Hindawi Publishing Corporation
Scientific Programming
Volume 2016, Article ID 9136107, 13 pages
http://dx.doi.org/10.1155/2016/9136107

Research Article

A Randomization Approach for Stochastic Workflow Scheduling in Clouds

Wei Zheng, Chen Wang, and Dongzhan Zhang

Department of Computer Science, School of Information Science and Engineering, Xiamen University, Xiamen 361005, China

Correspondence should be addressed to Dongzhan Zhang; zdz@xmu.edu.cn

Received 21 January 2016; Accepted 24 April 2016

Academic Editor: Laurence T. Yang

In cloud systems consisting of heterogeneous distributed resources, scheduling plays a key role to obtain good performance when complex applications are run. However, there is unavoidable error in predicting individual task execution times and data transmission times. When this error is being not negligible, deterministic scheduling approaches (i.e., scheduling based on accurate time prediction) may suffer. In this paper, we assume the error in time predictions is modelled in stochastic manner, and a novel randomization approach making use of the properties of random variables is proposed to improve deterministic scheduling. The randomization approach is applied to a classic deterministic scheduling heuristic, but its applicability is not limited to this one heuristic. Evaluation results obtained from extensive simulation show that the randomized scheduling approach can significantly outperform its static counterpart and the extra overhead introduced is not only controllable but also acceptable.

1. Introduction

Workflows have been widely used in various domains to depict complex computational application with multiple indivisible tasks and the data dependencies between tasks [1]. These workflows are normally derived from scientific problems in the fields of mathematics, astronomy, and so forth, for example, Montage [2]. As the massive requirements of calculation and communication became overwhelming in these disciplines, clusters and grids, as evolutionary forms of distributed computing, have been used to run workflow applications since the end of the 20th century [3, 4]. Recently, with more flexible and scalable capacity on computation and storage and less hardware/software installation expense, it has been gaining increasing popularity of using cloud computing infrastructure to run workflows [5–9].

The assignment of workflow tasks to computing resources, which is called *workflow scheduling*, is one of the key effects on the execution performance of the workflow in distributed computing environments like clouds. In general form of scheduling problems, a workflow is frequently represented by a Directed Acyclic Graph (DAG), where the nodes symbolize the tasks and the arcs with direction symbolize the data dependencies between tasks. The two terms "node" and "arc" will be interchangeably used in the rest of this paper. The most commonly focused objective of DAG scheduling is the minimization of the makespan (namely, overall execution time) of the workflow. Generally a DAG scheduling problem has been proven to be NP-Hard [10]. For getting closer favourable solution to this problem with acceptable time and space complexity, many researches have been carried out, and many heuristics have been proposed and published in the literature [11, 12].

Nevertheless, for majority of the existing DAG scheduling heuristics, the targeted DAG is modelled deterministically [13]. This means that the heuristics assumptions to the problem of inputs like task execution times and inter-task communication times are deterministic and precisely defined before. Clearly, the real-world workflow execution is much more complex than the above assumption because it is not possible to obtain accurate forecast of calculation and communication. Intuitively, the modelling of task computation times and communication times as random variables might be more realistic; it is also reasonabe to assume that the expected random variables and the variance can be predicted. In contrast to their deterministic counterparts,

the DAG scheduling problems modelled in stochastic fashion are called *stochastic scheduling* [14]. A big number of heuristics have been proposed for deterministic scheduling, but a small number for stochastic. There have been a plethora of studies showing deterministic DAG scheduling heuristics can hardly work well for their stochastic counterpart problems. This motivates the research on somehow adapting the existing deterministic DAG scheduling heuristics into the stochastic context and making improvement in terms of minimizing makespan, which is also the main focus of this paper.

In this paper, we consider the problem of scheduling a workflow onto a set of heterogeneous resources based on stochastic modelling of task execution times and task communication times with the objective of minimizing the makespan. For such a problem, a novel randomization scheduling approach is proposed. The proposed approach is applied to a classic deterministic scheduling heuristic and evaluated via extensive simulation experiments and the results exhibit a significant improvement of workflow execution performance with an acceptable extra overhead.

The rest of the paper is structured as follows. Related work is discussed in Section 2. The stochastic DAG scheduling problem and relevant definitions are presented in Section 3. The proposed randomization scheduling approach with an illustrative example is described in Section 4. The evaluation results are provided and discussed in Section 5. Finally, a conclusion is provided in Section 6.

2. Related Work

The past few decades witnessed a large number of efforts on developing various deterministic DAG scheduling heuristics, including HEFT [11], HBMCT [16], CPOP [11], GDL (DLS) [17], WBA [18], ILS [19], GA [20], SA [21], Triplet [22], TDS [23], STDS [24], and LDBS [25]. For an extensive list and classification of these heuristics we refer to [15, 26]. These heuristics differentiate with our work at the fact that they are based on deterministic scheduling model other than our stochastic model. Apparently, even when task execution times and data transmission times are modelled by some sort of random distribution, one can still apply one of the aforementioned deterministic scheduling heuristics by using the means of the random variables as inputs. However, as will be demonstrated later in this paper, this idea does not lead to a preferable schedule in most cases. Therefore, we derive a randomization approach by taking advantage of properties of random task execution times and data transmission times (as opposed to constant values). Although our approach can work with any deterministic heuristic in stochastic scheduling problems, we choose using the commonly cited and well-known deterministic heuristic: HEFT [11].

HEFT [11] is a deterministic list scheduling heuristic which aims to minimize the makespan of DAG applications on a bounded number of heterogeneous resources. The heuristic consists of two phases. In the first phase, the heuristic computes numeric ranks for all tasks based on their execution time predictions and data dependencies and then prioritizes tasks in a list. In the second phase, by the prioritized order, each task is allocated in turn to the resource which is estimated to minimize finished time of the task. It is worth mentioning that HEFT allows a task to be inserted into the existing task queue of a resource as long as the task dependency is not violated.

It has been widely recognized that due to the inaccuracy in time prediction deterministic scheduling heuristic relying on constant input may result in bad decision [27]. Over the recent years, some works have been carried out to evaluate the performance of deterministic DAG scheduling heuristics with stochastic model, such as [26, 28]. However, the main focus of these works is not on proposing an efficient scheduling heuristic for the stochastic DAG scheduling problem. One of recently popular ideas of addressing stochastic DAG scheduling problems is adapting the existing deterministic heuristics by changing their ranking function and/or the way of comparing task attributes. Examples can be found in [29–31]. Nevertheless, these heuristics still make scheduling decision in a deterministic manner. In contrast, our heuristic is randomized.

In our previous study [15], a Monte-Carlo based approach has been applied to the stochastic DAG scheduling problem to acquire a schedule which can outperform schedules generated by other comparable means. In essence, this approach relies on a significant amount of random searches on the solution space as well as an extensive evaluation for picking up the result schedule. As a result, a result schedule with a reasonable performance requires a considerable overhead. This paper is an extension to our previous work [32]. In this paper, we only do local search around a well-crafted schedule and the cost of evaluation of the searching results is trivial. That is to say, the overhead of the heuristic proposed in this paper is much less than the Monte-Carlo based approach, while a significant improvement on makespan can still be achieved.

3. Problem Description

3.1. Application Model. In this paper, a workflow application is represented by a Directed Acyclic Graph (DAG) $G = \{V, E\}$, where V denotes a set of interdependent tasks, each of which is represented by v_i, that is, $V = \{v_i \mid i = 1, 2, \ldots, n\}$, and E denotes a set of directed arcs, each of which represents data dependency between two tasks, that is, $E = \{e_{j,k}\}$. For instance, $e_{j,k} = v_j \rightarrow v_k$ means the input of task v_k depends on the output of task v_j. In this case, v_k is a child of v_j and v_j is a parent of v_k; moreover, v_k cannot start to run before receiving all necessary data from its parents. The node without parents is called entry node, and the node without children exit node. For the sake of standardization we assume that every DAG here has only one single entry node and one single exit node. For illustration, Figure 1 presents an example of DAG with 10 nodes.

