A Heuristic Algorithm for Workflow-Based Job Scheduling in Decentralized Distributed Systems with Heterogeneous Resources

Nasi Tantitharanukul, Juggapong Natwichai and Pruet Boonma

Abstract Decentralized distributed systems, such as grids, clouds or networks of sensors, have been widely investigated recently. An important nature of such systems is the heterogeneity of their resources; in order to archive the availability, scalability and flexibility. As a consequence, managing the systems to meet requirements is obviously a nontrivial work. The issue is even more challenging in term of job scheduling when the task dependency within each job exists. In this paper, we address such problem of job scheduling, so called workflow-based job scheduling, in the decentralized distributed systems with heterogeneous resources. As such problem is proven to be an NP-complete problem, an efficient heuristic algorithm to address this problem is proposed. The algorithm is based on an observation that the heterogeneity of the resources can affect the execution time of the scheduling. We compare the effectiveness and efficiency of the proposed algorithm with a baseline algorithm. The result shows that our algorithm is highly effective and efficient for the scheduling problem in the decentralized distributed system with heterogeneous resources environment both in terms of the solution quality and the execution time respectively.

Keywords Decentralized algorithm · Heterogeneous resources · Workflow-based job scheduling

1 Introduction

In recent years, distributed systems such as P2P and computational grid become a generic platform for high performance computing [1]. In term of management, a distributed system can be centralized or decentralized [5]. In the centralized

N. Tantitharanukul · J. Natwichai (✉) · P. Boonma
Data Engineering and Network Technology Laboratory, Department of Computer Engineering, Faculty of Engineering, Chiang Mai University, Chiang Mai 50200, Thailand
e-mail: juggapong@eng.cmu.ac.th

N. Tantitharanukul
e-mail: n.tantitharanukul@gmail.com

P. Boonma
e-mail: pruet@eng.cmu.ac.th

R. Lee (ed.), *Computer and Information Science*, Studies in Computational Intelligence 566, DOI 10.1007/978-3-319-10509-3_8

distributed system (CDS) setting, a central controller is required for coordinating resource sharing and computing activities among computing machines. That is, all the system-wide decision makings are coordinated by the central controller. Contrary to the CDSs, in decentralized distributed systems (DDSs), multiple controllers can coexist and cooperate. Thus, management services such as job scheduling, resource discovery and resource allocation can be managed by the multiple controllers. If a controller fails, the other controllers can take over its responsibility autonomously. This is the major advantage of the DDSs. Examples of the DDSs are grid and cloud computing systems.

Although DDSs have the aforementioned advantage, completely decentralized nature of DDSs raises a big challenge in job scheduling [5, 6]. In addition, for composite jobs, i.e. jobs with multiple sub-processs or tasks, the tasks in the jobs can have dependency, e.g. a task requires an output of another task as its input. Thus, some tasks are prohibited to be executed concurrently. For instance, in order to calculate the histogram of collected sensing data, data grouping need to be performed first; thus, histogram calculation depends on data grouping. Therefore, the jobs must be executed under valid flow-constraints, generally, described as a workflow template. We can represent a workflow template using a directed acrylic graph (DAG) where the tasks are represented by nodes while the dependency (the execution order) between tasks is represented by edges. Figure 1 shows examples of workflow template where t_i represents a task with index i and w_j represents a workflow template with index j.

From Fig. 1, to finish a job with workflow template w_1, the tasks must be executed in the order of t_1, t_2, and t_3. In workflow template w_2, the tasks t_5 and t_6 can be executed in parallel after t_4 is completed. In workflow template w_3, each task has no dependency on the others, in this case, the set of edges is empty.

There are several attempts on decentralized job scheduling, e.g. [2–4, 8]. For example, in [8], a decentralized scheduling algorithm for web services using linear workflow is proposed. In this work, linear workflow allows only one incoming edge for each node. The goal of this work is to minimize the response time. However, this work only focuses on linear workflow which might not be practical in the real world.

In [7], the authors proposed a workflow-based composite job scheduling for decentralized distributed systems algorithm. The algorithm is based on an observation that

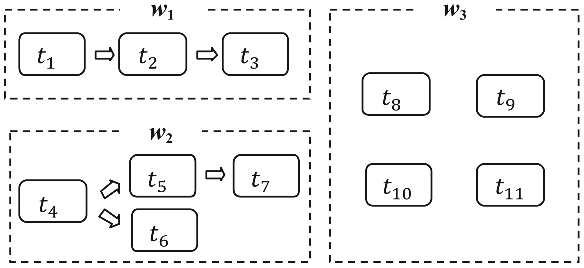


Fig. 1 Examples of workflow templates

the degree of each task significantly affects the execution time of all jobs. The result shows that this algorithm is efficient, the solution of this algorithm is mostly close to the optimal solution. However, this algorithm is evaluated only in the homogeneous resources decentralized distributed systems.

