# Uncertainty-Aware Online Scheduling for Real-Time Workflows in Cloud Service Environment

Huangke Chen, Xiaomin Zhu, *Member, IEEE*, Guipeng Liu, and Witold Pedrycz, *Fellow, IEEE*

**Abstract**—Scheduling workflows in cloud service environment has attracted great enthusiasm, and various approaches have been reported up to now. However, these approaches often ignored the uncertainties in the scheduling environment, such as the uncertain task start/execution/finish time, the uncertain data transfer time among tasks, the sudden arrival of new workflows. Ignoring these uncertain factors often leads to the violation of workflow deadlines and increases service renting costs of executing workflows. This study devotes to improving the performance for cloud service platforms by minimizing uncertainty propagation in scheduling workflow applications that have both uncertain task execution time and data transfer time. To be specific, a novel scheduling architecture is designed to control the count of workflow tasks directly waiting on each service instance (e.g., virtual machine and container). Once a task is completed, its start/execution/finish time are available, which means its uncertainties disappearing, and will not affect the subsequent waiting tasks on the same service instance. Thus, controlling the count of waiting tasks on service instances can prohibit the propagation of uncertainties. Based on this architecture, we develop an unceRtainty-aware Online Scheduling Algorithm (*ROSA*) to schedule dynamic and multiple workflows with deadlines. The proposed *ROSA* skillfully integrates both the proactive and reactive strategies. During the execution of the generated baseline schedules, the reactive strategy in *ROSA* will be dynamically called to produce new proactive baseline schedules for dealing with uncertainties. Then, on the basis of real-world workflow traces, five groups of simulation experiments are carried out to compare *ROSA* with five typical algorithms. The comparison results reveal that *ROSA* performs better than the five compared algorithms with respect to costs (up to 56 percent), deviation (up to 70 percent), resource utilization (up to 37 percent), and fairness (up to 37 percent).

**Index Terms**—Workflow scheduling, uncertain, proactive and reactive strategies, cloud service

## 1 Introduction

Cloud computing has become a new service-based paradigm that cloud service providers deliver on-demand services (e.g., applications, platforms, and resources) for customers by the "pay-as-you-go" fashion [1]. In this paradigm, cloud service providers often provide many types of service instances, each type corresponds to different levels of services (e.g., core count, CPU frequency, storage size, memory size, etc.). Note that a service instance can refer to a virtual machine (VM), container, server, and alike [2]. Further, the available service instances in the platforms can be expanded and shrunk dynamically [3]. From the customers' perspective, the cloud service paradigm is scalable and cost-effective because they can access unlimited service instances on demand and just pay for the actual usages. With these advantages, cloud service is increasingly applied in various fields, such as retail industry, scientific experiments, and banking [4]. It is worth noting that the applications coming from these fields often consist of a series of data-dependent tasks [5], [6]. Since some tasks in these applications must wait for data from their predecessors, idle time slots between tasks will be inevitably left on service instances. Then, these idle time slots lead to a non-negligible number of poorly utilized service instances, resulting in low resource usage of cloud service platforms [7], [8].

It is imperative to design efficient scheduling approaches to address the foregoing issues for cloud service platforms. So far, substantial relevant approaches have been reported to schedule workflow applications in clouds. The obvious weakness of these approaches is that they assume that the accurate information on task execution time and data transfer time among precedence constrained tasks is available before scheduling. However, in real cloud service environment, the execution time of workflow tasks often cannot be accurately predicted [9], and the actual execution time can only be obtained after these tasks are completed [10], [11]. This can be contributed to the following two facts. First of all, each workflow task typically contains conditional instructions under different inputs [10], [12]. This can be further explained by that each task in workflows may compose of many selection and condition statements, which correspond to different program

- *H. Chen, X. Zhu, and G. Liu are with the College of Systems Engineering, National University of Defense Technology, Changsha, Hunan 410073, P. R. China. E-mail: {hkchen, xmzhu, laurelgp}@nudt.edu.cn.*
- *W. Pedrycz is with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB T6G 2V4, Canada, with the Department of Electrical and Computer Engineering, Faculty of Engineering, King Abdulaziz University, Jeddah 21589, Saudi Arabia, and also with the Systems Research Institute, Polish Academy of Sciences, Warsaw 01447, Poland. E-mail: wpedrycz@ualberta.ca.*

*Manuscript received 19 Mar. 2017; revised 1 Aug. 2018; accepted 17 Aug. 2018. Date of publication 21 Aug. 2018; date of current version 5 Aug. 2021. (Corresponding author: Xiaomin Zhu.)*
*Digital Object Identifier no. 10.1109/TSC.2018.2866421*

branches and loops. Different branches/loops bring different amounts of computations to a task, which also mean that the execution time of the task is quite different with different input data. Second, the performance of service instances in cloud service platforms fluctuates over time. The reason is that to improve the utilization of servers' hardware resources (e.g., CPU, network, I/O, etc.), multiple service instances will be collocated on one server to synchronously use the same resources. This way of resource sharing can easily lead to resource interference among different service instances, which further brings about fluctuation in the performance of service instances [11], [13].

*Motivation.* The uncertain factors of cloud service platforms inevitably result in substantial execution disturbances, such as the sudden arrival of new workflows, fluctuation in task execution time and fluctuation in data transfer time among tasks. In particular, the uncertainty in task execution time has a seriously negative impact on the effectiveness of the baseline schedules. When using tasks' execution time at the worst-case in baseline schedules, the workflow processing will waste a large amount of resources. The reason is that the actual execution time of workflow tasks is usually much shorter than their execution time at the worst-case. In this case, the idle time slots between assumed execution time of tasks and their actual execution time will be wasted in the service instances. When reserving too short time for a workflow task, its actual execution time will be much longer than the reserved time, which will successively postpone the completion of a series of workflow tasks, including its successors and those workflow tasks being allocated after these delayed tasks and their successors. Further, the successive postponements may cause the workflows to be completed later than their deadlines. To address the above problem, we strive to resolve the following two sub-problems: 1) how to mitigate the negative effects of uncertain factors on the baseline schedules; and 2) how to compress the idle time slots on service instances, such that reducing service renting cost and improving resource utilization of service instances, while satisfying the deadlines of dynamic workflows.

*Contributions.* The key contributions of this paper are outlined as follows:

- A novel uncertainty-aware architecture for executing dynamic workflows with uncertain task execution time and data transfer time in cloud service platforms.
- An unceRtainty-aware Online Scheduling Algorithm, namely *ROSA*, that skillfully integrates both the proactive and reactive strategies for scheduling service instances and dynamic workflows with deadlines.
- The experimental verification for the *ROSA* in the context of real-world workflow traces.

A preliminary result of our work was presented at the conference IEEE Cloud 2016 [14]. Compared with the conference version, this paper includes significant new contents: 1) the section of related studies is extended to include more recent relevant works; 2) the uncertainty in data transfer time is considered; 3) the lower bound of service renting cost for executing workflows is derived; 4) the computational complexity of *ROSA* is analyzed; 5) two visual examples are added to illustrate the proposed approach; 6) more experiments are added, including two more existing approaches for comparison, and a group of experiments.

TABLE 1
Notation Used in the Study

| Notation | Definition |
|---|---|
| $s_u^k$ | $k$th service instance with type $u$ |
| $Price(u)$ | price of service instance with type $u$ |
| $w_i$ | $i$th workflow in the workflow set $W$ |
| $a_i$ | arrival time of workflow $w_i$ |
| $d_i$ | deadline of workflow $w_i$ |
| $T_i$ | set of tasks in workflow $w_i$ |
| $t_{ij}$ | $j$th task in workflow $w_i$ |
| $E_i$ | set of directed edges among tasks in $w_i$ |
| $pred(t_{ij})$ | set of task $t_{ij}$'s immediate predecessors |
| $succ(t_{ij})$ | set of task $t_{ij}$'s immediate successors |
| $x_{ij,k}$ | mapping relationship between task $t_{ij}$ and service instance $s_u^k$ |
| $r(t_{ij})$ | index of the instance where $t_{ij}$ is assigned |
| $et_{ij,k}^{\alpha}$ | $\alpha$-quantile of the execution time of task $t_{ij}$ on service instance $s_u^k$ |
| $tt_{pj}^{i,\beta}$ | $\beta$-quantile of the data transfer time between task $t_{ip}$ and task $t_{ij}$ |
| $pst_{ij,k}$ | predicted start time of task $t_{ij}$ on service $s_u^k$ |
| $plst_{ij}$ | predicted latest start time of task $t_{ij}$ |
| $pft_{ij,k}$ | predicted finish time of task $t_{ij}$ on service $s_u^k$ |
| $plft_{ij}$ | predicted latest finish time of task $t_{ij}$ |

The outline of the paper is organized as follows. The scheduling architecture and the problem formulation are given in Section 2. Then, the proposed scheduling algorithm is described in Section 3, followed by the experimental verification of the *ROSA* in Section 4. The conclusions and one future direction are provided in Section 5. Due to the space limitation, the section of related studies and a part of experimental results and analysis are given in the supplementary materials, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSC.2018.2866421.

