World Applied Sciences Journal 7 (Special Issue of Computer & IT): 152-160, 2009
ISSN 1818.4952

RASA: A New Task Scheduling Algorithm in Grid Environment

Saeed Parsa and Reza Entezari-Maleki

Parallel Processing and Concurrent Systems Laboratory,
Department of Computer Engineering, Iran University of Science and Technology (IUST), Tehran, Iran

Abstract: In this paper, a new task scheduling algorithm called RASA, considering the distribution and scalability characteristics of grid resources, is proposed. The algorithm is built through a comprehensive study and analysis of two well known task scheduling algorithms, Min-min and Max-min. RASA uses the advantages of the both algorithms and covers their disadvantages. To achieve this, RASA firstly estimates the completion time of the tasks on each of the available grid resources and then applies the Max-min and Min-min algorithms, alternatively. In this respect, RASA uses the Min-min strategy to execute small tasks before the large ones and applies the Max-min strategy to avoid delays in the execution of large tasks and to support concurrency in the execution of large and small tasks. Our experimental results of applying RASA on scheduling independent tasks within grid environments demonstrate the applicability of RASA in achieving schedules with comparatively lower makespan.

Key words: Gird computing . task scheduling algorithm . task completion time . max-min . min-min

INTRODUCTION

Grid is a large scale distributed system, concerned with coordinated resource sharing and problem solving. The grid infrastructure provides a mechanism to execute applications over autonomous and geographically distributed nodes by sharing resources which may belong to different individuals and institutions [1]. Computational grid is a kind of grid environments, targeted at solving computationally intensive problems. Computational grid is defined as a hardware and software infrastructure which provides dependable, consistent, pervasive and inexpensive access to computational resources existing on the network [1].

To make effective use of the tremendous capabilities of the computational grids, efficient task scheduling algorithms are required. Task scheduling algorithms are commonly applied by gird resource manager to optimally dispatch tasks to the grid resources [2-10]. Typically, grid users submit their own tasks to the grid manager to take full advantage of the grid facilities. The grid manager in a computational grid tries to distribute the submitted tasks amongst the grid resources in such a way that the total response time is minimized.

There are relatively a large number of task scheduling algorithms to minimize the total completion time of the tasks in distributed systems [3, 9, 11-14]. Actually, these algorithms try to minimize the overall completion time of the tasks by finding the most suitable resources to be allocated to the tasks. It should be noticed that minimizing the overall completion time of the tasks does not necessarily result in the minimization of execution time of each individual task.

Two well known examples of such algorithms are Max-min and Min-min [3, 9, 11, 12, 14]. These two algorithms estimate the execution and completion times of each of the tasks on each of the grid resources. Estimating the execution time of each task on different resources, the Min-min algorithm selects the task with minimum completion time and assigns it to the resource on which the minimum execution time is achieved. The algorithm applies a same procedure to the remaining tasks. A major difficulty with the Min-min algorithm is to assign the smaller tasks to the resources with relatively higher computational power. As a result, the makespan of the system is determined by the larger tasks if the number of the smaller tasks is more than the longer ones. To resolve the difficulty, the Max-min algorithm gives priority to the larger tasks. The Max-min algorithm firstly assigns the large or in other words time consuming tasks to the resources and then assigns the small ones. The Max-min seems to do better than the Min-min algorithm whenever the number of shorter tasks is much more than the longer ones, but in the other cases, early execution of the large tasks might increase the total response time of the system. Also, in the Max-min algorithm, the small tasks may wait for large ones to be executed.

Corresponding Author: Dr. Saeed Parsa, Parallel Processing and Concurrent Systems Laboratory, Department of Computer Engineering, Iran University of Science and Technology (IUST), Tehran, Iran

To resolve the difficulties of the Min-min and Max-min algorithms, these two algorithms could be executed alternatively, when assigning a new task to an appropriate resource. In this case, a large task is selected immediately after a small one and vice versa. Thereby, a comparatively better load balancing is achieved and the total response time of the grid system is improved.

This paper offers a novel task scheduling algorithm to resolve the above mentioned problems with the Min-min and Max-min algorithms. The algorithm, RASA (Resource Aware Scheduling Algorithm), applies the Max-min and Min-min strategies alternatively to assign tasks to the resources. RASA is implemented in GridSim environment. Practical experiments with GridSim demonstrate the benefits of applying RASA to schedule independent tasks within grid environments.

The remaining parts of this paper are organized as follows: Section 2 presents the related works. In Section 3, task scheduling in grid environments and several well known scheduling algorithms which are benchmarks of many other works, are introduced. In section 4, the new scheduling algorithm is proposed and one example is presented to illustrate the prominence of the new algorithm. Section 5 compares the scheduling algorithms in an assumed grid environment using GridSim; consequently, reports the obtained results of the simulation. Finally, section 6 concludes the paper and presents future works.

