SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

September 2022, Vol. 65 192106:1–192106:18
https://doi.org/10.1007/s11432-020-3295-2

Scheduling multi-tenant cloud workflow tasks with resource reliability

Xiaoping LI[1*], Dongyuan PAN[1], Yadi WANG[2] & Rubén RUIZ[3]

[1]*School of Computer Science and Engineering, Southeast University, Nanjing* 211189*, China;*
[2]*Institute of Data and Knowledge Engineering, School of Computer and Information Engineering, Henan University, Kaifeng* 475004*, China;*
[3]*Grupo de Sistemas de Optimización Aplicada, Universitat Politècnica de València, Camino de Vera s/n, València* 46022*, Spain*

Received 16 December 2020/Revised 30 April 2021/Accepted 18 June 2021/Published online 29 August 2022

Abstract Resource reliability is crucial in scheduling workflow instances for different tenants. Both cloud resource reliability and precedence constraints in workflows bring about great challenges for these kinds of scheduling problems. In this paper, we construct a hybrid resource reliability model which is adaptively evaluated in every time window. The objective is to optimize the QoS (quality of service) for tenants which is measured by the introduced AISE (all instance success entropy) index. A scheduling algorithmic framework is proposed for the studied workflows which consider cloud resource reliability. Deadline and budget division (BD) methods are presented to divide deadlines and budgets of instances into those of tasks. A tenant sequence method is developed to determine the order of tenants. A task allocation strategy is investigated to schedule tasks that are ready to appropriate available resources. Parameters and algorithm component candidates are statistically calibrated over a comprehensive set of random instances using the analysis of variance technique. The performance of the proposed algorithm is also evaluated in practical instances.

Keywords multi-tenancy, resource reliability, cloud computing, scheduling

Citation Li X P, Pan D Y, Wang Y D, et al. Scheduling multi-tenant cloud workflow tasks with resource reliability. Sci China Inf Sci, 2022, 65(9): 192106, https://doi.org/10.1007/s11432-020-3295-2

1 Introduction

Multi-tenant instance-intensive workflows are common in cloud computing settings, e.g., stock trading in securities markets, online shopping, and e-commerce (for example the Double 11 Shopping Carnival in China and Black Friday in the United States) and bank transfer businesses. Generally, a tenant is a group of users who share common access to some resources (software, system, data, etc.). However, vendors provide dedicated cloud services to each tenant and the resources used by different tenants are independent of each other in single-tenant clouds, i.e., applications and data between different customers are generally isolated by hardware. The single-tenant mode is widely used in customization scenarios. From the perspective of cloud providers, a single-tenant cannot provide flexible and scalable cloud services. Resource sharing by all tenants in a multi-tenant cloud computing environment results in better resource utilization, more flexible services, and less rental costs while incurring more complicated management. In a multi-tenant cloud system, each tenant submits one or more workflow instances constrained by the same deadline and each task in these instances is usually simple. A large number of tenants implies a very large number of concurrent workflow instances. The fact that resources are limited in practical cloud computing environments means that some workflow instances might not finish before their deadlines, i.e., some tenants cannot be satisfied.

From the service provider's perspective, maximizing the execution success rate of tenants and minimizing the total execution costs are of great importance, i.e., both scheduling tasks to appropriate resources

* Corresponding author (email: xpli@seu.edu.cn)

and reducing the execution failures caused by unreliable resources are desirable for tenants. The execution success rate of tenants can be maximized by scheduling instances from the same tenant to reliable resources before their deadlines. Higher costs and additional waiting times might arise if rescheduling instances are allocated to unreliable resources. To minimize the total execution cost it is necessary to select reliable and cheap resources for workflow instances. In other words, resource reliability is crucial for service vendors' decisions.

Guaranteeing the quality of service (QoS) by service providers can be regarded as one of the main concerns for companies tending to use it [1]. In this paper, we consider the instance-intensive workflow scheduling problem with resource reliability in a multi-tenant cloud to maximize the QoS of tenants. AISE (all instance success entropy) is introduced to measure the QoS of tenants which is closely related to the finish time and the cost of each instance of a tenant. Workflow instances for the same tenant have the same deadline. Every instance contains a small number of tasks. Servers are heterogeneous in a single datacenter, i.e., they have different processing speeds. Each task is conducted by only one VM (virtual machine) on a server. Server breakdowns or removals lead to unreliable cloud resources.

For the problem under study, there are several challenges: (i) Execution success rates for tenants are closely related to the workflow scheduling strategy and resource reliability. Generally, higher reliability implies higher execution success rates for tenants with a higher cost for service providers. However, lower costs cannot guarantee reliability and possibly result in lower execution success rates for tenants. Furthermore, lower reliability might imply a higher cost rather than a lower one. Rescheduling is necessary if tasks on unreliable VMs cannot finish on time which might lead to a higher cost. (ii) To minimize the AISE index, the number of tenants with all their instances completed before the deadline is maximized. It is crucial to sequence all tenants. In addition, the workflow deadline is transferred to soft sub-deadlines of the tasks. Since the sub-deadlines might be violated when instances for different tenants, or even instances for the same tenant, compete for the same VM, it is desirable to schedule tasks to appropriate available VMs. (iii) Though cloud resource reliability can be estimated in advance, breakdowns happen stochastically. Scheduling a large number of instances to resources with different reliability is challenging for the considered objective.

Workflow tasks for tenants are scheduled time window by time window. Resource reliability is updated according to past time windows. Tasks scheduled in the previous time window are executed and those tasks to be executed in the next time window are scheduled. The main contributions of this paper are:

(i) By taking advantage of both the Bayesian method and the reliability-driven (RD) reputation method, a hybrid resource reliability evaluation method is developed.

(ii) The properties of the AISE index to measure QoS for tenants are analyzed.

(iii) A workflow scheduling framework with resource reliability along with its algorithmic components is proposed.

The rest of the paper is organized as follows. Related studies are reviewed in Section 2. Section 3 describes and formalizes the problem under study. A heuristic framework is proposed for the considered problem in Section 4. Section 5 statistically calibrates the parameters and component candidates and evaluates the performance of the proposed algorithm followed by the conclusion and future work in Section 6.

2 Related work

The problem of scheduling workflow tasks to cloud resources is NP-hard in distributed systems [2–7]. Commonly studied workflow scheduling problems are computing-intensive and data-intensive. Only a few instance-intensive cases have been considered. Most of them focused on general QoS aspects such as time and/or cost, without considering the reliability of cloud resources.

