Abstract

A ‘Grid’ is an infrastructure for resource sharing. It is used for large-scale data processing, many of the applications being scientific ones. Grid scheduling is a vital component of a Grid infrastructure. Reliability, efficiency (in terms of time consumption) and effectiveness in resource utilization are the desired characteristics of Grid scheduling systems. Many algorithms have been developed for Grid scheduling. In our previous work, we proposed two scheduling algorithms (the Multilevel Hybrid Scheduling Algorithm and the Multilevel Dual Queue Scheduling Algorithm) for optimum utilization of processors in a Grid computing environment. In this paper, we propose two more flavours of Multilevel Hybrid scheduling algorithms; i.e. the Dynamic Multilevel Hybrid Scheduling Algorithm using Median and the Dynamic Multilevel Hybrid Scheduling Algorithm using Square root. We evaluate our proposed Grid scheduling using real workload traces, taken from leading computational centers. The main idea of the proposed algorithms is to execute jobs optimally, i.e. with minimum average waiting, turnaround and response times. An extensive performance comparison is presented using real workload traces to evaluate the efficiency of scheduling algorithms. To facilitate the research, a software tool has been developed which produces a comprehensive simulation of a number of Grid scheduling algorithms. The tool’s output is in the form of scheduling performance metrics.

The experimental results, based on performance metrics, demonstrate that the performances of our Grid scheduling algorithms give good results. Our proposed scheduling algorithms also support true scalability, that is, they maintain an efficient approach when increasing the number of processors or nodes. This paper also includes a statistical analysis of real workload traces to present the nature and behavior of jobs.

Our proposed scheduling algorithms are unique. They have three key features. First, they favor the shortest job for execution. Second, they execute the job on the basis of a dynamic time quantum, to fairly distribute processor time among Grid jobs. A third feature is that they always execute the longest job, thus avoiding starvation.

Keywords: Distributed systems; Cluster; Grid computing; Grid scheduling; Workload modeling; Performance evaluation; Simulation; Load balancing; Task synchronization; Parallel processing.

* Corresponding author. Tel.: +60-125936195
E-mail address: nasirnsyed.utp@gmail.com

1877–0509 © 2011 Published by Elsevier Ltd. Selection and/or peer-review under responsibility of Prof. Mitsuhisa Sato and Prof. Satoshi Matsuoka
1. Introduction

‘Scheduling’ is described by the Grid Scheduling Dictionary Project as follows: “The process of ordering tasks on compute resources and ordering communication between tasks. Also, known as the allocation of computation and communication over time” [1].

There are three main phases of Grid scheduling. Phase one is resource discovery, which provides a list of available resources. Phase two is resource allocation, which involves the selection of feasible resources and the mapping of jobs to the resources. The third phase includes job execution. In the second phase, the selection of the best match of jobs to resources is an NP-complete problem [2].

In our previous research work, we proposed two Grid scheduling algorithms (the Multilevel Hybrid Scheduling Algorithm (MH) and the Multilevel Dual Queue Scheduling Algorithm (MDQ) with a view to improved performance [3]. The performance of these scheduling algorithms is based on the value of a fixed time quantum. If the value of the time quantum is too small then MH results in too many context switches. If the value of the time quantum is too large then MH also loses its efficiency and behaves like the First Come First Served Scheduling Algorithm (FCFS). In this paper we present a solution to the fixed time quantum problem by proposing two more flavours of the Multilevel Hybrid Scheduling Algorithm (namely the Dynamic Multilevel Hybrid Scheduling Algorithm using Median (MHM) and the Dynamic Multilevel Hybrid Scheduling Algorithm using Square root (MHR)).

Scalability testing is a significant success factor for the design and development of a Grid scheduling algorithm. Scalability means a measure of optimizing the application to use more processing power given to it in the form of additional processors or cores. Grid vendors often refer to scalability as a measure of parallelizing an application across different machines. In [4] the authors defined the concept of performance and scalability as "The terms ‘performance’ and ‘scalability’ are commonly used interchangeably, but the two are distinct: performance measures the speed with which a single request can be executed, while scalability measures the ability of a request to maintain its performance under increasing load."

