A Heuristic Scheduling Algorithm for Distributed Systems with Workflow Constraints

Nasi Tantitharanukul, Juggapong Natwichai, Pruet Boonma
Data Engineering and Network Technology Laboratory
Department of Computer Engineering, Faculty of Engineering,
Chiang Mai University, Chiang Mai 50200, Thailand
n.tantitharanukul@gmail.com, {juggapong, pruet}@eng.cmu.ac.th

ABSTRACT: *Distributed systems become one of the most important computing platform because they can efficiently handle large amount of data with a high computing workload. However, the job scheduling in the distributed systems is not a trivial issue. It can be even more complex when dealing with workflow-based composite jobs, i.e., each job has multiple tasks with dependencies between them. As the job scheduling problem has been proven to be an NP-hard, we propose a trail-based algorithm; Large Trail First (LTF), which is an effective heuristic approach for scheduling problem in the distributed systems when workflows exist. In this paper, "trail" of each task is the number of remaining tasks in each workflow. Thus, the idea of the algorithm is that, for each workflow job, the task with the largest size of the trail will be executed earliest. The experimental results show that the proposed approach is more effective and efficient than the other three approaches including a well-known 2-approximation algorithm.*

Keywords: Distributed system, Job scheduling, Workflow

Received: 20 June 2014, Revised 26 July 2014, Accepted 8 August 2014

1. Introduction

Distributed systems such as P2P, cloud computing, and computational grid have been widely used as platforms for high performance computing[2]. There has been a dramatic increase in the popularity of cloud computing and a reason of this increasing usage of the distributed system is thedevices in this system do not have any specific limitations to access resources. In particular, cloud environments provide an abstraction of infinite computing resources to their users so that the users can increase or decrease their resource consumption rate accordingly to the demands.

Although the distributed system has the aforementioned advantage, the job scheduling in the system is an challenging aspect. This is because the resources and computing tasks are distributed on many computing devices in the distributed system. Thus, assigning a job to a resource is not a trivial work; particularly for composite jobs, i.e., jobs with multiple sub-processes or tasks [6, 7]. In composite jobs, the tasks in the jobs can have dependencies, i.e., a task requires the outcome of another task as its input; thus, some tasks are prohibited to be executed concurrently. As a consequence, the jobs must be executed under valid flow-constraints, so called workflow templates. A workflow template is represented by a directed acyclic graph where the tasks are represented by nodes and the dependencies between tasks are represented by edges. In Figure 1, a few examples of

workflow template are presented, where t_i represents a task with index *i* and w_j represents a workflow template with index *j*.

From Figure 1, it can be seen that in order to execute a job with workflow template w_1, the tasks must be executed in the order of t_1, t_2, and t_3. For workflow template w_2, the tasks t_2 and t_3 can be executed in parallel after task t_1 is completed. In workflow template w_3, tasks have no dependency with the others; in this case, any arbitrary schedule can be applied.

There have been several attempts to effectively schedule jobs in distributed systems, especially in the past decade, such as [3], [4], [5], [8], and [11]. For example, in [11], a scheduling algorithm for web services using "*linear*" workflow is presented. The goal of this work is to minimize the response time. In such work, such type of workflow allows only one incoming edge for each node. Thus, this work might not be practical in the real world where a task normally requires multiple input.

As finding the minimum execution time of multiple jobs from multiple workflows has been proven to be an NP-complete problem [8]. In this paper, we propose a novel heuristic scheduling algorithms, namely, Large Trail First (LTF). The algorithm assigns the task that has largest set of remaining tasks into a resource first. In addition, we present the experiment results that compare the effectiveness and efficiency of the proposed algorithm with other two heuristic algorithms; Maximum-Degree First (MDF) [8] and Lowest-Level First (LLF), where MDF and LLF use the degree of successors and the level of task dependency to be the criterion of tasks assignment, respectively. As the work has been published in [10], in this paper, we report additional experiment results in which the proposed algorithm is compared with the classical 2-approximation algorithm, Coffman- Graham algorithm [1]. Such algorithm will be used as a baseline for our work.

The rest of this paper is organized as follows. Section 2 formulates the resource scheduling problem for workflow templates. In Section 3, LTF algorithm is proposed. Experimental results are presented in Section 4. Finally, Section 5 concludes this paper.

