded grid resources can be balanced by migrating jobs to the idle processors, i.e. a set of processors to which a processor is directly connected [4]. Our contributions are: First, we proposed a hierarchical load balancing algorithm. Second, we verified the proposed algorithm through Alea 2 simulator. The objective of proposed algorithm is enhancing the performance of application by minimizing slowdown, and the waiting time in the global queue, maximizing the resources usage rate and load balancing among the resources.

II. Related Works

The authors[5] have suggested a Priority based Dynamic Load Balancing Algorithm (PBDLB). When a node is overloaded, it calls the MSN which then finds a suitable node and then performs the load balancing, a function msn ( ) finds the available under-loaded nodes by looking into a queue where all the processors are scheduled in the decreasing order of their computing power. here CPU queue length is considered. The Job Migration strategy is used through which the migration of jobs takes place from the heavily node to the lightly-loaded node. The advantage of this algorithm is that it takes into account the resource processing capability, where the nodes with high computing power have high priority; also it decreases the communication overhead and proves to be cost real. The drawback of this study is not considering the fault tolerance.

In the study of [6] an Augmented Hierarchical Load Balancing with Intelligence Algorithm is proposed (AHLBI). When a job request comes, the scheduler initializes job and cluster parameters comes, the scheduler initializes job parameters and calculates the Expected computing power, ECP for each job together with ALC, Average System Load and ACP of clusters before job allocation. The algorithm find the deviation of ALC with the Average system load and find out the probability value of deviation for every cluster. If the probability of deviation is within the range of 0 and 1, the cluster is marked as under loaded. The ACP of under loaded clusters is compared with the ECP of jobs. If the ACP value of a cluster is less than or equal to ECP of jobs, the cluster is considered as fittest and job is allocated to it. After job allocation to clusters, some clusters may remain underutilized. To avoid this, AHLBI compares the queue length of all the clusters. Jobs from clusters with large queue size are stolen and allocated to free clusters. Similarly, when the number of jobs waiting to be executed in a cluster's queue increases, jobs from queue tail is allocated to free clusters for execution. the advantages of this algorithm are reducing idle time of clusters and makespan. The drawback of this strategy is that it does not take into account the resource processing capacity and the fault tolerance.

In this paper [7], the authors proposed the Distributed Load Balancing Model for Grid Computing that represents a Grid topology based on a forest structure. Jobs migration is presented on two levels, namely (a) intra-cluster and (b) inter-cluster load balancing. The nodes of the cluster send their load information to cluster managers. Cluster manager is responsible of saving the nodes load information and also distribution of the information with other cluster managers through inter-cluster communication. The advantage of this algorithm is that it takes into consideration the heterogeneity of the resources, it reduces the response time and the communication cost. The drawback of this strategy is that it does not take into account the resource processing capabilities and the fault tolerance.

III. PROPOSED LOAD BALANCING ALGORITHM

The proposed Load Balancing implements three policies: Information Policy, selection Policy and location Policy. For implementation of Information Policy we use activity based approach. We use FIFO strategy For implementing the Selection Policy, for implementation of location policy We use as Load Index Queue Length. On the basis of Load Index Load Balancer decides to activate Load Balancing process.

Notations used in our algorithms are summarized and shown in Table I:

TABLE I. NOTATIONS USED

Parameter	Description
N	Node
$Load_R$	Load of resource
Qlength	Queue length of resource
TH_H	The higher threshold
TH_L	The lower threshold
OLD-list	Overloaded List
ULD-list	Underloaded List
BLD-list	Balanced List
$Load_{avg}$	Average Load
NBR_N	Number of Nodes of cluster c
C	cluster

A. *Intra-cluster load balancing algorithm*

Depending on its current load, each cluster manager decides to start a Job Migration operation. In this case, the cluster manager tries, in priority, to balance its Load among its nodes. To implement this local load balancing, we propose the following algorithms:

Algorithm 1: Information policy

We considered the load of node at given time was described simply by CPU queue length. it denotes the number of processes which are waiting to be executed.

We calculated this parameter as follow:

Load (Qlength) = (Q1+Q2+……...+QT)/T
Where:Q1,Q2,……...,QT is the value of Qlength in a previous one second interval.