Resource reliability is usually measured by reputation or trust. Based on the Markov theory, queuing theory, and Bayesian theory, resource reliability models have been mathematically established [8,9]. Roy et al. [10] proposed a software reliability training model based on an artificial neural network (ANN), which is used to enhance software reliability prediction and train the proposed neural network effectively with software fault data. Using the non-sequential Monte Carlo simulation, Snyder et al. [11] presented a scalable algorithm to evaluate the reliability of large scale cloud computing systems. Zhang et al. [12] proposed a direct trust evaluation model based on transaction interaction, and divided the trust into five levels. In order to evaluate the recent trust, the author proposes a method of storing and updating

trust values based on a sliding window. Qiu et al. [13] developed a correlated modeling approach to analyze reliability-performance and reliability-energy correlations for cloud services. Wang [14] evaluated the resource reliability using a performance-aware probability of failure on Demand which is related to the number of successful responses satisfying the time constraint. Jøsang et al. [15] evaluated resource reliability using its reputation which was defined as the probability of the resource providing the expected service. A trust value is generally obtained by weighted direct trusts and recommended trusts [16]. A direct resource trust is the rate of successfully completed tasks. In [17], resource reliability was only defined by direct trusts. Kavitha et al. [18] evaluated resource reliability using subjective and objective trusts. A subjective trust further includes a direct and a recommendation trust. An objective trust is closely related to the attributes of a resource, e.g., network bandwidth, load, and availability. However, it is unfair to evaluate a trust only by the successful execution rate. For example, the processing times of some tasks could be very long and the number of successfully completed tasks could be very small and a low trust might result. Wang et al. [19] presented an RD reputation calculation model considering task execution times using time windows. However, no historical information was considered.

Reliability is very important in workflow scheduling. Wen et al. [20] introduced an entropy-based model to quantify the most reliable workflow deployments. An algorithm was proposed to optimize entropy and cost. Energy consumption and system reliability are two conflicting objectives. Zhang et al. [21] presented a bi-objective genetic algorithm for workflow scheduling to pursue low energy consumption and high system reliability. Based on RD reputations, Wang et al. [19] proposed a look-ahead genetic algorithm to optimize both makespan and reliability for computing-intensive workflow applications. Wang et al. [22] proposed a trust dynamic level scheduling algorithm for data-intensive workflows where the failure rate of task execution was reduced with additional computation time. Based on resource availability, Tao et al. [23] developed a workflow scheduling strategy for dependable grids. Little attention has been paid to instance-intensive workflow scheduling problems with resource reliability.

For instance-intensive workflow scheduling problems, maximizing the system throughput and minimizing the total execution cost are two common objectives. Wang et al. [24] investigated an energy-efficient instance-intensive cloud workflow scheduling method with a batch processing procedure. Taking into account several activity instances and balanced utilization of resources of physical machines, energy efficiency was improved for instance-intensive cloud workflows. Wen et al. [25] presented an energy and cost aware algorithm for instance-intensive IoT workflow scheduling, with batch processing in clouds, to improve energy efficiency and to reduce the execution cost while meeting deadline requirements. Liu et al. [26] proposed a throughout maximization strategy in a peer-to-peer based SwinDeW-G. Two algorithms were proposed to schedule instance-intensive workflows from the instance and task levels respectively. Liu et al. [27] proposed a compromised time-cost scheduling algorithm to balance time and costs in cloud environments that perform well on the SwinDeW-C platform. Yang et al. [28] proposed a cost-constrained algorithm for a large number of instances in instance-intensive cloud workflows in order to optimize the total execution cost while satisfying the deadline constraint. Li et al. [29] classified workflow tasks based on users' QoS preferences. A QoS-based deadline allocation scheduling algorithm was proposed for instance-intensive workflows in cloud environments to better meet user requirements and obtain a better load balance. However, the above work did not consider multi-tenancy in cloud computing environments.

More and more attention has been paid to multi-tenant workflow scheduling in cloud computing. Rodriguez and Buyya [30] leveraged containers to address resource utilization inefficiency for multi-tenant workflow scheduling. The objective was to minimize the overall cost of leasing infrastructure resources while meeting the deadline constraint of each workflow. Cui et al. [31] proposed a time-saving-degree and cost optimization scheduling algorithm for multi-tenant instance-intensive workflows to improve the QoS for tenants and to reduce the execution cost of workflow instances. Rimal and Maier [32] proposed a cloud-based workflow scheduling policy for computing-intensive workflow applications in multi-tenant cloud computing environments. The policy was compared to existing studies on several metrics, such as the overall workflow completion time, tardiness, the cost of workflows, and idle resource utilization in clouds. Resource reliability was not considered in these studies either.

The resource reliability of a cloud resource has a great influence on task execution because higher costs and longer execution times might result. To the best of our knowledge, the considered multi-tenant instance-intensive workflow scheduling problem with resource reliability is common in practice while it has not been studied yet in the literature.

