Comparative Analysis of Some Grid Scheduling Algorithms on Real Computational Grid Environment

Haruna Ahmed Abba, Nordin B. Zakaria, Anindya. J. Pal

High Performance Computing Service Center (HPCC), Universiti Teknologi PETRONAS, Seri Iskandar, 31750 Tronoh, Perak, Malaysia
ahmadydee@gmail.com, nordinzakaria@petronas.com.my, anindyaip@gmail.com

Abstract
The development of computers as well as network technologies encompass us towards the world of high performance computing termed as Grid computing, which is the evolution of parallel processing. Grid scheduling is a vital component of computational grid environment. As regard to that, grid scheduling tend to be one of the major challenges of grid computing, yet an NP-complete problem. Moreover, some researchers have considered slack time using different techniques, but not yet considered operational research slack time concept by combining hard real time system and soft real time system techniques which will maximize the number of deadlines met, minimize tardiness of tasks, maximize the number of high priority tasks in meeting their deadlines and creating fairness. However, we propose two hybrid scheduling algorithms (Least Slack Time Rate First Round Robin Based Scheduling Algorithm (LSTRRGCD), Earliest Deadline First Round Robin Based Scheduling Algorithm (EDFRRGCD)), based on deadline, slack time using a dynamic time quantum, computed as the greatest common divisor (GCD) of all jobs burst time rate and also evaluate with First Come First Served Scheduling Algorithm (FCFS) and Round Robin Scheduling Algorithm (RR)) as baseline approaches in a real grid environment using real workload traces, taken from leading computational centers.

Keywords: Grid scheduling, Parallel Processing, LSTRFRRGCD, EDFRRGCD, FCFS, RR, GCD and Deadline.

1 Introduction
Grid computing originates early in the 1990’s like a mark to make computing resources easily accessible computer researcher Ian Promote [1] was promoting a program to consider shared computing to some global level. Just like the internet which is a tool for mass communication. Grids are a tool that provides computer resources and space for storage and so on using networks. it provides the infrastructure that can comprehend high processing capabilities. Grid scheduling is a vital component of computational grid environment.

In recent years, lots of research have been done by different researchers using various kind of techniques in different notations. Yet, some researchers have considered slack time using different techniques, but not yet considered operational research slack time concept by combining hard real time system and soft real time system techniques which will maximize the number of deadlines met, minimize tardiness of tasks, maximize the number of high priority tasks in meeting their deadlines and creating fairness.

However, two essential things were considered in the performance evaluation of our grid scheduling algorithms. First of all, we develop two hybrid scheduling algorithms (LSTRFRRGCD and EDFRRGCD) based on deadline, slack time and evaluate with baseline approaches (FCFS and RR) in a real grid environment. Secondly, for the comparison, we used a real workload trace called...
AuverGrid [2], as the majority of the scheduling algorithms highlighted as part of our literature has not been evaluated using real grid workload traces in a real grid environment.

The objectives of this paper is to evaluate the performance, scalability as well as fairness of the proposed hybrid grid scheduling algorithms, by comparing them with other baseline scheduling algorithms based on the following performance metrics: Average Turnaround Time, Average Waiting Time and Maximum Tardiness.

The rest of this paper is organized as follows. Section 2 overview on previous work in grid scheduling. Section 3 scheduling algorithms, Section 4 result and discussion of the proposed scheduling algorithms respectively. Section 5 conclusions.

2 Related Research

In the work of Rajkumar Buyya et al. [3], deadline and budget constraint (DBC) which allows allocation of resources depending on the user’s QoS requirements, such as the deadline, budget, and optimization strategy were proposed. The proposed algorithm called cost time optimization was developed and evaluated using the Grid Sim toolkit by comparing its performance and a lot of service delivery with the cost optimization. However, when there are multiple resources with the same cost and capability, the cost time optimization algorithm schedules jobs on them using the time optimization strategy for the deadline period.

Lui et at. [4], construct a novel Generalized Distributed Scheduler (GDS) for tasks with different priorities and deadlines. They considered a non-preemptive scheduling strategy applied over a bag of independent mixed tasks in computational grids. Furthermore, tasks are ranked based upon priority and deadline then got scheduled. Tasks are shuffled to earlier points to pack the schedule and create fault tolerance. However, dispatching is based upon task-resource matching and accounts for computation as well as communication capacities.