Two fundamental issues have to be considered for the performance evaluation of new Grid scheduling algorithms. Firstly, representative workload traces are required to produce dependable results. Secondly, a good testing environment should be set up, which is most commonly done by simulations [5].

Grid scheduling presents several challenges that make the implementation of practical systems a very difficult problem. Our research aims to design and develop Grid scheduling algorithms that makes efficient utilization of resources, maintain a high level of performance and possess a high degree of scalability.

The structure of the paper will now be described. Section 2 is a literature review of Grid scheduling methodologies. Section 3 presents the proposed scheduling algorithms and section 4 is about the statistical analysis of real workload traces. Section 5 shows the homogenous implementation of new scheduling algorithms. In section 6, the scheduling simulator’s design and development are discussed. Section 7 shows the experimental setup and section 8 describes the performance analysis of the Grid scheduling algorithms. Section 9 concludes the paper.

2. Related Research

A Grid is a high performance computational system which consists of a large number of distributed and heterogeneous resources. Grid computing enables sharing, selection and aggregation of resources to solve the complex large scale problems in science, engineering and commerce. Scientific applications usually consist of numerous jobs that process and generate large datasets. Processing complex scientific applications in a Grid imposes many challenges due to the large number of jobs, file transfers and the storage needed to process them. The scheduling of jobs focuses on mapping and managing the execution of tasks on shared resources [6].

Most of the parallel jobs demand a fixed number of processors, which cannot be changed during execution [7]. Good job scheduling policies are very essential to manage Grid systems in a more efficient and productive way [8].

Grid job scheduling policies can be generally divided into space-sharing and time-sharing approaches. In time-sharing policies, processors are temporally shared by jobs. In space-sharing policies, however, processors are exclusively allocated to a single job until its completion. The well known space-sharing policies are FCFS, Job Rotate Scheduling Policy (JR), Multilevel Opportunistic Feedback ((MOF)), Shortest Job First (SJF), Shortest Remaining Time First (SRTF), Longest Job First (LJF), Priority (P) and Non Preemptive Priority (P-NP).
approaches. The famous time-sharing scheduling policies are Round Robin (RR) and Proportional Local Round Robin Scheduling [9, 10, 11].

In [9] the authors have extended the working of basic space sharing techniques like FCFS, SJF, and LJF and proposed an SJF-backfilled scheduling heuristic. The main theme of this research was to backfill the shortest job first (length) to reduce the job killing probability. The proposed method also considers the reservation order of jobs in making the scheduling decisions. In this way, the authors have achieved the advantages of both the backfilling and the SJF scheduling policies.

In [11] the author has performed an experimental performance analysis of three space-sharing policies (namely FCFS, JR and MOF) and two time-sharing policies (namely Global Round Robin and Proportional Local Round Robin Scheduling) that have been developed for Grid computing. It is concluded that time-sharing scheduling policies perform better than space-sharing scheduling policies.

In [12] the authors have performed an analysis of processor scheduling algorithms using simulation of a Grid computing environment. Three space-sharing scheduling algorithms (namely FCFS, SJF and P) have been considered for simulation.

[13] proposes Grid level resource scheduling with a Job Grouping strategy in order to maximize the resource utilization and minimize the processing time of jobs. A combination of the Best Fit and RR scheduling policies is applied at the local level to achieve better performance. With RR, a fixed time quantum is given to each process that is present in the circular queue, for fair distribution of CPU times. The RR scheduling policy is extensively used for job scheduling in Grid computing [11, 13, 14].

In [15] the authors proposed three performance based scheduling algorithms, namely Deadline Sort (DS), Deadline Sort with Node Limitation (DS-NL), and Genetic algorithm (GA). They evaluated the proposed algorithms and the FCFS algorithm using a simulation. GA showed good results as compared to the other ones. GA can be applied to the scheduling of Grid tasks. However, the scheduling process of GA and its prediction process will use a lot of computing resources.

In [16] the author introduced a dynamic scheduling model for parallel machines, from an implementation perspective. The proposed model of a parallel job is based on a penalty factor. This paper also addresses open issues for the researchers. First, the theoretical and experimental analysis of the idle regulation is needed with more variations of job scheduling strategies (largest job first, backfill, etc.) and optimization criteria, from both a user and a system perspective. Second, what is needed is the analysis of the system in a practical scheduling environment that supports dependent jobs, and jobs that can arrive at any moment.