3.2. Platform Model. We assume that the underlying computing platform comprises a fixed number of heterogeneous resources and uses $R = \{r_1, r_2, \ldots, r_m\}$ to denote the set of resources. A task v can be executed at any resource p; however a resource cannot execute more than one task at a time. In addition, no temporal interruption is allowed

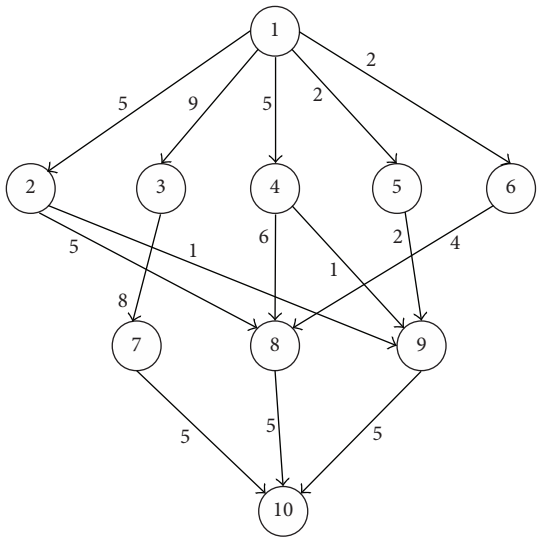


Figure 1: A DAG example.

during this execution. There is a dedicated dual-direction communication link between every pair of resources. This means that the tasks allocated onto different resources can transmit data to each other with no contention. For any random variable RV, we use $\mu(\mathrm{RV})$ and $\sigma(\mathrm{RV})$ to represent the expectation and the standard deviation of RV. Given that $\mathrm{ET}_{i,p}$ represents the random variable modelling the execution time of task v_i on resource r_p and $\mathrm{CT}_{i,j,p,q}$ the time for transmitting data from v_i located on r_p to v_j located on r_q, we assume that for each i and p ($1 \le i \le n$, $1 \le p \le m$) $\mu(\mathrm{ET}_{i,p})$ and $\sigma(\mathrm{ET}_{i,p})$ are known. We also assume the data amount transmitted on each edge (denoted by $\delta_{i,j}$), and the average time needed for transmitting one unit of data from one resource to another (denoted by $\gamma_{p,q}$) is known, so

$$\mu\left(\mathrm{CT}_{i,j,p,q}\right) = \delta_{i,j} \cdot \gamma_{p,q}, \tag{1}$$

as well as $\sigma(\mathrm{CT}_{i,j,p,q})$, is known. For illustration, Tables 1(a) and 1(b) show the stochastic model of task execution times and communication times of the DAG depicted in Figure 1. Table 2 shows an example of possible outcome of the stochastic model presented in Tables 1(a) and 1(b).

We use $\mathrm{ST}_{j,q}$ to denote the start time of task v_j on resource r_q and $\mathrm{FT}_{j,q}$ the finish time. In addition, we let $\mathrm{pre}(j)$ and $\mathrm{suc}(j)$ stand for the set of parents and children of v_j, respectively, and $\mathrm{alloc}(j)$ symbolize the resource where v_j is allocated. Obviously, we have

$$\mathrm{FT}_{j,q} = \mathrm{ST}_{j,q} + \mathrm{ET}_{j,q}. \tag{2}$$

In addition,

$$\mathrm{ST}_{j,q} = \max\left\{\mathrm{FT}_{i,p} + \mathrm{CT}_{i,j,p,q} \mid i \in \mathrm{pre}\left(j\right),\ p = \mathrm{alloc}\left(i\right)\right\}. \tag{3}$$

Table 1: Stochastic model of the DAG example shown in Figure 1.

(a) Stochastic model of execution times

	Resource 1		Resource 2		Resource 3	
	Exp. val.	Std. dev.	Exp. val.	Std. dev.	Exp. val.	Std. dev.
Task 1	24.00	4.00	2.00	0.33	7.00	1.17
Task 2	36.00	6.00	32.00	5.33	49.00	8.17
Task 3	42.00	7.00	49.00	8.17	12.00	2.00
Task 4	12.00	2.00	8.00	1.33	56.00	9.33
Task 5	81.00	13.50	20.00	3.33	16.00	2.67
Task 6	9.00	1.50	30.00	5.00	24.00	4.00
Task 7	48.00	8.00	4.00	0.67	8.00	1.33
Task 8	64.00	10.67	18.00	3.00	42.00	7.00
Task 9	36.00	6.00	16.00	2.67	1.00	0.17
Task 10	54.00	9.00	28.00	4.67	63.00	10.50

(b) Stochastic model of communication times

	Resource 1		Resource 2		Resource 3	
	Exp. val.	Std. dev.	Exp. val.	Std. dev.	Exp. val.	Std. dev.
Resource 1	0.00	0	2.00	0.33	8.00	1.33
Resource 2	2.00	0.33	0.00	0.00	4.00	0.67
Resource 3	8.00	1.33	4.00	0.67	0.00	0.00

Table 2: A sampling of the stochastic model shown in Table 1.

(a) A sampling of stochastic task execution times

	Resource 1	Resource 2	Resource 3
Task 1	13.45	2.05	6.75
Task 2	33.44	22.64	45.42
Task 3	25.05	51.33	11.61
Task 4	14.82	8.85	47.48
Task 5	88.77	22.64	15.39
Task 6	6.59	25.23	26.17
Task 7	35.47	4.05	9.08
Task 8	54.31	22.29	50.60
Task 9	33.52	19.80	1.28
Task 10	56.57	28.59	77.36

(b) A sampling of stochastic task communication times

	Resource 1	Resource 2	Resource 3
Resource 1	0.00	2.58	8.46
Resource 2	2.58	0.00	5.58
Resource 3	8.46	5.58	0.00

3.3. Problem Definition. The main objective of this paper is to minimize the overall execution time of a DAG application. Given that the start time of entry node is always time 0, based on the definitions and assumptions presented above, the problem to be addressed in this paper is to generate a schedule S, which specifies the mapping of tasks and resources, as well as the execution order of tasks on each resource, so that the random variable $\mathrm{FT}_{\mathrm{exit\_node},\,\mathrm{alloc(exit\_node)}}$ can be minimized.

Input: DAG G on a set of resources R with stochastic model
Output: A schedule specifying task mapping and execution order
(1) Create a candidate schedule list $\mathscr{L}$, which is initially empty.
(2) Run a deterministic heuristic DH to generate the schedule entirely based on all mean values of task execution times and communication times, and put the schedule into $\mathscr{L}$.
(3) **while** the termination condition of the producing phase is not met **repeat**
(4) Run the randomized heuristic RH and push the result schedule into $\mathscr{L}$.
(5) **endwhile**
(6) Compute the expected makespan based on mean values of all stochastic inputs for each schedule in $\mathscr{L}$.
(7) **Return** the schedule with the minimum expected makespan as the result schedule.

Algorithm 1: The outline of the randomization scheduling approach.

4. Method

In this section, we firstly present the basic idea of the proposed randomization approach and then describe the details of the randomized HEFT heuristic (RHEFT), which is derived from the combination of our randomization approach and the classic HEFT heuristic. Furthermore, an enhanced version of RHEFT (named, ERHEFT) is proposed.

4.1. Basic Idea. Among different categories of deterministic DAG scheduling heuristics, list scheduling seems receiving most research attention, because this kind of heuristics can usually obtain a reasonably good schedule result without too much overhead. As mentioned in Section 1, deterministic DAG scheduling heuristics receive constant time prediction as inputs. List scheduling heuristics firstly sort all workflow tasks in terms of a rank, which is computed based on the time prediction and associated with each task, and then allocate the sorted tasks one after another onto a specific resource. The resource where a task is allocated is normally decided by which resource can minimize a certain time-related attribute (e.g., estimated finish time of the current task) that can be calculated based on the time prediction and is distinctive on different resources. By doing so, list scheduling aims at a good trade-off between optimization and algorithm complexity. However, there is no guarantee that the resource allocation made is the best possible decision for minimizing makespan.