In this paper, we propose an algorithm for the mentioned problem with heterogeneous resources condition, i.e. each resource can only be used to execute some specific tasks from any workflow template. Such difference in resources often exists in the real-world scenarios e.g. the cost of some data centers might be too high for executing some tasks, or the result of some tasks might be too large to be propagated from one site to another economically. The main aim of the proposed algorithm is to minimize the total execution time when a DDS has to process multiple jobs simultaneously with such heterogeneous resources. As the problem has been proven as an NP-complete problem we propose a heuristic algorithm to find the solution.

The idea of the proposed algorithm is to allocate the resource to a task based on the heterogeneity of such resource. A resource with less heterogeneity, or more restriction, will be allocated to the task earlier. This can preserve the more heterogenous resources for the later tasks. The experimental results, which compare the proposed algorithm with a baseline algorithm in [7], is shown to illustrate the effectiveness and the efficiency of our work.

The rest of this paper is organized as follows: Sect. 2 introduces the basic definitions and formulates the **minimum length of time-slot with heterogeneous resource (MLTH)** problem. Section 3 presents a heuristic algorithm to find the minimum execution time. Experimental results are presented in Sects. 4 and 5 concludes this paper.

2 Basic Definitions and MLTH Problem

In this section, we introduce the basic notations and concepts based on the previous work in [7]. Then, the MLTH problem is defined with our additional heterogeneity notations.

Definition 1 (*Distributed System*) A distributed system D is presented by an undirected graph where each node corresponds to a machine in the system. The finite set $N(D)$ denotes the set of nodes in D, and the finite set $E(D)$ is the set of edges where each edge corresponds to a non-directed connection between two nodes.

Definition 2 (*Resource*) Let n_i be a node in $N(D)$, $n_i \in N(D)$. The resources of n_i are the computing units, that n_i can use to execute a computing process. The set of the resources of n_i is denoted as $R(n_i)$ whereas the set of resources of D is denated as Ω, $\Omega = \bigcup R(n_i)$.

Then, the decentralized distributed systems (DDSs) as in [5] can be defined as follows,

Definition 3 (*DDS*) Let D be a distributed system, D is classified as a decentralized distributed system (DDS) if it has multiple controllers, where the controller is a node that can allocate the resources of itself and some other nodes.

As the dependencies between some tasks processed by a DDS exist, then, tasks, workflow templates, jobs are defined as follows,

Definition 4 (*Task*) Let D be a DDS, a task in D is a unit of computing process that a node in D can complete execution in a unit of time. A set of all tasks that can be executed by the resources in Ω denotes by T.

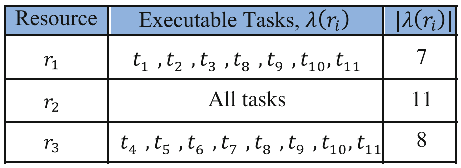
Definition 5 (*Workflow Template*) Workflow templates in D are the directed acyclic graph where each node corresponds to a task, t_i, and each edge indicates the data flow between two tasks. Given a workflow template w_x in the set of all workflow templates W, $\overline{N}(w_x)$ denotes the node set of w_x where $\overline{N}(w_x) \subseteq T$. On the other hand, $\overline{E}(w_x)$ denotes the directed edge set in the workflow template. For any workflow template w_x, task t_l is called a predecessor of task $t_{l'}$, if and only if, the order pair $(t_l, t_{l'}) \in \overline{E}(w_x)$. This indicates that task t_l must be executed and completed before execution of task $t_{l'}$. Meanwhile, task $t_{l'}$ is called a successor of task t_l. The node without incoming edge from other nodes is called the start task. On the other hand, the node without outgoing edge to other nodes is called the end task.

From Definition 5, we use the workflow template to be the template of any job that the system can execute. It can show the execution flow of a job, So, we define the definition of a job in Definition 6.

Definition 6 (*Job*) Let W be the set of workflow templates, a job j_k in D is an instance of a workflow template w_k in W. The task t_l of job j_k is denoted by t_l^k and $\overline{T}$ is the set of all tasks from J, where J is the set of all jobs in D.

