ISSN NO. 2320-5407

Journal homepage: http://www.journalijar.com

INTERNATIONAL JOURNAL OF ADVANCED RESEARCH

RESEARCH ARTICLE

An Improved Sufferage Meta-Task Scheduling Algorithm in Grid Computing Systems

Naglaa M. Reda*
Department of Mathematics, Faculty of Science, Ain Shams University, Cairo, Egypt

Manuscript Info

Manuscript History:

Received: 19 August 2015
Final Accepted: 29 September 2015
Published Online: October 2015

Key words:

Grid computing, Heuristic algorithm, Task scheduling, Resource utilization, Makespan, Flow time.

Corresponding Author

Naglaa M. Reda

Abstract

Scheduling tasks on heterogeneous resources distributed over a grid computing system is an NP-complete problem. Many scheduling algorithms have been developed aiming at reaching optimality. The Sufferage algorithm has shown a superlative performance over most meta-task scheduling algorithms regarding resources selection. However, providing a full power use of resources is still a challenge. In this paper, an improved heuristic algorithm of Sufferage is proposed. Its goal is to maximizing the resource utilization and minimizing the makespan. We adapt a new strategy for selecting proper resources. The main two criteria are the sufferage value and the minimum completion time. Our experimental results show that the proposed algorithm outperforms other algorithms in terms of flow time, utilization, and makespan.

INTRODUCTION

Grid computing system [1, 2] is a distributed system that enables large-scale resource sharing among millions of computer systems across a worldwide network such as the Internet. Grid resources are different from resources in conventional distributed computing systems by their dynamism, heterogeneity, and geographic distribution. The organization of the grid infrastructure involves four levels which are in ordering as follows. The foundation level includes the physical components. The middleware level is actually the software responsible for resource management, task execution, task scheduling, and security. The services level provides vendors/users with efficient services. The application level contains the services such as operational utilities and business solutions/tools.

Task scheduling [3] is the main step of grid resource management. It manages tasks to allocate resources by using scheduling algorithms and polices. In static scheduling, the information regarding all the resources in the grid as well as all the tasks in an application are assumed to be known in advance. Furthermore, task is assigned once to a resource. While in dynamic scheduling, the task allocation is done on the go as the application executes, where it is not possible to find the execution time. The tasks are entering dynamically and the scheduler has to work hard in decision making to allocate resources. The advantage of the dynamic over the static scheduling is that the system need not possess the run time behavior of the application before it runs. Since scheduling directly influences the performance of grid applications, it has become one of the major research objectives in grid to devise new methods for improving computational efficiency.

The main contribution of this work is to introduce an efficient heuristic for scheduling tasks to resources on computational grids with maximum utilization and minimum makespane. The proposed algorithm (M-Safferage) depends on the sufferage value and the minimum completion time. Our algorithm overcomes the weakness point of Sufferage when there is more than one task having the same maximum sufferage value. Constrains to map the most appropriate task that increases the grid efficiency are formalized. Performance tests of the proposed algorithm show a good improvement over the original Sufferage algorithm and others.

I. RELATED WORK

Since the late nineties, several heuristic algorithms for grid task scheduling (GTS) [4-6] have been developed to improve grid performance. They are classified into meta-task algorithms where all tasks can run independently and DAG algorithms where a DAG represents the partial ordering dependence relation between tasks execution. In the following, most popular meta-task GTS algorithms are briefly introduced.

Opportunistic Load Balancing (OLB) algorithm [7] assigns each task in arbitrary order to the next available resource regardless of its expected execution time. If two or more resources are idle, then a resource is selected arbitrarily. This strategy takes small time for scheduling and keeps almost all the resources busy as possible all the time. Thus, it gives rich resources utilization because it takes into account the load balancing. Its disadvantage is that it has the worst case makespan because it doesn't consider the task expected execution time.

Minimum Execution Time (MET) algorithm [8] assigns each task in arbitrary order to the resource with the minimum execution time MET (that is the least time exhausted by the resource during task execution) without considering resource availability. This method causes load imbalance that weaken grid resources utilization.

Minimum Completion Time (MCT) algorithm [8] assigns each task in arbitrary order to the resource with the earliest completion time MCT (that is the smallest sum of the execution times on the selected resources and the ready time). It combines the benefits of OLB and MET, while avoiding the circumstances in which they perform poorly. Its weakness point is that some tasks may be assigned to resources that don't have minimum execution time for them which poorly consume some resources.

