Journal of Engineering and Computer Innovations Vol. 2(5), pp. 90-97 May 2011
Available online at http://www.academicjournals.org/JECI
ISSN 2141-6508 ©2011 Academic Journals

Full Length Research Paper

Dynamic task scheduling using service time error and virtual finish time

S. V. Kasmir Raja[1] and Daphne Lopez[2]*

[1]SRM University, Kattangulathur, Tamil Nadu, India.
[2]VIT University, Vellore, Tamil Nadu, India.

Accepted 29 March, 2011

The computational grid has emerged as an attractive platform to tackle various science and engineering problems. One of the challenging issues in the grid associated with the effective utilization of the heterogeneous resources is scheduling. This paper designs and implements a task-scheduling algorithm considering the dynamicity of the resources and the tasks. We explain the concept of queue's virtual time and combine it with virtual finish time and the service time error to allocate resources to the tasks for improved fairness and better throughput. The detailed performance evaluation of virtual finish time driven scheduling algorithm is carried out through a series of simulations by varying the number of tasks and processors of different capacities to optimize the cost and execution time of the tasks to achieve fairness.

Key words: Computational grid, heterogeneous resources, dynamicity, task-scheduling.

INTRODUCTION

Despite efforts that current grid schedulers with various scheduling algorithms have made to provide comprehensive and sophisticated functionalities, they have difficulty guaranteeing the quality of schedules they produce. The single most challenging issue that they encounter is the dynamicity of resources, that is, the availability and capability of a grid resource change dynamically (Foster and Kesselman, 1999a, b). Although a resource may be participating in a grid, its main purpose is for use by local users of the organization that it belongs to. Therefore, the load on the resource imposes a great strain on grid scheduling. Though there are a number of scheduling algorithms existing, identifying the best algorithm in a grid environment is complex and critical. All the tasks that are submitted to the grid will have to be executed in the stipulated time and its complexity increases due to dynamic change of the resources. An important issue in practical scheduling is fairness in user service. The scheduling policies could be preemptive or non-preemptive as shown in Figure 1.

Non-preemptive scheduling is performed only when processing the previous task is completed and is attractive due to the simplicity of its implementation for it is not necessary to maintain a distinction between an unserviced task and a partially serviced one. Preemptive scheduling (Jackson and Rouskas, 2002) involves the interruption of the task after it has executed for its time quantum and added to queue of pending requests. Irrespective of the type of policy, the objective function is to reduce the execution time and cost associated with the execution of the job, which increases the throughput of the system.

Proportional share resource management provides a flexible and useful abstraction for multiplexing scarce resources among users and applications.

Virtual time is a paradigm for organizing and synchronizing distributed systems which can be applied to such problems as distributed discrete event simulation and distributed database concurrency control. Virtual time provides a flexible abstraction of real time in much the same way that virtual memory provides an abstraction of real memory. This paper introduces virtual time (Mattern, 1989), a concept that allows a distributed system to be organized around a linear global clock; rather than maintain a synchronized clock, it achieves efficiency by having each node maintain its own local virtual time and performing rollback when a node receives a message in the past. Although not widely adopted, it has served as an influential model of a general system with optimistic results.

*Corresponding author. E-mail: daphnelopez@vit.ac.in.

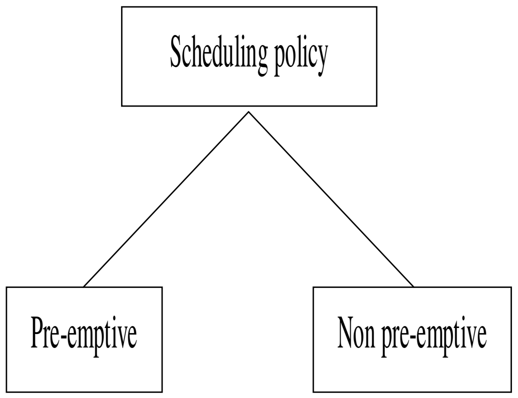

Figure 1. Scheduling policy.