The evaluation of job scheduling algorithms should be based on two things, first, the use of appropriate metrics, and second, the use of an appropriate workload on which the scheduler should operate. A standard workload should be used as a benchmark for scheduling algorithms [17].

The workload plays a significant role in the experimental performance evaluation of computer systems. In [18] the authors also emphasized that scheduling algorithms and policies should be designed and evaluated at both the local cluster and Grid level. Most of the scheduling algorithms highlighted in the literature have not been evaluated using real workload traces. The aim of this paper is to propose new Grid scheduling algorithms and to evaluate their performance and scalability in comparison to other well known Grid scheduling algorithms by simulation using real workload traces. Our scheduling performance metrics include three significant key factors, namely average waiting time, average turnaround time and average response time.

3. Proposed Scheduling Algorithms

In [3, 19, 20] we proposed two scheduling algorithms- MH and MDQ. They are based on a fixed time quantum. In this paper we propose two new Dynamic Multilevel Hybrid scheduling algorithms namely MHM and MHR. MH, MHM and MHR will now be described.

3.1. Multilevel Hybrid Scheduling Algorithm (MH)

MH is based on a master/slave architecture as shown in Figure 1. MH uses the RR allocation strategy for job distribution among the slave processors; and the Hybrid Scheduling Algorithm (H) is used on each slave processor for computation.
For H the ready queue is maintained in order of CPU burst length, with the least burst length at the head of the queue. Two numbers are maintained. The first number, $t_{\text{large}}$, shows the burst length of the largest process in the ready queue while the second one, $t_{\text{exec}}$, represents a running total of the execution time of all processes (since a reset was made). A new process submitted to the system is linked to the queue in accordance with its CPU burst length. The process state diagram of H is shown in Figure 2.

H dispatches processes from the head of the ready queue for execution by the CPU. Processes being executed are preempted on expiry of a time quantum, which is a system-defined variable. Following preemption, $t_{\text{exec}}$ is updated as follows:

$$t_{\text{exec}} = t_{\text{exec}} + \text{quantum}$$

The numbers are then compared. If

$$t_{\text{exec}} < t_{\text{large}}$$

then the preempted process is linked to the tail of the ready queue. The next process is then dispatched from the head of the ready queue. If

$$t_{\text{exec}} \geq t_{\text{large}}$$

then the process with the largest CPU burst length is given a turn for execution. Upon preemption, the ready queue is sorted on the basis of SJF.

The value of $t_{\text{large}}$ is reset to the burst length of the largest PCB, which is lying at the tail of the queue, and $t_{\text{exec}}$ is reset to 0. The next process is then dispatched from the head of the ready queue.

When a process has completed its task it terminates and is deleted from the system. $t_{\text{exec}}$ is updated as follows:

$$t_{\text{exec}} = t_{\text{exec}} + \text{time to complete}$$

The numbers are then compared and the actions taken are the same as those for a preempted process.

3.2. Dynamic Multilevel Hybrid Scheduling Algorithm using Median (MHM)

Our proposed MHM algorithm is a variant of MH. MHM works in the same way as the MH but uses a dynamic time quantum approach instead of fixed time quantum. MHM computes the dynamic time quantum using the median of CPU times of processes in the ready queue. We used the dynamic time quantum approach as detailed in [21].

$$\text{Time} - \text{Quantum} = \text{median}(C_1, C_2, C_3, ..., C_n)$$

where $C_i$ is the CPU time of Process $i$ and $i$ ranges from 1 to n.

3.3. Dynamic Multilevel Hybrid Scheduling Algorithm using Square root (MHR)

Our proposed MHR algorithm is another flavour of MH. MHR calculates the dynamic time quantum using the square root of the average of CPU times of processes in the ready queue. MHR computes the time quantum for each round and executes processes for the computed dynamic time quantum value. This approach also reduces the number of context switches in the system.

$$\text{Time} - \text{Quantum} = \sqrt{\text{avg}(C_1, C_2, C_3, ..., C_n)}$$

where $C_i$ is the CPU time of Process $i$ and $i$ ranges from 1 to n.

