MSSA: A M-Level Sufferage-based Scheduling Algorithm in Grid Environment

Sanjaya Kumar Panda and Pabitra Mohan Khilar

Department of Computer Science and Engineering
National Institute of Technology, Rourkela, India
sanjayauce@gmail.com, pmkhilar@nitrkl.ac.in

Abstract. Scheduling is an emergent area in Grid Environment. It is essential to utilize the processors efficiently and minimize the schedule length. In Grid Environment, tasks are dependent on each other. We use Directed Acyclic Graph (DAG) to solve task scheduling problems. In this paper, we have proposed a new scheduling algorithm called M-Level Sufferage-based Scheduling Algorithm (MSSA) for minimizing the schedule length. It has two-phase process: m-level and sufferage value. M-level is used to calculate the earliest time. Sufferage is used to assign priority and select an optimal machine. MSSA always gives optimal or sub-optimal solution. Our result shows better results than other scheduling algorithms such as MET, MCT, Min-Min and Max-Min with respect to scheduling length and resource utilization.

Keywords: Scheduling algorithm, Grid environment, M-level, Sufferage, Directed acyclic graph, Makespan.

1 Introduction

Grid computing is a loosely coupled system and an innovative way to solve complex problems. In loosely coupled system, the inter-processor communication delay is large due to lack of coordination between systems. Each system in grid has different computation power, operating systems, peripherals and many more [4] [9]. It enables sharing in computational grid environment. Grid computing creates a structure to use the underutilized systems. A complex computation job can be divided into number of small partitions and it can be executed parallel in grid environment. So, we need a Grid Resource Broker (GRB) which divides the job into number of tasks [5]. GRB allocates task for a processor.

Scheduling is the way for allocation of the tasks. It depends on the criteria or requirements of tasks. The aim of scheduling are reducing Makespan (or scheduling length) and efficient utilization of the processors [10]. Scheduling is a NP-complete problem [6] [7]. Tasks are two types: dependent and independent task. Independent task can be scheduled in any order. Some of the algorithms are Min-Min, Max-Min, Minimum Completion Time (MCT), and Minimum Execution Time (MET). But, dependent task cannot be scheduled in any order. The dependency among tasks must

be preserved. In order to represent it, task graph or Directed Acyclic Graph (DAG) is used. Some of the algorithms are Priority-based Task Scheduling (P-TSA) [3], Highest Level First with Estimated Times (HLFET) [12].

Parallelism in our approach is of two types: computation and communication. If a task is assigned to parent task processor then communication time is zero. Two tasks can share communication parallelism but cannot share computation parallelism. Let us assume that a task has two parents. The task has to wait until both parents complete its execution and communication between the parents to the given task is over. It is called as Strong dependency. If the task starts its execution partially before the parent task has finished, then it is called as Weak dependency [11].

Distributed system contains a huge number of workstations. They are connected through high speed buses. Nodes and interconnection are different from one environment to another environment [2]. For solving large scale complex problem, it is not possible to enhance the capability of a single computer. It is not only costly but also bulkier. But, in distributed system, the complex problem is divided to small chunks and it can be executed more efficiently. Idleness of CPU is reduced to greater extent.

The remaining part of this paper is organized as follows: related work is devoted in section 2. Section 3 elaborates preliminaries such as notations, assumptions, traditional scheduling algorithms. Section 4 proposes the M-Level Sufferage-based Scheduling Algorithm (MSSA). Section 5 discusses performance analysis with a suitable illustration. We conclude by summarizing the work in Section 6.

2 Related Works

There are numerous algorithms in the area of grid scheduling. Algorithm gives sub-optimal or optimal solution based on the criteria. We can say the algorithms are approximate solutions. If an algorithm gives optimum result for particular types of data set then it is a sub-optimum solution. Hemamalini compares different task scheduling algorithms in heterogeneous environment [5]. Bozdag et al. proposes a generic algorithm which preserves the Makespan by integrating processor schedules [8]. Sun et al. introduces a priority scheduling algorithm [3]. It calculates the task priority. Based on the priority, it groups the task into teams.

