

TWO WAY PROPORTIONAL JOB SCHEDULING IN COMPUTATIONAL GRIDS USING LEARNING AUTOMATA

S.SUGANTHI¹, S.NANDHINI DEVI²

Department of Computer Science, MAM SCHOOL OF ENGINEERING, Trichy
suganthi460@gmail.com, devinandhini1982@gmail.com,

ABSTRACT

Job scheduling is one of the most challenging issues in Grid resource management that strongly affects the performance of the whole Grid environment. The major drawback of the existing Grid scheduling algorithms is that they are unable to adapt with the dynamicity of the resources and the network conditions. Therefore, these methods do not scale well as Grid size grows and do not perform well as the environmental conditions change with time. In this project, a two-way proportional learning automaton based job scheduling algorithm for Grids is proposed. In this method, the workload that is placed on each Grid node is proportional to its computational capacity and varies with time according to the Grid constraints. Also, users are given priority based on their needs and the number of jobs they submit. The performance of the proposed algorithm is evaluated through conducting several simulation experiments under different Grid scenarios.

Keywords: Ant Colony Optimization, Genetic programming-based grid Job Scheduling, Memetic Algorithm, Variable action-set Learning Automata

1. INTRODUCTION

Grids are large scale collections of heterogeneous and autonomous systems from multiple administrative domains geographically distributed and interconnected by the wide area network. Grid implies to an extensive concept that is often referred to as the parallel system of the 1970s, the large scale cluster system of the 1980s, and the distributed system of the 1990s. Grid technology is an emerging paradigm for large scale distributed computing. Scientific problems are becoming more and more complicated and hard to solve. These complex problems need a huge amount of computing resources that cannot be sufficiently provided in distributed or parallel systems. The

computational Grid is a promising approach that exploits the synergy between a set of interconnected Grid nodes to reach a common goal to solve the massive computational problems. Grid resources can be freely added or withdrawn at any time according to the owners' discretion. Performance of the Grid nodes and their load frequently change with time. Grids allow the selection, aggregation, and sharing of the software and hardware resources of different computers in a distributed fashion. These systems provide pervasive, inexpensive, and reliable access to the high end computational capabilities. Although the Grid technology is still in the early stage of the research and development, due to the low cost of computing resources and recent advances in computing and wide-area networking, it has extensively grown and moved from an obscure research subject to a practical, highly popular technology during the last decade.

1.1.1 Computational Grids

A computational grid is a loose network of computers linked to perform grid computing. In a computational grid, a large computational task is divided up among individual machines, which run calculations in parallel and then return results to the original computer. These individual machines are nodes in a network, which may span multiple administrative domains and may be geographically distant. Each of the nodes may be thought of as a discrete system that can perform work and has access to a network. Computational grids are often more cost-effective than supercomputers of equal computing power.

1.1.2 Job Scheduling in Grids

Generally, the Grid job scheduling is defined as the process of decomposition of a large problem into a number of subtasks, and allocation of the subtasks to

available computing resources. The efficiency of a Grid environment is strongly dependent on the job scheduling technique it follows. Different forms of the job scheduling problem are computationally hard to solve. It has been shown that the finding of an optimal solution to the job scheduling problem in heterogeneous Grid systems is known to be NP-hard in general. Due to the hardness of the job scheduling problem and the dynamicity and extensiveness of the Grid environments, there is an urgent need for an adaptive, efficient and cost effective algorithm to schedule the Grid jobs. A host of scheduling techniques have been already presented and implemented in different types of Grid.

The existing job scheduling algorithms cannot efficiently adapt to the dynamicity of the resources and environment conditions. Such scheduling methods make a single schedule for the entire workflow in advance, and then the tasks are conducted according to this schedule. This significantly degrades the Grid performance if the resource availability or the environmental conditions changes over time.

1.2.2 Families

LA can be classified into two main families: fixed structure learning automata and variable structure learning automata. Variable structure learning automata are represented by a triple $\langle \beta, \alpha, L \rangle$ where β is the set of inputs, α is the set of actions and L is learning algorithm. The learning algorithm is a recurrence relation which is used to modify the action probability vector. Let $ai(k) \in \alpha$ and $p(k)$ denote the action selected by learning automaton and the probability vector defined over the action set at instant k , respectively. Let a and b denote the reward and penalty parameters and determine the amount of increases and decreases of the action probabilities, respectively. Let r be the number of actions that can be taken by learning automaton. At each instant k , the action probability vector $p(k)$ is updated by (1.1), if the selected action $ai(k)$ is rewarded by the random environment, and it is updated by (1.2), if the taken action is penalized.

If $a = b$, the algorithms are called linear reward-penalty (L_{R-P}) algorithm. If $a \gg b$, the algorithm is called linear reward- ϵ penalty ($L_{R-\epsilon P}$) algorithm, and finally if $b = 0$ it is called linear reward-Inaction (L_{R-I}). In L_{R-I} , the action probability vectors remain unchanged when the taken action is penalized by the environment.