RELATED WORKS

Due to relatively high communication costs in grid environments most of the well known scheduling algorithms are not applicable in large scale distributed systems such as grid environments [3, 7, 8]. There has been an ongoing attempt to build scheduling algorithms specifically within grid environments.

X. He *et al*. have presented a new algorithm based on the conventional Min-min algorithm [3]. The proposed algorithm which is called QoS guided Min-min, schedules tasks requiring high bandwidth before the others. Therefore, if the bandwidth required by different tasks varies highly, the QoS guided Min-min algorithm provides better results than the Min-min algorithm. Whenever the bandwidth requirement of all of the tasks is almost the same, the QoS guided Min-min algorithm acts similar to the Min-min algorithm.

F. Dong *et al*. have proposed a similar algorithm called QoS priority grouping scheduling [12]. This algorithm, considers deadline and acceptation rate of the tasks and the makespan of the wholes system as major factors for task scheduling. In comparison with Min-min and QoS guided Min-min, the QoS priority grouping scheduling algorithm achieves better acceptance rate and completion time for the submitted tasks.

E. Ullah Munir *et al*. have presented a new task scheduling algorithm for grid environments called QoS Sufferage [13]. This algorithm considers network bandwidth and schedules tasks based on their bandwidth requirement as the QoS guided Min-min algorithm does. Compared with the Max-min, Min-min, QoS guided Min-min and QoS priority grouping algorithms, QoS Sufferage obtains smaller makespans.

K. Etminani *et al*. have proposed a new algorithm which uses Max-min and Min-min algorithms [14]. The algorithm determines to select one of these two algorithms, dependent on the standard deviation of the expected completion times of the tasks on each of the resources.

L. Mohammad Khanli *et al*. have presented QoS based scheduling solutions in a specific architecture called Grid-JQA [4, 5]. This scheduling solution applies an aggregation formula that is a combination of parameters together with weighting factors to evaluate QoS. Despite outperforming the Max-min, Min-min and Sufferage, the Khanli's scheduling algorithm is not practical and seems to be an unpractical mathematical solution [5].

A. Afzal *et al*. have proposed a new grid scheduling algorithm that minimizes the cost of the execution of workflows while ensuring that their associated QoS constraints are satisfied [15]. The algorithm views a grid environment as a queuing system and schedules tasks within this system. This algorithm is system oriented and considers the execution cost. Hence, it is suitable for economic grids. Since the algorithm is non-linear, as the size of the problem is gets large the time it takes to obtain a suitable scheduling becomes very long and unacceptable.

E. Elmroth *et al*. have proposed a user oriented algorithm for task scheduling in grid environments, using advanced reservation and resource selection [6]. The algorithm minimizes the total execution time of the individual tasks without considering the total execution time of all of the submitted tasks. Therefore, the overall makespan of the system does not necessarily get small.

B.T. Benjamin Khoo *et al*. have presented a new scheduling algorithm for workload distribution in grid environments [7]. This algorithm which is known as multiple resources scheduling (MRS) algorithm, takes into account both the site capabilities and the resource requirements of tasks. To quantify the performance of the algorithm, the average task wait times, queue completion times and average resource utilization factor, of the algorithm are compared to the

conventional backfill and replication algorithms. The comparison results indicate that this algorithm is more applicable than the other two algorithms.

B. Yagoubi *et al*. have offered a model to demonstrate grid architecture and an algorithm to schedule tasks within grid resources [8]. The algorithm tries to distribute the workload of the grid environment amongst the grid resources, fairly. Although, the mechanism used in [8] and other similar strategies which try to create load balancing within grid resources can improve the throughput of the whole grid environment, the total makespan of the system does not decrease, necessarily.

TASK SCHEDULING ALGORITHMS

Suppose that m resources $R_j(j = 1,...,m)$ have to process n tasks $T_i(i = 1,..., n)$. A schedule for each task is an allocation of one or more time intervals to one or more resources [16]. The expected execution time E_{ij} of task T_i on resource R_j is defined as the amount of time taken by R_j to execute T_i given R_j has no load when T_i is assigned. The expected completion time C_{ij} of task T_i on resource R_j is defined as the wall-clock time at which R_j completes T_i (after having finished any previously assigned tasks). Let b_i denote to the beginning of the execution of task T_i. From the above definitions, $C_{ij}=b_i+E_{ij}$. Let C_i be the completion time for task T_i and it is equal to C_{ij} where resource R_j is assigned to execute task T_i. The makespan for the complete schedule is then defined as $\text{Max}_{T_i \in K}(C_i)$. Makespan is a measure of the throughput of the heterogeneous computing system (like computational grid) [9, 11].