Scheduling may be centralized or decentralized. Pop et al. proposes a decentralized task scheduling using genetic algorithm [9]. Also, Navimipour et al. introduces a linear genetic representation for computational grid tasks. It uses different crossover operations. Genetic algorithm may not provide optimal solution [6]. But, it provides approximate solution. Priority among tasks and allocation of machine are two important steps of list scheduling. Hagras et al. introduces an approach for machine allocation which can be applied to list scheduling [13]. It can be applied to both insertion and non-insertion approach. If the task is scheduled without looking the hole, it is termed as non-insertion approach. But in insertion approach,

task is filled to the first available hole [14]. Of course, insertion is a good approach then non-insertion one because idle time is reduced in some extent.

3 Preliminaries

3.1 Notations

MSSA: M-Level Sufferage-based Scheduling Algorithm
DAG: Directed Acyclic Graph
C_T: Computation Time
C_t: Communication Time
R_T: Ready Time
C_t (T_i, T_j): Communication Time between task T_i and task T_j
SV: Sufferage Value
M-Level (T_i): M-Level of task T_i
P-TSA: Priority-based Task Scheduling
MET: Minimum Execution Time
MCT: Minimum Completion Time
FCFS: First Come First Served

3.2 The DAG Model

Task may be dependent or independent in real time environment. In banking system (or airline control system), each transaction has a sequential process. So, we cannot start in a random order. It is necessary to represent the task in a graph. Generally, DAG is used to represent the dependency among tasks. From DAG, we can schedule the task in an order.

Let us consider a DAG D = (T, E), where T represents the number of tasks and E represents the number of edges. Each task requires some time to execute a set of instructions. The time is called Computational time. Two tasks are using a directed edge. This time is called Communicational time. In DAG, edges are unidirectional. So, it creates scheduling holes [14]. The task does not have parent task is known as Entry task. Similarly, the task does not have children task is known as Exit task. C_{Ts} of tasks are different in each processor because we are considering heterogeneous environment. It is very difficult to choose an optimal machine in grid environment. Finally, the problem statement is to find an optimal schedule in a heterogeneous environment.

3.3 Assumptions

Let us take a heterogeneous environment for results analysis. It contains systems with different architecture and different requirements. For our approach, C_T and C_t are

provided before scheduling takes place. But, task can be added in between scheduling. C_t is ignored if and only if the task is assigned to parent processors. It is assumed that communication may overlap with each other.

3.4 Scheduling Algorithms

To get a close to optimal schedule, many traditional algorithms are developed. The algorithms are listed below.

3.4.1 P-TSA [3]

It computes task priority. For task priority, it considers depth of the task, estimated complete time, up link cost and down link cost. If two nodes are in the same depth, then they are in the same group. We calculate the priority value of each group tasks. Then, we will get a scheduling order.

3.4.2 MET

It focuses on minimum execution time. It schedules the tasks in FCFS sequence. Load balancing is not considered in this approach.

3.4.3 MCT

It focuses on minimum completion time. Like, it schedules the tasks in FCFS sequence. Completion time is the sum of execution time and ready time. Load imbalance is the demerit of this approach.

3.4.4 Min-Min

It selects small task before large task. It combines MET and MCT. First Min shows MET whereas second Min shows MCT. It works efficiently if and only if there are so many small tasks present in Grid. Otherwise, it will cause starvation to large tasks.

3.4.5 Max-Min

It is similar to Min-Min. But, it selects large task before small one. It is better than Min-Min algorithm. It causes starvation to small tasks.

4 Proposed Algorithm

4.1 Description

Our proposed algorithm MSSA contains two phases. In first phase, scheduler calculates m-level of each ready task. M-level is the sum of C_T, ready time and C_t between parent to given task. In second phase, scheduler calculates SV. It is the

difference between two best or optimal machine. According to SV, tasks are sorted in descending order. The task having high SV will be given highest priority. We continue the above process until the ready list is empty.

4.2 Algorithm

1. Initially, ready list contains the entry task.
2. Assign the entry task to the processor.
3. Set C_T of entry task as R_T.
4. **Repeat**
   Calculate m-level for ready list tasks on each processor.
   M-Level $(T_i) = C_T + R_T + C_t (T_f, T_i)$
5. Calculate SV.
6. Sort the tasks in descending order of SV.
7. Select the task-processor pair which gives the earliest m-level time. Ties are broken randomly.
8. Assign the task to the respective processor.
9. Remove the task from ready list.
10. Update the R_T of each processor.
11. **if** (ready list is not empty)
12. Go to Step 7.
13. **else**
14. Repeat.

Until the ready list do not have any task.