The work of Eddy et al. [5], was an extension of the paper: “A Study of Deadline Scheduling for Client-Server Systems on the Computational Grid” Takefusa et al. [6], considers both priority and deadline of the tasks to select a server. They showed that the number of tasks can meet their deadlines by the increase of 1) using task priorities and by 2) using a fallback mechanism to reschedule tasks that were not able to meet their deadline on the selected servers.

In the work of Alexis et al. [7], a simulation of the pure Repeated Placing Policy was derived, which does not ignore jobs and considered a job’s execution within a short period of time before the global deadline. However, considering the jobs too early may cause many jobs to fail; the first job, indeed, ran fine but wasted a lot of processor time, while the next ones do not have enough processors to run.

In the work of Akshaikumar et al. [8], a new resource characteristic based optimization method (RCBO), was combined with Earlier Gap, Earliest Deadline First (EG-EDF) policy [9] to schedule jobs in a dynamic environment. Moreover, at each time new jobs arrived, after the last application of RCBO, has reached the value of ten. RCBO is applied to change the positions of some jobs that have already arrived and are waiting in the schedules of some machines. RCBO may move the jobs to a better evaluated position.

In the work of Hwang et al [10], maximum occupation time (MOT) was defined, it limits the maximum time which one job (of a task) can occupy one processor. The MOT is determined by the timing constraints of a given task set. The aim of Least Slack Time Rate first (LSTR) is not to allow idle state in any processor. All tasks have the deadline and execution time as a timing constraint. The scheduler determines the task at the scheduled time to be executed on a processor. Tasks are executed on the processor(s) and then both the remaining execution time and the remaining deadline of these tasks decreases.
Vasumathi et al. [11], applied a technique which fills earliest existing gaps in the schedule with newly arriving jobs. If no gap for a coming job is available EG-EDF rule uses Earliest Deadline First (EDF) strategy for including new job into the existing schedule. Scheduling choices are taken to meet the Quality of Service (QoS) requested by the submitted jobs, and to optimize the usage of hardware resources.

Dalibor et al. [12], shows that combining redundant scheduling with deadline-based scheduling could lead to a fundamental tradeoff between throughput and fairness. However, came up with a new scheduling algorithm called Limited Resource Earliest Deadline (LRED) that couples redundant scheduling with deadline driven scheduling in a flexible way by using a simple tunable parameter to exploit this tradeoff.

3 Scheduling algorithms

Here, we described our proposed scheduling algorithms such as: FCFS, RR and EDFRRGCD as a baseline to compare and evaluate the performance of LSTRFRRGCD.

A. Baseline approaches

I. First Come First Served Scheduling algorithm (FCFS): Based on this algorithm dispatching processes are based on their arrival time on the ready queue. Once a process has a processor, it will keep running until it finished executing. Meanwhile, it will be terminated and then the next process will be dispatched from the ready queue.

II. Round Robin Scheduling Algorithm (RR): based on this algorithm, the ready queue is preserved as a first come first served (FIFO) queue. Dispatching processes is from the head of the ready queue for execution by the processor. The preemption of a process for execution is based on system defined variable, named as time quantum. However, as soon as a process execution is completed, Before its time quantum expired, it will be terminated as well as deleted from the system. Therefore, next process will be dispatched from the ready queue.

B. Proposed Job Scheduling Algorithms

I. Earliest Deadline First Round Robin Based Scheduling Algorithm (EDFRRGCD): Based on this algorithm, the earlier the deadline is, the higher the priority is. Allocation is carried out based on a master slave architecture. EDFRRGCD employs a round robin allocation strategy for jobs distribution among slave processors; and used on each slave processor for computation. Processes are dispatched based on minimum deadline on the ready queue. The preemption of a process for execution is based on system defined variable, named as time quantum. However, as soon as a process execution is completed, Before its time quantum expired, it will be terminated as well as deleted from the system and then the next job with a minimum deadline will be dispatched from the ready queue. Once a computation is done at slave processor, then the results are sent to the master processor.

