Hindawi
Complexity
Volume 2020, Article ID 1467274, 15 pages
https://doi.org/10.1155/2020/1467274

Research Article

Cost-Effective Resource Provisioning for Real-Time Workflow in Cloud

Lei Wu, Ran Ding, Zhaohong Jia, and Xuejun Li

School of Computer Science and Technology, Anhui University, Hefei, China

Correspondence should be addressed to Xuejun Li; xjli@ahu.edu.cn

Received 21 January 2020; Revised 18 February 2020; Accepted 25 February 2020; Published 30 March 2020

Guest Editor: Xuyun Zhang

In the era of big data, mining and analysis of the enormous amount of data has been widely used to support decision-making. This complex process including huge-volume data collecting, storage, transmission, and analysis could be modeled as workflow. Meanwhile, cloud environment provides sufficient computing and storage resources for big data management and analytics. Due to the clouds providing the pay-as-you-go pricing scheme, executing a workflow in clouds should pay for the provisioned resources. Thus, cost-effective resource provisioning for workflow in clouds is still a critical challenge. Also, the responses of the complex data management process are usually required to be real-time. Therefore, deadline is the most crucial constraint for workflow execution. In order to address the challenge of cost-effective resource provisioning while meeting the real-time requirements of workflow execution, a resource provisioning strategy based on dynamic programming is proposed to achieve cost-effectiveness of workflow execution in clouds and a critical-path based workflow partition algorithm is presented to guarantee that the workflow can be completed before deadline. Our approach is evaluated by simulation experiments with real-time workflows of different sizes and different structures. The results demonstrate that our algorithm outperforms the existing classical algorithms.

1. Introduction

Nowadays, the big data technology has been used in a wide range of applications including complex systems to support decision-making [1, 2]. Along with the enormous commercial benefits, scientific advances, management efficiency, and analytical accuracy brought by big data, this new technology raises many challenging problems such as high cost and latency of big data storage, transmission, and processing [3–5]. To tackle these problems, cloud computing environment and workflow modeling methods are recognized as the effective way.

Many large-scale scientific applications in areas such as astronomy, bioinformatics, and meteorology would generate and process large amounts of data; such applications consist of a large number of data processing tasks that are frequently modeled as workflows [6]. Normally, a workflow is represented by a Directed Acyclic Graph (DAG) with nodes and edges, where nodes represent tasks and edges represent data/control dependencies between tasks in a complex application system. Meanwhile, once a time-critical or real-time application system [7, 8] is modeled as a real-time workflow, the deadline constraint can be used to ensure the tasks complete on time effectively.

Cloud computing is being investigated as an effective platform that delivers hardware infrastructure and software applications as services for the tasks of big data management and analytics. And real-time workflows can also be executed in such high performance computing environment. There are various cloud providers offering large amount of services with different quality of service (QoS) [9–11] as well as different prices. A lot of efforts have been made in the area of service recommendation [12, 13] based on QoS from the perspective of service providers [14–16], but few concerns have focused on how to select the services in cloud platform based on QoS from the perspective of users. Particularly, cloud computing provides a flexible pricing model (i.e., pay-as-you-go and on-demands services); users are charged based on their consumption of various resources with different QoS. Therefore, one of the most challenging problems

with real-time workflows in cloud computing is to get a cost-effective way to complete the workflow within the deadline [17].

In reality, there are two main stages when executing a workflow in a cloud computing environment [18]. The first one is the resource provisioning stage: during this phase, computing resources from the clouds will be selected and reserved to prepare for the workflow's execution. The second one is the task scheduling stage: during this phase, a schedule is generated and each task will be mapped into the best-suited resource. That is, the second stage determines where and when each task of a workflow will be executed, while the first stage decides what types and how many resources will be leased from the cloud service providers and therefore the total cost of the workflow is mainly decided at this stage. To address these distinctions between task scheduling and resource provisioning, we propose in this paper a novel cost optimization algorithm that focuses only on resource provisioning.

In order to satisfy the deadline of real-time workflow, our proposal partitions the original workflow into some sub-workflows which can be executed in parallel based on the critical path methodology. Then we use the dynamic programming knapsack algorithm to provision resources in clouds for these sub-workflows to minimize the total cost. Our approach is evaluated by simulation experiments with real-time workflows of different sizes and different structures. The results demonstrate that our algorithm outperforms the existing classical algorithms. Major contributions of this paper are stated as follows:

(i) A global resource provisioning for real-time workflow strategy is addressed for cost optimization under deadline-constrained.

(ii) A workflow partition algorithm based on critical path technique is proposed to get some sub-workflows being executed in parallel to ensure the deadline constraint.

(iii) A dynamic programming algorithm is used to provision the most cost-effective resources to the sub-workflows.

(iv) We perform extensive simulations and show the efficacy of our strategy over two existing algorithms, namely, simply DPK and IC-PCP.

The rest of the paper is organized as follows. Section 2 introduces related work followed by problem specification in Section 3. Section 4 explains the proposed algorithm while Section 5 presents the evaluation of the algorithm performance. Finally, conclusions and future work are summarized in Section 6.

2. Related Work

Cost optimization for workflow execution has been widely studied over the years. Efficient resource utilization in parallel and distributed computing environment is a key issue for cost-effectiveness. To resolve this issue, numerous researches have been done by a variety of workflow scheduling methods in clouds. Accordingly, a significant number of real-time workflow scheduling algorithms focusing on reducing the overall execution cost of real-time workflow have been proposed. Alkhanak et al. [19] classified the cost optimization method of workflow in cloud computing environment into two categories: heuristic methods and meta-heuristic methods.

Abrishami et al. [17] presented a static algorithm IC-PCP (IaaS Clouds-Partial Critical Path) based on the heuristic for scheduling a single workflow instance on an IaaS cloud. This algorithm considers cloud features such as VM heterogeneity, pay-as-you-go, and time interval pricing. They try to minimize the execution cost of scheduling all tasks in a partial critical path on a single machine that can finish the tasks before their latest finishing time (which is calculated based on the application's deadline and the fastest available instance). In [20], Verma and Kaushal proposed a greedy algorithm based on the classical HEFT for providing a suitable trade-off between execution cost and time. Zheng et al. [21] presented three novel scheduling heuristic algorithms to help users to schedule their big data processing workflow application on clouds so that the cost can be minimized and the deadline constraints can be satisfied; different configurations of CPU frequency were considered in their work. Meta-heuristic approaches such as Genetic Algorithm (GA) based [22], Ant Colony Optimization (ACO) based [23], Particle Swarm Optimization (PSO) based [24], and symbiotic organism search algorithms were used to address the same objectives that minimize the cost of workflow execution while considering the deadline. Wu et al. [25] proposed a meta-heuristic algorithm L-ACO as well as a simple heuristic ProLiS to minimize execution cost of a workflow in clouds under a deadline constraint; experimental results show that the meta-heuristic L-ACO performs better in terms of execution costs and success ratios of meeting deadlines but the heuristic ProLiS is more efficient. That is to say, the meta-heuristic based algorithm has achieved improved performance on cost optimization but with some compromise on the execution time.

Task scheduling assigns tasks to the most cost-efficient resource to optimize the execution cost of a workflow. But task scheduling is a well-known NP-hard problem. There is no scheduling algorithm that can obtain an optimal solution in polynomial time [17]. Otherwise, to tackle deadline-constrained workflow scheduling, deadline-distribution method is the most widely used method by both meta-heuristic based and heuristic based algorithms. The deadline-distribution method always distributes the deadline to each task in proportion to its minimum execution time [25]. In most case, this kind of methods ignores the tasks of workflow that can be executed in parallel, so the sub-deadline of each single task may be not appropriate. According to these sub-deadlines, task scheduling algorithms cannot get a global optimal result, because a task level optimization was used and hence failed to utilize the whole workflow's structure and characteristics. Resource provisioning can get a minimum cost by selecting an optimal assembly of resources for the global workflow execution. Our proposal minimized the total cost of workflow

execution from the perspective of resource provisioning to the whole workflow (or sub-workflows) without concerning each single task's resource mapping, which can further improve resource utilization.

With respect to cost optimization, resource provisioning is more efficient and effective than task scheduling for a real-time workflow in cloud computing environment [26]. Many researches have been done to minimize the overall cost of workflow by resource provisioning [27, 28]. Both static and dynamic methods are used to provision the cloud resources [29]. Static method assumes that accurate information about workflow and cloud resource performance can be obtained before scheduling. Dynamic provisioning adopts no such assumption. Most scientists run the same workflows, thereby enabling the collection of such information through several trial runs. Therefore static method is an appropriate method for workflows [30]. Static scheduling is also eased by the fact that cloud providers usually declare the performance specifications of their resources. We use static method for resource provisioning in this paper.

3. Problem Specification

In order to facilitate the reading of the paper, all of the symbols used in this section are listed in Table 1.

3.1. Related Model

3.1.1. Workflow Model. A real-time workflow refers to using the workflow technology for modeling of a real-time application system. That is, temporal constraints are added to the original workflow model using DAG graphs.

