Available online at www.sciencedirect.com

SciVerse ScienceDirect

AASRI Procedia 1 (2012) 531 – 543

AASRI Procedia

www.elsevier.com/locate/procedia

AASRI Conference on Computational Intelligence and Bioinformatics, CIB 2012

Design and Evaluation of Agent Based Prioritized Dynamic Round Robin Scheduling Algorithm on Computational Grids

Syed Nasir Mehmood Shah[a,*], M Nordin B Zakaria[a], Nazleeni Haron[a], Ahmad Kamil Bin Mahmood[a], Ken Naono[b]

[a] *Department of Computer and Information Sciences*
Universiti Teknologi PETRONAS, Bandar Seri Iskandar, 31750 Tronoh, Perak, Malaysia
[b] *R&D Center, Hitachi Asia Ltd. Malaysia*

Abstract

Grid computing enables sharing, selection and aggregation of computing resources for solving complex and large-scale scientific problems. Grid scheduling is playing a vital role for the efficient and effective execution of jobs on computational grids. Most of the scheduling algorithms do not consider user and system objectives at the same time. Therefore, in this paper we introduce a concept of fairness to scheduling and present a new agent based job scheduling algorithm called Agent based Prioritized Dynamic Round Robin (APDRR). APDRR is designed and developed by combining the best features of round robin and priority job scheduling algorithm using agent technology.
APDRR is fair scheduling algorithm from a user point of view while also regarding the optimization criteria that are anticipated from the system perspective. This paper also presents the comparative performance analysis of our proposed APDRR along with other well known job scheduling algorithms, considering the performance metrics comprised of average waiting time, average turnaround time, average response time, total completion time, average bounded slowdown time and maximum job stretch time. Performance evaluation of job scheduling algorithms has been carried out on a computational grid using real workload traces. Experimental evaluation confirmed that the proposed APDRR scheduling algorithm posses a high degree of optimality in performance, efficiency and scalability. This paper also includes a statistical analysis of real workload traces to present the nature and behavior of jobs.

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

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

doi:10.1016/j.aasri.2012.06.083

1. Introduction

Computational grid has the potential for solving large-scale scientific problems using geographically distributed and heterogeneous resources. Grid scheduling is a vital component of a computational grid infrastructure, which plays an important role in the efficient and effective execution of various kinds of scientific and engineering applications [1, 2].

Grid scheduling presents several interesting challenges that make the implementation of practical systems a very difficult problem. Job scheduling policies not only can manage the various computational resources needed in computing, but also can make the decisions regarding the dynamic, efficient and effective execution of jobs. Optimization of the grid performance is dependent on the scheduling policies [1, 3, 4, 5, 11]. In order to obtain a grid environment, which works with high computational power, the effective and efficient resource management is a must. Due to the high heterogeneity, scalability and dynamicity of a grid, some challenges then arise in the development of algorithms for scheduling to be used along with grid computing [6].

In this paper, job scheduling algorithms have been studied extensively and a new agent based prioritized dynamic round robin job scheduling algorithm (APDRR) has been proposed. The proposed scheduling algorithm has shown its optimal performance compared to the existing ones on an experimental computational grid using real workload traces. Apart from the development of these algorithms, software has been developed to facilitate the study with a greater ease and more user friendly manner.

This paper presents the comparative performance analysis of our proposed job scheduling algorithm with other well known scheduling approaches; e.g.; First Come First Served (FCFS), Round Robin (RR), Proportional Local Round Robin (PLRR), Shortest Process Next (SPN), Priority (P) and Longest Job First (LJF). We evaluated the efficiency, performance and scalability of each scheduling algorithm on a computational grid using six key performance parameters, i.e., average waiting time, average turnaround time, average response time, average bounded slowdown times, total completion time and maximum job stretch times.

The structure of the paper will now be described. Section 2 is a literature review of grid scheduling methodologies. Section 3 presents the proposed grid scheduling algorithm and section 4 is about the statistical analysis of real workload traces. Section 5 shows the homogenous implementation of 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

Scheduling applications on computational grid has proved to be a challenging task Job scheduling is playing a vital role in an efficient grid resource management. Good job scheduling policies are very essential to manage grid computational resources more efficiently and productively [1, 2].

Grid job scheduling policies can generally be divided into space-sharing and time-sharing approaches. In time-sharing policies, processors are temporally shared by jobs. In space-sharing policies, conversely, processors are exclusively allocated to a single job until its completion. The well known space-sharing policies are FCFS, Backfilling, Job Rotate Scheduling Policy (JR), Multilevel Opportunistic Feedback (MOF), Shortest Process Next (SPN), Shortest Remaining Time First (SRTF), Longest Job First (LJF) and Priority (P) approaches. The famous time-sharing scheduling policies on the other hand are Round Robin (RR) and Proportional Local Round Robin Scheduling (PLRR) [7, 8, 9].

The FCFS is the simplest and non preemptive job scheduling algorithm. For this algorithm the ready queue is maintained as a FIFO queue. Each new job/process is added to the tail of the ready queue and then the algorithm dispatches processes from the head of the ready queue for execution by the CPU. A process

terminates and is deleted from the system after completing its task. The next process is then selected from the head of the ready queue [10, 11].