II. Least Slack Time Rate First Round Robin Based Scheduling Algorithm (LSTRFRRGCD): LSTRFRRGCD executes the process based minimum slack time priority rate. Allocation is carried out based on a master slave architecture. LSTRFRRGCD employs a round robin allocation strategy for jobs distribution among slave processors; and used on each slave processor for computation. Processes are dispatched based on minimum slack time priority
rate on the ready queue. The preemption of a process for execution is based on system defined variable, named as time quantum. However, as soon as a process execution is completed, before its time quantum expired, it will be terminated as well as deleted from the system and then the next job with a least slack time rate will be dispatched from the ready queue. Once a computation is done at slave processor, then the results are sent to the master processor.

Before describing the actual algorithms few terminologies used in the algorithms must be described as follows:

Let us assume $J_i$: ith Job; $n$: number of jobs; $n_{si}$: number of slaves of job $i$; $x_i$: number of jobs per slave of job $i$; $TQ_i$: time quantum of job $i$; $T_i$: arrival time of job $i$; $d_i$: deadline of job $i$; $Ed$: minimum deadline of job $i$; $a_i$: burst time of job $i$; $C_i$: Job completion time of job $i$; $D_i$: absolute deadline time of job $i$; $T_{REi}$: remaining execution time of job $i$; $T_{RD}$: remaining absolute deadline time of job $i$; $Pr_i$: priority rate of job $i$; $GCD(a_1, a_n)$: greatest common divisor of overall burst time rate of job $i$; $T_{TRM}$: master turnaround time of job $i$; $T_{TRD}$: slave turnaround time of job $i$; $T_{WTM}$: master waiting time of job $i$; $T_{WTS}$: slave waiting time of job $i$; $R_{ET}$: starting execution time of job $i$; $S_{ET}$: ending execution time of job $i$; $S_{TE}$: slave total execution time of job $i$; $M_{TE}$: master total execution time of job $i$; $T_{CTi}$: total communication time of job $i$; $T_{Max_TRD}$: maximum tardiness; $S$-list: Sorted list;

I. Number of jobs per slave $x_i$: refers to the number of jobs per each slave for execution.

$$x_i = \frac{n}{n_{si}}$$

II. Time quantum $TQ_i$: referred to a dynamic time for each job to be executed, meaning as soon as a process execution is completed, before its time quantum expired, it will be terminated as well as deleted from the system. The next process will be dispatched from the head of the queue. However, it is computed based on burst time priority rate of jobs.

Time quantum, take $TQ_i = GCD(a_1, a_n)$

III. Minimum deadline time $Ed$: sorting jobs based on minimum deadline first.

$$Ed$$

IV. Absolute deadline: referred to the time a process execution should be finished.

$$D_i = (d_i + r_i)$$

V. Remaining execution time: referred to the time remain of a job in the process of execution.

$$T_{REi} = (a_i - r_i)$$

VI. Remaining absolute deadline: referred to the remaining deadline time of job in the process of execution.

$$T_{RD} = (d_i + r_i) - r_i$$

VII. Priority rate: it determines the priority of which job to be executed first in d ready queue.

$$Pr_i = \frac{T_{RD} - r_i}{T_{RD}}$$

VIII. Total communication time $T_{CTi}$: refers to total execution time taken for each master or slaves to finish its execution process:

$$M_{TE} = R_{ET} - S_{ET}$$

$$S_{TE} = R_{ET} - S_{ET}$$

Therefore: $T_{CT} = M_{TE} - S_{TE}$

IX. Turnaround time: Referred to the total time occupied between job submission for execution and the return of its completed result.
Turnaround time $T_{TRSi} = C_i - T_i$ ......................... (11)
Therefore: $T_{TRMi} = T_{TRSi} + T_{CTi}$ ..................(12)
Average turnaround time,
$$T_{Avg, TR} = \frac{\sum_{i=1}^{n} T_{TRMi}}{n}$$ ................................................. (13)

X. Waiting time: Referred to the total time waited for a job before its final execution.
Waiting time $T_{WTSi} = T_{TRSi} - \alpha_i$ ......................... (14)
Therefore: $T_{WTSi} = T_{TRSi} + T_{CTi}$ ..................(15)
Average waiting time,
$$T_{Avg, WT} = \frac{\sum_{i=1}^{n} T_{WTSi}}{n}$$ ................................................. (16)

XI. Maximum tardiness: Referred to the maximum time delayed between turnaround time and deadline time.
Tardiness, $T_{TRDSi} = d_i - T_{TRSi}$ ................................................ (17)
Therefore: $T_{TRDMi} = T_{TRDSi} + T_{CTMi}$ ..................(18)
Maximum Tardiness $T_{Max, TRD} = \text{Max} (T_{TRDM1}, T_{TRDM2}, \ldots, T_{TRDMn})$ ........ (19)

We have used master slave architecture, for testing the developed scheduling algorithms, as shown in Fig.1. This involves the use of an actual cluster. The master takes process as the input and distributes the processes on the cluster processors using a simple allocation strategy for parallel computation. Moreover, a real workload traces, AuverGrid [2], is used as input. The total number of jobs is divided by the number of processors, and that number of jobs are distributed to each slave where the scheduling algorithms are executed for computation.