Decision about the assigning of tasks to the resources and Finding the best match between tasks and resources is NP-complete problem [3, 10, 11, 16]. Lots of scheduling algorithms are proposed to assign the tasks to the resources by considering one or several QoS parameters. These algorithms show different performances based on the environment in which they are used. The traditional parallel scheduling problem is to schedule the subtasks of an application on the parallel machines in order to reduce the turnaround time. In a grid environment, the scheduling problem is to schedule a set of tasks from different users on a set of computing resources to minimize the completion time of a specific task or the makespan of a system. Also, other parameters such as load balancing, system throughput, service reliability, service cost, system utilization and so forth can be considered.

Generally, the scheduling algorithms are divided into two basic categories; immediate mode scheduling and batch mode scheduling. In the immediate mode, a task is mapped onto a resource as soon as it arrives at the scheduler. In the batch mode, tasks are not mapped onto the resources as they arrive; instead they are collected into a set that is examined for mapping at prescheduled times called mapping events. The independent set of tasks which is considered for mapping at the mapping events is called a meta-task [14]. Some algorithms estimate the execution time of the tasks existing in the meta-task on the resources; then assign each task to the resource with the minimum expected execution time for that task. The algorithms with this mechanism are named as minimum execution time (MET) algorithms [3, 9, 11, 14]. The minimum completion time (MCT) algorithms assign each task to the resource which results in that task's earliest completion time. This causes some tasks to be assigned to the resources that do not have the minimum execution time for them [3, 9, 11, 14].

One of the earlier scheduling algorithms which do not use the execution or completion time of the tasks and schedules the tasks in the arbitrary order is opportunistic load balancing (OLB) algorithm. The insight behind OLB is to keep all resources as busy as possible. One advantage of OLB is its simplicity, but because OLB does not consider expected task execution times, the mappings it finds can result in very poor makespans [11]. In OLB algorithm, if multiple resources become ready at the same time, then one resource is arbitrarily chosen. The complexity of the OLB is dependent on the implementation. As an example, in the implementation considered in [9], the complexity of the algorithm is $O(m)$, where m is the number of all of the resources.

In contrast with OLB, Min-min and Max-min algorithms schedule tasks by considering the execution time of the tasks on the resources. The Min-min algorithm begins with the set U of all unscheduled tasks. Then, the set of minimum completion times for each of the tasks exiting in U is found. Next, the task with the overall minimum completion time from unscheduled tasks is selected and assigned to the corresponding resource (hence the name Min-min). Last, the newly scheduled task is removed from U and the process repeats until all tasks are scheduled [11]. The Min-min algorithm is shown in Fig. 1.

In Fig. 1, r_j denotes the expected time which resource R_j will become ready to execute a task after finishing the execution of all tasks assigned to it. First, the C_{ij} entries are computed using the E_{ij} (the estimated execution time of task T_i on resource R_j) and r_j values. For each task T_i, the resource that gives the earliest expected completion time is determined by scanning the ith row of the C matrix (composed of the C_{ij} values). The task T_k that has the minimum earliest

1. **for** all tasks T_i in meta-task M_v
2. **for** all resources R_j
3. $C_{ij}=E_{ij}+r_j$
4. **do** until all tasks in M_v are mapped
5. for each task in M_v find the earliest completion time and the resource that obtines it
6. find the task T_k with the ***minimum*** earliest completion time
7. assigne task T_k to the resource R_l that gives the earliest completion time
8. delete task T_k from M_v
9. update r_l
10. update C_{il} for all i
11. **end do**

Fig. 1: The Min-min algorithm

expected completion time is determined and then assigned to the corresponding resource. The matrix C and vector r are updated and the above process is repeated for tasks that have not yet been assigned to a resource.

The Max-min algorithm is similar to the Min-min algorithm. It differs from the Min-min algorithm in that once the resource that provides the earliest completion time is found for every task, the task T_k that has the maximum earliest completion time is determined and then assigned to the corresponding resource. That is, in line (6) of Fig. 1, *"minimum"* would be changed to *"maximum"* [9].

Both the Min-min and Max-min algorithms consider a hypothetical assignment of tasks to resources, projecting when a resource will become idle based on the hypothetical assignment. Both algorithms have time complexities of $O(mn^2)$, where m is the number of resources in the system and n is the number of tasks which should be scheduled to be executed [10].