The SPN algorithm takes the processes using the shortest CPU time first. For this algorithm the ready queue is maintained in order of CPU burst length with the shortest CPU demand at the head of the queue [11]. While the LJF algorithm takes the processes that use the longest CPU time first. For this algorithm the ready queue is maintained in order of CPU demands (runtime) in descending order [11, 12].

Round-robin scheduling [11, 13] is a simple way of scheduling in which all processes form a circular array and the scheduler gives control to each process at a time. The ready queue for this algorithm is maintained as a FIFO queue. A process submitted to the system is linked to the tail of the queue. The algorithm 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. A preempted process is linked to the tail of the ready queue. When a process has completed its task, i.e., before the expiry of the time quantum, it terminates and is deleted from the system. The next process is then dispatched from the head of the ready queue. This algorithm produces a good response time as compared to other scheduling algorithms.

In the priority (P) scheduling algorithm; the processes are prioritized in accordance with their operational significance. For this algorithm, the ready queue is maintained in the order of the system-defined priorities. Every process is assigned a priority and a new process submitted to the system is linked to the process in the ready queue having the same or a higher priority. The algorithm dispatches processes from the head of the ready queue for execution by the CPU. When a process has completed its task, it terminates and is deleted from the system. The next process afterward is dispatched from the head of the ready queue. If the priority criterion for execution is the order of arrival of the jobs into the system, then the priority scheduling behaves like FCFS scheduling. Alternatively, if the priority criterion is such that the jobs with shorter CPU demands are assigned higher priorities, then this makes the priority scheduling behave like SPN scheduling [11].

Several scheduling policies have been implemented in computational grids for high performance computing. The first come first serve (FCFS) with backfilling [14, 15] is the most commonly used; as on average, a good utilization of the system and good response times of the jobs are achieved. However, with certain job characteristics, other scheduling policies might be superior to FCFS. For example, for mostly long running jobs, the longest job first (LJF) is beneficial, while the shortest process next (SPN) is used with mostly short jobs [16].

In [7], the author has performed an experimental performance analysis of three space-sharing policies (FCFS, JR and MOF) and two time-sharing policies (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. The RR scheduling policy is extensively used for job scheduling in grid computing [7, 17]. In [18], the authors have performed an analysis of the processor scheduling algorithms using a simulation of a grid computing environment. Three space-sharing scheduling algorithms (FCFS, SPN and P) have been considered for simulation.

In [19], an adaptive dynamic load balancing model using agent-based distributed simulations has been proposed. Experimentation has been conducted to evaluate the efficiency of proposed algorithms under dynamic agent scheduling. Static random agent distribution has been employed in this simulation. The proposed algorithm has resulted in shorted execution time for synthetic workload traces. But this research work has not been evaluated using real workload traces under real time environment. Agents can play an important role for efficient and effective job scheduling on an experimental grid using real workload traces.

The evaluation of job scheduling algorithms should be based on two things, firstly, the use of appropriate performance metrics, and secondly, the use of an appropriate workload on which the scheduler should operate. A standard workload should be used as benchmark for scheduling algorithms [21, 22].

Many studies have been carried out to design better and more efficient job scheduling algorithms. Most of the scheduling algorithms; highlighted in the literature have not been evaluated using real workload traces.

The aim of this paper is to evaluate the performance, efficiency and scalability of our proposed grid scheduling algorithm and compare it with other well known scheduling algorithms on a computational grid

using real workload traces. Our scheduling performance metrics includes six key performance factors; i.e. average waiting time, average turnaround time, average response time, average bounded slowdown times, total completion times and maximum job stretch times.

3. Proposed Agent Based Prioritized Dynamic Round Robin Scheduling Algorithm

Grid scheduling is an NP complete problem, i.e., no such deterministic algorithm exists which can generate an optimum solution in polynomial time. To predict the demand of grid jobs in a dynamic scheduling environment however is not simple. The dynamic scheduling means jobs that are arriving in the system with different processing demands and have different priorities. The priorities are assigned to the jobs on the basis of user classifications.

Agent based Prioritized Dynamic Round Robin Scheduling (APDRR) uses task agent for job distribution in such a way to achieve the optimum solution. Task agent receives the jobs/processes from the users, and distributes them among different prioritize global queues based on user levels. Number of global queues can be customized in the grid system according to defined priority classifications at global level. We proposed a dynamic time quantum strategy in [23]. We have integrated the same dynamic time quantum strategy in designing and development of APDRR. Block diagram of APDRR is shown in Figure 1.