Our proposed dynamic scheduling algorithms (MHM and MHR) radically solve the fixed time quantum problem encountered in the MH.
4. Statistical Analysis of Real Workload Traces

In [22] a comprehensive statistical analysis has been carried out for a variety of workload traces on clusters and Grids. We reproduced the graphs of [22] to study the behaviour of the dynamic nature of workload ‘LCG1’ [23], using MS Excel. The total numbers of jobs in LCG1 is 188041. We looked at the number of jobs arriving in each 64 second period. The number of jobs arriving in a particular period is its ‘job count’. Figure 3 shows the distribution of job counts and run time demand for the whole trace. Next we performed an autocorrelation of the job counts at different lags. The left hand graph of figure 4 shows the autocorrelation plot for 800 lags - values from 0 to 799. Then we performed a Fourier analysis by applying the FFT on the 800 values of the autocorrelation output. This is shown in the right hand graph of figure 4.

Figures 3 and 4 depict that job arrivals show a diversity of correlation structures, including short range dependence, pseudo periodicity, and long range dependence. Long range dependencies can results in a large performance degradation whose effects should be taken into consideration for evaluation of scheduling algorithms. Real Grid workload LCG1 is shown to have rich correlation and scaling behaviour, which are different from conventional the parallel workload and cannot be captured by simple models such as Poisson or other distribution based methods [22]. Self similarity and long range dependency are the characteristics of LCG1 jobs.

LCG1 will play a key role in the performance evaluation of our proposed scheduling algorithms in comparison to other well known Grid scheduling algorithms.

![Fig. 3. The sequence plot and run time demand for the count process of LCG1](image1)

![Fig. 4. The autocorrelation function(ACF) and Discrete Fourier transformation(DFT) for the count process of LCG1](image2)
5. Homogeneous Implementation of Proposed Scheduling Algorithms

We used a master/slave architecture for implementation of our proposed algorithms, as shown in Fig.5. One processor is dedicated as the master processor among the cluster nodes. The master processor is responsible for distribution of the workload among the slave processors using round robin allocation strategy (i.e. 1, 2, 3…, n, 1) for parallel computation.

![Block diagram of master/slave architecture](image)

The same algorithm, either MHM or MHR, is used on each slave processor. Once computation is complete, the results are sent to the master processor.

6. Scheduling Simulator Design and Development

In this paper we used the same development strategy as we discussed in [3]. For comprehensiveness, the development strategy will now be explained.

The MPJ-express is widely used Java message passing library that allows writing and executing parallel applications for distributed and multicore systems. We developed a Java based simulator using MPJ-express API to evaluate the efficiency of our proposed scheduling algorithms. The metadata for each process includes its ID, its arrival time, its CPU time and the number of slaves that the job is to be divided between. The simulation software encounters the arrival time for each process and then submits processes to the system. The software has two main programs. One program runs on the master node (SimM). The other program runs on each slave processor (SimS). SimM accepts a workload and distributes among slave processors using RR. SimM receives notification from each slave processor for each job (or part of a job) that has finished. Each slave runs SimS and computes the average waiting time, the average turnaround time and the average response times. SimS processes the metadata for the list of processes that have been assigned to it. Upon completion of a process, SimM is informed. No ‘useful’ work is done by a slave other than that associated with scheduling. SimS keeps a detailed record of the processes being run on the slave - process ID; submit time; CPU time; time quantum.

All slaves use the same scheduling algorithm, which is input by the user of SimM. The user can select one of a range of algorithms including the newly developed ones, MHM, MHR, MH and MDQ, as well as established ones, FCFS, SJF, SRTF, RR and P. The purpose of the simulator is to produce a comparative performance analysis of scheduling algorithms.