5 Performance Analysis

5.1 Illustration

Let us consider a complex DAG (Figure 1) having seventeen tasks and three processors. Each node (or ellipse) in the DAG represents a task. The time required to execute the task is called C_T. It is shown in Table 1. We are considering a heterogeneous grid environment. So, the execution time is different in each processor. Connection between two tasks is called C_t. It is represented in rectangular box. Communication between task T_i and task T_j is represented as $C_t (T_i, T_j)$. A task can only execute if and only if its entire parent has been completed.

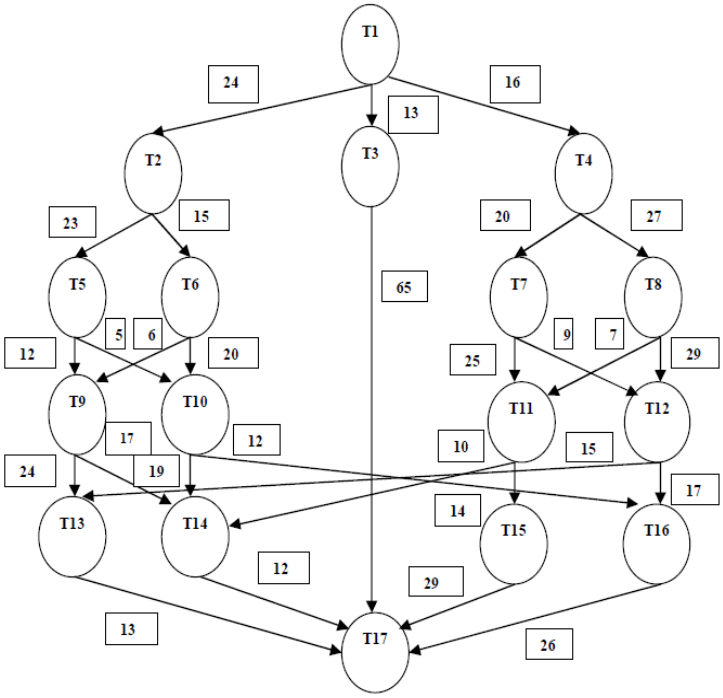

Fig. 1. A Complex DAG.

Table 1. C_T of tasks

Task	P1	P2	P3
1	41	46	34
2	46	40	41
3	26	48	35
4	46	33	16
5	32	11	48
6	40	43	41
7	14	27	22
8	27	34	19
9	48	38	39
10	49	37	40
11	28	20	29
12	49	33	24

13	48	28	22
14	24	36	32
15	40	32	36
16	17	14	38
17	21	16	14

Here, T_1 is the start (or entry) task. So, ready list contains only one task T_1. Now, we assign the task to a processor which takes less time. For T_1, it is processor 3. All other processors are idle at the same time because T_1 is the parent of next level tasks. The C_T of T_1 is 34. Ready time (R_T) is initialized as 34.

After T_1 has successfully processed, the next level tasks are T_2, T_3 and T4. In order to assign the tasks to the processors, we have designed a two-phase process. At first phase, we calculate m-level of the tasks. M-Level of the task is the sum of C_T, ready time and C_t between parent tasks to the given task. The m-level of tasks are shown in Table II. In second phase, we calculate SV of the tasks [1]. SV is the difference between the m-level of two best processors. It is shown in Table III.

Table 2. M-Level of tasks

Task	Processor 1			Processor 2			Processor 3		
	C_T	R_T+C_t	M-level	C_T	R_T+C_t	M-level	C_T	R_T+C_t	M-level
2	46	58	104	40	58	98	41	34	75
3	26	47	73	48	47	95	35	34	69
4	46	50	96	33	50	83	16	34	50

Table 3. SV of tasks

	Processor 1	Processor 2	Processor 3	
Task	M-level	M-level	M-level	SV
2	104	98	75	23
3	73	95	69	4
4	96	83	50	33

Now, we sort the tasks in descending order of SV. Ready list is updated in the following sequence: T_4, T_2 and T_3. T_4 is assigned to processor 3. The R_T of processor 3 is modified to 50. The updated m-level is shown in Table IV. We need not calculate SV again.

Table 4. Updated SV of tasks

	Processor 1	Processor 2	Processor 3
Task	M-level	M-level	M-level
2	104	98	91
3	73	95	85

Again, T_2 is assigned to processor 3. The R_T of processor 3 is altered to 91. The updated m-level is shown in Table V.

