ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal
Regular Issue, Vol. 8 N. 4 (2019), 5-18
eISSN: 2255-2863
DOI: http://dx.doi.org/10.14201/ADCAIJ201984518

Load balancing with Job Migration Algorithm for improving performance on grid computing: Experimental Results

Ali Wided[b,c], Kazar Okba[c] and Bouakkaz Fatima[a]

[a] Computer Sciences Department, Esi University,Alger, Algeria
[b] Computer Sciences Department, University of Tebessa, Algeria
[c] Computer Sciences Department, University of Biskra, Algeria
aliwided1984@gmail.com, kazarokba@yahoo.fr , f_bouakkez@esi.dz

KEYWORD

Job Migration; Resource Utilization; Queue Length; CPU utilization; Memory utilization; Threshold; grid computing; load balancing

ABSTRACT

Grid is the collection of geographically distributed computing resources. For efficient management of these resources, the manager must maximize its utilization, which can be achieved by efficient load balancing with Job Migration techniques. Job Migration from overloaded resources to underloaded is an attempt to load balancing across all processors, thus reduce average response time. The decision of migration is based on the information exchange between resources.In this paper, the authors propose a novel Job Migration Algorithm for Dynamic Load Balancing (JMADLB), in which parameters such as CPU load and queue length have been considered and have been used for the selection of overloaded resources (or underloaded ones) in Grid. Here, the overloaded resources do not accept any new job; but, the new jobs are migrated to underloaded resources, even though this mechanism migrate extra jobs to obtain load balancing. The performances of the proposed algorithms were tested in Alea 2 simulator by using different parameters like response time, resources utilization and waiting time in the global queue. In addition, they were compared with other scheduling algorithms such as First Come First Served (FCFS) and Earliest Deadline First (EDF).

1. Introduction

Grid computing has recently become one of the most vital research themes in the field of computing. The Grid paradigm has acquired popularity as a result of its capability to give facile access to geographically distributed resources functioning through various administrative domains. The Grid environment looks as a combination of heterogeneous, dynamic, and shared resources in order to give reliable and faster access to Grid resources (Rathore & Chana, 2016). The main aim of load balancing is to enhance the application response time by which load would be distributed according to resources. This is even more vital in computational Grid where the main concern is to equitably submit jobs to resources and to reduce the difference between the overloaded and underloaded resources (Yagoubi & Slimani, 2007). For efficient resource management in Grid, the overloaded resource must be prohibited which can be achieved by Job Migration Techniques. In Job Migration, a job running on resource is reallocated on another one in a way that the migration does not cause any modification in the job execution. It means that, the job is paused and later resumed on a new resource. Here, once the job is submitted to a selected resource, job queue allocates to Processing Element (PE) and starts the Job Migration. If any of the resources is failed in the job processes, Local Job Migration Scheduler (LJMS) interrupts the job execution and attempts to restart it on a new resource in the same cluster, then transfer the job into other resource which is based on the availability of the underloaded resource. Grid Information System periodically monitors the status of the job. If it detects that the job has been in the restart state for a long period, it tries to find a new resource to which the job can be migrated.

Generally, a load balancing structure consists of three stages: collection of information, making a decision and migration of jobs. During the information collection phase, load balancer collects the load information, state of resources and discovers whether there is imbalance in the load. The decision making phase focuses on calculating an optimal jobs distribution, while the Job Migration phase transfers the excess quantity of load from overloaded resource to under loaded ones.

In this paper, we try to reach the following goals:

- Decreasing the average response time, whenever possible, of jobs submitted to the Grid;
- Maximize throughput and resource utilization

The paper is ordered as follows: Section 2 presents the existing related works. Section 3 defines the contribution of the proposed work and section 4 describes the proposed algorithms. Experimental results are presented in Section 5. At last, authors conclude the paper in section 6.

