ORIGINAL ARTICLE

Sort-Mid tasks scheduling algorithm in grid computing

Naglaa M. Reda, A. Tawfik, Mohamed A. Marzok*, Soheir M. Khamis

a Department of Mathematics, Faculty of Science, Ain Shams University, Cairo, Egypt
b Egypt. Ctr. for Theo. Phys., Faculty of Engineering, Modern University, Cairo, Egypt

ABSTRACT

Scheduling tasks on heterogeneous resources distributed over a grid computing system is an NP-complete problem. The main aim for several researchers is to develop variant scheduling algorithms for achieving optimality, and they have shown a good performance for tasks scheduling regarding resources selection. However, using of the full power of resources is still a challenge. In this paper, a new heuristic algorithm called Sort-Mid is proposed. It aims to maximizing the utilization and minimizing the makespan. The new strategy of Sort-Mid algorithm is to find appropriate resources. The base step is to get the average value via sorting list of completion time of each task. Then, the maximum average is obtained. Finally, the task has the maximum average is allocated to the machine that has the minimum completion time. The allocated task is deleted and then, these steps are repeated until all tasks are allocated. Experimental tests show that the proposed algorithm outperforms almost other algorithms in terms of resources utilization and makespan.

© 2014 Production and hosting by Elsevier B.V. on behalf of Cairo University.

Introduction

Grid computing systems [1,2] are distributed systems, enable 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. First: the foundation level, it includes the physical components. Second: the middleware level, it is actually the software responsible for resource management, task execution, task scheduling, and security. Third: the services level, it provides vendors/users with efficient services. Fourth: the application level, it contains the services such as operational utilities and business tools.

The scheduling has become one of the major research objectives, since it directly influences the performance of grid applications. Task scheduling [3] is the main step of grid resource management. It manages jobs to allocate appropriate resources by using scheduling algorithms and polices. In static scheduling, the information regarding all the resources as well as all the tasks is assumed to be known in advance, by the time the application is scheduled. Furthermore, each 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. 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 does not need to pose the run time behavior of the application before it runs.
Since the late nineties, several heuristic algorithms for grid task scheduling (GTS) \cite{4-6} have been developed to improve grid performance. They are classified into task algorithms in which all tasks can be run independently and DAG algorithms, where a DAG represents the partial ordering dependence relation between tasks execution.

The main contribution of this work is to introduce an efficient heuristic algorithm for scheduling tasks to resources on computational grids with maximum utilization and minimum makespan. The proposed algorithm (Sort-Mid) depends on the minimum completion time and the average value AV of completion times for each task. It puts constrains to map the most appropriate task to the best convenient resource, which increases the grid efficiency. Performance tests show a good improvement over existing popular scheduling algorithms.

The most popular task GTS algorithms are surveyed in the following subsection. The rest of this paper is organized as follows. The proposed methodology with the suggested algorithm for the scheduling problem in grid computing system is introduced. Then, the used experimental materials followed by results and discussion are presented. Finally, the conclusion of the overall work is given.

Related work

Opportunistic load balancing (OLB) algorithm \cite{7} assigns each task in arbitrary order to the next available machine regardless of the task’s expected execution time on the machine, while, minimum execution time (MET) algorithm \cite{8} assigns each task in arbitrary order to the machine with the minimum execution time without considering resource availability. But, minimum completion time (MCT) algorithm \cite{8} assigns each task in arbitrary order to the machine with the earliest completion time. On the other hand, Min-Min algorithm \cite{9,10} selects the machine with minimum expected completion time and assigns task with the MCT to it. Where, Max-Min algorithm \cite{9,10} selects the machine with minimum expected completion time and the task with the maximum completion time is mapped to it. And, switching algorithm (SA) \cite{9} combines MCT and MET to overcome some limitations of both methods.

Furthermore, Suffrage heuristic \cite{9} maps the machine to the task that would suffer most in terms of expected completion time according to an evaluated suffrage value. Switcher heuristic \cite{11} switches between the Max-Min and Min-Min algorithms by taking a scheduling decision based on the standard deviation of minimum completion time of unassigned jobs. RASA heuristic \cite{12} has built a matrix representing the completion time of each task on every resource, and applies Min-Min if the number of available resources is odd, otherwise it applies Max-Min. Min-mean heuristic \cite{13} reschedules the Min-Min produced schedule by considering the mean makespan of all the resources. Load balanced Min-Min (LBMM) heuristic \cite{14} has reduced the makespan and has increased the resource utilization by choosing the resources with heavy load and reassigns them to the resources with light load. Mact-mini heuristic \cite{15} maps the task with the maximum average completion time to the machine with minimum completion time. Recently, a new heuristic algorithm based on Min-Min has been presented \cite{16}. It selected resources according to a new makespan value and the maximum value of possibilities tasks (MVPT).