The issue is more complicated in the case of stochastic scheduling model. It will be more doubtable whether the scheduling decision made by deterministic list scheduling results will be favourable for minimizing the makespan, because the time prediction based on which decision is made is unreliable.

Assuming the time prediction can be modelled by probability distribution, our hypothesis is that it is almost impossible to develop a static algorithm which can always obtain an optimal schedule. Therefore, we consider generating a set of various schedules based on the random prediction and pick up that one which has the best chance to win. As the time prediction is modelled randomly, when comparing two time-related variables, for example, task finish times of different tasks, there is no certain result. To decide which task finished earlier, we need to roll a dice. Apparently, by randomly deciding the comparison result of task finish times, which is important factors for making scheduling decision, we will generate various scheduling results. We regard all these schedules as candidates and the one which has the smallest expectation of makespan as the final output of our approach.

The outline of our randomization scheduling approach is presented in Algorithm 1, where DH denotes a given deterministic DAG scheduling heuristic and RH means a randomized scheduling heuristic.

4.2. The Randomized HEFT Heuristic (RHEFT). Among existing list scheduling heuristics, the most well-known and commonly cited one is heterogeneous-earliest-finish-time (HEFT) heuristic [11]. We thereby choose to apply the aforementioned randomization approach to HEFT and propose a novel approach named RHEFT which is based on a randomized HEFT. Namely, we let DH be HEFT and RH the randomized HEFT.

The details of the randomized HEFT heuristic are presented in Algorithm 2. Similar to HEFT, the randomized HEFT has two phases. In the first phase (Algorithm 2, lines: 1-2), the upward ranking of each node (denoted by Urank) is computed as follows:

$$\mathrm{Urank}\left(v_i\right) = \overline{\mathrm{ET}_i} + \max_{v_j \in \mathrm{suc}(i)} \left\{\overline{\mathrm{CT}_{i,j}} + \mathrm{Urank}\left(v_j\right)\right\}, \tag{4}$$

where

$$\overline{\mathrm{ET}_i} = \frac{\sum_{p=1}^{m} \mu\left(\mathrm{ET}_{i,p}\right)}{m},$$
$$\overline{\mathrm{CT}_{i,j}} = \frac{\sum_{p=1}^{m} \sum_{q=1}^{m} \mu\left(\mathrm{CT}_{i,j,p,q}\right)}{m \cdot m}. \tag{5}$$

Especially for exit node, we have $\mathrm{Urank}(v_{\mathrm{exit\_node}}) = \overline{\mathrm{ET}_{\mathrm{exit\_node}}}$.

In the second phase of the randomized HEFT (Algorithm 2, lines: 3–9), it is needed to compute for each node on each resource the earliest estimate start time (denoted by

(1) Compute Urank (as defined in (4)) for all tasks.
(2) Sort all tasks in a list $\mathscr{L}$ in the non-descending order of Urank.
(3) **for** $k \coloneqq 1$ to n **do** (where n is the number of tasks)
(4) Select the kth task v^* from the list $\mathscr{L}$.
(5) **for** each resource $r_p \in R$ **do**
(6) Compute the mean value and variance of estimated finish time of v^* on r_p (as defined in (7)).
(7) **endfor**
(8) Decide the winner resource of estimated finish time (r^*) for v^* according to the *random comparison policy*.
(9) Allocate v^* to r^*.
(10) **endfor**

Algorithm 2: The randomized HEFT heuristic.

EST) and the earliest estimate finish time (denoted by EFT), which are defined as follows:

$$\mathrm{EST}_{j,q} = \max\left\{\mathrm{AVT}_q, \max_{v_i \in \mathrm{pre}(j)}\left\{\mathrm{EFT}_{i,p} + \mathrm{CT}_{i,j,p,q}\right\}\right\}, \quad (6)$$

$$\mathrm{EFT}_{j,q} = \mathrm{EST}_{j,q} + \mathrm{ET}_{j,q}, \quad (7)$$

where AVT_q is the earliest available time of resource r_q to allocate v_j, taking into account the current load of r_q and the estimate execution time of v_j on r_q (i.e., $\omega(\mathrm{ET}_{j,q})$). In addition, $r_p = \mathrm{alloc}(i)$. Apparently, at this moment, v_i has already been allocated. Particularly, for the entry node, $\mathrm{EST}_{\mathrm{entry\_node},q} = 0$ for every r_q. The main difference between the randomized HEFT and its initial version is the random comparison policy mentioned in line 8. For each task, we use this comparison policy to compare the estimated finish times (i.e., EFT) of different resources, which are random variables in our stochastic model, as the estimated finish time is determined by summing up the task execution times and the communication times along the critical path. Let CP denote the critical path, ND the set of nodes, and ED the set of edges along with CP; then we have the earliest finish time as below:

$$\mathrm{EFT} = \sum_{i \in \mathrm{ND}} \mathrm{ET}_i + \sum_{j \in \mathrm{ED}} \mathrm{CT}_j. \quad (8)$$

For ease of analysis, we assume all task execution times and communication times follow normal distribution. Then we have

$$\begin{aligned} \mu(\mathrm{EFT}) &= \sum_{i \in \mathrm{ND}} \mu\left(\mathrm{ET}_i\right) + \sum_{j \in \mathrm{ED}} \mu\left(\mathrm{CT}_j\right), \\ \sigma(\mathrm{EFT})^2 &= \sum_{i \in \mathrm{ND}} \left(\sigma\left(\mathrm{ET}_i\right)^2\right) + \sum_{j \in \mathrm{ED}} \left(\sigma\left(\mathrm{CT}_i\right)^2\right). \end{aligned} \quad (9)$$

Say there are two random variables EFT and EFT′ to compare and determine which is larger. We let DV = EFT − EFT′. Then we have

$$\begin{aligned} \mu(\mathrm{DV}) &= \mu(\mathrm{EFT}) - \mu\left(\mathrm{EFT}'\right), \\ \sigma(\mathrm{DV})^2 &= \sigma(\mathrm{EFT})^2 + \sigma\left(\mathrm{EFT}'\right)^2. \end{aligned} \quad (10)$$

Apparently, the value of DV corresponds to the comparison result of EFT and EFT′. The key idea of our random comparison policy is to make a random draw of DV value, if the outcome is less than zero, EFT < EFT′; otherwise, EFT ≥ EFT′. However, to implement a precise random draw of DV is not easy. Therefore, we use μ(DV) and σ(DV) for approximation. In detail, without losing generality, we assume $\mu(\mathrm{DV}) > 0$, and if $2 \times \sigma(\mathrm{DV}) > \mu(\mathrm{DV})$,

$$P(\mathrm{DV} < 0) = \frac{2 \times \sigma(\mathrm{DV}) - \mu(\mathrm{DV})}{4 \times \sigma(\mathrm{DV})}; \quad (11)$$