Min-Min algorithm [9] starts with computing the completion time for each task on each resource. Then, it selects the resource with minimum expected completion time and assigns task with the minimum completion time to it. After deleting the assigned task from the set of tasks and updating the completion time for unassigned tasks, it repeats the same procedure until the unassigned task list get exhausted. This algorithm produces a smaller makespan when most tasks have small execution time, since it maps them earliest and executes them faster. Nevertheless, if there are a lot of small tasks, it will act badly resulting low utilization.

Max-Min algorithm [9] likewise computes the completion time for each task on each resource at first. After that, the resource with minimum expected completion time is selected and the task with the maximum completion time is assigned to it. This procedure is repeated until no unassigned task remains. This algorithm acts better than Min-min when the number of tasks having little time is much more than the long ones and vice versa.

Switching Algorithm (SA) [9] is a combination of MCT and MET. It tries to overcome some limitations of both methods by combining their best features. The MET algorithm can potentially create load imbalance across resources by assigning many more tasks to some resources than to others, whereas the MCT heuristic tries to balance the load by assigning tasks for earliest completion time. Essentially, the idea is to use MET till a threshold is reached and then use MCT to achieve a good load balancing. This is done by combining MET and MCT cyclically based on the workload of resources.

Sufferage algorithm [9] maps a resource to a task that would suffer most in terms of expected completion time according to its sufferage value. It first computes the completion time for each task on each resource. Second, the two consecutive minimum completion time for each task are found. The difference between these two values is defined as the suffrage value. Third, the task with maximum suffrage value is assigned to a resource with minimum completion time. Then the completion times for resources are updated and the above steps are repeated until the set of tasks becomes empty. This strategy works perfectly, but it has one shortcoming in case of more than one task has the same maximum suffrage value. It simply selects the first task without taking into account the other tasks which may cause a starvation problem.

Switcher Algorithm [10] switches between the Max-Min and Min-Min algorithms and selects the best between them while making each scheduling decision. The decision of applying which algorithm depends on the basis of Standard Deviation (SD) of minimum completion time of unassigned tasks. A position in the list of unassigned tasks where the difference in completion time between the two successive tasks is more than the value of SD is searched. If it lies in first half of the list, then Min-Min algorithm is selected as the number of longer tasks is more, otherwise Max-Min is chosen. If this position does not exist, then SD is compared with a threshold value. If SD is smaller than threshold value, allocation of task to a resource is implemented using Min-Min strategy. Otherwise, Max-Min is selected.

Resource Aware Scheduling Algorithm (RASA) [11] builds a matrix representing the completion time of each task on every resource at the grid. 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.

Min-mean heuristic [12] reschedules the Min-Min produced schedule by considering the mean makespan of all the resources. It starts by assigning tasks based on the Min-min algorithm. Then, it takes the mean of all resource completion time and selects the resource with the greatest value. After that, it reschedules the tasks assigned to the selected resource to the resources whose completion time is less than the mean value. This algorithm deviates in producing a better schedule than the Min-Min algorithm when the task heterogeneity increases.

Load Balanced Min-Min (LBMM) algorithm [13] has been proposed to reduce the makespan and increase the resource utilization. It posses two-phases, in the first phase the traditional Min-Min algorithm is executed and in the second phase the tasks are rescheduled to use the unutilized resources effectively. It chooses the resources with heavy load and reassigns them to the resources with slight load. It chooses the resource with high makespan in the schedule produced by Min-Min then, 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. When 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.

Mact-min [14] algorithm starts by computing the completion time for each task on each resource. Then, the task with the maximum average completion time is selected. Finally, it maps the selected task to the resource with minimum completion time. The assigned task is deleted from the set of tasks and the completion times with average completion time for all the remaining tasks are updated. This process is repeated until all tasks are mapped.

II. PROPOSED ALGORITHM

In this section, we give an efficient algorithm called M-Sufferage for scheduling a set of meta-tasks on a computational grid system. The aim of this algorithm is to maximize the utilization and minimize the makespane. Given a Grid of *m* resources $G=\{M_1, M_2, \dots, M_m\}$ whose availability time $R=\{r_1, r_2, \dots\dots, r_m\}$, and a set of *n* tasks $T=\{T_1, T_2, \dots\dots, T_n\}$, where *m*, $n \in \mathbb{N}^+$, M-Sufferage algorithm can be viewed as a map *S* from *T* onto *G* which is determined by:

$$\forall\, i \in [1 \,..\, n] \;\; \exists\; j \in [1 \,..\, m] \quad \text{s.t.} \quad S(T_i) = M_j$$

Scheduling decision is based on the suffrage value of each task T_i in *T*. A suffrage vector *SF* is computed for all tasks by positive subtracting the two consecutive minimum completion time $\mathrm{MCT1}_i$ and $\mathrm{MCT2}_i$ for each task. When there exists only one task having the maximum suffrage value, then *S* maps this task to the resource that obtain the minimum completion time. Otherwise, all tasks having the same maximum suffrage values are gathered in a set named by T_{max} . Then, the average completion time for all tasks in T_{max} is computed in terms of the estimated completion time ETC.

