# Real-time Multiple-Workflow Scheduling in Cloud Environment

**Xiaojin Ma** ( xjma@shu.edu.cn )
Shanghai University https://orcid.org/0000-0001-5793-4099
**Huahu Xu**
Shanghai University
**Honghao Gao**
Shanghai University
**Minjie Bian**
Shanghai University

**Research Article**

**Keywords:** Cloud Computing, Multiple Workflows, Online Scheduling, VM, Optimization

**Posted Date:** February 11th, 2021

**DOI:** https://doi.org/10.21203/rs.3.rs-170491/v1

# Real-time Multiple-Workflow Scheduling in Cloud Environment

Xiaojin Ma[a,d], Huahu Xu[a,b], Honghao Gao[a,c,*], Minjie Bian[b]

[a]School of Computer Engineering and Science, Shanghai University, Shanghai 200444, China
[b]Information Technology Office, Shanghai University, Shanghai 200444, China
[c]Gachon University, Gyeonggi-Do 461-701, South Korea
[d]School of Management, Henan University of Science and Technology, Luoyang 471000, China

**Abstract**—With the development of cloud computing, an increasing number of applications in different fields have been deployed to the cloud. In this process, the real-time scheduling of multiple workflows composed of tasks from these different applications must consider various influencing factors which strongly affect scheduling performance. This paper proposes a real-time multiple-workflow scheduling (RMWS) scheme to schedule workflows dynamically with minimum cost under different deadline constraints. Due to the uncertainty of workflow arrival time and specification, RMWS dynamically allocates tasks and divides the scheduling process into three stages. First, when a new workflow arrives, the latest start time and the latest finish time of each task are calculated according to the deadline, and the subdeadline of each task is obtained by probabilistic upward ranking. Then, each ready task is allocated according to its subdeadline and the increased cost of the virtual machine (VM). Meanwhile, only one waiting task can be assigned to each VM to reduce delay fluctuations. Finally, when the task is completed on the assigned VM, all the parameters of the relevant tasks are updated before allocating them to appropriate VMs. The experimental results based on four real-world workflow traces show that the proposed algorithm is superior to two state-of-the-art algorithms in terms of total rental cost, resource utilization, success rate and deadline deviation under different conditions.

**Keywords** Cloud Computing, Multiple Workflows, Online Scheduling, VM, Optimization

## 1 INTRODUCTION

Cloud computing facilitates IaaS, PaaS and SaaS modes by adopting virtualization technology, which has the advantages of heterogeneity, convenience and scalability and provides external services through the network [1]. In the IaaS model, users can customize the use of different types and amounts of computing resources on their actual needs, usually in the form of VM and pay based on rental hours. This mode, which is different from traditional computing platforms, not only avoids investment and maintenance costs by users for hardware but also greatly improves the utilization rate of infrastructure in cloud platforms, thus benefiting both users and cloud providers. Due to the advantages of this new computing mode, with its pay-as-you-go basis, efficiency and scalability, increasingly more applications requiring large-scale computing, such as weather forecasting, earthquake prediction and gene sequencing, are being migrated to cloud environments [2].

These different applications, which require large-scale computing, are typically represented as workflows in the cloud. A workflow is mainly modeled as a Directed Acyclic Graph (DAG), in which the vertices represent various tasks, and the edges represent the sequential relationships between tasks [3]. The weights of the vertices and edges represent the calculation quantity of the tasks and the transfer of data between tasks. As an NP-hard problem, workflow scheduling has always been an important research field [4]. Furthermore, workflow scheduling in cloud computing refers to assigning and executing tasks of workflow to appropriate VMs to meet user-defined Quality of Service (QoS) requirements. Because it is difficult to solve this NP-hard problem directly, different methods are usually adopted to obtain an approximate optimal solution. When scheduling workflows in grid, cluster and other environments, the most popular optimization objective is to shorten the completion time of the workflow as much as possible. Round Robin, Max-Min, Min-Min and other traditional scheduling algorithms have been adopted to achieve better solutions. Further, both reducing task execution time and considering implementation costs are important for workflow scheduling in the cloud. Moreover, given the fluctuations of hardware performances in cloud environments, the task execution time of workflows can be delayed, thereby requiring dynamic scheduling.

However, many studies on workflow scheduling fail to fully consider the characteristics of cloud computing, such as its heterogeneity, unlimited resources, and fluctuating performance. Most studies on this topic have emphasized the scheduling of a single workflow in a cloud, whereas few studies have focused on multiple-workflow scheduling in real time [5][6]. Due to the sharing method, tasks in multiple workflows will be implemented on the same resource simultaneously. Different deadlines for multiple workflows will make real-time scheduling in cloud computing more difficult than for single workflow. In addition, VM performance fluctuations caused by the hardware devices, network environment and other factors will affect the makespan and cost of workflow scheduling [7][8]. Therefore, various factors, such as the task execution time, data transfer time, and VM rental cost, need to be considered for multiple-workflow scheduling in cloud environments.

Given the nature of cloud computing, this paper pro-

*Corresponding author: Honghao Gao (e-mail: gaohonghao@shu.edu.cn).

poses a Real-time Multiple-Workflow Scheduling (RMWS) scheme to minimize the rental cost of leased VMs under deadline constraints. The main contributions of this paper are as follows:

1. An online scheduling algorithm for multiple workflows conforming to different applications in cloud computing is proposed.

2. All the characteristics of cloud computing such as dynamic expansion, heterogeneity, VM performance deviation, and bandwidth variation are considered to reduce their impacts on the makespan and cost of scheduling.

3. A simulation cloud environment is constructed, and extensive experiments are implemented based on real workflow traces. Via comparisons with two other state-of-the-art algorithms, it shows that the proposed algorithm in this paper achieves superior performance under various conditions.

The remainder of the paper is organized as follows: Section 2 surveys and analyses relevant works. Section 3 presents the platform structure and the scheduling problem. Then, the real-time multiple-workflow scheduling algorithm is proposed in Section 4, followed by the related experiments carried out in Section 5. Finally, conclusions and future work are given in Section 6.

## 2 Related Work

As an NP-hard problem, workflow scheduling on distributed resources has been studied extensively. In this section, the existing scheduling strategies are briefly reviewed and divided into three categories, i.e., scheduling in a distributed environment, single-workflow scheduling in a cloud and multiple-workflow scheduling in a cloud.

### 2.1 Scheduling in a Distributed Environment

Distributed system, which provides users with high-performance and high reliability resource sharing services, can be seen as an assemble of multiple individual compute nodes interconnected through the network [9][10]. A wide spectrum of approaches, such as integer-programming, Pareto optimality and graph theory, are used for scheduling in multiprocessor systems and distributed environments [11]. For instance, Topcuoglu et al. [12] proposed a Heterogeneous Earliest-Finish-Time (HEFT) algorithm to schedule tasks among a fixed number of heterogeneous processors. According to the average cost of computation and communication, the HEFT algorithm calculates the rank of each task and sorts them in descending order of priority. Then, tasks are successively selected and assigned to the appropriate processor, which can minimize the completion time of the selected task. In addition, some similar methods, such as RHEFT [13] and BHEFT [14], have been extended from HEFT for the scheduling optimization problem in different environments.

Moreover, the work in [15] introduced two schedulers based on integer linear programming to schedule VMs in grid resources and tasks on those VMs. Based on each task was exactly scheduled on a single host by the first scheduler, another scheduler applied relaxation techniques to shorten the execution time without significantly reducing the scheduling quality. Yu et al. [16] presented a cost-based workflow scheduling algorithm to minimize the execution cost under deadline constraints. They adopted task partitioning and total deadline assignment to generate the optimal execution scheme and rescheduling. To minimize the total execution cost of branch tasks, the Markov Decision Process (MDP) [17] approach is used to decide on which service to execute each task after the completion of its parent task. In another work, Yu et al. [18] adapted the Genetic Algorithm (GA) [19] to schedule workflows in utility grids. In their method, each individual is coded by a 2D string that includes the numbers of services and the order of tasks in each service. Then, the algorithm measures the quality of the individuals in the population based on the cost fitness and time fitness, which represent the optimization goal. Subsequently, the authors adopted random selection and exchange for the operations of crossover and mutation to generate new individuals.

Although many studies have used different methods to study the scheduling problem in distributed systems, most of them have focused on the single-user and single-workflow mode, and the main optimization goal is to reduce the task execution time in a relatively stable environment.

### 2.2 Scheduling for Single Workflows in a Cloud

Unlike the traditional computing model, cloud computing, given its characteristic on-demand procurement, heterogeneity, elasticity and dynamics, provides a simple and efficient platform for different applications but creates additional challenges to workflow scheduling. Correspondingly, many novel methods have been proposed to solve this problem.

Using the concept of critical path in DAGs, an algorithm called IaaS Cloud Partial Critical Paths (IC-PCP) [20] has been applied to workflow scheduling in cloud computing. After calculating the latest finish time of each task, IC-PCP recursively assigns tasks on the same partial critical path to the least expensive instance that can finish the chosen task before its latest finish time, and this process is repeated until all tasks are assigned. Calheiros et al. [21] used the idle time of leased resources and the remaining budget to mitigate the effects of performance variations in public clouds. Their method, which is an Enhanced IC-PCP with Replication (EIPR) algorithm, attempts to increase the likelihood of completing the execution of a workflow under a user-defined deadline via three distinct processes. The first is to combine task scheduling and provisioned resources, which means determining the number and type of VMs and the order and placement of tasks on VMs. The next step is adjusting the data transfer process to dictate the start and stop times of the VMs. Finally, EIPR attempts to increase the performance of task execution by replicating tasks on different VMs.

With the increasing complexity, proliferation, and ambition of cloud computing, heuristic algorithms, such as

Simulated Annealing (SA) [22], Ant Colony Optimization (ACO) [23], Particle Swarm Optimization (PSO) [24], and other nature-inspired methods, have been used for workflow scheduling [25]. Rodriguez et al. [26] adjusted the PSO to solve the scientific workflow scheduling problem with cost minimization and deadline constraints in a cloud environment. With the update of particle position and velocity, the task execution time and the data transfer time on different VMs were constantly calculated during the iteration process until the globally optimal solution was generated. In [27], a Temporal Task Scheduling Algorithm (TTSA) used hybrid simulated-annealing PSO to solve the cost minimization scheduling problem in hybrid clouds. The algorithm decomposes each arriving task into multiple parallelized subtasks to complete them within one time slot. The constraint cost minimization problem was converted into an unconstrained problem in each time slot. Based on the advantages and disadvantages of the SA and PSO algorithms, their method updates the position of each particle according to the metropolis acceptance criterion, which can avoid the algorithm falling into a local optimum.

### 2.3 Scheduling for Multiple Workflows in a Cloud

Compared with single-workflow scheduling, multiple-workflow scheduling in clouds is more complex. When applying various technologies to obtain feasible solutions, such scheduling not only must resolve the relationships among tasks in the same workflow but also should consider the structures of different workflows.

Malawski et al. [28] used exponential scoring to characterize the optimization problem to complete as many workflows as possible based on their priorities, budget and deadline constraints. In their first algorithm named Dynamic Provisioning Dynamic Scheduling (DPDS), VMs are created and terminated based on the complete billing cycle, budget and deadline constraints in the provisioning phase. And all tasks with different priorities are added to a queue and successively scheduled to idle VMs. Similarly, Wang et al. [29] proposed a Mixed-parallel Online Workflow Scheduling (MOWS), in which the scheduling phase is performed based on task prioritizing, waiting queue scheduling, task rearrangement, and task allocation. And four methods, named shortest-workflow first, priority-based backfilling, preemptive task execution and All-EFT task allocation, are used in these phases. Zhou et al. [30] proposed a Dyna scheduling system to minimize costs under user-specified probabilistic deadline constraints in an IaaS cloud. The Dyna system defines a hybrid instance configuration of a task as an n-dimension vector, which is used for the sequential execution of the task to meet the deadline. Then, their method applies a two-step approach to find an efficient solution while satisfying the probabilistic performance requirement.

Toward controlling the number of tasks directly waiting for each VM, Chen et al. [31] developed an Uncertainty-aware Online Scheduling Algorithm (ROSA) to schedule workflows in the cloud. ROSA obtains the latest start and finish times of all tasks in the new workflow according to the task execution time and data transfer time. Then, the algorithm allocates the ready tasks to service instances that minimize the expected cost under the deadline, while all the non-mapped tasks are moved to the task pool. If a service instance has more than one awaiting task, the first task will be executed after the service instance finishes a task. Liu et al. [32] proposed an Online Multiple-workflow Scheduling Framework named NOSF to schedule multiple workflows in real time. The algorithm can be divided into three parts: workflow preprocessing, VM allocation and feedback process. First, NOSF calculates the initial parameters of each task based on the structures and deadlines of the randomly arriving workflows. Then, the ready tasks are scheduled by the sub-deadline and increasing cost, while other tasks are stored in a task pool. Finally, the feedback process recalculates the original parameters of certain relevant tasks after a task has completed.

Though there are other scheduling strategies have been widely studied in the workflow scheduling problem [33-38], most of the existing scheduling algorithms focus on fixed resource allocation optimization whereas few can realize online scheduling. Meanwhile, almost all existing scheduling algorithms fail to completely incorporate all the characteristics of cloud, such as dynamic expansion, heterogeneity, VM performance deviation and bandwidth variation, or ignore the data transfer time between tasks of a workflow. Based on the shortcomings of these studies and our previous researches [39-42], this paper proposes a cost-efficient method inspired by the above-mentioned literature for the scheduling of multiple workflows in IaaS clouds. Considered all the characteristics of workflow scheduling in cloud computing, the proposed algorithm divides the online scheduling process into three phases and adjusts the priority of relevant tasks according to the feedback process of the completed tasks. The main processes are described below.

## 3 PROBLEM DEFINITION

This section introduces the model of cloud platforms and workflows and then describes the architecture of multiple-workflow scheduling in the cloud. Based on the model and architecture, the multiple-workflow scheduling problem is formulated. Some important symbols used in this study are provided in Table 1.