## 2 Modeling and Formulation

This section starts with the models of service instances and workflows, and then describes the novel uncertainty-aware scheduling architecture for cloud service platforms. On the basis of the architecture, the workflow scheduling problem is formulated. To improve the readability, the symbols used many times in this study are summarized in Table 1.

### 2.1 Service Instance Modeling

A cloud service platform generally provides a large number of service instances of different service types (e.g., VM types) [15], [16], [17]. The symbol $s_u^k$ is used to represent the $k$th service instance with service type $u$. Assuming that parameter $m$ represents the number of available service types, the symbol $u \in \{1, 2, \ldots, m\}$ refers to the index of the service type. Different service types in $\{1, 2, \ldots, m\}$ often associate with different prices and configurations. The symbol $Price(u)$ stands for the price of service type $u$, and the configurations of different service types are different in terms of the count of CPU cores, size of memory, size of storage, and network bandwidth.

In the cloud service environment, service instances are billed by the integer time units, and partial usage of a time unit also gives rise to the cost of one time unit. For instance, if the time unit is 60 minutes, using a service instance for 60.1 minutes brings about the bill of 120 minutes (two time units). Further, users can lease and release service instances at any time [3], [15], [18]. When a service instance is leased, it requires startup time (e.g., one minute) for it to be properly initialized and to be available for executing tasks. In this paper, we suppose that service instances will be released when they are idle (i.e., they complete all the mapped tasks and the data transfer) and its lease time reaches the integer multiple of the time unit.

In cloud service environment, service instances may be located at different geographical locations, and the network topologies supporting these service instances are complicated and heterogeneous. Without loss of generality, we use the symbol $l_{kl}$ to denote the network bandwidth between service instance $s_u^k$ and service instance $s_{u'}^l$.

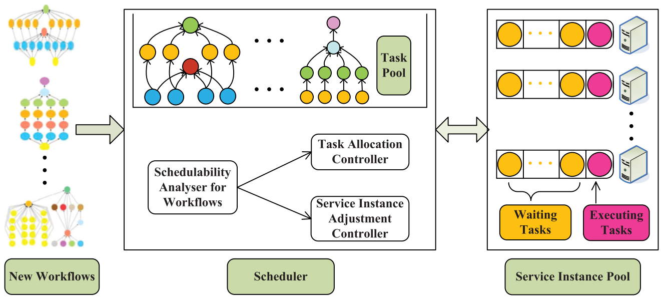
## 2.2 Workflow Modeling

In clouds, a wide range of customers dynamically submit their applications to the cloud service platform. Before submitting, the features of customers' applications (e.g., arrival time, deadlines, and structures, etc.) are unknown to the platform. When submitting the applications, the customers provide the features of their applications to the cloud platform. For a cloud platform, the dynamic workflows are denoted as $W = \{w_1, w_2, \dots, w_n\}$. We model a workflow $w_i \in W$ as $w_i = \{a_i, d_i, G_i\}$, where the $a_i$, $d_i$, and $G_i$ denote workflow $w_i$'s arrival time, deadline, and structure, respectively. For the structure $G_i$, we further model it as a directed acyclic graph (DAG), i.e., $G_i = (T_i, E_i)$, where $T_i = \{t_{i1}, t_{i2}, \dots, t_{i|T_i|}\}$ refers to a set of tasks, and the symbol $t_{ij} \in T_i$ corresponds to the $j$th task of $w_i$. In addition, $E_i \subseteq T_i \times T_i$ represents the set of directed edges among workflow tasks. A directed edge $e_{pj}^i \in E_i$ in the form of $(t_{ip}, t_{ij})$ exists if there exists data dependence between $t_{ip}$ and $t_{ij}$, where task $t_{ip}$ is called task $t_{ij}$'s immediate predecessor and the task $t_{ij}$ is called task $t_{ip}$'s immediate successor. The weight $w(e_{pj}^i)$ of edge $e_{pj}^i$ indicates the size of data being transferred from task $t_{ip}$ to $t_{ij}$. Since a workflow task may have more than one predecessor or successor, we let the $pred(t_{ij})$ and $succ(t_{ij})$ imply the set of all the immediate predecessor tasks and successor tasks for task $t_{ij}$. Due to the data dependencies among workflow tasks, a task $t_{ij}$ can only start executing after receiving the output data of all its predecessor tasks $pred(t_{ij})$.

Different from deterministic scheduling that assumes the execution time of workflow tasks and data transfer time among workflow tasks are deterministic, the uncertain scheduling considers the uncertainties in these relevant parameters. In this study, for simplicity reasons, these parameters are supposed to be independent and normally distributed random variables. After a workflow is submitted by a user, the distribution functions of the random variables can be gained.

## 2.3 Scheduling Architecture

To mitigate the negative effects of uncertain task execution time and data transfer time on the baseline schedules, we develop a novel uncertainty-aware architecture for executing workflows in the cloud service platform, which is illustrated in Fig. 1. The service platform can be divided into three modules: end-users, a pool of service instances, and a scheduler. For the end-users, they can submit workflow applications to the cloud service platforms at any time. To process the submitted applications, the cloud service platforms provide a pool of service instances, and these service instances can be expanded and shrunk dynamically. The scheduler is the bridge between the applications and the service instances, and its function is to schedule workflow tasks to service instances to optimize the predefined objectives, while satisfying the constraints coming from applications and platforms.


Task Pool
Schedulability Analyser for Workflows
Task Allocation Controller
Service Instance Adjustment Controller
Waiting Tasks
Executing Tasks
New Workflows
Scheduler
Service Instance Pool
Fig. 1. The uncertainty-aware scheduling architecture.

This work is interested in the scheduling strategy of the scheduler, and we mainly focus on the following components of scheduler: task pool (TP), schedulability analyser for workflows, task allocation controller, and service instance adjustment controller. Most of the waiting workflow tasks are placed in the TP, and the schedulability analyser is to map the waiting tasks in TP to service instances and to generate the plan of adjusting service instances. The adjustment plan of service instances comprises of when to lease or release the number of service instances of different types, and then the adjustment controller of service instances will execute the generated adjustment plan. Additionally, the task controller is responsible for dynamically allocating workflow tasks from TP to the selected service instances in accordance with the mappings of tasks and service instances.

**Definition 1.** *Ready task: a workflow task is defined as ready if this task has no predecessor, i.e., $pred(t_{ij}) = \emptyset$; or all the immediate predecessor tasks of this task have been mapped to service instances and at least one predecessor task has been completed.*

For this scheduling architecture, the overview of its operation process can be roughly classified as follows.

- When new workflows arrive, all the tasks in this workflow will be assigned corresponding ranks. Then, all the ready tasks coming from these workflows will be mapped to service instances as executing or waiting tasks immediately. If the available service instances are insufficient for these ready tasks, more service instances will be leased. Additionally, the other tasks (i.e., non-ready tasks) will be added to the task pool.
- The other case is when a workflow task is completed by a service instance. On the completion of this task,

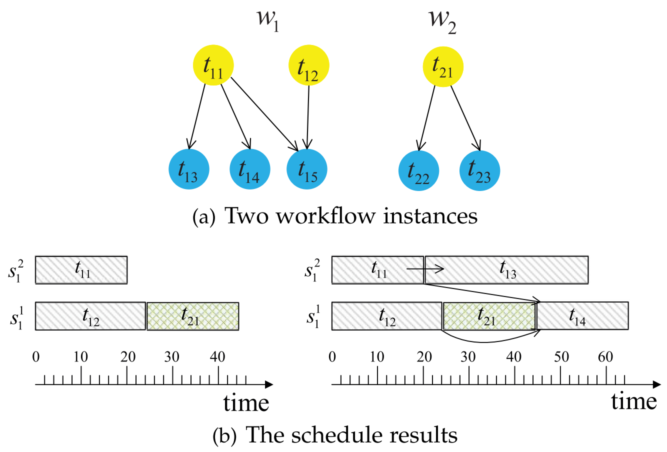


Fig. 2. Illustration the advantages of the scheduling architecture.

the first waiting task on the service instance will be executed immediately after receiving the output data of all its predecessor tasks. Additionally, the completion of a task may cause its successor tasks to become ready. Then, these ready workflow tasks are mapped to service instances immediately. Similarly, if the available service instances are insufficient for these ready tasks, more service instances will also be leased.

The novel feature of the proposed architecture is that only after the workflow tasks become ready are they mapped and allocated to service instances, and other non-ready waiting tasks are placed in the TP rather than on the service instances directly. The main advantages of the architecture can be summed up as follows.