Definition 1. A workflow application is represented by a Directed Acyclic Graph (DAG) $G = (T, W, E)$, where $T = \{t_1, t_2, \ldots, t_n\}$ is a set of n tasks, $W = \{w_1, w_2, \ldots, w_n\}$ is the computational workload of each task in T, and $E = \{e_{ij} \mid 0 \le i \ne j \le n\}$ is the set of directed edges between two tasks. An edge e_{ij} of the form (t_i, t_j) denotes that there is data dependency between t_i and t_j; t_i is said to be the parent task of t_j and t_j is said to be the child task of t_i. This relationship between t_i and t_j can be represented by $t_i = \text{Parent}(t_j)$ and $t_j = \text{Child}(t_i)$. The computational workload in W is given by the number of mega-floating point operations that need to be executed. The temporal constraint for a real-time workflow is the deadline which is denoted by D.Based on this definition, a child task cannot be executed until all of its parent tasks are completed. Figure 1 shows an example workflow with six tasks. t_{entry} and t_{exit} are additional dummy nodes with computational workload 0 to give the whole workflow a single entry and a single exit. The task t_1 must be finished before tasks t_3 and t_4 can start. Also, both t_3 and t_4 must be finished before t_5 can start. However, there is no path between t_3 and t_4. So t_3 and t_4 do not need to be executed in any particular sequence. Therefore, the tasks $\{t_1, t_3, t_4, t_5\}$ can be executed as $t_1 \longrightarrow t_3(t_4) \longrightarrow t_5$. This means that the workflow segment t_1, t_3, t_5 or t_1, t_4, t_5 must be executed sequentially, but t_3, t_4 can be executed in parallel. There are two classic types of workflow processes in a large real-time workflow as shown in Figure 2. One is sequential workflow process Figure 2(a) in which all tasks must be executed one after the other in a certain order. The other is parallel workflow process Figure 2(b) in which all tasks can be executed simultaneously without any particular order.

3.1.2. Resource Model. This paper focuses on Infrastructure as a Service (IaaS) clouds which offer the user a virtual machine (VM) pool of unlimited and heterogeneous resources with different computational performance, memory capacity, and price that can be accessed on demand such as Amazon's EC2 (Elastic Compute Cloud) [31]. Better computational performance or more memory implies higher price. We assume that there is no limitation on using each resource; i.e., the workflow can order any number of resources from each cloud provider at any time. Based on the profiling results about workflows given in [1] and the VM types offered by Amazon EC2, we assume that the VMs have sufficient memory to execute the workflow tasks. So, memory capacity of VMs will no longer be considered in this paper. We define VM_j as a VM type in terms of its computational performance P_j and PRICE_j per billing cycle of τ time units.

For each VM_j of a certain type, we assume that the computational performance in terms of mega-floating point operations per second (MFLOPS) either is available from the provider or can be estimated [32]. This information is used to deduce the computational performance by $P_j = \text{MFLOPS}_j * \tau$, where τ denotes the billing cycle of VM_j which is specified by the cloud provider. User's payment for the usage of each VM_j is based on the billing cycle τ. Any partial utilization of the leased VM is charged as if the full billing cycle was consumed. For example, if $\tau = 60$ seconds and a VM is used for 61 seconds, then the user will pay for two cycles of 60 seconds, that is, 120 seconds. Also, we assume that there is no limitation on the number of billing cycles of VMs that can be leased from the provider.

Definition 2. Assume there are m types of VMs that can be provided by cloud providers; the collection of available VMs will be denoted as $\text{VMs} = \{\text{VM}_j \mid 1 \le j \le m\}$. Each VM_j has two parameters; one is MFLOPS_j and the other is PRICE_j. MFLOPS_j is defined as mega-floating point operations per second of the VM_j, and PRICE_j is the price per τ seconds of the VM_j, where τ is the billing cycle of the VM_j. $P_j = \text{MFLOPS}_j \times \tau$ is the computational performance of VM_j per billing cycle.

3.1.3. Cost Model. Normally, the final cost is based not only on the utilization of computational resources, but also on the data transfer between the parent task and its child task. If both parent and child tasks are in the same VM, there is no data transfer fee because the tasks share the same data center in a single VM. When the parent task and the child task are executed on different VMs that belong to the same cloud provider, there will not be any data transfer fee because most of the public cloud providers, such as Amazon EC2, do not charge for internal data transfers among their computational

Table 1: List of symbols.

Symbol	Meaning
$G(T, W, E)$	A workflow represented by DAG
$T\{t_1, t_2, \ldots, t_n\}$	A set of n tasks
$W\{w_1, w_2, \ldots, w_n\}$	The related workloads of n tasks in T
$E\{e_{ij}\}$	A set of directed edges between two tasks in G
D	The deadline of workflow G
Parent(t_i)	Parent task of t_i
Child(t_i)	Child task of t_i
TC(G)	Total cost of workflow G
TW(G)	Total workload of workflow G
ET(G)	Total execution time of workflow G
ET_{ij}	Execution time of task t_i on VM_j
RT	Resource occupation time
BT	Resource initial booting time
C_i	Cost of task t_i
TT	Data transfer time
BL(d)	A set of tasks in layer d
BD(t_i)	The longest path of t_i to t_{exit}
LMET(d)	The minimum execution time of layer d
CSW	Sub-workflow containing all of critical tasks
NSW	Sub-workflow not containing any of critical tasks
$VMs\{VM_1, \ldots, VM_n\}$	A set of m types of VMs
$VM_j(MFLOPS_j, PRICE_j)$	A type of VM
$MFLOPS_j$	Mega-floating point operations per second of VM_j
$PRICE_j$	Price of VM_j
τ	Billing cycle defined by VMS provider
P_j	Computational performance of VM_j
C_j	Cost of resource VM_j
$X\{x_1, x_2, \ldots, x_m\}$	VMs vector of provisioning results
x_j	Number of billing cycles of VM_j
$f_k(W)$	First k types of VMs satisfying the workload W

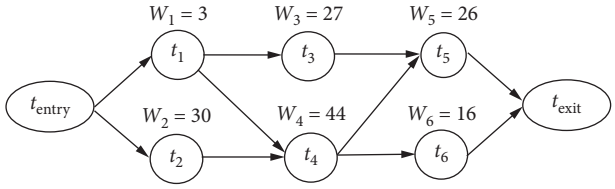

Figure 1: Sample workflow.

resources. So, when we use these public cloud computing resources, we can ignore the data transfer fee at this moment. For the rest of this paper, we will assume that we utilize computing resources from one cloud provider only.

The cost of using the computational resources is calculated from the resource occupation time multiplied by the price of the resource. The resource occupation time, denoted by RT, concerns not only the tasks execution time (ET), but also the data transfer time (TT) and the initial boot time (BT) of each resource provisioned by the workflow. Thus, RT = BT + TT + ET. However, as the development of Docker technology, the initial boot speed of virtual resources is of the order of seconds or milliseconds. It is so small compared with the execution time of a workflow that it can be ignored. That is, we can assume that BT = 0. When the data transfer occurs between a parent task t_i and its child task t_j, the transfer time depends only on the amount of data to be transfered and the bandwidth provided by the cloud provider. Both of these values are fixed, regardless of the type of VMs that is provisioned by the workflow. Therefore, the value of data transfer time of each task t_i can be denoted by TT_i, which is VM-independent. For a VM_j, the resource occupation time can be denoted by $RT_j = \sum_{i=1}^{n} ET_{ij} + TT_i$, where ET_{ij} is the execution time of task t_i $(1 \le i \le n)$ on VM_j and TT_i is the data transfer time of t_i which is VM-independent. Each task t_i $(1 \le i \le n)$ may or may not be executed by VM_j $(1 \le i \le m)$. Let a_{ij} be a 0-1 variable that indicate whether t_i is executed on VM_j; when t_i is executed by VM_j, $a_{ij} = 1$, otherwise, $a_{ij} = 0$. So the cost of each VM_j can be denoted by $C_j = RT_j \times PRICE_j = \sum_{i=1}^{n} (ET_{ij} + TT_i) \times a_{ij} \times PRICE_j$. In order to optimize the total cost, we need to minimize each C_j. From the formula of C_j, we see that ET_{ij} and a_{ij} are the two variables that depend on both t_i and VM_j, while TT_i depends only on t_i but not VM_j. Thus, TT_i can be regarded as a constant from the cost optimization point of view. For the rest of the paper, we can ignore TT_i.

Actually, a task does not have to be executed entirely by one VM. It can execute sequentially on several VMs as long as the execution coincides with a billing cycle. For example, suppose t_i has executed on a VM for one billing cycle and it has not yet finished; then it has the option of staying with the same VM in the next billing cycle or switching to a different VM. Let ET_i denote the execution time of task t_i. Then, t_i will be charged $\lceil ET_i \rceil = \sum_{j=1}^{m} x_{ij}$ $(x_{ij} \ge 0$ and x_{ij} is an integer) billing cycles, where the ceiling of ET_i is due to the last billing cycle of VM_j (assuming that VM_j is the last VM provisioned by t_i). Therefore, the cost of resource provisioning to task t_i can be denoted as $C_i = \lceil ET_i \rceil \times PRICE$, where PRICE is the price vector $PRICE = (PRICE_1, PRICE_2, \ldots, PRICE_m)^T$. So, $C_i = \sum_{j=1}^{m} x_{ij} \times PRICE_j$. The total cost of a workflow can be defined as follows:

Definition 3. Assume there is a workflow $G = (T, W, E)$ with n tasks as depicted in Definition 1, while there are m $VMs = \{VM_1, VM_2, \ldots, VM_m\}$ as defined in Definition 2 that can be provisioned to workflow G. The total cost of resource provisioning to the workflow G *is* $TC(G) = \sum_{i=1}^{n} C_i$, where $C_i = \sum_{j=1}^{n} x_{ij} \times PRICE_j$, x_{ij} is the number of billing cycles of VM_j which provision to task t_i. So, $TC(G) = \sum_{i=1}^{n} \sum_{j=1}^{m} x_{ij} \times PRICE_j = \sum_{j=1}^{m} (\sum_{i=1}^{n} x_{ij} \times PRICE_j)$. Letting $x_j = \sum_{i=1}^{n} x_{ij}$, we have $TC(G) = \sum_{j=1}^{m} x_j \times PRICE_j$.