TABLE 1

NOTATION

| Notation | Definition |
|---|---|
| $v_m^k$ | the $m$th VM of type $k$ |
| $price(v^k)$ | the price of VM of type $k$ |
| $M$ | the number of leased VMs |
| $W$ | the workflow set |
| $I$ | the number of workflows in $W$ |
| $w_i$ | the $i$th workflow in $W$ |
| $a_i$ | the arrival time of workflow $w_i$ |
| $d_i$ | the deadline of workflow $w_i$ |
| $T_i$ | the set of tasks in workflow $w_i$ |
| $t_{ij}$ | the $j$th task in $T_i$ |
| $E_i$ | the set of edges among tasks in workflow $w_i$ |
| $e_{i,pj}$ | the directed edge between task$t_{ip}$ and $t_{ij}$ |
| $pred(t_{ij})$ | the set of all immediate predecessors of task $t_{ij}$ |
| $succ(t_{ij})$ | the set of all immediate successors of task $t_{ij}$ |
| $subd_{ij}$ | the subdeadline of task $t_{ij}$ |
| $bet_{ij}$ | the base execution time of task $t_{ij}$ |
| $aet_{ij}$ | the actual execution time of task $t_{ij}$ |
| $ast_{ij}$ | the actual start time of $t_{ij}$ |
| $btt_{i,pj}$ | the base data transfer time from task $t_{ip}$ to task $t_{ij}$ |
| $att_{i,pj}$ | the actual data transfer time from task $t_{ip}$ to task $t_{ij}$ |
| $ptt_{i,pj}$ | the predicted data transfer time from task $t_{ip}$ to task $t_{ij}$ |
| $lst_{ij}$ | the latest start time of $t_{ij}$ |
| $lft_{ij}$ | the latest finish time of $t_{ij}$ |
| $pst_{ij}^k$ | the predicted start time of $t_{ij}$ on VM $v_m^k$ |
| $pet_{ij}^k$ | the predicted execution time of task $t_{ij}$ on VM $v_m^k$ |
| $pft_{ij}^k$ | the predicted finish time of $t_{ij}$ on VM $v_m^k$ |
| $avail(v_m^k)$ | the available time of VM $v_m^k$ |
| $vlst(v_m^k)$ | the lease start time of VM $v_m^k$ |
| $vlft(v_m^k)$ | the lease finish time of VM $v_m^k$ |
| $TFT(w_i)$ | the finish time of workflow $w_i$ |
| $TEC(W)$ | the total execution cost of all workflows |

### 3.1 System Model

After users submit their workflows and requirements, the cloud platform allocates the tasks to appropriate resources and charges the user according to the usage time after the workflows are completed. In this process, the IaaS cloud platform delivers IT infrastructure based on virtual resources as a commodity to customers. These virtual resources providing services to end users are commonly measured in terms of CPU type, memory, storage, and network bandwidth; this allows VMs to be classified based on rankings and costs. A VM with a higher ranking provides a higher task execution speed; however, it is more expensive to rent. We use a weight $F(k)$ to represent the processing performance of VMs with different rankings. For a VM of type $k$, we define its weight $F(k)$ as the ratio of its task execution time to the task execution time of the VM with the highest level. Furthermore, a new VM must take time to boot before it can be used normally, and the shutdown time when a VM is released is ignored because of the negligible impact on scheduling.

In our proposed model, $v_m^k$ is used to represent the $m$th VM of type $k$, and $price(v^k)$ represents the rental price per unit time for a VM of type $k$. This means that a VM with a higher CPU speed usually has a higher rental price, and the rental time is generally billed in multiples of a unit of time. For example, if the unit of time is an hour, a VM leased for 1.3 hours will be charged for two time units.

### 3.2 Workflow Model

In this paper, different applications submitted by users in real time are presented in the form of a workflow, which constitutes a dynamic workflow set $W = \{w_1, w_2, \ldots, w_i\}$. Each workflow $w_i$ has a different arrival time $a_i$ and deadline $d_i$. Furthermore, workflow $w_i$ can be regarded as a DAG $G_i = (T_i, E_i)$, in which $T_i$ represents the set of tasks and $E_i$ represents the set of edges between tasks with directed relationships. In addition, $t_{ij}$ is used to represent the $j$th task in $w_i$. If tasks $t_{ip}$ and $t_{ij}$ have a direct dependence, which is represented by an edge $e_{i,pj}$, task $t_{ip}$ is called the immediate predecessor or parent task of task $t_{ij}$, while task $t_{ij}$ is called the immediate successor or child task of task $t_{ip}$. $Pred(t_{ij})$ and $succ(t_{ij})$ represent the set of all the immediate predecessors and all the immediate successors of task $t_{ij}$, respectively. Task $t_{ij}$ can only start executing after all the tasks in $pred(t_{ij})$ have been completed and after all the transferred data have been received.

Fig. 1 shows two workflows with different structures. For example, task $t_{25}$ has two immediate predecessors $t_{22}$ and $t_{23}$, which means that task $t_{25}$ cannot be executed unless tasks $t_{22}$ and $t_{23}$ are both finished and after all the output data have been received.

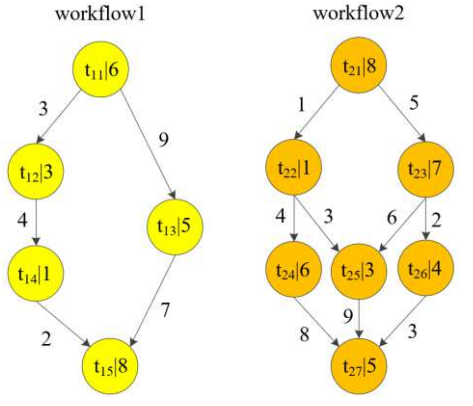



Fig. 1. Two example workflows.

### 3.3 Scheduling Architecture

The scheduling architecture in the cloud environment is shown in Fig. 2. Different Users may randomly submit new workflows with various deadlines to the cloud at any time. Then, the scheduler module assigns tasks of each arrived workflow to different VMs provided by the IaaS cloud. After all tasks in a submitted workflow have been completed, the result is returned to the corresponding user. Furthermore, the scheduler can be divided into four units: workflow initialization, task pooling, task assignment and VM management. The main processes of these four units are as follows:

• Workflow initialization: If a new workflow has arrived, the latest start time, the latest finish time and the

subdeadline of each task are calculated based on the deadline and structure of the workflow. After tasks with no immediate predecessors are marked as ready tasks and after other tasks are marked as waiting tasks, all tasks are sent to the task pool.

•Task pooling: All the tasks whereby their immediate predecessors have not been completed will be stored in the task pool and are regarded as waiting tasks. If all the immediate predecessors of a task are completed, the task status will be changed to the ready state.

•Task assignment: If there are ready tasks in the task pool, this process will allocate them to the appropriate VMs by the proposed algorithm. The algorithm first tries to assign tasks to the active VM that can satisfy the subdeadline and has the minimum cost. If all the active VMs cannot meet these requirements, a new VM will be leased by a VM manager.

•VM management: The process of leasing and releasing VMs is based on the execution of tasks. When existing VMs cannot meet the requirements of a ready task, a new suitable VM is leased. When all tasks on a VM are completed and after all data have been transferred, idle VMs will be released.

To minimize the fluctuations caused by task execution time delays, only one waiting task can be allocated on the same VM in each phase. Each VM can only execute one task at a time with a non-preemptive model. Furthermore, after a task is completed on the assigned VM, all parameters, such as the latest start time, the latest finish time, and the subdeadline of the relevant tasks in the task pool, are updated.

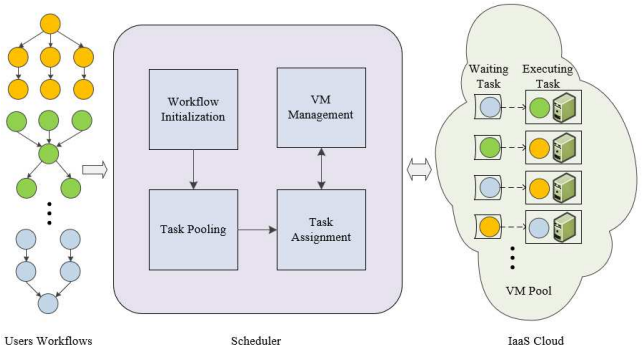



Fig. 2. The scheduling architecture in the cloud.

### 3.4 Problem Formulation

The purpose of the proposed algorithm in this paper is to schedule workflows in real time while minimizing the total rental cost under different deadline constraints. Based on the above model, we make the following definitions.

1) Task execution time and data transfer time. When task $t_{ij}$ is scheduled on VM $v_m^k$, the execution time of $t_{ij}$ depends on its workload and on the computing performance of $v_m^k$. Meanwhile, the data transfer time between $t_{ij}$ and its parent $t_{ip}$ depends on the size of the data transmitted and the bandwidth between VMs if $t_{ij}$ and $t_{ip}$ are assigned to different VMs. This means that the data transfer time will be zero if $t_{ij}$ and $t_{ip}$ are assigned to the same VM.

Moreover, the CPU performance of VMs and the bandwidth of the cloud cannot always be assumed constant, and fluctuations will cause uncertainty in the task execution time and the data transfer time. The task execution time is always regarded as a stochastic variable with independent and normal distributions [43][44]. For an expected value $\mu$ and variance $\sigma^2$, $N(\mu,\sigma^2)$ can denote the normal distribution of the variable task execution time. We define three parameters, $bet_{ij}$, $aet_{ij}$ and $pet_{ij}^k$, to represent the base execution time, the actual execution time and the predicted execution time, respectively. Furthermore, $bet_{ij}$ is the expected task execution time when a task is executed under the fixed computing performance of the VM with the highest level, $aet_{ij}$ is the actual task execution time, which can be obtained when the task is ready assigned to a VM with the computing performance at the corresponding time, and $pet_{ij}^k$ is the predicted task execution time if a task will be executed on a VM with weight $F(k)$. Similarly, we use $btt_{i,pj}$, $att_{i,pj}$ and $ptt_{i,pj}$ to represent the base data transfer time, the actual data transfer time and the predicted data transfer time, respectively.

2) Latest finish time $lft_{ij}$ and latest start time $lst_{ij}$. The latest finish time of task $t_{ij}$ is the time in which $t_{ij}$ can complete its execution on the fastest VM such that the finish time of workflow $w_i$ is no greater than its deadline $d_i$. The definition of the latest start time is similar. $lft_{ij}$ and $lst_{ij}$ are recursively obtained as

$$lft_{ij} = \begin{cases} d_i, if\ t_{ij}\ is\ the\ exit\ task\ t_{iexit} \\ \min\limits_{t_{is} \in succ(t_{ij})} \left\{ lst_{is} - btt_{i,js} \right\}, otherwise \end{cases} \quad (1)$$

$$lst_{ij} = lft_{ij} - bet_{ij} \quad (2)$$

3) Predicted start time $pst_{ij}^k$ and predicted finish time $pft_{ij}^k$. The predicted start time of task $t_{ij}$ on VM $vm_m^k$ depends on the arrive time of workflow $w_i$, the actual finish time of all predecessors of $t_{ij}$, the predicted data transfer time $ptt_{i,pj}$ between $t_{ij}$ and all its predecessors, and the available time of $vm_m^k$. The recurrence relations are

$$pst_{ij}^k = \begin{cases} a_i + avail(vm_m^k), if\ t_{ij}\ is\ the\ entry\ task\ t_{ientry} \\ max\left\{ avail(vm_m^k), \max\limits_{t_{ip} \in pred(t_{ij})} \left\{ ast_{ip} + aet_{ip} + ptt_{i,pj} \right\} \right\}, otherwise \end{cases} \quad (3)$$

$$pft_{ij}^k = pst_{ij}^k + pet_{ij}^k \quad (4)$$

where $avail(vm_m^k)$ and $ast_{ip}$ represent the available time of VM $vm_m^k$ and the actual start time of task $t_{ip}$, respectively.

4) VM available time $avail(vm_m^k)$. The available time of VM $vm_m^k$ is the time at which a new task can be executed on the VM. If task $t_{ij}$ is the waiting task on $vm_m^k$, the $avail(vm_m^k)$ will be updated to $pft_{ij}$. The $avail(vm_m^k)$ of a new leased VM is equal to the initial boot time (*init_time*).

5) VM lease start time $vlst(vm_m^k)$, VM lease finish time $vlft(vm_m^k)$ and VM rental cost $cost(vm_m^k)$. The lease start time of VM $vm_m^k$ is the time at which $vm_m^k$ is ready to execute tasks and is equal to $ast_{if}$ if task $t_{if}$ is the first executed task on $vm_m^k$. The VM lease finish time is the time at which $vm_m^k$ is released by the VM manager and can be seen as the actual finish time of the last task $t_{il}$ executed

on $vm_m^k$. Therefore, the rental cost of $vm_m^k$ can be calculated as follows:

$$cost(vm_m^k) = \left\lceil \frac{vlft(vm_m^k) - vlst(vm_m^k)}{time\_interval} \right\rceil * price(v^k) \quad (5)$$

$$\text{where } vlst(vm_m^k) = ast_{if}, vlft(vm_m^k) = ast_{il} + aet_{il}$$

6) The finish time $TFT(w_i)$ and total execution cost $TEC(W)$. The finish time of workflow $w_i$ is equal to the latest finish time of all tasks in $w_i$, and the total execution cost of all the workflows is the total rental cost for all of the leased VMs.

Finally, the problem of workflow scheduling can be defined as finding a feasible solution for the workflows such that $TEC(W)$ is minimized, and $TFT(w_i)$ does not exceed the deadline $d_i$ of workflow $w_i$. The problem is defined as

$$\text{Minimize } TEC(W) = \sum_{m=1}^{M} cost(vm_m^k) \quad (6)$$

$$\text{Subject to } TFT(w_i) = max(ast_{ij} + aet_{ij}) \le d_i \quad (7)$$

Based on Eqs. (6) and (7), the objective of multiple-workflow scheduling optimization in this paper is to minimize the overall rental cost of leased VMs while meeting the deadline constraints of each workflow. Furthermore, the decision variables in the optimization process can be seen as the unit rental price and the lease time of each VM. Therefore, when selecting the appropriate VM for each ready task, the start execution time of the task should be shortened as much as possible to minimize the rental cost by reducing the idle time of the allocated VM and improving the resource utilization.