- This design can mitigate the propagation of uncertainties throughout the schedules. As only the mapped workflow tasks are allocated to wait on service instances, the uncertainties of these mapped tasks can only be propagated to other mapped tasks on the same service instances. After these mapped tasks are completed, the uncertainty in their execution time will disappear, and these completed tasks have no effect on the subsequent tasks mapped to that service instances. Thus, this design can mitigate the propagation of uncertainties.
- It makes the computation and transmission overlap each other. When service instances are executing tasks, they can concurrently receive workflow tasks as waiting tasks and input data of waiting tasks. By doing so, computation and transmission can be efficiently overlapped to save time.
- This architecture enables a waiting task on a service instance to be executed as soon as the input data from all its predecessor tasks has been received and the tasks ahead of it on the same service instance have been completed, such that removing the possible execution delay for the waiting task.

To visually illustrate the above advantages of the uncertainty-aware scheduling architecture, an example is given in Fig. 2.

We assume two sample workflows, i.e., $w_1$ and $w_2$, arrive at the cloud platform at time 0. As illustrated in Fig. 2a, the workflow $w_1$ contains five tasks (i.e., $\{t_{11}, t_{12}, \ldots, t_{15}\}$), and workflow $w_2$ consists of three tasks $t_{21}$, $t_{22}$, and $t_{23}$. When these two workflows arrive, tasks $t_{11}$, $t_{12}$ and $t_{21}$ are ready, and will be scheduled to service instances immediately. Suppose that tasks $t_{12}$ and $t_{21}$ are scheduled to service instance $s_1^1$ while the task $t_{11}$ is scheduled to service instance $s_1^2$, which is described on the left side of Fig. 2b. As time goes on, task $t_{11}$ is finished in time 20, and then its successors $t_{13}$, $t_{14}$ and $t_{15}$ become ready based on Definition 1. At time 20, these three ready tasks will be scheduled. As shown on the right side of Fig. 2b, we assume that tasks $t_{13}$ and $t_{15}$ are scheduled to $s_1^2$, while task $t_{14}$ is scheduled to $s_1^1$. From the scheduling result on the right side of Fig. 2b, we can observe that at time 20, task $t_{11}$ has been finished, thus the uncertainty of its execution time does no exist and has no influence on the subsequent task $t_{13}$. In addition, service instance $s_1^1$ can receive the output data of task $t_{11}$ for task $t_{14}$ when executing tasks $t_{12}$ and $t_{21}$, which enables the overlapping of communications and computations. Moreover, when service instance $s_1^1$ finishes task $t_{12}$ at time 25, the waiting task $t_{21}$ on $s_1^1$ starts running right away, which is helpful for reducing the time slots between tasks.

## 2.4 Problem Formulation

**Definition 2.** *Suppose $x$ and $f(x)$ be a continuous random variable and its probability density function, respectively. Let $\alpha$ be a number between 0 and 1, i.e., $0 \le \alpha \le 1$. The $\alpha$-quantile for the distribution of $f(x)$, denoted by $x^\alpha$, can be defined by [19]*

$$\alpha = F(x^\alpha) = \int_{-\infty}^{x^\alpha} f(x)dx. \tag{1}$$

According to (1), $x^\alpha$ is the argument such that $\alpha$ of the area under the graph of $f(x)$ lies to the left of $x^\alpha$ and $(1-\alpha)$ lies to the right.

The assignment variable $x_{ij,k}$ is used to denote the mapping of task $t_{ij}$ and service instance $s_u^k$ in the cloud service platforms. The $x_{ij,k}$ equals 1 if task $t_{ij}$ is mapped to service instance $s_u^k$, otherwise, $x_{ij,k}$ is equal to 0, i.e.,

$$x_{ij,k} = \begin{cases} 1, & \text{if } t_{ij} \text{ is mapped to } s_u^k, \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

**Definition 3.** *The symbol $r(t_{ij})$ is defined as the index of the service instance where task $t_{ij}$ is mapped to. For example, when task $t_{12}$ coming from workflow $w_1$ is mapped to service instance $s_3^8$, then $r(t_{ij}) = 8$.*

When scheduling workflows, the execution time of their tasks and data transfer time among tasks are random variables, and we utilize their quantiles to approximate them. The symbol $et_{ij,k}^\alpha$ stands for the $\alpha$-quantile of the execution time of task $t_{ij}$ on service instance $s_u^k$. The symbol $tt_{pj}^{i,\beta}$ denotes the $\beta$-quantile of the data transfer time between task $t_{ip}$ and task $t_{ij}$. On the basis of Definition 2, $et_{ij,k}^\alpha$ and $tt_{pj}^{i,\beta}$ are all real numbers, and the basic arithmetic operations apply here.

Notably, the sum of $\alpha$-quartiles is not the $\alpha$-quartile of the sum, so are the max/min operations. For simplicity reasons, we define the predicted start/finish time of task $t_{ij}$ on service instance $s_u^k$ as follows. For the predicted start time $pst_{ij,k}$, it is defined as

$$pst_{ij,k} = \max\{pft_{lh,k}, ct + \triangle, \max_{t_{ip} \in pred(t_{ij})} \{pft_{ip,r(t_{ip})} + tt^{i,\beta}_{pj}\}\}, \tag{3}$$

where $pft_{lh,k}$ denotes the predicted finish time of $t_{lh}$ that is the last task on service instance $s^k_u$; $ct$ and $\triangle$ denote the current time and the schedule delay; $pred(t_{ij})$ represents the set of all the predecessor tasks of $t_{ij}$; $r(t_{ip})$ refers to the index of the service instance where task $t_{ip}$ is mapped to.

Then, the predicted finish time $pft_{ij,k}$ of task $t_{ij}$ on service instance $s^k_u$ is expressed as

$$pft_{ij,k} = pst_{ij,k} + et^{\alpha}_{ij,k}. \tag{4}$$

After mapping all the tasks in workflow $w_i$ to service instances, the predicted finish time $pft_i$ of $w_i$ in the baseline schedule can be written as

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

Once tasks are completed, their actual start/execution/finish time can be obtained, and we employ the symbols $rst_{ij,k}$, $ret_{ij,k}$ and $rft_{ij,k}$ to indicate the actual start time, actual execution time and actual finish time of task $t_{ij}$ on service instance $s^k_u$, respectively. For instance, we assume task $t_{11}$'s execution time on service instance $s^2_3$ be $et_{11,2} \sim N(120, 10^2)$s and the 0.9-quantile of $et_{11,2}$ can be calculated as $et^{0.9}_{11,2} = 132.8$s; the actual execution time $ret_{11,2}$ may be 125s when task $t_{11}$ is completed on service instance $s^2_3$. Additionally, the actual finish time of workflow $w_i$ refers to the maximal finish time of all its tasks, i.e.,

$$rft_i = \max_{t_{ij} \in T_i}\{rft_{ij,r(t_{ij})}\}. \tag{6}$$

For workflows with uncertain task execution time and data transfer time among tasks, whether their deadlines have been satisfied or not depends on their actual finish time, which makes the following constraint:

$$\max_{t_{ij} \in T_i}\{rft_{ij,r(t_{ij})}\} \le d_i, \qquad \forall w_i \in W, \tag{7}$$

where $\max_{t_{ij} \in T_i}\{rft_{ij,r(t_{ij})}\}$ is the makespan of the workflow $w_i$.

Due to precedence constraints among workflow tasks, a task can start to execute only after receiving the input data from all its predecessors. Then, we have the constraint

$$rst_{ij,k} \ge rft_{ip,r(t_{ip})} + rtt^i_{pj}, \quad \forall e^i_{pj} \in E_i, \tag{8}$$

where $rtt^i_{pj}$ represents the actual data transfer time between task $t_{ip}$ and task $t_{ij}$.

Subject to the constraints in Formulas (7) and (8), we first optimize the service renting cost for completing the workflows in $W$, which is formulated as

$$\text{Minimize} \sum_{k=1}^{|SI|} Price(u) \cdot P_k, \tag{9}$$

where $|SI|$ refers to the count of used service instances when executing the workflows in $W$, and $P_k$ indicates the count of time units of using service instance $s^k_u$.

Minimizing the deviation cost function is also an important optimization objective under uncertain computing environment [20]. We define this function as the average of weighted sum of the absolute fluctuations between workflows' predicted finish time in the baseline schedule and their actual finish time after executing the schedule, i.e.,

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

where $mc_i$ stands for the marginal cost of difference between the predicted and actual finish time. Furthermore, parameter $mc_i$ is defined as the reciprocal of the expected makespan ($1/em_i$) in the shortest schedule of workflow $w_i$. For a workflow $w_i$, we define its shortest schedule as scheduling all its tasks on the fastest service instances, and assuming all data transfer time among tasks be zero. The above definitions are reasonable. For different workflows, their marginal cost $mc_i$ should be different. In general, for a workflow with longer expected makespan $em_i$ in the shortest schedule, it can tolerate larger deviation. Thus its $mc_i$ should be lower, and the reciprocal of the expected makespan ($1/em_i$) can reflect this relationship.

The resource utilization has become another vital indicator to assess the performance of the cloud service platforms. Hence, we also strive to maximize average resource utilization for service instances, which is written as

$$\text{Maximize} \sum_{k=1}^{|SI|}(wt_k) / \sum_{k=1}^{|SI|}(tt_k), \tag{11}$$

where $wt_k$ and $tt_k$ denote the working time and the total leasing time of service instance $s^k_u$ during executing the workflows.

