A MLFQ SCHEDULING TECHNIQUE USING M/M/c QUEUES FOR GRID COMPUTING

DHARAMENDRA CHOUHAN[1], S. M. DILIP KUMAR[1] AND JERRY ANTONY AJAY[1]
[1]Dept. of Computer Science and Engineering
University Visvesvaraya College of Engineering
BANGALORE, INDIA

Abstract

We model the objective function, that the jobs entering the scheduler have a Poisson's distribution and the jobs that are sent out from the multilevel feedback scheduler are also distributed as a Poisson's distribution. We also assume that the number of CPU's in a processing element is not restricted to one, but rather many CPUs integrated into one PE. Therefore, we assume the *M/M/c* queue model for our calculations. In Kendall's notation, we describes a system where arrivals form a single queue and are governed by a Poisson process, where there are *c* servers and job service times are exponentially distributed. Gridlets provided by the users are assigned to processing elements (PEs), and gridlets whose remaining service time is shifted between queues of the MLFQ scheduler to be completed. In MLFQ, the total architecture is divided into multiple prioritized queues. This approach provides gridlets which starve in the lower priority queue for long time to get resources. As a result, the response time of the starved gridlets decreases and overall turnaround time of the scheduling process decreases. This scheduling policy is simulated using Alea GridSim toolkit to test the performance. The proposed MLFQ scheduling algorithm works better in most of the scenarios when compared to FCFS and PBS_PRO algorithms.

Keywords: Grid Computing, Job Scheduling, Multilevel Feedback Queue, GridSim.

1 Introduction

Grid computing is a distributed computing which has emerged for solving a large scale intensive data through sharing of resources over the network [1]. In grid computing systems, there are often large amounts of resources available to be used for computing jobs. Scheduling in a grid computing system is not as simple as scheduling on a multi-processor machine because of several factors. These factors include the fact that grid resources are sometimes used by paying customers who have interest in how their jobs are being scheduled [2]. However, grid computing systems usually operate in remote locations so scheduling tasks for the clusters may be occurring over a network [3]. Job scheduling algorithms are commonly applied to grid resources to optimally post jobs to grid resources [4, 5]. Usually, grid users submit their jobs to the grid manager to utilize and fulfill the facilities provided by grid. The grid manager distributes the submitted jobs among the grid resources to minimize the total response time.

In a Grid environment, there are moderately large number of job scheduling algorithms proposed to minimize the total completion time of the jobs [6, 7]. These algorithms works on minimizing the overall completion time of the jobs by analyzing the suitable resources to be assigned to the jobs. In contrast with minimizing the overall completion time of the jobs does not necessarily result in the minimization of execution time of each individual task. In this paper, we propose a new scheduling policy for grid computing which uses multilevel feedback queue technique concept to avoid the starvation of low priority jobs for a longer duration to get resources to complete their requested services. In this technique, jobs are scheduled according to their remaining service time and they are shifted down from queue to queue as they have some remaining service time. Every queue has unique time quanta that gradually increase from top level to bottom level queues so that longer jobs gradually moves from top to bottom level queues for getting completed. All low priority jobs will process on intermediate queues and gets completed with minimal duration, so that all jobs will get an equal opportunity to utilize grid resources efficiently. The rest of the paper is organized as follows. Section 2 presents the related works. In Section 3, a grid system model for scheduling is presented. In section 4, the MLFQ scheduling technique is proposed.

The simulation of the MLFQ scheduling algorithm using Alea GridSim is presented in section 5. Finally, section 6 concludes the paper.

2 Related Work

There has been significant research continuing to attempt to devise scheduling algorithms for grid environments' problem of efficient job assignment. Some of the jobs scheduling algorithms in a grid environment are given below.