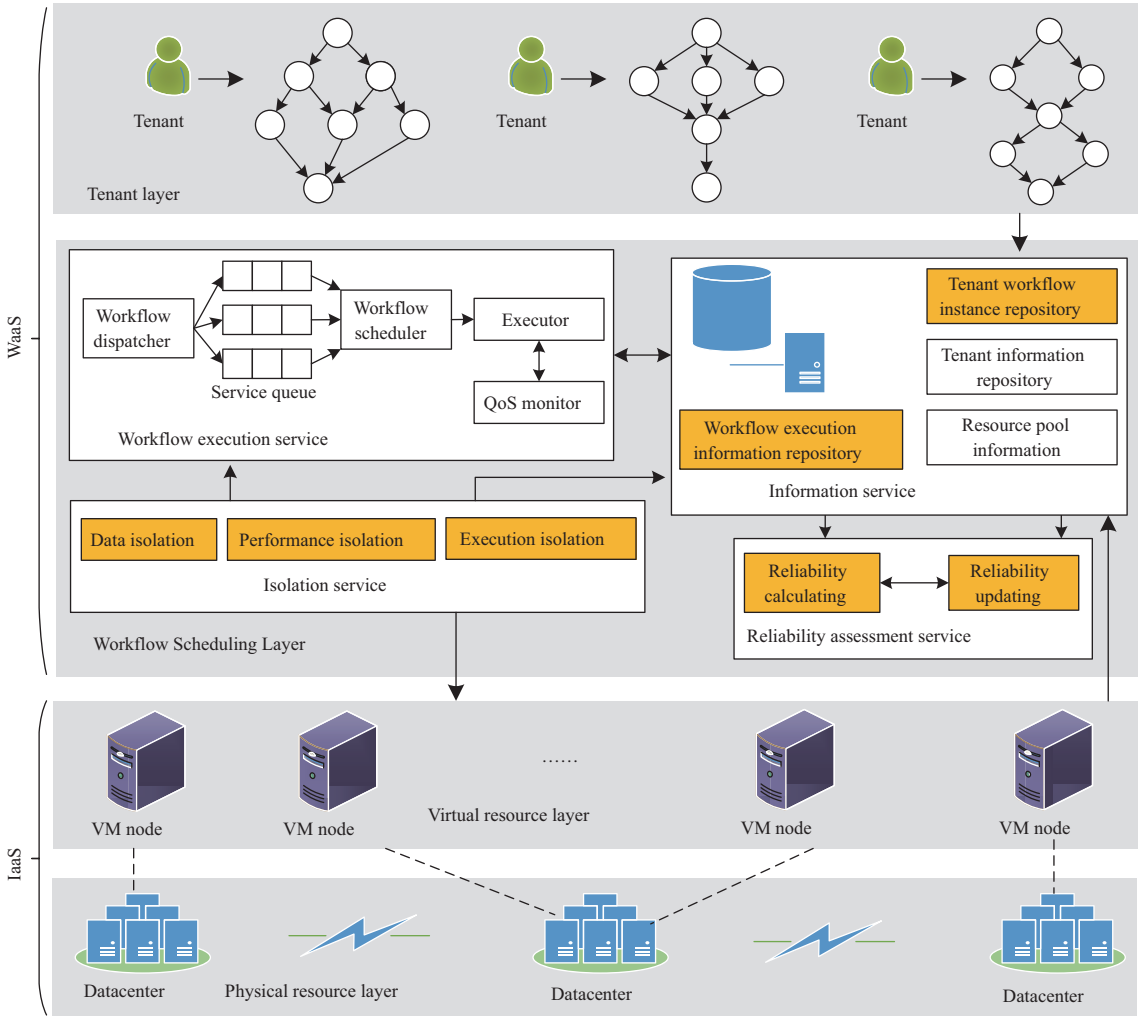

Figure 1 (Color online) The proposed system architecture for multi-tenant clouds with resource reliability.

3 Problem description

Since resource reliability is considered in the problem under study, the reference architecture proposed by Rimal and Maier [32] for workflows in a multi-tenant cloud environment is no longer valid. A modified architecture is developed as shown in Figure 1.

3.1 System architecture

There are four layers in the system architecture: the tenant layer, the workflow scheduling layer, the virtual resource layer, and the physical resource layer. The first two layers are composed of WaaS (workflow as a service) which mainly provide services to tenants. The latter two layers belong to the IaaS (infrastructure as a service) of the cloud computing architecture which dynamically provides on-demand virtual resource services to the WaaS layer. The tenant layer creates workflow instances for tenants. The workflow scheduling layer receives the workflow instances submitted by tenants, collects resource information, evaluates the reliability of resources, and provides tenants with reliable services. The virtual resource layer turns the underlying physical resources into virtual resources. The physical resource layer consists of infrastructures such as servers, storage, and routers. With the exception of the workflow scheduling layer, the other three layers are identical to those in [32]. Details of the main components of the workflow scheduling layer are explained below.

The workflow scheduling layer processes workflow instances submitted by tenants and generates workflow scheduling policies according to the tenant information, the workflow instance information submitted by tenants and the currently available resource information. This layer guarantees that isolation among

different tenants and the scheduling performance of workflow instances meet the service level agreement (SLA) requirements of the tenants. (i) In the workflow execution service module, the workflow dispatcher extracts instances from the tenant workflow instance repository and distributes them to the workflow scheduler. In terms of resource pool availability, the executor schedules instances. The QoS monitor monitors the execution progress and records the progress to the workflow execution information repository. (ii) The information service module includes all the information needed by the workflow instance execution, e.g., tenants, workflow instances, hardware and software resources, and the workflow execution progress. (iii) The isolation service module ensures no mutual interference. Data isolation ensures that each tenant cannot obtain data from the other tenants. Performance isolation prevents the executing instances of one tenant influencing the other tenants. Execution isolation guarantees the executing processes are independent of each other. (iv) The reliability assessment service module assesses resource reliability using historical information from the workflow execution information repository. Resource reliability in the resource pool information is calculated and updated.

3.2 Resource reliability

Resource reliability has a great influence on the performance of scheduling algorithms. Several evaluation methods have been proposed among which the typical ones are the Bayesian [22] and the RD reputation methods [19]. The notations used are given in Table 1.

Suppose there are u successful tasks and v failed ones among the N tasks executed by a resource, i.e., $N = u + v$. The probability of a task being successfully executed in one iteration is represented by θ. The posterior probability of successful task execution on a resource follows a Beta distribution. The successful execution probability among the $N + 1$ executions is $\hat{\theta}_{\text{Bay}} = E(\text{Beta}(\theta \mid u + 1, v + 1)) = \frac{u+1}{N+2}$ which is the direct trust value of the resource.

According to Wang et al. [22], there are two main pitfalls in the Bayesian method: (i) If a resource does not interact with the task, it is impossible to measure. (ii) Trust evaluation cannot be performed if a resource interacts with tasks on only a few occasions. To get a confidence level measure of $\hat{\theta}_{\text{Bay}}$ with sampling error ε and confidence level threshold γ_0, the sample size satisfies $N_0 \geqslant -\frac{1}{2\varepsilon^2}\ln(\frac{1-\gamma_0}{2})$, i.e., a lot of samples are needed for resource reliability evaluation in the Bayesian method. However, this is seldom the case.

Different from the Bayesian method, the RD reputation method measures resource reliability by calculating a real-time reputation. With the assumption that the failure event of a resource follows an exponential distribution, the failure rate of a resource is reciprocal to the mean time to failure. The RD reputation or the real-time failure rate θ_t for a resource is the average number of jobs that the resource does not successfully complete within the t-th time window. In each time window, every resource maintains a reputation statistic vector (T_s, T_f, T_e, c). T_s and T_f are the start and finish times of the time window respectively. T_e is the total CPU time employed by a resource for task execution in the time window and c is the number of tasks that failed to execute during this time window. The failure rate of the resource in $T_f - T_s - T_e$ is assumed to be θ_{t-1}, i.e., the failure rate of the previous time window. Therefore, $\theta_t = \frac{c+\theta_{t-1}(T_f-T_s-T_e)}{T_f-T_s}$. Since the failure event of the resource generally follows an exponential distribution, the failure probability of the resource is $\hat{\theta}_{\text{fail}} = 1 - \mathrm{e}^{-t_s\theta_t}$. t_s represents the execution duration of all tasks assigned to a resource before resource failure occurs. The probability of resource reliability is $\hat{\theta}_R = \mathrm{e}^{-t_s\theta_t}$ which is independent of the number of samples. In this paper, we compute the failure rate of a resource in every time window with length T^w. For the ℓ-th time window, $\theta_\ell = \frac{c+\theta_{\ell-1}(T^w-T_e)}{T^w}$. Let ξ be the number of consecutive time windows with reliable resources (there might be many breakdowns in the ℓ time windows). The reliability probability of the resource is determined by $\hat{\theta}_R(\xi, \ell) = \mathrm{e}^{-\xi T^w\theta_\ell}$. However, the algorithm is easily affected by local conditions in the time window and does not consider the execution of resource history.

