INTERNATIONAL JOURNAL OF RESEARCH IN COMPUTER APPLICATIONS AND ROBOTICS
ISSN 2320-7345

A SURVEY OF SCHEDULING ALGORITHMS IN GRID COMPUTING

Maryam Masoudi Khorsand[1], Mehdi Effatparvar[2], Mahdi Kanani [3]

[1] Department of Computer Engineering, Sama Technical and Vocational Training College, Islamic Azad University, Ardabil Branch, Ardabil, Iran, maryam_masoudi_khorsand@yahoo.com
[2]Department of Computer Engineering, Ardabil Branch, Islamic Azad University, Ardabil, Iran, Me.Effatparvar@gmail.com
[3]Department of Computer Engineering, Germi Branch, Islamic Azad University, Germi, Ardabil, Iran, Kanani.mahdi@ymail.com

Author Correspondence: Iran, Ardabil, Sama Technical and Vocational Training College, Islamic Azad University, Ardabil Branch, 00989141508575//00984516617525maryam_masoudi_khorsand@yahoo.com

Abstract

Grid computing systems involve a collection of programs and resources that are distributed among the machines of the grid. The goal of grid computing is to manage the system such that set the jobs to be done in shortest time. one of the aspects that improve the efficiency and decrease the time to performance, is the jobs Scheduling. So the scheduling of the most fundamental challenges in the areas in this systems, which is considered in this paper. Various algorithms used for scheduling the jobs in grid computing. In this paper, we compared three examples of this algorithms and evaluated their considering the advantages and disadvantages.

Keywords: Grid Computing; Load Balancing; Scheduling, Makespan.

1. INTRODUCTION

Grid computing is a new technology that the goal is to share resources for the job to come. In computing grid, job scheduling is a very important problem. This causes its resource allocation process, the interaction with user tasks and so on are different with grid computation. The grid computing system management should be done in such a way that the set of jobs to be performed in the shortest possible time. A good scheduling algorithm can assign jobs to resources efficiently and can balance the system load. In this paper be reviewed scheduling algorithms that overall goal in all of them is increase the efficiency and reduce the makespan for grid computing.

2. BACKGROUND

2-1 Scheduling in Grid Computing

One of the aspects that improve the efficiency and reduce the execution time in grid computing, is job scheduling. The purpose of scheduling job this is that be specified each job run when and in what machines. In this paper three scheduling algorithm is Reviewed, that overall goal of their is increasing the efficiency in grid computing. Scheduling system, controls different duties in grid computing to increase job completion rates and productivity of resources and as a result increase in computing power. Therefore scheduling of jobs across the different non uniform physical machines is a critical duty.
Large numbers of task scheduling algorithms are available to minimize the makespan [1], [2]. All these algorithms try to find resources to be allocated to the tasks which will minimize the overall completion time of the jobs [3], [4], [5], [6]. Minimizing overall completion time of the tasks does not mean that it minimizes the actual execution time of individual task.

2-2 Makespan

In static heuristics, the accurate estimate of the expected execution time for each task on each machine is known in advance[7]. These times are stored in an ETC (Expected Time to Compute) matrix where ETC (t_i,m_j) is the estimated execution time of task i on machine j. The main objective of the heuristic scheduling algorithms is to minimize the completion time of last finished task. The makespan is computed as follows:
Let task set T = t1,t2,t3,…,tn. be the group of tasks submitted to scheduler and Let Resource set R =m1,m2,m3,…,mk Be the set of resources available at the time of task arrival makespan produced by any algorithm for a scheduler can be calculated as follows:

$$\text{makespan} = \max(CT(t_i, m_j)) \quad (1)$$

$$CTij = Rj + ETij \quad (2)$$

where CTij is Completion Time of task i on resource j, ETij expected execution time of job i on resource j. and Rj is the ready time or availability time of resource j; the time when machine mj complete execution of all the prior assigned tasks.

3. REVIEWS OF SCHEDULING ALGORITHMS IN GRID

In this section three scheduling algorithms in grid environments are compared and in next section their advantages and disadvantages are discussed.