X. He et al. [9] have proposed an algorithm based on the conventional min-min algorithm known as QoS guided min-min which schedules the jobs requiring high bandwidth before others. L. Mohammad Khanli et al. [10, 11] have proposed a QoS based scheduling algorithm for an architecture called Grid-JQA. In this method the solution involves applying an aggregation formula which includes a combination of different parameters together with weighting factors to perform operations on QoS. F. Dong et al. [12] have proposed an algorithm called QoS priority grouping scheduling which considers completion time, accept rate of the jobs and the makespan of the entire system as key factors for job scheduling. E. Ullah Munir et al. [13] have proposed a new job scheduling algorithm which makes use of grid computing environments known as QoS Sufferage. K. Etminani et al. [14] have proposed an algorithm which provides a solution on basis of max-min and min-min algorithms. The algorithm discovers the situations where to adopt one of these two algorithms, based on the standard deviation of the estimated completion times of the jobs on every computing resources. In [15] a game-theoretic-based solution is proposed to the grid load-balancing problem. The developed algorithm combines the inherent efficiency of the centralized approach and the fault-tolerant nature of the decentralized approach. The scheme can be considered semistatic, as it responds to changes in system states during runtime. However, it does not use as much information as traditional dynamic schemes; as such, it has relatively low overhead.

3 Grid System Model

We consider the computational grid system consists of a set of gridresources, *G*, connected via communication systems. In general, each grid resource may contain multiple machines having one or more processing elements.

The processing elements in the machines are heterogeneous, meaning that they may have different processing capacity.

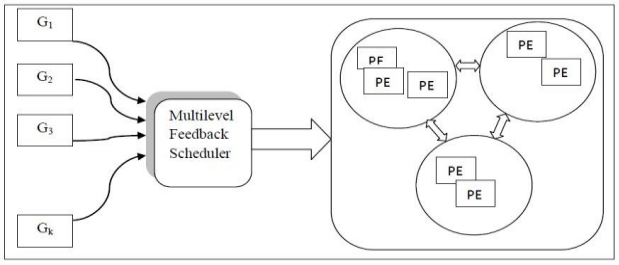

Figure.1: A queuing network model for the GridSim Scheduling system

The grid resources G_1, ... ,G_n in G are fully interconnected, meaning that there exists a communication path between any two grids (G_i, G_j) in G. Inter grid communication is done via message passing, and the underlying network protocol guarantees that messages are received by the intended recipient. Considering the grid computing scenario, the link is viewed as Internet links and modeled. Our communication model represents network performance between a grid G_i to a grid G_j using two parameters-a transmission delay t_j representing the setup cost and contention delays at links on the path from G_i to G_j and a data transmission rate d_j representing the bandwidth available on the path from G_i to G_j. For a message of size s to be transmitted from site G_i to G_j, the transmission time is then given by

$$C_j = t_j + \frac{s}{d_j} \qquad (1)$$

t_j and d_j can be calculated from analytical models or pre-existing information or dynamically forecasted by facilities such as the Network Weather Service (NWS) [16].

Each grid G_i in the grid system can represent one or a combination of the following:

Gridlet: This generates tasks to be executed by the processing elements. Each gridlet is assigned to the to the scheduler to be scheduled for processing.

Scheduler: This receives gridlets and assigns them to the processing elements in the grid system. Every time a gridlet is assigned to the scheduler which is implemented as a multilevel feedback queue, it selects the gridlets based on certain assumed time quanta. Ideally, a large number of gridlets exists. Therefore, the tasks scheduled by the scheduler are a collective from many gridlets.

Processing Elements: Each processing element (PE) executes and processes tasks sent to it. Each PE has a queue that holds tasks to be executed; each task is then processed on a first-come, first-serve (FCFS) basis.

Figure1 shows the relationship between gridlets, scheduler, and processing elements. A site that generate gridlets, can also work as a scheduler, and also process gridlets parallelly. Thus, the tasks that are executed locally at a site which was generated by the site itself will have minimal communication delay C_{ij}.