Apart from the resource utilization, the fairness of resource utilizations among different service instances is another optimization objective, which is defined as follows:

$$\text{Maximize } \frac{|SI| - 1}{\sum_{k=1}^{|SI|}(wt_k/tt_k - ARU)}, \tag{12}$$

where $ARU$ represents the average resource utilization of all the service instances, i.e., $ARU = \sum_{k=1}^{|SI|}(wt_k)/\sum_{k=1}^{|SI|}(tt_k)$.

## 3 Algorithm Design

This section first derives the lower bound of the renting cost for executing a set of workflows, and then analyses how to approximate the lower bound under the dynamic and uncertain cloud service environment. After that, to realize the of the function of the schedulability analyzer in Fig. 1, we propose an uncertainty-aware online scheduling algorithm, followed by its time complexity.

### 3.1 Lower Bound of Service Renting Cost

The service renting cost is the primary objective to be optimized when scheduling dynamic workflows with deadlines in cloud service environments. To assist the algorithm design in Section 3.2, we derive the lower bound of service renting cost for executing a set of workflows.

**Lemma 1.** *Suppose the workflow set is $W = \{w_1, w_2, \ldots, w_n\}$, and the available types of service instances are $U = \{1, 2, \ldots, m\}$, then the service renting cost for executing the workflow set $W$ has the following lower bound:*

$$\sum_{k=1}^{|SI|} C(u_k) \cdot P_k \geq \sum_{i=1}^{n} \sum_{j=1}^{|w_i|} \min_{u_k \in U} \{C(u_k) \cdot ret_{ij,k}\}, \tag{13}$$

*where $u_k$ denotes the type of the kth service instance, $C(u_k)$ represents the cost of service type $u_k$ per billing period, $P_k$ corresponds to the number of billing periods of the k-the service instance, and $ret_{ij,k}$ represents the actual execution time of task $t_{ij}$ on the kth service instance.*

*Note that the unit of tasks' real execution time is the same as the length of the billing time unit. For example, when the billing time unit is one hour, the unit of tasks' actual execution time is one hour.*

**Proof.** Assume $ret_l$ denotes the $l$th task's execution time on a service instance, and $\Delta_l$ is the time slot between the real finish time of the $l$th task and the real start time of $(l+1)$th task on the same service instance. If the $l$th task is the last task on the service instance, the $\Delta_l$ corresponds to the time slot between its real finish time and the released time of the service instance. Then, the working periods $P_k$ of a service instance can be divided into a series of sub-periods $\{ret_1 + \Delta_1, ret_2 + \Delta_2, \ldots\}$, so we have

$$\begin{aligned}\sum_{k=1}^{|SI|} C(u_k) \cdot P_k &= \sum_{k=1}^{|SI|} \left( C(u_k) \cdot \sum_{l=1}^{n_k} (ret_l + \Delta_l) \right) \\ &\geq \sum_{k=1}^{|SI|} \left( C(u_k) \cdot \sum_{l=1}^{n_k} ret_l \right),\end{aligned} \tag{14}$$

where $n_k$ denotes the number of tasks that are executed by the $k$th service instance.

The formula $\sum_{k=1}^{|SI|}(C(u_k) \cdot \sum_{l=1}^{n_k} ret_l)$ refers to the part of the renting cost when the service instances are running workflow tasks, and it can be rewritten as follows:

$$\begin{aligned}\sum_{k=1}^{|SI|} \left( C(u_k) \cdot \sum_{l=1}^{n_k} ret_l \right) &= \sum_{i=1}^{n} \sum_{j=1}^{|w_i|} (ret_{ij,k} \cdot C(u_k)) \\ &\geq \sum_{i=1}^{n} \sum_{j=1}^{|w_i|} \min_{u_k \in U} \{C(u_k) \cdot ret_{ij,k}\}.\end{aligned} \tag{15}$$

Therefore, the renting cost for executing the workflow set satisfies that $\sum_{k=1}^{|SI|} C(u_k) \cdot P_k \geq \sum_{i=1}^{n} \sum_{j=1}^{|w_i|} \min_{u_k \in U} \{C(u_k) \cdot ret_{ij,k}\}$, thus this lemma is proved. □

The Lemma 1 inspires us to optimize the service renting cost via the following two ways: 1) compressing the idle time slots for each service instance, which is similar to removing the parameter $\Delta_l$ in (14); and 2) minimize the cost for executing each task, which is consistent with the operation $\min_{u_k \in U}\{C(u_k) \cdot ret_{ij,k}\}$ in (15). With this two inspirations, we propose a heuristic in the next section to schedule real-time worklfows with uncertain task execution time and data transfer time among tasks.

### 3.2 Proposed Algorithm

In the cloud service environments, the workflow applications will be submitted by various customers, and the features of these workflows are unknown before being submitted. Only after the applications are submitted, their feature parameters (e.g., deadlines, DAG structures, etc.) are available, and their tasks can be mapped and allocated to the service instances for executing. In addition, due to the execution time of tasks is uncertain, the tasks' actual finish time is available only after they have been completed. Thus, this study treats the following two uncertain cases as disturbances: (1) the arrival of new workflows; (2) the completion of a task on a service instance. The above two disturbances may occur at any time. During the course of executions, when the first disturbance appears, all the new workflow tasks are either mapped to service instances or placed to the task pool by the first reactive scheduling strategy, which will be detailed in the form of Algorithm 1. In addition, when a service instance finishes a task suddenly (i.e., second disturbance happens), the ready tasks in task pool will be found out and mapped by the second reactive scheduling strategy, which will be described as Algorithm 2. Since Algorithms 1 and 2 use the same function to map ready workflow tasks, we extract this function, and describe it as Algorithm 3. In Algorithm 3, one proactive strategy is designed to generate baseline schedules, where $\alpha$-quantile of execution time of workflow tasks is considered and the $\alpha$ is very close to 1.

#### 3.2.1 Rank Tasks with Uncertain Execution Time

How to sort workflow tasks is one of the key issues in scheduling workflows. In this study, we rank the ready workflow tasks on the basis of their predicted latest start time $plst_{ij}$. For a task $t_{ij}$ in workflow $w_i$, we define its $plst_{ij}$ as the time, if the task $t_{ij}$ starts to execute after this time, the predicted finish time $pft_i$ of workflow $w_i$ is later than the deadline $d_i$.

Since the service instances in the cloud environments are heterogeneous, a task has different execution time on different service instances. In addition, the data transfer time among tasks also depends on the network bandwidth among the mapped service instances. Hence, obtaining the exact $plst_{ij}$ for each task $t_{ij}$ is impossible when scheduling them. In this paper, the minimal $\alpha$-quantile of execution time and the minimal $\beta$-quantile of data transfer time are used to estimate the $plst_{ij}$ before scheduling the workflows. The minimal $\alpha$-quantile of task execution time, denoted as $met_{ij}^{\alpha}$, and the minimal $\beta$-quantile of data transfer time of the data dependency $e_{pj}^{i}$, denoted as $mtt_{pj}^{i,\beta}$, are defined as

$$met_{ij}^{\alpha} = \min_{s_u^k \in SI} \{et_{ij,k}^{\alpha}\}; \tag{16}$$

$$mtt_{pj}^{i,\beta} = \min_{\substack{r(t_{ip}), r(t_{ij}) \in SI \\ r(t_{ip}) \neq r(t_{ij})}} \{tt_{pj}^{i,\beta}\}; \tag{17}$$

where symbol $SI$ is the set of service instances in the cloud platform.

Having the above definitions, we can define the predicted latest start time $plst_{ij}$ for each task $t_{ij}$ as follows.

**Definition 4.** *The $plst_{ij}$ for task $t_{ij}$ is recursively defined in the form*

$$plst_{ij} = \begin{cases} d_i - met_{ij}^{\alpha} - dt_{vm}, & \text{if } succ(t_{ij}) = \varnothing, \\ \min\limits_{t_{is} \in succ(t_{ij})} \{plst_{is} - mtt_{js}^{i,\beta}\} - met_{ij}^{\alpha}, & \text{otherwise,} \end{cases} \tag{18}$$

*where $succ(t_{ij})$ refers to the set of immediate successor tasks of $t_{ij}$; $dt_{vm}$ denotes the delay time for deploying a new service instance.*

For a task $t_{ij}$, its predicted latest finish time $plft_{ij}$ can be calculated as

$$plft_{ij} = plst_{ij} + met^{\alpha}_{ij}. \tag{19}$$

### 3.2.2 Description of the Proposed Algorithm

In order to clearly describe the process of our scheduling strategies, some rules are given below.

*Rule 1.* A service instance can execute at most one workflow task at any time point.

*Rule 2.* The waiting tasks on service instances start to execute as soon as the service instances become available and they have received the input data from all their predecessor tasks.

*Rule 3.* The service instances are released if they meet the following two requirements: 1) they are idle, i.e., they have completed all the workflow tasks that have been mapped to them and the data transfer; 2) their lease time reaches an integer multiple of a time unit.