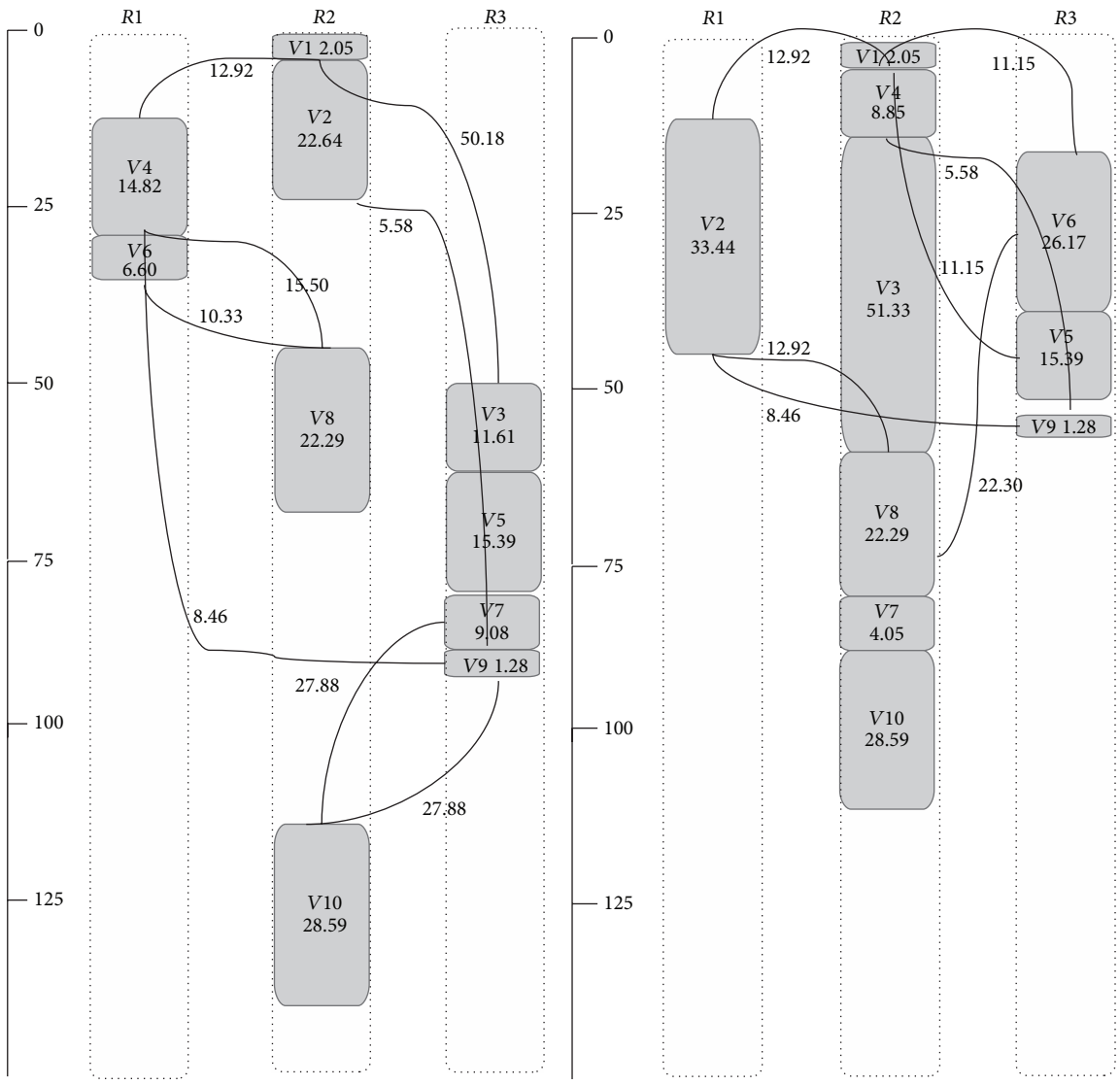
else $P(\mathrm{DV} < 0) = 0$. One can easily get $P(\mathrm{DV} \geq 0) = 1 - P(\mathrm{DV} < 0)$. That is to say, the comparison result between EFT and EFT′ is a 0-1 random variable. With probability of $P(\mathrm{DV} < 0)$, the outcome is EFT < EFT′; with probability of $P(\mathrm{DV} \geq 0)$, EFT ≥ EFT′. By this random comparison policy, the deterministic HEFT is randomized. Note that as described in Algorithm 1 (lines 3–5), RHEFT requires running the randomized HEFT for a certain number of times to generate candidate schedules.

For illustration purpose, we run HEFT and RHEFT to the DAG example modelled by Figure 1 and Table 1 and obtain scheduling results, respectively. For RHEFT, the scheduling result is picked up from 10 candidates. In order to present these scheduling results, we assume the values shown in Table 2 are the real task execution times and data transmission times. Then the scheduling details can be depicted by Figure 2. In this case, RHEFT obtains a makespan of 117.17, which makes a significant improvement (19.8%) to HEFT with a makespan of 146.15.

4.3. Further Investigation on RHEFT. Although the illustrative example shows that RHEFT can significantly outperform HEFT on minimizing makespan, there are still two open issues regarding with the configuration of RHEFT which are worthy of further investigation:

(i) In RHEFT, how many candidate schedules should we generate to gain substantial improvement on makespan without paying too much overhead?

(ii) How often and to what extent can RHEFT improve the makespan obtained by HEFT over various DAG examples?



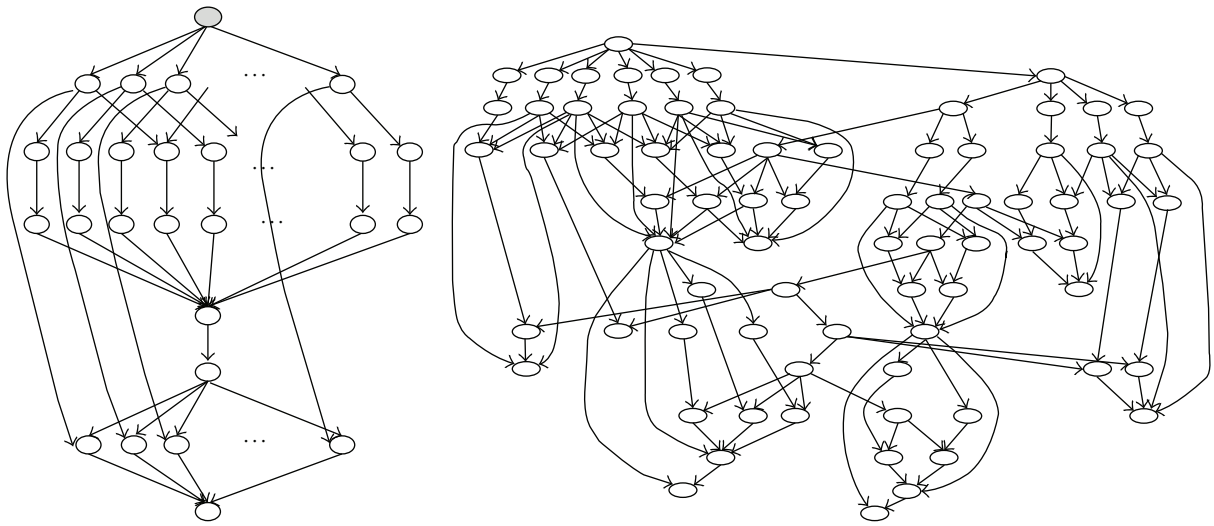
(a) HEFT (b) RHEFT

Figure 2: Scheduling result of the DAG example shown in Figure 1 by using (a) HEFT; (b) RHEFT.

We examine these two issues by evaluation in the rest of this subsection.

4.3.1. Evaluation Setting. We built a simulated computing system and a stochastic DAG generator, which allow various parameter settings. We use two types of DAGs: Montage [2] and LIGO [33] as shown in Figure 3, which are derived from real-world applications and have distinctive structures and sizes. We consider the number of resources used to be 3 and 8. All random variables used in our stochastic model are assumed to follow normal distribution. For the execution time of each task on each resource, we randomly select its expected value from the range of $[1, 100]$ and its standard deviation value as 1/6 of its expected value. Similar setting is applied to the communication time between tasks on different resources. We specify a parameter named communication-computation-ratio (CCR), which means the ratio between the average communication cost and the computation one, and adjust the value of communication times to meet the specified CCR value. We randomly specify the CCR value from [0.5, 1.5] in our experiments.

4.3.2. How Many Candidate Schedules Are Needed? One can easily imagine that the more times RHEFT runs the randomized HEFT heuristic, the more likely a better candidate schedule may be obtained, and on the other hand the more time cost is needed to be paid. In order to examine how many times RHEFT should repeat running the randomized scheduling procedure, we specify four DAG instances and observe the expected makespan RHEFT can gain for these DAGs as the number of repetition grows. Figure 4 shows the observation results.



(a) Montage, 34 nodes (b) LIGO, 77 nodes

Figure 3: DAG applications used in the evaluation [15].