A large number of samples are needed by Bayesian methods to precisely evaluate the reliability of resources while the number of samples is not large enough at the beginning of processing tasks by a resource. The RD reputation method measures resource reliability by calculating a real-time reputation whereas the algorithm is easily affected by local conditions in the time window and does not consider the execution of resource history. It is desirable to integrate Bayesian and RD reputation methods. In this paper, a hybrid model is proposed to guarantee the predicting precision of resource reliability as follows:

$$\hat{\theta}(\xi, \ell; N, u) = \left(1 - \mathrm{e}^{-\frac{N}{N_0}}\right)\hat{\theta}_{\text{Bay}} + \mathrm{e}^{-\frac{N}{N_0}}\hat{\theta}_R(\xi, \ell). \tag{1}$$

Table 1 Notations used in the proposed framework

Notation	Description
u	Number of successful task executions of a resource
v	Number of unsuccessful task executions of a resource
N	Number of task executions of a resource, $N = u + v$
N_0	Sample size
ε	Sampling error
γ_0	Confidence level threshold
$\hat{\theta}_{\text{Bay}}$	Resource reliability based on a Bayesian method
$\hat{\theta}_R(\xi, \ell)$	Resource reliability based on an RD reputation method
$\hat{\theta}(\xi, \ell; N, u)$	Resource reliability based on a hybrid method
θ_ℓ	Failure rate of a resource in the ℓ-th time window
T_s	Start time of a time window
T_f	Finish time of a time window
T_e	Total CPU time for task executions in a time window
c	Number of failed task executions in a time window
t_s	Duration time before resource failure occurs
n	Number of tenants
m	Number of VMs
r_i	Number of instances for the i-th tenant
n_i	Number of tasks in the i-th tenant's instance
M_l	l-th VM
G_i	Workflow model of the i-th tenant
D_i	Deadline of the i-th tenant
$B_{i,j}$	Budget of $I_{i,j}$
$v_{i,j,k}$	k-th task of $I_{i,j}$
$V_{i,j}$	Set of tasks in $I_{i,j}$
$E_{i,j}$	Set of directional edges in $I_{i,j}$
$I_{i,j}$	j-th instance of the i-th tenant, $I_{i,j} = (V_{i,j}, E_{i,j})$
$\boldsymbol{R}^r_{i,j,k}$	Resource requirement vector of $v_{i,j,k}$
$\tau_{i,j,k}$	Computational burden of $v_{i,j,k}$
$\text{PR}_{i,j,k}$	Immediate predecessor set of $v_{i,j,k}$
$\boldsymbol{R}^p_l$	Capability vector of M_l
S_l	Processing speed of M_l
p_l	Unit price of M_l
$T^s_{i,j,k}$	Start time of $v_{i,j,k}$
$T^f_{i,j,k}$	Finish time of $v_{i,j,k}$
$C_{i,j}$	Total execution cost of instance $I_{i,j}$
z_i	Success rate of the i-th tenant

$\hat{\theta}(\xi, \ell; N, u)$ can be calculated adaptively according to N. Initially, resource reliability is mainly dependent on the RD reputation. With an increase in the number of executed tasks N, the Bayesian method gains precision gradually, i.e., the probability of resource reliability is determined by both $\hat{\theta}_R(\xi, \ell)$ and $\hat{\theta}_{\text{Bay}}$, and which model has a greater impact on resource reliability is adaptive by N.

3.3 Problem formulation

Requests of n tenants are submitted to m VMs $\{M_1, \ldots, M_m\}$ in a cloud datacenter. Each tenant submits a batch of requests, each of which contains the same number of precedence constrained tasks. The requests have the same deadline. Therefore, we model the requests as a workflow model with a number of activities (which are tasks in each workflow instance).

• The requests of each tenant are described by a workflow model, i.e., there are n workflow models $\{G_1, \ldots, G_n\}$. r_i instances $\{I_{i,1}, \ldots, I_{i,r_i}\}$ are generated by G_i with the same deadline D_i and the same number of tasks n_i. Instance $I_{i,j}$ is constrained by budget $B_{i,j}$. The j-th instance $I_{i,j}$ of G_i is described by DAG (directed acyclic graph) $I_{i,j} = (V_{i,j}, E_{i,j})$, where $V_{i,j} = \{v_{i,j,0}, v_{i,j,1}, \ldots, v_{i,j,n_i}, v_{i,j,n_i+1}\}$ and $E_{i,j} = \{(v_{i,j,k}, v_{i,j,k'}) | v_{i,j,k}, v_{i,j,k'} \in V_{i,j}\}$. $(v_{i,j,k}, v_{i,j,k'}) \in E_{i,j}$ implies that task $v_{i,j,k}$ cannot start until task $v_{i,j,k'}$ finishes. $v_{i,j,0}$ and v_{i,j,n_i+1} are the source and sink dummy tasks, respectively, both with 0

processing time. Task $v_{i,j,k}$ requires several types of resources, e.g., CPU, memory and I/O. The resource requirements of $v_{i,j,k}$ are denoted by a resource vector $\boldsymbol{R}^r_{i,j,k}$. The computational burden of $v_{i,j,k}$ is a number of instructions which is represented by $\tau_{i,j,k}$. $\mathrm{PR}_{i,j,k}$ is the immediate predecessor task set of $v_{i,j,k}$.

• The processing speed of a VM M_l is S_l. The resource provision capability vector of M_l is $\boldsymbol{R}^p_l$ with a unit price p_l. A decision variable $x_{i,j,k,l} \in \{0,1\}$ is defined which takes 1 if and only if task $v_{i,j,k}$ is allocated to a candidate VM M_l with $\boldsymbol{R}^r_{i,j,k} - \boldsymbol{R}^p_l \geqslant 0$. The processing time of $v_{i,j,k}$ is $\frac{\tau_{i,j,k}}{S_l}$ if it is assigned to M_l with cost $\frac{\tau_{i,j,k} \times p_l}{S_l}$.

• Let $T^s_{i,j,k}$ and $T^f_{i,j,k}$ be the start and finish times of $v_{i,j,k}$ respectively. The completion time of $I_{i,j}$ is T^f_{i,j,n_i} with the total cost $C_{i,j} = \sum_{k=1}^{n_i} c_{i,j,k}$ where $c_{i,j,k} = \sum_{l=1}^{m} x_{i,j,k,l} \times \frac{\tau_{i,j,k} \times p_l}{S_l}$ is the cost of $v_{i,j,k}$. $I_{i,j}$ is successfully executed if and only if $T^f_{i,j,n_i} \leqslant D_i$ and $C_{i,j} \leqslant B_{i,j}$. The binary variable $\eta_l \in \{0,1\}$ indicates that VM M_l does not fail ($\eta_l = 1$) or fails ($\eta_l = 0$) which is verified at the beginning of the ℓ-th time window. For resource reliability consideration, the variable $\beta_{i,j} \in \{0,1\}$ is introduced. $\beta_{i,j} = \sum_{k=1}^{n_i} \sum_{l=1}^{m} x_{i,j,k,l} \times \eta_l$. $\beta_{i,j} = n_i$ implies that all the VMs executing tasks of $I_{i,j}$ do not fail. $\beta_{i,j} < n_i$ means that at least one task cannot finish. A binary variable $y_{i,j} \in \{0,1\}$ represents whether if $I_{i,j}$ is successful or not, i.e., $y_{i,j} = 1$ if and only if the three conditions ($T^f_{i,j,n_i} \leqslant D_i$), ($C_{i,j} \leqslant B_{i,j}$) and ($\beta_{i,j} = n_i$) are true. The success rate of the i-th tenant z_i is defined as $\frac{\sum_{j=1}^{r_i} y_{i,j}}{r_i}$. It is obvious that $z_i \in [0,1]$. For the extreme case, z_i is 0.