Note that, in the above definition, we have assumed that no two tasks are combined together. For example, suppose there are two tasks t_1 and t_2 each requiring execution of 1.5 billing cycles. If we combine the two tasks together, it only requires 3.0 billing cycles. However, in the above definition, they will require a total of 4 billing cycles, two billing cycles for each task. We will discuss how to combine two tasks together in later sections.

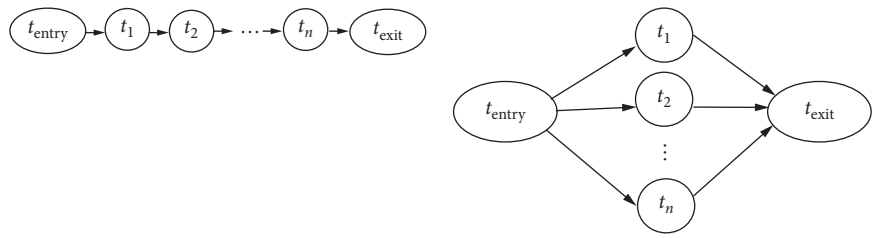

Figure 2: The structure of workflow. (a) Sequencial workflow process. (b) Paralleled workflow process.

3.2. Cost-Effective Resource Provisioning

3.2.1. Optimization Objective. Resource provisioning in cloud computing environment may have different objectives. This work focuses on finding a provisioning strategy to execute a workflow on IaaS clouds such that the total execution cost is minimized and the deadline is met. We assume that there are m types of VMs; the computational performance and price per billing cycle of each VM_j ($j = 1, 2, \ldots, m$) are defined as in Definition 2. According to Definition 3, the objective of cost optimization is to find a resource provisioning $X = \{x_j \mid x_j$ is the number of billing cycles of VM_j ($j = 1, 2, \ldots, m$)$\}$ to satisfy the following:

$$\begin{aligned} \text{minmize} \quad & \mathrm{TC}(G) = \sum_{j=1}^{m} \mathrm{PRICE}_j \times x_j \\ \text{s.t.} \quad & \sum_{j=1}^{m} P_j * x_j \geq \mathrm{TW} \\ & x_j \geq 0 \text{ and } x_j \text{ is an integer}, \quad j = 1, 2, \ldots, m \\ & \mathrm{ET}(G) \leq D, \end{aligned} \tag{1}$$

where $\mathrm{TW} = \sum_{i=1}^{n} w_i$ is the sum of all tasks' workload, x_j is the number of billing cycles used by VM_j for all the tasks, D is the deadline of the workflow G as defined in Definition 1, and $\mathrm{ET}(G)$ is the total execution time of the workflow G. It is generally known that $\mathrm{ET}(G)$ is not only affected by each task's execution time but also related to the execution sequence for all tasks in a workflow. That is, the execution time of a sequencial workflow as shown in Figure 2(a) will be the sum of each task's execution time while a parallel workflow as shown in Figure 2(b) will take the longest running task's run time as its execution time. So the execution time of a workflow is defined as follows:

Definition 4. Assume there is a workflow $G = (T, W, E)$ with n tasks as depicted in Definition 1. If the workflow is a sequential process, the execution time can be expressed as $\mathrm{ET}(G) = \sum_{t_i \in G} \mathrm{ET}_i$. If the workflow is a parallel process, the execution time can be expressed as $\mathrm{ET}(G) = \mathrm{MAX}_{t_i \in G} \mathrm{ET}_i$. ET_i in both formulas denote the execution time of each task in the workflow. So, if the workflow contains the same tasks the execution time of parallel workflow must be less than sequential workflow.

Shown in Table 2 is an example of three types of VMs with computational performance P_j and price PRICE_j per billing cycle. In this example, we assume the billing cycle is one hour.

Table 2: Sample virtual machines.

VM types	P_j	PRICE_j
VM_1	30	0.8
VM_2	14	0.4
VM_3	3	0.1

When we provision these three types of VMs to the tasks in the sample workflow shown in Figure 1, the cost of each task executing on each VM is shown in Table 3. As can be seen from this table, when $w_1 = 3$, provisioning VM_3 would take 1 hour and is charged 0.1, while provisioning VM_1 or VM_2 would take less than 1 hour but is charged 0.8 or 0.4. So, the optimal provisioning for t_1 is one VM_3 and with a charge of 0.1. In Table 3, the optimal single VM provisioning of each task is given in bold. Notice that for t_1, t_2, t_3, and t_5, the optimal provisioning for each of them is to use one type of VM. For t_4, the optimal provisioning is to use one VM_1 and one VM_2 for a total charge of 1.2. For t_6, the optimal provisioning is to use one VM_2 and one VM_3 for a total charge of 0.5.

The total workload of all the tasks in the workflow shown in Figure 1 is 146. The optimal provisioning is to use five VM_1 for a total charge of 4.0. Note that this is less than the sum of each task's optimal provisioning which is 4.2. This is because each individual optimal provisioning might have idle time. For example, for t_5, we use one VM_1 which can execute 30 MFLOPS in one hour, but t_5 has a workload of only 26 MFLOPS. So, 4 MFLOPS are wasted. The columns CSW and NSW will be explained later.

3.2.2. Cost Optimization Stragety. W is assumed to be the computational workload that need to be provisioned. The decision variable x_k is the number of billing cycles of VM_k that are leased from the IaaS provider. Let the VMs be sorted in nonincreasing order of performance versus price; i.e., $P_1/\mathrm{PRICE}_1 \geq P_2/\mathrm{PRICE}_2 \geq \ldots \geq P_n/\mathrm{PRICE}_n$. The VMs are considered in this order. That is, VM_1 is considered first, followed by VM_2, and so on. The optimal value function when restricted to using the first k VMs can be defined by the following equation.

$$\begin{aligned} f_k(W) = & \min_{\sum_{j=1}^{k} P_j * x_j \geq W} \sum_{j=1}^{k} \mathrm{PRICE}_j * x_j \\ \text{s.t.} \quad & x_j \geq 0 \text{ and } x_j \text{ is an integer}, \quad j = 1, 2, \ldots, k. \end{aligned} \tag{2}$$

Table 3: The results of provisioning sample VMs to sample workflow.

	w_1	w_2	w_3	w_4	w_5	w_6	TW	CSW	NSW
	3	30	27	44	26	16	146	116	30
VM_1	0.8	**0.8**	**0.8**	1.6	**0.8**	0.8	4.0		
VM_2	0.4	1.2	0.8	1.6	0.8	0.8	4.4		
VM_3	**0.1**	1	0.9	**1.5**	0.9	**0.6**	4.9		
Optimized results	VM_3	VM_1	VM_1	$VM_1 + VM_2$	VM_1	$VM_2 + VM_3$	$VM_1 \times 5$	$VM_1 \times 4$	VM_1
Cost	0.1	0.8	0.8	1.2	0.8	0.5	4.0	3.2	0.8
Makespan	1	1	0.9	2	0.87	1.67	4.87	3.87	1

The optimal value function is given by $f_m(W)$. To solve equation (2) we use dynamic programming by considering VM_1 first. If there is some idle time left in using VM_1 (because it does not use a full billing cycle), VM_2 will then be considered for using the residual time left by VM_1. If VM_2 again has idle time left, VM_3 will be considered next. This process is repeated until we reach VM_m. The following is the dynamic programming algorithm to solve equation (2).

$$
\begin{aligned}
W_1 &= W,\\
x_1 &= \left\lceil \frac{W_1}{P_1} \right\rceil,\\
f_1(W) &= \mathrm{PRICE}_1 \times x_1,\\
j &= 1,\\
&\text{For } k = 2, 3, \ldots, m \text{ do},\\
W_k &= W_j - \left(x_j - 1\right) \times P_j,\\
x &= \left\lceil \frac{W_k}{P_k} \right\rceil,\\
f_k(W) &= \min\left\{ f_j(W), f_j(W) - \mathrm{PRICE}_j + \mathrm{PRICE}_k \times x \right\},\\
x_k &= \begin{cases} 0, & f_k(W) = f_j(W),\\ x, & f_k(W) = f_j(W) - \mathrm{PRICE}_j + \mathrm{PRICE}_k \times x, \end{cases}\\
&\text{If } x_k \neq 0 \text{ then } x_j = x_j - 1 \text{ and } j = k.
\end{aligned}
\tag{3}
$$

The final $f_m(W)$ is the minimum cost of provisioning all m types of VMs to achieve the computational workload W.