![Figure 1: Master/Slave Architecture](image)

Each slave will receive job, described by its process ID, arrival time, burst time and deadline. It assigns a dynamic time quantum, computed as computed as the greatest common divisor (GCD) of all jobs burst time priority rate and then compute the value of priority rate for each job by sorting out the jobs on the basis of Earliest deadline first (criteria III) / Priority rate (criteria VII). If multiple jobs have same time value then, it will break the tie by selecting a job from job set based on FCFS. Dispatching processes is from the head of the ready queue for execution by the processor. The preemption of a process for execution is based on system defined variable, named as time quantum.

However, as soon as a process execution is completed, Before its time quantum expired, it will be terminated as well as deleted from the system. The next process is then dispatched from the head of the ready queue. This process will continue until the pool is empty. Turnaround time, waiting time and tardiness values are computed for each job and return to master.
Moreover, masters computes total turnaround time, total waiting time and total tardiness and then finally the average turnaround time, average waiting time and maximum tardiness value, to identify the maximum time delay of jobs execution.

5 Results and Discussion

Experiments were carried out using the facilities of high performance computing center (HPCC) at Universiti Teknologi PETRONAS using SGI Altixs 4700. The simulations of the algorithms have generated useful data that has been analyzed.

However, to check the performance of the proposed algorithms, i.e. LSTRFRRGCD, EDFRRGCD, FCFS and RR scheduling algorithm, we incorporate scalability test of scheduling algorithms by increasing number of processors by 100% from 32 to 64 and 128 processors respectively. Meaning, each process set has been given a time quantum for simulation. Each process is specified by its burst time, arrival time and deadline.

Moreover, performance metrics are based on the following factors - Average Turnaround Time, Average Waiting Time and Maximum tardiness.

Below is the graph derived from LSTRFRRGCD, EDFRRGCD, FCFS and RR Scheduling algorithms followed by a discussion. Figure 2 shows graphs of the Average Turnaround Times, Fig.3 Average Waiting Times, and Fig.4 Maximum tardiness, respectively.

![Figure 2: Average Turnaround Time](image)

a. Average Turnaround Time

This experiment was carried out varying the number of processors by 100% from 32, 64 and 128 numbers of processors. Figure 2 illustrate that LSTRFRRGCD, EDFRRGCD, FCFS and RR are smooth and steady from 32 to 64 and 128 processors. Results showed that LSTRFRRGCD has the best performance. While, FCFS showed the worst performance.
**b. Average waiting Time**

It can be observed that figure 3 has maintain the same flow of result. Where by, LSTRFRRGCD has shown and maintain the best performance. While, FCFS showed the worst performance.

**c. Maximum Tardiness**

It can be observed from figure 4 that LSTRFRRGCD, EDFRRGCD, FCFS and RR are sharp, as well as prominent to each other from 32 to 64 and 128 processors. However, there is a slight
difference between the algorithms. Results showed that RR has the best performance. While, EDFRRGCD showed the worst performance.

6 Conclusions and Future Work
According to our comparative performance analysis of the proposed scheduling algorithms, results has shown maintained performance under dynamic environment. Therefore, we came to conclusion that LSTRFRRGCD is a scheduling policy from the real grid computational point of observation; it gratify the real grid infrastructure requirements (Short Turnaround Time and Short Average Waiting Time) as well as maintain scalability under heavy workload and varying number of processors in a real grid environment.

In future, we will enhance and evaluate the proposed hybrid scheduling algorithms based on the derived results. Moreover, we will perform detailed comparative performance analysis with other scheduling approaches.

Acknowledgements
Our gratitude to all HPCC members, structural equation modeling for team leader high performance call center (FRGS) for their support and to AuverGrid team as well as the Grid Workloads Archive for their contribution, by making the traces available.

References