7. Experimental Setup

The experiments made use of a HPC facility in the High Performance Computing Centre at Universiti Teknologi PETRONAS. We ran our experiment using a cluster of 128 processors. The ‘hpc.local’ was used as the default execution site for job submission. A detailed experimental setup is shown in Table I.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Location</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>gillani</td>
<td>Shell terminal</td>
<td>Lab</td>
<td>Intel Core 2 Duo CPU 2.0GHz</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Workstation</td>
<td>2 GB Memory</td>
</tr>
<tr>
<td>hpc.local</td>
<td>Execution site</td>
<td>HPC facility</td>
<td>128 Core Intel(R) Itanium2(R) Processor 9030</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>arch : IA-64</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CPU MHz : 1.6 GHz</td>
</tr>
</tbody>
</table>

Table 1. Experimental Setup
8. Performance Analysis of Grid Scheduling Algorithms

Performance metrics for the Grid scheduling algorithms are based on three factors - Average Waiting Time, Average Turnaround Time, and Average Response Time. We performed experiments for different scheduling algorithms using LCG1. Our experiments include the scalability test of scheduling algorithms under an increasing real workload. We formed three data sets by using first 3%, 5%, and 10% of the LCG1 workload i.e. 6000, 9402, and 18804 processes, respectively. The ‘runtime’ attribute is given for each process in ‘LCG1’. The ‘runtime’ is taken as CPU time in our experiment. We performed an experiment by varying the number of CPUs from 16 to 128. We used ‘5’ units as the fixed time quantum for our experiment. In this section, we describe a comparative performance analysis of our proposed algorithms, i.e. MHM and MHR, with six other Grid scheduling algorithms; i.e. FCFS, SJF, SRTF, RR, MH and MDQ.

8.1. Average Waiting Times Analysis

Figure 6 shows that the average waiting times computed by each scheduling algorithm for each real workload trace. Figure 6 illustrates that the SRTF, MH and MHR scheduling algorithms produce the shortest average waiting times as compared to the other scheduling algorithms. The average waiting time computed for SRTF is slightly shorter than the value computed for the MH and MHR scheduling algorithms. By increasing the number of CPUs, each algorithm shows the relative improvement in performance, except for the MHRM algorithm. MHRM shows better results as compared to the MDQ, RR and FCFS algorithms. All scheduling algorithms, with the exception of MHRM, show that the relative performance is independent of the nature of the workload, the workload size and the number of CPUs used for computation.

8.2. Average Turnaround Times Analysis

Figure 7 presents the pictorial view of the average turnaround times computed for the scheduling algorithms using real workload traces. Figure 7 illustrates that the average turnaround time computed by the SRTF, MH and MHR scheduling algorithms are shorter than the other Grid scheduling algorithms. By increasing the number of CPUs, each algorithm has an improved average turnaround time, except for the MHM scheduling algorithm.

Experimental results show that SRTF, MH and MHR are at the same performance level as regards average turnaround time. Figure 7 also shows that the average turnaround times computed for MHRM are slightly longer than those for the MHR and SJF scheduling algorithms but better than the values computed for the MDQ, RR and FCFS scheduling algorithms. Moreover, all scheduling algorithms, with the exception of MHRM, show that relative performance is independent of the nature of the workload, the workload size and the number of CPUs used in the experiment.

8.3. Average Response Times Analysis

Figure 8 shows that RR, MDQ, MH and MHR produce the shortest average response times as compared to the other scheduling algorithms. The average response times computed for MDQ are slightly longer than those for RR and slightly shorter than those for MH and MHR. The SJF and SRTF scheduling algorithms result in poor response times as compared to the other scheduling algorithms. All scheduling algorithms, except for MHM, show that the relative performance is independent of the nature of the workload, the workload size and the number of CPUs. MDQ gives consistently good results for different workloads and numbers of CPUs.

Table 2 shows the average performance measure for each algorithm, running 18804 processes on 128 CPUs.

<p>| Table 2. Performance Analysis for Scheduling Algorithms using 18804 processes (part of LCG1) and 128 processors |
|---------------------------------------------------|------------------|------------------|------------------|</p>
<table>
<thead>
<tr>
<th>Scheduling Algorithm</th>
<th>Average Waiting Times (seconds)</th>
<th>Average Turnaround Times (seconds)</th>
<th>Average Response Times (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRTF</td>
<td>3003063.02</td>
<td>3700388.26</td>
<td>2839722.35</td>
</tr>
<tr>
<td>MH</td>
<td>3006800.71</td>
<td>3704125.95</td>
<td>2424819.42</td>
</tr>
<tr>
<td>MHR</td>
<td>3015282.76</td>
<td>3712607.99</td>
<td>2431736.22</td>
</tr>
<tr>
<td>SJF</td>
<td>3931193.44</td>
<td>4628518.67</td>
<td>3931193.44</td>
</tr>
<tr>
<td>MHRM</td>
<td>4328777.08</td>
<td>5026102.31</td>
<td>3969414.97</td>
</tr>
<tr>
<td>MDQ</td>
<td>6938822.31</td>
<td>7636147.54</td>
<td>91929.73</td>
</tr>
<tr>
<td>RR</td>
<td>6950589.30</td>
<td>7647914.53</td>
<td>57669.57</td>
</tr>
<tr>
<td>FCFS</td>
<td>39153756.02</td>
<td>39851081.25</td>
<td>39153756.02</td>
</tr>
</tbody>
</table>
Fig. 6. Average Waiting Time Analysis for 6000, 9402 and 18804 Processes of LCG1