2. Related work

The load balancing problem in Grid computing has been extensively studied. Indeed, the literature comprises many solutions using different approaches to resolve the problem from diverse points of view. They can be classified according to different criteria: local vs global, centralized vs decentralized, static vs dynamic, threshold-based vs threshold-independent, adaptive vs non-adaptive, … etc. More details of the largely adopted taxonomy are well explicated by (Yagoubi & Slimani, 2007).
Authors (Khan *et al*., 2017) have divided the load balancing strategies into two categories, some favoring Job migration and some of them not favor for Job migration during the load balancing process. The authors divided the load balancing strategies that supports Job migration into two groups on the base of topology, namely: flat and hierarchical topology.

2.1. Load Balancing Strategies based on Hierarchical Topology

(Khanli *et al*., 2012) have proposed the strategy of a New Step toward Load Balancing (NSLB), it has been based on dynamic load balancing and it is an enhancement of Branch and Bound (B & B) algorithm (Mezmaz *et al*., 2007). It is based on the transitional phases and competency rank in grid networks. The proposed strategy assigns the jobs to resources on the basis of their processing power. Resources are explored with different priorities at different clusters by the use of competency rank and resource searching is achieved based upon a control word. As the job enters into the system, it is located in one of the queues from three queues namely: (a) high (b) middle and (c) low agreeing to priority. The resource is choosing the priority which matches with the job priority.The advantages of this algorithm are taking into account resource processing power and cost. The drawback of this strategy is that is does not consider the fault tolerance.

Another study of (Nandagopal & Uthariaraj, 2010) has presented the Hierarchical Status Information Exchange Scheduling Balancing for computational Grid environments which is a scheduling strategy that accomplishes load balancing by using the Minimum Cost and Minimum Load policies. The resources are heterogeneous and distributed through multi-cluster environment. Load balancing in the proposed strategy is applied at two levels, namely: (a) Grid level and (b) local level. At the Grid level, by using the Minimum Cost policy, a Grid scheduler migrate the job to the suitable cluster. At the local level, on the basis of the Minimum Load policy, the Local Scheduler assigns the job to the Grid resource. The Job Migration mechanism is also a part of the strategy where a job is migrated from an overloaded cluster to the under-loaded cluster; the proposed work addresses the problem of load balancing while scheduling jobs to the resources in multi-cluster environment using Min-Load and Min-Cost policies.The advantages of this algorithm is that it takes into account the resource load, resource processing capabilities and the network cost. The drawback of this algorithm is that no priority constraint among different jobs in addition to not taking into consideration fault tolerance.

(Payli *et al*., 2011) have proposed the dynamic hierarchical load balancing protocol which is based on the clusters with one coordinator allocated to each of the clusters. The responsibility of coordinator is load balancing in the cluster. If the load balancing in cluster fails, then the coordinator connects with the coordinators of other clusters to send or to receive the load. For load balancing, the coordinator of each of the clusters periodically checks the nodes within the cluster to discover whether they are overloaded or not. With the help of daisy chain architecture, a token is circulated in the coordinator ring, and the global load information is also passed to all the coordinators when it is required. The advantages of this method are that it is scalable and has low message and time complexities. The drawbacks of this algorithm are that it does not take into account the resource processing capabilities, the fault tolerance and the communication overhead.

(Samuel *et al*., 2012) has presented the Augmenting Hierarchical Load Balancing with Intelligence (AHL-BI) strategy for Grid environment which assigns jobs to the resources of Grid based on the comparison between the estimated computing powers of jobs with the average computing power of the clusters. It also uses the advantages of Job Migration so that no cluster stay idle, if the cluster with jobs queued length greater than the threshold value, then it migrates the job from the overloaded resource and assigns it to underloaded resource or resource with current lower queue length than the threshold. The threshold is a fixed number of jobs in the queue of a resource, 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.

Authors (Yagoubi & Meddeber, 2010) have proposed the Distributed Load Balancing Model for Grid Computing that represents a Grid topology based on a forest structure. This strategy is based on distributed load balancing and supports Grid scalability, heterogeneity, and task's independency. Jobs migration is presented on two levels, namely (a) intra-cluster and (b) inter-cluster load balancing. The strategy describes a Grid at two levels. The nodes are considered as leaves and the root resembles to the cluster manager in a virtual node form related with the cluster. 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.