3.1 Application Model

The system consists of *n* gridlets, one multilevel feedback queue scheduler, and *m* processing elements. Each gridlet *k* is assumed to generate tasks with average rate φ_k (tasks per second) according to a Poisson process and independent of the other generated gridlets. Gridlets are then sent to the scheduler that dispatches them to the processing elements (Figure 1). Depending on the computational power of the processing elements, each processing element *k* executes tasks at an average rate μ_j (tasks per second).

In our model, we assume a Poisson distribution for task execution time and the task execution time of the applications running on the system is assumed to be a Poisson distribution as well. Each grid service provider can be modeled as a *M/M/c* queuing system. We also assume that the task distribution of the applications, once chosen, is consistent throughout the system. For stability, we have the condition/constraint that gridlets must not arrive faster than the system can process them (otherwise, the queues will build up to infinity):

$$\sum_{i=1}^{n} \lambda_i < \sum_{j=1}^{m} \mu_j \qquad (2)$$

where λ_i is the average arrival rate of tasks (in tasks per second) at the scheduler and μ_j is the average processing rate of tasks at the processing element j. The multilevel feedback queue scheduler then sends a fraction r_j of it's gridlets to each processing element j, in which

$$r_j > 0 \qquad (3)$$

$$\sum_{j=1}^{m} r_j = 1 \qquad (4)$$

For stability, the rate of gridlets sent to a processing element j must not exceed the rate at which jobs can be executed by the processor j (otherwise, the queue at processor j will build up to infinity):

$$\sum_{i=1}^{n} r_j * \lambda_i < \mu_j \qquad (5)$$

3.2 Objective Function

The task completion time includes communication delays, the waiting time at the queue, and the task processing time itself. In order to proceed further in deriving the objective function, we assume that the jobs entering the scheduler have a Poisson's distribution and the jobs that are sent out from the multilevel feedback scheduler is also distributed as a Poisson's distribution. We also assume that the number of CPU's in a processing element is not restricted to one, but rather many CPUs integrated into one PE. Therefore, we assume the *M/M/c* queue model for our calculations.

In queuing theory, the *M/M/c* queue is a multi-server queuing model. In Kendall's notation it describes a system where arrivals form a single queue and are governed by a Poisson process, there are *c* servers and job service times are exponentially distributed. It is a generalization of the *M/M/1* queue which considers only a single server.

An *M/M/c* queue is a stochastic process whose state space is the set *{0,1,2,3,...}* where the value corresponds to the number of jobs in the system, including any currently in service.

Arrivals occur at rate λ according to a Poisson process and move the process from state *i* to *i + 1*.

Service times have an exponential distribution with parameter μ in the *M/M/c* queue, as mentioned above.

The buffer is of infinite size, so there is no limit on the number of customers it can contain.

The model can be described as a continuous time Markov chain with generator matrix

$$Q = \begin{pmatrix} -\lambda & \lambda & & & & & \\ \mu & -(\mu+\lambda) & \lambda & & & & \\ & 2\mu & -(2\mu+\lambda) & \lambda & & & \\ & & 3\mu & -(3\mu+\lambda) & \lambda & & \\ & & & \ddots & & & \\ & & & c\mu & -(c\mu+\lambda) & \lambda & \\ & & & & c\mu & -(c\mu+\lambda) & \lambda \\ & & & & & \ddots & \end{pmatrix} \qquad (6)$$

on the state space {0,1,2,3,...}. The model is a type of birth–death process.

The queuing model M/G/1 is employed in [15] it assumes general distributions for its output queue of the schedulers. Therefore we model a M/M/c queuing system, the jobs arrivals are exponential and the output from the scheduler is also considered exponential distribution . In M/M/c queuing system, the average processing time of a task including the waiting time at the queue at a processing element j is given by

$$F_j = \frac{1}{\mu} + \frac{1}{\mu} \cdot \frac{1}{s.s!} \cdot \frac{(\frac{\lambda}{\mu})^2}{(1 - \frac{\lambda}{\mu s})^2} \cdot P_0 \quad (7)$$

where $\frac{1}{\mu}$ is the mean of the job execution distribution, μ_j is the average service rate of tasks (in tasks per second) at processing element j, s denotes the number of processing elements, λ is the arrival rate of gridlets, and finally , for the Poisson queue system,

$$P_0 = [1 + \sum_{n=1}^{\infty} (\frac{\lambda_0 \lambda_1 \lambda_2 \dots . \lambda_{n-1}}{\mu_1 \mu_2 \mu_3 \dots . \mu_n})] \quad (8)$$