2. Problem Definition

In this section, we introduce the basic notations and concepts used in this paper. Then, the the minimum length time-slot problem is formulated.

Definition 2.1 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 2.2 Resource
Let n_i be a node in *D*, i.e., $n_i \in N(D)$. The resources of n_i are the computing units that n_i can use to execute computing processes. The set of the resources of n_i is denoted as $R(n_i)$ whereas the set of resources of *D* is denoted by Ω , i.e., $\Omega = \cup R(n_i)$. Based on the definition of distributed systems and resources , tasks, workflow templates, and jobs are defined as follows,

Definition 2.3 Task
Let *D* be a distributed system, 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 is denoted by *T*.

Definition 2.4 Workflow Template
Workflow templates in *D* are directed acyclic graphs where each node corresponds to a task, t_i, and each edge indicates the dependency 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 *wx* where $\overline{N}(w_x) \subseteq T$. On the other hand, $\overline{E}(w_x)$ denotes the directed edge set in the workflow template.

Definition 2.5 Predecessor and Successor
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 $t_{l'}$. Conversely, task $t_{l'}$ is called a successor of task t_l.

Definition 2.6 Start Task and End Task

Let w_x be a workflow template, the node in w_x without incoming edges is called the start task. On the other hand, the node without outgoing edges to other nodes is called the end task.

Definition 2.7 Job

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

Based on the definition of jobs and resources, time slot which is a basic notation for job scheduling on resources is defined as follows.

Definition 2.8 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 I^+ is the set of natural numbers.

The domain of S is the order pair of time sequence α_q, $\alpha_q \in I^+$, and resource r_p, $r_p \in \Omega$. The range of S, $S(\alpha_q, r_p)$, is the executed task that uses resource r_p at time sequence α_q. When there is no task to be executed on resource r_p at time sequence 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 $S(\alpha_{q'}, r_p)$.

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

Definition 2.9 Length of Time-slot

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

Finally, the minimum length time-slot (MLT) problem can be formulated as follows.

Problem 1 MLT. Given a set of jobs J in a distributed system D that belongs to a set of workflow templates W, find a minimal time-slot S of J on the set of resources Ω.

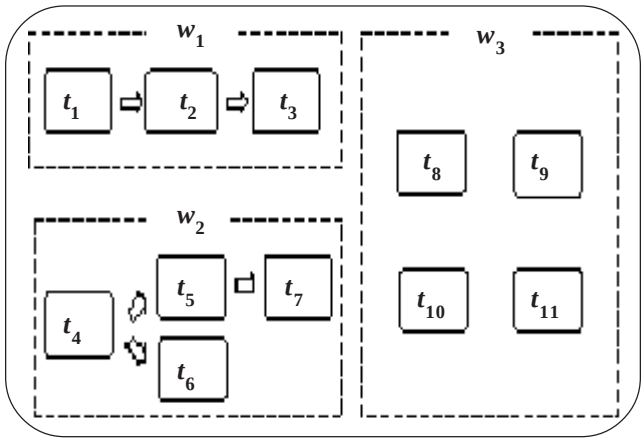

Figure 1. Examples of workflow template

3. A Heuristic Algorithm for MLT Problem

In this section we present the proposed algorithm, LTF algorithm.

3.1 LTF Algorithm

Algorithm 1 LTF algorithm.

Input: a set of resources Ω of a distributed system D and a set of jobs J with a set of workflow templates W.

Output: a potentially minimal length Time-slot S.

$taskSet \leftarrow \varnothing$, $usableTask \leftarrow \varnothing$, and $timeSlotLength = 0$

for in each job $j_x \in W$ **do**

Determine the number of tasks in the trail of each task t_y in j_x.

Determine the set of predecessors of each task t_y in j_x

as pdr^x_y .

$pdrDeg^x_y \leftarrow | pdr^x_y |$.

$taskSet \leftarrow taskSet \cup \{t^x_y\}$ where t^x_y is t_y from j_x

end for

while $taskSet \neq \varnothing$ **do**

$usableTask \leftarrow \varnothing$

for each task $t^x_y \in taskSet$ **do**

if $pdrDeg^x_y = 0$ **then**

$usableTask \leftarrow usableTask \cup \{t^x_y\}$

end if

end for

Determine the set of tasks with largest size of the trail from *usableTask* and select task t^g_h from this set

Determine *preAssignedTime* which is *max* ({*assignedTime* (t) $|t$ is the predecessor of t^g_h}),

if t^g_h is the start task, *preAssignedTime* = 0.