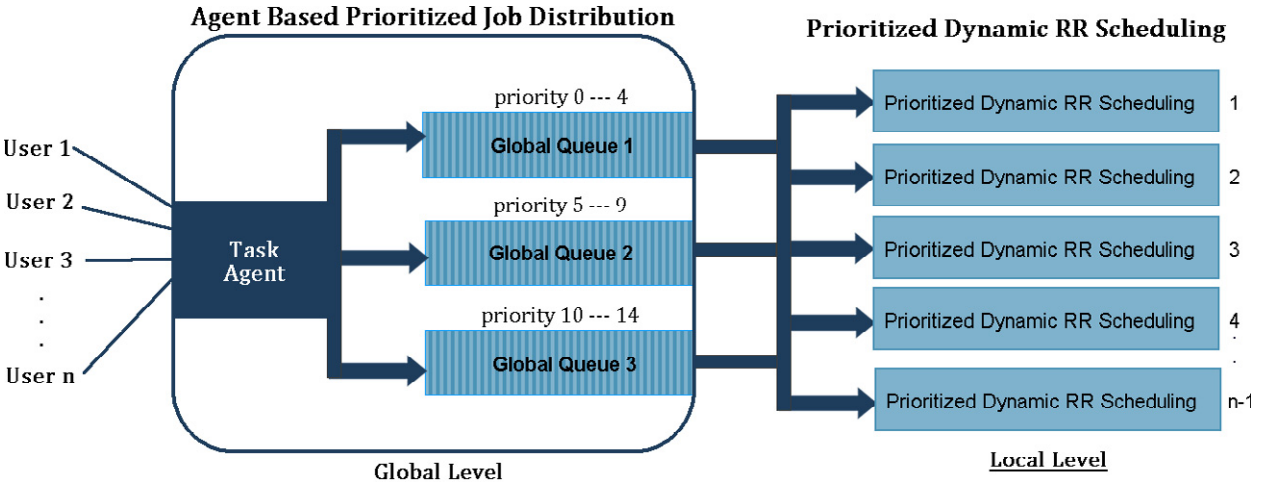

Fig. 1. Block Diagram of APDRR

APDRR uses agent based job distribution strategy at global level for optimal job distribution based on user levels and job priorities. APDRR uses its Agent based prioritized job scheduling strategy at local level for efficient and effective execution of jobs. P_i denotes a process i; where i ranges from '*1*' to '*n*'. Algorithms for each proposed strategy at global and local levels are as follows:

3.1. Agent based Prioritized Job Distribution Strategy

Step 1. Set the number of global queues for the computational grid

Step 2. Prioritization of global queues on the basis of job priorities and user levels

Priority (*global queue*$_1$)>> Priority (*global queue*$_2$)>> Priority (*global queue*$_3$)

Step 3. Define the value of priority threshold for each global queue based on priority classification; e.g. $Threshold_1$ for $global\ queue_1$, $Threshold_2$ for $global\ queue_2$ and $Threshold_3$ for $global\ queue_3$

Step 4. Task agent receives the user jobs/processes and distributes them among the global queues while considering their priorities

Step 5. If $P_i.priority <= Threshold_1$ then

Assign the *process* (P_i) to the $global\ queue_1$

else If $P_i.priority <= Threshold_2$ then

Assign the *process* (P_i) to the $global\ queue_2$

else

Assign the *process* (P_i) to the $global\ queue_3$

end if

Step 6. *if* $global\ queue_1$ is not empty *then*

Distribute the *processes* from $global\ queue_1$ among the compute nodes of a grid

else if $global\ queue_2$ is not empty then

Distribute the *processes* from $global\ queue_2$ among the compute nodes of a grid

else Distribute the *processes* from $global\ queue_3$ among the compute nodes of a grid

end if

Step 7. Master node receives the performance parameters (i.e., results) computed by the employed prioritized dynamic round robin scheduling algorithm at each slave processor (also called worker).

Step 8. After receiving, master node computes the summation of the values for the performance parameters of waiting times, turnaround time, response times and slowdown times, taken from each slave processor.

Step 9. Master node also computes the maximum values, out of all the maximal values for the total completion times as well as for the job stretch times, taken from each slave processor.

Step 10. Master node calculates the average values for performance factors of waiting times, turnaround time, response times and slowdown times.

3.2. Prioritized Dynamic Round Robin Job Scheduling Strategy

Step 1. Receive the processes from the global queue and put in the local queue

P_i. *Process_Status=New*

Step 2. Sort the *processes* in the local queue on the basis of *process priorities in* ascending order

Step 3. Compute the value of time quantum dynamically based on process runtime values, i.e.,

$Time\_Qunatum=sqrt(avg(P_1.runtime_1, P_2.runtime_2,.....P_n.runtime_n)$

Step 4. Allocate the CPU to a *process* that has the highest *priority*

Step 5. Execute the job on a compute resource (i.e., CPU) for a computed *Time_Quantum* value

Step 6. *If* a process completes its execution within the *Time_Quantum then*

P_i. *Process_Status=Completed*

else

i. P_i. *Process_Status=Partially_Completed*

ii. Place the process at the end of local queue

end if

Step 7. If all processes present in the ready queue have been executed for a computed time quantum value then

Goto Step 1

Step 8. Compute performance parameters - waiting times, turnaround times, response times, slowdown times, total

completion times and job stretch times

Step 9. Send computed performance parameters (i.e., output) back to master processor whose processor id is '0'.

Step 10. End

4. Statistical Analysis of Real Workload Traces

In [12, 24], a comprehensive statistical analysis has been carried out for a variety of workload traces on clusters and grids. We reproduced the graphs of [24, 25] to study the behaviour of the dynamic nature of workload 'AuverGrid' using our developed SyedWSim [26]. The total numbers of jobs in 'AuverGrid' are '404176'. We looked at the number of jobs arriving in each 1024 second period. The number of jobs arriving in a particular period is its 'job count'. The left hand graph of Figure 2 shows the distribution of job counts for the whole trace. Next we performed an autocorrelation of the job counts at different lags. The middle graph of Figure 2 shows the autocorrelation plot at different lags. Then we performed a Fourier analysis by applying the FFT on the values of the autocorrelation output. This is shown in the right hand graph of Figure 2.