In this paper, the QoS for all the n tenants is measured by AISE. For the considered instance-intensive workflow scheduling problem, the objective is to minimize AISE of the tenants, i.e.,

$$Z = -\sum_{i=1}^{n} z_i \log z_i, \tag{2}$$

$$z_i = \max\left\{\frac{\sum_{j=1}^{r_i} y_{i,j}}{r_i}, \frac{1}{e}\right\}. \tag{3}$$

The function $-\sum_{i=1}^{n} a_i \log a_i$ is the entropy of n positive numbers $a_i \in [0,1]$ with the assumption that $0 \log 0 = 0$.

Theorem 1. $-a \log a < -b \log b$ if $0 < a < b < \frac{1}{e}$ and $-a \log a > -b \log b$ if $\frac{1}{e} \leqslant a < b \leqslant 1$.

Proof. For the function $f(x) = -x \log x$, $(0 < x \leqslant 1)$, $f'(x) = -\log x - \frac{1}{\ln 2}$. It follows that $f'(x) > 0$ when $0 < x < \frac{1}{e}$, i.e., $-a \log a < -b \log b$ when $0 < a < b < \frac{1}{e}$. In addition, $f'(x) < 0$ which implies that $-a \log a > -b \log b$ when $\frac{1}{e} \leqslant a < b \leqslant 1$.

Theorem 1 means that the maximum value of $f(z_i) = -z_i \log z_i$ is $\frac{1}{e \ln 2}$, i.e., $f(z_i) \in [0, \frac{1}{e \ln 2}]$. Therefore, $Z \in [0, \frac{n}{e \ln 2}]$.

Theorem 2. For n positive numbers $a_1, \ldots, a_n$, $-\sum_{i=1}^{n} a_i \log a_i \geqslant -(\sum_{i=1}^{n} a_i)(\log \sum_{i=1}^{n} a_i)$.

Proof.

$$\begin{aligned}
&-\sum_{i=1}^{n} a_i \log a_i + \left(\sum_{i=1}^{n} a_i\right)\left(\log \sum_{j=1}^{n} a_j\right) \\
&= -\log \prod_{i=1}^{n} a_i^{a_i} + \log \left(\sum_{i=1}^{n} a_i\right)^{\sum_{i=1}^{n} a_i} \\
&= \log \prod_{i=1}^{n} \left(\frac{\sum_{i=1}^{n} a_i}{a_i}\right)^{a_i} > \log 1 = 0.
\end{aligned}$$

Therefore, $-\sum_{i=1}^{n} (a_i \log a_i) \geqslant -(\sum_{i=1}^{n} a_i)(\log \sum_{i=1}^{n} a_i)$.

Lemma 1. For $\forall x, y \in (0,1)$ and $x + y > 1$, $-x \log x - y \log y + (x + y - 1) \log(x + y - 1) > 0$.

Proof. The function $f(x,y) = (x+y-1)\log(x+y-1) - x \log x - y \log y$ is differentiable in (0,1). For the two extreme points $A = (0,0)$ and $B = (1,1)$, $\boldsymbol{l}$ is the line from A to B. According to the monotonicity discriminant rule of a multivariate function, the directional vector of $\boldsymbol{l}$ is $\boldsymbol{l} = \overrightarrow{AB} = \Delta x \cdot i + \Delta y \cdot j = 1 \cdot i + 1 \cdot j$. The direction cosine is $\cos \alpha = \frac{\Delta x}{\sqrt{(\Delta x)^2 + (\Delta y)^2}} = \frac{1}{\sqrt{2}}$ and $\cos \beta = \frac{\Delta y}{\sqrt{(\Delta x)^2 + (\Delta y)^2}} = \frac{1}{\sqrt{2}}$ where α is the angle

between $\boldsymbol{l}$ and the horizontal axis while β is that between $\boldsymbol{l}$ and the vertical axis. We can obtain

$$\frac{\partial f}{\partial l} = \frac{\partial f}{\partial x}\cos\alpha + \frac{\partial f}{\partial y}\cos\beta = \log\left(\frac{x+y-1}{x}\right)\frac{1}{\sqrt{2}} + \log\left(\frac{x+y-1}{y}\right)\frac{1}{\sqrt{2}}$$
$$= \frac{1}{\sqrt{2}}\log\left[\left(\frac{x+y-1}{x}\right)\left(\frac{x+y-1}{y}\right)\right].$$

Since $0 < x+y-1 < x$ and $0 < x+y-1 < y$, $\frac{\partial f}{\partial l} = \frac{1}{\sqrt{2}}\log[(\frac{x+y-1}{x})(\frac{x+y-1}{y})] < \frac{1}{\sqrt{2}}\log 1 = 0$. Because $f(x,y)$ is decreasing monotonously in the direction $\overrightarrow{AB}$, we can obtain $f(x,y) > f(1,1) = 0$, i.e., $f(x,y) = (x+y-1)\log(x+y-1) - x\log x - y\log y > 0$.

Theorem 3. Let $k = \lfloor\sum_{i=1}^{n} a_i\rfloor$ and $\sum_{i=1}^{n} a_i = k + b$ (i.e., $0 \leqslant b = \sum_{i=1}^{n} a_i - k < 1$) for $\forall a_i \in [0,1]$, $-\sum_{i=1}^{n} a_i \log a_i \geqslant k \times (-1 \times \log 1) - b\log b - (n-k-1)0\log 0 = -b\log b$.

Proof. We use the mathematical induction method for the proof.

(1) $n = 1$ Since $a_1 \in [0,1]$, there are two cases. (i) $a_1 \in [0,1)$. The fact that $k = 0$ and $b = a_1$ implies that $-a_1\log a_1 = -b\log b$. (ii) $a_1 = 1$. The fact that $k = \lfloor a_1\rfloor = 1$ and $b = 0$ means that $-a_1\log a_1 = 0 = -b\log b = 0$.

(2) $n = 2$. Since $0 \leqslant a_1 + a_2 \leqslant 2$, there are three cases:

(i) $0 \leqslant a_1 + a_2 < 1$. Then $k = \lfloor a_1 + a_2\rfloor = 0$, $b = a_1 + a_2 - k = a_1 + a_2$. According to Theorem 2, $-a_1\log a_1 - a_2\log a_2 \geqslant -(a_1+a_2)\log(a_1+a_2) = -b\log b$.