## 4 THE PROPOSED ALGORITHM

In this section, the proposed algorithm, named RMWS, is presented in detail. We divide the algorithm into three stages: workflow initialization, task allocation and feedback processing. The detailed process of each stage is as follows.

### 4.1 Overview

According to the architecture of workflow scheduling in the cloud, the workflow scheduler can be divided into four main parts: workflow initialization, the task pool, task assignment and the VM manager. Based on this, the core of the proposed algorithm contains three processes: the initialization of the workflow, the allocation of tasks, and the feedback of task completion.

When a user submits a new workflow to the cloud platform, the workflow initialization process is triggered. The process preprocesses all tasks according to the arrival time, deadline and DAG structure, including the calculation of the latest start time, the latest finish time and the subdeadline of each task. Subsequently, the algorithm sends the ready tasks that have no immediate predecessors to the task allocation process, while other tasks are stored in the task pool. Then, the task allocation process first attempts to assign the selected task to an appropriate active VM. If all the active VMs fail to meet the conditions, a new VM will be leased to assign the selected task. When a task is finished, the latest start time, latest finish time and the subdeadline of its immediate successors for which their parent tasks have completed will be updated during feedback processing.

### 4.2 Workflow Initialization

The main purpose of workflow initialization is to determine the initial priority of tasks in the arriving workflow, including the main parameters such as the latest start time, the latest finish time, and the subdeadlines. The pseudocode is shown in Algorithm 1.

**Algorithm 1.** WorkflowInitialization($w_i$)

**Input:** An arriving workflow $w_i$ with arrival time $a_i$ and deadline $d_i$
**Output:** A queue $Q_r$ saving the ready tasks in $w_i$, the task pool $P_w$ contains the waiting tasks

1. $Q_r \leftarrow \emptyset$
2. **for** each task $t_{ij}$ **from** the exit tasks **to** the entry tasks
3. calculate $lst_{ij}$ and $lft_{ij}$ by Eqs. (1) and (2)
4. calculate $subd_{ij}$ by Eq. (12)
5. **endfor**
6. add all the entry tasks to $Q_r$
7. add all the other tasks *to* $P_w$
8. **Call** TaskAllocation($Q_r$)

First, when a new workflow arrives, by traversing all tasks in reverse order, the latest start times $lst_{ij}$ and the latest finish times $lft_{ij}$ of each task $t_{ij}$ are calculated according to Eqs. (1) and (2) (line 3).

Then, probabilistic upward rank is used to generate a subdeadline for each task (line 4) [45]. The upward rank of task $t_{ij}$ can be defined as the longest path from it to the exit task $t_{iexit}$, and is represented recursively as

$$ur_{ij} = \max_{t_{is} \in succ(t_{ij})} \left\{ ut_{is} + btt_{i,js} \right\} + bet_{ij} \quad (8)$$

Eq. (8) shows that the upward rank of a task is related to its execution time and data transfer time between the task and its immediate successors. However, when a task and its immediate parents are assigned to the same VM, the corresponding data transfer time is zero and will not affect the upward rank of the task. Therefore, a probabilistic variable is introduced to eliminate possible interference. The probabilistic upward rank $put_{ij}$ of a task $t_{ij}$ is defined as

$$pur_{ij} = \max_{t_{is} \in succ(t_{ij})} \left\{ put_{is} + \eta_{ij} * btt_{i,js} \right\} + bet_{ij} \quad (9)$$

$$\eta_{ij} = \begin{cases} 0, & if\ \theta^{-ccr_{ij}} > p \\ 1, & otherwise \end{cases} \quad (10)$$

$$ccr_{ij} = bet_{ij} / btt_{i,js} \quad (11)$$

where $\eta_{ij}$ is a Boolean variable indicating whether to consider the data transfer time. $\eta_{ij}$ can be calculated according to Eqs. (10) and (11), in which $ccr_{ij}$ is the computation-to-communication ratio, $\theta$ is a parameter greater than 1, and $p$ is a random number that generates in [0,1). The larger $ccr_{ij}$ is, the more probability $\eta_{ij}$ is 1, and vice versa.

According to the above definition, a larger $\theta$ means

that the algorithm is more likely to consider the data transfer time when calculating $put_{ij}$. The data transfer time will never be considered if $\theta$ is set to 1. Based on this, the subdeadline $subd_{ij}$ of task $t_{ij}$ can be calculated proportionally as

$$subd_{ij} = d_i * \frac{pur_{ientry} - pur_{ij} + bet_{ij}}{pur_{ientry}} \quad (12)$$

where $pur_{ientry}$ means the probabilistic upward rank of the entry task $t_{ientry}$.

Finally, all the entry tasks are added to the ready queue $Q_r$, while other tasks are stored in task pool $P_w$ (lines 6-7). After that, the ready tasks in queue $Q_r$ will be scheduled through the task allocation process (line 8).

### 4.3 Task Allocation

The task allocation process is the core of multiple-workflow scheduling. It not only determines the makespan and rental cost of each workflow but also affects the resource utilization in the cloud platform. If new ready tasks are generated after a new workflow has arrived or after an assigned task is completed, the task allocation process shown in Algorithm 2 will be triggered.

**Algorithm 2.** TaskAllocation($Q_r$)

**Input:** A queue $Q_r$ saving the ready tasks, a set of VMs of type 1 to K
**Output:** The assignment between tasks and VMs

1. sort the ready tasks in $Q_r$ by ascending order of the latest finish time
2. **while** $Q_r$ is not empty
3. $t_{ij}$←remove the first task from $Q_r$
4. $cost_{t_{ij}}$←+∞, $vm_{chosen}$←∅, $k_{chosen}$←0
5. **for** each active $vm_m^k$ has no waiting task
6. $pst_{ij}^k$←calculate by Eq. (3)
7. $pft_{ij}^k$←calculate by Eq. (4)
8. $cost(vm_m^k)_{old}$, $cost(vm_m^k)_{new}$←calculate by Eq. (5)
9. $pcost(t_{ij})$←$cost(vm_m^k)_{new} - cost(vm_m^k)_{old}$
10. **if** $pft_{ij}^k \le subd_{ij}$ and $pcost(t_{ij}) \le cost_{t_{ij}}$
11. $cost_{t_{ij}}$←$pcost(t_{ij})$
12. $vm_{chosen}$←$vm_m^k$
13. **endif**
14. **endfor**
15. **if** $vm_{chosen}$=∅
16. **for** $k$=1 to K
17. $pst_{ij}^k$←calculate by Eq. (3)
18. $pft_{ij}^k$←calculate by Eq. (4)
19. $pcost(t_{ij})$←$\left\lceil \frac{pet_{ij}^k}{time\_interval} \right\rceil * cost(v^k)$
20. **if** $pft_{ij}^k \le subd_{ij}$ and $pcost(t_{ij}) \le cost_{t_{ij}}$
21. $cost_{t_{ij}}$←$pcost(t_{ij})$
22. $k_{chosen}$←$k$
23. **endif**
24. **endfor**
25. **if** $k_{chosen}$=0
26. $k_{chosen}$←K
27. **endif**
28. $vm_{chosen}$←lease a new VM of type $k_{chosen}$
29. **endif**
30. **if** $vm_{chosen}$ is idle
31. execute $t_{ij}$ on $vm_{chosen}$ after receiving all transfer data from $pred(t_{ij})$
32. **else**
33. set $t_{ij}$ as the waiting task on $vm_{chosen}$
34. **endif**
35. **endwhile**

The main purpose of Algorithm 2 is to find suitable assigned VMs for the ready tasks to minimize the rental cost under the subdeadline constraints. First, all tasks in the ready queue $Q_r$ are sorted in ascending order according to the latest finish time (line 1). Then, the first task $t_{ij}$ in queue $Q_r$ will be selected in turn, and the predicted start time $pst_{ij}^k$, the predicted finish time $pft_{ij}^k$, and the predicted increased cost $pcost(t_{ij})$ of $t_{ij}$ on each active VM with no awaiting tasks are calculated (lines 3-9). Moreover, the predicted start time $pst_{ij}^k$ is the maximum time between the available time $avail(vm_m^k)$ and the data transfer time from $pred(t_{ij})$ to task $t_{ij}$ (line 6). The predicted finish time $pft_{ij}^k$ is the sum of the predicted start time $pst_{ij}^k$ and the predicted execution time of task $t_{ij}$ on the selected VM $vm_m^k$ (line 7). The predicted increased cost $pcost(t_{ij})$ is the difference in cost before and after assigning $t_{ij}$ to $vm_m^k$ (lines 8-9). The VM with the minimal increased cost under the subdeadline constraint of $t_{ij}$ will be selected (lines 10-13). Next, if the appropriate VM cannot be found from the active VMs, $pst_{ij}^k$, $pft_{ij}^k$, and $pcost(t_{ij})$ with different VM types are calculated (lines 17-19). A new VM of type $k_{chosen}$, on which $t_{ij}$ has the minimum rental cost under the subdeadline, or a new VM with the highest rank K, will be leased as the selected VM (lines 20-28). Finally, if the selected VM is idle, the task will begin to execute after receiving all the transferred data from its immediate predecessors (line 31); otherwise, the task will be the waiting task on the selected VM (line 33).

### 4.4 Feedback Processing

Algorithm 3 shows the feedback process, which will be triggered after a task $t_{ij}$ is completed on the assigned VM $vm_m^k$. First, the waiting task will be executed after receiving all the transferred data from its immediately parent (lines 2-4). Subsequently, for each immediate successor $t_{is}$ of $t_{ij}$, if all of $pred(t_{is})$ have been completed, the task will be considered a ready task (lines 5-6). After the latest start time, the latest finish time, and the subdeadline of $t_{is}$ are calculated, $t_{is}$ will be moved from $P_w$ into $Q_r$ (lines 7-8). Finally, the task allocation process is called for scheduling the tasks in $Q_r$ (line 11).

**Algorithm 3.** Feedback($t_{ij}$)

**Input:** A completed task $t_{ij}$ and its assigned VM $vm_m^k$, the task pool $P_w$
**Output:** A queue $Q_r$ saving the ready tasks from $succ(t_{ij})$

1. $Q_r$←∅
2. **if** there exists a waiting task on $vm_m^k$
3. execute the waiting task after receiving the transferred data
4. **endif**
5. **for** each task $t_{is} \in succ(t_{ij})$
6. **if** all of $pred(t_{is})$ have been completed
7. calculate $lst_{is}$, $lft_{is}$, and $subd_{is}$ by Eqs. (1), (2) and (12)
8. move $t_{is}$ from $P_w$ to $Q_r$
9. **endif**
10. **endfor**
11. **Call** TaskAllocation($Q_r$)

### 4.5 Algorithm analysis

According to the formal definition of multiple-workflow scheduling optimization problem in Section 3, the data transmission time between a task and its predecessors

should be considered while calculating the predicted start time of the task. Meanwhile, the tasks on the critical path plays a decisive role in the total execution time of the workflow. We use the concept of probabilistic upward rank to generate the subdeadline for each task on the critical path according to the deadline of workflow, and realize the constraints in Eq. (7). Based on these, the latest start time, latest finish time and subdeadline of ready tasks are calculated or updated respectively in Algorithms 1 and 3, and the appropriate VM is selected according to these indicators. For example, the VM with higher performance and higher rental cost will be selected when the task to be assigned has a tight subdeadline, and vice versa. Therefore, the rental cost of all leased VMs can be minimized under the deadline constraint of workflows by the real-time adjustment feedback of these three parameters.

In addition, we define the actual execution time of tasks as a random variable according to the performance fluctuation of VMs. It means that the exact execution time of tasks cannot be obtained by the processing performance of VMs before task allocation, which is also an important factor in the scheduling optimization problem. To reduce the fluctuation caused by the change of VM performance, only one waiting task is assigned to each active VM at the same time in Algorithm 2, which avoids the delay of subsequent tasks. Therefore, the proposed algorithm not only meets the subdeadline constraint of each task, but also optimizes the total rental cost by reducing the idle waiting time of active VMs.

### 4.6 Time Complexity

**Theorem1**. *The time complexity of scheduling an arriving workflow $w_i$ using the proposed algorithm RMWS is $O(n^2 + n \cdot log(n) + s \cdot p + p \cdot m \cdot n)$, where $n$ represents the task number, $p$ is the maximum in-degree of a task among $w_i$, $s$ is the maximum out-degree of a task among $w_i$, and $m$ denotes the number of active VMs.*

**Proof**. The scheduling process for an arriving workflow can be divided into three parts: calculating parameters, searching ready tasks, and assigning VMs. In the DAG composed of $n$ tasks, the maximum number of dependencies is $n(n-1)/2$, the time to calculate $lst_{ij}$, $lft_{ij}$ and $subd_{ij}$ of each task $t_{ij}$ is $O(n^2)$ (lines 2-5 in Algorithm 1 and line 7 in Algorithm 3). Hence, the time complexity of the first part is $O(n^2)$.

The search of ready tasks is performed when a new workflow arrives and when a task is completed. For the former, a task with no in-edges will be seen as a ready task, and it requires time $O(n)$ to traverse all tasks to find the ready tasks (line 6 in Algorithm 1). When a task $t_{ij}$ is completed, the algorithm requires time $O(s)$ to traverse all immediate successors. Moreover, the algorithm needs to determine if each immediate successor is ready by examining whether the immediate predecessors have been completed, which requires time $O(p)$ (lines 5-10 in Algorithm 3). Thus, the time complexity of searching for ready tasks is $O(n)+O(s \cdot p)$.

