Uncertainty-Aware Real-Time Workflow Scheduling in the Cloud

Huangke Chen, Xiaomin Zhu and Dishan Qiu
Science and Technology on Information Systems Engineering Laboratory
National University of Defense Technology
Changsha, China 410073
Email: {hkchen, xmzhu}@nudt.edu.cn, ds_qiu@sina. com

Ling Liu
College of Computing
Georgia Institute of Technology
266 Ferst Drive, Atlanta, GA 30332-0765, USA
Email: lingliu@cc.gatech.edu

***Abstract*—Scheduling real-time workflows running in the Cloud often need to deal with uncertain task execution times and minimize uncertainty propagation during the workflow runtime. Efficient scheduling approaches can minimize the operational cost of Cloud providers and provide higher guarantee of the quality of services (QoSs) for Cloud consumers. However, most of the existing workflow scheduling approaches is designed for the individual workflow runtime environments that are deterministic. Such static workflow schedulers are inadequate for multiple and dynamic workflows, each with possibly uncertain task execution times. In this paper, we address the problem of minimizing uncertainty propagation in real-time workflow scheduling. We first introduce an uncertainty-aware scheduling architecture to mitigate the impact of uncertainty factors on the quality of workflow schedules. Then we present a dynamic workflow scheduling algorithm (PRS) that can dynamically exploit proactive and reactive scheduling methods. Finally, we conduct extensive experiments using real-world workflow traces and our experimental results show that PRS outperforms two representative scheduling algorithms in terms of costs (up to 60%), resource utilization (up to 40%) and deviation (up to 70%).**

I. Introduction

Cloud computing has become a new paradigm in distributed computing. In this paradigm, cloud providers delivery on-demand services (e.g., application, platforms and computing resources) to customers in a "pay-as-you-go" model [1]. From the customers' perspective, the cloud model is cost-effective because customers pay for their actual usage without upfront costs, and scalable because customers can access unlimited resources on demand. Due to its advantages, cloud computing has been increasingly adopted in many areas, such as banking, e-commerce, retail industry, and academy [2]. Notably, the applications in these fields usually comprise many inter-related computing and data transfer tasks [2]. As the precedence constraints among tasks in these applications, a large number of idle time slots between tasks will be left on virtual machines (VMs), which often leads to a non-negligible number of poorly utilized VMs [3]. In addition, the low resource usage in cloud platforms also wastes tremendous costs, and the improvement in resource usage for large companies (like Animoto) can be translated to significant cost savings [4].

Effective and efficient scheduling algorithms show promising ways to solve the above problems for the cloud platforms. Up to now, considerable work has been devoted to scheduling workflows for cloud platforms. However, the majority of these existing scheduling approaches are based on the accuracy of the information about task execution times and communication times among tasks. In real cloud computing environments, task execution times usually cannot be reliably estimated, and the actual values are available only after tasks have been completed. This may be contributed to the following two reasons. Firstly, tasks usually contain conditional instructions under different inputs [5]. This can be interpreted that tasks in parallel applications may contain multiple choice and conditional statements, which will lead to different program branches and loops. Different branches or loops make the task computation with large differences, which will leads directly to the same task in the face of different data input may also lead to different task execution times. Secondly, the VMs' performance in clouds varies over the time. This can be contributed to the following fact. With the advanced virtualization technology (e.g., Xen and VMware), multiple VMs can simultaneously share the same hardware resources (e.g., CPU, I/O, network, etc.) of a physical host. Such resource sharing may cause the performance of VMs subjecting to considerable uncertainties mainly due to resource interference between VMs [6], [7].

Motivation. Due to the dynamic and uncertain nature of cloud computing environments, numerous schedule disruptions (e.g., variation of task execution time, variation of VM performance, arrival of new workflows, etc.) may occur and the pre-computed baseline schedule may not be executed strictly or effective as expected in real execution. Unfortunately, the vast majority of researches did not consider these dynamic and uncertain factors, which may leave a large gap between the real execution behavior and the behavior initially expected. To address this issue, we study how to control the impact of uncertainties on scheduling results, and how to improve resource utilization for VMs and reduce cost for cloud providers, while guaranteeing the timing requirements of these workflows.

Contributions. The key contributions of this work are:

- An uncertainty-aware architecture for scheduling dynamic workflows in the cloud environments.
- A novel algorithm named PRS that combines proactive with reactive scheduling methods for scheduling real-time workflows.

- The experimental verification of the proposed PRS algorithm based on real-world workflow traces.

The outline of this paper is organized as follows. Section 2 briefly presents the related work. Section 3 gives an overview of the scheduling architecture and the problem formulation, followed by detailing the scheduling algorithm in Section 4. In section 5, we present the experimental results and analysis. Section 6 concludes this paper.

II. Related Work

Workflow is one of the most typical applications in distributed computing, and workflow scheduling has drawn intensive interests in the recent years. So far, a large number of workflow scheduling algorithms have been developed.