3-1 Min-Min Algorithm

The Min-min heuristic begins with the set U of all unmapped tasks[7]. Then, the set of minimum completion time M for each task in U is found. Next, the task with the overall minimum completion time from M is selected and assigned to the corresponding machine (hence the name Min-min). Last, the newly mapped task is removed from U, and the process repeats until all tasks are mapped (i.e., U is empty) [8],[9].
Min-min is based on the minimum completion time, as is MCT. However, Min-min considers all unmapped tasks during each mapping decision and MCT only considers one task at a time. Min-min maps the tasks in the order that changes the machine availability status by the least amount that any assignment could. Let ti be the first task mapped by Min-min onto an empty system. The machine that finishes ti the earliest, say mj, is also the machine that executes ti the fastest. For every task that Min-min maps after ti, the Min-min heuristic changes the availability status of mj by the least possible amount for every assignment. Therefore, the percentage of tasks assigned to their first choice (on the basis of execution time) is likely to be higher for Min-min than for Max-min (see below). The expectation is that a smaller makespan can be obtained if more tasks are assigned to the machines that complete them the earliest and also execute them the fastest. min-min procedure is shown in figure1. Min-min flowchart is shown in figure2.

1. **For** all tasks $\mathbf{t_i}$ in **MT** (in an arbitrary order)
2. **For** all machines m_j (in fixed arbitrary order)
3. $\mathbf{C_{ij}=E_{ij}+r_j}$
4. **Do** until all tasks in **MT** are mapped
5. For each task in **MT** find the earliest completion time and the machine that obtains it.
6. Find the task $\mathbf{t_k}$ with the **minimum** earliest completion time.
7. assign task $\mathbf{t_k}$ to the machine e $\mathbf{m_l}$ that gives the earliest completion time.
8. Delete task $\mathbf{t_k}$ from **MT**
9. Update $\mathbf{r_l}$
10. Update $\mathbf{C_{il}}$ for all **i**
11. **End Do**.

Figure1: Min-Min Algorithm

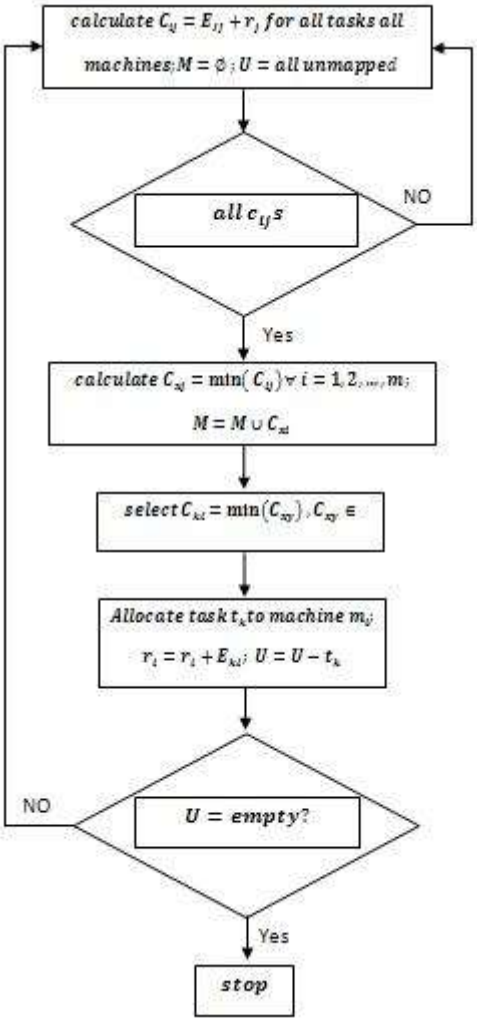

Figure2: Flowchart of Min-Min Algorithm

3.2 LBMM Algorithm

The algorithm starts by executing the steps in Min-Min strategy first.[10] It first identifies the task having minimum execution time and the resource producing it. Thus the task with minimum execution time is scheduled first in Min-Min. After that it considers the minimum completion time since some resources are scheduled with some tasks. Since Min-Min chooses the smallest tasks first it loads the fast executing resource more which leaves the other resources idle. But it is simple and produces a good makespan compared to other algorithms. LBMM procedure is shown in figure3.

```
for all tasks Ti
 for all resources
    Cij=Eij+rj
 do until all tasks  are mapped
    for each task find the earliest completion time and the
resource that obtains it
    find the task Tk with the minimum earliest completion time
    assign task Tk to the resource Rl that gives the earliest
completion time
   delete task Tk from list
   update ready time of resource Rl
   update Cil for all i
end do
// rescheduling to balance the load
sort the resources in the order of completion time
for all resources R
Compute makespan = max(CT(R))
End for
for all resources
  for all tasks
    find the task Ti that has minimum ET in Rj
    find the MCT of task Ti
    if MCT < makespan
       Reschedule the task Ti to the resource that produces it
       Update the ready time of both resources
    End if
  End for
End for
//Where MCT represents Maximum Completion Time
```