RELATED WORK

Proportional fair is a compromise-based (Kushner and Whiting, 2004) scheduling algorithm. It is based upon maintaining a balance between two competing interests: This concept is basically applied to networks in the context to maximize total wireless network throughput, while at the same time allowing all users at least a minimal level of service. Fair queuing (Doulamis et al., 2007; Hosaagrahara and Sethu, 2008) can be interpreted as a packet approximation of generalized processor sharing (GPS). This is done by assigning each data flow a data rate or a scheduling priority (depending on the implementation) that is inversely proportional to its anticipated resource consumption. Demers et al. (1989) propose fair queuing for network packet scheduling as Weighted Fair Queueing (WFQ), with a more extensive analysis provided by Parekh and Gallager (1993), and later applied by Waldspurger (1995) to CPU scheduling as stride scheduling. Fair queuing in network emulates fairness of bitwise round-robin (Shreedhar and Varghese, 1996) sharing of resources among competing flows. Elshaikh et al. (2006) propose a fair new weighted fair queuing algorithm used in networks where two types of queues and the queue length is used as a parameter incalculating the virtual time, to ensure that the flows or aggregates are not punished for using uncounted bandwidth.

Doulamis et al. (2007) uses a max-min fair sharing approach for providing fair access to users. When there is no shortage of resources, the algorithm assigns to each task enough computational power for it to finish within its deadline. When there is congestion, the main idea is to fairly reduce the CPU rates assigned to the tasks so that the share of resources that each user gets is proportional to the user's weight. All tasks whose requirements are lower than their fair share CPU rate are served at their demanded CPU rates which they define as fairness. Distributed scheduling algorithms with multiprocessor systems (Ramamritham et al., 1990) and metascheduler (Shreedhar and Varghese, 1996) is also proposed.

In order to reduce the computational load, the concept of virtual time is introduced. Weighted fair queuing introduces the idea of a virtual finishing time (VFT) to do proportional sharing scheduling. The virtual finish time is dependent on the virtual time. Virtual time of a task is defined as the degree to which a task has received its proportional allocation relative to other tasks. Given a task's virtual time, the virtual finish time (VFT) is the virtual time the task has after executing for one time quantum. Virtual time round robin is another scheduling algorithm that uses the virtual finish time parameter in Linux to achieve fairness.

Sanjay and Vadhiyar (2008) calculated the time taken to execute parallel application by considering the problem size, the varying number of processors and the transient CPU and network characteristics respectively. The execution time is split into two, one for representing the computation and the other for communication costs.

DYNAMIC VIRTUAL TIME FAIR QUEUE ALGORITHM

Grid model

The grid model consists of a number of computational nodes and each node consists of a number of processors of varying capacity. Once the user is issued, an acceptance of the task it belongs to the grid administrator.

The resources are discovered that are capable of executing the tasks that are submitted to the site. The resources that do not meet the tasks requirements are moved to the other site.

The tasks are placed in the task queue and move to the active state. Tasks in the active state are scheduled and ready for execution. When the task is allocated, the resources transits to the execution state and on expiry of the time quantum the task is either moved to the end of the queue or removed from the queue if it is not ready for the second time quantum. On successful completion the tasks enters a finished state and is moved to the user's site.

Problem formulation

The problem of task scheduling in a grid G basically consists of a dynamic set of T independent tasks to be scheduled on a dynamic set of N computational nodes (resources).

An instance of the problem consists of:

1. A set of T independent tasks to be scheduled. Each task has associated with it a workload (in million of instructions). Every task must be entirely executed in a unique machine.
2. A set of M number of processors which has its corresponding computing capacity (in MIPS).

ALGORITHM

The user submits the task to the grid environment. The grid scheduler allocates the submitted tasks to the computational nodes. A queue is maintained for each computational node and the queue size depends on the number of tasks submitted initially in the grid environment. The size may not be equal at all intervals of time because of the dynamic nature of the grid.