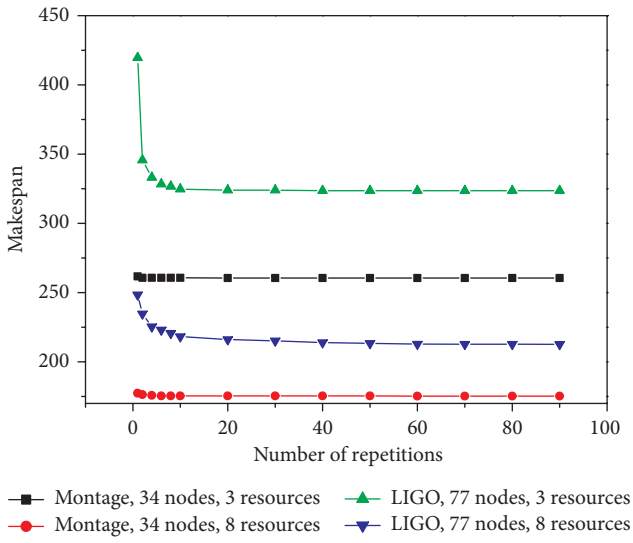


Figure 4: Change of expected makespan as the number of candidates generated by RHEFT increases.

In the diagram shown in Figure 4, RHEFT with zero repetition actually boils down to HEFT. As we can see from the curves denoting "Montage on 3 resources" and "LIGO on 8 resources," the expected makespan of RHEFT reduces rapidly as the number of repetitions grows. After the number of repetitions reaches 10, no significant improvement can be observed on the expected makespan. This observation indicates that RHEFT is promising, as it can generate a candidate schedule which is fairly better than HEFT's with only few more repetitions.

4.3.3. Improvement Rate on Makespan. Next, we observe to what extent can a RHEFT schedule improve a HEFT schedule on the expected makespan. Again, we consider the DAG type to be Montage with 34 nodes and LIGO with 77 nodes and the number of resources to be 3 and 8. Then for each combination of the DAG type and the number of resources, we generate 100 instances of the stochastic model. For each instance, we collect the expected makespans obtained by RHEFT and HEFT, respectively. For comparison, we define the metric "improvement rate," which is the ratio between the reduced expected makespan and the expected makespan of HEFT. Figure 5 shows the results of improvement rate. In general, RHEFT obtains significant improvement on the expected makespan (above 20%) in the case where LIGO is used. Nevertheless, when Montage is used, RHEFT obtains a similar result in the majority of cases of the 100 instances. It can also be seen that the advantage of RHEFT over HEFT may be weakened as the number of resources increases. Anyway, with every setting of DAG type and resource number, there is always a chance that RHEFT can reduce the expected makespan more than 20%.

4.4. The Enhanced RHEFT Heuristic. By making random decision in the resource allocation phase of HEFT, we derive a novel scheduling approach RHEFT which significantly outperforms HEFT on minimizing makespan. This encourages us to extend RHEFT by making random decision when prioritizing the tasks in the listing phase.

The extension is fairly straightforward. In the first phase of the randomized HEFT (Algorithm 2, lines: 1-2), instead of defining task rank by the constant value Urank (as defined in (4)), we consider it as a random variable Rrank. The definition of Rrank is somehow correlated with Urank.

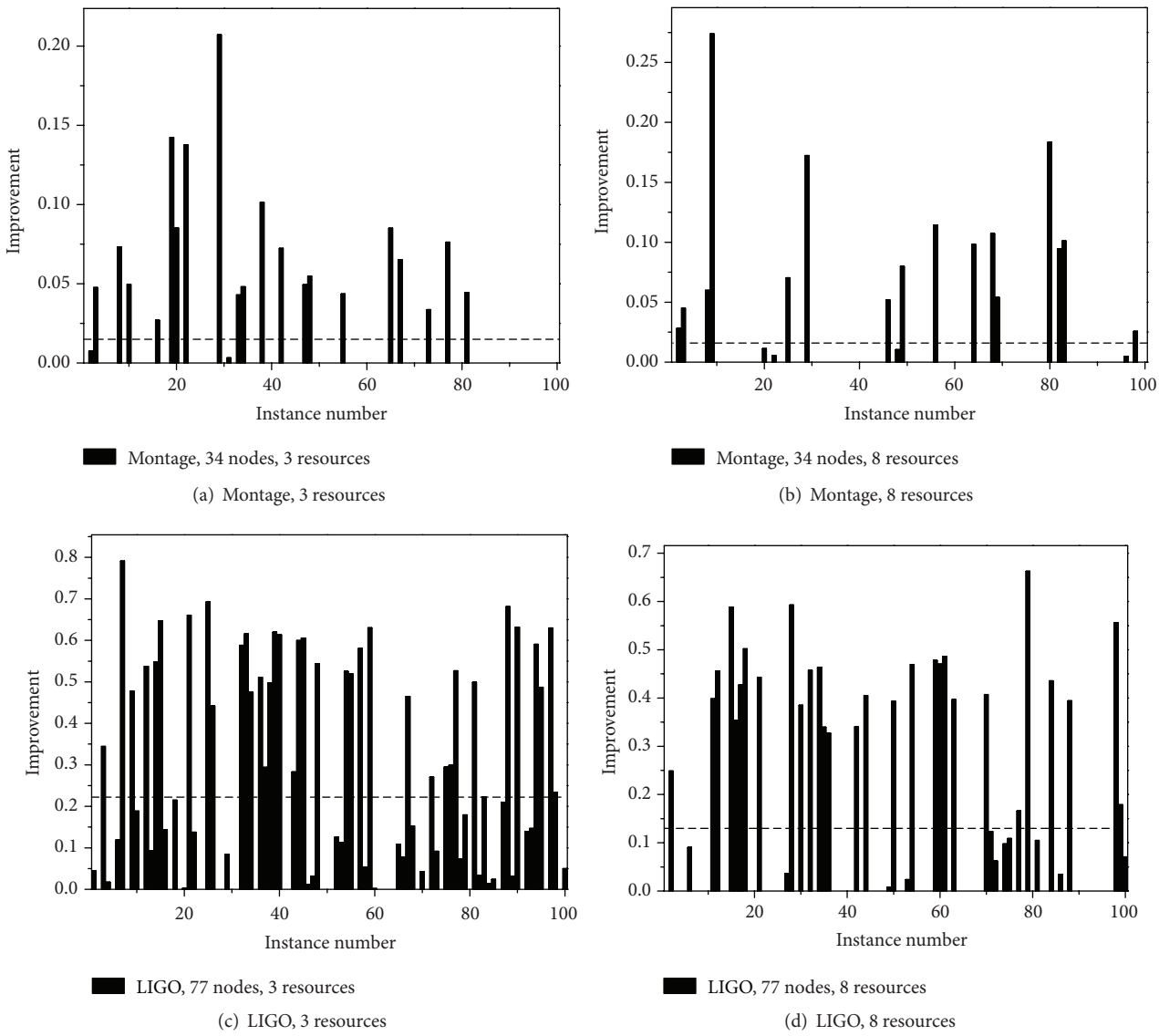


Figure 5: Makespan improvement of RHEFT over HEFT over 100 instances.

Recall that by (4), for task v_i, $\mathrm{Urank}(v_i)$ is calculated by accumulating the time costs associated with the nodes and edges along the critical path from v_i to the exit node. Let Ph_i denote this critical path for v_i, N_i the set of nodes, and E_i the set of edges along with Ph_i; then we have the definition of $\mathrm{Rrank}(v_i)$ as below:

$$\mathrm{Rrank}\left(v_i\right) = \sum_{i \in N_i} \mathrm{MET}_i + \sum_{j \in E_i} \mathrm{MCT}_j, \tag{12}$$

where

$$\mathrm{MET}_i = \frac{\sum_{p=1}^{m} \mathrm{ET}_{i,p}}{m},$$
$$\mathrm{MCT}_j = \frac{\sum_{p=1}^{m} \sum_{q=1}^{m} \mathrm{CT}_{j,p,q}}{m \cdot m}. \tag{13}$$

Especially for exit node, we have $\mathrm{Rrank}(v_{\mathrm{exit\_node}}) = \mathrm{MET}_{\mathrm{exit\_node}}$.