Among the existing workflow scheduling approaches, they can be divided into three categories: list-based, cluster-based, meta-heuristic-based. For instance, Durillo et al. proposed a list-based workflow scheduling heuristic (denoted as MOHEFT) to make tradeoff between makespan and energy consumption [8]. Lee et al. proposed two scheduling solutions that firstly stretch out the schedule to preserve makespan, and then compact the output schedule to minimize resource usage [3]. There is a large body of work in designing workflow scheduling approaches, based on task. For example, Abrishami et al. proposed a scheduling algorithm, named PCP, for utility grids to achieve both minimizing a workflow's execution cost and guaranteeing the workflow's deadline [4]. Abrishami et al. then extend their previous algorithm (i.e., PCP) to design two new algorithms, i.e., IC-PCP and IC-PCPD2, for cloud environment [9]. In addition, meta-heuristics have become another active ways to solve the workflow scheduling problems. For instance, Xu et al. applied a genetic algorithm (GA) to assign a priority to each subtask while using a heuristic approach to map taks to processors [10]. Zhu et al. proposed an Evolutionary Multi-objective Optimization-based algorithm to solve this workflow scheduling problem on cloud platform [11]. However, the above existing approaches are designed for a single workflow, and neglected the uncertainties of task execution times and the dynamic nature of workflow applications in cloud environments.

There also exist some work investigating the workflow scheduling strategies under stochastic computing environments. Calheiros et al. proposed an algorithm to replicate workflow tasks, such that mitigating the effects of resources' varied performance on workflows' deadlines [12]. Tang et al. developed a stochastic heuristic to minimize the makespan for workflow [5]. Zheng et al. proposed an approach, based on a Monte Carlo method, to cope with the uncertainties in task execution times [13]. Rodriguez et al. focused on the performance variation of VMs, the presented an algorithm, based on Particle Swarm Optimization (PSO), to minimize the overall workflow execution cost while meeting its deadline constraint [14]. However, these approaches are designed for a single workflow, and are not appropriate for dynamic cloud environments, where multiple workflows will be submitted from time to time.

III. Modeling and problem formulation

In this section, we firstly give the model of virtual machines (VMs) and workflows, and then propose an uncertainty-aware scheduling architecture for a cloud platform. After that, we form our scheduling problem.

A. Virtual machine modeling

The cloud platforms often provide abundant virtual machines of many types, denoted as $S = \{s_1, s_2, \cdots, s_m\}$ [14]. Each VM type s_u has a specific configuration and a price associated with it. The configuration of VM type differs with respect to CPU performance, memory, storage, network and OS. Each s_u has a price $Price(s_u)$ associated with it, charged on an unit time basis. The time duration is rounded to the next full hour, e.g., 5.1 hours is rounded to 6 hours. We utilize the symbol $vm_k^{s_u}$ to denote the k-th VM with type s_u.

In cloud environments, VMs may locate in different data centers, and the underlying network topology among VMs is complex and heterogeneous. Without loss of generality, the parameter l_{kl} is utilized to represent the communication bandwidth between VM $vm_k^{s_u}$ and VM $vm_l^{s'_u}$. To simplify the problem, the network congestion will be ignored in this study, which is similar to [14].

B. Modeling Workflows with Uncertain Task Execution Times

In cloud environment, the workflows are continuously submitted by customers, and these workflows can be denoted as $W = \{w_1, w_2, ..., w_m\}$. For a certain workflow application $w_i \in W$, it can be modeled as $w_i = \{a_i, d_i, G_i\}$, where a_i, d_i and G_i represent the arrival time, the deadline and the structure of workflow w_i, respectively. The structure G_i for a certain workflow w_i can be formally expressed as a directed acyclic graph (DAG), i.e., $G_i = (T_i, E_i)$, where $T_i = \{t_{i1}, t_{i2}, ..., t_{i|T_i|}\}$ represents the task set in workflow w_i. Additionally, $E_i \subseteq T_i \times T_i$ represents a set of directed arcs between tasks. An edge $e_{pj}^i \in E_i$ of the form (t_{ip}, t_{ij}) exists if there is a precedence constraint between task t_{ip} and task t_{ij}, where t_{ip} is an immediate predecessor of task t_{ij} and the task t_{ij} is an immediate successor of task t_{ip}. The weight $w(e_{pj}^i)$, that assigned to edge e_{pj}^i, represents the size of data that needs to be transferred. As a task may have many predecessors and successors, we let the $pred(t_{ij})$ and $succ(t_{ij})$ implies the set of all the immediate predecessors and successors of task t_{ij}.

Notably, the main difference between uncertain scheduling and deterministic scheduling is that task execution times are random or deterministic. As the network performance is beyond our scope, the communication times between tasks are assumed to be deterministic. Besides, the task execution times are interpreted as random variables, and assumed to be independent and normal distributions [5].

C. Scheduling architecture

In this paper, we design an uncertainty-aware scheduling architecture for a cloud platform, as shown in Fig. 1. The platform consists of three layers: user layer, scheduling layer and resource layer. In cloud environment, users will dynamically

submit their workflow applications to the service provider. The scheduling layer is responsible for generating task-to-VM mappings, according to certain objectives and predicted resource performance. The resource layer consists of large-scale heterogeneous VMs, and these VMs can be scaled up and down dynamically.