As the DDSs may have the heterogeneous resources as mentioned before, each resource can only be used to execute the specific tasks from a workflow due to some limitation, e.g. cost of the communication. For example, assume that we have 3 resources in the system denoted as r_1, r_2, and r_3 as shown in Fig. 2. In the figure, an example of the executable tasks of each resource is shown. It can be seen that, resource r_1 can be used to execute only 7 tasks which are the tasks from workflow w_1 and w_2 in Fig. 1. Meanwhile resource r_2 can be used to execute all the tasks of the jobs that are an instance of any workflow template, i.e. w_1, w_2 or w_3.

In order to allow the heterogeneity in the DDSs, we define the executable set as follows.



Resource	Executable Tasks, $\lambda(r_i)$	$\lvert\lambda(r_i)\rvert$
r_1	t_1 , t_2 , t_3 , t_8 , t_9 , t_{10}, t_{11}	7
r_2	All tasks	11
r_3	t_4 , t_5 , t_6 , t_7 , t_8 , t_9 , t_{10}, t_{11}	8

Fig. 2 The executable tasks of r_1, r_2, and r_3

Definition 7 (*Executable Set*) Let r_p be the resource, $r_p \in \Omega$, an executable set of resource r_p is the set of tasks from all workflow templates that r_p can execute, denotes by $\lambda(r_p)$ where $\lambda(r_p) \subseteq \bigcup \overline{N}(w_x)$ where $w_x \in W$.

After the basic definitions related to the jobs have been defined, the time-slot is defined for describing the job scheduling, or the execution flow, as follows.

Definition 8 (*Time-slot*) Let J be a set of current jobs in D, the time-slot of J on Ω is the function $S : I^+ \times \Omega \rightarrow \overline{T} \cup \{null\}$ where $S(\alpha_q, r_p) \in \lambda(r_p)$, $r_p \in \Omega$, and α_q is a time unit where $\alpha_q \in I^+$ and I^+ is the set of natural numbers.

The domain of S is the order pair of time unit α_q, $\alpha_q \in I^+$, and resource r_p, $r_p \in \Omega$. Thus the range of S, $S(\alpha_q, r_p)$, is the executed task that uses resource r_p at time unit α_q, and $S(\alpha_q, r_p) \in \lambda(r_p)$. When there is no task to be executed on resource r_p at time unit α_q, $S(\alpha_q, r_p)$ is *null*.

For any $S(\alpha_q, r_p)$ and $S(\alpha_{q'}, r_{p'})$, where $p \neq p'$, if $q = q'$ then $S(\alpha_q, r_p)$ and $S(\alpha_{q'}, r_{p'})$ are executed in parallel. If $q < q'$ then $S(\alpha_q, r_p)$ is executed before $S(\alpha_{q'}, r_{p'})$, also, $S(\alpha_q, r_p)$ is executed before $S(\alpha_{q'}, r_p)$.

In order to illustrate the concepts clearly, the time-slot can be represented as a 2-dimensional matrix. Figure 3 shows a time-slot structure, a cell at row q and column p in the structure represents $S(\alpha_q, r_p)$. We can see that the representation can illustrate the execution flow of multiple tasks from multiple jobs along with their workflow template.

In order to define the problem precisely, we also introduce the length of the time-slot, as follows.

Definition 9 (*Length of Time-slot*) The length of time-slot S is the maximum value of time unit α_q which is $S(\alpha_q, r_p)$ is not *null*, $\exists r_p \in \Omega$.

For a DDS, there can be many versions of the time-slots for a given $\overline{T}$. For example, let us reconsider the given workflow templates in Fig. 1. Suppose that there are five

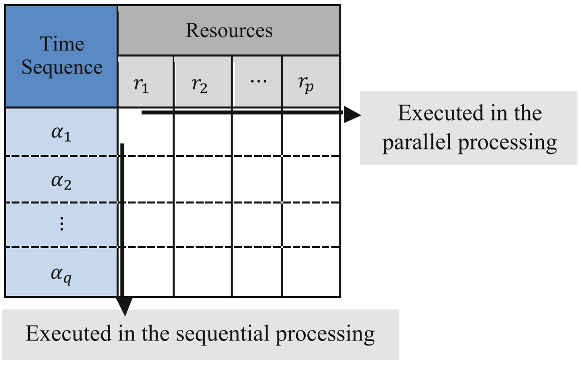


Fig. 3 The structure of timeslots

Fig. 4 Workflow template of j_1 to j_5

Job	Workflow template
j_1 , j_2	w_1
j_3	w_2
j_4 , j_5	w_3

Fig. 5 Time-slot I (length = 7)