When prioritizing tasks in the listing phase, we need to compare the rank of v_i with v_j. The comparison procedure, which is named *randomized prioritizing procedure*, is carried out as follows:

(1) We firstly examine if there is any task dependency between v_i and v_j.

(2) If v_i is an ancestor of v_j, v_i should be given higher priority and placed before v_j in the list.

(3) If there is no dependency between v_i and v_j, random variables $\mathrm{Rrank}(v_i)$ and $\mathrm{Rrank}(v_j)$ will be compared by the random comparison policy as described in Section 4 to determine which one has the higher priority.

(1) Compute Rrank (as defined in (12)) for all tasks.
(2) Sort all tasks in a list $\mathscr{L}$ in the non-descending order of Rrank according to the *randomized prioritizing procedure*.
(3) **for** $k \coloneqq 1$ to n **do** (where n is the number of tasks)
(4) Select the kth task v^* from the list $\mathscr{L}$.
(5) **for** each resource $r_p \in R$ **do**
(6) Compute the mean value and variance of estimated finish time of v^* on r_p (as defined in (7)).
(7) **endfor**
(8) Decide the winner resource of estimated finish time (r^*) for v^* according to the *random comparison policy*.
(9) Allocate v^* to r^*.
(10) **endfor**

Algorithm 3: The enhanced randomized HEFT heuristic.

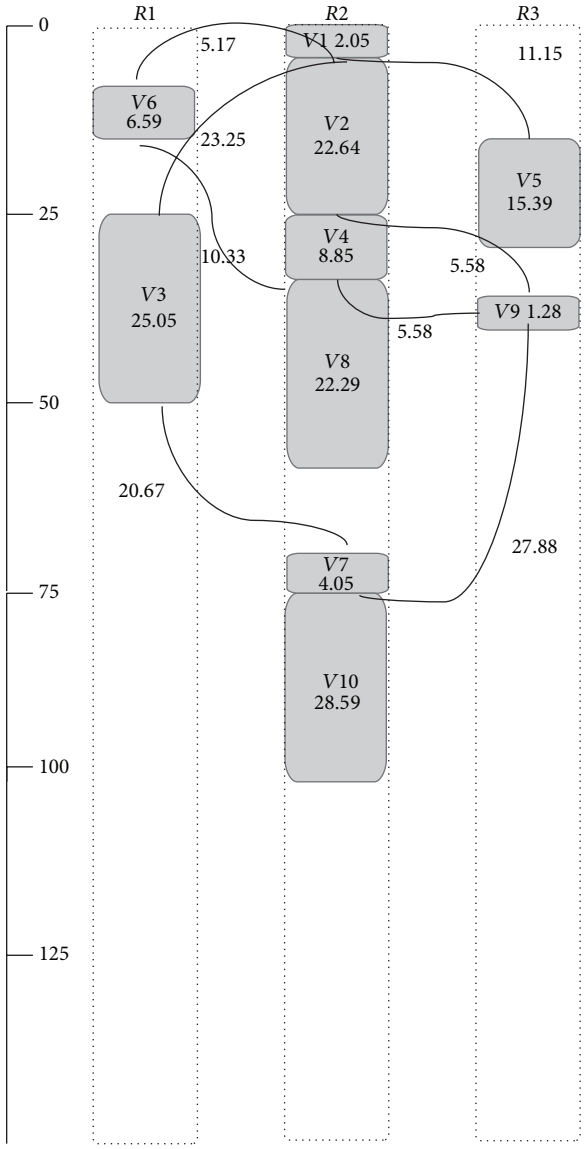


Figure 6: Scheduling result of the DAG example shown in Figure 1 by using ERHEFT.

We apply the above procedure to the listing phase of RHEFT and then derive the enhanced RHEFT heuristic, namely, ERHEFT. The details of the enhanced randomized HEFT heuristic are provided in Algorithm 3.

Apparently, when ERHEFT generates candidate schedules, the prioritized task list may be different from that of RHEFT. This makes it possible for ERHEFT to generate more candidate schedules than RHEFT can do. As a result, a schedule with better makespan may be obtained. For illustration, Figure 6 shows the scheduling result by applying ERHEFT to the DAG example modelled by Figure 1 and Table 1. This schedule has a makespan of 103.67 which is better than the schedule acquired by RHEFT as shown in Figure 2(b).

Similar to the way by which we investigate in Section 4.3.2, we observe how the expected makespan of ERHEFT changes as the number of repetitions used increases. Here, the same evaluation setting as specified in Section 4.3.2 is used and the result is shown in Figure 7. This result indicates that 200 may be the appropriate number of repetitions that should be used by ERHEFT.

5. Evaluation

In order to compare the performance of HEFT, RHEFT, and ERHEFT in stochastic scheduling model, we adopt the evaluation setting as mentioned in Section 4.3.1 and evaluate the expected makespan and the time cost of the competitors with different configurations of evaluation parameters. The machine we used to carry out the evaluation has the following hardware configuration: CPU Intel I3-4130 3.40 GHz, 4 G DDR3 memory, and 500 G hard disk. We used Java (JDK 1.7) to implement all heuristics and the simulation.

First, we evaluate the average makespan that HEFT, RHEFT, and ERHEFT can obtain with stochastic model. For each experiment, we firstly specify the DAG type and the number of resources we are going to use. Then we generate the expected value and variance for stochastically modelling

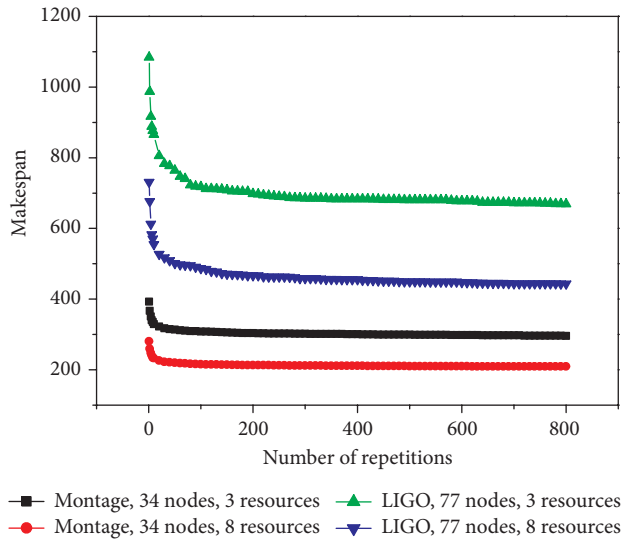


Figure 7: Change of expected makespan as the number of candidates generated by ERHEFT increases.

each task execution time and then generate communication times with specified CCR value, which as a whole is called a stochastic model of the given DAG and resources. For each combination of a DAG and a set of resources, we generate 100 stochastic models. And for each stochastic model, we run HEFT, RHEFT, and ERHEFT and obtain their schedules, respectively. Each time we take a sample for the stochastic model (as shown in Table 2, including all execution times and communication times), we can view them as runtime information gathered after the DAG application completes and use them to evaluate the performance of each schedule. To be fair, we take 100 samples from each stochastic model. So the result for each compared heuristic on a given DAG and given resources is averaged over 10000 experiments (100 stochastic models, each of which has 100 samples) in total. For comparison, we use the metric of "speedup" which is defined as the ratio between the average sequential execution time of all tasks and the makespan obtained by a heuristic.

In order to compare the heuristic competitors in different scenario, we consider the number of resources to be 3, 6, and 8 and collect the average speedup results for CCR being equal to 0.1, 1, and 10, respectively. The collected results are shown in Figure 8.

One can easily see that in all combination of evaluation parameter settings ERHEFT has better average speedup than RHEFT, while RHEFT has better average speedup than HEFT. The improvement on average speedup of ERHEFT over HEFT can be up to around 20% in the case where Montage is executed on 3 resources and CCR = 0.1 is used. It is interesting to see that the effectiveness of ERHEFT and RHEFT is closely related to CCR. When CCR is high, ERHEFT and RHEFT usually achieve more significant improvement on average speedup over HEFT. This indicates our randomization approach may work better with workflow applications which are data intensive. However, when CCR turns to be as low as 0.1, the difference in the average speedup obtained by HEFT, RHEFT, and ERHEFT seems trivial. From a different perspective, this may also imply that HEFT works especially well with computation intensive applications and thus leave little space for further improving its makespan.