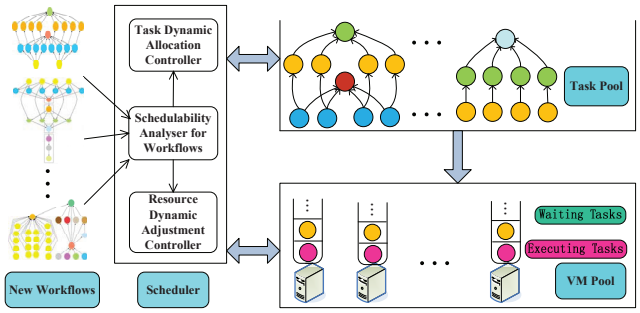


Fig. 1. The Uncertainty-Aware Scheduling Architecture

As a study of scheduling algorithm is our primary concern here, we focus on the scheduling layer, which consists of a task pool (TP), a schedulability analyser, a resource controller, and a task controller. The TP accommodates most of the waiting tasks, and the schedulability analyzer is responsible for producing the blue print of scaling up/down the computing resources and the mappings of waiting tasks in TP to VMs. The blue print of computing resources adjustments includes when to add/delete the number of different type VMs, and the resource controller will conduct them. In addition, the task controller will dynamically allocate waiting tasks from TP to corresponding VMs according to the task-to-VM mappings.

The unique features in this scheduling architecture are that most of waiting tasks are waiting in the TP instead of waiting on the VMs directly, and only the tasks, which have been mapped to VMs, are allowed to wait on each VM. The benefits of this scheduling architecture are summarized as follows.

- It can prohibit propagation of uncertainties throughout the schedule. Since only the scheduled tasks are allowed to wait on VMs, the uncertainty of the executing task can only transfer to the waiting tasks on the same VM. When the executing task is completed, its uncertainty does not exist, and the subsequent waiting tasks on that VM will not be affected by the task finished. Therefore, this architecture can prohibit propagation of uncertainties.
- It is convenient for the system to reoptimize its schedule when new workflows arrive by fully using all information available at that time.
- This design allows each task waiting on VMs to start as soon as its preceding task has finished, so the possible execution delay for a new task is removed.
- This design enables overlapping of communications and computations. When a VM is executing a task, it can simultaneously receive another tasks as waiting tasks. By doing so, communications and computations are efficiently overlapped to save time.

D. Problem formulations

The assignment variable $x_{ij,k}$ is utilized to reflect the mapping relationship between task t_{ij} and VM $vm_k^{s_u}$. It is 1 if t_{ij} is assigned to $vm_k^{s_u}$, otherwise 0, i.e.,

$$x_{ij,k} = \begin{cases} 1, & \text{if } t_{ij} \text{ is assigned to } vm_k^{s_u}, \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Definition 1. The index of the VM where task t_{ij} is assigned to is defined as $r(t_{ij})$. For instance, if task t_{12} in workflow w_1 is assigned to VM $vm_8^{s_3}$, then $r(t_{ij})$=8.

Since the execution time of a task is a random variable, we utilize its α quantile to approximate it when scheduling workflows. The symbol $et_{ij,k}^{\alpha}$ is utilized to denote the α quantile of the execution time of task t_{ij} on VM $vm_k^{s_u}$. In addition, symbols $pst_{ij,k}$ and $pft_{ij,k}$ are utilized to denote the predicted start time and the predicted finish time of task t_{ij} on VM $vm_k^{s_u}$, respectively. The predicted start time $pst_{ij,k}$ can be calculated as follows:

$$pst_{ij,k} = \max\{pft_{lh,k}, \max_{t_{ip} \in pred(t_{ij})}\{pft_{ip,r(t_{ip})} + tt_{pj}^{i}\}\}. \tag{2}$$

where $pft_{lh,k}$ represents the predicted finish time of task t_{lh} which is the currently last task on VM $vm_k^{s_u}$; $r(t_{ip})$ represents the index of the VM that task t_{ip} is assigned to; and tt_{pj}^{i} represents the transfer time of the data dependency e_{pj}^{i}.

Apparently, the predicted finish time $pft_{ij,k}$ of task t_{ij} on VM $vm_k^{s_u}$ can be written as

$$pft_{ij,k} = pst_{ij,k} + et_{ij,k}^{\alpha}. \tag{3}$$

After all the tasks in w_i are scheduled, w_i's predicted finish time pft_i in the baseline schedule is defined as:

$$pft_i = \max_{t_{ij} \in T_i}\{pft_{ij,r(t_{ij})}\}. \tag{4}$$

After tasks have been finished, their real start times, execution times and finish times will be available. The symbols $rst_{ij,k}$, $rst_{ij,k}$ and $rft_{ij,k}$ are used to denote the real start time, the real execution time and the real finish time of task t_{ij} on VM $vm_k^{s_u}$, respectively. For example, the execution time of task t_{11} on VM $vm_2^{s_3}$ is assumed to be $et_{11,2} \sim N(120, 10^2)$s and the 0.9 quantile of $et_{11,2}$ is $et_{11,2}^{0.9} = 132.8$ s before scheduling; the real finish time $rft_{11,2}$ may be 135s after task t_{11} has been finished on VM $vm_2^{s_3}$. Thus, the real finish time of workflow w_i is defined as

$$rft_i = \max_{t_{ij} \in T_i}\{rft_{ij,r(t_{ij})}\}. \tag{5}$$

Under uncertain scheduling environments, it is the real finish time rft_i of workflow w_i that determines whether its timing requirement has been guaranteed or not. So we have the following constraint:

$$\max_{t_{ij} \in T_i}\{rft_{ij,r(t_{ij})}\} \leq d_i \qquad \forall w_i \in W. \tag{6}$$

Due to precedence constraints (data dependencies exist between tasks) in a workflow, a task can be executed only

after all its predecessor tasks are completed. This constraint is shown as following:

$$rft_{ip,r(t_{ip})} + tt^i_{pj} \leq rst_{ij,k}, \qquad \forall e^i_{pj} \in E_i, \tag{7}$$

Subjecting to aforementioned constraints, as specified in formula (6) and (7), the primary optimization objective is to minimize total cost for executing the workflow set W, i.e.,

$$\text{Minimize} \sum_{k=1}^{|VM|} Price(vm_k^{s_u}) \cdot tp_k \ . \tag{8}$$

where $|VM|$ denotes the total number of VMs utilized to execute the workflow set W, and tp_k is the working time periods of VM $vm_k^{s_u}$.