T is the number of time intervals.

```
gathering information algorithm
Begin
T←5 seconds
Waiting for jobs;
Create jobs queue for each node;
For every Node N and in each one second of T intervals do
    Calculate (Qlength);
End For
Load (Qlength)=($Q_0$+$Q_1$+…..$Q_T$)/T;
According to its period cluster manager receives Load
informations from all nodes and compute load of cluster C
associated.
Cluster manager Sends Load information of C to Grid
manager
Loop
wait for load change // happening of any of defined events
if (events_happens ()=1 or  events_happens ()=4) then
begin
Remove terminated or migrated job from the waiting queue
Subtract their load value from the total local load of node.
Send new load to its  cluster manager associated ;
End
if (events_happens ()=2 or  events_happens ()=3) then
begin
Add the newly created or incoming job for the waiting queue
Add their load value for the total local load of node
Send new load to its cluster manager associated;
End
If ((events_happens () =6) and (events_happens () =7)) then
begin
ask the slowest resource to send a portion of its load to
the idle resource .
End
End Loop
```

```
Function events_happens ()
output Type: integer
begin
If (Job.state=Termination) then events_happens () =1;
If (Job.state=Start) then events_happens () =2;
If (Job.state=Incoming Migrating ) then events_happens ()=3;
If (Job.state = migrated) then events_happens ()=4;
If (Arrival of any new resource) then events_happens ()=5;
If(resouce.state= idle) then    events_happens () =6;
if(resource.state= slowest) then events_happens ()=7;
if(cluster.state=saturated then events_happens ()=8;
if (cluster.state=unbalanced) then events_happens ()=9;
end
```

Algorithm 2: Location policy

This algorithm classifies the nodes according to their load. it used three states for classifying: overloaded, underloaded and balanced. In the first time, we must calculate two threshold values for Qlength parameter.

The calculation of these thresholds is done as follow:

Calculate load average of Qlength parameter over all nodes

$Load_{avg}(Qlength)=(load1+load2+\dots loadn)/nbr$; Where

$Load_{avg}(Qlength)$ is the average load of Qlength over all nodes.

load1,load2,….loadn are the current load of Qlength of each node calculated by Load estimation algorithm. nbr is the number of nodes.

Calculate the threshold values

The higher and lower threshold values of Qlength parameter are calculated by multiplying the average load of Qlength and a constant value.

$TH_H=H*Load_{avg}$

$TH_L=L*Load_{avg}$

Where TH_H is the high threshold and TH_L is the low threshold

H and L are constants.

The next step is to partition the nodes for balanced, overloaded and underloaded nodes by using the threshold values as follow:

- Overloaded : the node will be added for overloaded list if Qlength is high, that is mean if the number of jobs in the queue of node is high then the node is classified as overloaded node.
- Underloaded: the resource will be added for underloaded list if Qlength is low.
- balanced :the node are not into overloaded list and underloaded list are the node in the balanced load state they are considered as more loaded than the low state and less loaded than the high state.

```
Location policy algorithm
Begin
If(events_happens ()=8) then Inter-cluster load balancing
algorithm
somme ←0;
For every Node N  of cluster C do
      Somme← Somme+ $Load_N(Qlength)$ ;
End For
$Load_{avg}(Qlength)$= somme1/NBR-N;
$TH_H(Qlength)$= $Load_{avg}(Qlength)*H$;
$TH_L(Qlength)$= $Load_{avg}(Qlength)*L$;
Partionning Nodes into overloaded list OLD-list , underloaded
list ULD-list and balanced list BLD-list
OLD-list← ∅; ULD-list←∅; BLD-list←∅;
For every Node N of cluster C do
 If ($Load_N(Qlength)$ )>$TH_H(Qlength)$ then
          OLD-list ←OLD-list ∪ N;
 Else  If ($Load_R(Qlength)$ )< $TH_L(Qlength)$)
   then       ULD-list← ULD-list ∪ N
        Else  BLD-list← BLD-list ∪ N;
End If
End For
Sort OLD_list by descending order relative to their  $Load_N$.
Sort ULD_list by ascending order relative to Their $Load_N$.
End.
```