Methodology

In this section, we present a new idea for solving the scheduling problem in a grid. Scheduling is the main step of grid machines management \cite{17}. Machines may be homogeneous or heterogeneous. A grid scheduler selects the best machine to a particular job and submits that job to the selected machine \cite{18}. The main aim of suggested heuristic algorithm for scheduling a set of tasks on a computational grid system is to maximize the machines utilization and to minimize the makespan. Given a grid \( G \) with a finite number, \( m \), of machines (resources); \( M_1, M_2, \ldots, M_m \), \( m > 1 \). Let \( T \) be a finite nonempty set of \( n \) tasks; \( T_1, T_2, \ldots, T_n \), \( n > 1 \) that needs to be executed in \( G \).

In the following work, the proposed algorithm called Sort-Mid is given. It’s steps to assign each task to a suitable machine are summarized below. It uses assignment function \( S: T \rightarrow G \) which is defined as follows. For every positive integer \( i \leq n \) s.t. \( S(T_i) = M_j \). The first step is to sort the completion times (SCT) of each task \( T_i \) in increasing order. The introduced scheduling decision is based on computing the average value AV of two consecutive completion times in \( SCT \) for each \( T_i \). \( AV \) is computed by \( (SCT_k + SCT_{k-1})/2 \), where \( k = \lfloor m/2 \rfloor \). In the second step, the task having the maximum AV is selected. In the third step, the task is assigned to the machine possessing minimum completion time. Next, the assigned task is deleted from \( T \). Finally, the waiting time for the machine that executes this task is updated. These steps are repeated until all \( n \) tasks are scheduled on \( m \) machines.

The pseudo code of the algorithm is as listed below.

\begin{algorithm}[H]
\caption{Sort-Mid:}
\begin{algorithmic}
\State Input: Number of tasks \( n \), Number of machines \( m \), Grid \( G = \{M_1, M_2, \ldots, M_m\} \), Tasks \( T = \{T_1, T_2, \ldots, T_n\} \), Machines availability \( R \), Estimated time of computation ET.
\State Output: The result of the assignment function \( S(S(T_i), S(T_2), \ldots, S(T_n)) \).
\State Begin
\State Initialization: \( A \leftarrow \{1, 2, \ldots, n\} \), \( K \leftarrow \lfloor m/2 \rfloor \), \( CT \leftarrow ETC \);
\State 1. While \( A \neq \emptyset \) do
\State 2. \textbf{If} \( |A| \neq 1 \) \textbf{Then}
\State 3. \( \text{Max\_value} \leftarrow 0 \), \( \text{Index\_machine} \leftarrow 0 \), \( \text{Index\_task} \leftarrow 0 \);
\State 4. For all \( i \in A \) do
\State 5. \( SCT \leftarrow \text{sort } CT[i] \) in ascending order;
\State 6. \( AV \leftarrow (SCT_k + SCT_{k-1})/2 \);
\State 7. \textbf{If} \( AV > \text{Max\_value} \) \textbf{Then}
\State 8. \( \text{Max\_value} \leftarrow AV \);
\State 9. \( \text{Index\_task} \leftarrow i \);
\State 10. \( \text{Index\_machine} \leftarrow \text{index of machine whose completion time equals } SCT; \)
\State 11. \textbf{End If}
\State 12. \textbf{End For}
\State 13. \( S(T_{\text{Index\_task}}) \leftarrow M_{\text{Index\_machine}} \);
\State 14. \( A \leftarrow A - \{\text{Index\_task}\}; \)
\State 15. \( R_{\text{Index\_machine}} \leftarrow R_{\text{Index\_machine}} + ETC_{\text{Index\_task}} \);
\State 16. \textbf{For all } \( i \in A \) do
\State 17. \( CT_{\text{Index\_machine}} \leftarrow ETC_{\text{Index\_machine}} + R_{\text{Index\_machine}} \);
\State 18. \textbf{End For}
\State 19. \textbf{Else}
\State 20. \( \text{Assign the remaining task to the machine having the minimum completion time and delete it;} \);
\State 21. \textbf{Update waiting time of machine executing it;}\)
\State 22. \textbf{End While}
\State \textbf{End}
\end{algorithmic}
\end{algorithm}
It is clear that Sort-Mid algorithm is correct, since at the end, the set of tasks indices are vanished, i.e., all tasks are assigned to appropriate machines.