(ii) $a_1 + a_2 = 1, 2$. $k = 1, 2$ and $b = 0$ which illustrate that $-a_1\log a_1 - a_2\log a_2 \geqslant 0 = -b\log b$.

(iii) $1 < a_1 + a_2 < 2$. There are three subcases: (a) $a_1, a_2 \in (0,1)$. $k = \lfloor a_1 + a_2\rfloor = 1$, $b = a_1 + a_2 - k = a_1 + a_2 - 1$. According to Lemma 1, $-a_1\log a_1 - a_2\log a_2 > -(a_1+a_2-1)\log(a_1+a_2-1) = -b\log b$. (b) $a_1 = 1$ and $a_2 \in (0,1)$. $k = 1$ and $b = a_2$ which means that $-a_1\log a_1 - a_2\log a_2 = -a_2\log a_2 = -b\log b$. (c) $a_1 \in (0,1)$ and $a_2 = 1$. Similarly to (b), we have $-a_1\log a_1 - a_2\log a_2 = -b\log b$. Therefore, $-a_1\log a_1 - a_2\log a_2 \geqslant -b\log b$.

(3) Suppose that $-\sum_{i=1}^{m-1} a_i\log a_i \geqslant -b\log b$ is true when $n = m-1$. Let $\sum_{i=1}^{m-1} a_i = k' + b'$ where $k' = \lfloor\sum_{i=1}^{m-1} a_i\rfloor$ and $b' \in [0,1)$. Therefore, $-\sum_{i=1}^{m-1} a_i\log a_i \geqslant k' \times (-1\times\log 1) - b'\log b' - (m-k'-1)0\log 0 = -b'\log b'$.

When $n = m$, $-\sum_{i=1}^{m} a_i\log a_i \geqslant -b'\log b' - a_m\log a_m$. $b' \in [0,1)$ and $a_m \in [0,1]$ implies that $0 \leqslant b' + a_m < 2$. There are three cases in total:

(i) $0 \leqslant b' + a_m < 1$. Let $k = k'$ and $b = b' + a_m$. In terms of Theorem 2, $-b'\log b' - a_m\log a_m \geqslant -(b'+a_m)\log(b'+a_m) = -b\log b$.

(ii) $b' + a_m = 1$. Let $k = k' + 1$ and $b = 0$. In terms of Theorem 2, $-b'\log b' - a_m\log a_m \geqslant -(b'+a_m)\log(b'+a_m) = -b\log b = 0$.

(iii) $1 < b' + a_m < 2$. Let $k = k' + 1$ and $b = b' + a_m - 1$. Obviously, $b \in (0,1)$. (a) $a_m \in (0,1)$. According to Lemma 1, $-b'\log b' - a_m\log a_m > -(b'+a_m-1)\log(b'+a_m-1) = -b\log b$. (b) $a_m = 1$. $b = b'$. $-b'\log b' - a_m\log a_m = -b\log b$. Therefore, $-b'\log b' - a_m\log a_m \geqslant -b\log b$.

Based on (1)–(3), $\forall a_i \in [0,1]$, $k = \lfloor\sum_{i=1}^{n} a_i\rfloor$ and $b = \sum_{i=1}^{n} a_i - k$. $-\sum_{i=1}^{n} a_i\log a_i \geqslant k\times(-1\times\log 1) - b\log b - (n-k-1)0\log 0 = -b\log b$.

According to Theorems 2 and 3, we can deduce Theorem 4 for the more general cases.

Theorem 4. For r groups of positive numbers $\{a_1^1, \ldots, a_{n_1}^1\}, \ldots, \{a_1^r, \ldots, a_{n_r}^r\}$, $-\sum_{k=1}^{r}\sum_{i=1}^{n_k}(a_i^k\log a_i^k) \geqslant -\sum_{k=1}^{r}[(\sum_{i=1}^{n_k} a_i^k)(\log\sum_{i=1}^{n} a_i^k)]$. Let $b_k = \sum_{i=1}^{n_k} a_i^k - \lfloor\sum_{i=1}^{n_k} a_i^k\rfloor \in [0,1)$ $(k = 1, \ldots, r)$, $-\sum_{k=1}^{r}\sum_{i=1}^{n_k}(a_i^k\log a_i^k) \geqslant -\sum_{k=1}^{r} b_k\log b_k$.

Theorem 3 illustrates that the entropy of the vector

$$(\underbrace{1, \ldots, 1}_{k}, b, \underbrace{0, \ldots, 0}_{n-k-1})$$

is not more than that of $(a_1, \ldots, a_n)$ where $b \in [0,1)$ and $a_i \in [0,1]$. In the considered scenario, it is desirable to maximize the number of tenants who are totally satisfied (i.e., all of their requests are met) rather than the tenant only being partially satisfied. Therefore, tenants with the former success rates are more satisfied than those with the latter, i.e., less instance success entropy means a better QoS. In other words, the QoS of all the tenants can be maximized by minimizing the AISE metric. Similarly, Theorem 4 demonstrates that the QoS of all tenants can be maximized if the number of fully satisfied tenants is maximized by sacrificing a few tenants with smaller instance success rates. This conclusion is in accordance with practical applications.

4 Proposed algorithm

Generally, some VMs will break down with uncertainty. As a result, it is necessary to check the statuses of all VMs at every time window. In this paper, we propose a framework referred to as WSRR (workflow scheduling with resource reliability) for the considered problem. Tasks are scheduled time window by time window. During the current ℓ-th time window, three operations are carried out: execution of the tasks scheduled in the $(\ell-1)$-th time window, calculation of the resource reliability of the current time window, and scheduling ready tasks to VMs for the next $(\ell+1)$-th time window. Ready tasks are collected into the set $\mathbb{S}^R$, among which tasks are repeatedly sorted and scheduled to the VMs with the highest priority. In each of the following time windows, the scheduled tasks are executed and the resource reliability of all VMs is updated. After all tasks for all tenants are scheduled, the objective value is computed by (2). The proposed WSRR framework is formally depicted in Algorithm 1. The computational time complexity of Algorithm 1 is hard to determine because of the uncertain number of iterations which depends on the size of the time window.