Algorithm 3: Job Migration Decision

After classifying the nodes, in the next step cluster manager decide to migrate jobs from overloaded to under-loaded nodes, it applies the following algorithm:

```
Job Migration Decision algorithm
begin
While (OLD-list ≠ ∅ .AND. ULD-list ≠ ∅ ) do
      For i = 1 To  ULD-list.Size()  do
                 Select job from queue of first node
                 belonging to OLD-List by FCFS algorithm
                 Migrate the selected job from first
                 Sender node of OLD-List to ith receiver
                 node of ULD-list;
                 Update the current $Load_N$ of receiver
                 and sender  nodes;
                 Update OLD-list, ULD-list and BLD-list;
                 Sort OLD-list by descending order of their
                 $Load_N$;
       End For
End
```

B. Inter-cluster load balancing algorithm

This algorithm applies a global load balancing among all clusters of the Grid. The Inter-cluster load balancing at this level is made if a cluster manager fails to balance its Load among its associated nodes. In this case the grid manager migrates Jobs from overloaded clusters to under loaded clusters. We propose the following algorithms:

```
Inter-cluster load balancing algorithm

begin
According to period T do
Grid manager receives Load informations of clusters from its
Cluster Managers .
Grid manager collect related informations of its clusters in the
clusters information table;
Grid manager partitions grid into overloaded  (OLD), under-
loaded  (ULD) and balanced (BLD)  clusters;
Create OLD_clusters_table;
Create ULD_clusters_table;
Sort clusters Cj of OLD_clusters_table by Descending order
of their Load;
For Every cluster $C_j$ of OLD_clusters_table  Do
 begin
          Sort clusters $C_r$ of ULD_clusters_table by  Ascending
          order of their Load
          Sort nodes of $C_j$ by descending order of their Load
While (OLD_clusters_table  ≠ Φ AND ULD_clusters_table ≠
Φ) Do Begin
           Sort the clusters $C_r$ of ULD_clusters_table by
```

```
        ascending order of inter clusters ($C_i$-$C_r$) WAN
        bandwidth sizes.
        Sort the nodes of $C_i$ by descending order of their
        load
        Sort Jobs of first node of $C_i$ by FCFS algorithm and
        communication cost
        Migrate the selected job from the first node of $C_i$ to
        j^th cluster of ULD_clusters_table
        Update the current Load of receiver cluster
        Update ULD_clusters_table, and OLD_clusters_table
End
Endfor
End
```

IV. EXPERIMENTAL RESULTS

We implemented the proposed load balancing algorithm in Java using JDK 1.8. The implementation is done on a Alea 2 grid simulator[8].

A. Simulated parameters

In order to evaluate the feasibility and the performance of our algorithms we have tested them in Alea 2 Grid simulator. We utilized the following parameters:

1. Resource parameters: these parameters give information about available resources during load balancing period such as:
    a. number of Clusters
    b. number of resources in each Cluster
    c. size of memory (RAM)
    d. date to send load information from resources
    e. tolerance factor.
2. job parameters: these parameters include:
    a. number of jobs queued at every resources;
    b. arrival time, waiting time, submission time ,start time , processing time and finish time
    c. job length
    d. job priority
3. Network parameter: LAN and various WAN bandwidth sizes.
4. Load index: we have used CPU queue Length Where CPU queue Length denotes number of waiting jobs in queue of resource.

B. Performance parameters

We have focused on the following parameters:

1. Average response time: the response time is the time a job spends in the system which means the time from its arrival to its termination.
2. Slowdown: denotes as the ratio of response time to processing time . Slowdown= Max (response time/processing time) of all jobs.
3. Average resource usage rate

C. Performance Evaluation

The proposed algorithm provides better Job Migration mechanism with DLBA. The important performance factors in estimating our proposed algorithm are decreasing response time, reducing slowdown and maximizing resource utilization. We performed some experiments for evaluating efficiency and performance of the proposed algorithm.

Experimentations 1:

In the first experimentation, we have focused on the average response time (in sec), according to various numbers of jobs and clusters. We have supposed different numbers of clusters and we considered that each cluster is composed of various numbers of resources. The results showed that proposed algorithm surpassed other algorithms by decreasing the average response time. In FCFS (first come, first served) algorithm, if the resource demanded by the first job in the queue is not available, the remaining jobs cannot schedule even if the demanded resources are available. In the proposed algorithm, it works as FCFS in the job selection but when the first job in the queue cannot be scheduled directly the proposed algorithm estimate the earliest probable starting time for the first job using the processing time calculated of running jobs. Then, it makes a reservation to run the job at this pre-estimated time. Next, it examines the queue of waiting jobs and directly schedules every job not intervening with the reservation of the first job.