In this algorithm, a task has six values associated with its execution state: share, service time error, virtual finishing time, time counter, task identity and the run state. A task's share value identifies its resource rights. Share is allocated to the task based on the price the user pays. *Perfect fairness* is an ideal state in which each task has received service exactly proportional to its share at all intervals of time. Denoting the share of a task as *S* and the service received by task during a time interval t_1, t_2 as *W,* the perfect fairness of a task is represented as:

$$W(t_1,t_2)=(t_2-t_1)S/\sum_{i=1}^{n}S \tag{1}$$

The service time error is calculated as the difference between the amount of service time allocated to the task during interval (t_1, t_2) under the given algorithm, and the amount of time that would have been allocated under an ideal scheme that maintains perfect fairness for all clients over all intervals:

$$E=st(t_1,t_2)-w(t_1,t_2) \tag{2}$$

The virtual time of a task is a measure of the degree to which a task has received its proportional allocation relative to other tasks. Virtual time is represented as:

$$VT=W(t)/S \tag{3}$$

Given a task's virtual time, the task's virtual finishing time (VFT) is defined, as the virtual time the task would have after executing for one time quantum. A task's VFT advances at a rate proportional to its resource consumption divided by its share. The VFT is used to decide the position of the task in the queue and that the task in the beginning of the queue would be allocating the resources. A task's time counter measures the number of allocations for that particular task in order to measure the fairness at the end of each scheduling cycle. Information about the scheduler as the time quantum, queue, total shares and the queue virtual time is also maintained. The total shares are the sum of the shares of all the tasks that are ready to run.

Dynamic considerations

Initially when the execution starts, the tasks are sorted to their share values and tasks would not have consumed any time quantum so the task's implicit virtual time is set to be the same as the queue virtual time (QVT). Virtual finish time of a new task is calculated as

$$VFT(t)=QVT(t)+Q/S \tag{4}$$

If the executable task is not in an active state, it is simply removed from the queue and the service time error for the current and the next task is calculated and is allotted the resource. The task that is in an inactive state lies somewhere in the queue then the task is removed and the pointer values are appropriately modified in the linked lists. This way, the tasks can be preempted and used. The pseudocode is given in Figure 2. The time complexity is O(n^2).

Arrival model

The tasks arrival is modeled as an application of a queuing system. They are allowed up to *Q* seconds of time and are fed back to the queue if they have not completed their processing. We assume that the task resumes its operation when it gets the processor for the next time quantum. We also assume that the arrival times are independent of each other. λ is the rate at which the jobs arrive at the system, μ is the rate at which the jobs are serviced. Assuming $\rho = \lambda / \mu$.

Modeling each queue as an M/D/1 system, it can be shown that the average length of the queue:

$$Q=\frac{\rho^2}{2(1-\rho)} \tag{5}$$

```
Input: A set of taks T, a set of N computational nodes with multiple processors
Output: A schedule of T onto N
1. Create a set Q of N queues qsize = T/N
2. Each user is associated with the broker entity and the resource characteristics are identified
3. Assign shares to the tasks, a positive value (depends on how much the user pays for his service)
4. Remove qsize tasks in T and enqueue them to $q_i$
5. SCHEDULING: (a) Assign shares to the tasks in each queue. SORT is performed (Arrange the tasks in decreasing order of their shares) Repeat c & d for one scheduling cycle
(b) The first task in queue is executed initially for the required time quantum
(c) Compute the service time error for the task in execution and the task in the head of the queue
(d) Schedule the job which has the least value
6. VIRTUAL TIME: Compute the virtual time and the virtual finish time and store the values in the counters.
7. Schedule the task with a negative value
8. Change position:  After every scheduling cycle the order of the tasks are based on their virtual finish time. The task with the smallest Virtual Finish Time is first chosen.
9. Repeat steps 5c through 5d for the current scheduling cycle
10. RESULT: Return the result set to the user
```

Figure 2. Pseudocode for dynamic virtual time fair queuing.