Time Sequence	Resources		
	r_1	r_2	r_3
α_1	t_1^1	t_8^5	t_8^4
α_2	t_2^1	t_9^5	t_9^4
α_3	t_3^1	t_4^3	t_{10}^4
α_4	t_{10}^5	t_5^3	t_{11}^4
α_5	t_1^2	t_6^3	t_{11}^5
α_6	t_2^2	t_7^3	
α_7	t_3^2		

Fig. 6 Time-slot II

Resources		
r_1	r_2	r_3
t_1^1	t_1^2	t_8^4
t_2^1	t_2^2	t_9^4
t_3^1	t_3^2	t_4^3
t_5^3	t_{10}^4	t_8^5
t_6^3	t_{11}^4	t_9^5
t_7^3	t_{10}^5	t_{11}^5

jobs as shown in Figs. 3 and 4 resources, r_1, r_2, and r_3 with their executable tasks as in Fig. 2.

In Figs. 5, 6 and 7, three versions of the time-slots are presented. First, the time-slot with length of 7 time units is shown in Fig. 5. Although it is a valid time-slot subjected to the definitions, it is not an optimal time-slot length. Figure 6 shows a time-slot with 6 time units, however, resource r_1 can not be used to execute tasks t_5, t_6, and t_7 as shown in the dash line (the tasks of job j_3). So, Fig. 6 is not a valid time-slot. Last, Fig. 7 shows a valid minimal time-slot for this example, which it is the desirable solution for the problem.

After the required notations are defined, the minimum length time-slot with heterogeneous resources (MLTH) problem can be formulated as follows.

Fig. 7 Time-slot III

Resources		
r_1	r_2	r_3
t_1^1	t_1^2	t_4^3
t_2^1	t_2^2	t_5^3
t_3^1	t_6^3	t_8^4
t_8^5	t_9^5	t_7^3
t_{10}^5	t_3^2	t_9^4
t_{10}^4	t_{11}^4	t_{11}^5

Problem 1 (*MLTH*) Given a set of jobs J in a DDS D that belongs to a set of workflow templates W, find a time-slot S of J on the set of heterogeneous resources Ω that is the length of the time-slot is minimized.

3 A Heuristic Algorithm for the MLTH Problem

In [7], the MLT (minimum length time-slot problem) with homogenous resources is proven as an NP-Complete problem by reducing the problem from the subset sum problem. It can be seen that MLTH is also an NP-Complete problem by reducing the problem from the subset sum problem as well. We omit this proof because of space limitations, however, the same proof approach can be applied. So, we propose an effective heuristic algorithm to schedule the given jobs to the heterogenous resources as follows.

First, we follow an approach presented in [7] to manage the dependency of the given tasks. Thus, the degree of successors of each task is to be determined. Then, the tasks with higher degree of successors are to be executed earlier in order to minimize the waiting time of their successors.

Then, the next issue is to manage the heterogeneity in the DDSs, e.g. how to assign each selected task into the time-slot. For such focused issue, since of the resources are heterogeneous, the resources those can execute the selected task, supposedly t_y^x, are to be considered. Such set of resources is $R = \{r_p | t_y \in \lambda(r_p)\}$. Here, the set of free resources $R' \subseteq R$ is considered, where $R' = \{r_{p'} | S(\alpha_q, r_{p'}) = null\}$. First, for the allocation $S(\alpha_q, r_p) = null, r_p \in R$, we propose that the domain value of S, α_q, must be higher than the time of the predecessors of the task t_y^x in order to guarantee the validity of the workflow template. We also propose to select the free resource with such minimum time to execute the task as soon as possible. Subsequently, we select a single one resource $r_{p''}$ that has the minimum size of its executable set from R', and assign task t_y^x into the slot $S(\alpha_q, r_{p''})$.

The reason for our proposed approach is that, if resources with more heterogeneous, larger executable set, are assigned before less-heterogeneous resources, it can cause the tasks that execute later to have less choice for resource acquisition. Thus, their execution can be delayed, and the whole time-slot length will be longer. As our problem setting also consider task dependencies, the delay can cause more if the successor tasks are effected.

Last, as there can be many tasks with the same degree of successors and many resources with the same size of executable sets, selecting different task or resource can lead to different time-slot assignment. Thus, we utilize the nature of the DDSs by letting all the controllers to determine the local solution differently using the *local algorithm* described above. Subsequently, the time-slot with minimal length will be selected as the final global solution using a simple *interconnect algorithm*.

The details of the proposed algorithms are presented as follows.