From figure 1 and 2, we can perceive that the proposed algorithm works much better than FCFS and EDF(Earliest deadline first) since jobs are equally distribute over available clusters. The average response time and average slowdown are slightly better for the proposed algorithm. we have tried solving the problem of staturation by preventing the overloaded resources and we don’t permit receiving more jobs. Evidently, simple solution is not enough for more complex problems. We will try to fully comprehend this phenomenon in the future since it is outside the focus of this paper.

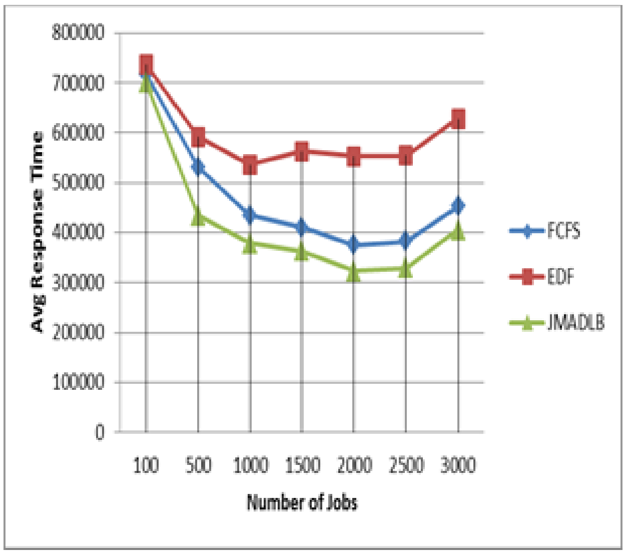

Fig 1. Comparison of avg response time(in sec), between FCFS, EDF and JMADLB with 20 clusters

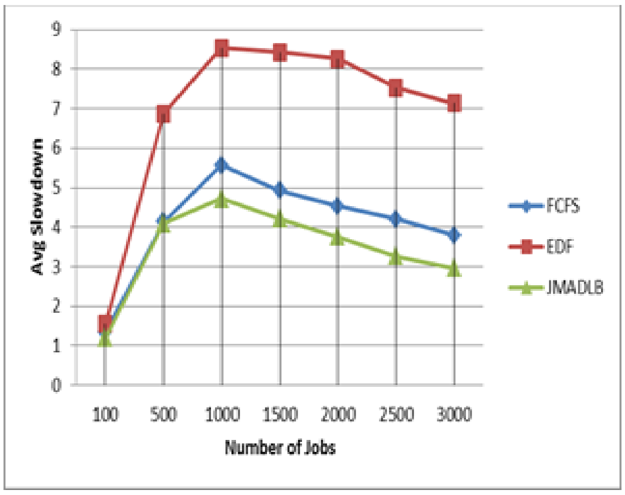

Fig 2.Comparison of avg slowdown(in sec) between FCFS, EDF and JMADLB with 20 clusters

Experimentations 2:

In the second experimentation, we have focused on the resource utilization (%). We have supposed number of clusters is 14, and we considered that each cluster is composed of various numbers of resources. Number of jobs is 3000. Figure 3 shows the cluster utilization of different algorithms.

Examination of the cluster utilization showed that FCFS and EDF did not perform well. It shows that the cluster-11 which is having the highest processing resources is over utilized, while clusters 2, 4, 5, 8 and 9 are idle in FCFS and EDF scheduling algorithms. The reason behind improvement is even utilization of all clusters which is achieved because JMADLB balances the load between clusters at the time of scheduling. It shows that the JMADLB is better performed compared to traditional scheduling algorithms, also because load balancing is used to make sure that none of existing clusters are idle while others are being utilized. The load balancing effects are caused by under-loaded clusters. In the proposed algorithm there is an increase of utilization of cluster 2 from 0% (before JMADLB algorithm) to 8% (after) and cluster 4 from 0% to 2.5% and cluster 8 from 0% to 4% and cluster 9 from 0% to 30%. In this case, the different utilizations of the participating clusters are balanced. On the other hand, jobs with specific requirements have to wait until the suitable resources become available. This in fact generates higher system utilization on particular clusters. We have tried solving that by migrate the jobs for idle resources for load balancing and preventing the overloaded clusters. Moreover the JMADLB algorithm permits the scattering of the job on the most available resources when there was no appropriate resource, unlike the other scheduling algorithms that try to select the best resource resembling to the job requirements; otherwise, the job will stay in the global queue, which indicates an under-utilization of the resources.