Apart from the total cost, resource utilization is also an important metric to evaluate the performance of a cloud platform. Thus, we also focus on maximizing the average resource utilization of VMs, which can be represented as follows.

$$\text{Maximize} \sum_{k=1}^{|VM|} (wt_k) / \sum_{k=1}^{|VM|} (tt_k), \tag{9}$$

where wt_k and tt_k represent the working time and the total active time (including working and idle time) of VM $vm_k^{s_u}$ during executing the workflow set W.

Another objective needed to be optimized under uncertain computing environments is to minimize the deviation cost function [15], defined as the average of weighted sum of the absolute deviations between the predicted finish time of workflows in the baseline schedule and their realized finish time during actual schedule execution, which can be described as follows:

$$\text{Minimize} \ \frac{1}{m} \sum_{i=1}^{m} w_i(|pft_i - rft_i|), \tag{10}$$

where w_i represents the marginal cost of time deviation between the predicted and actual finish time.

IV. Algorithm design

In this paper, we propose a heuristic that incorporates both the proactive and the reactive scheduling methods, to obtain a sub-optimal schedule with much cheaper computational overhead. The proactive scheduling is used to build baseline schedules based on redundancy, where α quantiles of task execution times are utilized when making schedule decisions. The reactive scheduling is dynamically triggered to generate proactive baseline schedules in order to account for various disruptions during the course of executions.

We treat the following two events as disruptions: (1) new workflow arrive; (2) a VM finishes a task. These two disruptions take place discretionarily and arbitrarily; if any of the disruptions occurs, the corresponding reactive scheduling will be triggered.

A. Ranking tasks with uncertain execution times

An important issue in workflow scheduling is how to rank the tasks. In this paper, all the tasks in the task pool will be ranked by their predicted latest start time $plst_{ij}$. The $plst_{ij}$ for each task t_{ij} is defined as the latest time, after which task starts its execution, such that the predicted finish time pft_i of workflow w_i will be more than its deadline d_i.

Definition 2. The $plst_{ij}$ for task t_{ij} is recursively defined as following.

$$plst_{ij} = \begin{cases} d_i - met^{\alpha}_{ij}, \text{ if } succ(t_{ij}) = \varnothing, \\ \min\limits_{t_{is} \in succ(t_{ij})} \{plst_{is} - tt^i_{js}\} - met^{\alpha}_{ij}, \text{ otherwise.} \end{cases} \tag{11}$$

where $succ(t_{ij})$ represents the set of immediate successors of task t_{ij}; met^{α}_{ij} denotes the minimum of α quantile of t_{ij}'s execution time.

Based on definition 2, the predicted least finish time $plft_{ij}$ for task t_{ij} can be calculated as

$$plft_{ij} = plst_{ij} + met^{\alpha}_{ij}. \tag{12}$$

B. Scheduling algorithm

Definition 3. Ready task: a task is ready if it has not any predecessors, i.e., $pred(t_{ij}) = \varnothing$; or all of its predecessors have been mapped to VMs and at least one of its predecessors has been completed.

With regard to the traditional scheduling schemes, once a new workflow arrives, all its tasks are mapped and dispatched immediately to the local queues of VMs or hosts. Unlike them, our approach puts most of waiting tasks in the task pool, and only ready tasks will be scheduled and dispatched to VMs. Over the actual execution of the cloud platform, new mappings for the waiting tasks in task pool will be generated continually. The PRS performs the following operations when a new workflow arrives, as shown in Algorithm 1.

Algorithm 1 PRS - On the arrival of new workflows

```
1: taskPool ← ∅;
2: for each new workflow w_i arrives do
3:     Calculate plst_ij and plft_ij for each task t_ij in w_i according
       to formula (11) and (12).
4:     readyTasks ← get all the ready tasks in workflow w_i;
5:     Sort readyTasks by tasks' ranks in an increasing order;
6:     for each ready task t_ij ∈ readyTasks do
7:         Schedule task t_ij by function ScheduleReadyTask();
8:     end for
9:     Add all the non-ready tasks in w_i into set taskPool;
10: end for
```

When a new workflow w_i arrives, algorithm **PRS** will calculate the predicted least start/finish time (i.e., $plst_{ij}$ and $plft_{ij}$) for each task in workflow w_i (Line 3). After that, all the ready tasks in this workflow w_i are selected and sorted by their $plst_{ij}$ in a non-descending order (Lines 4-5). Then, these ready tasks, starting from the first task, will be scheduled to VMs by the function **ScheduleReadyTask()** (Lines 6-8). Additionally, all the non-ready tasks in this new workflow w_i will be added into $taskPool$ (Line 9).

Since the finish time of a task on a VM is random, we regard the completion of a task by a VM as a sudden event. When this sudden event occurs, if there exist waiting tasks on the VM, the first waiting task starts to execute immediately if all the predecessors of the first waiting task have been completed. In addition, when a task has been completed, its successors may become ready, and the algorithm PRS will be triggered to schedule these ready tasks to VMs. When a task, denoted as t_{ij}, is finished by a VM, denoted as $vm_k^{s_u}$, the algorithm PRS performs the following operations, as shown in Algorithm 2.