VMs provisioning of individual task could cause idle time unless the computational workload of the task matches exactly the computational performance of the VMs provisioned for it. But the probability of this matching is very low. If we assemble some tasks together to provision VMs, the efficiency of resource utilization can be significantly increased. As shown in Table 3, the optimal cost for w_1 and w_3 is 0.1 and 0.8, respectively. However, if we combine w_1 and w_3 together, the total workload is exactly 30. Therefore, the optimal result is to use one VM_1 that will cost 0.8. Such cost is lower than the total cost of provisioning VMs to w_1 and w_3 separately. As a result, the more tasks we can assemble as a whole for provisioning, the more we can save money in leasing resources. In order to minimize the total cost of a workflow execution, it would be better to provision all tasks in the workflow as a whole. If we assemble all tasks as a single task, it would require that all tasks be executed sequentially. The execution time may exceed the temporal constraint of the workflow. According to Definition 4, some tasks should be partitioned from this sequential process while other tasks can be executed in parallel. In order to reduce the total execution time to be satisfied with the temporal constraint, some strategy of workflow partition will be discussed in the next section.

3.2.3. Deadline Assurance Method. When $\mathrm{ET}(G)$ is greater than the temporal constraint of the workflow G, the workflow cannot be finished before the deadline. In real life, meeting deadline constraint is very important because many real-time workflows are Urgent Computing [33] to support emergency computations such as severe weather prediction for hurricanes, flooding, earthquake, etc. Therefore, temporal constraint is a very important dimension for workflow quality of service (QoS). The temporal constraint of a workflow execution must be considered when we optimize the cost. To guarantee the workflow can be finished before the deadline, the workflow can be partitioned into several sub-workflows, where each sub-workflow can be executed in parallel and all tasks in one sub-workflow should be executed sequentially without exceeding the temporal constraint. Each sub-workflow executing on time will ensure the whole workflow to be finished before its deadline. So a series of temporal constraints must be defined for every sub-workflow.

The critical path of a workflow is the longest execution path between the entry and exit tasks of the workflow [34]. All the tasks that belong to the critical path are called critical tasks. The sum of the computational workload of the critical tasks is maximum compared with any other path in the workflow. The critical tasks are executed sequentially because of the parent-child relationship between them. The execution time of the critical path by the best performing VM must not exceed the temporal constraint D. Otherwise, there is no solution that can meet the deadline constraint. In the rest of this section, we assume that the execution time of the critical path does not exceed D.

Suppose the workflow G is partitioned into a number of sub-workflows. One of the sub-workflows G_{CSW} contains all of the critical tasks and some other (possibly none) non-critical tasks. This sub-workflow will be called the critical sub-workflow (CSW). The sub-workflows that contain no

critical tasks are called noncritical sub-workflow (NSW). There can only be one CSW but r ($r \geq 0$) NSWs.

Suppose there is a CSW G_{CSW}. If we provision the best performing VM to G_{CSW} with a resulting total execution time $ET(G_{CSW})$ and if $ET(G_{CSW}) \leq D$, then there will be a successful provisioning that can guarantee the workflow completed by the deadline, because the other NSWs can be executed with the CSW in parallel. On the other hand, if $ET(G_{CSW}) > D$, it is impossible to get a successful provisioning of VMs to the CSW because of the violation of deadline. In this case, we have to partition this CSW into a new NSW and a new CSW which includes fewer tasks. The new CSW includes all critical tasks but fewer noncritical tasks. This procedure can be repeated until the CSW can be completed before the deadline. Also, any other NSWs' execution time must be satisfied with the temporal constraints gained from the layered approach.

As a workflow, there must be a structure that determines the execution order of each task in the workflow. Actually, all parent tasks should be finished before their child tasks can begin. We can use a layered approach to ensure such a sequence. A workflow can be divided into several levels (layers) that would be executed sequentially. At the same time tasks within one level do not depend on each other, so they can execute in parallel.

Definition 5. Assume there is a workflow G as defined in Definition 1. Define the exit task to be Layer 0. Layer d is composed of all tasks that have a longest path with d edges to Layer 0 in the workflow G. So, Layer d is defined as $BL(d) = \{t_i \mid BD(t_i) = d,\ t_i \in G\}$, where

$$BD(t_i) = \begin{cases} 0, & i = n, \\ \max\limits_{t_c \in Child(t_i)} BD(t_c) + 1, & \text{otherwise.} \end{cases} \quad (4)$$

For example, the workflow shown in Figure 1 can be divided into five levels as shown in Figure 3. As long as executing tasks from the high level to the low level, all parent tasks will finish before their child tasks start. There is no execution order of the tasks in the same level. So, the sample workflow shown in Figure 3 can be arranged as a sequence: $t_1 \longrightarrow t_2 \longrightarrow t_3 \longrightarrow t_4 \longrightarrow t_5 \longrightarrow t_6$ or $t_2 \longrightarrow t_1 \longrightarrow t_4 \longrightarrow t_3 \longrightarrow t_6 \longrightarrow t_5$.

In order to guarantee that the deadline can be met, we must have $ET(G_{CSW}) \leq D$, where G_{CSW} is the critical sub-workflow. Also, the total execution time of all tasks in each layer of each NSW must not be more than the execution time of all the tasks in the same layer of CSW. Therefore, according to the execution time of CSW, we present the Layer Minimum Execution Time (LMET) as follows:

Definition 6. Assume a workflow G as defined in Definition 1 is divided into l layers as BL(0), BL(1), ..., BL($l-1$). If $BL(d) = \{t_{d_1}, t_{d_2}, \ldots, t_{d_s}\}$, $(0 \leq d \leq l-1)$, where the number of tasks in layer d is s, then the Layer Minimum Execution Time of BL(d) is $LMET(d) = (\sum_{t_{d_i} \in CSW} w_{d_i})/P_j$, where P_j is the best performance that IaaS can provide and w_{d_i} is the workload of t_{d_i}.

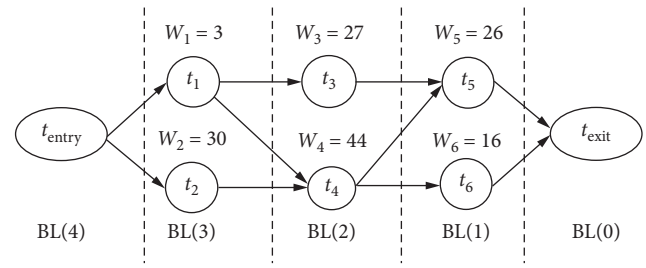

Figure 3: Layers of sample workflow.

As for a NSW, if $ET(BL(d)) \leq LMET(d)$, where $BL(d) = \{t_i \mid BD(t_i) = d,\ t_i \in NSW\}$, then the temporal constraint of NSW can be satisfied. Otherwise, a partial NSW should be partitioned from the original NSW. This process is repeated until the inequality $ET(BL(d)) \leq LMET(d)$ can be met.

Take the sample workflow G shown in Figure 1 for instance; we assume that the deadline is four hours. Using the best performing VM, VM_1, shown in Table 2 to execute this workflow, the result (shown in the TW column of Table 3) shows that using one VM_1 with five billing cycles will have $ET(G) = 4.87 > D$. So the workflow have to be partitioned into two sub-workflows. Based on the critical path $t_2 \longrightarrow t_4 \longrightarrow t_5$, we can get a CSW and a NSW as shown in Figure 4. After this partition, $ET(G_{CSW}) = 3.87 < D$, while $ET(t_1) < LMET(3) = 1$ and $ET(t_3) < LMET(2) = 1.47$. So, the temporal constraint of NSW as shown in Figure 4 can be met. The cost of each sub-workflow is also shown in Table 3 (the CSW and NSW columns of Table 3). The total cost of both CSW and NSW is 4 which is equal to the optimal cost. So the workflow partition as shown in Figure 4 is a successful cost optimization scheme.

4. Cost-Effective Resource Provisioning Algorithm

4.1. Algorithm Design. The basic idea of the global resource provisioning algorithm is that, for a given workflow G as shown in Definition 1, a dynamic programming method is used to optimize the cost of resource provisioning, and a critical path based workflow partition technique is used to guarantee the deadline requirement of the real-time workflow. Specifically, whether the deadline is met has to be determined by a Resource Provisioning in Parallel Algorithm before dynamic programming procedure is employed; once the deadline cannot be satisfied, a workflow partition procedure is employed to divide the workflow into some sub-workflow; each sub-workflow can meet its own temporal constraint; then these sub-workflows can apply the Dynamic Programming Knapsack Algorithm to get the most cost-effective resource provisioning scheme. All of these algorithms are based on a Workflow Layer Algorithm. Therefore, the whole process of our strategy is presented in detail as follows.