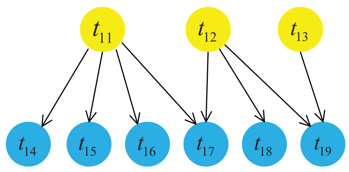
With regard to the traditional scheduling schemes [4], [10], [12], [21], when new workflows arrive, all the tasks in these workflows will be mapped and allocated to the local queues of service instances. By contrast, in this study, only when the tasks are ready will they be mapped and allocated to the service instance, and all the non-ready tasks are placed in the task pool. When new workflows arrive, the reactive strategy of *ROSA* is given in Algorithm 1.

**Algorithm 1.** *ROSA* - When New Workflows Arrive

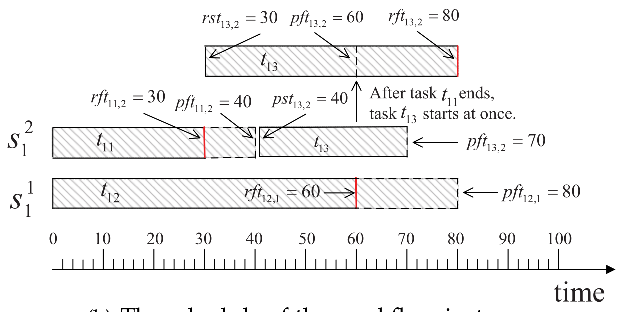
```
1:  $TP \leftarrow \emptyset$;
2:  while new workflows arrive do
3:      Compute $plst_{ij}$ and $plft_{ij}$ for all the tasks of new workflows as Formula (18) and (19);
4:      $RT \leftarrow$ Select the ready tasks from new workflows;
5:      Sort $RT$ by tasks' $plst_{ij}$ in a non-descending order;
6:      for each task $t_{ij} \in RT$ do
7:          Map task $t_{ij}$ using function MapReadyTask();
8:      end for
9:      Place the non-ready tasks in new workflows to set $TP$;
10: end while
```

The symbols $TP$ and $RT$ are used to record the tasks in task pool and the ready tasks, respectively. Once new workflows arrive, the proposed *ROSA* calculates the predicted latest start and finish time (i.e., $plst_{ij}$ and $plft_{ij}$) for all the tasks in new workflows (Line 3). Then, algorithm *ROSA* selects and sorts all the ready tasks from these new workflows (Lines 4-5). After that, the ready workflow tasks are mapped and allocated to service instances via the function *MapReadyTask()* (Lines 6-8). In addition, all the non-mapped tasks in these new workflows are placed to the task pool $TP$ (Line 9).

Due to the completion time of workflow tasks on service instances is uncertain, the completions of workflow tasks are regarded as disturbing events. Once these events appear, if the service instances have waiting tasks, the first waiting task will be executed immediately after receiving the input data from all its predecessor tasks, as shown in Rule 1. Moreover, after workflow tasks have been finished, their successor tasks may become ready, and these ready tasks will be mapped to service instances by algorithm *ROSA*. Fig. 3 depicts an example of the above scenes.



(a) A workflow instance



(b) The schedule of the workflow instance

Fig. 3. An example of tasks are completed.

As illustrated in Fig. 3a, the sample workflow $w_1$ has nine tasks, i.e., $\{t_{11}, t_{12}, \ldots, t_{19}\}$. As task $t_{11}$, $t_{12}$ and $t_{13}$ do not have any predecessors, when workflow $w_1$ arrives, these three tasks are ready and will be scheduled to service instances by Algorithm 1. We suppose that task $t_{11}$ and $t_{13}$ have been mapped to service instance $s_1^2$, and task $t_{12}$ is assigned to service instance $s_1^1$, as shown in Fig. 3b. In addition, the time unit in this example is supposed to be in seconds. For task $t_{11}$ and $t_{12}$, their predicted finish time is assumed to be $pft_{11,2}=40$ and $pft_{12,1}=80$; the predicted start and finish time of task $t_{13}$ are assumed to be $pst_{13,2}=40$ and $pft_{13,2}=70$, respectively. Since the finish time of these three tasks are random variables, their real finish time are not equal to their predicted finish time in most case and will only be available after they are finished. Assuming that task $t_{11}$ is finished suddenly at time 30 as time goes on, the completion of task $t_{11}$ is regarded as a disturbance, then task $t_{13}$ will be executed at once. Furthermore, the completion of task $t_{11}$ makes task $t_{14}$, $t_{15}$, $t_{16}$ and $t_{17}$ ready, and they will be mapped to service instances by algorithm *ROSA*.

Once a workflow task $t_{ij}$ is completed by a service instance $s_u^k$, the service instance will execute its waiting tasks, and then the tasks that just become ready will be selected and mapped, which is detailed by Algorithm 2. More specifically, on the completion of one executing task by service instance $s_u^k$, if the waiting tasks on this service instance are non-empty, the first waiting task starts to execute immediately after receiving the input data from all its predecessor tasks (Lines 1-3). Next, the algorithm *ROSA* selects the ready tasks among the successor tasks of task $t_{ij}$ (Lines 4-9), and removes the selected tasks from the task pool $TP$ (Line 10). After the selected tasks are sorted by their predicted latest start time $plst_{ij}$ (Line 11), these selected tasks will be mapped and allocated to service instances by function *MapReadyTask()*.

**Algorithm 2.** *ROSA* - When a Service Instance Completes a Task

```
1:  if service instance $s_u^k$ has waiting tasks then
2:      Start to execute the first waiting task on instance $s_u^k$;
3:  end if
4:  $RT \leftarrow \emptyset$;
5:  for $t_{is} \in succ(t_{ij})$ do
6:      if task $t_{is}$ is ready then
7:          $RT \leftarrow (RT \cup t_{is})$;
8:      end if
9:  end for
10: $TP \leftarrow (TP - RT)$;
11: Sort tasks in $RT$ by $plst_{ij}$ in a non-descending order;
12: for each task $t_{ij} \in RT$ do
13:     Map task $t_{ij}$ using function MapReadyTask();
14: end for
```

For a workflow task $t_{ij}$, this study defines its predicted service renting cost $pc_{ij,k}$ on service instance $s_u^k$ as

$$pc_{ij,k} = Price(s_u^k) \times (pft_{ij,k} - prt_k), \tag{20}$$

where $prt_k$ refers to the predicted time of service instance $s_u^k$ being available for task $t_{ij}$, and the $prt_k$ is calculated as follows: (1) if service instance $s_u^k$ is idle (i.e., both its executing and waiting tasks are empty), its $prt_k$ is the current time; (2) if service instance $s_u^k$ is not idle, its $prt_k$ corresponds to the predicted finish time of the last task on service instance $s_u^k$; (3) if service instance $s_u^k$ is just leased, its $prt_k$ is the beginning time of the lease.

The main steps of function *MapReadyTask()* is described by Algorithm 3.

**Algorithm 3.** *Function* MapReadyTask()

```
1:  $targetSi \leftarrow \emptyset$; $minCost \leftarrow +\infty$;
2:  for each service instance $s_u^k$ in the system do
3:      Calculate the predicted finish time $pft_{ij,k}$ as Formula (5),
        and the predicted cost $pc_{ij,k}$ as Formula (20) of task $t_{ij}$ on
        service instance $s_u^k$;
4:      if $pft_{ij,k} \le plft_{ij}$ && $pc_{ij,k} < minCost$ then
5:          $targetSi \leftarrow s_u^k$; $minCost \leftarrow pc_{ij,k}$;
6:      end if
7:  end for
8:  $u \leftarrow 1$; $types \leftarrow$ available service instance types;
9:  for each service instance type $t \in types$ do
10:     Calculate the predicted finish time $pft_{ij,k}$, and the
        predicted cost $pc_{ij,k}$ of task $t_{ij}$ on a new service
        instance with type $t$;
11:     if $pft_{ij,k} \le plft_{ij}$ && $pc_{ij,k} < minCost$ then
12:         $u \leftarrow t$; $targetSi \leftarrow \emptyset$; $minCost \leftarrow pc_{ij,k}$;
13:     end if
14: end for
15: if $targetSi$ != $\emptyset$ then
16:     Allocate task $t_{ij}$ to the service instance $targetSi$;
17: else
18:     Lease a new service instance $s_u^k$ with type $u$;
19:     Allocate task $t_{ij}$ to the new service instance $s_u^k$ after this
        service instance has been initiated;
20: end if
```

The function *MapReadyTask()* strives to map a task to a service instance to guarantee that the predicted latest finish time (i.e., $plft_{ij}$) of the task is less than its deadline while conserving cost. This function employs two steps to map a ready task to a service instance. For the step one, it selects one available service instance that completing the task before the $plft_{ij}$ and producing the minimum expected cost $pc_{ij,k}$ (Lines 4-6) for the ready task (Lines 2-7). After that, if step one is infeasible, the step two will be employed to lease a new service instance that producing the minimum expected service renting cost $pc_{ij,k}$ and completing this task before the $plft_{ij}$ (Lines 8-14). Next, this function will map the ready workflow task to a service instance with the minimal predicted cost (Lines 15-20).