In addition, we measure the time cost needed by HEFT, RHEFT, and ERHEFT with different sizes of Montage DAG and different numbers of resources. We use the ratio of the time cost of RHEFT (ERHEFT) over that of HEFT as the metric and the measurement result is shown in Figure 9. From the diagram we can see that in most of the cases the ratio of RHEFT over HEFT is within the range of 5 to 15. Moreover, there is no rapid ascending trend as the DAG size or the number of resources, which represents the scale of the scheduling problem, grows. This indicates that the time complexity of RHEFT is close to HEFT. Because HEFT usually needs very little time to compute a schedule, the additional overhead introduced by a certain number of loops of running HEFT and extra computation of random variables, which results in 5 to 15 times of the time cost of HEFT, is acceptable. The radio of ERHEFT over HEFT exhibits a curve similar to RHEFT over HEFT. Even though the radio of ERHEFT over HEFT reaches the range of 100 to 500, as the time cost for a single execution of HEFT is tiny, the overall time cost for ERHEFT is still acceptable. For instance, in our empirical results, for DAG with 234 nodes ERHEFT runs for 3.2 seconds while for 12 resources ERHEFT runs for only 1.9 seconds. Moreover, by adjusting the number of loops used in the RHEFT approach, we can flexibly get a good trade-off between the scheduling performance and the heuristic overhead.

6. Conclusion

In this paper, we explore into the problem of scheduling workflow tasks onto a set of heterogeneous cloud resources with stochastic model of task execution and communication. We attempt to extend deterministic DAG scheduling heuristic, to gain better average makespan. As a progress, a randomization scheduling approach is proposed. We apply the randomization approach to the classic deterministic heuristic HEFT and two versions of novel randomized heuristic: RHEFT and ERHEFT, are produced. We evaluate and compare the performance of HEFT, RHEFT, and ERHEFT with extensive simulation experiments where two real-world workflow applications are used. The experimental results suggest that RHEFT and ERHEFT are both promising for stochastic workflow scheduling, as RHEFT and ERHEFT not only significantly reduce the average makespan in most cases of experimental setting, but also exhibit reasonable scalability. Our future work may consider more stochastic model other than normal distribution and/or randomizing other deterministic DAG scheduling heuristics.

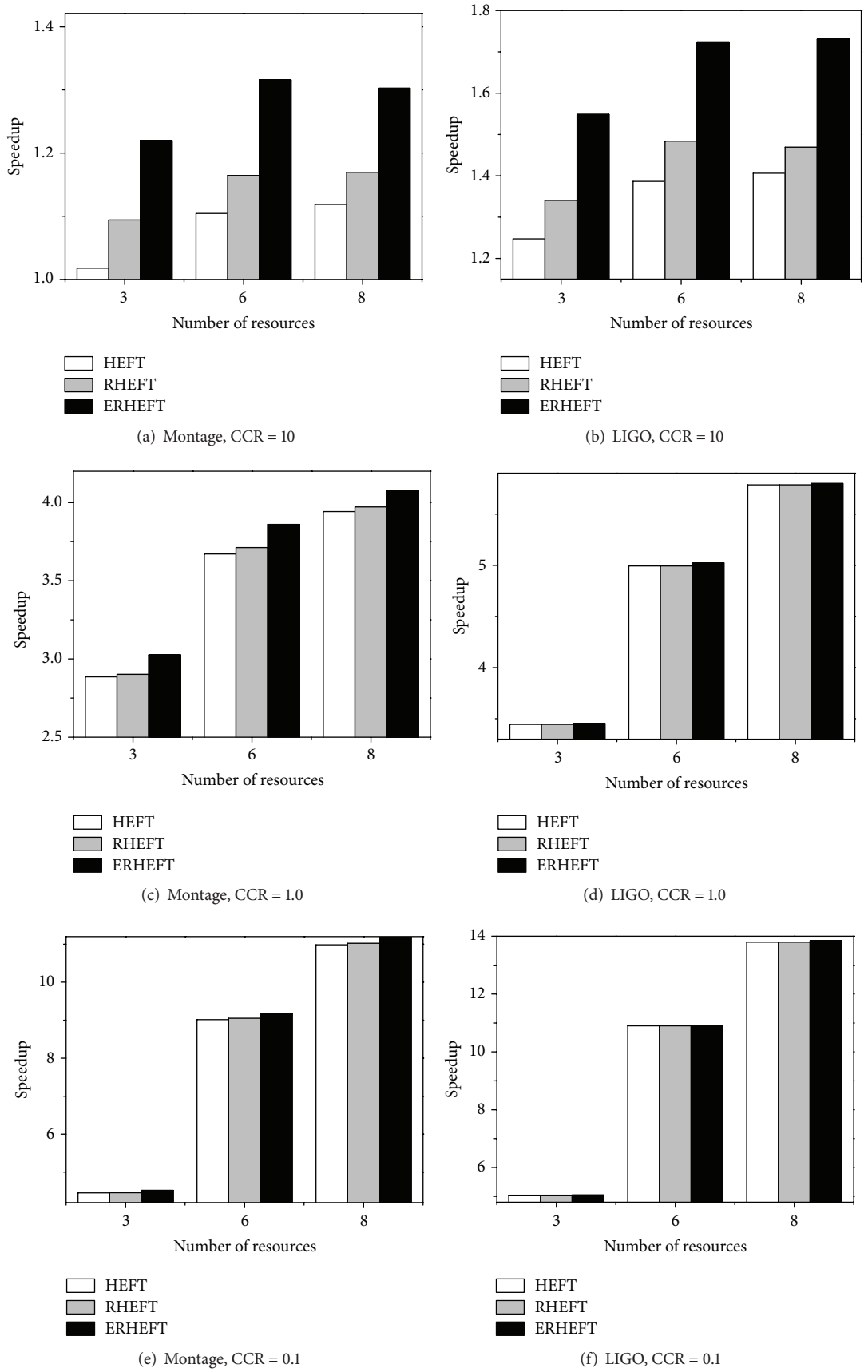


Figure 8: Average speedup results with different CCR values.

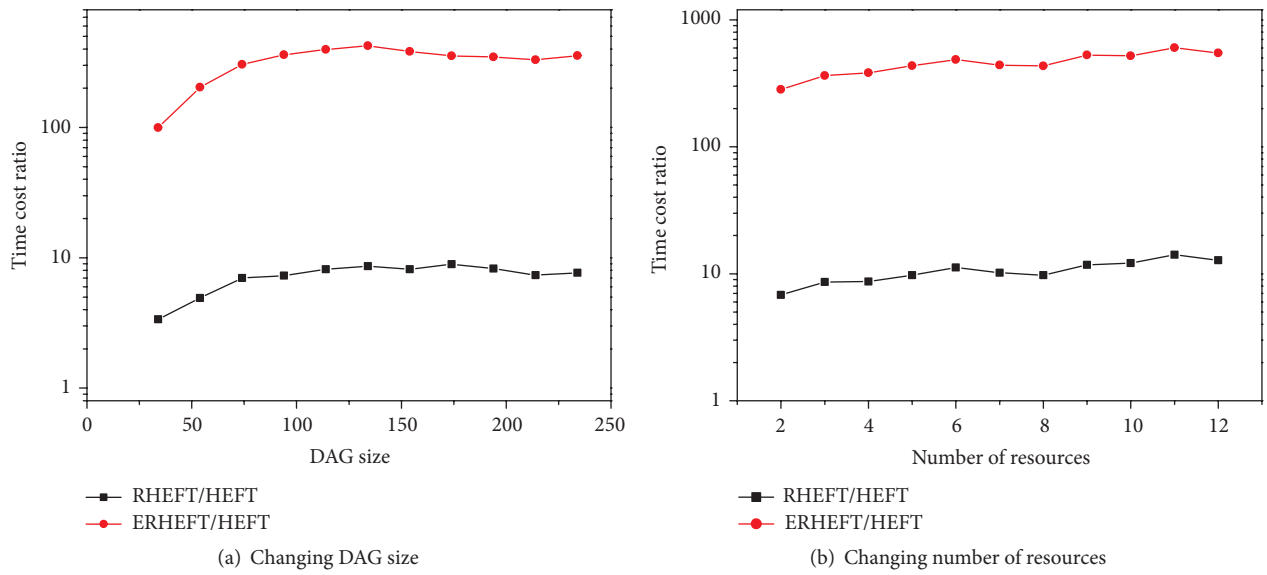


Figure 9: Time cost ratio of RHEFT and ERHEFT over HEFT.