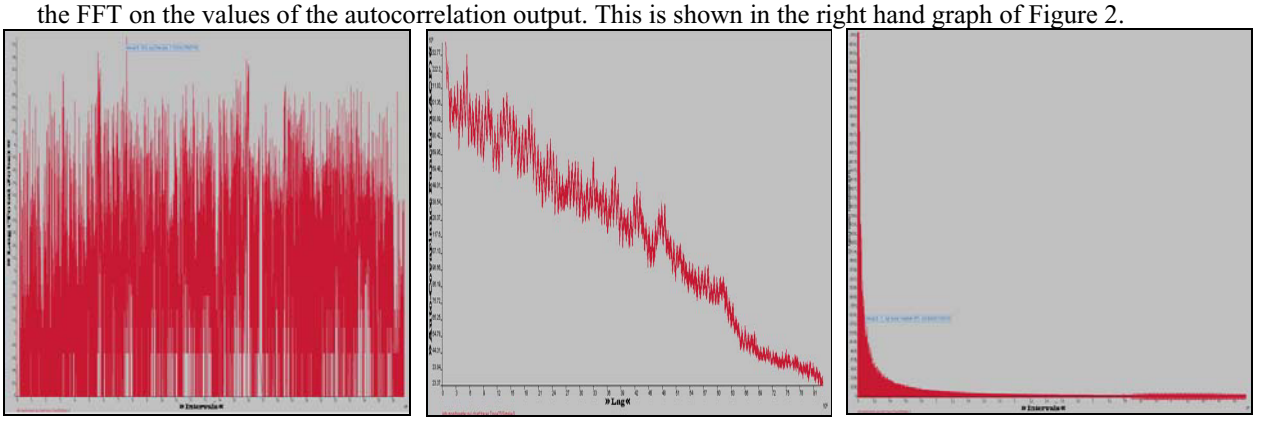

Fig. 2. The sequence plot, autocorrelation function(ACF) and Fast Fourier transformation(FFT) for the count process of AuverGrid

Figure 2 depicts that, job arrivals show a diversity of correlation structures, including short range dependencies, pseudo periodicity, and long range dependence. Long range dependencies can results in big performance degradation, which effects should be taken into consideration for evaluation of scheduling algorithms. Real grid workload 'AuverGrid' has shown rich correlation and scaling behaviour, which are

different from conventional parallel workload [24, 25]. 'AuverGrid' has been used in this work for performance evaluation of proposed and other existing scheduling algorithms on an experimental computational grid.

5. Homogeneous Implementation of Proposed Scheduling Algorithms

We used a master/slave architecture for implementation of job scheduling algorithms, as shown in Figure 3. 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.

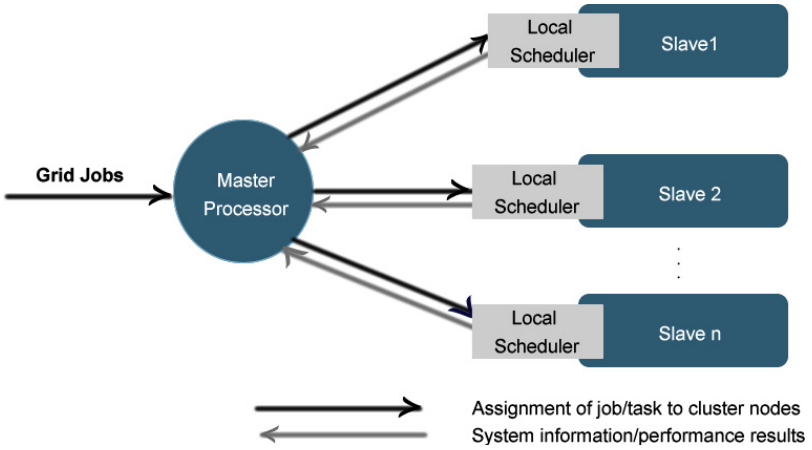

Fig. 3. Block diagram of master/slave architecture

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

6. Scheduling Simulator Design and Development

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. 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 one, APDRR, as well as established ones, PLRR,

FCFS, SPN, SPN, 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 1.

Table 1. Experimental Setup

Name	Type	Location	Configuration
gillani	Shell terminal	Lab Workstation	Intel Core 2 Duo CPU 2.0GHZ 2 GB Memory
hpc.local	Execution site	HPC facility	128 Core Intel(R) Itanium2(R) Processor 9030 arch : IA-64 CPU MHz : 1.6 GHz

8. Performance Analysis of Grid Scheduling Algorithms

Experiments have been performed on an experimental computational grid using 'AuverGrid'. Experimentation includes the efficiency, performance and scalability test of scheduling algorithms under an increased real workload and increased processors availability. Two data sets have been formed, first by using '3%', and second by using '5%' of the AuverGrid workload (i.e. 12125, and 20208 processes), respectively. The 'runtime' attribute is given for each process in 'AuverGrid'. The 'runtime' is taken as CPU time in this experiment. A series of experiments have carried on experimental grid by varying the number of CPUs successively from '16' to '128'. This experimentation had used '50' units as the fixed time quantum.