### 3.3 Time Complexity of the Proposed *ROSA*

**Lemma 2.** *The time complexity of scheduling a workflow $w_i$ using algorithm* ROSA *is $O(n \cdot e + n \cdot log(n) + n \cdot s \cdot log(s) + n \cdot N_{vm} \cdot p)$, where $n$ and $e$ respectively represent the task number and edge number of workflow $w_i$; $s$ and $p$ are the maximum out-degree and in-degree of a task in $w_i$; $N_{vm}$ denotes the count of used service instances.*

**Proof.** Similar to the algorithm SHEFT [12], the time complexity of calculating the ranks for all the tasks in workflow $w_i$ is $O(n \cdot e)$ (Line 3, Algorithm 1). In algorithm *ROSA*, only when tasks are ready, they will be scheduled to service instances. Thus, for algorithm *ROSA*, other sources of time complexity lie in searching ready tasks, sorting ready tasks and scheduling ready tasks.

When workflow $w_i$ just arrives, searching the ready workflow tasks takes $O(n)$ (Line 4, Algorithm 1). When a task is completed, the time complexity of checking if its successors are ready is $O(s)$ (Lines 4-9, Algorithm 2). Hence, the time complexity of searching ready tasks while scheduling workflow $w_i$ is $O(n + n \cdot s) = O(n \cdot s)$.

To sort ready tasks in line 5 of Algorithm 1, its time complexity is $O(n \cdot log(n))$. Besides, it takes $O(n \cdot s \cdot log(s))$ to sort ready tasks (Line 11, Algorithm 2). Hence, the time complexity of sorting ready tasks is $O(n \cdot log(n) + n \cdot s \cdot log(s))$.

For scheduling ready tasks by function *ScheduleReadyTask()*, the selection of a service instance for a task costs $O(N_{vm} \cdot p)$ (Lines 2-7, Algorithm 3). In addition, it takes $O(m)$, where $m$ is the count of service instance types, to select a service instance type to lease (Lines 8-14, Algorithm 3). Hence, scheduling all the tasks in workflow $w_i$ by function *ScheduleReadyTask()* takes $O(n \cdot N_{vm} \cdot p + n \cdot m) = O(n \cdot N_{vm} \cdot p)$, since $m < N_{vm}$.

In conclusion, the time complexity of scheduling a workflow $w_i$ by the proposed *ROSA* is $O(n \cdot e + n \cdot s + n \cdot log(n) + n \cdot slog(s) + n \cdot N_{vm} \cdot p) = O(n \cdot e + n(n) + n \cdot slog(s) + n \cdot N_{vm} \cdot p)$, since $s \le e$. □

## 4 Experimental Studies

To test the effectiveness of the proposed *ROSA*, five groups of experiments are performed to compare its performance with other existing algorithms. As there exist no work reporting the similar algorithm, the compared algorithms are designed by modifying five previously available workflow scheduling algorithms: *EPSM* [22], *CWSA* [23], Uncertainty-Aware Evolutionary Scheduling Method (*UES*) [24],

TABLE 2
The Configuration and Prices for Service Instances

| Service type ($u$) | Type name | Price ($/h) | CPU (#) | Mem. (GB) | $F(u)$ |
|---|---|---|---|---|---|
| 1 | t2.small | 0.023 | 1.0 | 2.0 | 8.00 |
| 2 | t2.medium | 0.0464 | 2.0 | 4.0 | 4.00 |
| 3 | t2.large | 0.0928 | 2.0 | 8.0 | 2.00 |
| 4 | t2.xlarge | 0.1856 | 4.0 | 16.0 | 1.00 |

*FIFO* and *SHEFT* [12]. The brief explanations of these five algorithms are as follows.

*EPSM* is an online heuristic-based algorithm that focuses on minimizing the cost of renting service instances while meeting workflows' deadlines. Once new workflows arrive, this algorithm will give each workflow task a sub-deadline, and all the tasks are placed in a queue. Then, all the waiting tasks in the queue are scheduled in a periodic way. In each period, all the ready tasks in the queue are scheduled to the service instances with the minimal cost.

*CWSA* is also an online workflow scheduling algorithm that optimizes the finish time, tardiness, and cost for executing workflows. Once new workflows arrive, all the tasks in these workflows will be scheduled to the service instances. When scheduling a task, *CWSA* attempts to search a suitable time slot on a service instance for this task. If the above strategy is infeasible, this task will be appended on the end of former scheduled tasks on a service instance.

*UES* includes two stages: baseline schedule generation and evolutionary schedule during execution. In the first stage, all the tasks in a workflow will be grouped by a partial critical path strategy, then each workflow task is assigned with a sub-deadline. After that, a reverse-auction-based strategy will be used to recursively schedule all the workflow tasks to service instances with optimizing the execution cost within their sub-deadlines. In evolutionary scheduling, when one of the three types of uncertainties (performance degradation of service instance, failure of service instance, and addition of new service instances) happens, an uncertainty-aware strategy will be employed to renovate the baseline schedule.

*FIFO* is a default scheduling algorithm in Spark with standalone mode. When a workflow is submitted, the DAG-Scheduler splits the workflow graph into multiple stages of tasks. When each stage becomes ready, it will be submitted to TaskScheduler, and then the tasks in this stage will be scheduled according to the order of their ID.

*SHEFT* can be divided into two stages. The first stage is to give a rank for each workflow task. In the second stage, the task with better rank is preferentially mapped and allocated to a service instance with the minimum expected completion time.

As the *SHEFT* is originally developed to schedule tasks in one workflow, it is extended to support real-time workflow applications under unlimited service instances. First, when new workflows arrive, *SHEFT* is triggered to map all the tasks coming from these new workflows. If the available service instances are insufficient for the tasks, more service instances will be added. Second, service instances can only be deleted when they reach multiples of full hour operation and no tasks need them.

Moreover, the performance of the six algorithms is analyzed with respect to the following four indicators: service renting cost referring to (9), deviation referring to (10), resource utilization referring to (11), and fairness referring to (12).

## 4.1 Experimental Setup

In this section, we use a discrete simulator, which is realized by JAVA, to conduct the comparative experiments. The code of the proposed ROSA is released on the Website[1]. We assume the cloud service platform has 4 types of service instances and the main parameters of these service instances are summarized in Table 2. The instance specifications and pricing scheme are set by referring to four types of instances in Amazon EC2[2]. Like the most prominent cloud service providers, the service instances can be leased dynamically and their initialization time is set to 30 seconds [4], [17]. The network bandwidth of service instances is set to 1 Gbps. Additionally, the length of one bill time unit of these service instances is 60 minutes.

For the applications, the following four classes of scientific workflow applications are used: Montage (astronomy), CyberShake (earthquake science), SIPHT (bioinformatics), LIGO Inspiral (gravitational physics) [25]. For applications in one class, their structures are similar, but the sizes of their tasks are different, such as small (about 30), medium (about 50) and large (about 100). Then, three sizes of each class of workflows are chosen, and a total of 12 different workflows are used in the experiments. For each class of workflow, the structure of the workflow having the small size of tasks is shown in Fig. 4, which are extracted from the Pegasus Workflow repository [26]. We can observe that these four classes of workflows cover all the basic structural features with respect to composition and components (in-tree, out-tree, fork-join, and pipeline). Table 3 summarizes the main parameters of these workflow applications, such as the number of tasks, number of edges, average runtime of workflow tasks.

To simulate that the workflow applications arrive the cloud service platforms dynamically, one of the 12 applications will be selected randomly and submitted to the platform at a random time interval. Moreover, we assume that the interval between two successive applications is a random variable, and obeys the Poisson distribution with $\lambda$.

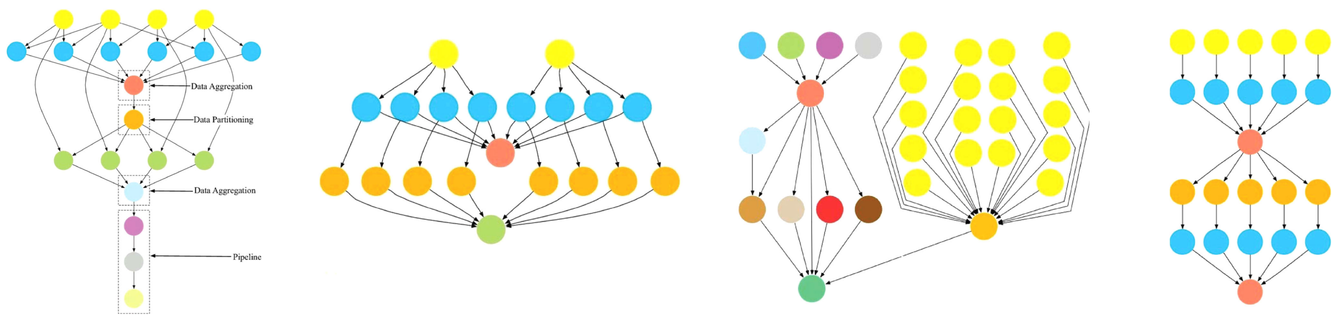
The symbol $bet_{ij}$ is used to denote task $t_{ij}$'s base execution time, which corresponds to the runtime of workflow tasks in the traces [26]. For this parameter, its cumulative distribution function (CDF) has been given in Fig. 5. Additionally, for the size of data among workflow tasks, its CDF is illustrated in Fig. 6.