Algorithm 1 WSRR

```
Initialize all parameters by initialization procedure (IP) (Algorithm 2);
repeat
    if ℓ > 1 then
        Call resource reliability updating (RRU) (Algorithm 6) to update resource reliability;
    end if
    Call task scheduling (TAS) (Algorithm 5) to schedule tasks in 𝕊^R;
    ℓ ← ℓ + 1;
until (𝕊^R = ∅)
Compute Z by Eq. (2);
return Z.
```

4.1 IP

Resource reliabilities are initialized based on historical data. The deadlines and budgets of all workflow instances for each tenant are divided into task sub-deadlines and sub-budgets based on the average speed of all VMs. Tasks are sorted by the corresponding sub-deadlines and sub-budgets before being scheduled. These task constraints are soft and might be violated to some extent. The considered optimization objective is the maximization of the number of tenants with all requests satisfied. This implies that all tenants have to be sorted before task scheduling. We denote the sorted tenant sequence as $(G_{[1]}, \ldots, G_{[n]})$. Instances of tenants with high priorities are selected. At the beginning of the first time window (or the 0th time window), ready tasks are collected to initialize the ready task set $\mathbb{S}^R$. To determine which tasks might be ready tasks, we construct an "artificial" VM which has a speed that is the average speed on m VMs, i.e., $\frac{\sum_{l=1}^{m} S_l}{m}$. The average execution time $\overline{T}^e_{i,j,k}$ of task $v_{i,j,k}$ is defined by

$$\overline{T}^e_{i,j,k} = \frac{\tau_{i,j,k} \times m}{\sum_{l=1}^{m} S_l}. \tag{4}$$

Two time windows are preserved for each VM. The estimated number of selected tasks is $\frac{2mT^w}{\overline{T}^e_{i,j,k}} = \frac{2T^w \sum_{l=1}^{m} S_l}{\tau_{i,j,k}}$, i.e., $\lceil \frac{2T^w \sum_{l=1}^{m} S_l}{\tau_{i,j,k}} \rceil$ tasks are randomly selected from $G_{[1]}$ following their topological orders (or without violating their precedence constraints). In the extreme case, $G_{[1]}$ contains fewer tasks than the number required. The remaining tasks are similarly selected from $G_{[2]}$. Assume the selected tasks are ready during the 0th time window. The tasks scheduled in the first time window are initialized to $\mathbb{S}^R$. During each time window, the resource reliabilities of all VMs are updated. Ready tasks in $\mathbb{S}^R$ are sorted to π^R. Tasks in π^R are sequentially scheduled. The IP is depicted in Algorithm 2 with the computational time complexity being $O(\max\{nm, nr_i n_i\})$.

4.2 Deadline and BD

The batch of instances for every tenant G_i are constrained by the same deadline D_i. Each instance in the batch is constrained by $B_{i,j}$. In order to schedule tasks to VMs in a balanced way, it is necessary to reasonably divide D_i and $B_{i,j}$ into sub-deadlines and sub-budgets in the corresponding tasks. This is performed by deadline division (DD) and by budget division (BD) explained in the following.

Algorithm 2 IP

$\ell \leftarrow 1$, $\mathbb{S}^R \leftarrow \varnothing$;
for $i = 1$ to n **do**
  for $j = 1$ to r_i **do**
    $T^s_{i,j,0} \leftarrow 0$;
  end for
  Call deadline division (DD) (Algorithm 3) to divide D_i into sub-deadlines in all tasks;
  for $j = 1$ to r_i **do**
    Call budget division (BD) (Algorithm 4) to divide $B_{i,j}$ into sub-budgets in all tasks;
  end for
end for
for $l = 1$ to m **do**
  Initialize the sample size N^l_0;
  $\hat{\theta}_l \leftarrow 1$;
  $T^{\text{avail}}_l \leftarrow 0$;
end for
Call the tenants sequencing (TES) (Algorithm 5) rule to get a sorted tenant sequence $\pi^G = (G_{[1]}, \ldots, G_{[n]})$;
Randomly select $\lceil \frac{2T^w \sum_{l=1}^m S_l}{\tau_{i,j,k}} \rceil$ tasks from π^G sequentially to $\mathbb{S}^R$;
Restore start times of VMs and instances to 0;
Call TAS to schedule tasks $\mathbb{S}^R$;
$\mathbb{S}^R$ is updated by tasks scheduled in the first time window;
return

4.2.1 *DD*

To avoid strong competition for attracting resources, it is desirable to evenly allocate all tasks in the same batch. Though instances from the same workflow model (or tenant) have the same number of tasks in the considered problem, task sizes of the same activity are usually different. In addition, all VMs are heterogeneous, i.e., they have different processing speeds. It is hard to calculate or estimate the completion time of each instance in advance. Therefore, the resulting DD problem is challenging. An alternative way is computing the completion time using the average execution times of its tasks. In addition, the number of identical tasks in any instance in a batch implies that it is possible to evenly divide the deadline of a tenant approximately into sub-deadlines for the corresponding instances.

As $\overline{T}^s_{i,j,0} = 0$ for any instance $I_{i,j}$, the average earliest finish time $\overline{T}^f_{i,j,k}$ for each task $v_{i,j,k}$ can be recursively computed by

$$\overline{T}^f_{i,j,k} = \overline{T}^s_{i,j,k} + \overline{T}^e_{i,j,k}, \tag{5}$$

$$\overline{T}^s_{i,j,k} = \max_{k' \in \mathrm{PR}_{i,j,k}} \left\{ \overline{T}^f_{i,j,k'} \right\}. \tag{6}$$

The average execution time $\overline{T}^e_{i,j}$ of instance $I_{i,j}$ is $\overline{T}^f_{i,j,n_i}$ (or $\overline{T}^f_{i,j,n_i+1}$). Generally, the instances' average completion times for the same tenant are different which implies that the floats (differences between the deadline and the average execution durations) are also different. Commonly, more floats are allocated to longer instances. The r_i instances of tenant G_i are sorted in non-decreasing order of average execution times $\overline{T}^e_{i,j}$, which is denoted as $(I_{i,[1]}, \ldots, I_{i,[r_i]})$. The total float of the i-th tenant is the difference between the deadline D_i and the longest average execution time $\overline{T}^e_{i,[r_i]}$, i.e., $\max\{D_i - \overline{T}^e_{i,[r_i]}, 0\}$. The float $\delta_{i,[j]}$ divided into instance $I_{i,[j]}$ is computed by

$$\delta_{i,[j]} = \frac{\max\{D_i - \overline{T}^e_{i,[r_i]}, 0\} \times \overline{T}^e_{i,[j]}}{\sum_{j=1}^{r_i} \overline{T}^e_{i,[j]}}. \tag{7}$$

The sub-deadline $\widetilde{D}_{i,[j]}$ of $I_{i,[j]}$ is determined by

$$\widetilde{D}_{i,[j]} = \overline{T}^e_{i,[j]} + \sum_{k=1}^{j} \delta_{i,[k]}. \tag{8}$$

Furthermore, tasks in an instance are precedence constrained. It is unnecessary to divide the instance float into task floats. In this paper, we only set the float time of each task $v_{i,j,k}$ as that of the instance $I_{i,j}$, i.e., $\widetilde{D}_{i,j} - \overline{T}^e_{i,j}$ in terms of (8). Therefore, the sub-deadline $\widetilde{d}_{i,j,k}$ of $v_{i,j,k}$ is

$$\widetilde{d}_{i,j,k} = \overline{T}^f_{i,j,k} + \widetilde{D}_{i,j} - \overline{T}^e_{i,j}. \tag{9}$$