2. RELATED WORK

2.1 Quad based K- means Algorithm

Quad Tree-based K-Means algorithm has been applied for predicting faults in program modules. First, Quad Trees are applied for finding the initial cluster centers to be input to the K-Means Algorithm. An input threshold parameter governs the number of initial cluster centers and by varying the user can generate desired initial cluster centers. The concept of clustering gain has been used to determine the quality of clusters for evaluation of the Quad Tree-based initialization algorithm as compared to other initialization techniques. The clusters obtained by Quad Tree-based algorithm were found to have maximum gain values. Second, the Quad Tree based algorithm is applied for predicting faults in program modules.

3. LEARNING AUTOMATA-BASED JOB SCHEDULING (LAJS) ALGORITHM

A Learning Automata (LA)-based algorithm was proposed by Dr.Javad Akbari Torkestani to solve the job scheduling problem in Grid environments. In this method, two LAs are associated with each scheduler.

One is for scheduling the user submissions and another one for allocating the workload to the Grid computational resources. In the algorithm called LAJS, the workload that is assigned to each Grid node is balanced, and each user client is permitted to submit its works whenever it needs. The method shortens the execution time and properly distributes the workload among the Grid resources. As the proposed algorithm proceeds, each user is assigned the portion of the Grid capacity as much as it needs.

Every user submits its jobs to a scheduler. One or more users may submit their works to one or more schedulers. Each scheduler can be in connection with one or more Grid nodes. Therefore, there is a many-to-many connection between the schedulers and Grid nodes.

3.1.1 Scheduling the user submissions

The LAJS algorithm uses learning automaton AsU to optimally schedule the user clients connected to scheduler S_s to submit their works. S_s schedules the user clients using a polling technique. It polls the users one-by-one to give them an opportunity to submit their works. At each stage, automaton AsU chooses one of its possible actions (say, action $\alpha^i_s U$) according to its action probability vector at random. By the selection of action $\alpha^i_s U$, user Ui is implicitly granted the permission to send its work. Scheduler S_s checks the selected user

clients to see if it has a ready job to submit. If so, scheduler S_s let user U_i send its work, queues the received works in the order of their arrival times, and increases the choice probability of (or rewards) the selected action $a^i_s U$ by (1.1). Otherwise (i.e., user U_i has no job to submit), scheduler S_s decreases the choice probability of (or penalizes) the selected action by (1.2). After updating the internal state of the learning automaton AsU , scheduler S_s initiates another stage and repeats the same operations as it did in the previous stage. As the scheduling algorithm proceeds, scheduler polls each user client (more probably) only when it has a job to submit.

3.1.2 Scheduling the Grid resources

The aim of the learning automaton AsG is to find a scheduling strategy to assign a computational resource to each task in such a way that the average execution time of the submitted jobs is minimized. Selection of an action from the action set means that a processing element is chosen to assign to a task. The automaton AsG schedules each task as follows:

A processor can be assigned to a task if the requirements of the task is less than or equal to the processor's computational capacity. If this condition is not met, the actions corresponding to these nodes are removed from the action-set of the automaton. The scheduler S_s updates the action-set. The remaining actions are candidates that can be assigned to the task. Automaton AsG then randomly chooses one of its possible actions based on its updated action probability vector. Let us assume that action corresponding to processing node p_1 is selected by the learning automaton. Scheduler S_s checks the queue of p_1 to see if its length is shorter than or equal to dynamic average length L_s . If so, scheduler increases the choice probability of the selected action by (1.1). Otherwise, the scheduler decreases it by (1.2). Regardless of rewarding or penalizing the selected action, scheduler assigns the task to the processing node p_1 . At the end of each assignment, all removed actions must be enabled again. For each processing node, the queue length is updated as soon as the action is selected by the learning automaton.

3.1.3 Advantages

Makespan (defined as the maximum execution time of all submitted jobs or completion time of the latest task), flowtime (defined as the sum of the completion time of all jobs) and load balancing using LAJS are improved when compared with those of Fuzzy logic-based Particle Swarm Optimization (FPSO) and Genetic programming-based grid Job Scheduling

(GJS). Users are entertained as per their needs. Queue length is kept below the dynamic average length.

4. TWO WAY PROPORTIONAL JOB SCHEDULING USING LEARNING AUTOMATA

In order to eradicate the disadvantages of LAJS algorithm, a modified learning automata is proposed. The proposed algorithm will give priority to the users who submit more number of jobs. It also will allocate the tasks to processors proportionate to their computational capacities while achieving load balancing so that the processors will be utilized fully.