Further, the multilevel feedback scheduler is connected to a processing element j via a link with capacity c in bits/s. Each task is assumed to require an average of b bits of data to be transferred. Using equation (1), the expected transfer time of a task from the scheduler to processing element j is therefore given by

$$C_j = t_j + \frac{b}{d_j} \quad (9)$$

This value represents the average communication delay if a task is to be sent from the scheduler to a processing element j.

The completion of a task involves the execution time of the task, the waiting time at the queue, and the communication delays and transfer time of the task to the processing element. Our objective, as always, is to minimize the average completion time of tasks. Using equation (7) and (9), the average completion time of tasks for the scheduler is given by

$$D_i = \sum_{j=1}^{m} r_j . (F_j + C_j) \quad (10)$$

$$= \sum_{j=1}^{m} [\frac{r_j}{\mu} + \frac{r_j}{\mu . s . s!} \cdot \frac{(\frac{\lambda}{\mu})^2}{(1 - \frac{\lambda}{\mu . s})^2}] \cdot P_0$$

We introduce a new variable μ_j shown in (10). μ_j defines the computational power of a processing element j that is available to the tasks coming out from the scheduler. μ_j can be estimated for each processing element j in equation (10). Where k is the gridlet count, λ_k is the arrival rate of gridlets and μ is the ideal computational power of the scheduler.

$$\mu_j = \mu - \sum_{k=1}^{n} r_{k,j} . \lambda_k \quad (11)$$

Using (11), (10) becomes

$$\sum_{j=1}^{m} \{\frac{r_{i,j}}{\mu - \sum_{k=1}^{n} r_{k,j} . \lambda_k} + \frac{r_{i,j}}{(\mu - \sum_{k=1}^{n} r_{k,j} . \lambda_k) . s . s!} \cdot \frac{(\frac{\lambda}{\mu})^2}{[1 - \frac{\lambda}{(\mu - \sum_{k=1}^{n} r_{k,j} . \lambda_k) . s}]^2}\} . P_0 \quad (12)$$

Equation (12) is the objective function that the multilevel feedback queue scheduler is based upon subject to the constraints of (3), (4), and (5). Note that D_i is a function of r_j. It can be proved that the expected response time function (see (12)) is continuous, convex, and increasing.

According to our model, the scheduler is considered to be a multilevel feedback queue. The fundamental problem MLFQ tries to address is to optimize turnaround time. As a common characteristic of a MLFQ, we implement the following rules:

- **Rule 1**: If Priority(Job A) > Priority(Job B), A runs (B doesn't).
- Rule 2: If Priority(Job A) = Priority(Job B), either A or B runs first.
- **Rule 3**: When a job enters the system, it is placed at the lowest priority (the topmost queue).
- **Rule 4**: Once a job uses up its time allotment at a given level, its priority is increased and shifted down to the next queue.
- **Rule 5**: Much of the length has been reduced by preceding queues therefore the final queue contains a list of high priority jobs. The final queue works in a FCFS manner.

In this work, we prove that our implementation of the multilevel feedback queue scheduler works in an efficient manner compared to a FCFS scheduler.

The proposed model works under the following assumptions:

1. Gridlets arriving into the system are independent of one another.

2. When gridlets are mapped to the machines, based on their requirement, it checks for the (resource) availability list.
3. No information is available on the workload of incoming gridlets.
4. The initial processing speed of each PE is provided and processing capacity of Grid resources is updated from time to time based on last gridlet executed and time taken for task completion.