$usableSlot \leftarrow \varnothing$

while $usableSlot = \varnothing$ **do**

$usableSlot \leftarrow \{S(\alpha_q, r_p) \mid S(\alpha_q, r_p) = \text{null}, \text{ and } \alpha_q = preAssignedTime + 1\}$

$preAssignedTime \leftarrow preAssignedTime + 1$

end while

Select slot $S(\alpha_{q'}, r_{p'})$ in *usableSlot*.

$S(\alpha_{q'}, r_{p'}) \leftarrow t^g_h$

$assignedTime(t^g_h) \leftarrow \alpha_{q'}$

$taskSet \leftarrow taskSet - \{t^g_h\}$

if $\alpha_{q'} > timeSlotLength$ **then**

$timeSlotLength \leftarrow \alpha_{q'}$

end if

for each successor $t^{g'}_{h'}$ of t^g_h **do**

$pdrDeg^{g'}_{h'} \leftarrow pdrDeg^{g'}_{h'} - 1$

end for

end while

return S and *timeSlotLength*

In [8], MLT (minimum length time-slot) problem is proven as an NP-Complete problem by reducing the problem from the subset sum problem.Therefore, we propose a heuristic algorithm which is both effective and efficient. The basic idea of the proposed algorithm can be separated into two parts; the task selection and the resource selection. In the latter part, i.e. resource selection, we apply the idea proposed in [9]. That is, when there are two or more available resources for executing the task, this method selects the resource that allows the earliest execution. This is because the dependency of the resource waiting can be relaxed and can consequently reduce the time-slot length.

For the task selection which is our focus in this paper, we propose a trail-based concept, i.e. Large Trail First (LTF) task selection. First, LTF determines the trail size of each task, which is the cardinality of the set of all tasks that depend on such task. Then, the algorithm selects the task with the largest size of trail to be selected into the resource first. Because the task with large trail can have many tasks depend on it; therefore, if this task is executed lately, the trail of it will be executed lately too. Eventually, this will increase the time-slot length.

The proposed work is shown in Algorithm 1. First, the algorithm begins with determination of the size of the trail of each task for all jobs $j_x \in W$. Also, the set of predecessors, *pdr*, of each task is determined. Note that the size of *pdr*, denoted as $|pdr^{x}{}_{y}|$, is the degree of predecessors of any task t_y in job j_x. 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 that its predecessor has been assigned, i.e., *pdrDeg* = 0, it is added to another set, called *useableT ask*. This set represents the candidate tasks that is ready to be assigned into the time-slot. Then, the task with the largest trail, in the other words, the task with the highest number of remaining tasks depended on it, is selected to be executed. For example, in Figure 1, tasks t_5, t_6, and t_7 are the trail of t_4 so, the trail size of t_4 is 3 while t_2 and t_3 are the trail of task t_1, so the trail size of t_1 is 2. If we are given two jobs that are the instance of workflow templates w_1 and w_2, our approach will select the task t_4 before t_1 to be assigned to a resource.

After selecting the task, the resource for its execution has to be decided. It begins with determining the *preAssignedTime* of the task. 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*.

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* of each 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 computational complexity of Algorithm 1 is $O(n^2 m)$, where n is the number of all tasks, and m is the number of all resources. The main cost comes from the *usableSlo*t determination, i.e. the set of slots that can assign the selected task into it. For each task, it takes $O(nm)$ to determine the *usableSlot*. Since, such computing is required until all tasks are completely assigned, thus, the cost is $O(n^2 m)$.

3.2 Baseline Algorithm: Coffman-Graham Algorithm

In this paper, we extend the experiments as reported in [10] by comparing our proposed algorithm with a well-known 2-approximation algorithm named Coffman-Graham algorithm. Coffman-Graham algorithm is the one of classical list scheduling algorithms, introduced in [1]. The basic idea of this approach is to construct the priority list of tasks, then select the task one by one to be assigned in to the resource that can execute the selected tasks earliest. The list construction of this method is based on tasks labeling. That is, the labels are assigned to the tasks, from the end task to the starting task of each workflow template. While labeling the task, a list of tasks is constructed from the end of the list to the head of the list. On the other hand, the labels on these lists are ordered decreasingly. The complexity of this algorithm is also $O(n^2 m)$ as our proposed one.