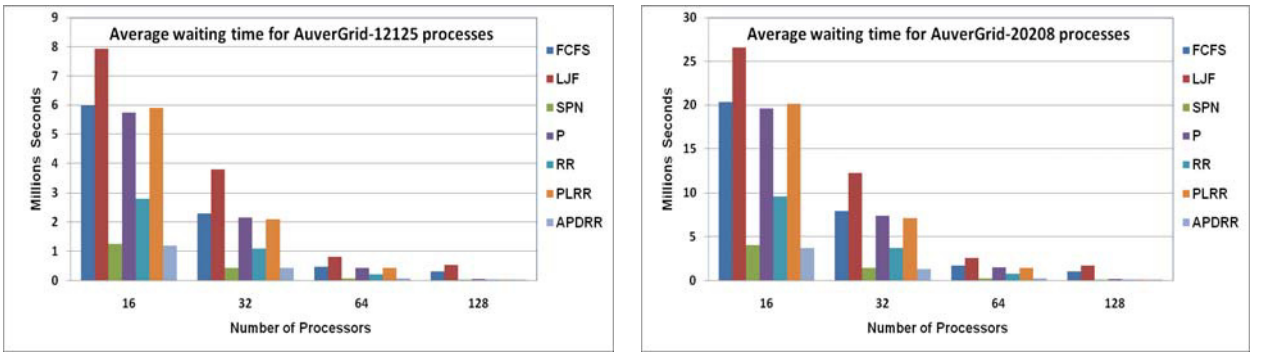

Fig. 4. Average waiting times of scheduling algorithms for 3% and 5% workload of AuverGrid

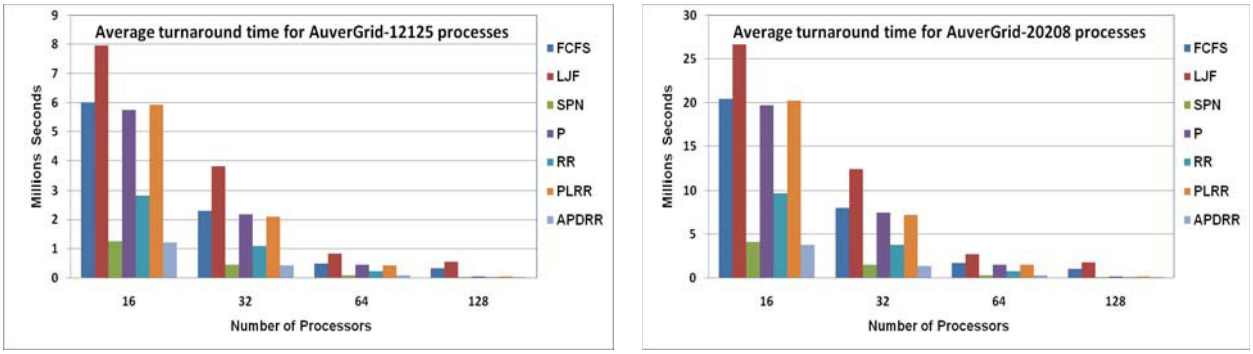

Fig. 5. Average turnaround times of scheduling algorithms for 3% and 5% workload of AuverGrid

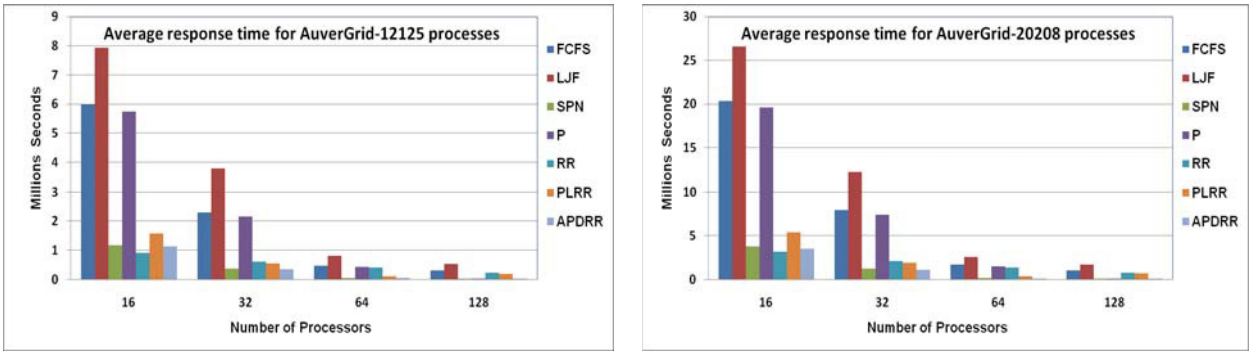

Fig. 6. Average response times of scheduling algorithms for 3% and 5% workload of AuverGrid