When assigning the ready tasks to VMs, the algorithm needs time $O(n \cdot \log(n))$ to reorder the tasks by merging sort [46] (line 1 in Algorithm 2). For each task $t_{ij}$ in $Q_r$, the algorithm traverses all $m$ active VMs and each immediate predecessor of $t_{ij}$ to calculate $pft_{ij}^k$, which requires time $O(p)$ (lines 5-14 in Algorithm 2). Thus, the corresponding time complexity is $O(p \cdot m \cdot n)$ while the maximum number of tasks in $Q_r$ is $n$. In addition, the time required to traverse each VM type can be regarded as a constant because the number of VM types is far less than that of leased VMs (lines 15-29 in Algorithm 2).

In summary, the time complexity of RMWS for scheduling an arriving workflow is $O(n^2) + O(n) + O(s \cdot p) + O(n \cdot \log(n)) + O(p \cdot m \cdot n) = O(n^2 + n \cdot \log(n) + s \cdot p + p \cdot m \cdot n)$. This proves the theorem.

## 5 Experiments

Based on the workflow scheduling model and algorithm proposed in this paper, a simulation platform is constructed to simulate the real-time multiple-workflow scheduling in the cloud. The performance of the proposed algorithm is analyzed based on the simulation results. The details of the experiments are discussed as follows.

### 5.1 Experiment Parameters

#### *5.1.1 Workflows*

Four real workflow traces, named LIGO, GENOME, Cybershake and SIPHT, are used as the experimental input data. These workflows have different structures and characteristics such as tree structures, pipelines, computing intensity, and I/O intensity. According to the number of tasks contained, we divide each type of workflow into small (approximately 50 tasks), medium (approximately 200 tasks) and large (approximately 1000 tasks) workflows. There are 12 different workflows used in the experiments. A simplified structure of each workflow is shown in Fig. 3, and a detailed description of these workflows is presented in [47].

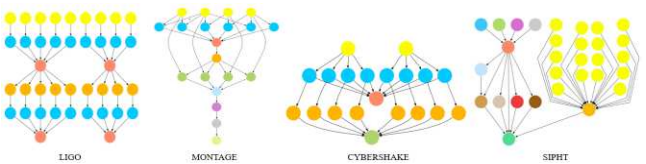



Fig. 3. The structures of the workflows.

Synthetic workflows with similar structures are created by workflow generator[1]. The generated workflows are represented in the form of a DAG in the XML (DAX) format, which contains tasks, dependency edges, transfer data and other information. Then, the experimental workflows can be constructed as DAX files.

#### *5.1.2 Baseline Algorithms*

To evaluate the performance of the proposed algorithm RMWS, we evaluated it along with ROSA [31] and NOSF [32], which are two online algorithms for multiple-workflow scheduling in the cloud.

ROSA estimates the task execution time and data transfer time by the quantile of a normal distribution. For

1 https://confluence.pegasus.isi.edu/display/pegasus/WorkflowHub.

randomly arriving workflows, the algorithm calculates the predicted latest start time and the predicted latest finish time of all tasks of the workflows according to their deadlines. Then, after the non-descending order of the predicted latest start time of the ready tasks is determined, a VM that has the lowest cost and satisfies the set conditions is selected to allocate each task.

NOSF uses a variance factor to simulate the normal distribution of the task execution time. After a new workflow arrives at a random time following a Poisson distribution, the algorithm first calculates the relevant parameters of all tasks such as the earliest start time, the earliest completion time and the subdeadline. Then, after sorting the tasks in a non-descending order according to the priority, the ready tasks are allocated to the VMs with the minimum idle time according to the predicted completion time and the predicted cost. When the assigned tasks are completed, the priority parameters of their relevant successors are updated.

#### 5.1.3 Evaluation Metrics

The core scheduling goal of multiple-workflow scheduling in the cloud is to minimize the rental cost of VMs under deadline constraints; thus, we use Eqs. (6) and (7) to obtain the total rental cost. In addition, to better evaluate the scheduling performance of the algorithms, other indicators are considered as follows:

1) Resource utilization

We use the proportion of the VM usage time in its lease time to express the resource utilization, which can be defined as

$$RU = \sum_{m=1}^{M} ET(vm_m^k) \Big/ \sum_{m=1}^{M} VLT(vm_m^k) \tag{13}$$

where $ET(vm_m^k)$ and $VLT(vm_m^k)$ represent the execution time and lease time of $vm_m^k$, respectively. Furthermore, $ET(vm_m^k)$ is equal to the sum of the execution times of all tasks assigned to $vm_m^k$, and $VLT(vm_m^k)$ is equal to the time from initial availability to the release of $vm_m^k$, which includes the execution time and the idle time.

2) Success rate

The success rate can be used to indicate the success and failure of workflow scheduling and can be defined as

$$SU = \tilde{I} / I \tag{14}$$

where $\tilde{I}$ represents the number of workflows that satisfied the deadlines, and $I$ is the number of workflows.

3) Deadline deviation

The arrive time, execution time and deadline of each workflow are used to reflect the average duration percentage of workflow deadlines exceeded, defined as

$$DD = \sum_{i=1}^{I} \frac{a_i + makespan_i - d_i}{d_i - a_i} \Big/ I \tag{15}$$

where $makespan_i$ represents the execution time of the workflow $w_i$, which can be obtained by subtracting the minimum task start time from the maximum task finish time of $w_i$.

4) Scheduling time

Scheduling time, which is calculated as the ratio of the total scheduling time to the total number of tasks, can be used to show the allocation efficiency of each task. For each task, the scheduling time mainly includes the time to calculate the priority parameters and the time to select the appropriate VM.

### 5.2 Experimental Setup

We implemented a discrete simulator using Java and deployed it in a computer with a 3.3 GHz Intel Core i5 processor and 12 GB of memory. The experiments simulated a cloud environment with a single datacenter and eight different VM types. The configurations and prices of the VMs, based on the Amazon EC2 services[1], are shown in Table 2. During the experiments, VMs of different types can be dynamically and indefinitely leased without any restrictions, and it is assumed that the VM can satisfy all workflow execution environments after initialization. Additionally, based on Amazon EC2, the boot time of a VM is set to 97 s, the bill unit time is 60 min, and the bandwidth between VMs is 1000 Mbps.

TABLE 2

VM TYPES USED IN THE EXPERIMENTS

| Type ($k$) | vCPU | Price ($) | $F(k)$* |
|---|---|---|---|
| t2.small | 1 | 0.023 | 3 |
| t2.medium | 2 | 0.0464 | 2.7 |
| m4.large | 2 | 0.10 | 2.5 |
| m4.xlarge | 4 | 0.20 | 2.2 |
| m4.2xlarge | 8 | 0.40 | 1.9 |
| m4.4xlarge | 16 | 0.80 | 1.6 |
| m4.10xlarge | 40 | 2.00 | 1.3 |
| m4.16xlarge | 64 | 3.20 | 1 |

**F(k) indicates the processing time weight of the corresponding VM type.*

To better simulate multiple-workflow scheduling in real-time environments, the relevant parameters are defined as follows:

•Workflow arrival time: Different workflows arrive at the cloud platform dynamically; thus, some of the 12 different workflows mentioned above are randomly selected and submitted to the platform at random intervals. Moreover, the arrival time interval is set to follow a Poisson distribution with $\lambda = 0.2$.

•Task execution time: Due to the performance deviations of the VM, the actual execution time of a task is assumed to follow a normal distribution. Furthermore, when task $t_{ij}$ is assigned to VM $vm_m^k$, the actual execution time can be expressed as

$$aet_{ij}^k = N(\mu_1, \delta_1) * F(k) \tag{16}$$

where $\mu_1$ corresponds to the expected value and $\delta_1$ means the standard deviation. They can be denoted as

$$u_1 = bet_{ij} \tag{17}$$

$$\delta_1 = bet_{ij} * \alpha \tag{18}$$

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

where the variance factor $\alpha$ represents the variation of the task base execution time. In the experiments, the actual execution time of $t_{ij}$ on $vm_m^k$ is obtained from a normal random number generator with the expected value of $\mu_1$ and the standard deviation of $\delta_1$.

•Data transfer time: Influenced by the network environment, the data transfer time between tasks is also considered to obey a normal distribution. Similar to the task execution time, the data transfer time in the experiments is obtained from a normal random number generator with the expected value of $\mu_2$ and the standard deviation of $\delta_2$. These values are defined as

$$att_{i,pj} = N(\mu_2, \delta_2) \tag{19}$$

$$u_2 = btt_{i,pj} \tag{20}$$

$$\delta_2 = btt_{i,pj} * \beta \tag{21}$$

where the variance factor $\beta$ represents the variation of the data transfer time.

•Workflow deadline: A benchmark deadline $D_b$ with the shortest makespan is used to assign deadlines to workflows. $D_b$ is defined as the makespan of a workflow when each workflow task is executed on a distinct VM of the fastest type, and the data transfer times among tasks are ignored. Then, the deadline of workflow $w_i$ can be calculated as

$$d_i = a_i + D_b * \gamma \tag{22}$$

where $\gamma$ denotes the deadline factor.

In addition, to make the result of the three algorithms more accurate and credible, the predicted execution time $pet_{ij}^k$ and the predicted data transfer time $ptt_{i,pj}$ in the task allocation processing are set as

$$pet_{ij}^k = (1+\alpha) * bet_{ij} * F(k) \tag{23}$$

$$ptt_{i,pj} = (1+\beta) * btt_{i,pj} \tag{24}$$

TABLE 3
PARAMETERS USED IN THE EXPERIMENTS

| Parameter | Fixed Value | Values |
|---|---|---|
| Workflow Number | 100 | (30, 50, 100, 200, 500,1000) |
| $\alpha$ | 0.2 | (0.1, 0.15, 0.2, …, 0.5) |
| $\beta$ | 0.2 | (0.1, 0.15, 0.2, …, 0.5) |
| $\gamma$ | 4.0 | (1, 2, 3, …, 8) |

In the experiments, we evaluate the performance of algorithms using different values of the main parameters, which are summarized in Table 3. The value of $\theta$, which is used in Eq. (10), is set to 1.5 after the tests on RMWS. For each group of settings, all three algorithms are run 10 times independently to calculate the average results.

### 5.3 Results and Analysis

#### *5.3.1 Variance of Task Execution Time*

To analyze the performance of the scheduling algorithm under varying task execution times due to fluctuations in VM performance and other factors, the variance factor $\alpha$ is increased from 0.1 to 0.5 in steps of 0.05 while holding the other parameters fixed. The experimental results for each evaluation metric of the three algorithms are shown in Fig. 4.

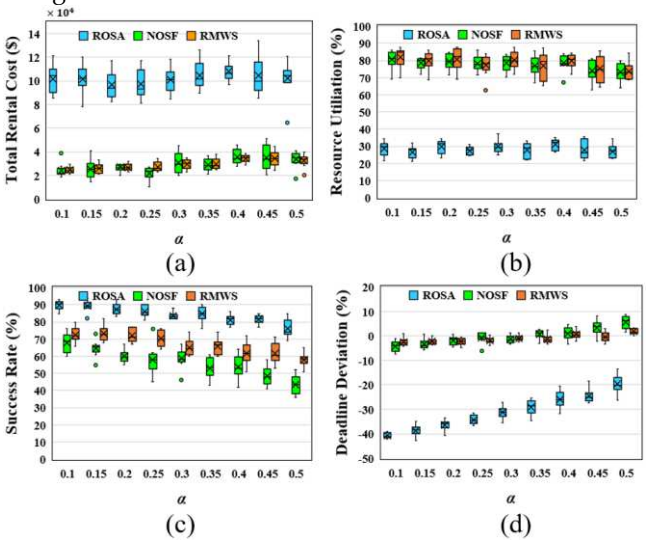



Fig. 4. The impact of variance in task execution time.

Fig. 4 (a) shows the change in the total rental cost under each algorithm under different $\alpha$ values. With an average value of $10.2*10^4$, ROSA consistently has a high cost, which is nearly 3.5 times higher than the costs of the other two algorithms. Although the average costs of NOSF and RMWS are close to $2.91*10^4$ and $2.95*10^4$, respectively, the cost fluctuation range of NOSF is larger than that of RMWS. In contrast, RMWS can maintain a fixed cost range; in particular, when the value of $\alpha$ exceeds 0.3, the average cost is approximately 2.17% lower than that of NOSF, and its cost advantage is greater. This also shows that the proposed RMWS algorithm is less susceptible to the impact of task execution time fluctuations than ROSA and NOSF. Fig. 4 (b) shows the resource utilization performance of the algorithms under different $\alpha$ values. ROSA shows significantly poor efficiency, being closely related to its task allocation strategy. Both NOSF and RMWS have high resource utilization rates of over 77%, which are much higher than the value of 28.44% of ROSA. This shows the obvious performance advantages of NOSF and RMWS in terms of resource utilization. Fig. 4 (c) shows the success rate results of the three algorithms. With increasing $\alpha$, the success rate of each algorithm decreases gradually. This is because when the task execution time increases, it will be difficult to complete the workflow within the deadline. From the results, ROSA always maintains a high success rate, which is mainly due to its large use of higher performance VMs with high lease costs. When $\alpha = 0.1$, the success rate of RMWS is approximately 7.08% higher than that of NOSF. However, with increasing $\alpha$, the success rate of RMWS gradually increases, and the average value is nearly 20.1% higher than that of NOSF. In particular, when $\alpha = 0.5$, the average success rate of RMWS is 58.1%, which is significantly higher than the average success rate of 43.5% of NOSF. Fig. 4 (d) shows the deviation from the workflow dead-

lines. Although ROSA achieves the shortest workflow execution time under different $\alpha$ values, it cannot reasonably allocate tasks to VMs according to the deadline, which is an important cause of the high execution cost and low resource utilization of the algorithm. In contrast, the deviation between the makespan and deadline of the workflows in the other two algorithms is significantly reduced. Among them, the average deviation value of NOSF when $\alpha \in [0.3,0.5]$ is 1.86%, whereas RMWS shows a superior value of -0.06%. The overall deviation value range of RMWS is significantly smaller than other algorithms.