3.1 Local Algorithm

The local algorithm for each controller is shown in Algorithm 1. The two major procedures of it are selecting a task for the allocation, and selecting a resource for the selected task as described above.

From the algorithm, first, the degree of successors, *scrDeg*, of each task in all jobs $j_x \in W$ is determined. Also, the set of predecessors, *pdr*, of each task is determined. From the algorithm, the size of *pdr*, denoted as $|pdr_y^x|$, is the degree of predecessors of the task. Then, the task is added to *taskSet* set, which it represents all the tasks in the system.

Subsequently, while the *taskSet* is not empty, the algorithm iterates through the *taskSet*. For each task without predecessor, i.e. $pdrDeg = 0$, it is added to another set, called *useableTask*. This set represents the candidate tasks that can be assigned into the time-slot. Then, the tasks with the highest degree of successors are selected. Though there could be many tasks with the same degree, the controller will randomly select one of them.

After selecting the task, the resource for it has to be decided. It begins with determining the *preAssignedTime* of the task. Formally, *preAssignedTime* of a task is the maximum time of the predecessor of such task, that has been assigned in the time-slot already. Next, the algorithm determines the slots of the resources that can execute the task where $preAssignedTime + 1$ is the beginning time of the valid slot. The *usableSlot* set therefore contains the resources that can execute the task.

Then, the algorithm selects a single slot $S(\alpha_{q'}, r_{p'})$ from *usableSlot* using the heterogeneity of the resource. More specifically, the algorithm selects the slot $S(\alpha_{q'}, r_{p'})$ which the size of the executable set of $r_{p'}$, $|\lambda(r_p)|$, is minimal. If there is more than one slot having the same level of heterogeneity, the algorithm selects the resource randomly.

Finally, the algorithm assigns the selected task to the selected resource. Also, it updates the *assignedTime* of this task, and the length of the time-slot. The *pdrDeg*

Algorithm 1 Local Algorithm

```
Require: a set of resources $\Omega$ of DDS $D$ with $\lambda$ and a set of jobs $J$ with a set of workflow templates $W$.
Ensure: a potentially minimal length Time-slot $S$.
  $taskSet \leftarrow \emptyset$, $usableTask \leftarrow \emptyset$, and $timeSlotLength = 0$
  for in each job $j_x \in W$ do
    Determine the degree of successors of each task $t_y$ in $j_x$,
      denoted as $scrDeg_y^x$.
    Determine the set of predecessors of each task $t_y$ in $j_x$
      as $pdr_y^x$.
    $pdrDeg_y^x = |pdr_y^x|$.
    $taskSet \leftarrow taskSet \cup \{t_y^x\}$ where $t_y^x$ is $t_y$ from $j_x$
  end for
  while $taskSet \neq \emptyset$ do
    $usableTask \leftarrow \emptyset$
    for each task $t_y^x \in taskSet$ do
      if $pdrDeg_y^x = 0$ then
        $usableTask \leftarrow usableTask \cup \{t_y^x\}$
      end if
    end for
    Determine the set of maximum-$scrDeg$ tasks from
      $usableTask$, denoted as $max\_scrDeg$.
    Select task $t_h^g$ from $max\_scrDeg$ randomly.
    Determine $preAssignedTime$ which is
      $max(\{assignedTime(t) | t$ is the predecessor of $t_h^g\})$,
      if $t_h^g$ is the start task, $preAssignedTime = 0$.
    $usableSlot = \emptyset$
    while $usableSlot = \emptyset$ do
      $usableSlot = \{S(\alpha_q, r_p) | S(\alpha_q, r_p) = null,$
        $t_h^g \in \lambda(r_p), \exists r_p \in \Omega,$
        and $\alpha_q = preAssignedTime + 1\}$
      $preAssignedTime = preAssignedTime + 1$
    end while
    Select slot $S(\alpha_{q'}, r_{p'})$ such that $|\lambda(r_{p'})|$ is the minimum
      from all elements in $usableSlot$, if there is more than
      one slot, select a slot randomly.
    $S(\alpha_{q'}, r_{p'}) = t_h^g$
    $assignedTime(t_h^g) = \alpha_{q'}$
    $taskSet = taskSet - \{t_h^g\}$
    if $\alpha_{q'} > timeSlotLength$ then
      $timeSlotLength = \alpha_{q'}$
    end if
    for each successor $t_{h'}^{g'}$ of $t_h^g$ do
      $pdrDeg_{h'}^{g'} = pdrDeg_{h'}^{g'} - 1$
    end for
  end while
  return $S$ and $timeSlotLength$
```

of successor of the assigned task is reduced by one. Such algorithm keeps repeating this described procedure until all the tasks are assigned to the time-slot.