Figure 3. LBMM Algorithm

So LBMM executes Min-Min in the first round. In the second round it chooses the resources with heavy load and reassigns them to the resources with light load. LBMM identifies the resources with heavy load by choosing the resource with high makespan in the schedule produced by Min-Min. It then considers the tasks assigned in that resource and chooses the task with minimum execution time on that resource. The completion time for that task is calculated for all resources in the current schedule. Then the maximum completion time of that task is compared with the makespan produced by Min-Min. if it is less than makespan then the task is rescheduled in the resource that produces it, and the ready time of both resources are updated. Otherwise the

next maximum completion time of that task is selected and the steps are repeated again. The process stops if all resources and all tasks assigned in them have been considered for rescheduling. Thus the possible resources are rescheduled in the resources which are idle or have minimum load. This makes LBMM to produce a schedule which increases load balancing. Since it compares the maximum completion time with makespan LBMM reduces the overall completion time also. The steps to be carried out in the second phase of LBMM are shown in figure 4.

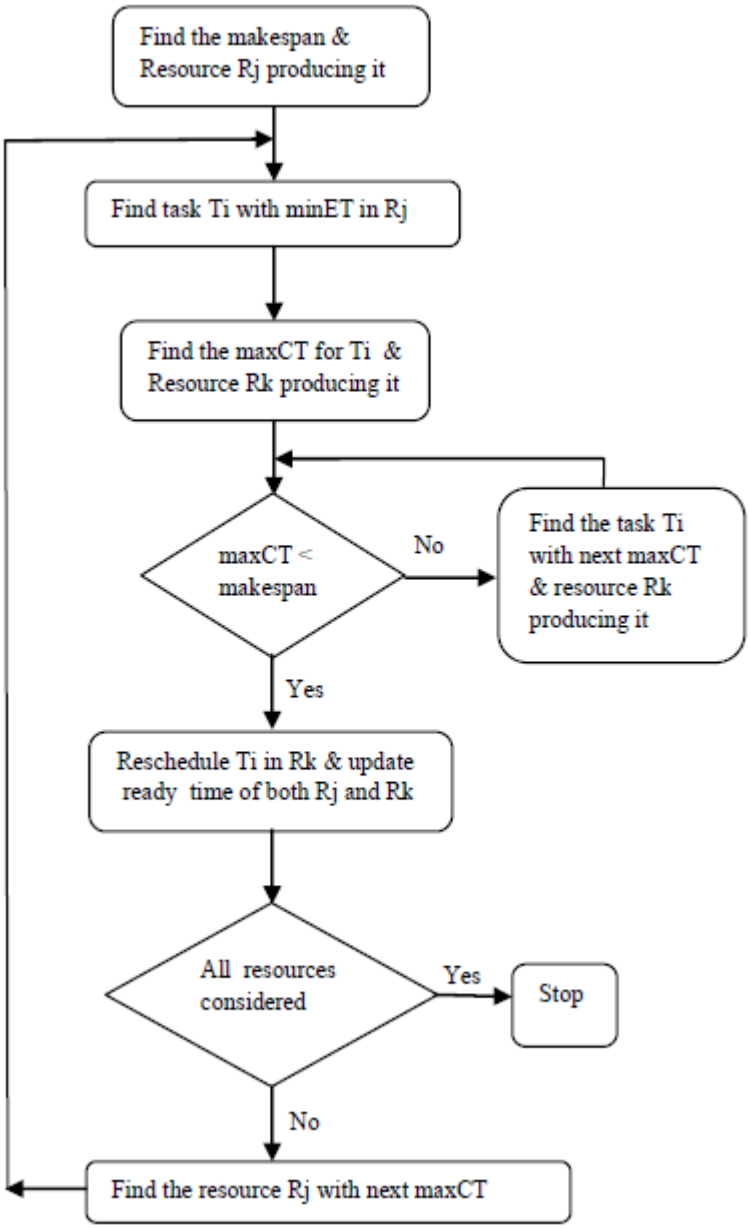

Figure 4. Flowchart of LBMM Algorithm

3-3 Best-Min Algorithm

In this section, the best-min scheduling algorithm will be described [7]. The best-min algorithm uses minmin to get the makespan in first step and reschedule the tasks in the second phase in order to reduce the makespan. There is a condition in algorithm that best-min should consider all the resources in grid environment and this is caused to maximize resource utilization as well. This algorithm uses the state of the art rescheduling technique. The algorithm is presented in algorithm 2. At the first step, best-min, starts with executing min-min. As described in section 2-3, since min-min starts mapping the tasks with minimum execution times first, it makes