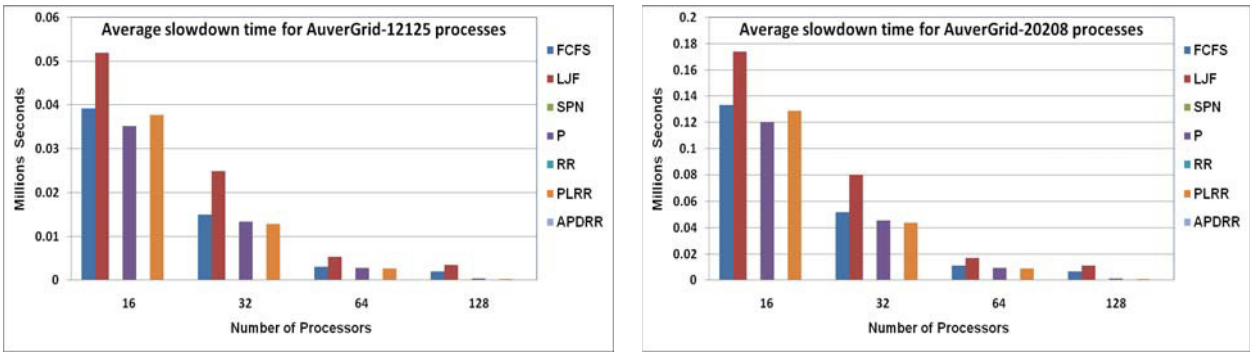

Fig. 7. Average bounded slowdown times of scheduling algorithms for 3% and 5% workload of AuverGrid

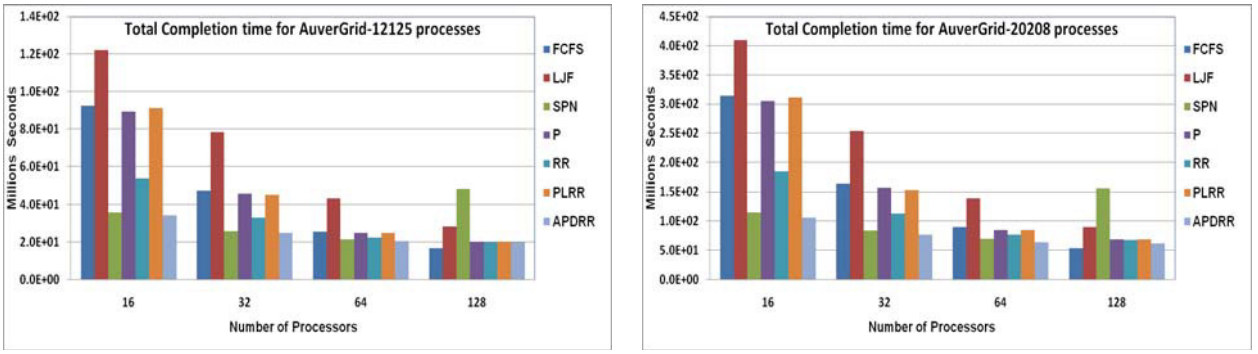

Fig. 8. Total completion times of scheduling algorithms for 3% and 5% workload of AuverGrid

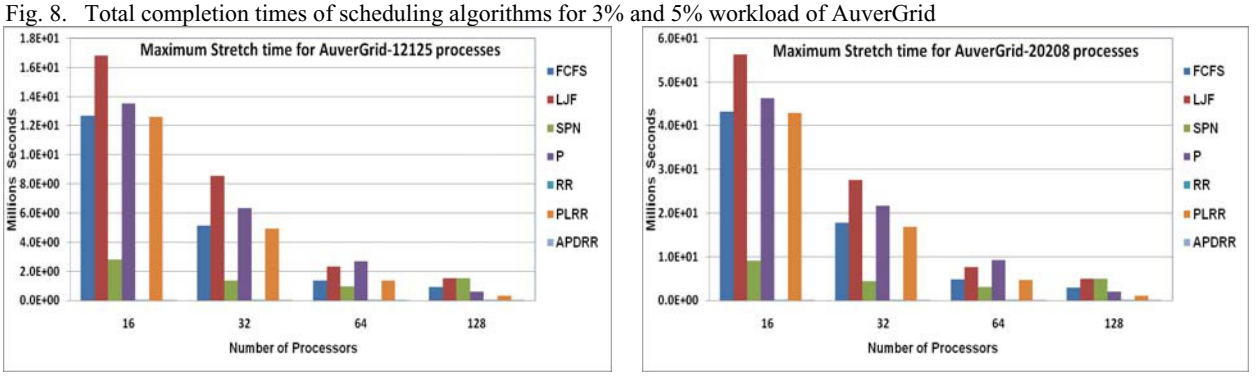

Fig. 9. Maximum job stretch times of scheduling algorithms for 3% and 5% workload of AuverGrid

8.1. Average Waiting Time Analysis

The *Waiting Time* is the time for which a process waits from its submission to completion in the local and global queues [11], [27]. Figure 4 illustrates the average waiting times for scheduling algorithm using AuverGrid workload trace of '12125' and '20208' processes. It has been found that APDRR and SPN scheduling algorithms have shown the best performance while producing the shortest average waiting times as compared to other scheduling algorithms. It also presents that RR has shown the average performance but result in higher average waiting time measures as compared to those for APDRR and SPN. Moreover, the RR, PLRR, P, FCFS and LJF have shown the worst performance w. r. t. the average waiting time measures. The LJF has shown to have the longest average waiting times. All scheduling algorithms have shown the improvement w. r. t. average waiting times by varying number of CPUs successively from '16' to '128'. As a result, APDRR has shown the optimal average waiting times for '3%' and '5%' workload of AuverGrid.