3. Job Migration Algorithm for Dynamic Load balancing (JMADLB)

We distinguish between two load balancing levels: Intra-cluster Job Migration algorithm and Inter-clusters Job Migration algorithm.
There are some specific events which modify the load configuration in Grid environment which can be categorized as following:

- Any new job is arrived
- Accomplishment of execution of any job
- Any new resource is arrived
- Any existing resource is withdrawal
- Machine failure at any resource
- resource become overloaded

When any of these events happen, the local load value is changed and the resource information table is updated. Notations used in our algorithms are summarized and shown in Table 1:

Table 1:. Notations used in our algorithms

Parameter	*Description*
R	Resource
$Load_R$	Load of resource
Qlength	Queue length of resource
CPU-U	CPU utilization of resource
Mem	Memory utilization
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_R C	Number of resources cluster c

3.1. Intra-cluster Job Migration algorithm

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

Algorithm 1: Load Estimation

The load of resource at a given time was described simply by CPU queue length. It denotes the number of processes which are waiting to be executed. In this algorithm, we considered CPU-U (CPU Utilization), Q length (Queue length) and Mem (memory utilization) as load information parameters to measure load of a resource.

We calculated these parameters as follow:

Load (CPU-U) = $(U_1+U_2+\ldots\ldots+U_T)/T$ Where: $U_1, U_2, \ldots\ldots, U_T$ is the value of CPU-U in a previous one second interval

Load (Qlength) = $(Q_1+Q_2+\ldots\ldots+Q_T)/T$ Where: $Q_1, Q_2, \ldots\ldots, Q_T$ is the value of Qlength in a previous one second interval

Load(Mem) = $(M_1+M_2+\ldots\ldots+M_T)/T$ Where: $M_1, M_2, \ldots\ldots, M_T$ is the value of Mem in a previous one second interval

T is the number of time intervals.

The averaged information of CPU-U, Qlength and Mem are the load parameters used to describe the load of resource.

```
Load estimation algorithm
Begin
T←5 seconds
Waiting for jobs;
Create jobs queue for each resource;
```

```
For every resource R and in each one second of T intervals do
    Calculate (CPU-U);
    Calculate (Qlength);
    Calculate (Mem);
End For
Load(CPU-U)=(U0+U1+…..UT)/T ;
Load (Qlength)=(Q0+Q1+…..QT)/T;
Load (Mem)=(M0+M1+…..MT)/T;
According to its period LocalScheduler receives Load informations from all resources and compute
load of cluster C associated.
LocalScheduler Sends Load information of C to GlobalScheduler
Loop
wait for load change // happening of any of defined events
if (events_happens ()=1 or events_happens ()=4) then
begin
Removes terminated or migrated job from the waiting queue
Subtracts their load value from the total local load of resource.
Send new load to its LocalScheduler associated ;
End
if (events_happens ()=2 or events_happens ()=3) then
begin
Adds the newly created or incoming job for the waiting queue
Adds their load value for the total local load of resource
Send new load to its LocalScheduler associated;
End
If ((events_happens () =6) and (events_happens () =7)) then begin
asks the slowest resource to send a portion of its load to
the idle resource
End
End Loop
End.

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: Decision Making

This algorithm classifies the resources according to their load parameters. It used three states for classifying: overloaded, underloaded and balanced. In the first time, we must calculate two threshold values for each load parameters (CPU-U and Q length)

The calculation of these thresholds is done as follow:

Calculate load average of each parameter (CPU-U and Qlength) over all resources
$Load_{avg}$(Qlength)=($load_1$+$load_2$+….$load_{NBRR}$)/NBR_R; Where
$Load_{avg}$(Qlength) is the average load of Qlength over all resources.
$load_1$,$load_2$,….load NBR_R are the current Qlength of each resource calculated by Load Estimation algorithm.
$Load_{avg}$ (CPU-U) =($load_1$+$load_2$+….$load_{NBRR}$)/ NBR_R; Where $Load_{avg}$ (CPU-U) is the average load of CPU-U over all resources.
$load_1$,$load_2$,….$load_{NBRR}$ are the current load of CPU-U of each resource calculated by Load Estimation algorithm.
Calculate the threshold values
The higher and lower threshold values of load parameters are calculated by multiplying the average load of each parameter 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 resources for balanced, overloaded and underloaded resources by using the threshold values as follow:
balanced otherwise
Load= overloaded (CPU-U is high) or (Qlength is high) or (Mem>85%)
Underloaded (CPU-U is low) or (Qlength is low)
Idle CPU-U<=1%

1. Overloaded: the resource will be added for overloaded List ,if any of these conditions is true:
   a. CPU-U is high, if the CPU usage is high then the resource is overloaded
   b. Qlength is high, if the number of jobs in the queue of resource is high then the resource is classified as overloaded resource.
   c. Memory usage is greater than 85% ,the resource with less than 15 % free memory at risk of memory paging and paging will slow down processing at that resource
2. Underloaded: the resource will be added for underloaded list if either of these conditions is true:
   a. CPU-U is low; if the CPU usage is low then the resource is added for underloaded list.
   b. Qlength is low; if the number of jobs in the queue of resource is low then the resource is classified in the low state.
3. balanced:the resource are not into overloaded list and underloaded list are the resources in the balanced load state. they are considered as more loaded than the low state and less loaded than the high state.

Decision Making algorithm