Since there are various types of service instances in cloud service environments, one workflow task usually has different base execution time on different types of service instances. To simulate the above scene, the parameter $F(u)$ is employed, and the experiments calculate the base execution time for workflow task $t_{ij}$ on the service instance of type $u$ as

$$bet_{ij,k} = bet_{ij} \cdot F(u), \tag{21}$$

where $k$ stands for the $k$th service instance with the type of $u$.

1. https://github.com/HuangkeChen
2. https://aws.amazon.com/cn/ec2/pricing/on-demand/



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

Fig. 4. The structures of four real-world workflow applications.

For a workflow task $t_{ij}$, its execution time is assumed to follow normal distribution (i.e., $\widetilde{et}_{ij,k} = N(\mu_1, \delta_1)$), where the mean $\mu_1$ corresponds to task's base execution time and the standard deviation $\delta_1$ corresponds to task's relative base execution time, which are written as

$$\begin{aligned} \mu_1 &= bet_{ij,k}; \\ \delta_1 &= bet_{ij,k} \cdot varianceET, \end{aligned} \tag{22}$$

where $varianceET$ represents the variation of task execution time.

In the experiments, the actual execution time of workflow tasks are calculated as

$$ret_{ij,k} = random\_N(\mu_1, (\delta_1)^2), \tag{23}$$

where $random\_N(\mu, \delta^2)$ is used to obtain values that follows the normal distribution with the mean $\mu$ and the variance $\delta$. Note that the parameter $ret_{ij,k}$ is not available before scheduling.

We also assume the data transfer time among workflow tasks follows the normal distribution (i.e., $\widetilde{tt}^i_{pj} = N(\mu_2, \delta_2)$), with the mean $\mu_2$ and the variance $\delta_2$, which are written as

$$\begin{aligned} \mu_2 &= lat + w(e^i_{pj})/l_{r(t_{ip})r(t_{ij})}; \\ \delta_2 &= \mu_2 \cdot varianceTT, \end{aligned} \tag{24}$$

where $lat$ is the network latency; $w(e^i_{pj})$ denotes the size of data being transferred from task $t_{ip}$ to task $t_{ij}$; $l_{r(t_{ip})r(t_{ij})}$ denotes the bandwidth between service instance $r(t_{ip})$ and service instance $r(t_{ij})$; $varianceTT$ denotes the variation of data transfer time.

TABLE 3
The Parameters of the Workflow Traces

| Workflows | Number of Tasks | Number of Edges | Average Task Runtime (Sec) |
|---|---|---|---|
| Montage 25 | 25 | 45 | 9.10 |
| Montage 50 | 50 | 106 | 10.17 |
| Montage 100 | 100 | 433 | 10.58 |
| CyberShake 30 | 30 | 52 | 27.62 |
| CyberShake 50 | 50 | 92 | 26.46 |
| CyberShake 100 | 100 | 380 | 31.53 |
| Sipht 30 | 29 | 33 | 191.26 |
| Sipht 60 | 58 | 66 | 201.19 |
| Sipht 100 | 100 | 335 | 175.55 |
| LIGO Inspiral 30 | 30 | 35 | 220.56 |
| LIGO Inspiral 50 | 50 | 60 | 235.24 |
| LIGO Inspiral 100 | 100 | 319 | 206.12 |

Then, the actual data transfer time among workflow tasks can be calculated as follows:

$$rtt_{ij,k} = random\_N(\mu_2, (\delta_2)^2). \tag{25}$$

Before assigning deadlines to workflows, we define the shortest makespan $M_i$ for workflow $w_i$ as the completion time of this workflow when each of its tasks is executed by the fastest service instance and the data transfer time among tasks is ignored. Also, the parameter $deadlineBase$ is used to control the deadlines of workflows, and the deadline of workflow $w_i$ is set as

$$d_i = a_i + deadlineBase \cdot M_i, \tag{26}$$

where $a_i$ denotes the arrival time of workflow $w_i$.

For the main parameters in the experiments, their values are summarized in Table 4.

In addition, the parameters $\alpha$ and $\beta$ to approximate the execution time of workflow tasks and data transfer time among tasks are set as $\alpha = \beta = 0.9$.

In the experiments, for each group of settings, all the six algorithms are run 30 times independently.

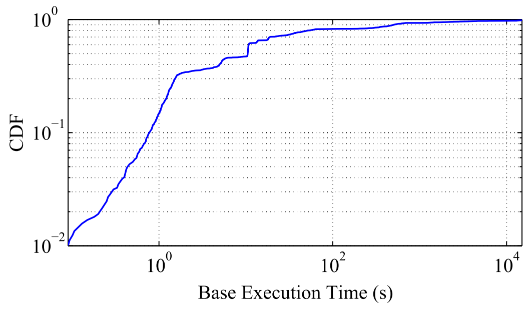


Fig. 5. CDF of task base execution time.

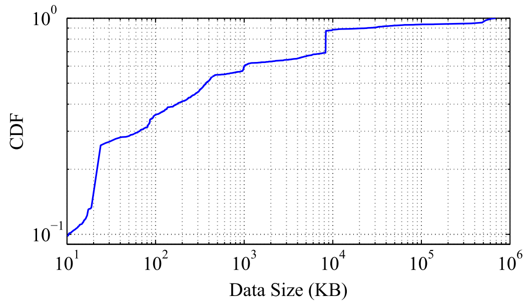


Fig. 6. CDF of the size of data among tasks.

TABLE 4
Parameters Used in Simulation Studies

| Parameter | Value (Fixed) – (Varied) |
|---|---|
| Workflow Count ($10^3$) | (4.0) – (1.0, 2.0, 3.0, 4.0, 5.0, 6.0) |
| $deadlineBase$ | (2.0) – (1.5, 2.0, 2.5, . . . , 10.0) |
| $\lambda$ | (0.1) – (0.01, 0.02, 0.03, . . . , 0.18) |
| $varianceET$ | (0.2) – (0.15, 0.2, 0.25, . . . , 0.4) |
| $varianceTT$ | (0.1) – (0.15, 0.2, 0.25, . . . , 0.4) |

### 4.2 Comparison with Lower Bounds

In this section, the parameters $deadlineBase$ and $\lambda$ are varied to compare the service renting cost of the six algorithms with the lower bound.

To observe the gaps between the six algorithms and the lower bounds in terms of the service renting cost, we vary the $deadlineBase$ from 1.5 to 10.0 with an increment of 0.5, and fix the workflow count, $\lambda$, $varianceET$ and $varianceTT$ as 4000, 0.1, 0.2, and 0.1. As illustrated in Fig. 7a that the service renting costs of the algorithms *ROSA*, *EPSM*, *CWSA*, *UES*, and *SHEFT* approach the lower bounds when increasing the $deadlineBase$, i.e., extending the deadlines of workflows. The reason is when extending the deadlines of workflows, the workflows can be completed later without violating their deadlines. Thus, cheaper service instances will be used to execute workflow tasks. In addition, more parallel tasks coming from the same workflows can be completed on the same service instances, and less service instances will be leased. As illustrated in Fig. 7a, the service renting cost of *ROSA* is less than *EPSM*, *CWSA*, *UES*, *FIFO*, and *SHEFT* on average by 10.07, 23.18, 28.90, 40.56, and 48.02 percent, respectively. The above results demonstrate that the proposed *ROSA*, incorporating both the proactive and the reactive strategies, has the advantage of reducing cost for executing dynamic workflows under uncertain cloud service environment. With respect to the service renting cost, the gap between *ROSA* and the lower bounds drops from 17.66 to 4.85 percent when increasing $deadlineBase$ from 1.5 to 10.0.

The Fig. 7b illustrates that increasing $\lambda$ compresses the gaps between the service renting cost of the six algorithms and the lower bounds. The reason is that the increase of $\lambda$ means that more workflows arrive at the cloud platform during each time unit. Thus, more non-dependent tasks will be scheduled, which is helpful for reducing the time slots on the service instances and the renting cost. Moreover, with the increase of $\lambda$ from 0.01 to 0.18, the cost of *ROSA* is less than *EPSM*, *CWSA*, *UES*, *FIFO*, and *SHEFT* on average by 8.14, 20.74, 27.94, 43.51, and 49.18 percent, respectively.

Due to the space limitation, other four groups of experimental results and analysis are presented in the supplementary materials, available online.