8.2. Average Turnaround Time Analysis

The *Turnaround time* of the job is defined as the time difference between the *completion time* and *release time* [11], [27]. Figure 5 shows the average turnaround times computed for each scheduling algorithm using '3%' and '5%' workload of AuverGrid. The values for average turnaround times computed by APDRR are found shorter than those for the other grid scheduling algorithms. This figure also shows that SPN has shown better performance w. r. t. the average turnaround times. Furthermore, it is found that RR, P, PLRR, FCFS and LJF scheduling algorithms have shown the longer average turnaround time measures.

8.3. Average Response Time Analysis

It is the amount of time taken from when a process is submitted until the first response is produced [11], [27]. In interactive grid applications, response time is a very important parameter. Average response times computed for the scheduling algorithms using '3%' and '5%' workload of AuverGrid, are shown in Figure 6. It is found that that average response times computed by the APDRR and RR are shorter than other scheduling algorithms. Average response times for each algorithm have decreased by increasing the number of CPUs. It also shows that SPN algorithm produces better average response time compared to other algorithms. However, P, PLRR, FCFS and LJF have shown the worst performance w. r. t. average response time measures, out of which LJF results in the longest average response times. All scheduling algorithms have shown the improvement w. r. t. average response time measures by increasing the number of CPUs successively from '16' to '128'.

8.4. Average Slowdown Time Analysis

Figure 7 shows the average slowdown times computed for each scheduling algorithm using '3%' and '5%' workload of AuverGrid. Figure 7 shows that APDRR and RR have produced the shortest average slowdown times compared to other scheduling algorithms. Figure 7 also presents that SPN has also shown average performance
w. r. t. the average slowdown times. It has also shown that PLRR, P, FCFS and LJF have shown the worst performance while resulting in longer average slowdown times. LJF has shown the longest average slowdown times. As a result, RR and APDRR have shown the best average slowdown times compared to other scheduling algorithms and presented improvement w. r. t. average slowdown times under the increasing number of CPUs successively from '16' to '128'.

8.5. Total Completion Time Analysis

Machine Completion time is defined as the time for which a machine *'m'* will finalize the processing of the previously assigned tasks as well as of those already planned tasks for the machine [6]. Figure 8 shows the total completion times computed for each scheduling algorithm using '3%' workload of AuverGrid. Figure 8 shows that APDRR has produced the shortest total completion times compared to the other scheduling algorithms. Figure 8 also presents that SPN and RR have shown slightly higher total completion times than those for APDRR. This figure also depicts that P, FCFS and LJF have shown the worst performance, resulting in longer completion times.

Moreover, all scheduling algorithms have shown improvement in total completion times by increasing the number of CPUs for '3%' to '5%' workload of AuverGrid. As a result, MH and MHR have shown best total completion times.

8.6. Maximum Job Stretch Time Analysis

Stretch time is defined as the flow of a job over the processing time. In order to avoid the starvation situation from the grid system, it is also required to minimize the stretch of each job [28], [29]. This motivates us to compute another performance parameter, i.e. Maximum Stretch time of job.

The maximum job stretch times for each scheduling algorithm using '3%' and '5%' workload of LCCG1 are shown in Figure 9. It can be depicted that APDRR have shown the shorter maximum job stretch times compared to the other scheduling algorithms. In addition, RR and SPN have shown the average measures of maximum job stretch times. Figure 9 also shows that P, PLRR, FCFS and LJF have produced the longest maximum job stretch times. Finally, APDRR, SPN and RR have shown the better maximum job stretch times.

9. Conclusion

In this paper we present agent based prioritized dynamic round robin job scheduling algorithm, namely APDRR. We compared the efficiency, performance and scalability of proposed job scheduling algorithm with other grid scheduling algorithms on a computational grid using real workload traces. In this paper we also performed a statistical analysis of the AuverGrid workload trace to study the dynamic nature of grid jobs.

Experimental results show that APDRR has shown the optimal performance from the system perspective while resulting in the least average waiting times, average turnaround times, average slowdown time, total completion times and maximum job stretch times. Moreover, APDRR has also shown the optimal performance from the user perspective while producing the least average response time measures under dynamic grid scheduling environment. It has been demonstrated and concluded that APDRR is an optimal job scheduling policy from system as well as from user perspective.