THE NEW SCHEDULING ALGORITHM

Let T_i be the first task mapped by Min-min onto the resource, R_j. According to the Min-min algorithm, R_j should execute T_i in the shortest possible time, compared with the other resources. The remaining tasks are assigned to the fastest resource, R_j, as long as the total execution times of the task assigned to the R_j is less than the time it takes to execute the tasks on the other resources. This approach results in a comparatively shorter makespan if the execution time of the tasks varies slightly because it attempts to assign the tasks to the fastest resources. However, if there are large and small tasks, the large ones may be assigned to slower resources and the makespan of the system will be increased, dramatically.

The Max-min algorithm seems to do better than the Min-min algorithm in the cases when the number of short tasks is much more than the long ones. For example, if there is only one long task, the Max-min algorithm executes many short tasks concurrently with the long task. In this case, the makespan of the system is most likely determined by the execution time of the long task. However, since the Min-min algorithm attempts to assign the short tasks before the long one, the makespan increases compared with the Max-min. On the other hand, mapping the longest task to the fastest resource provides a better opportunity for concurrent execution of the small tasks on different resources. In this certain situation, the Max-min provides a better mapping which supports load balancing across the grid resources more than the Min-min [11]. Although load balancing in small scale distributed systems is desirable and leads to reduced total completion times however, in large scale distributed systems load balancing does not necessarily results in the shortest makespan. The proposed algorithm outperforms Max-min in large scale systems, because it focuses on minimizing the completion time of tasks rather than load balancing.

In subsection 4.1, the new algorithm is proposed and in subsection 4.2, an illustrative example is presented to compare the proposed algorithm with existing algorithms.

RASA: Our proposed grid scheduling algorithm, RASA, is presented in Fig. 2. The algorithm builds a matrix C where C_{ij} represents the completion time of the task T_i on the resource R_j. If the number of available resources is odd, the Min-min strategy is applied to assign the first task, otherwise the Max-min strategy is applied. The remaining tasks are assigned to their appropriate resources by one of the two strategies, alternatively. For instance, if the first task is assigned to a resource by the Min-min strategy, the next task will be assigned by the Max-min strategy. In the next round the task assignment begins with a strategy different from the last round. For instance if the first round begins with the Max-min strategy, the second round will begin with the Min-min strategy.

Experimental results show that if the number of available resources is odd it is preferred to apply the Min-min strategy the first in the first round otherwise is better to apply the max-min strategy the first. Alternative exchange of the Min-min and Max-min strategies results in consecutive execution of a small and a large task on different resources and hereby, the waiting time of the small tasks in Max-min algorithm and the waiting time of the large tasks in Min-min algorithm are ignored. As RASA is consist of the

```
1. for all tasks Ti in meta-task Mv
2.     for all resources Rj
3.         Cij=Ej+rj
4. do until all tasks in Mv are mapped
5.     if the number of resources is even then
6.         for each task in Mv find the earliest completion
               time and the resource that obtines it
7.         find the task Tk with the maximum earliest
               completion time
8.         assigne task Tk to the resource Rl that gives the
               earliest completion time
9.         delete task Tk from Mv
10.        update rl
11.        update Cil for all i
12.    else
13.        for each task in Mv find the earliest completion
               time and the resource that obtines it
14.        find the task Tk with the minimum earliest
               completion time
15.        assigne task Tk to the resource Rl that gives the
               earliest completion time
16.        delete task Tk from Mv
17.        update rl
18.        update Cil for all i
19.    end if
20.end do
```

Fig. 2: The proposed algorithm (RASA)

Max-Min and Min-Min algorithms and have no time consuming instruction, the time complexity of RASA is $O(mn^2)$ where m is the number of resources and n is the number of tasks (similar to Max-min and Min-min algorithms).

An illustrative example: As a simple example, assume there is a grid environment with two resources. The processing speed of the resources and the bandwidth of the communication links which connect each of the resources to the grid manager are shown in Table 1. Four tasks T_1, T_2, T_3 and T_4 are in the meta-task M_v and the grid manager is supposed to schedule all the tasks within M_v on two resources R_1 and R_2. Table 2 represents the volume of instructions and data in the tasks T_1 to T_4.

Applying the data presented in Table 1 and 2, it is possible to calculate the expected completion time of the tasks on each of the resources. The calculated completion time of the tasks are demonstrated in Table 3.

Figure 3 includes two Gantt charts representing the results of applying Max-min and Min-min algorithms on the meta-task M_v, described in Table 1 and 2. Although the orders of the tasks scheduled in Max-min and Min-min algorithms are different, the makespan of the system is equal when applying each of the algorithms.