The cost to resolve the MLTH problem using Algorithm 1 is $O(n^3m)$ where n is the number of all tasks, and m is the number of all resources. The main cost comes from the *usableSlot* determination, i.e. the set of slots that can assign the selected task into it. In each task, it takes $O(n^2m)$ to determine the *usableSlot*. Since, such computing is required until all tasks are completely assigned, so, the cost is $O(n^3m)$.

For the sake of clarity, we present an example to illustrate our local algorithm execution as follows. Let the set of jobs are given as shown in Fig. 4, and the set of resources and their executable set are as shown in Fig. 2. First, all of the start tasks from all jobs are considered as *useableTask* since their $pdrDeg = 0$. Then, the algorithm selects task t_4^3 from *useableTask* because it has the maximum degree of successors. As task t_4 of job j_3 is either in $\lambda(r_2)$ and $\lambda(r_3)$, so $usableSlot = \{S(\alpha_1, r_2), S(\alpha_1, r_3)\}$. Since t_4^3 is the start task, its $preAssignTime = 0$, and $S(\alpha_1, r_2)$ and $S(\alpha_1, r_3)$ is $null$, both slots are in *usableSlot* set. Finally, the algorithm selects $S(\alpha_1, r_3)$ to execute task t_4^3 because $|\lambda(r_3)| < |\lambda(r_2)|$. It also updates the *assignedTime* of t_4^3, $timeSlotLength$ and $pdrDeg$ of successors of t_4^3. The algorithm repeats all of the procedures until $taskSet = \emptyset$. Figure 7 is the solution from this running example.

3.2 Interconnect Algorithm

Once the local solutions have been computed by all the controllers using Algorithm 1, the global final solution is to be determined. It can be done simply by comparing the local solution from each controller, i.e. the time-slot length information. Subsequently, the minimal global time-slot length is determined, and such assignment is ready to be executed.

4 Evaluation

In this section, we present the experiment results to evaluate our proposed work.

4.1 Simulation Setup

To evaluate our work, a workflow synthetic-data generator, that generates the workflow template using the number of workflow templates, number of minimal and maximal tasks per jobs, and number of jobs as the inputs, is implemented.

In the experiments, the number of resources, the number of controllers, and the number of workflow tempts are fixed at 100 resources, 50 controllers, and 10 workflow templates respectively. The number of tasks of each workflow is fixed between 45–50 tasks. The degree of successors of each node is set between 0–5. The number of tasks in the longest path of each workflow template is set between 15–35 tasks. The workflow template for each job is selected uniform randomly. In order to guarantee that all jobs can be executed, in each experiment, 10 % of the resources are set to be able to execute all tasks from all workflow templates.

The experiment results of our proposed work are compared with a baseline algorithm in [7]. Such algorithm has demonstrated for its efficiency and effectiveness, i.e. its polynomial-time complexity and the solutions which are close to the theoretical results respectively. Both algorithms are implemented using Java SE 7. The experiments are conducted on an Intel Core 2 Duo 2.4 GHz with 4 GB memory running Mac OS X.

4.2 Result

First, to evaluate the impact of the heterogeneity on the performance of our algorithm, the size of the executable sets is varied from 10 to 100 % of the number of tasks from all workflow templates, while the number of jobs is fixed to 500 jobs.

Figure 8 shows the experimental results. It can be seen that when the size of executable set of each resource is small (10–30 %), the time-slot lengths from both