3.3 Multilevel feedback queue (MLFQ)

Multilevel feedback queue plays a significant role in multilevel queue scheduling. In MLFQ, jobs are scheduled according to their remaining CPU burst and they are shifted down from queue to queue as they have some remaining CPU burst. Every queue has unique time slice that gradually increases from upper level queue to lower level queue. So the CPU intensive jobs go down from upper queues to lower queues gradually for getting completed. Thus, lower priority queues are filled with CPU intensive jobs and as a result these processes start to starve for getting CPU attention. The MLFQ scheduling organizes the queues to minimize the queuing delay and optimize the queuing environment efficiency [8].

3.4 State diagram

The system is modeled in a state transition diagram as shown in Figure 2. As gridlets arrives to the input queue, each gridlet is selected and it acquires the requested resources from grid resource list. Once it acquires the requested resources, it finds the suitability of the resources and checks for the required PEs, MIPS, bandwidth and storage. If the suitability is fulfilled, the scheduler assigns gridlets to the resources selected from the resource list.

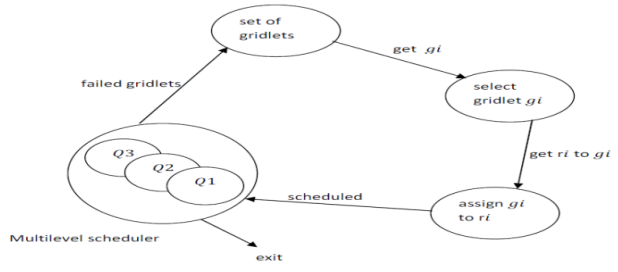

Figure 2. State Diagram

Gridlets are scheduled according to their remaining service time and they are shifted down from queue to queue as they have some remaining service time.

Every queue has unique time slice that gradually increases from upper level queue to lower level queue. So the PEs intensive gridlets go down from upper queues to lower level queues gradually for getting executed. If the gridlet fails to execute at this stage then it is placed back into input queue during the course of execution for later resumption.

4 Proposed Solution

In this section, we briefly explain the proposed solution for scheduling the jobs using MLFQ technique in Grid environment. The user submits gridlets along with the requirements to the Alea GridSim scheduling system. The submission of gridlets to the resources involves checking the suitability of the available PEs. If the requirement is satisfied, the gridlets are assigned to the respective resources. This technique uses a dynamic priority mechanism to schedule the gridlets to the system efficiently and maximize the resource utilization. The MLFQ scheduling model is depicted in the Figure 3. The gridlet waiting for the service is placed in the waiting queue. The gridlets that are scheduled in the queue Q_1 are executed. If the gridlets in Q_1 submitted for execution do not complete in the given time quanta of Q_1 then those gridlets are pushed onto the next level queue Q_2. Then the gridlets pushed on to Q_2 are executed along with the gridlets present in queue Q_1. Similarly, if the gridlets in Q_2 submitted for execution do not complete in the fixed time quanta of Q_2 then those gridlets are pushed onto the next level queue Q_3. However, the gridlets present in Q_3 are executed based on FCFS scheduling policy. The shorter gridlets completes its execution quickly, without migrating to lower level queues. All gridlets gets an opportunity to execute and thus reduces starvation of gridlets by promoting the gridlets in lower queues to a higher priority.