Table 1: Specification of the resources

Resources	Processing speed (MIPS)	Related bandwidth (Mbps)
R_1	50	100
R_2	100	5

Table 2: Specification of the tasks

Tasks	Volume of instructions (MI)	Volume of data (Mb)
T_1	128	44
T_2	69	62
T_3	218	64
T_4	21	59

Table 3: Completion time of the tasks on each of the resources

Tasks/Resources	R_1	R_2
T_1	3	10
T_2	2	13
T_3	5	15
T_4	1	12

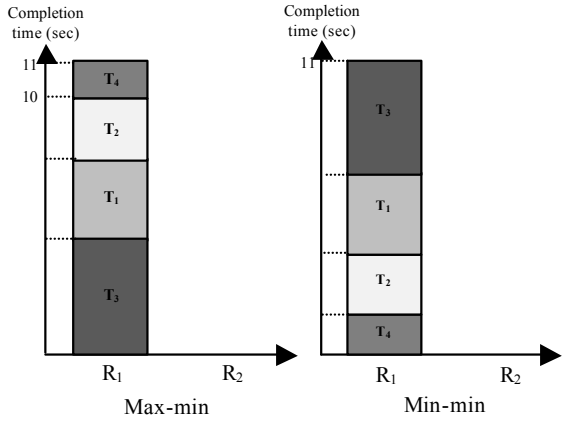

Fig. 3: Gantt chart of the Max-min and Min-min algorithms

The result of applying RASA on meta-task M_v is exposed in Fig. 4. As shown in Fig. 4, the makespan of the system when applying RASA is 10 second, whilst Max-min and Min-min provide a scheduling with a makespan of 11 seconds.

Also, considering the completion times in Fig. 3 and 4 it is observed that RASA outperforms Max-min and Min-min algorithms by providing relatively smaller makespan and higher load balancing. It is assumed that tasks can be executed on any of the resources, independently. Therefore, the QoS guided Min-min and QoS priority grouping algorithms act the same as Min-min algorithm. The above example is only one

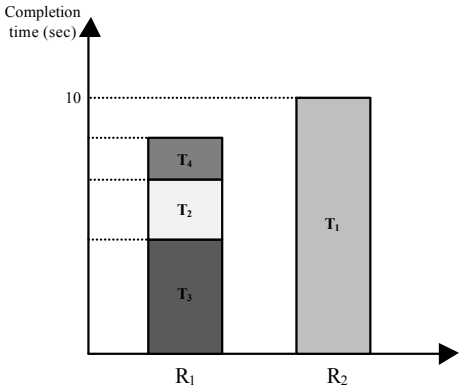

Fig. 4: Gantt chart of RASA

case which shows the privilege of RASA compared to the other algorithms. However, one can provide an example in which the results of RASA are equal or even worse than the other above mentioned algorithms. In general when the submitted tasks have almost the same size, the Min-min or Max-min algorithms may act better than RASA.

SIMULATION AND EXPERIMENTAL RESULTS

To evaluate and compare RASA with other algorithms such as Max-min, Min-min, OLB, QoS guided Min-min and QoS priority grouping, a simulation environment known as GridSim toolkit [17] has been used. In order to demonstrate the preeminence of RASA in comparison with other algorithms, two different assumptions are made:

Assumptions I: The computation time of the tasks overcomes to the communication time of them. This situation occurs in multiprocessors and small scale distributed systems (such as cluster environments).

Assumptions II: The communication time becomes more and even can be overcomes to the computation time of the tasks. This situation occurs in large scale distributed systems such as grid environments in which the resources are widely distributed and connected via the communication links.

The above mentioned algorithms have been simulated in GridSim environment. It has been assumed that there are no constraints for executing tasks on different resources and each of the tasks could be executed on each of the resources. In this situation, the QoS guided Min-min and QoS priority grouping algorithms performed the same as the Min-min algorithm. Therefore, in all the plots presented in the following figures, only the results of Min-min are depicted and the plots related to QoS guided Min-min and QoS priority grouping algorithms are eliminated. In order to study the effect of the volume of workload on the efficacy of RASA and to compare it with the other algorithms, in each of the above mentioned assumptions, three different workloads of light, medium and heavy load are considered. In light load, 200 tasks are dispatched within 10 to 11 grid resources and