In the following, we analyze the time complexity of the above given algorithm.

Lemma. The time complexity of Algorithm Sort-Mid is in \( O(n^2m \log n) \), where \( n \) and \( m \) are the numbers of tasks and machines in a grid computing system, respectively.

Proof. It is obvious that the first For-loop starting from step 4 to step 12 iterates \( n \) time. Each iteration costs at least \((m \log m)\), which is one run of step 5 to sort the elements at row number \( i \) of CT in an ascending order. Also, the second For-loop starting from step 16 to step 18 which updates the wait time, costs \( O(n) \).

And, one run to select task and delete it and update time take \( n + m + n \), respectively. Since the while-loop (Starting from step 1 to step 22) executes \( n \) time, each run of them costs of \((mn \log m + 2n + m)\). This implies that the total time complexity of the algorithm is in \( O(n^2m \log n) \).

An illustrative example

To clarify how the proposed algorithm Sort-Mid schedules tasks perfectly, consider the following example for a grid environment with three machines and three tasks. Its ETC matrix with special form is given in Fig. 1.

The initialization step initializes the CT by ETC and machines availability vector \( R \) by zeros.

At first iteration, \( \text{Max}_\text{value} = \text{Index}_\text{machine} = \text{Index}_\text{task} = 0 \) and \( A = \{1,2,3\} \), then the number of elements \(|A| = 3\), and the created SCT after sorting illustrates as follows.

\[
\text{SCT} = \begin{bmatrix} M_3 : 22 & M_2 : 23 & M_1 : 45 \\ M_1 : 22 & M_1 : 45 & M_1 : 70 \\ M_3 : 23 & M_2 : 25 & M_2 : 63 \end{bmatrix}
\]

For the first row of SCT, the average value (AV) of the first task is \( \text{AV (1)} = (\text{SCT}_2 + \text{SCT}_3)/2 = (23 + 45)/2 \). After that, the new value 34 is compared with the value of \( \text{Max}_\text{value} = 0 \), so the \( \text{Max}_\text{value} = 34 \), \( \text{Index}_\text{task} = 1 \) and \( \text{Index}_\text{machine} = 3 \).

For the second row, the average value is \( \text{AV (2)} = (\text{SCT}_2 + \text{SCT}_3)/2 = (45 + 70)/2 \). Then after comparison, \( \text{Max}_\text{value} = 57.5 \), hence \( \text{Index}_\text{task} = 2 \) and \( \text{Index}_\text{machine} = 3 \).

For the third row, the average value \( \text{AV (3)} = (\text{SCT}_2 + \text{SCT}_3)/2 = (25 + 63)/2 \). And, the values of \( \text{Max}_\text{value} \) are still maximum value, \( \text{Index}_\text{task} \) and \( \text{Index}_\text{machine} \) will not change.

At the end of the first iteration, the task having \( \text{Index}_\text{task} = 2 \) is deleted from the set \( A \), then \( A = \{1,3\} \) and \( R_1 = 0 + 22 = 22 \). And so, CT updates to the following matrix:

\[
\text{CT} = \begin{bmatrix} M_1 : 45 & M_2 : 23 & M_3 : 44 \\ M_1 : 25 & M_2 : 63 & M_1 : 45 \end{bmatrix}
\]

At the second iteration for while-loop, first put \( \text{Max}_\text{value} = \text{Index}_\text{machine} = \text{Index}_\text{task} = 0 \), \( A = \{1,3\} \) and \( |A| = 2 \). Then, SCT is arranged as follows:

\[
\begin{align*}
t_1 & : M_1 : 45 & M_2 : 23 & M_3 : 44 \\
t_2 & : M_1 : 45 & M_2 : 70 & M_3 : 22 \\
t_3 & : M_1 : 25 & M_2 : 63 & M_3 : 23
\end{align*}
\]

Fig. 1 The matrix ETC of the given.

| Algorithms | M1 | M2 | M3 | Makespan |
|------------|----|----|----|-----------|
| MET        |    | T1 | T2 | T3        | 67         |
| OLB        | T1 |    | T2 | T3        | 70         |
| MCT        | T3 | T1 | T2 | T3        | 44         |
| Max-Min    | T3 | T1 | T2 | T3        | 45         |
| Min-Min    | T3 | T1 | T2 | T3        | 44         |
| Sort-Mid   | T3 | T1 | T2 | T3        | 25         |

Table 1 A comparison between algorithms in makespan and tasks scheduling.