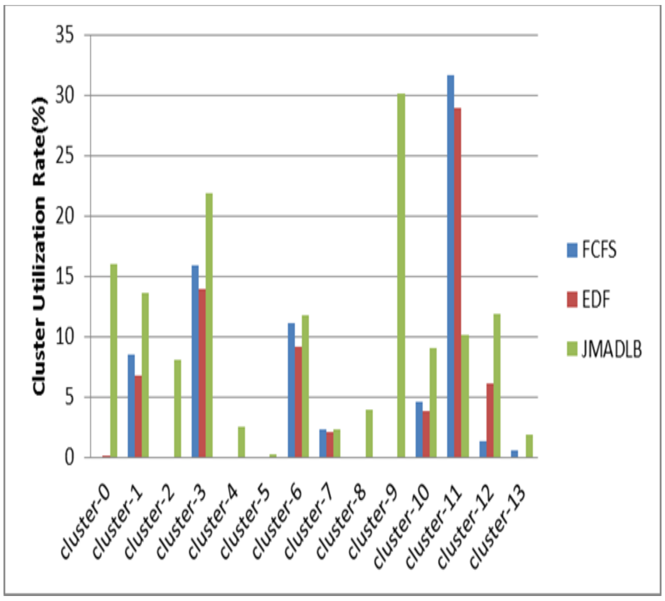

Fig 3. Comparison of Cluster Utilization (%) between FCFS, EDF and JMADLB with 14 clusters

Experimentations 3:

In the third experimentation, we have focused on waiting job in queue and we have compared it with running job. We have supposed the number of clusters is 20 and we considered that each cluster is composed of various numbers of resources. Number of jobs is 3000.

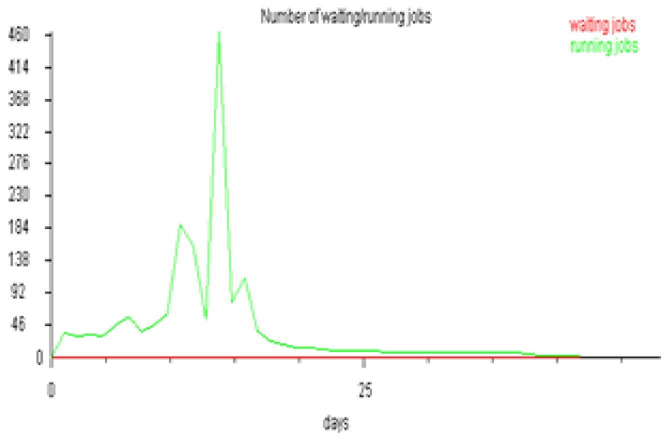

Fig 4 . Comparison of running jobs and waiting jobs of JMADLB algorithm with 20 clusters

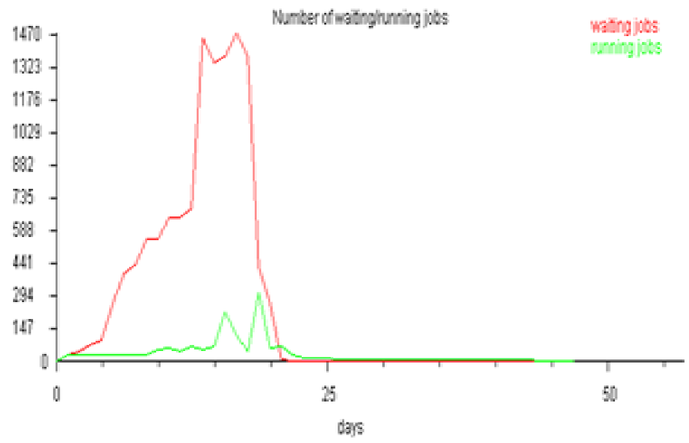

Fig 5 . Comparison of running jobs and waiting jobs of EDF algorithm with 20 clusters