Remark 3.1

$Int(t)$ denotes the decreasing sequence of integers formed by ordering of the set $\{label(t_{x'}) | t_{x'} \in Suc(t)\}$ where $Suc(t)$ is the set of successors of task t.

The list construction of Coffman-Graham algorithm can be shown as follows.

Algorithm 2 Coffman-Graham list construction algorithm.

Input: a set of jobs J with a set of workflow templates W.

Output: a list of tasks L.

Choose an arbitrary task t_k from $\overline{T}$ such that $|\text{Suc}(t_k)| = 0$, and assign $label(t_k) = 1$

for $i = 2$ to $|\overline{T}|$ **do**

Let $R = \{t_x \mid t_x$ be the unlabeled tasks with no unlabeled successors $\}$

Let t_{x*} be the task in R such that $Int(t_{x*})$ is lexicographically smaller than $Int(t_x)$ for all t_x in R

Let $label(t_{x*}) = i$

end for

Construct a list of tasks $L = < t'_n, t'_{n-1}, , t'_2, t'_1 >$ such that $label(t'_i) = i$ for all i where $1 \le i \le |\overline{T}|$

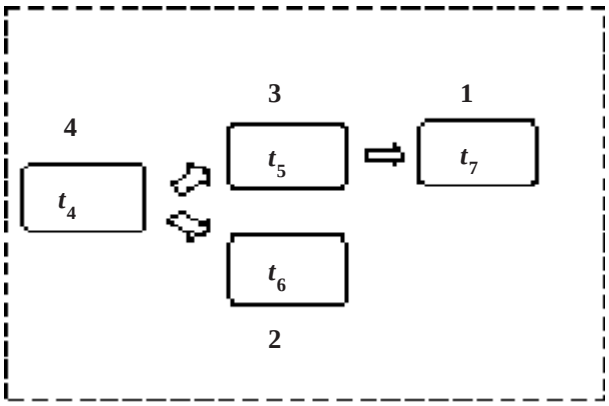

Figure 2. The example of tasks labeling

To construct a list of tasks by Coffman-Graham list scheduling algorithm, firstly an arbitrary task t_k in j' is selected such that its $|Suc(t_k)| = 0$, and its label is set at one. For the unlabeled tasks where their successors are all labeled, the algorithm labels a task that the label of its successors are smaller than the successor's label of the other tasks. Noe that the comparison is based on function *Int* (). The algorithm continues the execution of this procedure until all tasks are labeled. Then, a list L is constructed using the label of each task. The task that has the maximum value of label will be put at the head of the list. This process is executed until all the tasks are inserted into the list.

For example, if we have a job that is an instance of workflow template w_2 in Figure 1, the task labeling of this job can be assigned as in Figure 2. Then, the list that is constructed by such labels is shown in Figure 3. From the list, Coffman - Graham algorithm selects the task with label 4 to the task with label 1, in the other words from the head of the list to the end, to be assigned to the resources. In which, the resource selection algorithm as in [9] can be applied.

4. Experiment Results

In this section, we present the experiment results to evaluate our proposed work. The impact of three variables on the time-slot length and computation time of each algorithm, i.e. the number of jobs, the maximum degree of tasks, and the maximum length of workflow templates, are evaluated.

4.1 Simulation Setup

Our proposed work is evaluated using synthetic datasets from a workflow synthetic-data generator as in [8]. The generator takes the number of workflow templates, number of minimal and maximal tasks per job, and number of jobs as the inputs, and

Label	4	3	2	1
Task	t_4	t_5	t_6	t_7

Figure 3. The example of tasks labeling

generates the jobs based on such inputs with uniform randomization. In the experiments, the number of resources, and the number of workflow templates are fixed at 100 resources, and 10 workflow templates respectively. The number of tasks of each workflow is fixed between 45 - 60 tasks. The workflow template for each job is selected uniform randomly where the number of jobs form each workflow template is 10% of the number of all jobs.