```
Begin
If(events_happens ()=8) then Inter-cluster JobMigration algorithm
    somme ←0; somme1←0;
For every resource R of cluster C do
    Somme ←somme+ LoadR(CPU-U) ;
    Somme1← Somme1+ LoadR(Qlength) ;
End For
$Load_{avg}$(CPU-U)= somme/$NBR_R$;
$Load_{avg}$(Qlength)= somme1/$NBR_R$;
$TH_H$(CPU-U)= $Load_{avg}$(CPU-U)*H; $TH_L$(CPU-U)= $Load_{avg}$(CPU-U)*L;
$TH_H$(Qlength)= $Load_{avg}$(Qlength)*H; $TH_L$(Qlength)= $Load_{avg}$(Qlength)*L;
Partionning resources into overloaded list OLD-list, underloaded list ULD-list and balanced list BLD-list
```

```
    OLD-list←—∅; ULD-list←—∅; BLD-list←—∅;
**For every** resource R of cluster c do
**If** (LoadR(CPU-U)> THH(CPU-U)) or (LoadR(Qlength) )>THH(Qlength) or Load (Mem)>85%)
**then**
    OLD-list ←OLD-list ∪ R;
**Else If**(LoadR(CPU-U)<THL(CPU-U)) or (LoadR(Qlength) )< THL(Qlength)) **then**
    ULD-list← ULD-list ∪ R
**Else** BLD-list← BLD-list ∪ R;
**End If**
**End For**
Sort OLD_list by descending order relative to their $Load_R$(Qlength).
Sort ULD_list by ascending order relative to Their $Load_R$(Qlength).
**End.**
```

Algorithm 3: Job Migration Decision

After classifying the resources, in the next step Local Scheduler decide to migrated jobs from overloaded to under-loaded resources, 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 resource
    belonging to OLD-List by FCFS algorithm
    Migrate the selected job from first
    Sender resource of OLD-List to ith receiver
    resource of ULD-list;
    Update the current $Load_R$ of receiver
    and sender resources;
    Update OLD-list, ULD-list and BLD-list;
    Sort OLD-list by descending order of their
    $Load_R$;
    End For
End
```

3.2. Inter-cluster Job Migration algorithm

This algorithm applies a global load balancing among all clusters of the Grid. The Inter-cluster Job Migration at this level is made if a Local Scheduler fails to balance its load among its associated resources. In this case, the Global Scheduler migrates jobs from overloaded clusters to underloaded clusters. In contrast to the intra-cluster level, we should consider the communication cost among clusters. Considering the global state of each cluster, the overloaded clusters can distribute its jobs among underloaded clusters. The chosen underloaded clusters are those that require minimal communication cost for transferring jobs from overloaded clusters. We propose the following algorithms:

Algorithm 1: Gathering Information

```
Gathering Information algorithm
begin
JobCreators sends Jobs to LocalScheduler;
According to period T do
GlobalScheduler receives Load informations of clusters from its LocalSchedulers .
GlobalScheduler collect related informations of its clusters in the clusters information table;
End
```