Algorithm 2 PRS - On a task completion by a VM

1: **if** there exist waiting tasks on VM $vm_k^{s_u}$ **then**
2: Starts to execute the first waiting task on VM $vm_k^{s_u}$;
3: **end if**
4: $readyTasks \leftarrow \emptyset$;
5: **for** $t_{is} \in succ(t_{ij})$ **do**
6: **if** task t_{is} is ready **then**
7: $readyTasks \leftarrow (readyTasks \cup t_{is})$;
8: **end if**
9: **end for**
10: $taskPool \leftarrow (taskPool - readyTasks)$;
11: Sort $readyTasks$ by their ranks $plst_{ij}$ in an increasing order;
12: **for** each ready task $t_{ij} \in readyTasks$ **do**
13: Schedule task t_{ij} by function **ScheduleReadyTask()**;
14: **end for**

As shown in algorithm 2, when VM $vm_k^{s_u}$ completes its executing task, if its waiting task wtk_k is not empty, this waiting task will be executed immediately (Lines 1-3). Then, task t_{ij}'s successors that become ready will be selected (Lines 4-9), and these ready tasks will be removed from $taskPool$ (Line 10). And algorithm PRS sorts the ready tasks by their predicted least start times $plst_{ij}$ (Line 11). After that, the ready tasks are scheduled to VMs by function **ScheduleReadyTask()**.

The predicted cost $pc_{ij,k}$ of task t_{ij} on $vm_k^{s_u}$ is defined as:

$$pc_{ij,k} = Price(vm_k^{s_u}) \times (pft_{ij,k} - prt_k). \quad (13)$$

where prt_k denotes the predicted time at which VM $vm_k^{s_u}$ is available for task t_{ij}.

The pseudo-code for function **ScheduleReadyTask()** is shown in Algorithm 3.

Algorithm 3 Function ScheduleReadyTask()

1: $minCost \leftarrow +\infty$; $targetVm \leftarrow \emptyset$;
2: **for** each VM $vm_k^{s_u}$ the system **do**
3: Calculate the $pft_{ij,k}$ and $pc_{ij,k}$ as formula (3) and (13) for task t_{ij} on VM $vm_k^{s_u}$;
4: **if** $pft_{ij,k} \leq plft_{ij}$ && $pc_{ij,k} < minCost$ **then**
5: $targetVm \leftarrow vm_k^{s_u}$; $minCost \leftarrow pc_{ij,k}$;
6: **end if**
7: **end for**
8: **if** $targetVm \; != \emptyset$ **then**
9: Assign task t_{ij} to the $targetVM$;
10: **else**
11: Lease a new VM $vm_k^{s_u}$ with the minimal $pc_{ij,k}$ while satisfying $pft_{ij,k} \leq plft_{ij}$;
12: Assign task t_{ij} to VM $vm_k^{s_u}$ after it has been initiated;
13: **end if**

TABLE I
THE CONFIGURATION AND PRICES FOR VMS.

Type	Name	Price (in $)	CPUs	$Factor(s_i)$
s_1	m1.small	0.02/hour	1	1.6
s_2	m1.large	0.08/hour	4	1.4
s_3	m1.xlarge	0.45/hour	8	1.4
s_4	m2.xlarge	0.66/hour	6.5	1.2
s_5	m2.2xlarge	0.80/hour	13	1.0

In function **ScheduleReadyTask()**, as shown in Algorithm 3, we employ two policies to schedule a ready task to a VM. In policy one, the initiated VM, which can finish this task within its $plft_{ij}$ and yields the minimal predicted cost $pc_{ij,k}$ (Lines 4-6), is selected for this ready task (Lines 2-7). If the first policy cannot find an applicable VM for this ready task, the policy two will lease a new VM that generates the minimal predicted cost $pc_{ij,k}$ and can finish this task before its $plft_{ij}$ (lines 10-13).

V. PERFORMANCE EVALUATION

Since there is no competitive algorithm, we choose to compare algorithm PRS with the modified versions of two previous algorithms: Stochastic Heterogeneous Earliest Finish Time (SHEFT) [5] and Robustness Time Cost (RTC) [7].

SHEFT: this algorithm firstly compute each task's priority, which is the length of the stochastic critical path from the task to the exit task. Then, the task with higher priority will be preferentially scheduled to a machine with minimal approximate finish time.

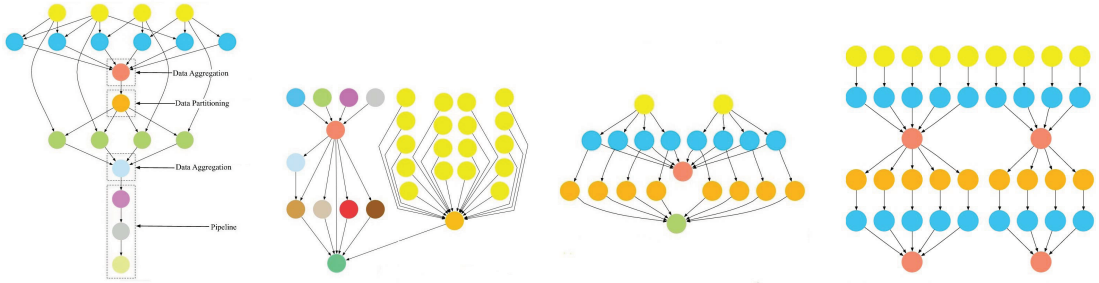
RTC: this algorithm consists of three phases. In the first phase, all the tasks in a workflow are clustered into multiple partial critical paths (PCPs). Then, an exhaustive solution set for each PCP will be generated according to deadline and budget constraints. After that, a feasible solution, which gives priority to robustness, followed by time and finally cost, is selected for each PCP.