We compare our work in Algorithm 1 with three algorithms, i.e. Coffman- Graham algorithm which is explained in the previous section, MDF and LLF algorithms. The MDF and the LLF are very similar to the proposed LTF algorithm. The differences is that MDF selects the task with maximum degree of successors to be executed earlier, meanwhile LLF selects the task with the lowest level to be executed earlier, considering the start task as the 0th level. All the algorithms are implemented using Java SE 7. The experiments are conducted on a computer with a Core 2 Duo 2.4 GHz processor and 4 GB RAM running Mac OS X. The resulting numbers are ten times average.

4.2 Results

First, to evaluate the impact of the number of jobs on the performance of the algorithms, the maximum degree of each task is fixed at 4, while the number of tasks in the longest path of each workflow template is fixed between 10 - 15 tasks.

Figure 4 shows the time-slot length and the computation time of each algorithm where the x-axis is the number of jobs, the left y-axis is the length of time-slot and the right y-axis is the computation time.

Max-path	5	10	15	20	25	30	35	40	45	50	55	60
Average	4.20	9.00	13.50	18.10	23.70	27.80	33.00	38.00	42.30	48.10	52.20	57.30
SD	0.92	1.05	1.78	2.18	2.26	3.36	3.30	2.87	3.27	2.88	3.74	3.53

Table 1. The average and the standard derivation of the longest path in all workflow templates

From Figure 4, it can be seen that the computation time of all algorithms are very close, and they are increased when the number of jobs is increased. Meanwhile the computational time of LLF algorithm is slightly lower than the other algorithms for all the job numbers. Also, it can be seen that the computational time of Coffman- Graham algorithm is slightly more than the other algorithms.

The reason behind this is that the LLF algorithm uses less time for the process of level determination of each task in any workflow template. Since, the level determination consumes only $O(k)$ where k is the constant number of the tasks in a workflow template. Meanwhile, the degree determination of each task in the workflow template, and finding the tasks number in the trail of any task, for MDF and LTF consume $O(k^2)$ and $O(k^3)$ respectively. For Coffman- Graham algorithm, its higher computation time comes from its process to determine t_{x^*} .

However, in term of the quality of the solution, the time-slot length, the proposed LTF algorithm along with MDF and Coffman-Graham algorithms can generate the solution with less time-slot length significantly. This comes from the fact that in this typical problem setting, taking the dependency into account can help reducing the time-slot length effectively.

Then, we evaluate the performance of the proposed algorithm when maximum degree of each task is varied. In this experiment, the number of jobs is fixed at 200. Figure 5 shows such experimental results. It can be seen that the time-slot length of our proposed LTF is not higher than the other three algorithms for all the maximum degree values. It means that prioritizing the task with larger trail size can effectively help its depended tasks to be executed earlier. For MDF algorithm, when the maximum degree