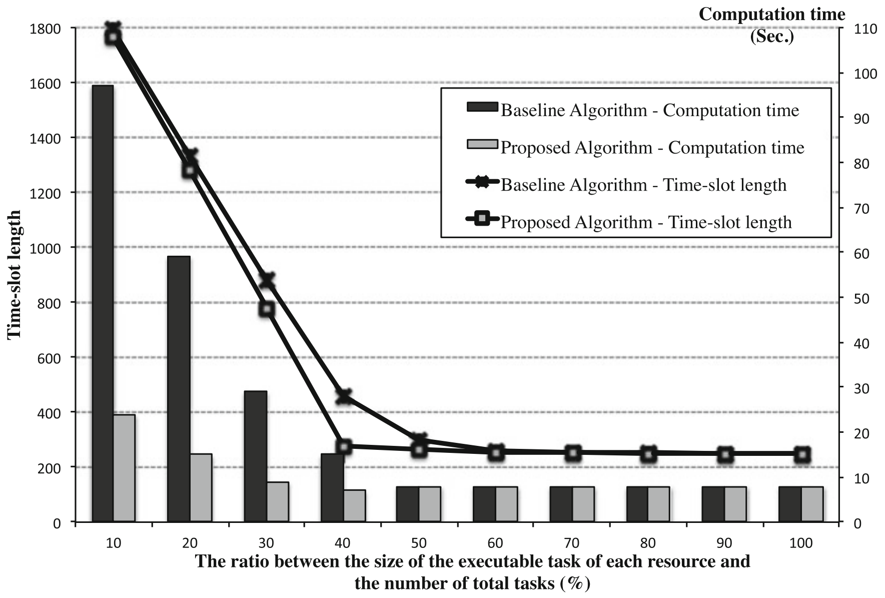


Fig. 8 The time-slot length and the computation time when the heterogenous level is varied

algorithms are quite large. However, the computation time of the proposed algorithm is very low comparing with the computation time of the algorithm in [7]. It is because the algorithm in [7] will specify the resource firstly, then it select a task form all available tasks to assign into the time-slot at this resource. If there is no task to be executed by this resource, the algorithm will select the next resource and finds a task for the assignment again. This can enormous delay the assignment in the heterogenous environment. Meanwhile, our proposed work will select the resource based on the executable set, which is also first available for the task (its $\alpha_q = \textit{preAssignedTime} + 1$). So, the execution time of our proposed work is much less than the baseline algorithm. When the size of the executable set is set at 30–50 %, the time-slot lengths from both algorithms are very different. The most different point is when the size of the executable set is fixed at 40 %, in which the time-slot length from proposed algorithm is 275 time units, while the time-slot length from [7] is 457 time units. The reason behind this is that the proposed algorithm considers the heterogeneity of the resources. Specifically, the resources with the small executable set are always allocated first. So, we preserve the resources with the larger executable set in the earlier phase of computing. When the algorithm finds the resources for the selected task later on, such task can be assigned earlier. In the other words, the delay due to waiting for the available executable resource is less. This gives more advantage particularly for the successor-task as discussed in the previous section. With this reasoning, the small time-slot length of the proposed algorithm can be achieved.

When the size of the executable sets are more than 50 %, the performance of both algorithms are similar. The reason is that each resource can execute many tasks from many workflow templates at these sizes, so, it has small number of *null* slots in both algorithms. Then, the time-slot lengths of both algorithms are too low. With the same reason, any selected task can be always assigned to the proper resource, this makes the computation time of the algorithm in [7] close to the computation time of the proposed algorithm.

After the impact of the task scheduling using the size of the executable sets in the heterogeneous resources has been evaluated, we then evaluate the impact of the number of jobs, or the scalability. The number of the jobs is varied from 50 to 500 jobs. In this experiment, we randomly generate the resources with the size of their executable sets, the heterogeneity of the resources, at a moderate level, i.e. 35–45 % of the number of total tasks from all workflow templates. The reason is that too large executable sets can cause the resources to be able to execute too many tasks from all workflow templates. In the other words, it causes the heterogeneity of the resources undistinguished. On the other hand, if the size of the executable sets is too small, it causes the resources to be able to execute too few tasks. So, the amount of delayed tasks can be extreme large due to waiting for available resources. This can be extreme cases in real-life applications.

Figure 9 shows the performance in terms of the time-slot length and the execution time. First, we consider the solutions from the experiments, i.e. minimal time-slot lengths. Figure 9 shows that the time-slot length from the proposed algorithm is obviously less than the time-slot length from the algorithm in [7]. And, when the