For the first row, the average value of the first task is \( \text{AV (1)} = (\text{SCT}_2 + \text{SCT}_3)/2 = (23 + 45)/2 \) in SCT. After comparing 44.5 with 0, then \( \text{Max}_\text{value} = 34 \), \( \text{Index}_\text{task} = 1 \) and \( \text{Index}_\text{machine} = 2 \).

For the second row, \( \text{AV (2)} = (\text{SCT}_2 + \text{SCT}_3)/2 = (45 + 63)/2 \). Then \( \text{Max}_\text{value} = 54 \), \( \text{Index}_\text{task} = 3 \) and \( \text{Index}_\text{machine} = 1 \).

At the end of second iteration, the task with index \( \text{Index}_\text{task} = 3 \) is deleted and \( A = \{1\} \), \( R_1 = 0 + 25 = 25 \), CT

\[
\text{CT} = \begin{bmatrix} M_1 : 70 & M_2 : 23 & M_3 : 44 \end{bmatrix}
\]

In the third iteration, \( A = \{1\} \) and \( |A| = 1 \), so the task with index 1 is assigned to the machine having the minimum completion time \( M_2 \), i.e., \( \text{Index}_\text{task} = 1 \) and \( \text{Index}_\text{machine} = 2 \).

Finally, at the end of third iteration, the remaining task is deleted, then \( A = \emptyset \). And \( R_2 = 0 + 23 = 23 \).

As a result of the above execution, the makespan for the above example equals \( \text{Max (22, 25 and 23)} = 25 \). The makespan produced by other previous algorithms compared to the result of Sort-Mid algorithm is shown in Table 1.

Experimental materials

For comparison of our proposed heuristic with other scheduling algorithm, various heuristic algorithms have been developed to compare with Sort-Mid algorithm. In this section, the benchmark description is given, and the ETC model used as in benchmark experiments [14–20] is specified.

In this paper, we used the benchmark model [4]. The simulation model is based on expected time to compute (ETC) matrix for 512 tasks and 16 machines. An ETC matrix is said to be consistent (C) if whenever a machine \( m_i \) executes any task \( t_j \) faster than machine \( m_k \), then machine \( m_j \) executes all tasks faster than machine \( m_k \). In contrast, inconsistent matrices (I) characterize the situation where machine \( m_i \) may be faster than machine \( m_k \) for some tasks and slower for others. Semi-consistent matrices (S) happen when some machines are consistent while others are inconsistent. Also, different ETC matrix task and machine heterogeneity are studied, each one has two cases.
high (hi) or low (lo). Thus, the twelve matrices are tested and abbreviated as shown in Table 2.

In addition, a computer program in VB language is developed for seven existing and proposed heuristic methods mentioned above. This program produces a schedule that maps tasks to available resources and calculates the objectives based on the ETC matrix supplied to it. The twelve different ETC matrices suggested by Braun et al. [4] for different scenarios mentioned in Table 2 are used as inputs to the computer program, and the results are analyzed in the following section.

Results and discussion

There are several performance metrics to evaluate the quality of a scheduling algorithm [3]. This section tests Sort-Mid algorithm mentioned in Section ‘Methodology’ according to these criteria. It considers the problem of scheduling \( n \) tasks on a heterogeneous grid system of \( m \) machines. It presents in the following a comparison of most recent and efficient algorithms against Sort-Mid in regard to each criterion for emphasizing its strength. In the following, we compare our heuristic algorithm with other scheduling algorithms via using benchmark experiments [4,19].

Computational complexity

The complexity is an essential metric in theoretical analysis of algorithms that asymptotically estimate their performance. It determines the amount of time to solve the given computational problem using selected mathematical notation such as the Big O. In our case, it indicates how fast the scheduling algorithm will be in finding a feasible solution in a highly dynamic heterogeneous grid system. Table 3 illustrated the complexity of Sort-Mid algorithm and other important ones. It is worth to remark that the number of machines in a grid \( m \) is much less than the number of tasks \( n \) and so \( \log m \). Therefore, in practical, the running time of Sort-Mid algorithm is approximately equal to the running time of Max-Min, Min-mean, Min-Min and suffrage algorithm.