Table 5. Updated SV of tasks

	Processor 1	Processor 2	Processor 3
Task	M-level	M-level	M-level
3	73	95	126

Finally, T_3 is assigned to processor 1. After this, ready list is empty. So, we repeat the algorithm again and again until there is no task in the ready list. Final Gantt chart is shown in Figure 2. Dotted mark represents idle time and star mark represents communication delay time.

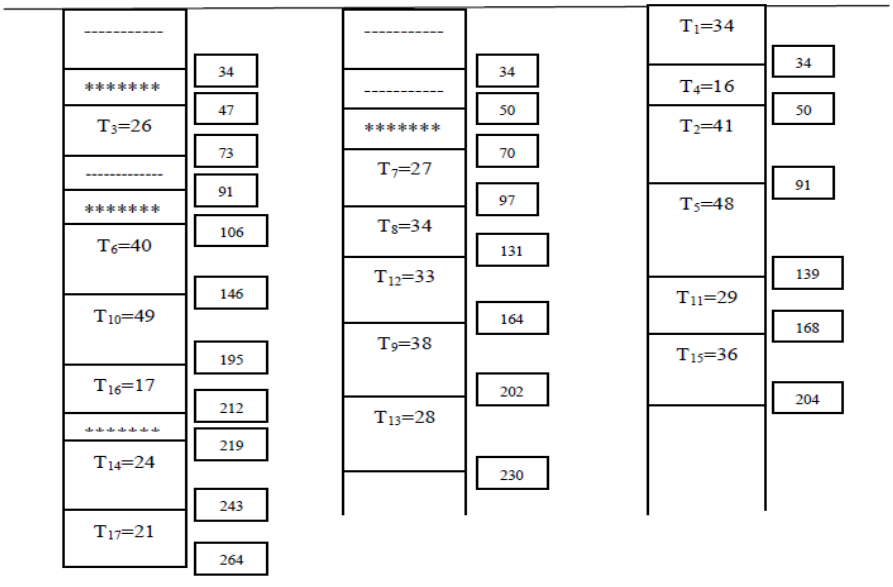

Fig. 2. Gantt chart

5.2 Results

The result shows that our proposed algorithm (MSSA) is better and efficient than the traditional scheduling algorithms. We have considered two performance measures to compare our proposed algorithm with other task scheduling algorithms. First, Makespan is used to calculate the scheduling length. It is the total time required to complete the task execution. Second, Resource utilization is the percentage of time a particular resource (or processor) is busy. We need maximum resource utilization as well as minimum scheduling length. We have taken two cases to evaluate the performance measure in all task scheduling algorithms. Performance metrics are shown in figure 3 (case 1), figure 4 (case 1), figure 5 (case 2) and figure 6 (case 2) respectively. Number of machines and tasks are varying in one case to another case. Each case has a complex DAG. In each case, we compare our results with the traditional algorithms results. X-axis denotes the number of machines and Y-axis denotes the Makespan (in figure 3 and 5), average resource utilization (in figure 4 and 6). If number of processor is restricted to one then the performance for all algorithms remains same.