References

[1] E. Shmueli and D. G. Feitelson, "Backfilling with lookahead to optimize the packing of parallel jobs," J. Parallel Distrib. Comput., vol. 65, pp. 1090-1107, 2005.
[2] Y. Zhang, H. Franke, J. E. Moreira, and A. Sivasubramaniam, "Improving parallel job scheduling by combining gang scheduling and backfilling techniques," in Parallel and Distributed Processing Symposium, 2000. IPDPS 2000. Proceedings. 14th International, 2000, pp. 133-142.
[3] J. Chin, M. Harvey, S. Jha, and P. Coveney, "Scientific Grid Computing: The First Generation," Computing in Science and Engineering, vol. 7, pp. 24-32, 2005.
[4] K. Krauter, Buyya, R., Maheswaran, M., "A taxonomy and survey of grid resource management systems for distributed computing," Softw. Pract. Exper., vol. 32, pp. 135-164, 2002.
[5] L. Chunlin, Z. J. Xiu, and L. Layuan, "Resource scheduling with conflicting objectives in grid environments: Model and evaluation," Journal of Network and Computer Applications, vol. 32, pp. 760-769, 2009.
[6] F. Xhafa and A. Abraham, "Computational models and heuristic methods for Grid scheduling problems," Future Gener. Comput. Syst., vol. 26, pp. 608-621, 2010.
[7] J. Abawajy, "Job Scheduling Policy for High Throughput Grid Computing," in Distributed and Parallel Computing. vol. 3719, M. Hobbs, et al., Eds., ed: Springer Berlin / Heidelberg, 2005, pp. 184-192.
[8] B. G. Lawson, E. Smirni, and D. Puiu, "Self-Adapting Backfilling Scheduling for Parallel Systems," presented at the Proceedings of the 2002 International Conference on Parallel Processing, 2002.
[9] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "Backfilling Using System-Generated Predictions Rather than User Runtime Estimates," Parallel and Distributed Systems, IEEE Transactions on, vol. 18, pp. 789-803, 2007.
[10] K. Li, "Job scheduling and processor allocation for grid computing on metacomputers," Journal of Parallel and Distributed Computing, vol. 65, pp. 1406-1418, 2005.
[11] W. Stallings, Operating Systems Internals and Design Principles: Prentice Hall, 2004.
[12] H. Li and R. Buyya, "Model-Driven Simulation of Grid Scheduling Strategies," presented at the Proceedings of the Third IEEE International Conference on e-Science and Grid Computing, 2007.
[13] S. Jang and J. Lee, "Predictive Grid Process Scheduling Model in Computational Grid," in Advanced Web and Network Technologies, and Applications. vol. 3842, H. Shen, et al., Eds., ed: Springer Berlin / Heidelberg, 2006, pp. 525-533.
[14] D. Lifka, "The ANL/IBM SP scheduling system," in Job Scheduling Strategies for Parallel Processing. vol. 949, D. Feitelson and L. Rudolph, Eds., ed: Springer Berlin / Heidelberg, 1995, pp. 295-303.
[15] J. Skovira, W. Chan, H. Zhou, and D. Lifka, "The EASY — LoadLeveler API project," in Job Scheduling Strategies for Parallel Processing. vol. 1162, D. Feitelson and L. Rudolph, Eds., ed: Springer Berlin / Heidelberg, 1996, pp. 41-47.
[16] D. G. Feitelson, "A Survey of Scheduling in Multiprogrammed Parallel Systems," IBM T.J.Watson Research Center, Yorktown Heights, NY1995.

[17] R. Sharma, V. K. Soni, and M. K. Mishra, "An improved resource scheduling approach using Job Grouping strategy in grid computing," in Educational and Network Technology (ICENT), 2010 International Conference on, 2010, pp. 94-96.
[18] S. S. Rawat and L. Rajamani, "Experiments with CPU Scheduling Algorithm on a Computational Grid," in Advance Computing Conference, 2009. IACC 2009. IEEE International, 2009, pp. 71-75.
[19] Q. Long, J. Lin, and Z. Sun, "Agent scheduling model for adaptive dynamic load balancing in agent-based distributed simulations," Simulation Modelling Practice and Theory, vol. 19, pp. 1021-1034, 2011.
[20] L. Hui and R. Buyya, "Model-Driven Simulation of Grid Scheduling Strategies," in e-Science and Grid Computing, IEEE International Conference on, 2007, pp. 287-294.
[21] D. Feitelson and L. Rudolph, "Metrics and benchmarking for parallel job scheduling," in Job Scheduling Strategies for Parallel Processing. vol. 1459, D. Feitelson and L. Rudolph, Eds., ed: Springer Berlin / Heidelberg, 1998, pp. 1-24.
[22] M. Calzarossa and G. Serazzi, "Workload characterization: a survey," Proceedings of the IEEE, vol. 81, pp. 1136-1150, 1993.
[23] S. N. M. Shah, A. K. B. Mahmood, and A. Oxley, "Dynamic Multilevel Hybrid Scheduling Algorithms for Grid Computing," Procedia Computer Science, vol. 4, pp. 402-411, 2011.
[24] H. Li, "Workload dynamics on clusters and grids," The Journal of Supercomputing, vol. 47, pp. 1-20, 2009.
[25] Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, and D. Epema, "The Grid Workloads Archive," Future Generation Computer Systems, vol. 24, pp. 672-686, 2008.
[26] S. N. Mehmood Shah, A. K. Bin Mahmood, and A. Oxley, "SYEDWSIM: A Web Based Simulator for Grid Workload Analysis," in Software Engineering and Computer Systems. vol. 181, J. M. Zain, et al., Eds., ed: Springer Berlin Heidelberg, 2011, pp. 677-692.
[27] J. Blazewicz, Ecker, K.H., Pesch, E., Schmidt, G. und J. Weglarz, Scheduling Computer and Manufacturing Processes: Berlin (Springer), 2001.
[28] Rodero, F. Guim, and J. Corbalan, "Evaluation of Coordinated Grid Scheduling Strategies," in High Performance Computing and Communications, 2009. HPCC '09. 11th IEEE International Conference on, 2009, pp. 1-10.
[29] M. A. Bender, S. Chakrabarti, and S. Muthukrishnan, "Flow and stretch metrics for scheduling continuous job streams," presented at the Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms, San Francisco, California, United States, 1998.