Resource utilization

The grid’s resource utilization is the most essential performance metric for grid managers. The Machine’s Utilization (MU) is defined as the amount of time at which a machine is busy in executing tasks, while the grid’s resource utilization (GU) is the average of machines’ utilization. They are computed as follows:

\[
GU = \frac{\sum_{j=1}^{m} MU_j}{m}
\]

where,

\[
MU_j = \frac{r_j}{\text{makespan}}, \quad \text{for } j = 1, 2, \ldots, m.
\]

Fig. 2 and Tables 4–6 show the values of GUs for the eight mentioned algorithms. Sort-Mid gives the second maximum resource utilization for ten instances and third maximum resource utilization for two instances. The Max-Min gives the highest maximum resource utilization for all instances but the difference is very small, while the computed makespan of Sort-Mid algorithm is better than that of Max-Min in all instances.

Makespan

The 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. Note that $C$ is the matrix of the completion times after executing given tasks in grid computing system and $R$ is the vector of waiting times of $m$ machines.

$$\text{Makespan} = \max \{c_{ij} | 1 \leq i \leq n, 1 \leq j \leq m\}, \quad \text{or}$$

$$\text{Makespan} = \max \{r_j | 1 \leq j \leq m\}.$$
Conclusions

Selecting the appropriate resource for a specific task is one of the challenging work in computational grid. This work introduces a new task scheduling algorithm called Sort-Mid. The implementation of Sort-Mid algorithm and various existing algorithms are tested using a benchmark simulation model. Min-Min is the simplest and common scheduling algorithm for grid computing. But, it works poorly when the number of large tasks is less than the number of small tasks. Also, the computed makespan by Min-Min in this case is not good. The computed grid’s resources utilization by Min-Min is not good. To avoid the disadvantages of grid’s resources utilization and makespan, Sort-Mid is designed to maximize grid’s resources utilization and to minimize the makespan. This algorithm overcomes the affection of large varies of task’s execution times. A comparison of makespan values between our algorithm and other seven scheduling algorithm has been conducted. Obviously, the result of Sort-Mid is better than all algorithms in the eleven underlying instances except for Min-Min. Nevertheless, Sort-Mid is the best in case of inconsistent high task and high machine heterogeneity. On the other hand, experimental results indicate that Sort-Mid utilizes the grid by more than 99% at 6 instances and more than 98% at 4 instances.

In conclusion, the rank of the proposed Sort-Mid algorithm regarding both makespan and utilization is very good.

Conflict of Interest

The authors have declared no conflict of interest.

Compliance with Ethics Requirements

This article does not contain any studies with human or animal subjects.

References

[1] Magoulès F, Pan J, Tan K, Kumar A. Introduction to grid computing. London, New York: CRC Press; 2009.
[2] Amalarethinam, George DJ, Muthulakshmi P. An overview of the scheduling policies and algorithms in grid computing. Int J Res Rev Comput Sci 2011;2(2):280–94.
[3] Chandak A, Sahoo B, Turuk A. An overview of task scheduling and performance metrics in grid computing. Int J Comput Appl 2011;30–3.
[4] Braun TD, Siegel HJ, Beck N, Boloni LL, Maheswaran M, Reuther AL, et al. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous
distributed computing systems. J Parallel Distrib Comput 2011;61(6):810–37.

[5] Ma T, Yan Q, Liu W, Guan D, Lee S. Grid task scheduling: algorithm review. IETE Tech Rev 2011;28(2):158–67.

[6] Elzeki OM, Rashad MZ, Elsoud MA. Overview of scheduling tasks in distributed computing systems. Int J Soft Comput Eng 2012;2(3):470–5.

[7] Maheswaran M, Ali S, Siegel HJ, Hensgen D, Richard FF. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. HCW ’99, DC, USA; 1999. p. 30–44.

[8] Freund RF, Gherrity M, Ambrosius S, Campbell M, Halderman M, Hensgen D, et al. Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet. HCW ’98, Orlando, FL; 1998. p. 184–99.

[9] Maheswaran M, Ali S, Siegel HJ, Hensgen D, Richard FF. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. J Parallel Distrib Comput 1999;59(2):107–31.

[10] Anousha S, Ahmadi M. An improved Min-Min task scheduling algorithm in grid computing. LNCS 7861; 2013. p. 103–13.

[11] Singh M, Suri PK. QPS_{Max-Min} < Min-Min; a QoS based predictive Max-Min, Min-Min switcher algorithm for job scheduling in a grid. Inform Technol J 2008;7(8):1176–81.

[12] Parsa S, Entezari-Maleki R. RASA: a new grid task scheduling algorithm. Int J Digital Content Technol Appl 2009;3(4):91–9.