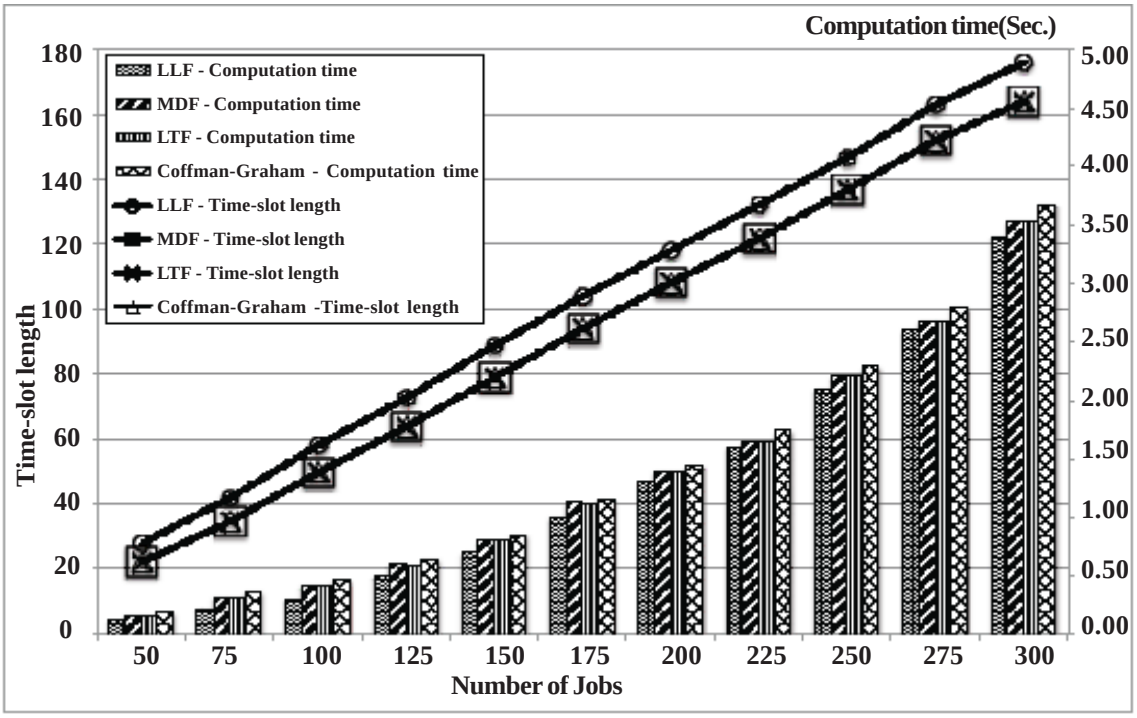

Figure 4. The time-slot length and the computation time of each algorithm when the number of jobs is varied

of each task in the workflow template is less (1-2), its performance is rather poor because of the limitation of the choice to select the task. For the computational time, it can be seen that the computation time is increased when the maximum degree of each task is increased for all algorithms. This is because, when we assign any task with higher degree of successors, its depended tasks can be assigned in the next time slot. So, higher maximum degree means that higher input for all the algorithms eventual. Additionally, it can be seen that the computational time of Coffman- Graham algorithm still higher than other algorithms.

In the last experiment, we evaluate the impact of the maximum number of tasks in the longest path of each workflow template. We fix number of jobs at 200. Meanwhile, in order to justify the result clearly, we also report the average and the standard derivation of the longest path of workflow templates for each setting in Table 1. In Figure 6, the result is shown. Obviously, when the number of the longest path of each workflow template is increased, the time-slot lengths from all algorithms are also increased. It is clear that our proposed LTF algorithm can generate the solution with less time-slot length, particularly when maximum length is set at 45-60. The reason behind this is there are 45 - 60 tasks in each workflow template, when the number of tasks in the longest path of each workflow template is increased, the degree of successors of each task is eventually decreased. So the result is similar to the previous experiment in Figure 5. Comparing with the Coffman- Graham algorithm with 2-approximation factor, it can be seen that our proposed work is very effective with less computation time.

From all of the experiment results, it is seen that LTF algorithm is as efficient as the other comparing algorithm, i.e. its computation time is not high. Also, it is highly effective as we can see from the performance in various settings.

5. Conclusion and FutureWork

In this paper, we have addressed a scheduling problem in distribted systems when the jobs have dependency among them; so called MLT problem. As it is an NP-Complete problem, thus, the heuristic algorithm, LTF, is proposed instead of aiming at the optimal solution. The idea of the LTF algorithm is to choose a task with the most number of tasks in its tail first. This aims at unblocking the depended tasks of a long-trail task from being executed effectively. In order to evaluate the proposed work, the experiment results are presented. The results show that the LTF algorithm is very effective, i.e. it can generate the solution with less time-slot length, for every kind of workflow template. Meanwhile, its computation time is close to the other three comparing algorithms. Thus, it is efficient to the number of tasks.