Competing Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

The work is supported by National Natural Science Foundation of China (NSFC, Grant no. 61202361).

References

[1] E. Deelman, D. Gannon, M. Shields, and I. Taylor, "Workflows and e-science: an overview of workflow system features and capabilities," *Future Generation Computer Systems*, vol. 25, no. 5, pp. 528–540, 2009.

[2] G. B. Berriman, J. C. Good, A. C. Laity et al., "A grid enabled image mosaic service for the national virtual observatory," in *Proceedings of the Conference Series of Astronomical Data Analysis Software and Systems XIII (ADASS XIII)*, pp. 593–596, 2004.

[3] I. J. Taylor, E. Deelman, D. B. Gannon, and M. Shields, *Workflows for E-Science: Scientific Workflows for Grids*, Springer, New York, NY, USA, 2007.

[4] H. Casanova, F. Dufossé, Y. Robert, and F. Vivien, "Scheduling parallel iterative applications on volatile resources," in *Proceedings of the IEEE International Parallel & Distributed Processing Symposium (IPDPS '11)*, pp. 1012–1023, IEEE, Anchorage, Alaska, USA, May 2011.

[5] S. Yeo and H. S. Lee, "Using mathematical modeling in provisioning a heterogeneous cloud computing environment," *IEEE Computer*, vol. 44, no. 8, pp. 55–62, 2011.

[6] G. Juve and E. Deelman, "Scientific workflows and clouds," *ACM Crossroads*, vol. 16, no. 3, pp. 14–18, 2010.

[7] G. Juve, E. Deelman, G. B. Berriman, B. P. Berman, and P. Maechling, "An evaluation of the cost and performance of scientific workflows on Amazon EC2," *Journal of Grid Computing*, vol. 10, no. 1, pp. 5–21, 2012.

[8] J. Li, D. Li, Y. Ye, and X. Lu, "Efficient multi-tenant virtual machine allocation in cloud data centers," *Tsinghua Science and Technology*, vol. 20, no. 1, pp. 81–89, 2015.

[9] C. Cheng, J. Li, and Y. Wang, "An energy-saving task scheduling strategy based on vacation queuing theory in cloud computing," *Tsinghua Science and Technology*, vol. 20, no. 1, pp. 28–39, 2015.

[10] M. R. Garey and D. S. Johnson, *Computer and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, 1979.

[11] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002.

[12] R. Sakellariou and H. Zhao, "A hybrid heuristic for DAG scheduling on heterogeneous systems," in *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS '04)*, IEEE Computer Society, Santa Fe , NM, USA, April 2004.

[13] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. Rinnooy Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," *Annals of Discrete Mathematics*, vol. 5, pp. 287–326, 1979.

[14] M. L. Pinedo, *Overview of Stochastic Scheduling Problems*, Springer, Berlin, Germany, 2011.

[15] W. Zheng and R. Sakellariou, "Stochastic DAG scheduling using a Monte Carlo approach," *Journal of Parallel and Distributed Computing*, vol. 73, no. 12, pp. 1673–1689, 2013.

[16] R. Sakellariou and H. Zhao, "A hybrid heuristic for DAG scheduling on heterogeneous systems," in *Proceedings of the 13th Heterogeneous Computing Workshop*, pp. 111–124, 2004.

[17] G. C. Sih and E. A. Lee, "A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 2, pp. 175–187, 1993.

[18] J. Blythe, S. Jain, E. Deelman et al., "Task scheduling strategies for workflow-based applications in grids," in *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid '05)*, vol. 2, pp. 759–767, May 2005.

[19] G. Q. Liu, K. L. Poh, and M. Xie, "Iterative list scheduling for heterogeneous computing," *Journal of Parallel and Distributed Computing*, vol. 65, no. 5, pp. 654–664, 2005.

[20] L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. MacIejewski, "Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach," *Journal of Parallel and Distributed Computing*, vol. 47, no. 1, pp. 8–22, 1997.

[21] M. Coli and P. Palazzari, "Real time pipelined system design through simulated annealing," *Journal of Systems Architecture*, vol. 42, no. 6-7, pp. 465–475, 1996.

[22] B. Cirou and E. Jeannot, "Triplet: a clustering scheduling algorithm for heterogeneous systems," in *Proceedings of the International Conference on Parallel Processing Workshops*, pp. 231–236, Valencia, Spain, 2001.

[23] S. Ranaweera and D. P. Agrawal, "A task duplication based scheduling algorithm for heterogeneous systems," in *Proceedings of the 14th International Parallel and Distributed Processing Symposium*, pp. 445–450, Cancun, Mexico, May 2000.

[24] S. Ranaweera and D. P. Agrawal, "A scalable task duplication based scheduling algorithm for heterogeneous systems," in *Proceedings of the International Conference on Parallel Processing*, pp. 383–390, Toronto, Canada, 2000.

[25] A. Dogan and R. Ozguner, "LDBS: a duplication based scheduling algorithm for heterogeneous computing systems," in *Proceedings of the International Conference on Parallel Processing (ICPP '02)*, pp. 352–359, IEEE, 2002.

[26] L. Canon, E. Jeannot, R. Sakellariou, and W. Zheng, "Comparative evaluation of the robustness of DAG scheduling heuristics," in *Grid Computing: Achievements and Prospects*, S. Gorlatch, P. Fragopoulou, and T. Priol, Eds., pp. 73–84, Springer, Berlin, Germany, 2008.

[27] S. A. Jarvis, L. He, D. P. Spooner, and G. R. Nudd, "The impact of predictive inaccuracies on execution scheduling," *Performance Evaluation*, vol. 60, no. 1–4, pp. 127–139, 2005.

[28] M. M. López, E. Heymann, and M. A. Senar, "Analysis of dynamic heuristics for workflow scheduling on grid systems," in *Proceedings of the 5th International Symposium on Parallel and Distributed Computing*, pp. 199–207, IEEE, Timisoara, Romania, July 2006.

[29] A. Kamthe and S.-Y. Lee, "A stochastic approach to estimating earliest start times of nodes for scheduling DAGs on heterogeneous distributed computing systems," in *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS '05)*, p. 121b, Denver, Colo, USA, April 2005.

[30] X. Tang, K. Li, G. Liao, K. Fang, and F. Wu, "A stochastic scheduling algorithm for precedence constrained tasks on grid," *Future Generation Computer Systems*, vol. 27, no. 8, pp. 1083–1091, 2011.

[31] K. Li, X. Tang, B. Veeravalli, and K. Li, "Scheduling precedence constrained stochastic tasks on heterogeneous cluster systems," *IEEE Transactions on Computers*, vol. 64, no. 1, pp. 191–204, 2015.

[32] W. Zheng, B. Emmanuel, and C. Wang, "A randomized heuristic for stochastic workflow scheduling on heterogeneous systems," in *Proceedings of the 3rd International Conference on Advanced Cloud and Big Data (CBD '15)*, pp. 88–95, Yangzhou, China, October 2015.

[33] E. Deelman, C. Kesselman, G. Mehta et al., "GriPhyN and LIGO, building a virtual data Grid for gravitational wave scientists," in *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC '02)*, pp. 225–234, IEEE, 2002.