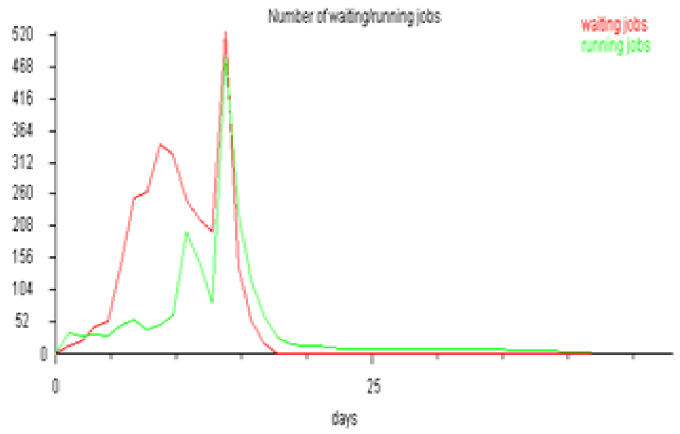

Fig 6. Comparison of running jobs and waiting jobs of FCFS algorithm with 20 clusters

The horizontal axis represents time (units of days) while the vertical axis indicates that the number of jobs. The red curve shows waiting job who says that a job in the waiting jobs queue, the green curve shows running job which say that a job is running or executing. EDF and FCFS are not able to schedule jobs easily, generating greatest waiting jobs during the time. For 3000 jobs and with 20 clusters JMADLB, is capable of a higher resource utilization and there is no waiting jobs in the queue through the time as can be seen in figure 4.

V. Conclusion and Future Work

In this paper we have presented a load balancing algorithm in grid computing environment. To validate the proposed algorithm, we have used a Grid simulator in order to measure its performance. The first experimentation results are very promising and lead to a better load balancing between nodes of a Grid without high computing overhead. We have obtained good results especially for resource utilization. In the future, the authors want to develop the proposed algorithm by adding the multi-agent systems and we will run our algorithm in decentralized manner. Nevertheless, our algorithm has some limitation that the authors intend to address in the future. In this work, the authors did not study the effect of increasing the number of Job Migration and the performance degradation due to the migration in addition to the drawback of the centralized system. So for the future work, the authors will be interested by these directions.

References

[1] Foster, C. Kesselman, J.M. Nick, and S.Tuecke.,Grid services for distributed system integration'. IEEE Computer, 35(6):pp.37-46,2002.

[2] Y. Zomaya, Y.Teh, Observations on using genetic algorithms for dynamic load-balancing, IEEE Transactions on Parallel and Distributed Systems, vol. 12, No.9, 2001, pp.899-911 .

[3] Belabbas Yagoubi , Hadj Tayeb Lilia and Halima Si Moussa , 2006. Load Balancing in Grid Computing. Asian Journal of Information Technology, 5: 1095-1103.

[4] B.Yagoubi ,Y. Slimani,Job Load Balancing Strategy for Grid Computing .Journal of Computer Science 3(3),pp.186-194,2007.

[5] S. Kumar, and N. Singhal. (Jul, 2012). A Priority based Dynamic Load Balancing Approach in a Grid based Distributed Computing Network, International Journal of Computer Applications, Vol. 49, No.5, pp. 819-828.

[6] Joshua Samuel Raj, Hridya K. S and V. Vasudevan, Augmenting Hierarchical Load Balancing with Intelligence in Grid Environment , International Journal of Grid and Distributed Computing Vol. 5, No. 2, June, 2012.

[7] B. Yagoubi, and M. Meddeber. (2010). Distributed Load Balancing Model for Grid Computing, Revue ARIMA, Vol. 12, pp. 43-60.

[8] Dalibor Klusáček and Hana Rudová.(2010). Alea 2 job scheduling simulator. *In Proceedings of the 3rd International Conference on Simulation Tools and Techniques* (SIMUTools 2010), Torremolinos, Malaga, Spain.

Website: Dalibor Klusáček and Hana Rudová .Retrieved from http://www.fi.muni.cz/~xklusac/workload