Algorithm 2: Inter-cluster Job Migration

```
Inter-cluster Job Migration algorithm
GlobalScheduler 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 Cj of OLD_clusters_table Do
begin
    Sort clusters Cr of ULD_clusters_table by Ascending order of their Load
    Sort resources of Cj by descending order of their Load
    While (OLD_clusters_table ≠ Φ AND ULD_clusters_table ≠ Φ) Do
    Sort the clusters Cr of ULD_clusters_table by ascending order of inter clusters
    (Ci-Cr) WAN bandwidth sizes.
    Sort the resources of Ci by descending order of their Load
    Sort Jobs of first resource of Ci by selection algorithm and communication cost
    Migrate the selected job from the first resource of Ci to jth cluster of ULD_clusters_table
    Update the current Load of receiver cluster
    Update ULD_clusters_table, and OLD_clusters_table
Endfor
Endfor    End
```

The last algorithm is implemented in Global Scheduler which determines the way a receiver cluster is selected for a job migrated from overloaded cluster. Global Scheduler calculates the minimum communication cost of sending jobs to receiver cluster resources based on the information collected in the last exchange interval. Global Scheduler selects the cluster that gives minimum overall cost.

4. Experimental resultats

The proposed algorithms were implemented in Alea 2 (Job Scheduling Simulator based on GridSim) (Dalibor & Hana 2010), Alea 2 is an event-based simulator used for testing scheduling algorithms below various conditions (different types of resources and jobs dynamic modifications in the environment, etc.). This simulator is based on GridSim Toolkit (Buyya & Murshed, 2002), and represents an extension that includes better tools for scheduling algorithm implementation visualization competency and an upper speed of simulations.

All experiments have been performed on Intel I3 Duo 2.00 GHz PC with 4GB of RAM running under Windows 2010.

4.1. Performance Evaluation

The proposed algorithm provides better Job Migration mechanism with dynamic load balancing algorithm The important performance factors in estimating our proposed algorithm are decreasing response time, reducing the number of waiting jobs in queue 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), average waiting 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. Tables 2 and 3 show the results of comparison between the average response time and the average waiting time of the proposed algorithm (JMADLB) with FCFS and EDF. The results showed that proposed algorithm surpassed other algorithms by decreasing the average response time and reducing the average waiting time.

In FCFS 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. This helps to decrease the average waiting time of jobs as shown in table 3.

Table 2: Average response Time (in sec) Vs various numbers of jobs and clusters

	Clusters	*14*	*20*	*30*	*40*
Jobs					
100	FCFS	717269.16	717271.54	717276.13	717280.43
	EDF	736509.73	736512.20	736516.98	736521.45
	JMADLB	637271.36	700912.49	698531.38	698080.46
500	FCFS	530716.63	530718.29	530721.74	530724.96
	EDF	591561.10	591743.39	591747.36	591751.077
	JMADLB	440809.53	434139	445092.81	449563.46
1000	FCFS	438615.00	434460.11	434461.78	434463.34
	EDF	537610.93	536181.01	535057.38	535059.01
	JMADLB	331294.50	378577.93	373628.36	353522.78
1500	FCFS	399480.80	410087.71	409660.10	409661.21
	EDF	567050.98	562347.11	561211.74	561213.04
	JMADLB	364341.22	362851.65	364643.87	346922.03
2000	FCFS	369632.99	375405.07	373870.80	373782.58
	EDF	552022.05	552652.35	551188.48	551189.61
	JMADLB	330353.53	323158.54	325187.02	319165.92
2500	FCFS	399492.28	382659.48	392318.16	392301.85
	EDF	578191.08	553857.99	559943.26	559944.21
	JMADLB	347443.27	328675.99	343730.72	348397.71
3000	FCFS	474139.31	452357.27	466811.40	463275.92
	EDF	639816.67	627373.21	639639.79	639640.58
	JMADLB	415538.12	405063.64	410469.78	404221.75