As algorithm SHEFT and RTC are designed for a single workflow, we enable them to be fit for dynamic workflows by triggering it to schedule all the tasks in the workflow immediately when a new workflow arrives.

A. Experimental setup

The CloudSim framework [16] is utilized to simulate the cloud environment, and implement our algorithms within this environment. We assume the cloud platform offer 6 different types of VMs, and the number of VMs with each configuration is infinite. Table I lists theses VMs' configurations and prices, which are borrowed from EC2 [17]. In addition, the charging period of these VMs is 60 minutes. The bandwidth among VMs is assumed to be 1 Gbps.

We make a workflow template set using four real-world scientific workflows: Montage (astronomy), SIPHT (bioinformatics), CyberShake (earthquake science), LIGO (gravitaltional physics). There are a total of 12 elements in the workflow template set, i.e., including three different sizes of these



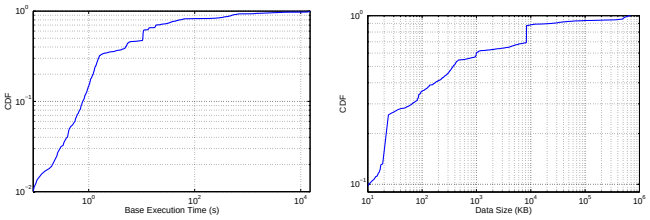
(a) Montage (b) Sipht (c) CyberShake (d) LIGO

Fig. 2. The structure of four realistic scientific workflows.

workflows, which are small (around 30 tasks), medium (around 50 tasks) and large (around 100 tasks). The approximate structure of a small instance of each workflow is shown in Fig. 2 [2], [18].

To realize the dynamic nature of workflows in cloud environments, the workflow templates are selected randomly after a time interval and submitted to the scheduler. In addition, the time interval between two consecutive workflows is a variable, and let it follow Poisson distribution with $1/\lambda = 100$s.

We assume the parameter bet_{ij} represents the base execution time of task t_{ij}, which is corresponding to the task runtime in the workflow trace [18]. The cumulative distribution function (CDF in short) of task base execution times is depicted in Fig. 3(a). In addition, Fig. 3(b) shows the CDF of data sizes between tasks.



(a) (b)

Fig. 3. The features of workflows.

As there exist multiple VM types in a cloud platform, the base execution times of a task (e.g., t_{ij}) on different VM types are not the same. We utilize parameter $Factor(s_u)$ to describe the above features, and the base execution times of task t_{ij} on different VM types is calculated as following.

$$bet_{ij,k} = bet_{ij} \cdot Factor(s_u), \tag{14}$$

where k is the symbol of the k-th VM, whose type is s_u.

The execution time of each task is modelled as a normal distribution (denoted as $\widetilde{et}_{ij,k} = N(\mu, \delta)$), with the task base execution time as the mean μ and a relative task runtime standard deviation δ, which can be denoted as

$$\begin{aligned} \mu &= bet_{ij,k}; \\ \delta &= bet_{ij,k} \cdot variance, \end{aligned} \tag{15}$$

TABLE II
PARAMETERS FOR SIMULATION STUDIES

Parameter	Value (Fixed)-(Varied)
Workflow Count (10^3)	(1.0)-(1.0,2.0,3.0,4.0,5.0)
$deadlineBase$	(1.5)-(1.5,2.0,2.5,3.0,3.5)
$timeInterval$ (s)	(Poisson($1/\lambda = 100$))
$variance$	(0.2)-(0.1,0.15,$\cdots$,0.40,0.45)

where $variance$ denotes the variation in task execution times.

In this experiment, we calculate the realized execution time for a task as follows.

$$ret_{ij,k} = rand\_N(bet_{ij,k}, (bet_{ij,k} \cdot variance)^2), \tag{16}$$

where $rand\_N(\mu, \delta^2)$ represents an random number generator that generate values from the uniform distribution with a mean value of $bet_{ij,k}$ and standard deviation of $(bet_{ij,k} \cdot variance)$.

Finally, we need to assign a deadline to each workflow. To do this, fastest schedule is defined as scheduling each workflow task on a fastest VM (i.e., m2.4xlarge), while all data transmission times are considered to be zero. The parameter M_F is defined as the makespan of the fastest schedule of a workflow. In order to set deadlines for workflows, we define the deadline factor $deadlineBase$, and we set the deadline of a workflow as follows.

$$d_i = a_i + deadlineBase \cdot M_F. \tag{17}$$

The values of the parameters in the experiments are listed in Table II. For each group of experimental settings, each algorithm is tested 30 times independently. Then, we present the average, minimum and maximum of experimental results, in terms of cost as formula (8), resource utilization as formula (9), deviation as formula (10).

B. Variance of task execution times

We conduct a group of experiments to observe the impact of uncertainties in task execution times on the performance of the three algorithms (see Fig. 4). We vary the $variance$ value from 0.10 to 0.45 with an increment of 0.05.