faster resources busy and slower resource idle. As a result, min-min produces a good makespan comparing to other heuristics. On the other hand, the main drawback of the min-min algorithm is its poor load balanced and resource utilization. Hence, best-min runs min-min first to find the makespan of the whole system and the resource (R_j) that produces that makespan. Makespan and Rj are used in the rest of the best-min algorithm. In the second step, the best-min finds the tasks that are assigned to R_j according to ETC matrix or MT conditions, and chooses the task(T_i) with maximum execution time(Max ET) on R_j .Then, the completion time of the task T_i is calculated for all the other resources.

The minimum completion time of T_i on all the resources is called Min CT. Min CT is compared against the makespan produced by min-min. If Min CT is less than makespan then the task is rescheduled on the resource that produced it and the available time for all resources is updated.

Otherwise, T_i is scheduled in current resource(R_j) and scheduler looks for the next task with Max ET. This will be continued till all resources have been considered and all tasks have been mapped.

Comparing the Min CT of the task T_i on other resources and the makespan produced by min-min results in best-min's makespan to never be bigger than min-min's makespan. Also best-min has a good resource utilization. As mentioned, the reason for this is that the scheduler should consider all the available resources to schedule tasks.

Figure 5 only shows the second phase of this algorithm since best-min runs min-min in the first step. The best-min flowchart is also shown in figure 6.

1. **For** all tasks $\mathbf{t_i}$ in **MT** (in an arbitrary order)
2. **For** all machines m_j (in fixed arbitrary order)
3. $\mathbf{C_{ij}=E_{ij}+r_j}$
4. **Do** until all tasks in **MT** are mapped
5. For each task in **MT** find the earliest completion time and the machine that obtains it.
6. Find the task $\mathbf{t_k}$ with the **minimum** earliest completion time.
7. assign task $\mathbf{t_k}$ to the machine e $\mathbf{m_l}$ that gives the earliest completion time.
8. Delete task $\mathbf{t_k}$ from **MT**
9. Update $\mathbf{r_l}$
10. Update $\mathbf{C_{il}}$ for all **i**
11. **End Do**.
12. For all machine
13. Compute Makespan =max(CT(R))
14. End for // The machine that produced the makespan is identified as R_j
15. For all machine
16. For all tasks
17. Find the task $\mathbf{T_i}$ with the **max ET** in $\mathbf{R_j}$
18. Find the **min CT** of task $\mathbf{T_i}$
19. End for
20. If **Min CT** < Makespan
21. Update the ready Time of both machine .
22. Reschedule task $\mathbf{T_i}$ to the machine that produces it
23. End if
24. End for.