As shown in Fig. 4, when the variance factor $\alpha$ of the task execution time changes, RMWS can obtain better overall results for various indicators such as the total rental cost, success rate, resource utilization and deadline deviation. The experimental analysis shows that the algorithm proposed in this paper has the best scheduling performance among the three algorithms under varying task execution times.

### 5.3.2 Variance of Data Transfer Time

Fig. 5 shows the performance of each algorithm when the data transmission time variance factor $\beta$ changes in the range of 0.1-0.5 and other parameters are fixed.

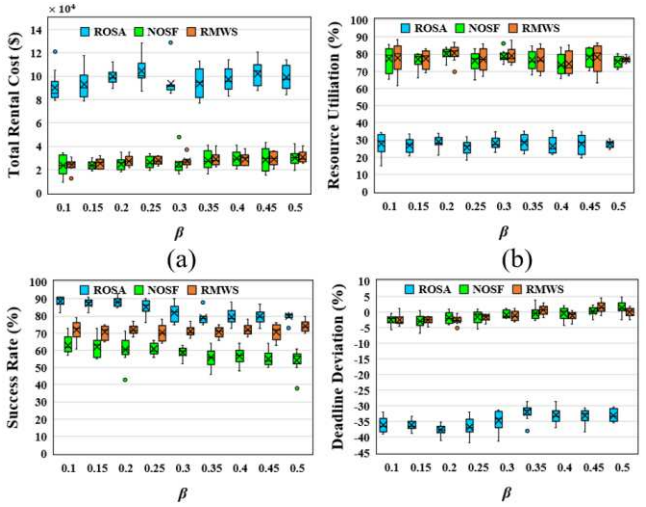



Fig. 5. The impact of variance in data transfer time.

Fig. 5 (a) and Fig. 5 (b) reflect the variation of the rental cost and resource utilization of the three algorithms under different $\beta$, respectively. All the algorithms still demonstrate their performance characteristics, as shown in Fig. 4 (a) and Fig. 4 (b). Specifically, ROSA always maintains the highest total rental cost and the lowest resource utilization, while the average cost and resource utilization of NOSF and RMWS are similar. However, Fig. 5 (a) shows that the cost of these algorithms under different $\beta$ is relatively stable and has not shown an upward trend under variations of the variance factor $\alpha$. This means that compared with fluctuations in the task execution time, changes in data transmission times caused by bandwidth instabilities and other reasons have a lower impact on workflow scheduling. Fig. 5 (c) shows the success rate with different $\beta$. When $\beta = 0.1$, the average success rates of ROSA, NOSF and RMWS are 88.6%, 63% and 71.8%, respectively. These values become 79.8%, 54.4% and 74.1%, respectively, when $\beta = 0.5$. Compared with the declining trend of the other two algorithms, RMWS maintains a relatively gentle change in the success rate. This is mainly because the proposed algorithm considers the data transmission time when calculating the subdeadlines of the tasks, while ROSA and NOSF do not consider this factor. Compared to Fig. 4 (d), the deadline deviation of ROSA does not change significantly in Fig. 5 (d), which is also slightly reflected in NOSF and RMWS. This further explains that the variation in the data transmission time will impact the cost, resource utilization, success rate and other indicators when scheduling workflow in the cloud; however, its impact is significantly weaker than the performance change provided by fluctuations in the task execution time. Especially in a high-bandwidth environment, this situation will be more obvious.

### 5.3.3 Variance of Deadline Factor

In addition to the rental cost, the deadline is another key optimization target in workflow scheduling. To reflect the performance of each algorithm under different deadline constraints, we increase the deadline factor $\gamma$ from 1 to 8 in step 1, and the evaluation indicators are shown in Fig. 6.

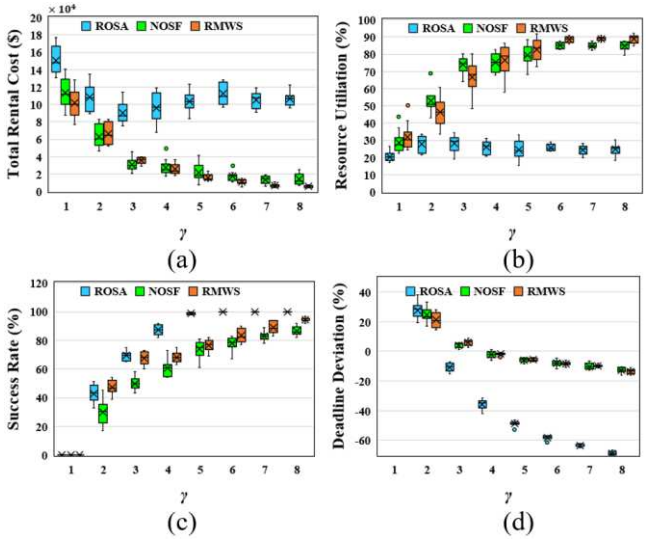



Fig. 6. The impact of variance in the deadline factor.

Fig. 6 (a) shows that when $\gamma$ increases from 1 to 4, the total rental costs of ROSA, NOSF and RMWS decreased extensively, by approximately 36.29%, 75.83% and 74.75%, respectively. This shows that each algorithm will give priority to assign tasks to low-performance VMs to reduce the lease cost when the deadline is further away. However, the cost of ROSA does not change significantly when $\gamma \in [5,8]$, and the average value is $10.7*10^4$. In comparison, both NOSF and RMWS achieve lower costs. When $\gamma$ increases from 5 to 8, RMWS, with an average cost of $0.96*10^4$, outperforms NOSF, with an average cost of $1.67*10^4$. In Fig. 6 (b) and Fig. 6 (c), the resource utilization rate and success rate of these algorithms are shown. Although the success rate of ROSA under the loose deadline constraint ($\gamma \in [5,8]$) is close to 100%, its resource utilization rate is low (24.88%). This is because ROSA fails

to fully utilize existing VMs when scheduling tasks; more tasks are allocated to the new VMs. However, the resource utilization rate and success rate of NOSF and RMWS follow an upward trend. Overall, RMWS achieves a better resource utilization rate and success rate than NOSF. Moreover, the difference is more pronounced when the deadline factor $\gamma$ is greater than 4. Under the loose deadline constraint ( $\gamma \in [5,8]$ ), for NOSF and RMWS, the average resource utilization rates were 83.66% and 87.44%, respectively, and the average success rates were 80.28% and 85.8%, respectively. Fig. 6 (d) shows the deadline deviation of the algorithms under different $\gamma$. The deadline deviations with abnormal values are not marked when $\gamma = 1$. With increasing deadline factor, the deadline deviation of ROSA increases sharply, as it uses many high-performance and new VMs to reduce the makespan of the workflow. NOSF and RMWS also show trends of increasing deadline deviation when $\gamma \in [1,4]$; however, this situation is alleviated under the condition of $\gamma \in [5,8]$. This shows that the two algorithms can effectively allocate VMs for tasks in workflow scheduling under various deadline constraints.

### 5.3.4 Variance of Workflow Count

In the cloud environment, the scale of the workflows submitted by users is uncertain; thus, we use different counts of workflows to simulate workflow scheduling on different scales. Fig. 7 shows the performance of each algorithm under different workflow counts.

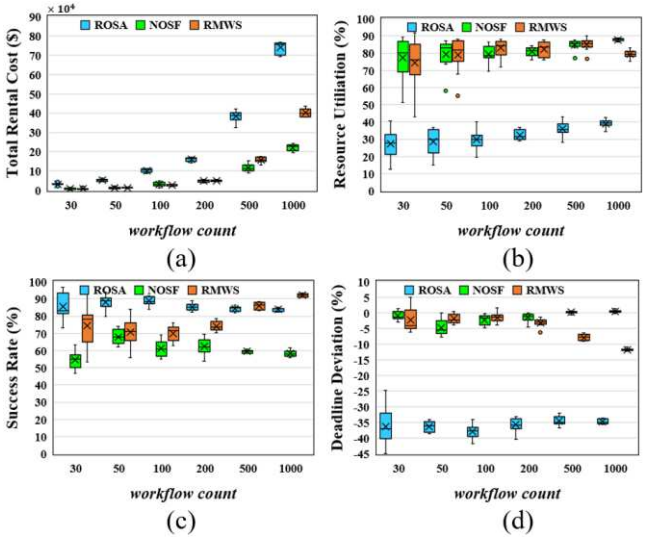



Fig. 7. The impact of variance in the workflow count.

Fig. 7 (a) shows that as the number of workflows increases, the total cost of the three algorithms increases due to the need for more VMs to execute tasks. The cost of ROSA is highest among the algorithms due to the lease of greater numbers of high-performance VMs; NOSF and RMWS have similar lease costs, with $1.68*10^4$ and $1.55*10^4$, respectively, on a small scale (workflow count is no more than 100). When the workflow count is greater than 100, the average cost of RMWS exceeds that of NOSF by approximately 58.42%, mainly because RMWS allocates additional tasks to high-performance VMs to meet the deadline. This is verified in Fig. 7 (b) and Fig. 7 (c). In the comparison of the resource utilization shown in Fig. 7 (b), the average value of RMWS is 80.63%, which is significantly better than the average value of 32.11% of ROSA and slightly lower than the 81.62% of NOSF. A more pronounced difference is evident in the comparison of success rate shown in Fig. 7 (c). Although ROSA has the highest average success rate, 86.06%, among the algorithms under the different numbers of workflows, it is accompanied by an overall high rental cost and low resource utilization. In contrast, the success rate of RMWS is approximately 4.12% higher than NOSF when the workflow count is 50 and reaches 58.22% when the workflow count is 1000. In the comparison of deadline deviations shown in Fig. 7 (d), ROSA obtains the maximum average value, -35.94%, with a lower workflow makespan than the other algorithms, and the average value of NOSF is -1.37%, which is slightly better than that of RMWS, -4.68%. In general, when the number of workflows is less than 500, RMWS can achieve performance better than or similar to that of the other two algorithms with respect to the four factors. Although RMWS is worse than NOSF in terms of rental cost, resource utilization and deadline deviation when the workflow count increases from 500 to 1000, it is more focused than NOSF on meeting the deadline constraints and thus has the highest success rate, which is particularly important in the scheduling process.

### 5.3.5 Discussion of algorithm performance

The comparisons of total rental cost, resource utilization, success rate and deadline deviation among the algorithms under various conditions reveal that ROSA achieves the best success rate but has far higher cost than NOSF and RMWS and consistently low resource utilization. ROSA allocates tasks according to the predicted completion time and the predicted cost of tasks. However, ROSA does not update the predicted completion time of subsequent tasks after each task is completed, which increases the uncertainty of this parameter. In addition, the start time of all waiting tasks will be delayed when the performance of the allocated VM is degraded. Therefore, ROSA has to lease VMs with higher performance to perform subsequent tasks, which will affect the related optimization objects of workflow scheduling.

NOSF effectively reduces the idle time and lease cost of the used VMs by continuously updating the subdeadline of the tasks and assigning only one waiting task to a VM at the same time. Compared with ROSA, its performance in most aspects, e.g. rental cost, resource utilization, and so on, has been significantly improved. However, NOSF ignores the change of the rental cost of each selected VM in the task allocation phase, and only relies on the estimated execution time of the assigned task to obtain the minimum cost, which cannot be determined as the optimal allocation. Moreover, when updating the subdeadline of the tasks, NOSF fails to sufficiently consider the impact of the data transmission time, which is also an important factor in workflow scheduling. Therefore, the scheduling performance of NOSF will be affected if the bandwidth performance fluctuates.

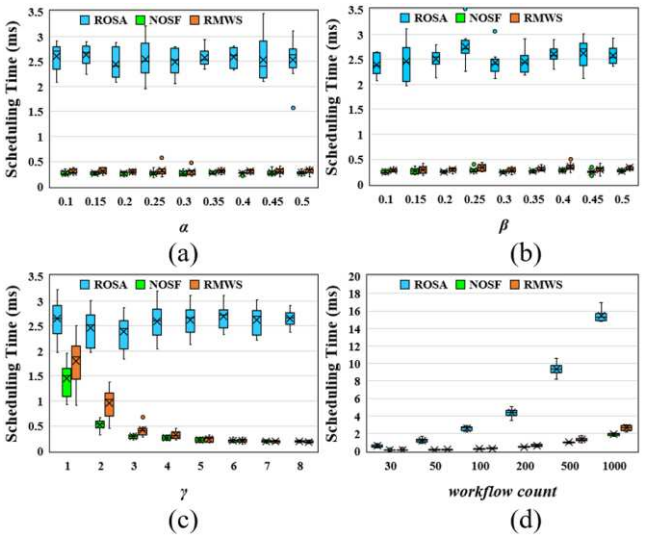



Fig. 8. The average scheduling time of each task.

By comparison, we consider all the characteristics of workflow scheduling in cloud environment, the proposed RMWS uses the strategy of probabilistic upward ranking to calculate the subdeadline of each task and takes the data transmission time as a variable. Afterwards, RMWS uses the change of the candidate VM rental cost to assign tasks to achieve the goal of global cost optimization. In general, RMWS achieves a better cost, resource utilization, and success rate and has other performance advantages compared to ROSA and NOSF under different conditions, so it is much more likely to generate superior workflow scheduling scheme under the defined deadline constraints.

Furthermore, as shown in Fig. 8, ROSA has the highest scheduling time among the algorithms, which reaches 3.31, and NOSF shows the best execution performance, with an average time of 0.39, which is lower than that of RMWS, 0.5. This is mainly because ROSA uses more new VMs to execute tasks. New VMs require a boot time that will result in an increase in the scheduling time. RMWS is more complex than NOSF in calculating the subdeadline of tasks, and it also takes more time to calculate the increased rental cost of VMs when assigning tasks. Therefore, the scheduling time of RMWS is slightly worse than that of NOSF, which will be optimized in the future.