Fig. 4(a) shows that the total cost of the three algorithms descends at different rate as the $variance$ increases, and this

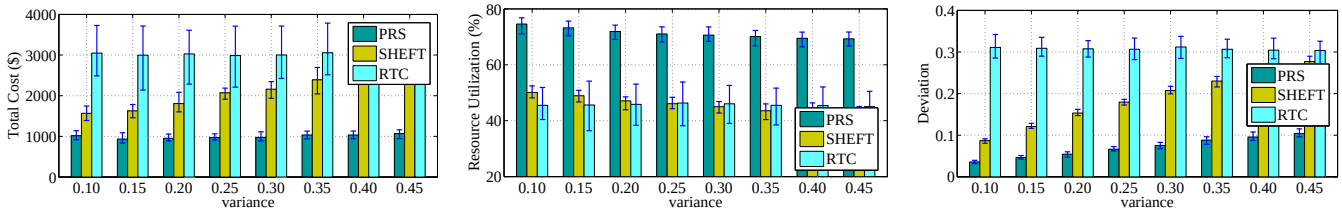


Fig. 4. Performance impact of variance in task execution times.

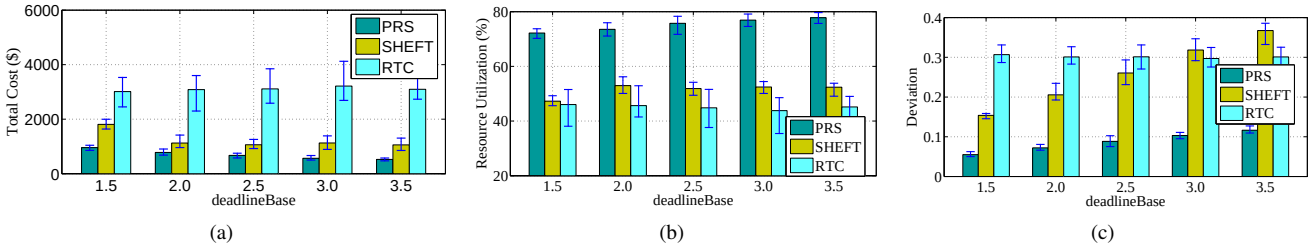


Fig. 5. Performance impact of workflow deadline.

trend is especially outstanding with SHEFT and RTC. This is because that SHEFT and RTC do not employ any reactive strategies to control the uncertain factors while executing baseline schedule. Additionally, the cost of PRS on average is less than SHEFT and RTC by 50.94% and 67.23%, respectively. The above results demonstrate that algorithm PRS can reduce the cost for cloud providers, regardless of the variance of task execution times.

Fig. 4(b) reveals that the resource utilization of PRS stays at a high level (round 71.17%), while that of SHEFT and RTC on average are 46.19% and 45.62%, respectively. This is due to a few reasons. Firstly, when the waiting tasks become ready, these ready tasks will be dynamically scheduled to VMs by PRS, such that the idle time slots in each VM can be compressed and removed as possible. Secondly, for SHEFT and RTC, the interval between tasks' predicted finish times and actual finish times becomes larger as the $variance$ increases, thus the time cushions wasted by them become larger, resulting in lower resource utilization.

Fig. 4(c) shows that when the $variance$ increases, the deviation of PRS, SHEFT and RTC increases. This is because that larger $variance$ makes the difference between the predicted and realized finish time of tasks becomes larger. Besides, the deviation of PRS is, on average, (76.78%) lower than that of RTC, since RTC does not control the propagation of uncertainties among waiting tasks. The result in this experiment account for that our method in this paper can alleviate the impact of uncertainties on the baseline schedule.

C. Workflow deadlines

Fig. 5 shows the impacts of deadlines on the performance of our proposed PRS as well as the existing algorithms - SHEFT and RTC.

We observe from Fig. 5(a) that the cost of the three algorithms descends slightly with the increase of $deadlineBase$ (i.e., workflw deadlines become looser). This can be interpreted that the deadlines of workflows are prolonged, making workflows can be finished later within their timing constraints. Consequently, more parallel tasks (i.e., there exists not dependence constraints between these tasks) in a workflow can be executed by the same VMs, such that less VMs being used. In addition, Fig. 5(a) shows that PRS costs less than SHEFT and RTC on average by 43.28% and 77.51%, respectively. This experimental result indicates that the algorithm PRS, incorporating both the proactive and the reactive strategies, is efficient with saving cost for cloud providers.

From Fig. 5(b), we can see that when $deadlineBase$ increases, the resource utilizations of PRS, SHEFT and RTC increase accordingly. This is due to the fact that as workflow deadlines become longer, their makespan can be longer without violating their deadlines, and more parallel tasks could share the same resources, such that the idle time slots in these resources can be compressed and reduced. Besides, the resource utilization of PRS on average outperforms SHEFT and RTC by 31.49% and 40.17%, respectively. It is not a surprise that PRS has so high resource utilization (about 75.02%) since only the tasks that have become ready are dynamically scheduled to VM, and the idle time slots could be cut down as possible.