Table 3: average waiting Time (in sec) vs. various numbers of jobs and clusters

	Clusters	*20*	*30*	*40*
Jobs				
100	FCFS	22306.28	22309.45	22312.41
	EDF	58405.46	58409.96	58414.18
	JMADLB	19258.45	19664.86	19372.73
500	FCFS	137967.61	137969.71	137971.68
	EDF	302655.43	302659.11	302662.55
	JMADLB	78715.0	84158.6	82029.63
1000	FCFS	162299.14	162299.97	162300.75
	EDF	323706.82	322631.32	322632.13
	JMADLB	81357.24	80710.37	81642.68
1500	FCFS	128063.37	128023.5	128024.02
	EDF	346174.1	344616.87	344617.41
	JMADLB	58285.76	57226.62	56832.25
2000	FCFS	107723.99	106842.64	106842.72
	EDF	356689.09	355514.91	355515.32
	JMADLB	45392.07	42603.41	46035.96
2500	FCFS	93796.16	93869.37	93863.96
	EDF	344555.89	339015.8	339016.15
	JMADLB	34909.71	35336.33	34526.03

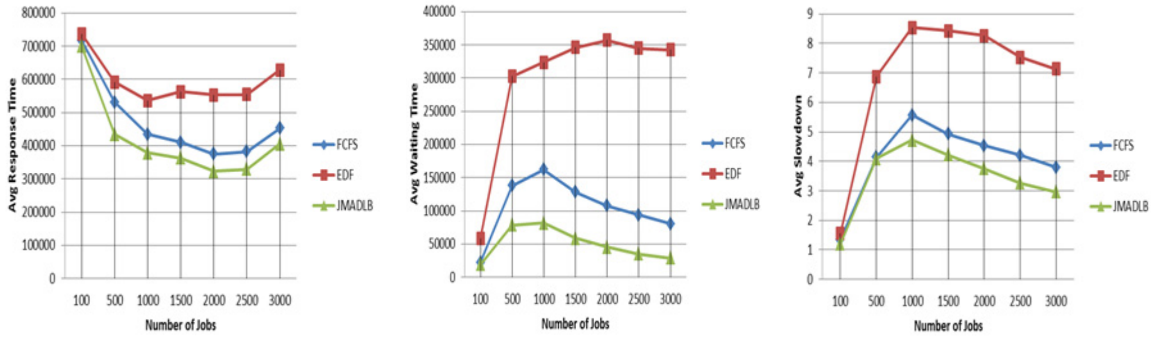


Figure 1: Comparison of avg response time(in sec), avg waiting time(in sec) and avg slowdown(in sec) between FCFS, EDF and JMADLB with 20 clusters.

From figure 1, we can perceive that the proposed algorithm works much better than FCFS and EDF since jobs are equally distribute over available clusters. The average response time, average waiting time and average slowdown are slightly better for the proposed algorithm. On the other hand, as soon as specific job requirements are considered, the proposed algorithm has some difficulties on some experimentation, because the model of specific job requirements mapped in Alea 2 constructed a list of clusters where jobs having the same application identifier were executed. Next, during the execution of application identifier detected for each job and the corresponding cluster from the list are selected to be the only suitable execution sites for the job. The problem here is that all executed algorithms using MetaCentrum dataset will permit job's execution on some cluster (s) if and only if the cluster (s) agree (s) with all specific job requirement. If the suitable cluster is saturated then the corresponding job wait until the suitable cluster is balanced. In the proposed algorithm, we have tried solving this problem 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.

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 2 shows the cluster utilization of different algorithms.