## 6 Conclusions and Future Work

To reduce the impact of VM performance fluctuations and heterogeneity in workflow scheduling in the cloud, this paper proposes a real-time multiple-workflow scheduling algorithm named RMWS to minimize the total rental cost under different deadline constraints. RMWS divides the scheduling process into three stages: workflow initialization, task assignment and feedback processing. The algorithm adjusts the parameters of relevant tasks in real time after the current task is completed, which reduces the delay fluctuations caused by variations in the VM performance and network environments. Compared with two state-of-the-art algorithms, RMWS achieves better performance in terms of total rental cost, resource utilization, success rate and deadline deviation under various conditions.

In the future, we will further optimize the performance of the algorithm and consider the scheduling strategy when task execution fails due to hardware faults. In addition, the load balancing of tasks on different VMs is also an important factor to be considered.

## Abbreviations

RMWS: Real-time Multiple-Workflow Scheduling; VM: Virtual Machine; DAG: Directed Acyclic Graph; QoS: Quality of Service; HEFT: Heterogeneous Earliest-Finish-Time; MDP: Markov Decision Process; GA: Genetic Algorithm; IC-PCP: IaaS Cloud Partial Critical Paths; EIPR: Enhanced IC-PCP with Replication; SA: Simulated Annealing; ACO: Ant Colony Optimization; PSO: Particle Swarm Optimization; TTSA: Temporal Task Scheduling Algorithm; DPDS: Dynamic Provisioning Dynamic Scheduling; MOWS: Mixed-parallel Online Workflow Scheduling; ROSA: Uncertainty-aware Online Scheduling Algorithm; NOSF: Online Multiple-workflow Scheduling Framework; DAX: DAG in the XML.

## Availability of Data and Materials

The datasets of all our measurements analysed in this study are available in the following repositories:
1.https://confluence.pegasus.isi.edu/display/pegasus/WorkflowHub
2.https://aws.amazon.com/cn/ec2/pricing/on-demand/

## Competing Interests

The authors declare that they have no competing interests.

## Funding

Not applicable.

## Authors' Contributions

Xiaojin Ma came up with the initial concept of RMWS, the system design, and implemented the prototype. He wrote the majority of this paper. Honghao Gao participated in the design process, provided feedback, and helped to improve the writing. Huahu Xu and Minjie Bian both supported the case study with their deep knowledge about workflow scheduling in cloud computing. All authors reviewed and edited the manuscript. All authors read and approved the final manuscript.

## Acknowledgements

We want to thank the authors of the literature cited in this paper for contributing useful ideas to this study.

## References

[1] P. Mell, and T. Grance, "The NIST Definition of Cloud Computing," *Communications of the Acm*, vol. 53, pp. 50, Jun.2010.

[2] I. Sadooghi, J. H. Martin, T. Li, K. Brandstatter, K. Maheshwari, T. P. P. de Lacerda Ruivo, G. Garzoglio, S. Timm, Y. Zhao, and I. Raicu, "Understanding the Performance and Potential of Cloud Computing for Scientific Applications," *IEEE Transactions on Cloud Computing*, vol. 5, no. 2, pp. 358-371, 2017.

[3] S. Aizad, A. Anjum, and R. Sakellariou, "Representing Variant Calling Format as Directed Acyclic Graphs to Enable the Use of Cloud Computing for Efficient and Cost Effective Genome Analysis," *in 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pp. 784-787, 2017.

[4] J. D. Ullman, "NP-complete scheduling problems," JOURNAL OF COMPUTER AND SYSTEM SCIENCE, vol. 10, no. 3, pp. 384-393, 1975.

[5] A. O. Francis, B. Emmanuel, D. Zhang, W. Zheng, Y. Qin, and D. Zhang, "Exploration of Secured Workflow Scheduling Models in Cloud Environment: A Survey," *in 2018 Sixth International Conference on Advanced Cloud and Big Data (CBD)*, pp. 71-76, 2018.

[6] N. AlMansour, and N. M. Allah, "A Survey of Scheduling Algorithms in Cloud," *in 2019 International Conference on Computer and Information Sciences*, Sakaka, Saudi Arabia, pp. 1-6, 2019.

[7] J. o. Schad, J. Dittrich, and J.-A. Quian´e-Ruiz, "Runtime measurements in the cloud observing, analyzing, and reducing variance," *Proceedings of the VLDB Endowment*, vol. 3, pp. 460-471, Sept.2010.

[8] D. Bruneo, "A Stochastic Model to Investigate Data Center Performance and QoS in IaaS Cloud Computing Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 3, pp. 560-569, 2014.

[9] D. L. Eager, E. D. Lazowska, and J. Zahorjan, "Adaptive load sharing in homogeneous distributed systems," *IEEE Transactions On Software Engineering*, vol. 12, no. 5, pp. 662–675, 1986.

[10] C. Hawblitzel, J. Howell, M. Kapritsos, J. R. Lorch, B. Parno, M. L. Roberts, S. Setty and Brian Zill, "IronFleet: proving safety and liveness of practical distributed systems, " *Communications of the ACM*, vol. 60, no. 7, pp. 83–92, 2017.

[11] Y.-K. Kwok, and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Comput. Surv.*, vol. 31, no. 4, pp. 406-471, 1999.

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

[13] J. J. Dongarra, E. Jeannot, E. Saule, and Z. Shi, "Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems," *in Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures*, San Diego, California, USA, pp. 280-288, 2007.

[14] W. Zheng, and R. Sakellariou, "Budget-Deadline Constrained Workflow Planning for Admission Control," *Journal of Grid Computing*, vol. 11, no. 4, pp. 633-651, 2013.

[15] C. G. Chaves, D. Macedo Batista and N. L. Saldanha Fonseca, "Scheduling Grid Applications with Software Requirements, " *IEEE Latin America Transactions*, vol. 9, no. 4, pp. 578-585, 2011.

[16] J. Yu, R. Buyya, and C. K. Tham, "Cost-based scheduling of scientific workflow applications on utility grids," *in First International Conference on e-Science and Grid Computing (e-Science'05)*, Melbourne, Vic., Australia, pp. 140-147, 2005.

[17] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning, " *in proceedings of the twenty-first international conference on Machine learning*, Banff, Alberta, Canada, 2004.

[18] J. Yu, and R. Buyya, "Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms," *Sci. Program.*, vol. 14, no. 3,4, pp. 217-230, 2006.

[19] D. Whitley, "A genetic algorithm tutorial," *Statistics and Computing*, vol. 4, no. 2, pp. 65-85, Jun.1994.

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

[21] 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.

[22] P. J. van Laarhoven and E. H. Aarts, "Simulated annealing: theory and applications, " *Kluwer Academic Publishers*, 1987.

[23] M. Dorigo, S. Member, and L. M. Gambardella, "Ant Colony System A Cooperative Learning Approach To The Traveling Salesman Problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53-66, Apr.1997.

[24] J. Kennedy, "Particle Swarm Optimization," *Encyclopedia of Machine Learning, C. Sammut and G. I. Webb, eds.*, Springer US, pp. 760-766, 2010.

[25] Z.-H. Zhan, X.-F. Liu, Y.-J. Gong, J. Zhang, H. S.-H. Chung, and Y. Li, "Cloud Computing Resource Scheduling and a Survey of Its Evolutionary Approaches," *ACM Computing Surveys*, vol. 47, no. 4, pp. 1-33, 2015.

[26] M. A. Rodriguez, 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.

[27] H. Yuan, J. Bi, W. Tan, M. Zhou, B. H. Li, and J. Li, "TTSA: An Effective Scheduling Approach for Delay Bounded Tasks in Hybrid Clouds," *IEEE Trans Cybern*, vol. 47, no. 11, pp. 3658-3668, Nov. 2017.

[28] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Algorithms for cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds," *Future Generation Computer Systems*, vol. 48, pp. 1-18, 2015.

[29] Y.-R. Wang, K.-C. Huang, and F.-J. Wang, "Scheduling online mixed-parallel workflows of rigid tasks in heterogeneous multi-cluster environments," *Future Generation Computer Systems*, vol. 60, pp. 35-47, 2016.

[30] A. C. Zhou, B. He, and C. Liu, "Monetary Cost Optimizations for Hosting Workflow-as-a-Service in IaaS Clouds," *IEEE Transactions on Cloud Computing*, vol. 4, no. 1, pp. 34-48, 2016.

[31] H. Chen, X. Zhu, G. Liu, and W. Pedrycz, "Uncertainty-Aware Online Scheduling for Real-Time Workflows in Cloud Service Environment," *IEEE Transactions on Services Computing*, vol. 11, preprint, 2018.

[32] J. Liu, J. Ren, W. Dai, D. Zhang, P. Zhou, Y. Zhang, G. Min, and N. Najjari, "Online Multi-Workflow Scheduling under Uncertain Task Execution Time in IaaS Clouds," *IEEE Transactions on Cloud Computing*, preprint, 2019.

[33] E.-K. Byun, Y.-S. Kee, J.-S. Kim, and S. Maeng, "Cost optimized

provisioning of elastic resources for application workflows," *Future Generation Computer Systems*, vol. 27, no. 8, pp. 1011-1026, 2011.

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

[35] J. Sahni, and P. Vidyarthi, "A Cost-Effective Deadline-Constrained Dynamic Scheduling Algorithm for Scientific Workflows in a Cloud Environment," *IEEE Transactions on Cloud Computing*, vol. 6, no. 1, pp. 2-18, 2018.

[36] Y. H. Zhu, W. L. Zhang, Y. H. Chen and H. H. Gao, "A novel approach to workload prediction using attention-based LSTM encoder-decoder network in cloud environment, " *EURASIP Journal on Wireless Communications and Networking*, vol.247, pp.1-18, 2019.

[37] Z. G. Chen, Z. H. Zhan, Y. Lin, Y. J. Gong, T. L. Gu, F. Zhao, H. Q. Yuan, X. Chen, Q. Li, and J. Zhang, "Multiobjective Cloud Workflow Scheduling: A Multiple Populations Ant Colony System Approach," *IEEE Trans Cybern*, vol. 49, no. 8, pp. 2912-2926, Aug.2019.

[38] X. X. Yang, S. J. Zhou and M. Cao, "An Approach to Alleviate the Sparsity Problem of Hybrid Collaborative Filtering Based Recommendations: The Product-Attribute Perspective from User Reviews," Mobile Networks & Applications, vol. 25, no. 2, pp. 376-390, 2020.

[39] X. J. Ma, H. H. Gao, H. H. Xu and M. J. Bian, "An IoT-based task scheduling optimization scheme considering the deadline and cost-aware scientific workflow for cloud computing," *EURASIP Journal on Wireless Communications and Networking*, (249)2019.

[40] H. H. Gao, C. Liu, Y. H. Z. Li and X. X. Yang. "V2VR: Reliable Hybrid-Network-Oriented V2V Data Transmission and Routing Considering RSUs and Connectivity Probability," *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[41] H. H. Gao, W. Q. Huang and Y. C. Duan, "The Cloud-edge-based Dynamic Reconfiguration to Service Workflow for Mobile Ecommerce Environments: A QoS Prediction Perspective," *ACM Transactions on Internet Technology*, vol. 21, no. 1, pp. 1-23, 2021.

[42] H. H. Gao, L. Kuang, Y. Y. Yin, B. Guo and K. Dou, "Mining Consuming Behaviors with Temporal Evolution for Personalized Recommendation in Mobile Marketing Apps," *ACM/Springer Mobile Networks and Applications*, vol. 25, no. 4, pp. 1233–1248,.

[43] L.-C. Canon, and E. Jeannot, "Correlation-Aware Heuristics for Evaluating the Distribution of the Longest Path Length of a DAG with Random Weights," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 11, pp. 3158 - 3171, 2016.

[44] W. Zheng, M. Zhou, L. Wu, Y. Xia, X. Luo, S. Pang, Q. Zhu, and Y. Wu, "Percentile Performance Estimation of Unreliable IaaS Clouds and Their Cost-Optimal Capacity Decision," *IEEE Access*, vol. 5, pp. 2808-2818, 2017.

[45] Q. Wu, F. Ishikawa, Q. Zhu, Y. Xia, and J. Wen, "Deadline-Constrained Cost Optimization Approaches for Workflow Scheduling in Clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 12, pp. 3401-3412, 2017.

[46] C. Lunardi, H. Quinn, L. Monroe, D. Oliveira, P. Navaux, and P. Rech, "Experimental and Analytical Analysis of Sorting Algorithms Error Criticality for HPC and Large Servers Applications," *IEEE Transactions on Nuclear Science*, vol. 64, no. 8, pp. 2169-2178, 2017.

[47] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, "Characterizing and Profiling Scientific Workflows," *Future Generation Computer Systems*, vol. 29, no. 3, pp. 682-692, 2013.

## AUTHOR INFORMATION

**Xiaojin Ma** received the BS, MS in Computer Science and Management Science & Engineering from Henan University of Science and Technology in 2003, 2013, respectively. He is working toward the Ph.D. degree in Shanghai University, China.

**Huahu XU** received the Ph.D. degree in Computer Application from Shanghai University.He is currently a Doctoral Supervisor and a Professor with the School of Computer Engineering and Science, Shanghai University, where he is also the Director of the Information Office of Shanghai University. He is the Chairman of the Shanghai Security and Technology Association.

**Honghao Gao** received the Ph.D. degree in Computer Science and started his academic career at Shanghai University in 2012. Prof. Gao is currently with the School of Computer Engineering and Science, Shanghai University, China. He is also a Professor at Gachon University, South Korea. Prior to that, he was a Research Fellow with the Software Engineering Information Technology Institute of Central Michigan University (CMU), USA, and was also an Adjunct Professor at Hangzhou Dianzi University, China. His research interests include Software Formal Verification, Industrial IoT/Wireless Networks, Service Collaborative Computing, and Intelligent Medical Image Processing. He has publications in IEEE TII, IEEE T-ITS, IEEE IoT-J, IEEE TNSE, IEEE TCCN, IEEE/ACM TCBB, ACM TOIT, ACM TOMM, IEEE TCSS, IEEE TETCI, IEEE Network, and IEEE JBHI.
Prof. Gao is a Fellow of IET, BCS, and EAI, and a Senior Member of IEEE, CCF, and CAAI. He is the Editor-in-Chief for International Journal of Intelligent Internet of Things Computing(IJIITC), Editor for Wireless Network(WINE) and IET Wireless Sensor Systems(IET WSS), and Associate Editor for IET Software, International Journal of Communication Systems(IJCS), Journal of Internet Technology(JIT), and Journal of Medical Imaging and Health Informatics(JMIHI). Moreover, he has broad working experiences in industry-university-research cooperation. He is a European Union Institutions appoint external expert for reviewing and monitoring EU Project, is a member of the EPSRC Peer Review Associate College for UK Research and Innovation in the UK, and is also a founding member of IEEE Computer Society Smart Manufacturing Standards Committee.