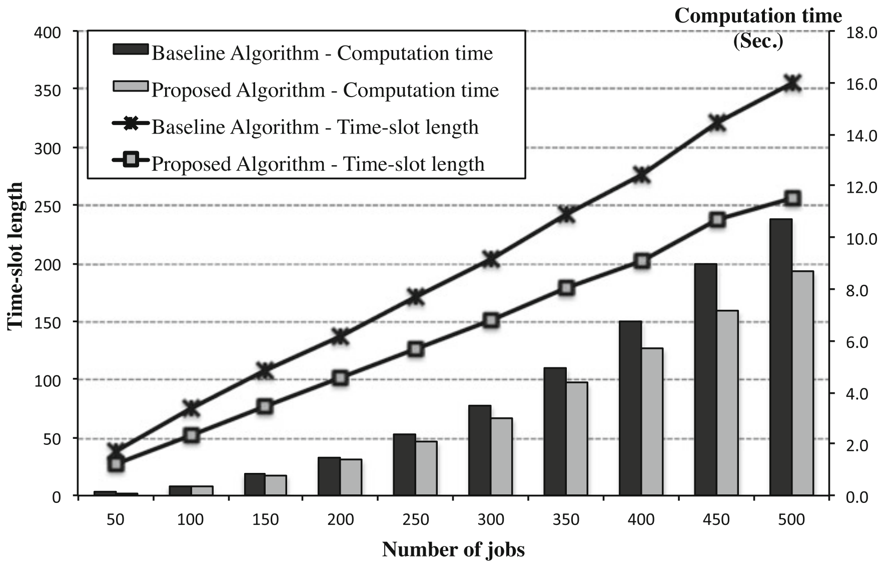


Fig. 9 The minimum length of time-slot and the computation time when the number of jobs is varied

number of jobs is increased, the difference of the time-slot length of two algorithms are increased. When considering the execution time, when the number of jobs is increased, the proposed algorithm is much efficient. The reason behind this is the complexity of the algorithm in [7] is $O(n^3m^2)$ meanwhile the complexity of the proposed algorithm is only $O(n^3m)$. When the execution time with the effectiveness of the solutions is considered, the proposed algorithm is highly effective. Even when the number of jobs is set at 500, our algorithm takes only 8.7 s to determine the time-slot which is very small. Thus, not only the algorithm is effective, but also it is efficient in decentralized distributed system with the heterogeneous resources.

5 Conclusion and Future Work

In this paper, we have addressed a scheduling problem in decentralized distributed systems with heterogeneous resources for the jobs with dependency, so called MLTH. As it is an NP-Complete problem, thus, a heuristic algorithm is proposed instead of aiming at the exact solution. Our proposed algorithm is based on the observation that the length of the schedule, time-slot length, can be reduced, if the resources with less heterogeneity is assigned to the tasks earlier. This not only can generate a smaller time-slot length, but the less execution time to generate the schedule can also be achieved. In order to evaluate the proposed work, the experiment results are presented. The results show that our approach is highly effective and also efficient, particularly, when the heterogeneity is at moderate level and the number of jobs is large. In the future work, we intend to investigate the approximation approach which can guarantee the quality of the solution. Moreover, we intend to address the similar problem with different scheduling objectives.

Acknowledgments The work is partially supported by Graduate School of Chiang Mai University. The authors would like to thank the colleagues at Data Engineering and Network Laboratory, Faculty of Engineering, Chiang Mai University for their support.

References

1. Kondo, D., Andrzejak, A., Anderson, D.P.: On correlated availability in internet-distributed systems. In: Proceedings of the 9th IEEE/ACM International Conference on Grid Computing, pp. 276–283. Washington, DC, USA (2008)
2. Lai, K., Huberman, B.A., Fine, L.R.: Tycoon: A distributed market-based resource allocation system. Comput. Res. Repos. cs.DC/0404013 (2004)
3. Mainland, G., Parkes, D.C., Welsh, M.: Decentralized, adaptive resource allocation for sensor networks. In: Proceedings of the 2nd conference on Symposium on Networked Systems Design and Implementation, Vol. 2, pp. 315–328. Berkeley, CA, USA (2005)
4. Masuishi, T., Kuriyama, H., Oki, Y., Mori, K.: Autonomous decentralized resource allocation for tracking dynamic load change. In: Proceedings of the International Symposium on Autonomous Decentralized Systems, pp. 277–283 (2005)
5. Pathan, AsK, Pathan, M., Lee, H.Y.: Advancements in Distributed Computing and Internet Technologies: Trends and Issues, 1st edn. Information Science Reference - Imprint of: IGI Publishing, Hershey, PA (2011)
6. Sotiriadis, S., Bessis, N., Xhafa, F., Antonopoulos, N.: From meta-computing to interoperable infrastructures: A review of meta-schedulers for hpc, grid and cloud. Advanced Information Networking and Applications, International Conference on 0, 874–883 (2012). http://doi.ieeecomputersociety.org/10.1109/AINA.2012.15
7. Tantitharanukul, N., Natwichai, J., Boonma., P.: Workflow-based composite job scheduling for decentralized distributed systems. In: Proceedings of the Sixteenth International Conference on Network-Based Information Systems (NBiS), pp. 583–588 (2013)
8. Tsamoura, E., Gounaris, A., Manolopoulos, Y.: Decentralized execution of linear workflows over web services. Futur. Gener. Comput. Syst. **27**(3), 290–291 (2011)