Finally, *S* maps the task having the maximum average to the resource that obtain the minimum completion time. These steps are repeated until all *n* tasks are distributed on *m* resources. In the following, the steps of suggested algorithm are outlined. Assuming *A* and *B* are two sets of indices of unassigned tasks and indices of resources, respectively; initialized by $A = \{1, 2, \dots, n\}$ and $B=\{1, 2, \dots, m\}$.

Algorithm: M-Sufferage

Input: Grid *G*, Meta-tasks *T*, Resources availability *R*, and Estimated computation time matrix *ETC*

Output: The scheduling mapping *S*

1: $A \leftarrow \{1, 2, \dots, n\}$; $B \leftarrow \{1, 2, \dots, m\}$
2: **while** $A \neq \phi$ **do**
3: **for** $i \in A$ **do**

4: **for** $j \in B$ **do**
5: Calculate the completion time $c_{ij} \leftarrow e_{ij} + r_j$
6: **end for** *j*
7: Find *MCT1* and *MCT2*
8: Calculate $SF_i \leftarrow MCT2 - MCT1$
9: **end for** *i*
10: Find the maximum suffrage value *SF* from { $SF_i : i \in A$ }
11: Collect indices *i* of all tasks having $SF_i = SF$ in T_{MAX}
12: **if** T_{MAX} has at least two elements **then**
13: **for** $x \in T_{MAX}$ **do**
14: Calculate the average value, $AV_x \leftarrow {\sum_{j=1}^{m} c_{xj}}/{m}$, of T_x completion time
15: **end for** *x*
16: Get $u \in T_{MAX}$ s.t. T_u has the maximum average value AV_x
17: **else**
18: Set $u \leftarrow x \in T_{MAX}$
19: **end if**
20: Find the resource M_v, $v \in B$, that posses the minimum completion time
21: Set $S(t_u) \leftarrow m_v$
22: Delete index *u* from the set *A*
23: Update the wait time r_v of M_v by $r_v = r_v + c_{uv}$
24: **end while**

III. PERFORMANCE ANALYSIS

There are several performance metrics to evaluate the quality of a scheduling algorithm [3]. In this section, the proposed algorithm is tested according to these criteria. As a case study, it considers a grid system with three resources and four tasks whose ETC matrix is given in table (1). It presents in the following a comparison of most recent and efficient algorithms against M-Sufferage in regard to each criterion for emphasizing its strength.

	m1	**m2**	**m3**
T1	**27.5**	**10**	**7.5**
T2	**10**	**30**	**7.5**
T3	**10**	**25**	**7.5**
T4	**10**	**20**	**7.5**

Table (1): The ETC Matrix

Computational complexity

The complexity is an essential metric in theoretical analysis of algorithms that asymptotically estimate their performance. It determines the required time to solve the given computational problem using selected mathematical notation such as the Big O notation. In our case, it indicates that the scheduling algorithm is approximately like others in finding a feasible solution in a highly dynamic heterogeneous grid system. Table (2) presents the complexity of M-Sufferage algorithm and other important ones when scheduling *N* tasks *T* on a heterogeneous grid system *G* of *M* resources. It is clear from steps 2, 3, and 4 of M-Sufferage algorithm that its complexity is N^2M. Evidently, M-Sufferage has the same complexity as Sufferage and most promising algorithms.

Algorithm	MET	OLB	Mact-min	Max-min	Min-mean	Min-min	Sufferage	M-Sufferage
Complexity	O(*NM*)	O(*NM*)	O(*NM*)	O(N^2M)	O(N^2M)	O(N^2M)	O(N^2M)	O(N^2M)

Table (2): Complexity Comparison

Flow time

One of the metrics used to measure the grid scheduling performance is the flow time. It is known as all the time needed by all resources to finish all tasks. It is computed by the following summation.

$$\text{Flowtime} = \sum_{j=1}^{m} r_j$$

The flow time comparison between the proposed M-Sufferage algorithm and other scheduling algorithms is given in Figure (1). Results show that M-Sufferage flow time is less than the Suffrage algorithm which means the waiting time to start task execution is small.