While the average waiting time is given by:

$$W = \frac{\rho}{2\mu(1-\rho)} \tag{6}$$

And the total time for task completion is:

$$t = \frac{2-\rho}{2\mu(1-\rho)} \tag{7}$$

Objective function

The objective function is to minimize the service time error. Minimization of error uses root mean square method as given in Equation (8):

$$Z = \sqrt{\left(\frac{\sum_{n-1 to m} \left(W_n - \{t_{2n} - t_{1n}\} * S_n / \sum_{n-1}^{m} S_n \right)^2}{m} \right)} \tag{9}$$

SIMULATION EXPERIMENT SETUP

GridSim requires the creation of resources with different capacities. We model the application as Gridlets and define all the parameters associated with the task. Then a GridSim user entity is created, that interacts with the resource broker scheduling entity to coordinate the execution of the tasks. Finally we implement a resource broker entity that performs scheduling on grid resources. The resources with their attributes used in scheduling are listed in Tables 1 and 2.

Table 1. The grid resources attributes.

Parameter	Value	Notation
Total number of resources R_0 to R_{20}	1-20	Machines
Speed	200-400	Million instructions per second
Number of processors	4-6	Processing elements
Resource manager type	Time-shared	

Table 2. Workload attributes.

Parameter	Value	Notation
Total number of tasks	100-500	
Length of a task	1,000-5,000	Million instructions (MI)
Number of processors required	4-6	Million instructions per second (MIPS)

Table 3. Experimental results for number of tasks 100 and 4 processing elements in each machine.

Parameter	Cost	Execution time	Min. error	Max. error
FCFS	461452.0	19342	-26.20000	25.54433
Round robin	422050 .9	17509	-25.80999	23.83333
Dynamic virtual time fair queue	300139.0	10766	-16.15	17.13205

Table 4. Experimental results for number of tasks 300 and 6 processing elements in each machine.

Parameter	Cost	Execution time	Min. error	Max. error
FCFS	521252.2	21251.93	-37.46000	37.21818
Round robin	495897.3	19690.05	-36.45126	36.63650
Dynamic virtual time fair queue	401112.19	14389.11	-29.7989	30.97099

Table 5. Experimental results for number of tasks 500 and 6 processing elements in each machine.

Parameter	Cost	Execution time	Min. error	Max. error
FCFS	656040.5	28765.54	-42.7814	42.72
Round Robin	615497.6	25721.15	-39.100	39.4535
Dynamic virtual time fair queue	551960.17	18435.12	-29.7576	28.4557

EXPERIMENTAL RESULTS

We perform simulations to implement fairness by using service time error as one parameter and using the virtual time fair queue performing simulations with 100 to 500 numbers of tasks. The experiment is carried out for FCFS (First Come First Serve), round robin and dynamic virtual time fair queuing. It minimizes error as well as using the virtual finish time it maximizes fairness. The experimental results in terms of cost, execution time, minimum and maximum error are shown in Tables 3 to 5, for number of tasks 100, 300 and 500, respectively. A comparison of execution time with tasks, maximum error with tasks and cost with tasks is given in Tables 6 to 8 respectively. Corresponding graphical representations are also shown in Figure 3 to 5. Maximum error for the number of tasks ranging from 100 to 500 is tabulated in Table 9. It guarantees that all the tasks are considered for execution.

The experimental results show that the maximum error reaches a maximum value and starts declining even

Table 6. Comparison of execution time with number of tasks.

No. of task	Execution time		
	FCFS	RR	Dynamic virtual time fair queue
100	19342	17509	10766
300	21251.93	19690.05	14389.11
500	28765.54	25721.15	18435.12

Table 7. Comparison of maximum error with number of tasks.

No. of task	Maximum error		
	FCFS	RR	Dynamic virtual time fair queue
100	25.54433	23.83333	17.13205
200	37.21818	36.63650	30.97099
300	42.72	39.4535	28.4557
500	48.96457	45.68740	25.16666

Table 8. Comparison of cost with number of tasks.