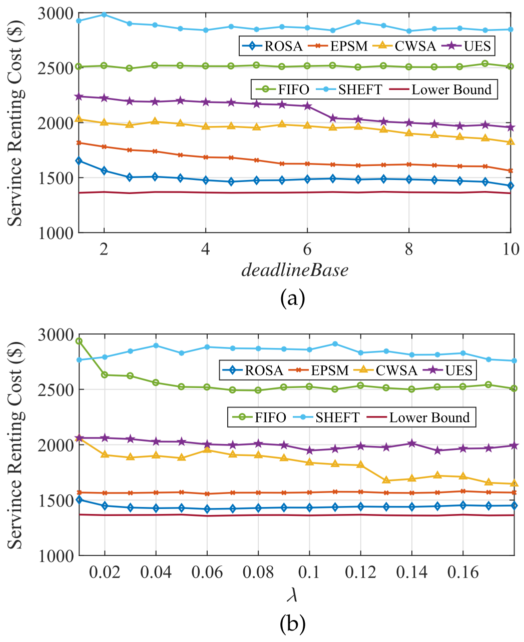


Fig. 7. Comparison with lower bounds.

## 5 Conclusions and Future Work

This study strives to optimize the service renting cost, deviation of schedules, resource utilization, and fairness of resource utilization for executing real-time workflows with uncertain task execution time and data transfer time in cloud service environment. We propose a novel uncertainty-aware architecture for cloud service platforms, and develop an uncertainty-aware scheduling algorithm *ROSA* to establish a good trade-off among cost, deviation, resource utilization, and fairness. Finally, in the context of real-world workflow traces, five groups of experiments have been carried out to verify the performance of the proposed algorithm.

Currently, cloud service is confronted with the problem of resource failure [16], [27]. Even worse, considering the enormous scale of inexpensive commodity computers operated by large cloud service providers, hardware failure is more often happening. Hence, the fault-tolerant and robust scheduling for workflows with uncertain task execution time is worth pursuing.

## Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grants 61872378, 61572511, and 91648204, in part by the China Postdoctoral Science Foundation under Grant 2016M602960 and Grant 2017T100796, in part by Science Fund for Distinguished Young Scholars in Hunan Province under grant 2018JJ1032.

## 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," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[2] H. Yan, H. Wang, X. Li, Y. Wang, D. Li, Y. Zhang, Y. Xie, Z. Liu, W. Cao, and F. Yu, "Cost-efficient consolidating service for Aliyun's cloud-scale computing," *IEEE Trans. Serv. Comput.*, 2016, doi:10.1109/TSC.2016.2612186.

[3] F. B. Charrada and S. Tata, "An efficient algorithm for the bursting of service-based applications in hybrid clouds," *IEEE Trans. Serv. Comput.*, vol. 9, no. 3, pp. 357–367, May/Jun. 2016.

[4] X. Zhu, J. Wang, H. Guo, D. Zhu, L. T. Yang, and L. Liu, "Fault-tolerant scheduling for real-time scientific workflows with elastic resource provisioning in virtualized clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 12, pp. 3501–3517, Dec. 2016.

[5] M. Vasile, P. Florin, N. Mihaela-Cătălina, and V. Cristea, "MLBox: Machine learning box for asymptotic scheduling," *Inf. Sci.*, vol. 433/434, pp. 401–416, 2018.

[6] G. Xie, J. Jiang, Y. Liu, R. Li, and K. Li, "Minimizing energy consumption of real-time parallel applications using downward and upward approaches on heterogeneous systems," *IEEE Trans. Ind. Informat.*, vol. 13, no. 3, pp. 1068–1078, Jun. 2017.

[7] H. Chen, X. Zhu, D. Qiu, L. Liu, and Z. Du, "Scheduling for workflows with security-sensitive intermediate data by selective tasks duplication in clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 9, pp. 2674–2688, Sep. 2017.

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

[9] Z. Cai, X. Li, R. Ruiz, and Q. Li, "A delay-based dynamic scheduling algorithm for bag-of-task workflows with stochastic task execution times in clouds," *Future Generation Comput. Syst.*, vol. 71, pp. 57–72, 2017.

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

[11] 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," *J. Syst. Softw.*, vol. 99, pp. 20–35, 2015.

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

[13] S. Verboven, K. Vanmechelen, and J. Broeckhove, "Network aware scheduling for virtual machine workloads with interference models," *IEEE Trans. Serv. Comput.*, vol. 8, no. 4, pp. 617–629, Jul./Aug. 2015.

[14] H. Chen, X. Zhu, D. Qiu, and L. Liu, "Uncertainty-aware real-time workflow scheduling in the cloud," in *Proc. IEEE 9th Int. Conf. Cloud Comput.*, 2016, pp. 577–584.

[15] Z. Cai, X. Li, and J. N. Gupta, "Heuristics for provisioning services to workflows in XaaS clouds," *IEEE Trans. Serv. Comput.*, vol. 9, no. 2, pp. 250–263, Mar./Apr. 2016.

[16] R. N. Calheiros and R. Buyya, "Meeting deadlines of scientific workflows in public clouds with tasks replication," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 7, pp. 1787–1796, Jul. 2014.

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

[18] X. Li, L. Qian, and R. Ruiz, "Cloud workflow scheduling with deadlines and time slot availability," *IEEE Trans. Serv. Comput.*, vol. 11, no. 2, pp. 329–340, Mar./Apr. 2018.

[19] J. L. Devore and K. N. Berk, *Modern Mathematical Statistics with Applications*. Boston, MA, USA: Cengage Learning, 2007.

[20] S. Van de Vonder, E. Demeulemeester, and W. Herroelen, "Proactive heuristic procedures for robust project scheduling: An experimental analysis," *Eur. J. Oper. Res.*, vol. 189, no. 3, pp. 723–733, 2008.

[21] B. P. Rimal and M. Maier, "Workflow scheduling in multi-tenant cloud computing environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 1, pp. 290–304, Jan. 2017.

[22] M. A. Rodriguez and R. Buyya, "Scheduling dynamic workloads in multi-tenant scientific workflow as a service platforms," *Future Generation Comput. Syst.*, vol. 79, pp. 739–750, 2018.

[23] B. P. Rimal and M. Maier, "Workflow scheduling in multi-tenant cloud computing environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 1, pp. 290–304, Jan. 2017.

[24] S. Meng, S. Wang, T. Wu, D. Li, T. Huang, X. Wu, X. Xu, and W. Dou, "An uncertainty-aware evolutionary scheduling method for cloud service provisioning," in *Proc. IEEE Int. Conf. Web Serv.*, 2016, pp. 506–513.

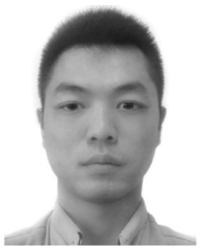
[25] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, "Characterizing and profiling scientific workflows," *Future Generation Comput. Syst.*, vol. 29, no. 3, pp. 682–692, 2013.

[26] [Online]. Available: https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator, 2014.

[27] Z. Wen, J. Cala, P. Watson, and A. Romanovsky, "Cost effective, reliable and secure workflow deployment over federated clouds," *IEEE Trans. Serv. Comput.*, vol. 10, no. 6, pp. 929–941, Nov./Dec. 2017.

**Huangke Chen** received the BS and MS degrees from the College of Information and System Management, National University of Defense Technology, China, in 2012 and 2014, respectively. Currently, he is working toward the PhD degree in the College of Systems Engineering, National University of Defense Technology. He was a visiting PhD student with the University of Alberta, Canada. His research interests include computational intelligence, large-scale parallel optimization, scheduling, resource management in cloud computing.

**Xiaomin Zhu** received the PhD degree in computer science from Fudan University, Shanghai, China, in 2009. He is currently an associate professor with the College of Systems Engineering, National University of Defense Technology, Changsha, China. His research interests include scheduling and resource management in distributed systems. He has published more than 90 research articles in refereed journals and conference proceedings such as the *IEEE Transactions on Computers*, the *IEEE Transactions on Parallel and Distributed Systems*, the *Journal of Parallel and Distributed Computing*, and ICDCS. He serves on the editorial boards of the *Future Generation Computer Systems* (FGCS) and the *AIMS Big Data and Information Analytics* (BigDIA). He is a member of the IEEE.



**Guipeng Liu** received the BS and MS degrees in information systems from the National University of Defense Technology, China, in 2015 and 2017, respectively. His research interests include big data processing and cloud resource management.

**Witold Pedrycz** (F'99) received the MSc, PhD, and DSc degrees from the Silesian University of Technology, Gliwice, Poland. He is a professor and Canada research chair (CRC-Computational Intelligence) with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, Canada. He is also with the Systems Research Institute, Polish Academy of Sciences, Warsaw, Poland. His main research interests include computational intelligence, fuzzy modeling and granular computing, knowledge discovery and data mining, fuzzy control, pattern recognition, knowledge-based neural networks, relational computing, and software engineering. He has published numerous papers in this area. He is intensively involved in editorial activities. He is the editor-in-chief of the *Information Sciences* and serves as an associate editor of the *IEEE Transactions on Fuzzy Systems*. He is also a member of a number of editorial boards of other international journals. He is a fellow of the IEEE.