4.1.1 Automaton for scheduling user submissions

This automaton aims at scheduling the user clients to submit their jobs. Each scheduler may receive the jobs from one or more users. Let $\{U_1, U_2, \dots, U_{Nu}\}$ be the set of users that must be scheduled by S_s . The action-set of automaton AsU has Nu actions, each for a user client. Selection of action $a_i s U$ means that scheduler S_s permits user U_i to submit its work. Let psU be the action probability vector of the learning automaton AsU . Clearly, action a_i is selected with probability p_i . All actions are initially chosen with the same probability $1/Nu$. This implies that all user clients initially have the same chance to submit their jobs to the system. The chosen action (user) will be rewarded.

Here also, S_s schedules the user clients using polling. Users can submit any number of jobs. Users submitting more number of jobs are rewarded more by modifying relation (1.1) to

$$p_j(k) = \frac{p_j(k) + n * a [1 - p_j(k)]}{k + 1}; \text{ for } j = i \quad (4.1)$$

$$(1 - a) p_j(k); \text{ otherwise}$$

Where n is the number of jobs submitted by the selected user. The other users are penalized the same way as in relation (1.1)

If a selected user does not have any job to submit, he/she is penalized in the same way using relation .The other users are rewarded the same way as in relation .

4.1.2 Automaton for scheduling Grid resources

This learning automaton is responsible for finding a near optimal solution to the problem of allocating the users' jobs to the Grid nodes according to their computational capacities. Initially all the processors are allocated at least one task so that all processors are

occupied. For each processor, the ratio of computational capacity to the queue length is computed and updated. When there is no idle processor and a task arrives, the scheduler checks the ratio computational capacity/queue length, the scheduler chooses the processor with the highest ratio. If the updated ratio does not cross the lowest value of the ratios, the selected processor (action) is rewarded using relation (1.1). If the updated ratio crosses the lowest value of the ratios, then the selected processor is penalized using relation (1.2).

4.2.4 Reward Penalty Module

The Reward Penalty module is responsible for updating the action probability vector of the action set by calculating reward and penalty. This module is used by both the User Submission module and the Grid Scheduling module. The reward and penalty is calculated using relations (1.1), (1.2) and (4.1) as needed.

5.CONCLUSION

The system architecture for two way proportional job scheduling using learning automata has been designed. The modules have been identified. The algorithm gives priority to the needy users and the users who submit more jobs are given preference in job submission in the future. Also, the algorithm allocates resources to tasks proportionate to the computational capacities of the processing nodes. As the proposed algorithm is an improvement over LAJS, it outperforms FPSO and GJS. The makespan, flowtime and load balancing advantages are preserved.

5.2 FUTURE WORK

The modules will be implemented using GridSim simulator using Java. The unpredictable arrival of jobs has to be dealt with. The fact that users could be busy for a while and idle thereafter also has to be taken into account. Also, the arrival of new users and the departure of old users has to be accounted for.

6.REFERENCES

[1] Xhafa, F., Gonzalez, J.A., Dahal, K.P. and Abraham, A.: A GA (TS) hybrid algorithm for scheduling in computational grids. In: Hybrid Artificial Intelligent Systems, Lecture Notes in Computer Science, vol. 5572, pp. 285–292 (2009).
[2] Xhafa, F., Alba, E., Dorronsoro, B., Duran, B. and Abraham, A.: Efficient batch job scheduling in Grids using cellular memetic algorithms. *J. Math. Model. Algorithms* 7(2), 217–236 (2008).

[3] Xhafa, F.: A hybrid evolutionary heuristic for job scheduling in computational Grids. In: *Studies in Computational Intelligence*, vol. 75. Springer, Berlin (2007) (Chap. 10).
[4] Bandieramonte, M., Di Stefano, A. and Morana, G.: An ACO inspired strategy to improve jobs scheduling in a Grid environment. In: *Lecture Notes in Computer Science*, vol. 5022, pp. 30–41 (2008).
[5] Cheng, W., Congfeng, J. and Xiaohu, L.: Fuzzy logic-based secure and fault tolerant job scheduling in Grid. *Tsinghua Sci. Technol.* 12(S1), 45–50 (2007).
[6] Kant, A., Sharma, A., Agarwal, S. and Chandra, S.: An ACO approach to job scheduling in Grid environment. In: *Lecture Notes in Computer Science*, vol. 6466, pp. 286–295 (2010).
[7] YarKhan, A. and Dongarra, J.: Experiments with scheduling using simulated annealing in a Grid environment. In: *Proceedings of GRID2002*, pp. 232–242 (2002).
[8] De Mello, R.F., Andrade Filho, J.A., Senger, L.J. and Yang, L.T.: Grid job scheduling using Route with genetic algorithm support. *Telecommun. Syst.* 38(3–4), 147–160 (2008).
[9] Kim S.S., Byeon J.H., Hongbo L., Abraham A. and McLoone S.: Optimal job scheduling in grid computing using efficient binary artificial bee colony optimization. *Soft Computing*, November 2012.
[10] Izakian H., Ladani B.T., Zamanifar K. and Abraham A.: A Novel Particle Swarm Optimization Approach for Grid Job Scheduling. *Communications in Computer and Information Science*, Volume 31, 2009, pp 100-109.