No. of task	Cost		
	FCFS	RR	Dynamic virtual time fair queue
100	461452.0	422050 .9	300139.0
200	521252.2	495897.3	401112.19
300	656040.5	615497.6	551960.17
500	721460.3	701412.3	591209.8

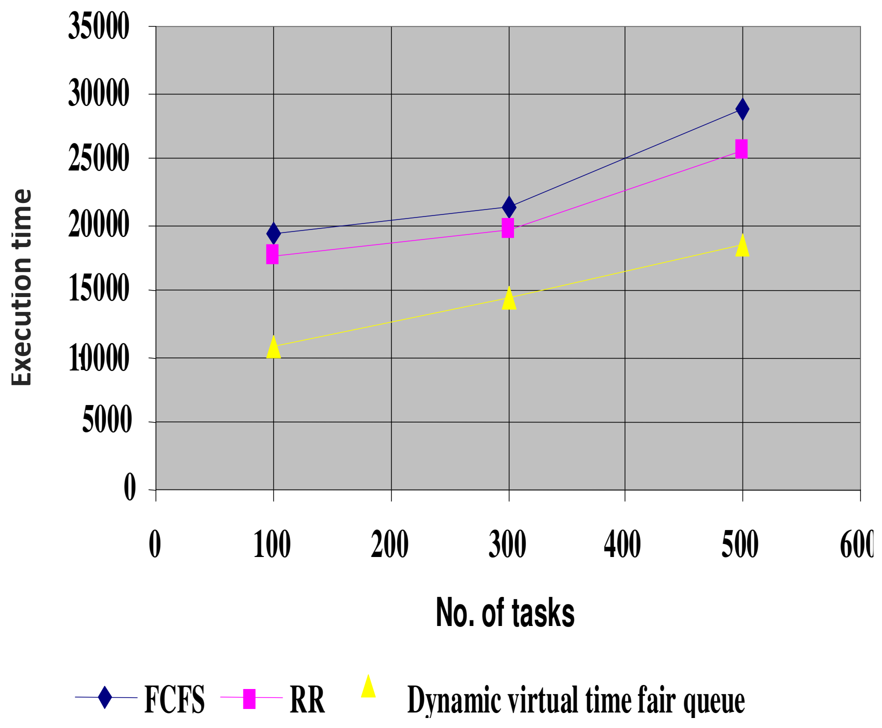

Figure 3. Execution time versus number of tasks.

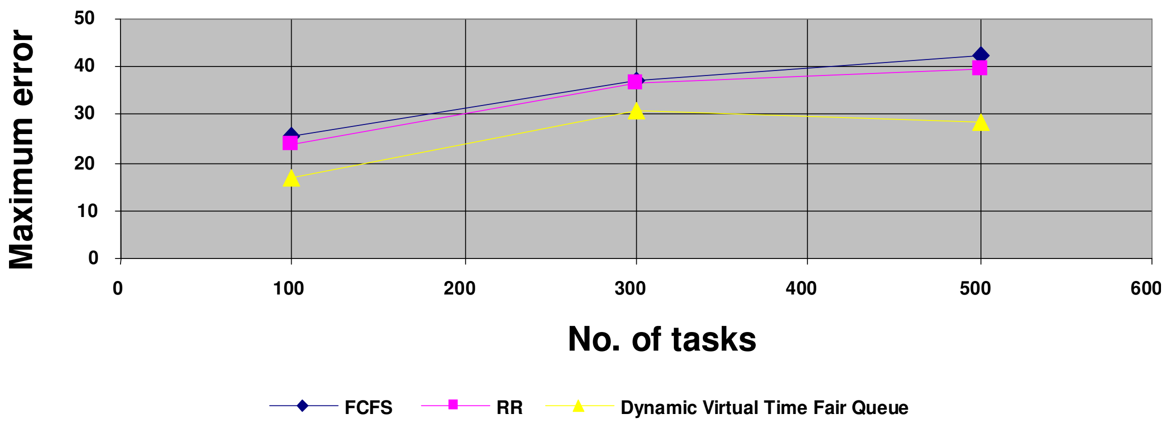

Figure 4. Maximum error versus number of tasks.