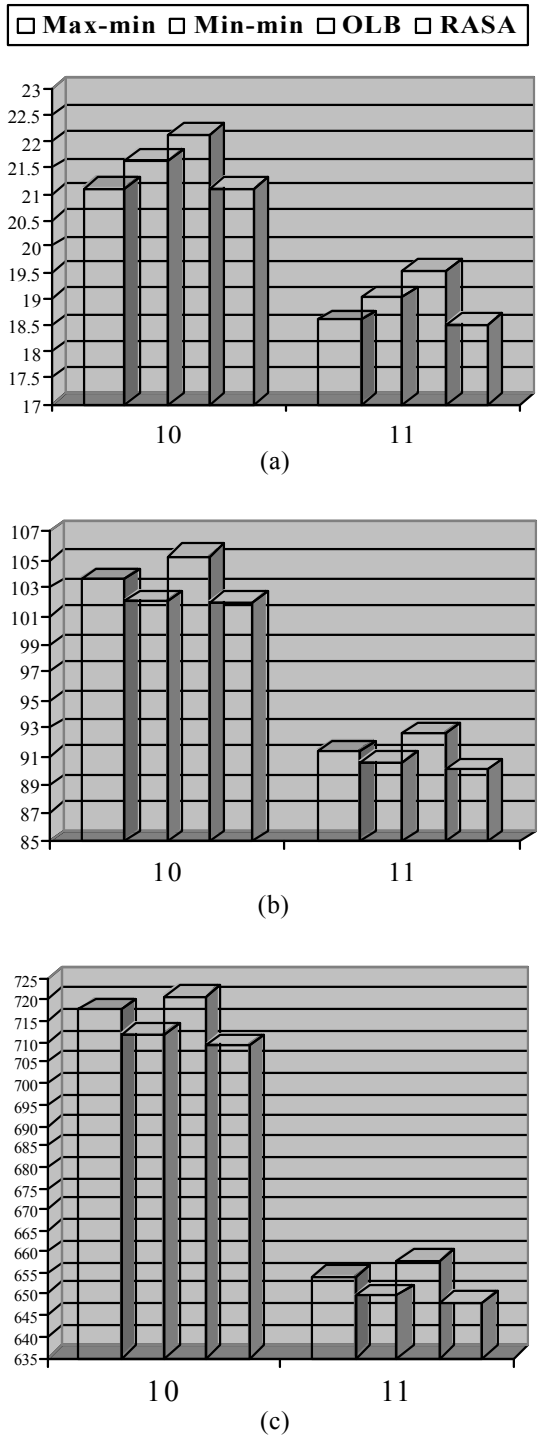

Fig. 5: Makespans of the algorithms in scenario I

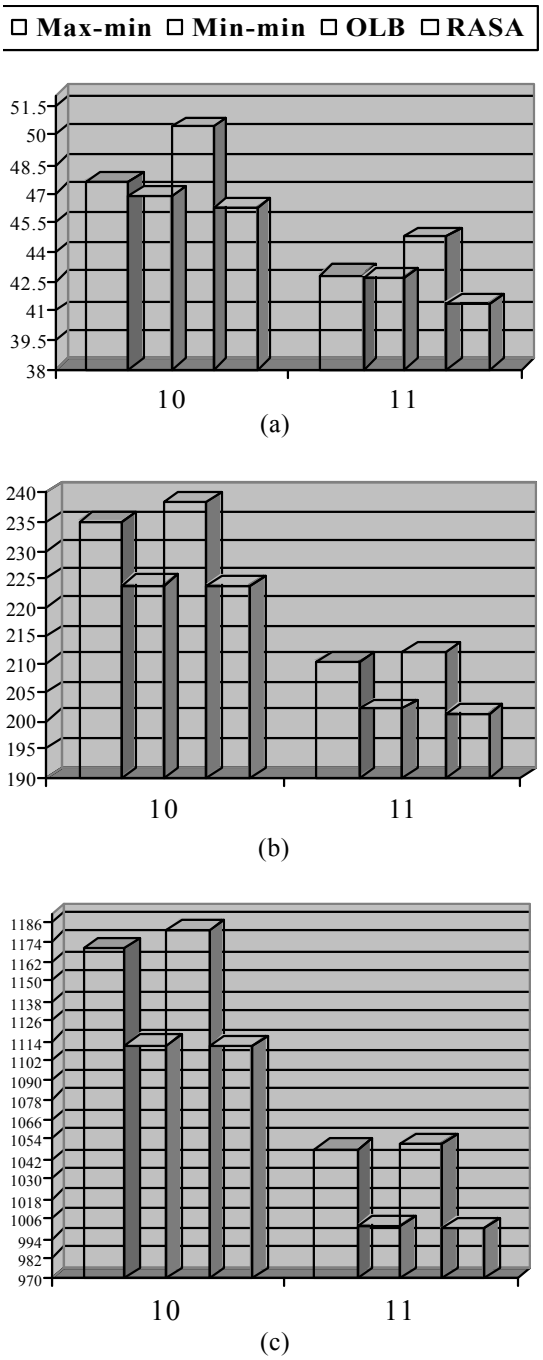

Fig. 6: Makespans of the algorithms in assumption II

makespan is estimated for each of the algorithms. Also, in medium load, 1000 tasks and in heavy load, 5000 tasks have been considered. Figure 5 and 6 show the makespans achieved by applying the algorithms, considering Assumption I and Assumption II, respectively.