We first divide the workflow into l layers according to Definition 5. Second, using the dynamic programming algorithm given in the previous section, we provision m types

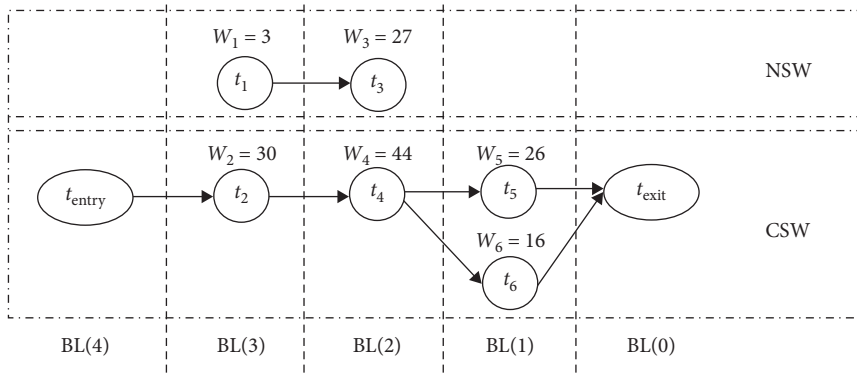

Figure 4: A partition of sample workflow.

of VMs to the workflow as though all tasks are executed in order from BL($l-1$) to BL(0). If $\mathrm{ET}(G) > D$, the provisioning fails. We then partition the workflow into a CSW and a NSW. While $\mathrm{ET}(G_{\mathrm{CSW}}) > D$, we keep partitioning until $\mathrm{ET}(G_{\mathrm{CSW}}) \leq D$ and for each layer $\mathrm{BL}(d) = \{t_i \mid \mathrm{BD}(t_i) = d,\ t_i \in \mathrm{NSW}\}$ we have $\mathrm{ET}(\mathrm{BL}(d)) \leq \mathrm{LMET}(d)$. Now we apply the dynamic programming algorithm to provision VMs to the CSW and all of the NSWs. $X = \{x_j \mid x_j \text{ is the number of VM}_j,\ (j = 1, 2, \ldots, m)\}$ obtained by this algorithm is the optimal resource provisioning results, and the minimum cost is $\mathrm{TC} = \sum_{j=1}^{m} \mathrm{PRICE}_j \times x_j$. As this algorithm can achieve the global optimization of cost, it is denoted as Global Resource Provisioning for Real-time Workflow Algorithm which is shown in Algorithm 1.

The called procedure WorkflowLayer(G) in GRP4RW is a algorithm to layer the workflow G according to Definition 5. The pseudocode of WorkflowLayer algorithm is shown in Algorithm 2.

First, (Lines 2–8) calculate the longest path of each task t_i to t_{exit} as $\mathrm{BD}(t_i)$. Next, (Lines 9–13) assign the tasks with the same value of $\mathrm{BD}(t_i)$ to the same layer. Then, sort the tasks in the same layer in nonincreasing order of their workload (Line 12). This has the effect of partitioning the larger task to a sub-workflow to minimize the number of sub-workflows. In this way the cost optimization will be more efficient. Using this algorithm, the workflow G can be divided into l layers, where each layer contains all the tasks with the same length to t_{exit}.

The next called procedure ParallelProvision(G VMs, D) in GRP4RW is an algorithm to realize parallel provisioning of resource to multiple sub-workflows in order to guarantee the cost-effectiveness of each sub-workflow on the premise of meeting deadline. The pseudocode of Resource Provisioning in Parallel Algorithm is shown in Algorithm 3.

This algorithm first attempts the Dynamic Programming Knapsack Algorithm (DPK) which is shown in Algorithm 4 to provision VMs to the original workflow (Lines 2-3). If successful, it returns the optimized provision X (Lines 4-5). Otherwise, it calls Partition Workflow Algorithm (PartitionPath) which is shown in Algorithm 5 to partition the workflow into G_{CSW} and G_{NSW} (Line 7). It then recursively call ParallelProvision procedure for CSW and NSW separately (Lines 8–12), until all sub-workflows provision VMs successfully. As to G_{CSW}, the temporal constraint is always the deadline. The temporal constraint for a G_{NSW} can get from the Layer Minimum Execution Time based on Definition 6.

This Dynamic Programming Knapsack Algorithm is based on the idea of dynamic programming elaborated in equation (3). Primarily, this algorithm will judge whether the workflow G can be completed within the fixed temporal constraint D (Line 2). If using the VM with the best computational performance to execute the workflow will not satisfy the deadline constraint, the procedure will return failure (Line 3). Otherwise, it utilizes the method of dynamic programming to find the VMs provisioning vector X and then return (Lines 5–20). Note that D is a real number expressed in terms of the number of billing cycles.

The mission of Partition Workflow Algorithm is to partition the current workflow or sub-workflow into two parts: G_{CSW} and G_{NSW}. The G_{CSW} includes, but are not limited to, critical tasks. The other part, G_{NSW}, does not contain any critical tasks. We use a Boolean variable t_i.assigned to denote the feasibility of putting t_i in G_{NSW}. Because all critical tasks cannot be put into G_{NSW}, the original value of t_i.assigned should be set as "false" except the critical tasks (Lines 2–9). We choose the noncritical path between two critical tasks as the basic of G_{NSW} (Lines 10–31). Once a task t belongs to G_{NSW}, the parent tasks of t that have not yet been assigned must belong to G_{NSW} (Lines 22–29). Such a principle can make the sub-workflow as large as possible to assign the best cost performance resources as explained in Section 3.2.2. Meanwhile, the parent and child tasks belonging to a single sub-workflow can guarantee the correct execution order of the tasks (parent tasks finished before child tasks).

4.2. Algorithm Analysis. We now consider the time complexity of our algorithm. We suppose the number of tasks in the workflow G is n, and the maximum number of types of VMs is m. The most time-consuming part of the Global Resource Provisioning for Real-time Workflow (GRP4RW) Algorithm is the WorkflowLayer and ParallelProvision procedures. The time complexity of the remaining parts of GRP4RW algorithm is $O(m)$.

Input: 1. A workflow $G = (T, W, E)$. 2. A collection of virtual machine types $\text{VMs} = \{\text{VM}_j \mid P_j, \text{PRICE}_j,\ j = 1, 2, \ldots, m\}$. 3. A fixed deadline D.
Output: 1. Optimized Provision X. 2. Minimum Cost TC.
(1) Initial $X \longleftarrow \{x_j \longleftarrow 0 \mid j = 1, 2, \ldots, m\}$;
(2) Add $t_0 \longleftarrow t_{\text{entry}}$, $t_{n+1} \longleftarrow t_{\text{exit}}$ to G;
(3) WorkflowLayer (G);
(4) X = ParallelProvision(G, VMs, D);
(5) $\text{TC} \longleftarrow \sum_{j=1}^{m} \text{PRICE}_j \times x_j$;
(6) Output X, TC

Algorithm 1: GRP4RW (global resource provisioning for real-time workflow) algorithm.

(1) **procedure** (WorkflowLayer(G))
(2) **for** $i = 0$ to $(n + 1)$ **do**
(3) **if** $i = 0$ **then**
(4) $\text{BD}(t_i) \longleftarrow 0$;
(5) **else**
(6) $\text{BD}(t_i) \longleftarrow \max_{t_c \in \text{Child}(t_i)} \text{BD}(t_c) + 1$;
(7) **end if**
(8) **end for**
(9) $l \longleftarrow \text{BD}(t_{n+1})$;
(10) **for** $d = 0$ to l **do**
(11) $\text{BL}(d) \longleftarrow \{t_i \mid \text{BD}(t_i) = d, t_i \in G\}$;
(12) Sort t_i in non-increasing order of w_i $(t_i \in \text{Bl}(d))$;
(13) **end for**
(14) **end procedure**

Algorithm 2: Workflow layered algorithm.

(1) **procedure** (ParallelProvision(G, VMs, D))
(2) $\text{TW} \longleftarrow \sum_{t_i \in G} w_i$;
(3) **if** DPK(TW, VMs, D) ≠ Failure **then**
(4) $X \longleftarrow$ DPK(TW, VMs, D);
(5) **return**;
(6) **else**
(7) PartitionPath (G)
(8) ParallelProvision(G_{CSW}, VMs, D)
(9) $t_F \longleftarrow$ the first task of G_{NSW};
(10) $t_L \longleftarrow$ the last task of G_{NSW};
(11) $D_2 \longleftarrow \sum_{d=\text{BD}(t_F)}^{\text{BD}(t_L)} \text{LMET}(d)$;
(12) ParallelProvision(G_{NSW}, VMs, D_2);
(13) **end if**
(14) **end procedure**

Algorithm 3: Resource provisioning in parallel algorithm.

The first part of WorkflowLayer algorithm (Lines 1–4) is a counting loop whose time complexity is $O(n)$. Another counting loop (Lines 6–9) will repeat l times, depending on the number of layers that the workflow is divided into. l will be no more than n because there must be at least a task in each layer. So the total time complexity of this WorkflowLayer procedure is $O(n)$.