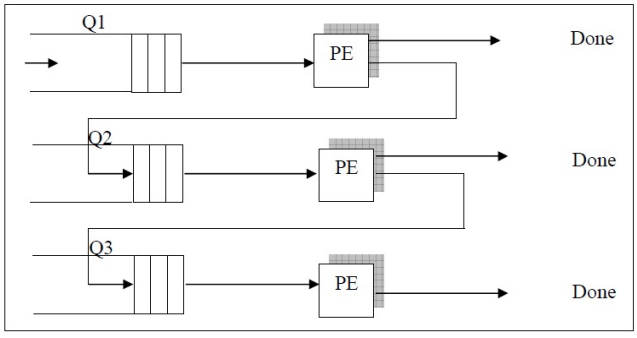

Figure 3.Multilevel Feedback Queue (MLFQ) Scheduling model

4.1 Algorithm

In this section, we present the pseudo code of the MLFQ scheduling Algorithm

1. ***Add_new_job()***
2. ***Repeat for*** *i=1 to inputqueue_size*
3. *Insert jobs into input queue*
4. ***end for***
5. ***SelectJob()***
6. *Move gridlets from input queue to ready queue*
7. ***Repeat for*** *i=1 to readyqueue_size*
8. *Get gridlet g_i from readyqueue*
9. ***Repeat for*** *i=1 to resourcelist_size*
10. *Get resource r_i from resourcelist r*
11. *Check for suitability of gridlet r_i with resource r_i*
12. ***If*** *suitable*
13. *Assign the gridlet g_i to the resource r_i*
14. ***Break****;*
15. ***End for***
16. *Submit gridlet g_i to Q_1 of the scheduler*
17. *Update the status of g_i as InExec*
18. ***If*** *gridlet g_i execution does not complete in Q_1 quanta*
19. ***Then*** *push gridlet g_i into Q_2*
20. ***Else***
21. *Terminate the gridlet in g_i*
22. ***Endif***
23. ***If*** *gridlet g_i execution does not complete in Q_2 quanta*
24. ***Then*** *push gridlet g_i into Q_3*
25. ***Else***
26. *Terminate the gridlet g_i*
27. ***Endif***
28. *Process each gridlet in Q_3 according to FCFS basis*
29. *Increment the scheduled gridlet*
30. *Decrement the remaining gridlet*
31. ***End for***

5 Simulation

In this section we show the performance of MLFQ scheduling technique through several experiments using Alea simulator, an extension of GridSim simulation toolkit. The experiment involved 5000 jobs that were executed on 14 clusters having 806 CPUs. We run the simulation by providing input data set and it completes all the jobs submitted to the grid over a span of time. These graphs shows the differences among the algorithms. Concerning the machine usage,as expected, FCFS generates very poor results[17].

FCFS is not able to utilize available resources when the first job in the queue requires some specific and currently unavailable machine(s). At this point, other more flexible jobs in the queue can be executed increasing the machine utilization. This is the main goal of the MLFQ algorithm. As we observe, MLFQ is able to increase the machine usage by shifting the jobs among the queues. Still, MLFQ will not allow to delay the execution of the first job in the queue, which restricts it from making more fair decisions that would increase the machine utilization. In case of the second criteria, similar reasons as in the previous example caused that PBS_PRO is not able to schedule jobs fluently, because higher priority jobs keep occupies resources generating huge peak of low priority waiting jobs during the time[17]. The resulting makespan of MLFQ is slightly much higher (by 50 days) than FCFS and PBS_PRO the average machine usage per day as depicted in Figure 4. MLFQ demonstrates the number of waiting and running jobs on an average against each day is depicted in Figure 5. MLFQ is capable of a higher resource utilization and reduction of the number of waiting jobs. The requested and available CPU usage per day is shown in Figure 6. Figure 7 presents the average machine usage per cluster. Simulation results show that there is a minimization of overall response time and waiting time for the gridlets.

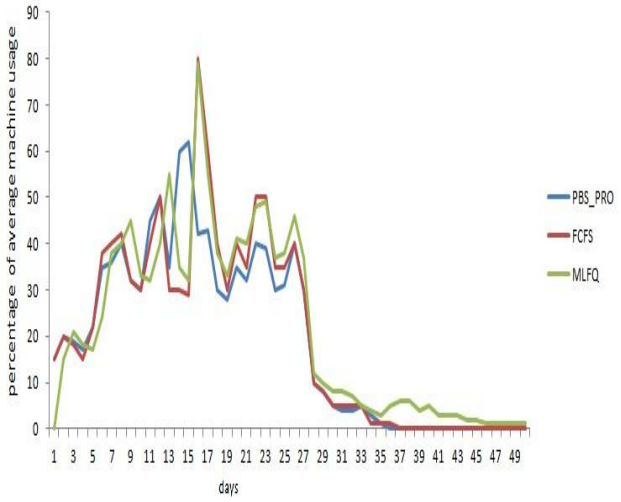

Figure 4. Percentage of average machine usage per day

Figure 4 describes a comparison between FCFS, PBS_PRO and MLFQ scheduling algorithms.
According to the graph, we observe that the MLFQ scheduling algorithm combines the best features of both the FCFS and PBS_PRO. As a case study, let's consider the 18th day.