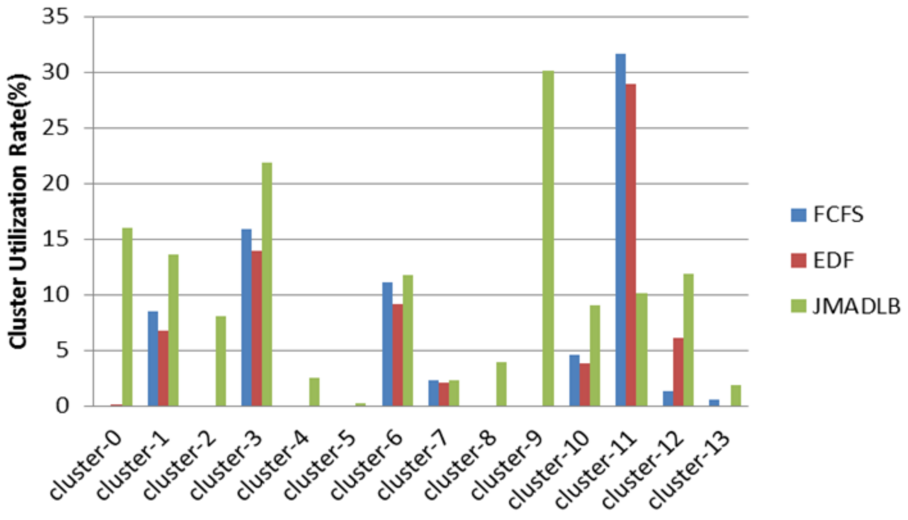
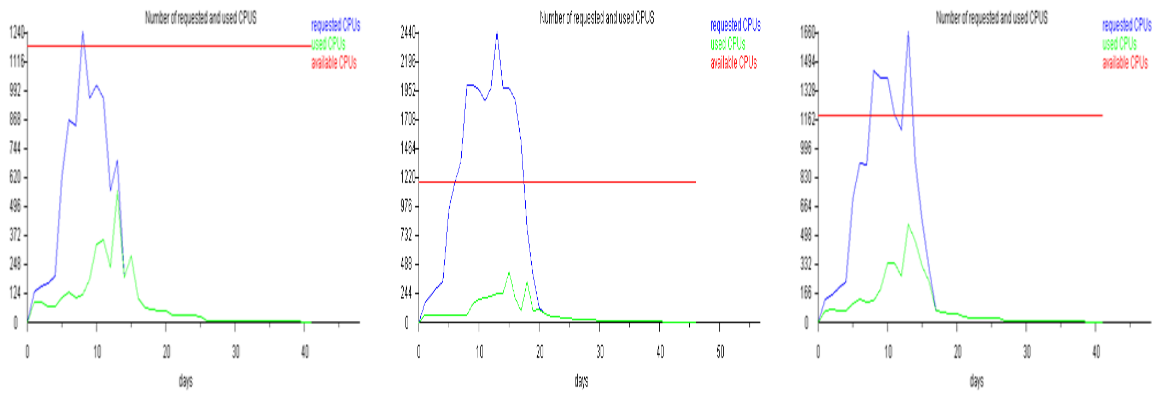


Figure 2: Comparison of Cluster Utilization (%) between FCFS, EDF and JMADLB with 14 clusters.

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.

Experimentations 3:

In the third experimentation, we have focused on used CPU and we have compared it with requested CPU. We have supposed the number of clusters is 20 and we considered that each cluster is composed of various numbers of resources.



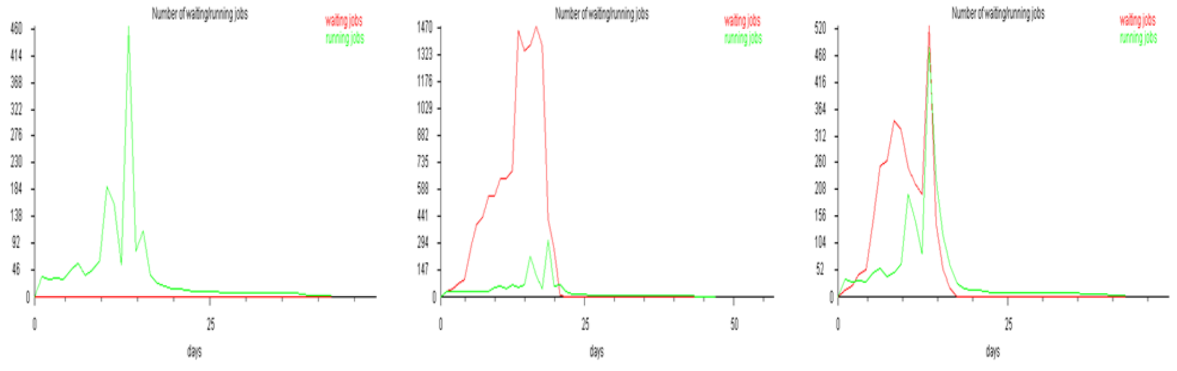
JMADLB EDF FCFS

Figure 3: Comparison of requested CPUS, used CPUS between FCFS, EDF and JMADLB with 20 clusters.