The ParallelProvision procedure is a recursive procedure. First, it calls the DPK procedure, which provisions each type of VMs only once. So, the time complexity of the DPK procedure is $O(m)$. Then, another procedure PartitionPath will likely be called, which has time complexity $O(n)$. This is because each task belongs to either G_{CSW} or G_{NSW}, and each task will be assigned only once. Because the workflow will be partitioned into two sub-workflows by calling the PartitionPath procedure, the maximum number of times the ParallelProvision procedure will be called will be $2^{\log_2 n} = n$. Therefore, the time complexity of the ParallelProvision procedure is $O(m \cdot n + n \cdot n) = O(n \cdot (m + n))$. Consequently, the overall time complexity of the Global

```
(1) procedure (DPK(TW, VMs, D))
(2)   if D × P_1 < TW then
(3)     return Failure;
(4)   else
(5)     m ⟵ the number of VMs;
(6)     TW_1 ⟵ TW;
(7)     x_1 ⟵ ⌈TW_1/P_1⌉;
(8)     f_1 ⟵ x_1 × PRICE_1;
(9)     for j = 2 to m do
(10)      TW_j ⟵ TW_{j−1} − (x_{j−1} − 1) × P_{j−1};
(11)      x_j ⟵ ⌈TW_j/P_j⌉;
(12)      f_j ⟵ f_{j−1} − PRICE_{j−1} + x_j × PRICE_j;
(13)      if f_j ≥ f_{j−1} or x_{j−1} − 1 + TW_j/P_j > D then;
(14)        f_j ⟵ f_{j−1};
(15)        x_j ⟵ 0;
(16)      else
(17)        x_{j−1} ⟵ x_{j−1} − 1;
(18)      end if
(19)    end for
(20)    return X;
(21)  end if
(22) end procedure
```

Algorithm 4: Dynamic programming knapsack algorithm.

```
(1) procedure (PartitionPath(G))
(2)   CP ⟵ Critical Path of G;
(3)   for each t_i ∈ G do
(4)     if t_i ∈ CP then
(5)       t_i.assigned ⟵ true;
(6)     else
(7)       t_i.assigned ⟵ false;
(8)     end if
(9)   end for
(10)  k ⟵ BD(t) where t is the first task of CP;
(11)  while all of t_i ∈ BL(k) with t_i.assigned = true do
(12)    k ⟵ k − 1;
(13)  end while
(14)  t ⟵ the first t_i ∈ BL(k) where t_i.assigned = false;
(15)  G_CSW ⟵ G;
(16)  G_NSW ⟵ Null;
(17)  while t.assigned = false do
(18)    G_NSW ⟵ G_NSW + {t};
(19)    G_CSW ⟵ G_CSW − {t};
(20)    t.assigned ⟵ true;
(21)    t′ ⟵ t;
(22)    for each tp′ ∈ Parent(t′) do
(23)      if tp′.assigned = false then
(24)        t′ = tp′;
(25)        G_NSW ⟵ G_NSW + {t′};
(26)        G_CSW ⟵ G_CSW − {t′};
(27)        t′.assigned ⟵ true;
(28)      end if
(29)    end for
(30)    t ⟵ Child(t);
(31)  end while
(32)  return G_CSW, G_NSW
(33) end procedure
```

Algorithm 5: Partition workflow algorithm.

Resource Provisioning for Real-time Workflow Algorithm is $O(n) + O(n \cdot (m+n)) + O(m) = O(n \cdot (m+n))$.

5. Performance Evaluation

In this section, we present our empirical studies of the Globlal Resource Provisioning for Real-time Workflow Algorithm.

5.1. Simulation Settings. To evaluate a resource provisioning algorithm, we measure its performance on some sample workflows. Deelman et al. developed a workflow generator, which can create synthetic workflows of arbitrary size that are similar to real world real-time workflows. Using this workflow generator, they created four different sizes for each workflow application in terms of the number of tasks. These workflows are available in DAX (Directed Acyclic Graph in XML) format from their website [35]. For our experiments, we chose three sizes which are small (about 50 tasks), medium (about 300 tasks), and large (about 1000 tasks). Meanwhile, In order to explore the influence of different workflow structures on our algorithm performance, three different lengths of critical path are concerned in our simulation. However, the critical path length of each kind of workflow generated by this workflow generator is fixed and less than 10 tasks, so we have to organize several workflows altogether to construct the longer critical path we need. For example, we can connect 5 critical paths with a length of 8 and 1 critical path with a length of 10 end-to-end, so as to get a critical path with a length of 50.

For our experiments, we modeled an IaaS provider that offers a single data center and six different types of VMs with different processor speeds and different prices. The VM configurations are based on the current Amazon EC2 offerings. We used the work of Ostermann et al. [32] to estimate the computational performance in MFLOPS based on the number of EC2 compute units. Another important parameter for the experiment is the time unit for one billing cycle. Most of the current commercial clouds, such as Amazon, charge users based on a billing cycle of one hour. So, we used a VM billing cycle of one hour.

As a large number of workflows with different attributes are used for our experiments, it is important to normalize the total cost of each workflow execution in order to clearly analyze the cost optimization issue. For this reason, we first take all the tasks of a workflow as a whole and use the dynamic programming method to solve it. We define this cost as Cheapest Cost (CC). Note that the Cheapest Cost is obtained by ignoring the deadline constraint. The Normalized Cost (NC) of a workflow execution is defined by NC = (total cost of provisioned VMs)/CC, where CC is the Cheapest Cost of provisioning the VMs for the same workflow. Note that the total cost of provisioning the VMs in NC has deadline constraint. Because of the deadline constraints, clearly we have NC ≥ CC.

To evaluate the proposed algorithm, we need to assign a deadline to each workflow. In order to set a series of proper deadlines to our experiments, we define an upper bound of the deadline as the makespan of the Cheapest Cost (CC) situation and a lower bound as the makespan of the critical path executing on the best performaning VM in a sequential order. D_U and D_L denote the upper and lower bounds of deadline, respectively. To set the deadlines for workflows, we use the deadline factor α, and we set the deadline of a workflow to be $D_L + \alpha \cdot (D_U - D_L)$, where $0 \le \alpha \le 1$. We let α goes up in steps of 0.1; i.e., $\alpha = 0, 0.1, 0.2, \ldots, 0.9, 1.0$.

To the best of our knowledge, many works have combined resource provisioning and task scheduling to minimize the cost of workflow execution; IC-PCP algorithm is the classical one among these works. Therefore, we adopted IC-PCP algorithm as a baseline to evaluate our algorithm. As mentioned in Section 2, IC-PCP algorithm was designed to minimize the cost of workflow execution while meeting a user-defined deadline [17]. This algorithm begins by calculating the EST (Earliest Start Time), EFT (Earliest Finish Time), and LFT (Latest Finish Time) of each task. It then finds the partial critical paths associated with the EST, EFT, and LFT. The tasks on each path are scheduled on the same VM and are preferably assigned to an already leased instance which can meet the LFT of the tasks while the values of EST, EFT, and LFT of each unassigned task are updated. Finally, each unassigned task on the scheduled path is calculated and the process is repeated until all tasks have been scheduled. At the end of this process, each task has been assigned a VM and has start and end times associated with it. The IC-PCP algorithm also aims at cost optimization but focuses on task scheduling. The time complexity of IC-PCP algorithm is higher and it lacks flexibility. Additionally, we use the Simple DPK algorithm to each task of the workflow as another comparative reference to manifest the importance of global idea on cost optimization.

5.2. Simulation Results. Figure 5 shows the relationship between the cost optimization of resource provisioning by the GRP4RW algorithm and the deadline. Figure 5(a) shows the results with different sizes: small size (50 tasks), medium size (300 tasks), and large size (1000 tasks). These different sizes have similar variations in the trends of the normalized cost with each different deadline. The more tasks a workflow has, the more gentle change of normalized costs with each different deadline. The fluctuation of normalized cost of the workflow with the smallest size is obvious. Figure 5(a) shows that the performance of cost optimization of our proposed algorithm is more stable as the size of workflow is larger.

Figure 5(b) shows how the different structures of workflows influence the variation of normalized costs. Because a large size workflow shows more stable normalized cost, we use three large size workflows (1000 tasks each) with different lengths of critical path (CP). The lengths of the critical paths are 50 tasks, 100 tasks, and 200 tasks. The results show that there is a nearly inversely proportional linear relationship between the normalized cost and the deadline to the workflows with the longest critical path. However, the workflow with the shortest critical path has the most expedite convergence to the optimization as the deadline grows larger. Therefore, our proposed cost

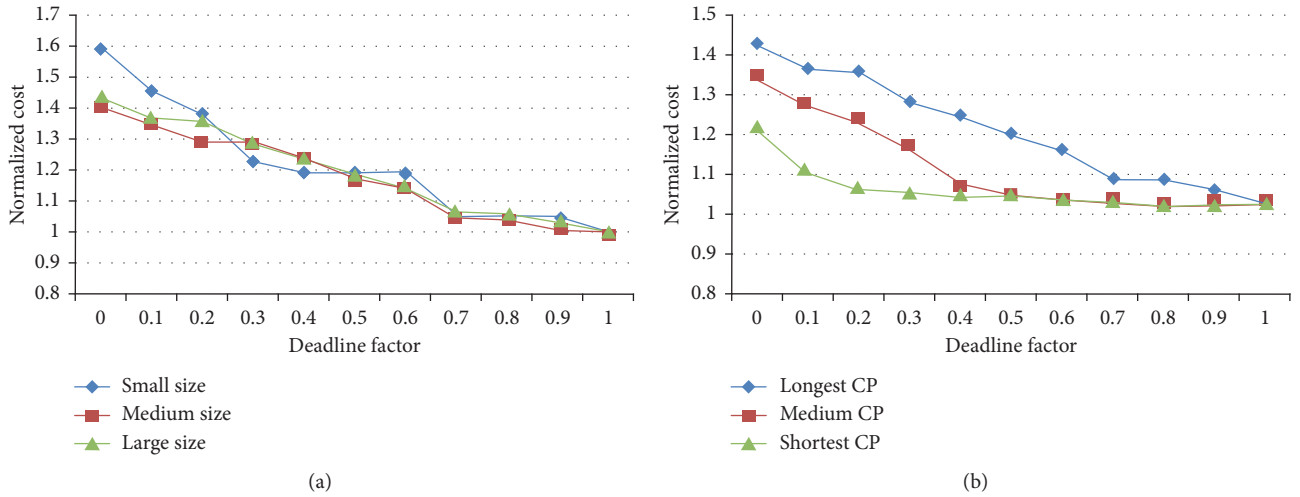

Figure 5: The normalized cost of resource provisioning with GRP4RW. (a) Three workflows with different sizes. (b) Same workflow size but different CP length.

optimization algorithm can be more efficient when the deadline is set to a low level for the workflow with shorter critical path.