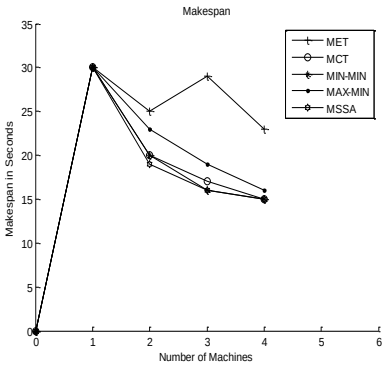

Fig. 3. Case 1 Makespan results

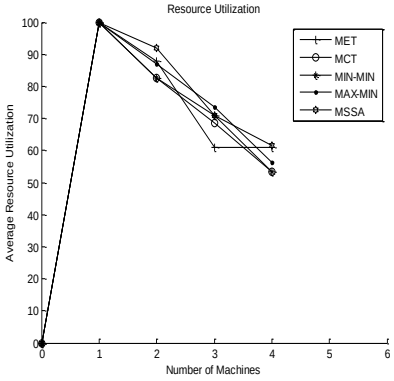

Fig. 4. Case 1 resource utilization results

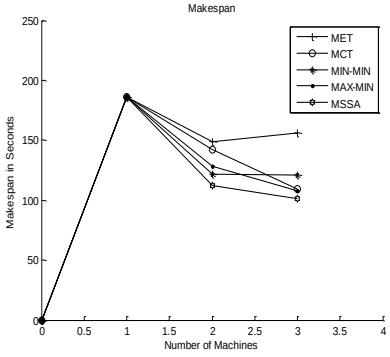

Fig. 5. Case 2 Makespan results

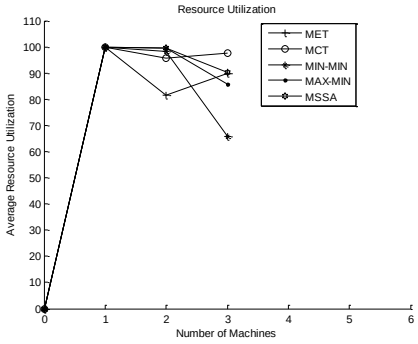

Fig. 6. Case 2 resource utilization results

6 Conclusion

MSSA scheduling introduces a new concept called m-level in which earliest time is calculated. By considering m-level value, SV is calculated. It provides priority among tasks. This scheduling has an improvement over all traditional algorithms in grid environment. Our result shows this. Makespan is reduced in greater extent. Resources are utilized efficiently in our algorithm. This proposed scheduling will help to faster execution in grid environment. Finally, it gives sub optimal or optimal solution.

In future work, we can further extend MSSA algorithm by using security, time constraints to the tasks and duplication of tasks. Communication between tasks and resources may be failed due to network problems, energy limitations and many more. It can be one of the best future works. It can be more realistic if all criteria are considered.

References

1. M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems", Journal of Parallel and Distributed Computing, Vol. 59, pp. 107-131, July 1999.
2. D. Janakiram, "Grid Computing", Tata McGraw-Hill Publishing Company Limited, ISBN 0-07-060096-1, 2005.
3. W. Sun, Y. Zhu, Z. Su, D. Jiao, and M. Li, "A Priority-based Task Scheduling Algorithm in Grid", IEEE Third International Symposium on Parallel Architectures, Algorithms and Programming, pp. 311-315, 2010.
4. R. Buyya, "High Performance Cluster Computing", Pearson Education, ISBN 81-317-1693-7, 2008.
5. M. Hemamalini, "Review on Grid Task Scheduling in Distributed Heterogeneous Environment", International Journal of Computer Applications, Vol. 40, No. 2, pp. 24-30, Feb 2012.
6. N. J. Navimipour, and L. M. Khanli, "The LGR Method for Task Scheduling in Computational Grid", IEEE International Conference on Advanced Computer Theory and Engineering, pp. 1062-1066, 2008.
7. Y. Zhang, C. Koelbel, and K. Kennedy, "Relative Performance of Scheduling Algorithms in Grid Environments", Seventh IEEE International Symposium on Cluster Computing and the Grid, 2007.
8. D. Bozdag, F. Ozguner, and U. V. Catalyurek, "Compaction of Schedules and a Two-Stage Approach for Duplication-Based DAG Scheduling", IEEE Transactions on Parallel and Distributed Systems, Vol. 20, No. 6, pp. 857-871, Jun 2009.
9. F. Pop, C. Dobre, and V. Cristea, "Genetic Algorithm for DAG Scheduling in Grid Environments", IEEE, pp. 299-305, 2009.
10. W. Yangyang, and Y. Hongfang, "Considering the Utilization of Idle Time Slots for DAG Scheduling in Optical Grid Applications", International Conference on Information, Networking and Automation, IEEE, pp. 303-307, 2010.
11. N. Ma, Y. Xia, and V. K. Prasanna, "Exploring Weak Dependencies in DAG Scheduling", IEEE International Parallel & Distributed Processing Symposium, pp. 591-598, 2011.
12. Y. K. Kwok, and I. Ahmed, "Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors", ACM, Vol. 31, No. 4, pp. 406-471, Dec 1999.
13. T. Hagras, and J. Janecek, "A Machine Assignment Mechanism For Compile Time List Scheduling Heuristics", Computing and Informatics, Vol. 24, pp. 341-350, May 2005.

14. B. Simion, C. Leordeanu, F. Pop, and V. Cristea, “A Hybrid Algorithm for Scheduling Workflow Applications in Grid Environments (ICPDP)”, Springer, pp. 1331-1348, 2007.