**Minjie Bian** received the Ph.D. degree in Computer Application Technology from Shanghai University in 2016. His research interests include cloud computing, Web3D and parallel computing.

# Figures

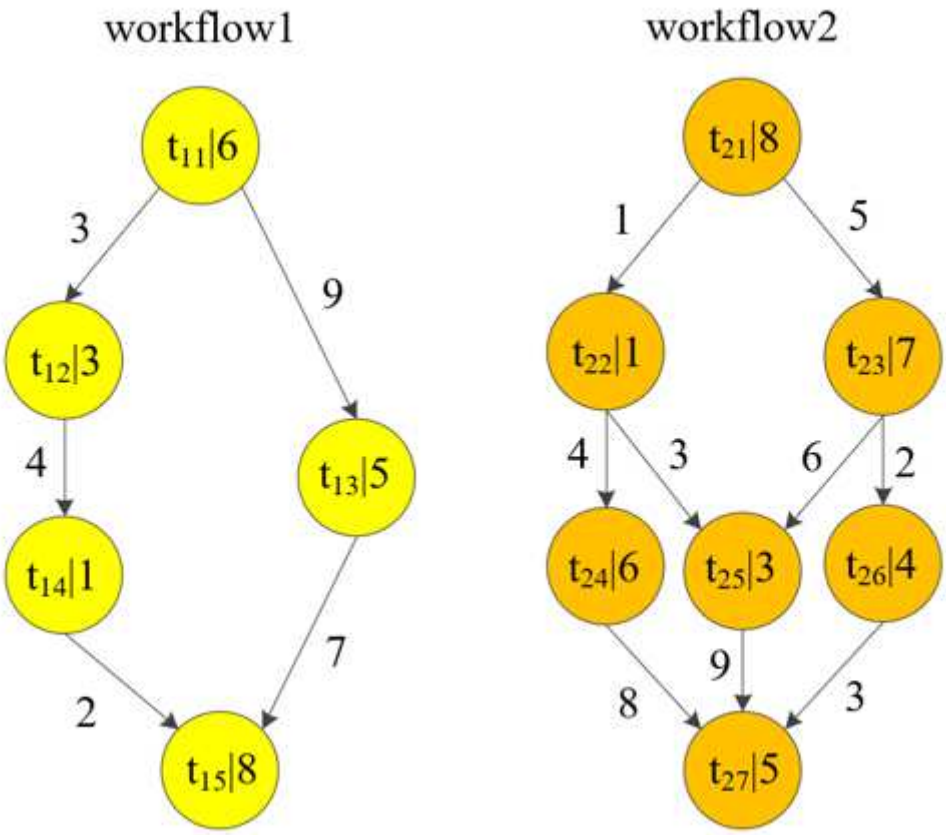



**Figure 1**

Two example workflows.

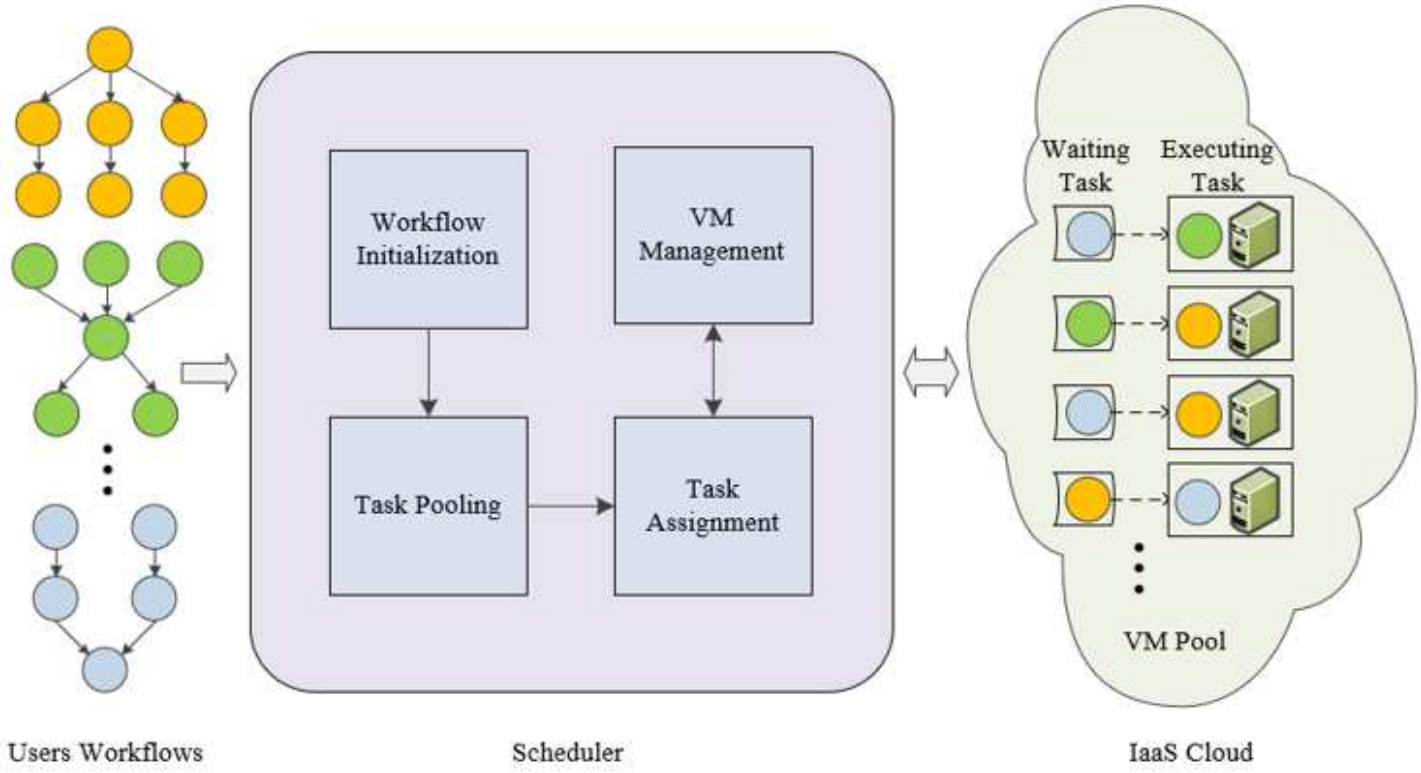



**Figure 2**

The scheduling architecture in the cloud.

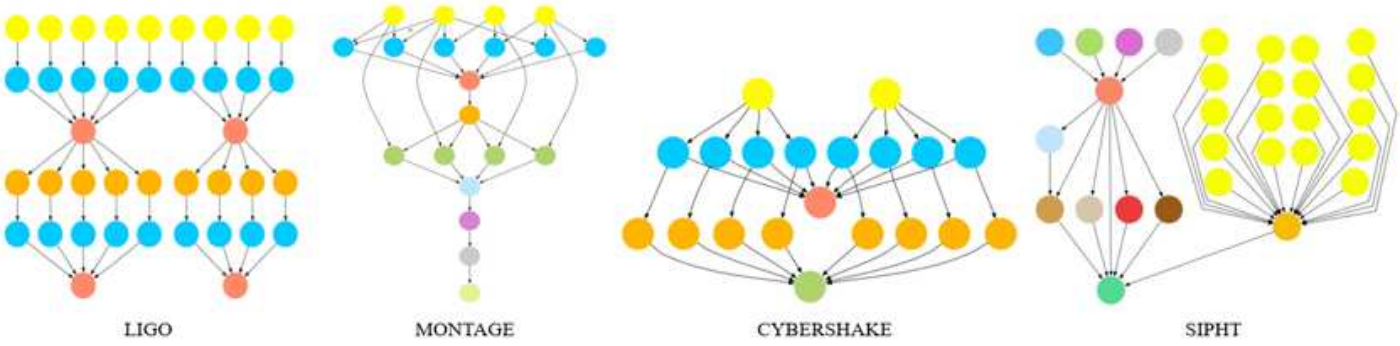



**Figure 3**

The structures of the workflows.

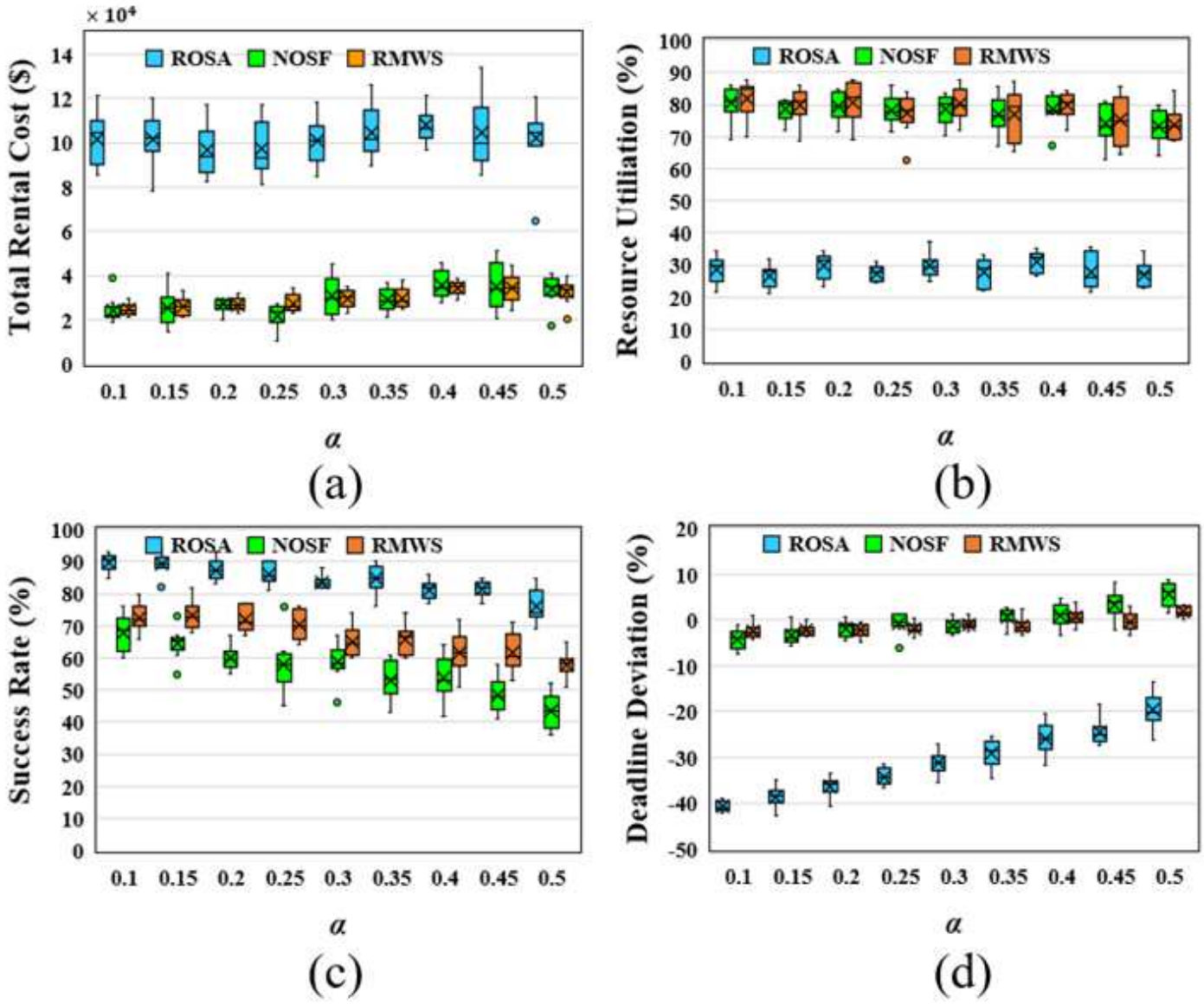



**Figure 4**

The impact of variance in task execution time.