Fig. 7. Average Turnaround Time Analysis for 6000, 9402 and 18804 Processes of LCG1

Fig. 8. Average Response Time Analysis for 6000, 9402 and 18804 Processes of LCG1
8.4. Performance Analysis of Scheduling Algorithms by Changing Time Quantum

The RR, MH and MDQ scheduling algorithms work on a fixed time quantum value. If the value of the time quantum is very small, then these scheduling algorithms result in better average response times but produce many context switches. If the value of the time quantum is very high, then their efficiency is also degraded. Our proposed MHM and MHR algorithms use a dynamic time quantum strategy instead of a static one and maintain system performance. The value of the time quantum is computed at runtime by considering the size of each job and the total number of jobs in the system. In the experiment we computed results for each algorithm using a workload of 18804 processes on 64 processors and varying the time quantum from 10 to 5000 as shown in Figure 9.

Figure 9 shows that all scheduling algorithms show stable performance measures (average waiting time, average turnaround time and response time) on varying time quantum, except the RR, MH and MDQ scheduling algorithms. It is apparent from the charts that our proposed scheduling algorithms (MH and MHR) markedly outperform the other Grid scheduling algorithms. A significant improvement is achieved in all of the performance parameters.

9. Conclusion

In this paper we present two new dynamic multilevel scheduling algorithms, namely MHM and MHR. Our proposed algorithms compute the time quantum dynamically. We have evaluated these algorithms on a simulator running on a cluster using a wide range of CPUs. We compared the efficiency of our algorithms with eight other Grid scheduling algorithms using real workload traces. In this paper we also performed a statistical analysis of the LCG1 workload trace to study the dynamic nature of jobs.

Experimental results show that the MH, MHR and SRTF scheduling algorithms are at the same performance level in producing the shortest average waiting times and average turnaround times for a variety of workloads. Experimental results also exhibit that MH and MHR produce shorter average response times in comparison to the SRTF scheduling algorithm. MHR shows better performance than MH because its performance is not affected by the value of a fixed time quantum.

We can say that MH and MHR are scheduling policies from the system point of view; they satisfy the system requirements (i.e. shorter Average Waiting Time and shorter Turnaround Time). The MHR scheduling algorithm always produces the best performance measures for real workload traces and for a wide range of processors. MDQ works well from the user perspective due to its short Average Response Time. Moreover, MH, MHR, MHM and MDQ are scalable, i.e. the relationship between each performance measure (e.g. average waiting time) and the workload size is very nearly linear.
Acknowledgements

We are grateful to the High Performance Computing Centre at University Teknologi PETRONAS for providing the HPC facility to perform the experiments. We want to express our gratitude to Dr. M. Fadzil Hassan, Dr. M Nordin B Zakaria, Dani Adhipta and Thayalan Sandran from University Teknologi PETRONAS and Dr Aamir Shafi from Massachusetts Institute of Technology (MIT) for their help during the research. We are also thankful to the University of Reading (UK) for providing the MPJ Express API. We thank the HEP e-Science Group at Imperial College London who provided the LCG data. We also thank Hui Li, the Parallel Workload Archive and the Grid Workloads Archive for their contribution in making the data publicly available.

References

7. E. Shmueli and D.G. Feitelson, Backfilling with look ahead to optimize the packing of parallel jobs. J Parallel Distrib Comput, vol. 65, no. 9, pp. 1090–1107. ISSN 0743-7315, 2005