From the above figure, the blue curve shows the total number of requested CPUs whereas the green curve depicts the number of used CPUs. The top in the blue curve indicates that at that particular time, requests for the CPU were too high. So, the favorable situation will be one when the number of blue tops will be less. The red line is the number of CPUs available for execution. With 3000 jobs in FCFS, the number of requested CPUs went above 1660 whereas in EDF it is 2440 CPUs. It is seen that in the proposed algorithm JMADLB has around 1240 CPUs. It is apparent that in FCFS and in EDF, some of the jobs are put into waiting state because there is a raise in the requested CPUs compared to the running CPUs. It is evident that the proposed algorithm removes this probability, however some of the jobs have to wait in JMADLB algorithm too, but the waiting time is small compared to the other two algorithms which is one of the advantages of the JMADLB algorithm.

Experimentations 4:

In the fourth 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.



JMADLB EDF FCFS

Figure 4.Comparison of running jobs, waiting jobs between FCFS, EDF and JMADLB 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 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.

5. Conclusion and future work

In this paper we have proposed a Job Migration algorithm for dynamic load balancing. The Job Migration technique is used to address the following objectives:decreasing the average response time and waiting time of jobs submitted to the Grid ,also maximizing the resource utilization.

To test and estimate the performance of our algorithm, we developed our algorithm under the Alea 2 simulator written in Java. The experimental results are encouraging since we can considerably decrease the average response, waiting time and maximize resource utilization. In the future, we want to develop the proposed algorithm by adding the multi-agent systems and we will run our algorithm in decentralized manner. We will also define other parameters of performance to evaluate and compare our approach with other existing algorithm. We plan also to simulate our algorithm into other known Grid simulators; this will allow us to evaluate the performance of our algorithm in existing simulators.

Nevertheless, our algorithm has some limitation that we intend to address in the future. In this work, we 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, we will be interested by these directions.

6. References

Buyya, R.and Murshed, M .(2002). Gridsim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing, The Journal of Concurrency and Computation: Practice and Experience (CCPE),Vol. 14, pp.13-15.

B. Yagoubi ,Y. Slimani.(2007, March). Load Balancing Strategy for Grid Computing. Journal of Information Technology and Applications ,Vol. 1 No.4, pp. 285-296.

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

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.

Khanli, L.M., Razzaghzadeh, S. and Zargari, S.V., A new step toward load balancing based on competency rank and transitional phases in Grid networks,Journal of Grid Computing, 2008.

N. K. Rathore, I. Chana. (2016, March).Job Migration Policies for Grid Environment. Wireless Personal Communica-tion, Springer Publication-New-York (USA), volume: 89 (1), pp.241-269, IF -0.979.

M. Mezmaz, N. Melab, and E.G. Talbi, An efficient load balancing strategy for Gridbased branch and bound algorithm, Parallel Computing, 2007.

M. Nandagopal, and R. V. Uthariaraj.(Feb. 2010)"Hierarchical Status Information Exchange Scheduling Balancing For Computational Grid Environments," International Journal of Computer Science and Network Security, Vol. 10,No. 2, pp. 177-185.

R. U. Payli, K. Erciyes, and O. Dagdeviren.(5, Sep. 2011). "cluster-based load balancing strategys for grids", International Journal of Computer Networks & Communications, Vol. 3, No, pp. 253-269.

R. J. Samuel, K.S. Hridya, and A. V. Vasudevan, Augmenting Hierarchical Load Balancing with Intelligence in Grid Environment,, International Journal of Grid and Distributed Computing, 2012.

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.

S Khan,B Nazir, IA Khan, SShamshirband, AT Chronopoulos.(20,feb.2017).Journal of Network and Computer Applications 88, 99-111

A. Touzene, S. Al-Yahai, H. AlMuqbali, A. Bouabdallah, and Y. Challa.(2011). "Performance Evaluation of Load Balancing in Hierarchical Architecture for Grid Computing Service Middleware," International Journal of Computer Science, Vol. 8, No.2, pp. 213-223.

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