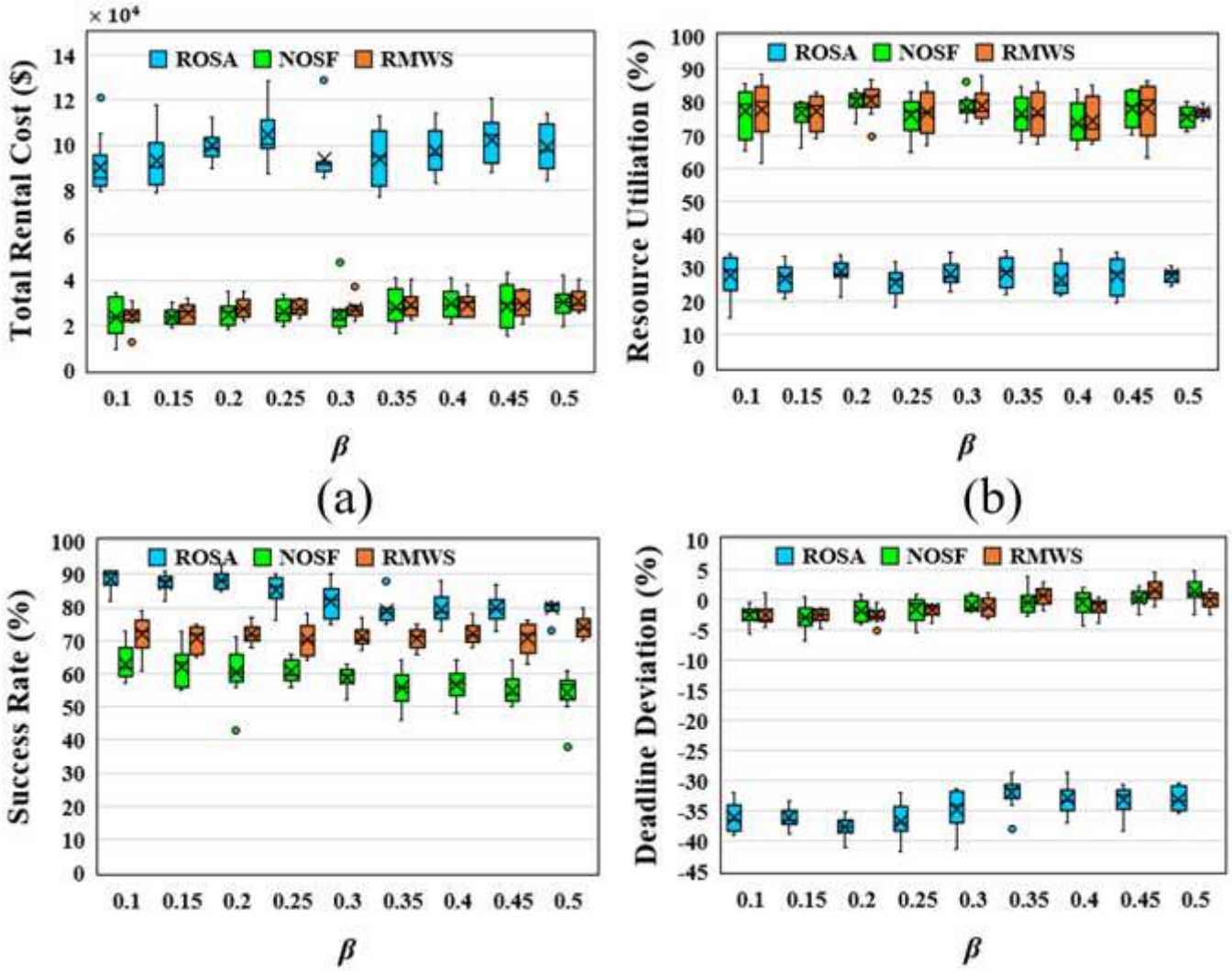



**Figure 5**

The impact of variance in data transfer time.

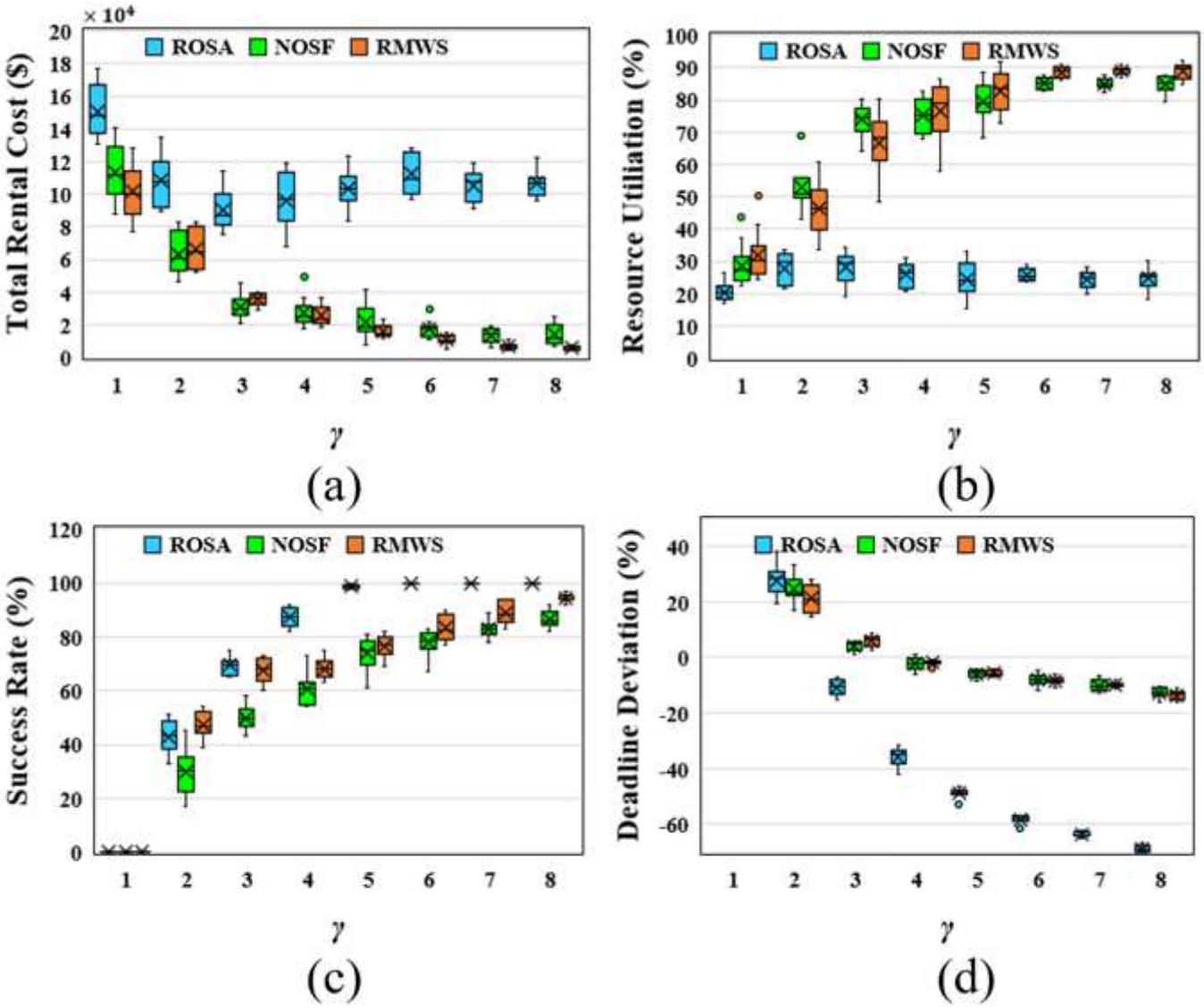



**Figure 6**

The impact of variance in the deadline factor.

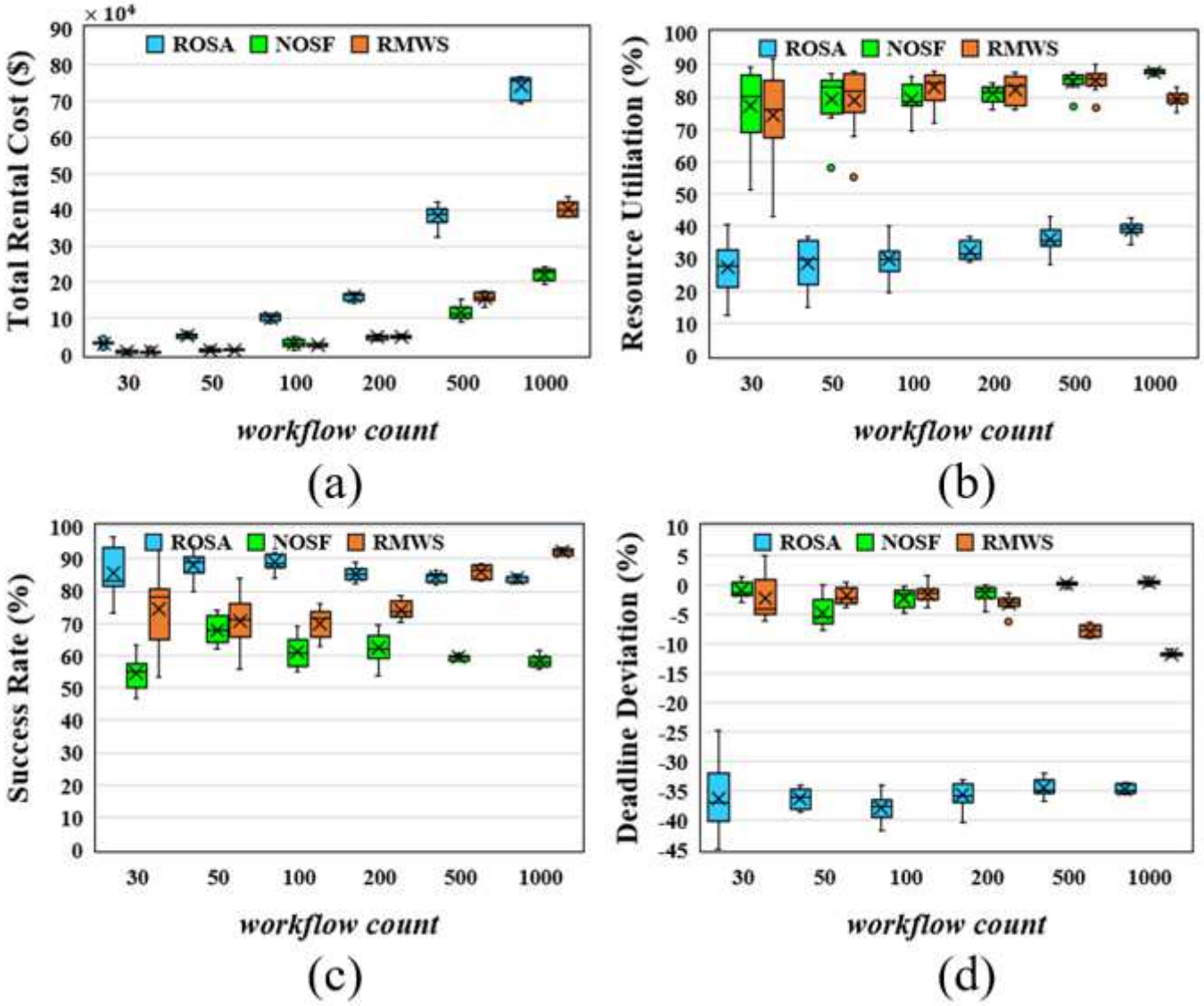
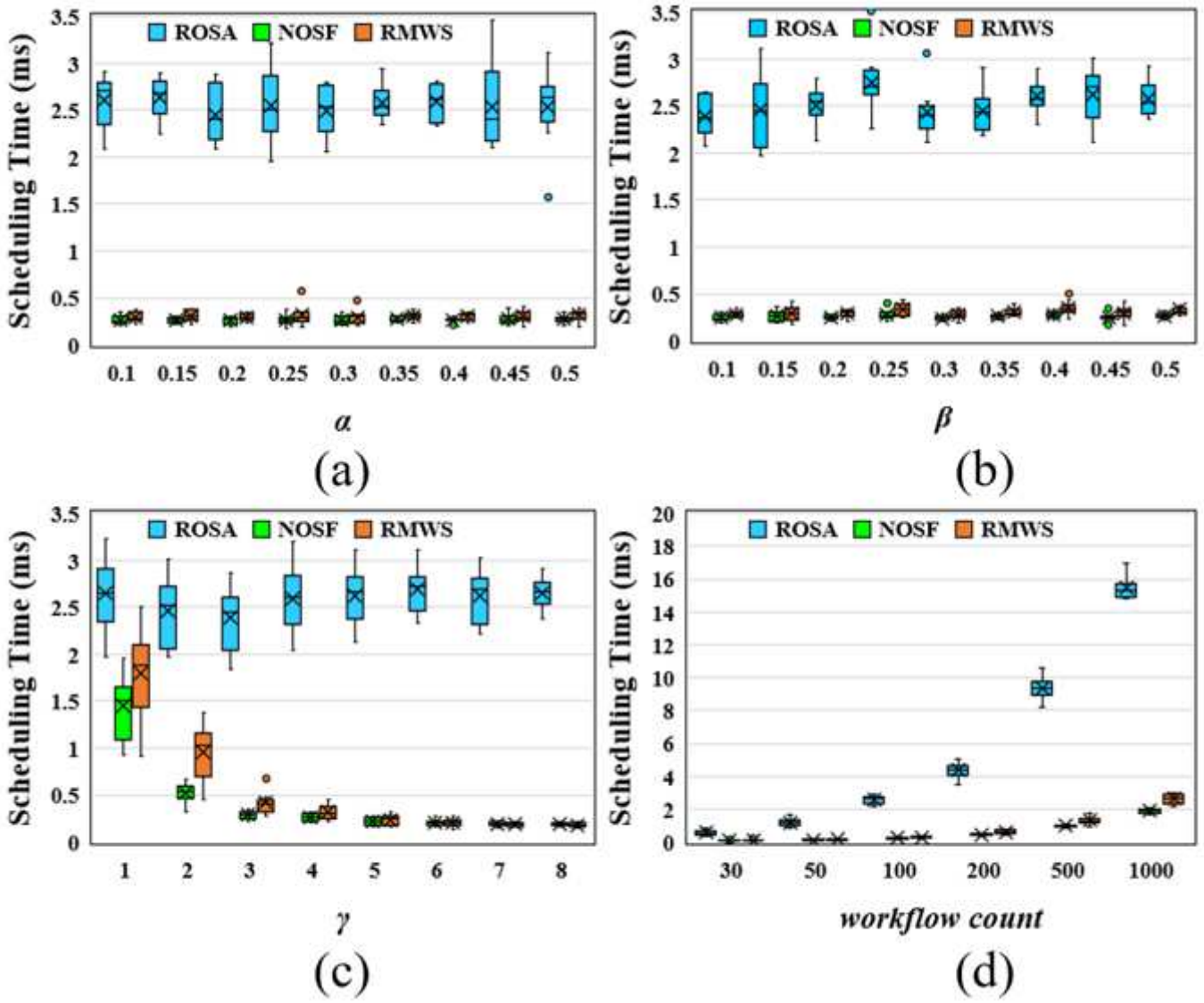



**Figure 7**

The impact of variance in the workflow count.




## Figure 8

The average scheduling time of each task.