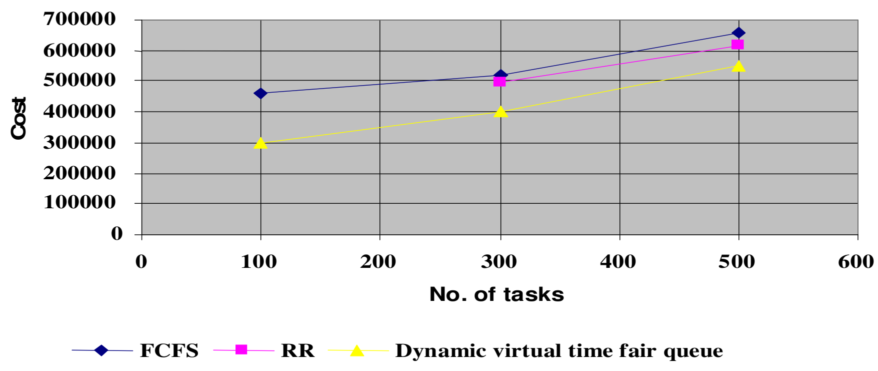

Figure 5. Cost versus number of tasks.

Table 9. Maximum error values for number of tasks 100 to 500.

No. of tasks	Max error
100	17.13205
200	18.002
300	20.013
400	21.1201
500	19.0

when the number of tasks increases. This is due to the consideration of the virtual finish time in the allocation of resources. The algorithm has proved to give the best results of all the algorithms even after considering the dynamic submission of the jobs. When new job arrives, the queue's virtual time is the virtual time of the job. In this way, after every scheduling cycle even, a new job that has arrived recently gets a proportional allocation of the resource. Even with dynamic considerations, virtual time fair queue algorithm shows a 50% higher performance than FCFS and round robin.

Conclusion

We discussed the use of service time error and virtual finish time for devising scheduling strategies for high end tasks on distributed resources. We simulated and evaluated the performance of scheduling algorithms in terms of error, cost and time for a variety of scenarios. This algorithm can be used to study the performance of various real time applications and can be embedded in a metascheduler also for enabling global scheduling. It is proposed to scale it up to the cloud infrastructure.

REFERENCES

Foster I, Kesselman C (1999a). “The Grid: Blueprint for a Future Computing Infrastructure”. San Francisco, CA: Morgan Kaufmann Publishers, pp. 21-26

Foster I, Kesselman C (1999b). “The Grid: Blueprint for a new computing infrastructure”. Chapter “The Globus toolkit”, 1st edition, San Francisco, CA: Morgan Kaufmann Publishers Inc., pp. 259-278.

Demers A, Keshav S, Shenker S, (1989). “Analysis and simulation of a Fair Queuing Algorithm”. In Proc. ACM SIGCOMM ’89, New York, NY: ACM, pp. 1-12.

Doulamis ND, Varvarigos E, Varvarigou T (2007). “Fair Scheduling Algorithms in Grids”. IEEE Trans. Parallel Distrib. Syst., 18(11): 1630-1648.

Hosaagrahara M, Sethu H (2008). “Max-Min Fair Scheduling in Input-Queued Switches”. IEEE Trans. Parallel Distrib. Syst., 19(4): 462-475.

Elshaikh MA, Othman M, Shamala S, Desa J (2006). “A New Fair Weighted Fair Queuing Scheduling Algorithm in Differentiated Services Network”. Int. J. Comput. Sci. Netw. Security, 6(11): 267-271.

Mattern F (1989). “Virtual Time and Global States of Distributed Systems”. J. Parallel Distrib. Algorithms. pp. 215-226.

Shreedhar M, Varghese G (1996). “Efficient Fair queuing using Deficit Round Robin”, IEEE Trans. Netw., 4(3): 375-385.

Waldspurger CA (1995). “Lottery and Stride Scheduling: Flexible Proportional-Share Resource Management”. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, pp. 12-34

Sanjay HA, Vadhiyar S (2008). “Performance modeling of parallel applications for grid scheduling,” J. Parallel Distrib. Comput., 68: 1135-1145.

Ramamritham K, Stankovic JA, Shiah PF (1990). “Efficient Scheduling Algorithms for Real-Time Multiprocessor Systems”. IEEE Trans. Parallel Distrib. Syst., 1(2): 184-194.

Kushner HJ, Whiting PA (2004). “Convergence of proportional-fair sharing algorithms under general conditions”. IEEE Trans. Wireless Commun., 3(4): 1250-1259.

Parekh AK, Gallager RG (1993). “A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case”. IEEE/ACM Trans. Netw., 1(3): 344-357.