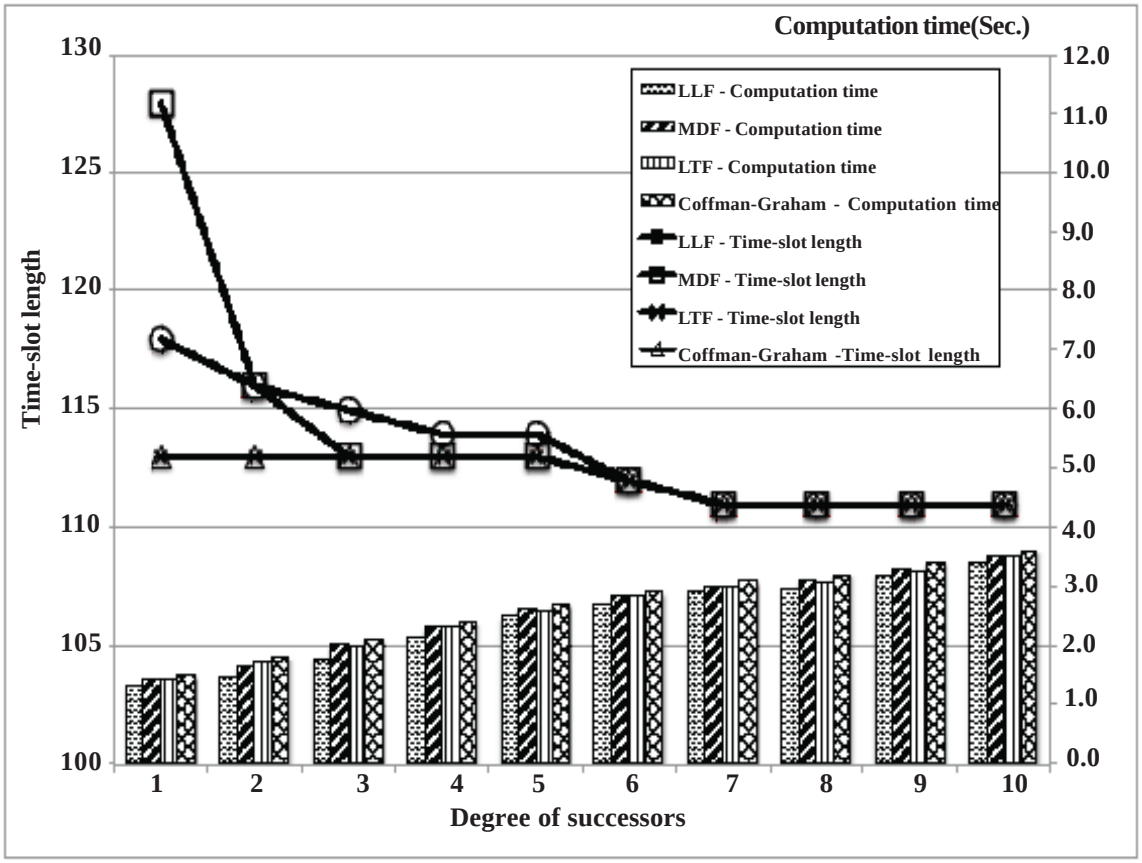

Figure 5. The time-slot length and the computation time of each algorithm when the maximum degree of each task is varied

References

[1] Jr. Coffman, E. G.., Graham, R. L. (1972). Optimal scheduling for two-processor systems. *Acta Informatica*, 1 (3) 200–213.

[2] Kondo, D., Andrzejak, A., Anderson, D. P. (2008). On correlated availability in internet-distributed systems. *In* : Proceedings of the 9*th* IEEE/ACM International Conference on Grid Computing, p. 276–283, Washington, DC, USA.

[3] Kevin Lai., Bernardo Huberman., A., Leslie Fine, R. (2004). Tycoon: A distributed market-based resource allocation system. Computing Research Repository, cs.DC/0404013.

[4] Geoffrey Mainland., David Parkes., C. (2005). Matt Welsh. *Decentralized, adaptive resource allocation for sensor networks*. *In :* Proceedings of the 2*nd* conference on Symposium on Networked Systems Design & Implementation - 2, p. 315–328, Berkeley, CA, USA.

[5] Tetsuya Masuishi., Hisayuki Kuriyama., Yasuyuki Oki., Kinji Mori. (2005). Autonomous decentralized resource allocation for tracking dynamic load change. *In*: Proceedings of the International Symposium on Autonomous Decentralized Systems, pages 277–283.

[6] Al-sakib Khan Pathan., Mukaddim Pathan., Hae Young Lee. (2011). Advancements in D*istributed Computing and Internet Technologies*: Trends and Issues. Information Science Reference - Imprint of: IGI Publishing, Hershey, PA, 1*st* edition.

[7] Stelios Sotiriadis, Nik Bessis., Fatos Xhafa., Nick Antonopoulos. 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.

[8] Nasi Tantitharanukul, Juggapong Natwichai, and Pruet Boonma. Workflow-based composite job scheduling for decentralized distributed systems. *In*: Proceedings of the Sixteenth International Conference on Network-Based Information Systems (NBiS), p 583–588, 2013.