Since the performance of cost optimization for larger size and longer critical path is better and more stable, we take a workflow with 1000 tasks with the length of the critical path being 200 as the simulation instance to compare the performances of three cost optimization algorithms: GRP4RW, IC-PCP, and Simply DPK. Figure 6 shows the result of the comparison. The normalized cost of Simply DPK algorithm does not vary much with the change of deadline factor and it always maintains a high value. The normalized cost of IC-PCP algorithm decreases with the increase of deadline. The normalized cost of our GRP4RW algorithm has a certain reduction with the deadline changes and the values of the costs are generally lower than the IC-PCP and Simply DPK algorithm at each deadline. There are three reasons for these results. First, the Simply DPK algorithm uses the dynamic programming knapsack method to each single task of a workflow. So, its optimization effect is modest and deadline-invariant. Second, the IC-PCP algorithm containing the combined idea uses the partial critical path as the optimization unit. This makes the optimization effect improve significantly, but dynamic programming method not utilized in this algorithm results in suboptimal normalized costs for all different deadlines. Finally, both the global idea and the dynamic programming knapsack method are used in GRP4RW algorithm. So the optimization effect achieves the optimal result. In conclusion, global idea is the most important force in the cost optimization of resource provisioning algorithm for real-time workflow. At the same time, dynamic programming knapsack method also plays a positive role in the whole process.

Considering that the length of billing cycles could impact on the performance of the cost optimization algorithms, we studied the effect of the relationship between the length of billing cycles and the execution times of tasks in workflow on the performance of the algorithms by simulation

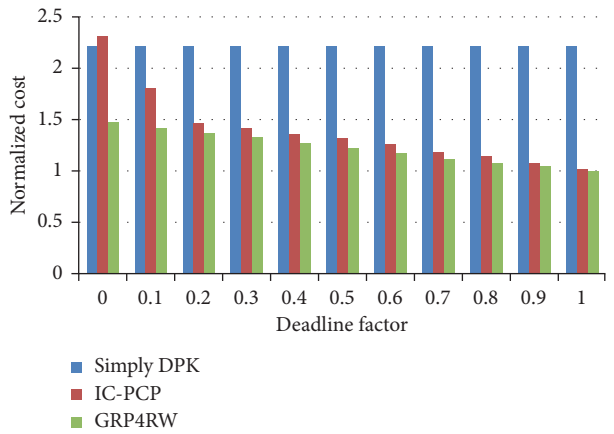

Figure 6: Normalized cost of large size workflow with simply DPK, IC-PCP, and GRP4RW.

experiments. As the execution time of task seriously depends on the workload of task, we used three different workflows with different workloads of tasks: one workflow denoted as Workflow with Small Workload in which more than 90% task workload are less than 1 MFLOPS, another workflow denoted as Workflow with Medium Workload in which more than 90% task workload are between 1 and 5 MFLOPS, and the third workflow denoted as Workflow with Large Workload in which more than 90% task workload are more than 5 MFLOPS. Meanwhile we use 30 seconds, 60 seconds, 120 seconds, 300 seconds, 600 seconds, and 1200 seconds as the different lengths of billing cycles. The experimental results are shown in Figure 7. It demonstrates that the lengths of billing cycles have impact on the algorithms performance; with the decrease of the length of billing cycles, the cost optimization effect of all algorithms is improved to some extent. When the length of billing cycle is 30 seconds, the performance of all these algorithms is optimal. When the length of billing cycle becomes 3600 seconds, the performance of algorithm Simply DPK decreases significantly,

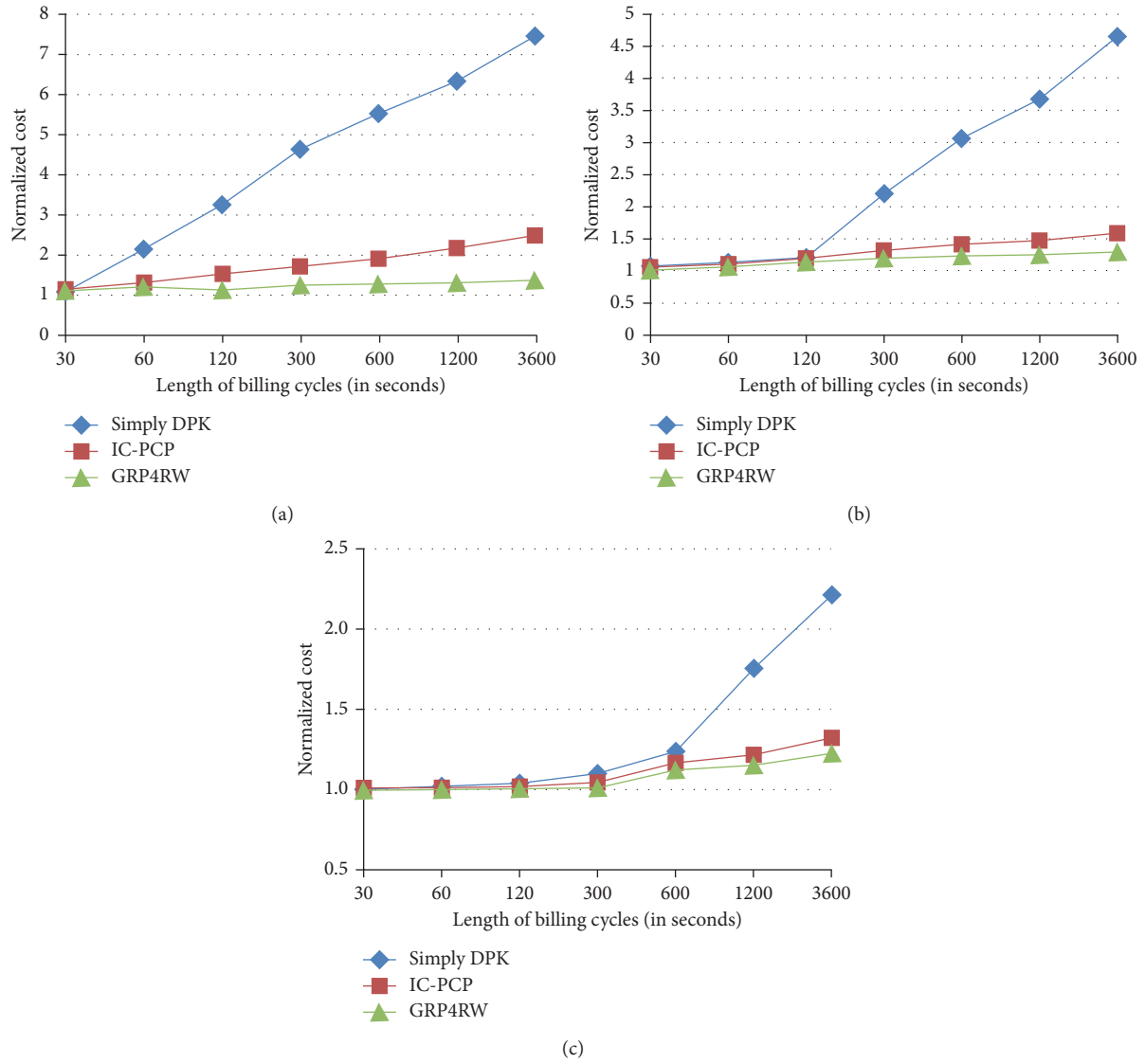

Figure 7: The Length of billing cycles impact on algorithms. (a) Workflow with small workload. (b) Workflow with medium workload. (c) Workflow with large workload.

while that of algorithms IC-PCP and GRP4SW is passable. That is to say, when the execution times of tasks are extremely smaller than billing cycle such as the workflow with small workload under the situation that the length of billing cycle is 3600 seconds (seen in Figure 7(a)), the length of billing cycles impact on the performance of algorithms is more significant, and our proposed algorithm GRP4SW has the most advantage at that condition.

6. Conclusion

In this paper, we presented a global resource provisioning algorithm for executing real-time workflow in cloud. We modeled the problem as an optimization problem that aims to get a cost-effective resource provisioning to executive a workflow while meeting a deadline constraint. The problem was solved by using the dynamic programming knapsack algorithm. Our approach embodies basic IaaS cloud properties such as heterogeneity and elasticity of resource performance and pay-as-you-go model of price mechanism.