Though PBS_PRO works better than FCFS in almost all instances, on the 18th day, when the load is pretty high, the PBS_PRO algorithm fails miserably whereas the FCFS algorithm performs in a much better way. As a second case study, consider the 15th day. Here, we observe that the PBS_PRO works in a much efficient way compared to FCFS. On the 15th day, though there was a large gridlet count, the time required by each gridlet was significantly low, hence, PBS_PRO proves itself to be more efficient than FCFS in this particular instance.

Our MLFQ algorithm combines the best properties of FCFS and PBS_PRO. From figure 4, we can observe that on the 18th day, MLFQ works with the same efficiency as FCFS. On the 15th day, though MLFQ doesn't drop to such an extent as that of FCFS, the efficiency is better than that of FCFS. Thus, it is proven from the test results that the MLFQ scheduling algorithm provides an optimum efficiency combining the feature of FCFS and PBS_PRO.

Figure 5, figure 6, and figure 7 provide us with a better understanding of the MLFQ scheduler.

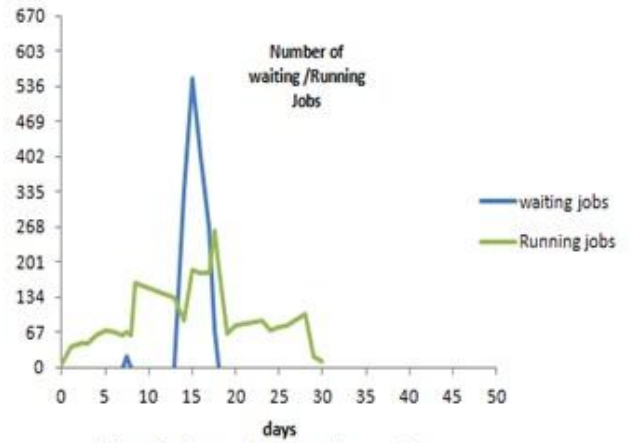

Figure 5. Average job execution per day

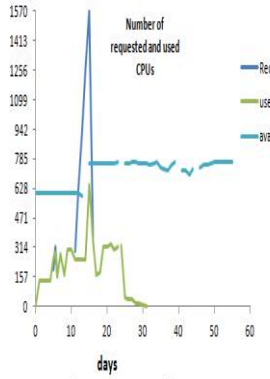

Figure 6. CPU usage per day

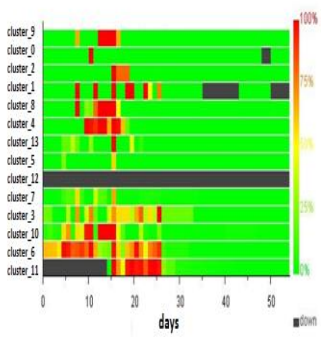

Figure 7.Percentage of cluster usage

6 Conclusions

The paper describes a new approach to schedule tasks efficiently in a grid environment.