Figure5: **Best-Min Algorithm**

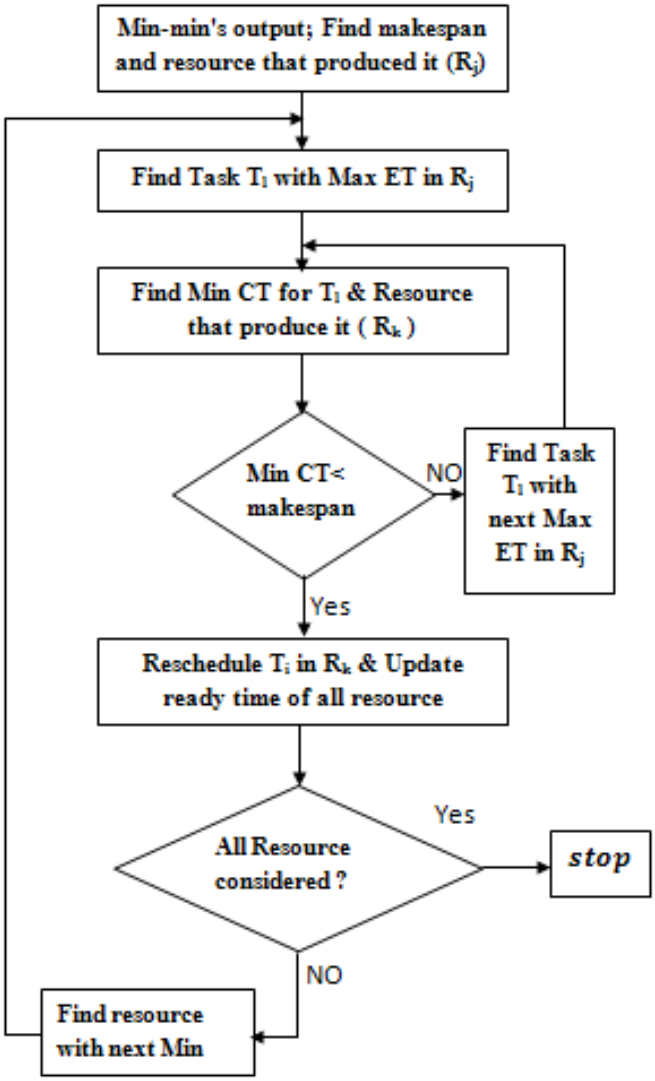

Figure6: Flowchart of Best Min Algorithm

4. THE COMPARISON OF MENTIONED SCHEDULING ALGORITHM

In this part three scheduling algorithms is investigated by comparing their advantages and disadvantages are shown in Table 1. Observations show that the min-min algorithm unable to load balancing reduction unlike LBMM and best-min algorithms. According to consider the runtime as well as min-min and LBMM algorithms optimal does not work. Also inattention to cost factors is the main disadvantage of the LBBM algorithm.

Table1. Comparison of Scheduling Algorithms

Parameters	Advantages	Disadvantages
Min- Min Algorithm	Reducing makespan	No load balancing
LBMM Algorithm	Reducing makespan and establishing load balancing	Disregarding cost factor when is high the number of tasks and resources
Best-min Algorithm	Increase resource utilization and establishing load balancing	Increasing the time for resource allocation

5. Conclusion and Future Studies

In this paper, three scheduling algorithm in the grid computing were investigated. although there have been a lot of studies concerning task scheduling in distributed and parallel systems, a new challenge still can be interesting and many research projects can be done. The present paper was an attempt to focus on current scheduling algorithms and the mentioned algorithms were compared considering a variety of aspect and advantages and disadvantages of each one were explained.

6. REFERENCES

[1] He. X, X-He Sun, and Laszewski. G.V, "QoS Guided Minmin Heuristic for Grid Task Scheduling," Journal of Computer Science and Technology, Vol. 18, pp. 442-451, 2003.

[2] Sameer Singh Chauhan,R. Joshi. C, QoS Guided Heuristic Algorithms for Grid Task Scheduling, International Journal of Computer Applications (0975 – 8887), pp 24-31, Volume 2, No.9, June 2010.

[3] Dong. F, Luo. J, Gao. L and Ge. L, "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, 2006.

[4] Etminani .K, and Naghibzadeh. M, "A Min-min Max-min Selective Algorithm for Grid Task Scheduling," The Third IEEE/IFIP International Conference on Internet, Uzbekistan, 2007.

[5] Ranganathan, K. and Foster, I., "Decoupling Computation and Data Scheduling in Distributed Data Intensive Applications", Proceedings of the 11th IEEE Symposium on High Performance Distributed Computing (HPDC-11), Edinburgh, Scotland, July 2002.

[6] Ullah Munir. E, Li. J, and Shi. Sh, QoS Sufferage Heuristic for Independent Task Scheduling in Grid. Information Technology Journal, 6 (8): 1166-1170, 2007.

[7] Meraji, S and Salehnamadi R, A Batch Mode Scheduling Algorithm for Grid Computing, Journal of Basic and Applied Scientific Research, pp174-176, 2013.

[8] Braun T.D., Siegel H.J., Beck N., 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, Vol. 61, pp.810-837.

[9] Ibarra O.H., Kim C.E., Apr 1977, "Heuristic algorithms for scheduling independent tasks on non identical processors", Journal of Association for Computing Machinery, Vol.24, Issue.2, , pp280-289.

[10] T. Kokilavani and Dr. D.I. George Amalarethinam, "Load Balanced Min-Min Algorithm for Static Meta-Task Scheduling in Grid Computing", International Journal of Computer Applications (0975 – 8887)Volume 20– No.2, April, 2011.