Makespan

Makespan is an important performance criterion of scheduling heuristics in grid computing systems. It is defined as the maximum completion time of application tasks executed on grid resources. Formally, it is computed by using the following equation where *C* is the computed completion time matrix:

$$makespan = max\ (cij)_{i \in [1..n],\ j \in [1..m]}$$

Figure (2) shows the Makespan Comparison between M-Sufferage and other algorithms for the above case study. It is obvious that M-Sufferage is the best, as whenever the makespan decreases, the scheduler becomes better.

Resource Utilization

Resource utilization is the most essential performance metric for grid managers. The resource's utilization (*MU*) is defined as the amount of time a resource is busy in executing tasks. And the grid's resource utilization U is the average of resources' utilization. They are computed as follows:

$$U = \frac{\sum_{j=1}^{m} MU_j}{M}$$

where

$$MU_j = \frac{r_j}{makespan} \quad ; \ j \in [1..m]$$

Figure (3) illustrates that M-Sufferage maximizes the utilization by presenting its results against other algorithms regarding the above studied case. It is clear that M-Sufferage is superlative, because more utilization leads to improving the scheduling.

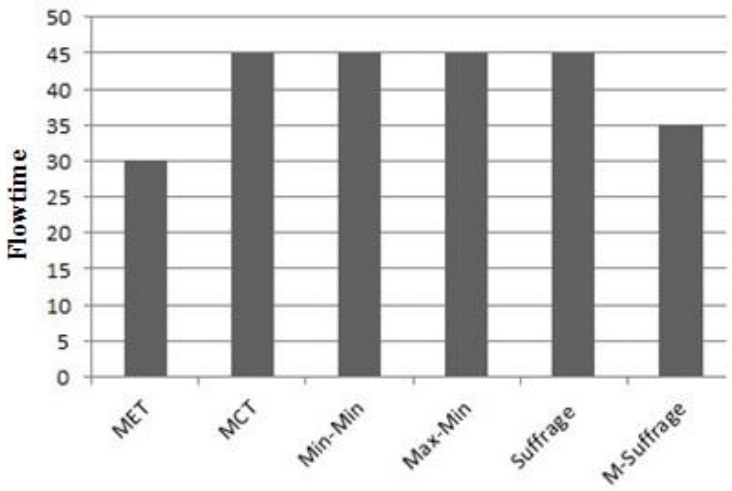

Figure (1): Comparison of Flow Time

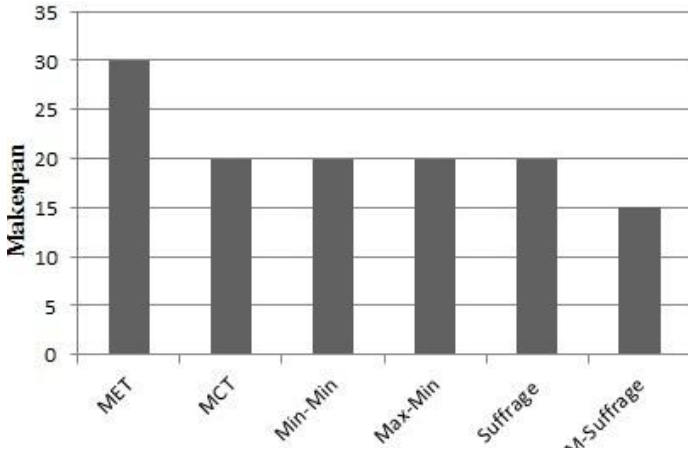

Figure (2): Comparison of Makespan

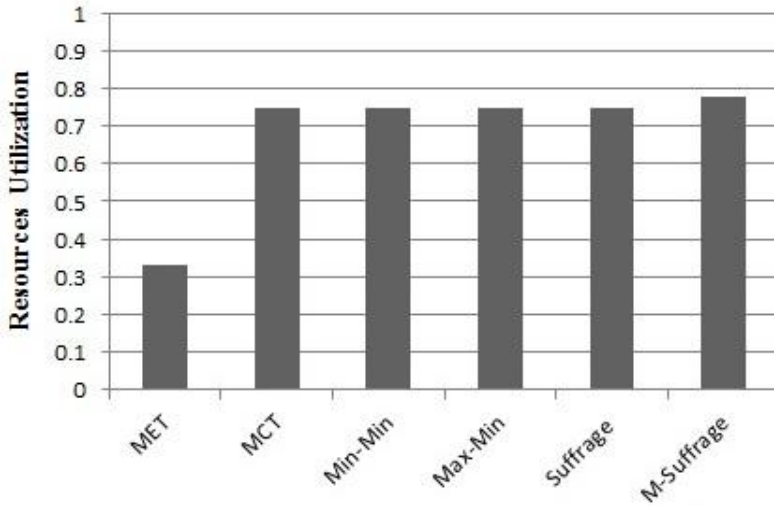

Figure (3): Comparison of Resource Utilization

IV. CONCLUSIONS AND FUTURE WORK

This paper introduces an improved meta-task scheduling algorithm called M-Sufferage. It is based on the well-known Sufferage algorithm. It is concerned with task execution time, resource utilization and makespan. It proposes new mapping *S* for scheduling tasks using the main of minimum completion time. It overcomes the shortage of Sufferage when handling more than one task having the same maximum sufferage value. Performance analysis illustrates that M-Sufferage algorithm has the same complexity as Sufferage. It also shows that M-Sufferage outperforms others with respect to flow time, minimizes the makespan, and maximizes the resource utilization.

For future work, experimental testing is planned to be performed using a simulation environment such as *GridSim* toolkit, to compare and evaluate M-Sufferage with other algorithms practically.

REFERENCES

[1] Frédéric Magoulès, Jie Pan, Kiat-An Tan, & Abhinit Kumar (2009). Introduction to grid computing. CRC Press, London, New York.