As shown in Fig. 5a, when the computation time of the tasks exceeds their communication time and the workloads of the resources are light, Max-min returns relatively smaller makespan than the other algorithms.

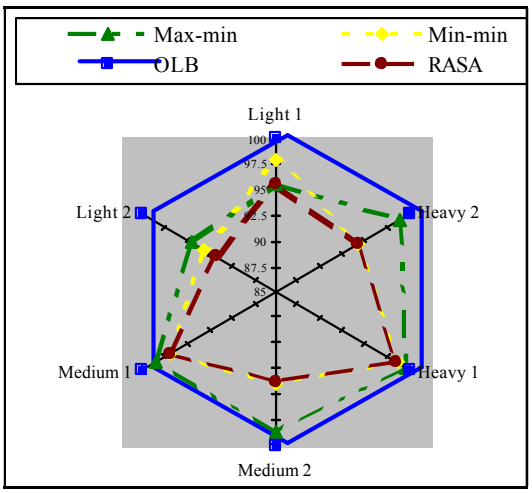

Fig. 7: Makespans of the algorithms when the number of resources is 10

As the workload of the resources increases, the makespans achieved by the Min-min and RASA get smaller. As shown in Fig. 5b and 5c, the makespan returned by RASA is less than the makespans returned by the other algorithms. Considering Fig. 5b and 5c, it can be concluded that; RASA returns smaller makespans compared with Max-min, Min-min, OLB, QoS guided Min-min and QoS priority grouping algorithms, in small scale distributed systems.

Figure 6 shows the makespans of the algorithms based on Assumption II. As shown in Fig. 6a, when the workload is light, the makespan returned by RASA is less than the other algorithms. Comparing Fig. 5a and 6a, it is observed that; Max-min is suitable for small scale distributed systems whilst Min-min well suits large scale distributed systems. However, RASA returns relatively smaller makespans than both the Min-min and Max-min algorithms in both small scale and large scale distributed systems. When the workload of the resources is heavy, RASA achieves smaller makespans in comparison with the other algorithms. Therefore, RASA achieves smaller makespans both in light and heavy load conditions.

For the sake of clarity, in Fig. 5 and 6 the makespans returned by different algorithms are rescaled. Here, the largest makespan, returned by OLB, is 100 and the other makespans are rescaled with respect to this amount. The six different cases observed in Fig. 5 and 6 are considered as vertices of a regular hexagon. The vertices of the hexagon are named as Light 1, Light 2, Medium 1, Medium 2, Heavy 1 and Heavy 2 which imply light load in Assumption I, light load in Assumption II, medium load in Assumption I, medium load in Assumption II, heavy load in Assumption I and heavy load in Assumption II,

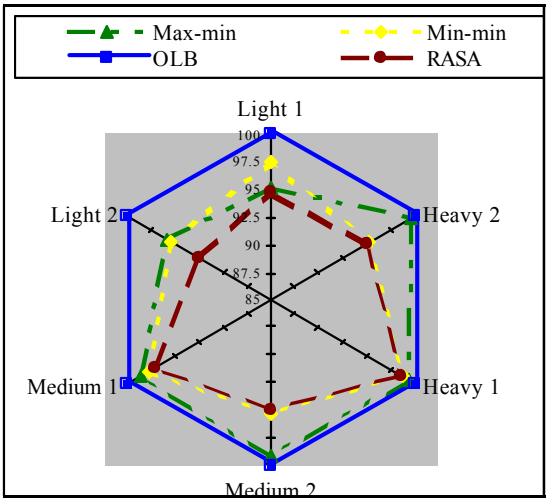

Fig. 8: Makespans of the algorithms when the number of resources is 11

respectively. The hexagon is shown in Fig. 7. In this hexagon the number of grid resources is assumed to be 10. The number of grid resources, in Fig. 8, is 11. As shown in Fig. 7 and 8 the area of the non-regular hexagon which is related to RASA, is smaller than the other algorithms. Therefore, it can be concluded that; the makespans returned by RASA are smaller than the other algorithms in almost all different conditions.

CONCLUSION AND FUTURE WORKS

Min-min and Max-min algorithms are applicable in small scale distributed systems. When the number of the small tasks is more than the number of the large tasks in a meta-task, the Min-min algorithm can not schedule tasks, appropriately and the makespan of the system gets relatively large. Unlike the Min-min algorithm, the Max-min algorithm attempts to achieve load balancing within resources by scheduling the large tasks prior to the small ones. However, within a computational grid environment high throughput is of great interest rather than the load balancing. To achieve this, in this article, a new task scheduling algorithm, RASA, is proposed.