We perform simulation experiments using three different sizes and three different structures of real-time workflows separately. Our results show that the cost optimization of our proposed solution improves as the deadline increases. We compared the performance of our algorithm against two other algorithms (Simply DPK and IC-PCP). Our results show that our proposed GRP4RW algorithm has an overall better performance than both the Simply DPK and IC-PCP algorithms under every deadline constraints.

For future work, we intend to improve our strategy for optimizing both data storage and data transfer cost of a real-time workflow execution. In addition, more and more researches have been focused on the optimization objectives about security or energy consumption property of QoS [36, 37]; the most important issue is realizing multiobjective optimization.

Data Availability

The workflows used in this study can be accessed via the website http://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work is the partially supported by the National Natural Science Foundation of China (Project no. 61972001).

References

[1] A. Ramlatchan, M. Yang, Q. Liu, M. Li, J. Wang, and Y. Li, "A survey of matrix completion methods for recommendation systems," *Big Data Mining and Analytics*, vol. 1, no. 4, pp. 308–323, 2018.

[2] C. Zhang, M. Yang, J. Lv, and W. Yang, "An improved hybrid collaborative filtering algorithm based on tags and time factor," *Big Data Mining and Analytics*, vol. 1, no. 2, pp. 128–136, 2018.

[3] X. Xu, Q. Liu, Y. Luo et al., "A computation offloading method over big data for IoT-enabled cloud-edge computing," *Future Generation Computer Systems*, vol. 95, pp. 522–533, 2019.

[4] X. Xu, S. Fu, L. Qi et al., "An IoT-oriented data placement method with privacy preservation in cloud environment," *Journal of Network and Computer Applications*, vol. 124, pp. 148–157, 2018.

[5] C. Zhang, M. Yang, J. Lv, and W. Yang, "A novel deep hybrid recommender system based on auto-encoder with neural collaborative filtering," *Big Data Mining and Analytics*, vol. 1, no. 3, pp. 211–221, 2018.

[6] X. Xu, R. Mo, F. Dai, W. Lin, S. Wan, and W. Dou, "Dynamic resource provisioning with fault tolerance for data-intensive meteorological workflows in cloud," *IEEE Transactions on Industrial Informatics*, 2019, In press.

[7] J. Zhou, X. S. Hu, Y. Ma, J. Sun, T. Wei, and S. Hu, "Improving availability of multicore real-time systems suffering both permanent and transient faults," *IEEE Transactions on Computers*, vol. 68, no. 12, pp. 1785–1801, 2019.

[8] J. Zhou, J. Sun, X. Zhou et al., "Resource management for improving soft-error and lifetime reliability of real-time MPSoCs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 12, pp. 2215–2228, 2019.

[9] Y. Zhang, K. Wang, Q. He et al., "Covering-based web service quality prediction via neighborhood-aware matrix factorization," *IEEE Transactions on Services Computing*, 2019, In press.

[10] Y. Zhang, G. Cui, S. Deng, F. Chen, Y. Wang, and Q. He, "Efficient query of quality correlation for service composition," *IEEE Transactions on Services Computing*, 2018, In press.

[11] W. Gong, L. Qi, and Y. Xu, "Privacy-aware multidimensional mobile service quality prediction and recommendation in distributed fog environment," *Wireless Communications and Mobile Computing*, vol. 2018, Article ID 3075849, 8 pages, 2018.

[12] Y.-W. Zhang, Y.-Y. Zhou, F.-T. Wang, Z. Sun, and Q. He, "Service recommendation based on quotient space granularity analysis and covering algorithm on spark," *Knowledge-Based Systems*, vol. 147, pp. 25–35, 2018.

[13] Y. Zhang, C. Yin, Q. Wu, Q. He, and H. Zhu, "Location-aware deep collaborative filtering for service recommendation," *IEEE Transactions on System, Man, and Cybernetics: Systems*, 2019, In press.

[14] L. Qi, Q. He, F. Chen et al., "Finding all you need: web apis recommendation in web of things through keywords search," *IEEE Transactions on Computational Social Systems*, vol. 6, no. 5, pp. 1063–1072, 2019.

[15] L. Qi, X. Zhang, S. Li, S. Wan, Y. Wen, and W. Gong, "Spatial-temporal data-driven service recommendation with privacy-preservation," *Information Sciences*, vol. 515, pp. 91–102. In press, 2020.

[16] H. Liu, H. Kou, C. Yan, and L. Qi, "Link prediction in paper citation network to construct paper correlation graph," *EURASIP Journal on Wireless Communications and Networking*, vol. 2019, no. 1, 2019, In press.

[17] S. Abrishami, M. Naghibzadeh, and D. H. J. Epema, "Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 158–169, 2013.

[18] M. A. Rodriguez and R. Buyya, "Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds," *IEEE Transactions on Cloud Computing*, vol. 2, no. 2, pp. 222–235, 2014.

[19] E. N. Alkhanak, S. P. Lee, R. Rezaei, and R. M. Parizi, "Cost optimization approaches for scientific workflow scheduling in cloud and grid computing: a review, classifications, and open issues," *Journal of Systems and Software*, vol. 113, pp. 1–26, 2016.

[20] A. Verma and S. Kaushal, "Cost-time efficient scheduling plan for executing workflows in the cloud," *Journal of Grid Computing*, vol. 13, no. 4, pp. 1–12, 2015.

[21] W. Zheng, Y. Qin, E. Bugingo, D. Zhang, and J. Chen, "Cost optimization for deadline-aware scheduling of big-data processing jobs on clouds," *Future Generation Computer Systems*, vol. 82, pp. 244–255, 2018.

[22] C. Yang and X. Zhang, "Workflow tasks scheduling optimization based on genetic algorithm in clouds," in *Proceedings of the 2018 IEEE International Conference on Cloud Computing and Big Data Analysis*, pp. 6–10, Chengdu, China, April 2018.

[23] Z. G. Chen, K. J. Du, Z. H. Zhan, and J. Zhang, "Deadline constrained cloud computing resources scheduling for cost optimization based on dynamic objective genetic algorithm," in *Proceedings of the 2015 IEEE Congress on Evolutionary Computation (CEC)*, pp. 708–714, Sendai, Japan, May 2015.

[24] A. Verma and S. Kaushal, "A hybrid multi-objective particle swarm optimization for scientific workflow scheduling," *Parallel Computing*, vol. 62, pp. 1–19, 2017.

[25] Q. Wu, F. Ishikawa, Q. Zhu, Y. Xia, and J. Wen, "Deadline-constrained cost optimization approaches for workflow scheduling in clouds," *IEEE Transactions on Parallel & Distributed Systems*, vol. 28, no. 12, pp. 3401–3412, 2017.

[26] J. Shi, J. Luo, F. Dong, and J. Zhang, "A budget and deadline aware scientific workflow resource provisioning and scheduling mechanism for cloud," in *Proceedings of the 2014 IEEE 18th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pp. 672–677, Hsinchu, Taiwan, May 2014.

[27] V. Arabnejad, K. Bubendorfer, and B. Ng, "Scheduling deadline constrained scientific workflows on dynamically

provisioned cloud resources," *Future Generation Computer Systems*, vol. 75, pp. 348–364, 2017.

[28] V. Singh, I. Gupta, and P. K. Jana, "A novel cost-efficient approach for deadline-constrained workflow scheduling by dynamic provisioning of resources," *Future Generation Computer Systems*, vol. 79, pp. 95–110, 2017.

[29] S. Mohammadi, H. Pedram, and L. Pourkarimi, "Integer linear programming-based cost optimization for scheduling scientific workflows in multi-cloud environments," *The Journal of Supercomputing*, vol. 74, no. 9, pp. 4717–4745, 2018.

[30] J. J. Durillo, R. Prodan, and J. G. Barbosa, "Pareto tradeoff scheduling of workflows on federated commercial clouds," *Simulation Modelling Practice and Theory*, vol. 58, pp. 95–111, 2015.

[31] Amazon Elastic Compute Cloud, 2016, http://aws.amazon.com/ec2.

[32] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "A performance analysis of EC2 cloud computing services for scientific computing," in *Proceedings of the 2009 International Conference on Cloud Computing*, Springer, Bangalore, India, pp. 115–131, September 2009.

[33] R. Tolosana-Calasanz, J. Á. Bañares, C. Pham, and O. F. Rana, "Enforcing QoS in scientific workflow systems enacted over cloud infrastructures," *Journal of Computer and System Sciences*, vol. 78, no. 5, pp. 1300–1315, 2012.

[34] S. Abrishami, M. Naghibzadeh, and D. H. J. Epema, "Cost-driven scheduling of grid workflows using partial critical paths," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 8, pp. 1400–1414, 2012.

[35] Workflowgenerator, 2014, http://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator.

[36] J. Zhou, J. Sun, P. Cong et al., "Security-critical energy-aware task scheduling for heterogeneous real-time MPSoCs in IoT," *IEEE Transactions on Services Computing*, 2019, In press.

[37] L. Qi, Y. Chen, Y. Yuan, S. Fu, X. Zhang, and X. Xu, "A QoS-aware virtual machine scheduling method for energy conservation in cloud-based cyber-physical systems," *World Wide Web*, vol. 23, no. 2, pp. 1275–1297, In press, 2019.