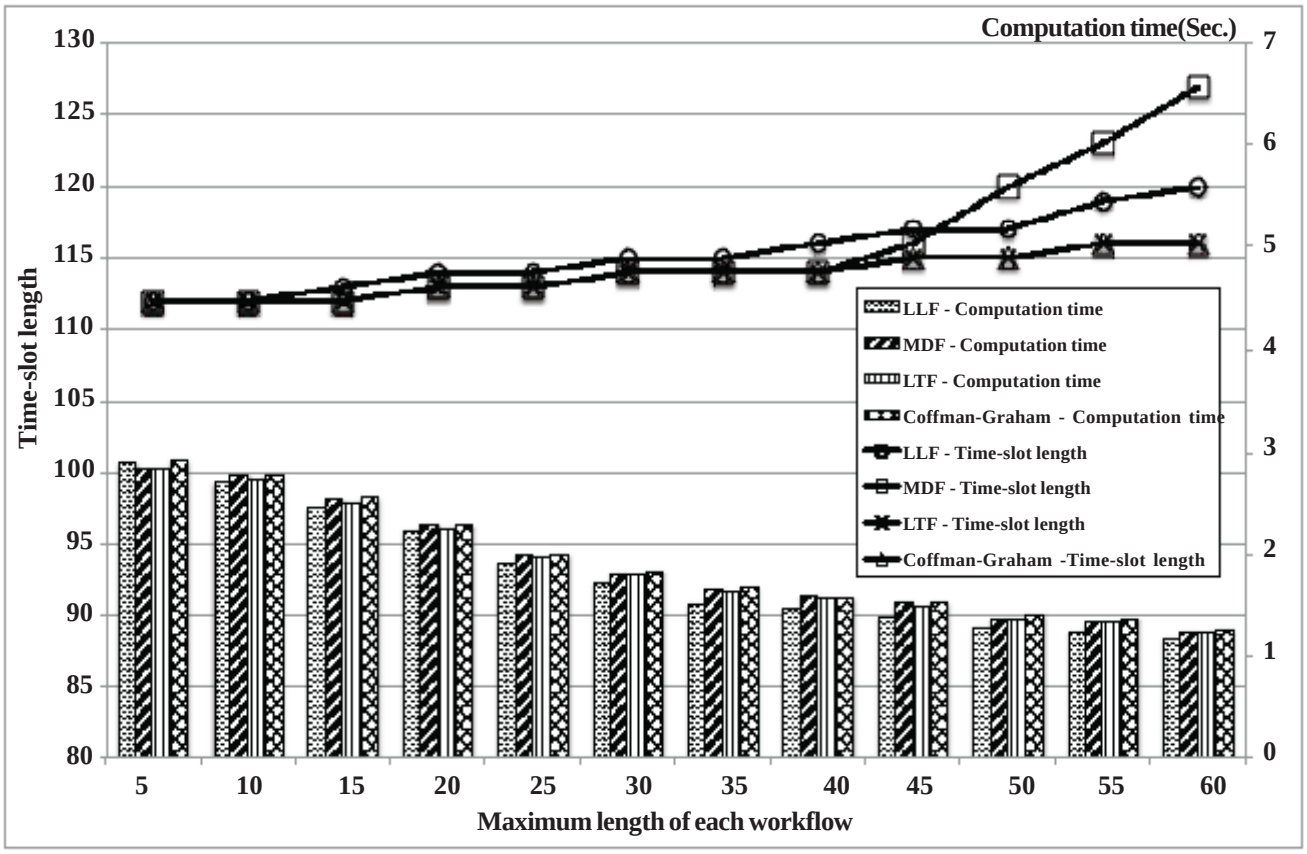

Figure 6. The time-slot length and the computation time of each algorithm when the maximum number of tasks in the longest path of each workflow template is varied

[9] Nasi Tantitharanukul, Juggapong Natwichai, and Pruet Boonma. (2014). A heuristic algorithm for workflow-based job scheduling in decentralized distributed systems with heterogeneous resources. Studies in Computational Intelligence, page to be appeared.

[10] Nasi Tantitharanukul, Juggapong Natwichai, and Pruet Boonma. (2014). On correlated availability in internet-distributed systems. In Proceedings of the 9th International Conference on Digital Information Management, Bangkok, Thailand.

[11] Efthymia Tsamoura, Anastasios Gounaris, and Yannis Manolopoulos. (2011). Decentralized execution of linear workflows over web services. Future Generation Computer Systems, 27(3), March.