RASA is composed of two traditional scheduling algorithms; Max-min and Min-min. RASA uses the advantages of Max-min and Min-min algorithms and covers their disadvantages. The experimental results obtained by applying RASA within the GridSim simulator, shows that RASA is outperforms the existing scheduling algorithms in large scale distributed systems.

This study is only concerned with the number of the resources to be odd or even and analyses the merits and drawbacks of two well known traditional algorithms, Max-min and Min-min. In this paper, the deadline of each task, arriving rate of the tasks, cost of the task execution on each of the resource, cost of the communication and many other cases that can be a topic of research are not considered. Also, applying the proposed algorithm on actual grid environment for practical evaluation can be other open problem in this area.

ACKNOWLEDGMENT

The authors want to express their gratitude to the Iranian National Elite Foundation for their financial support of this paper. Also, the authors would like to express their cordial thanks to Miss Pegah Moradi-Hamed and Miss Marzie Mehdi-Beyraghdar for their valuable assist.

REFERENCES

1. Foster, I. and C. Kesselman, 2004. The Grid 2: Blueprint for a New Computing Infrastructure. Second Edition. Elsevier and Morgan Kaufmann Press.
2. Chunlin, L. and L. Layuan, 2006. QoS based resource scheduling by computational economy in computational grid. Journal of Information Processing Letters, 98: 119-126.
3. He, X., X-He Sun and G.V. Laszewski, 2003. QoS Guided Min-min Heuristic for Grid Task Scheduling. Journal of Computer Science and Technology, 18: 442-451.
4. Mohammad Khanli, L. and M. Analoui, 2008. Resource Scheduling in Desktop Grid by Grid-JQA. The 3rd International Conference on Grid and Pervasive Computing, IEEE.
5. Mohammad Khanli, L. and M. Analoui, 2007. Grid_JQA: A QoS Guided Scheduling Algorithm for Grid Computing. The Sixth International Symposium on Parallel and Distributed Computing (ISPDC'07), IEEE.
6. Elmroth, E. and J. Tordsson, 2008. Grid resource brokering algorithms enabling advance reservations and resource selection based on performance predictions. Journal of Future Generation Computer Systems, 24: 585-593.
7. Benjamin Khoo, B.T. B. Veeravalli, T. Hung and C.W. Simon See, 2007. A multi-dimensional scheduling scheme in a Grid computing environment. Journal of Parallel and Distributed Computing, 67: 659-673.
8. Yagoubi, B. and Y. Slimani, 2007. Task Load Balancing Strategy for Grid Computing. Journal of Computer Science, 3 (3): 186-194.

9. Maheswaran, M., Sh. Ali, H. Jay Siegel, D. Hensgen and R.F. Freund, 1999. Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems. Journal of Parallel and Distributed Computing, 59: 107-131.
10. Freund, R.F., M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J.D. Lima, F. Mirabile, L. Moore, B. Rust and H.J. Siegel, 1998. Scheduling Resource in Multi-User, Heterogeneous, Computing Environment with SmartNet. In the Proceeding of the Seventh Heterogeneous Computing Workshop.
11. Braun, T.D., H. Jay Siegel, N. Beck, L.L. Boloni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys and B. Yao, 2001. A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. Journal of Parallel and Distributed Computing, 61: 810-837.
12. Dong, F., J. Luo, L. Gao and L. Ge, 2006. A Grid Task Scheduling Algorithm Based on QoS Priority Grouping. In the Proceedings of the Fifth International Conference on Grid and Cooperative Computing (GCC'06), IEEE.
13. Ullah Munir, E., J. Li and Sh Shi, 2007. QoS Sufferage Heuristic for Independent Task Scheduling in Grid. Information Technology Journal, 6 (8): 1166-1170.
14. Etminani, K. and M. Naghibzadeh, 2007. A Min-min Max-min Selective Algorithm for Grid Task Scheduling. The Third IEEE/IFIP International Conference on Internet. Uzbekistan.
15. Afzal, A., A. Stephen McGough and J. Darlington, 2008. Capacity planning and scheduling in Grid computing environment. Journal of Future Generation Computer Systems, 24: 404-414.
16. Brucker, P., 2007. Scheduling Algorithms. Fifth Edition. Springer Press.
17. Buyya, R. and M. Murshed, 2002. GridSim: A toolkit for the odeling and simulation of distributed resource management and scheduling for grid computing . Journal of Concurrency and Computation Practice and Experience, pp: 1175-1220.