[2] George Amalarethinam, Dr.D.I., & Muthulakshmi, P. (2011). An Overview of the Scheduling Policies and Algorithms in Grid Computing. International Journal of Research and Reviews in Computer Science (IJRRCS), 2(2), 280-294.

[3] Chandak, A., Sahoo, B., & Turuk, A. (2011). An Overview of Task Scheduling and Performance Metrics in Grid Computing. Proc. of 2nd National Conference-Computing, Communication and Sensor Network, 30-33.
[4] Braun, R., Siegel, H., et al. (2001). A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing System. Journal of Parallel and Distributed Computing Systems, 61(6), 810-837.
[5] Tinghuai Ma, Qiaoqiao Yan, Wenjie Liu, Donghai Guan, & Sungyoung Lee (2011). Grid Task Scheduling: Algorithm Review. IETE Technical Review, 28(2), 158-167.
[6] Elzeki, O.M., Rashad, M.Z., & Elsoud, M.A. (2012). Overview of Scheduling Tasks in Distributed Computing Systems. International Journal of Soft Computing and Engineering, 2(2), 36-39.
[7] Maheswaran, M., Ali, S., Siegel, H.J., Hensgen, D., & Freund, R.F. (1999). Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems. Journal of Parallel and Distributed Computing, 59(2), 107-131.
[8] Freund, R.F., Gherrity, M., et al. (1998). Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet. 7th IEEE Heterogeneous Computing Workshop, 184-199.
[9] M. Maheswaran, S. Ali, H.J. Siegel, D. Hensgen, R.F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems", Journal of Parallel and Distributed Computing 59 (2) (1999) 107_131.
[10] Singh, M., & Suri, P.K. (2008). $\mathrm{QPS}_{\mathrm{Max-Min}<>Min-Min}$: A QoS Based Predictive Max Min, Min-Min Switcher Algorithm for Job Scheduling in a Grid. Information Technology Journal, 7(8), 1176-1181.
[11] Parsa, S., Entezari-Maleki, R. (2009). RASA: A New Grid Task Scheduling Algorithm. International Journal of Digital Content Technology and its Applications, 3(4), 91-99.
[12] Kamalam, G.K., & Muralibhaskaran,V. (2010). A New Heuristic Approach: Min-Mean Algorithm For Scheduling Meta-Tasks On Heterogenous Computing Systems. International Journal of Computer Science and Network Security, 10(1), 24-31.
[13] Kokilavani, T., & George Amalarethinam, Dr.D.I. (2011). Load Balanced Min-Min Algorithm for Static Meta-Task Scheduling in Grid Computing. International Journal of Computer Applications, 20(2), 43-49.
[14] Fahd Alharbi (2012). Multi Objectives heuristic Algorithm for Grid Computing. International Journal of Computer Applications, 46(18), 39-45.
[15] Soheil Anousha & Mahmoud Ahmadi (2013). An Improved Min-Min Task Scheduling Algorithm in Grid Computing. Lecture Notes in Computer Science 7861, 103–113.