In Fig. 5(c), we can observe that when the $deadlineBase$ increases, the deviation of PRS and SHEFT increase visibly, but that of RTC is almost constant. Besides, the deviation of PRS outperforms SHEFT and RTC on average by 66.03% and 70.97%, respectively. The reason is that algorithm SHEFT and RTC will schedule all the tasks in the workflows to VMs as soon as workflows arrive, thus resulting in the accumulating

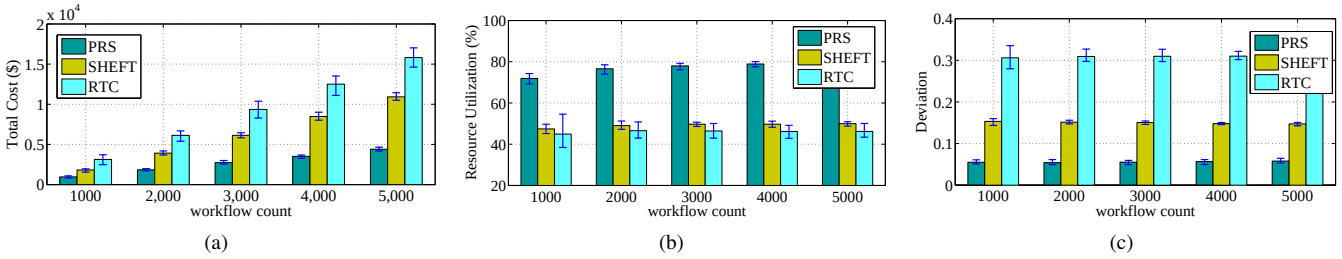


Fig. 6. Performance impact of workflow count.

of uncertainties. The result in Fig. 5(c) demonstrates the efficiency of our strategies in term of controlling the uncertainties while scheduling.

D. Count of workflows

In this group of experiments, we study the impact of the count of workflows on system performance. Fig. 6 illustrates the experimental results of PRS, SHEFT and RTC when the count of workflows varies from 1000 to 5000 with an increment of 1000.

The first observation drawn from Fig. 6(a) is that, for all the three algorithms, the total cost is improved with the increase of workflow count. It is obvious that the more workflows need the more VMs working longer, such that causing the more cost. In addition, our findings show that PRS on average costs less than SHEFT and RTC by 55.05% and 71.12%, respectively.

In Fig. 6(b), we can see that with the workflow count increases, the resource utilization for algorithm PRS, SHEFT and RTC are stable at 76.96%, 49.16% and 45.84%, respectively. This experimental result shows that workflow count has negligible impact on the resource utilization of cloud platforms.

Fig. 6(c) shows that the deviation of PRS, SHEFT and RTC are 0.0552, 0.1505 and 0.3082, respectively. This can be attributed to that the increase of workflow count seldom affects the baseline schedule. The reason for the low deviation for PRS is that it includes a scheduling architecture that can mitigate the impact of uncertainties on schedule and the reactive strategies to cope with the uncertain events during baseline execution.

VI. Conclusion

In this paper, we investigate how to reduce the cost and improve the resource utilization for cloud platforms while guaranteeing the timing constraints for real-time workflows with uncertain task execution times. We proposed an uncertainty-aware scheduling architecture for a cloud platform, and developed a novel scheduling algorithm, namely PRS, to make good trade-offs among cost, system's resource utilization and deviation. In addition, the extensive experiments conducted using real-world workflow applications demonstrate that our approach dominates the two baseline algorithms for all the benchmarks.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[2] J. Gideon, C. Ann, D. Ewa, B. Shishir, M. Gaurang, and V. Karan, "Characterizing and profiling scientific workflows," *Future Generation Computer Systems*, vol. 29, pp. 682–692, 2013.

[3] Y. C. Lee and A. Y. Zomaya, "Stretch out and compact: Workflow scheduling with resource abundance," in *Proceedings of the 2013 International Symposium on Cluster Cloud and the Grid (CCGRID)*. IEEE, 2013, Conference Proceedings, pp. 367–381.

[4] S. Abrishami, M. Naghibzadeh, and D. 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.

[5] 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.

[6] H. Chen, X. Zhu, H. Guo, J. Zhu, X. Qin, and J. Wu, "Towards energy-efficient scheduling for real-time tasks under uncertain cloud computing environment," *Journal of Systems and Software*, vol. 99, pp. 20–35, 2015.

[7] D. Poola, S. K. Garg, R. Buyya, Y. Yang, and K. Ramamohanarao, "Robust scheduling of scientific workflows with deadline and budget constraints in clouds," in *Advanced Information Networking and Applications (AINA), 2014 IEEE 28th International Conference on*. IEEE, 2014, pp. 858–865.

[8] J. J. Durillo, V. Nae, and R. Prodan, "Multi-objective energy-efficient workflow scheduling using list-based heuristics," *Future Generation Computer Systems*, vol. 36, pp. 221–236, 2014.

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

[10] Y. Xu, K. Li, J. Hu, and K. Li, "A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues," *Information Sciences*, vol. 270, pp. 255–287, 2014.

[11] Z. Zhu, G. Zhang, M. Li, and X. Liu, "Evolutionary multi-objective workflow scheduling in cloud," *DOI:10.1109/TPDS.2015.2446459*, 2015.

[12] R. N. Calheiros and R. Buyya, "Meeting deadlines of scientific workflows in public clouds with tasks replication," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 7, pp. 1787–1796, 2014.

[13] 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.

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

[15] S. Van de Vonder, E. Demeulemeester, and W. Herroelen, "Proactive heuristic procedures for robust project scheduling: An experimental analysis," *European Journal of Operational Research*, vol. 189, no. 3, pp. 723–733, 2008.

[16] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[17] *Amazon Web Service, http://aws.amazon.com/autoscaling*.

[18] *https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator*.