We proposed a Multilevel Feedback Queue Scheduling (MLFQ) for Alea, a GridSim based simulator. The approach is based on processing capability of individual grid resources. Our policy provides a solution by implementing MLFQ scheduler where lower priority gridlets will complete quickly, without migrating to the lower levels of the hierarchy, due to which we are able to achieve high throughput and good response time by considering waiting and service times.Concerning the machine usage, as expected, FCFS generates very poor results. FCFS is not able to utilize available resources when the first job in the queue requires some specific and currently unavailable machine(s). At this point, other more flexible jobs in the queue can be executed increasing the machine utilization. This is the main goal of the MLFQ algorithm. As we observe, MLFQ is able to increase the machine usage by shifting the jobs among the queues. MLFQ will not allow to delay the execution of the first job in the queue, which restricts it from making more fair decisions that would increase the machine utilization. In case of the second criteria, similar reasons as in the previous example caused that PBS_PRO is not able to schedule jobs fluently, because higher priority jobs keep occupies resources generating huge peak of low priority waiting jobs during the time. The resulting

makespan of MLFQ yields better results. The transportation cost and overall communication delay and prices charged by the resource owners are obtained based on a pricing model is considered for future work .

7 References

[1]. I Foster, C Kesselman (2004) The Grid 2: Blueprint for a New Computing Infrastructure II Ed. Elsevier and Morgan Kaufmann Press.

[2]. W Hoschek et al (2000) Data Management in an International Data Grid Project. Proc. 1st International Workshop on Grid Computing (GRID Bangalore.

[3]. Buyya R, Steve Chapin S, DiNucci D (2000) Architectural Models for Resource Management in the Grid. IEEE/ACM International Workshop on Grid Computing.

[4]. L Mohammad Khanli, M Analoui (2008) Resource Scheduling in Desktop Grid by Grid-JQA The IEEE 3rd International Conference on Grid and Pervasive Computing.

[5]. L Mohammad Khanli, M Analoui (2007) Grid_JQA: A QoS Guided Scheduling Algorithm for Grid Computing The 6th IEEE International Symp on Parallel and Distributed Computing.

[6]. F. Dong et al (2006) A Grid Task Scheduling Algorithm Based on QoS Priority Grouping Proc. of the 5th IEEE International Conf on Grid and Cooperative Computing.

[7]. K. Etminani, M Naghibzadeh (2007) A Min-min Max-min Selective Algorithm for Grid Task Scheduling The 3rd IEEE/IFIP International Conf on Internet, Uzbekistan.

[8]. Hoganson, Kenneth (2009) Reducing MLFQ Scheduling Starvation with Feedback and Exponential Averaging Consortium for Computing Sciences in Colleges, Southeastern Conference, Georgia.

[9]. X. He, X-He Sun, G V Laszewski (2003) QoS Guided Min-min Heuristic for Grid Task Scheduling J Computer Science and Technology 18:442-451.

[10]. L Mohammad Khanli, and M Analoui (2008) Resource Scheduling in Desktop Grid by Grid-JQA The 3rd IEEE International Conf on Grid and Pervasive Computing.

[11]. L Mohammad Khanli, M Analoui (2007) Grid_JQA: A QoS Guided Scheduling Algorithm for Grid Computing The 6th IEEE International Symp on Parallel and Distributed Computing.

[12]. F Dong, J Luo, et al (2006) A Grid Task Scheduling Algorithm Based on QoS Priority Grouping Proc of 5th IEEE International Conf on Grid and Cooperative Computing.

[13]. E Ullah Munir, J Li, Sh Shi (2007) QoS Sufferage Heuristic for Independent Task Scheduling in Grid J Information Technology 6 (8):1166-1170.

[14]. K Etminani, and M Naghibzadeh(2007) A Min-min Max-min Selective Algorithm for Grid Task Scheduling 3rd IEEE/IFIP International Conf on Internet, Uzbekistan.

[15]. Riky Subrata, Albert Y. Zomaya, and Bjorn Landfeldt, "Game-Theoretic Approach for Load Balancing in Computational Grids" IEEE Transactions on parallel and Distributed Systems Vol.19.no.1 2008.

[16]. R. Wolski, N.T. Spring, and J. Hayes, "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing, " J. Future Generation Computer Systems, vol. 15,pp. 757-768, 1998.

[17]. Dalibor Klusáˇcek, Hana Rudová "Alea 2 – Job Scheduling Simulator " SIMUTools 2010 March 15–19, Torremolinos, Malaga, Spain.