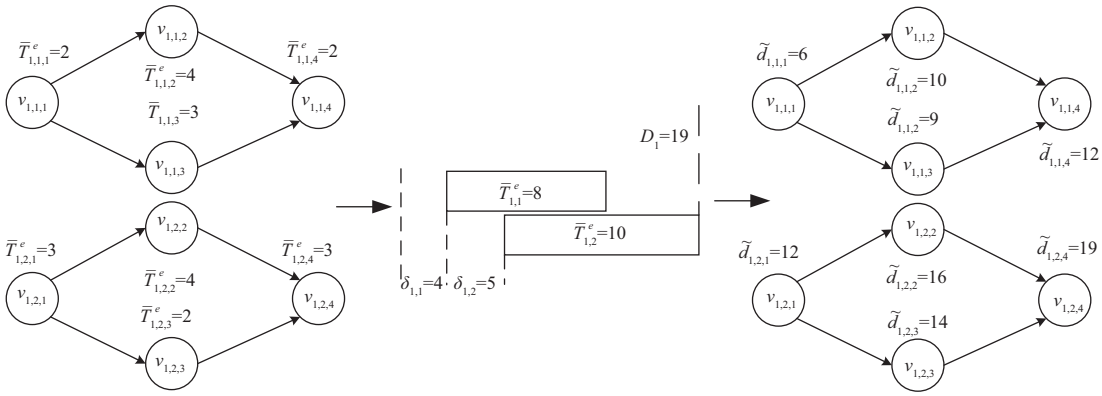

Figure 2 An example for DD.

The DD method is formally depicted in Algorithm 3 and has a computational complexity of $O(\max\{m, r_i n_i\})$.

Algorithm 3 DD

Input: Deadline D_i of tenant G_i.
1: $S \leftarrow 0$;
2: **for** $l = 1$ to m **do**
3: $S \leftarrow S + S_l$;
4: **end for**
5: **for** $j = 1$ to r_i **do**
6: **for** $k = 1$ to n_i **do**
7: $\overline{T}^e_{i,j,k} \leftarrow \frac{\tau_{i,j,k} \times m}{S}$;
8: Compute the average earliest finish and start times $\overline{T}^f_{i,j,k}$ and $\overline{T}^b_{i,j,k}$ using Eqs. (5) and (6);
9: **end for**
10: $\overline{T}^e_{i,j} \leftarrow \overline{T}^f_{i,j,n_i}$;
11: **end for**
12: Generate $(I_{i,[1]}, \ldots, I_{i,[r_i]})$ by sorting all instances of G_i in non-decreasing order of $\overline{T}^e_{i,j}$;
13: **for** $j = 1$ to r_i **do**
14: Calculate the sub-deadline $\widetilde{D}_{i,[j]}$ of $I_{i,[j]}$ using Eqs. (7) and (8);
15: **for** $k = 1$ to n_i **do**
16: Calculate the sub-deadline $\widetilde{d}_{i,[j],k}$ of $v_{i,[j],k}$ using Eq. (9);
17: **end for**
18: **end for**
19: **return**

Figure 2 shows an example of a division of deadlines of instances into those of tasks. There are two instances required by a tenant, which are constrained by the same deadline D_1. Each instance has four tasks. Initially, the average execution times of all tasks are estimated. The critical path of the instance $I_{1,1}$ is ($v_{1,1,1}$, $v_{1,1,2}$ and $v_{1,1,4}$) with $\overline{T}^e_{1,1}$ being 8. The critical path of $I_{1,2}$ is ($v_{1,2,1}$, $v_{1,2,2}$ and $v_{1,2,4}$) with $\overline{T}^e_{1,2}$ being 10. Therefore, $\delta_{1,1}$ is 4 and $\delta_{1,2}$ is 5. The sub-deadlines of $I_{1,1}$ and $I_{1,2}$ are 12 and 19 respectively. The obtained sub-deadlines of tasks are shown in Figure 2.

4.2.2 *BD*

Since the deadline $\widetilde{D}_{i,j}$ of $I_{i,j}$ is obtained by assuming all tasks have been executed on the artificial VM with an average speed, $\widetilde{D}_{i,j}$ is an approximated value. However, the budget $B_{i,j}$ for each instance $I_{i,j}$ is deterministic. Therefore, different division methods are adopted. $B_{i,j}$ is dynamically divided in terms of the progress of $I_{i,j}$ at every time window.

Suppose μ tasks have been scheduled so far for the task sequence $\pi_{i,j} = (v_{i,j,[1]}, \ldots, v_{i,j,[n_i]})$, i.e., tasks $v_{i,j,[1]}, \ldots, v_{i,j,[\mu]}$ have been executed or are being executed. The total cost is $C'_{i,j} = \sum_{k=1}^{\mu} c_{i,j,[\mu]}$ and the remaining budget $B'_{i,j} = B_{i,j} - C'_{i,j}$. The set of unscheduled tasks is $S^u_{i,j} = V_{i,j} - \{v_{i,j,[1]}, \ldots, v_{i,j,[\mu]}\}$. For any task $v_{i,j,k} \in S^u_{i,j}$, the divided budget $\widetilde{b}_{i,j,k}$ is determined by

$$\widetilde{b}_{i,j,k} = \frac{\overline{T}^e_{i,j,k}}{\sum_{\forall v_{i,j,k'} \in S^u_{i,j}} \overline{T}^e_{i,j,k'}} \times B'_{i,j}. \tag{10}$$

The BD process is formally depicted in Algorithm 4 with a computational complexity of $O(n_i)$.

Algorithm 4 BD

Input: $B'_{i,j}$, $S^u_{i,j}$.
$\zeta \leftarrow 0$, $\Psi \leftarrow \emptyset$;
for $v_{i,j,k} \in S^u_{i,j}$ **do**
  $\zeta \leftarrow \zeta + \overline{T}^e_{i,j,k}$;
end for
for $v_{i,j,k} \in S^u_{i,j}$ **do**
  $\widetilde{b}_{i,j,k} \leftarrow \frac{\overline{T}^e_{i,j,k}}{\zeta} \times B'_{i,j}$;
  $\Psi \leftarrow \Psi \bigcup \{\widetilde{b}_{i,j,k}\}$;
end for
return Ψ.

4.3 TES

The objective of the considered problem is to minimize AISE by which the QoS for all the tenants is maximized. According to Theorems 3 and 4, minimizing AISE is the same as maximizing the number of tenants where all instances are completely fulfilled (i.e., completely satisfying as many tenants as possible). Therefore, it is more important to determine the priorities of the tenants than just to sort all tasks. Instances with higher tenant priorities are scheduled earlier than those with lower priorities. Instances of the same tenant have the same priority. In other words, all tenants are sequenced. All instances with the highest priority are scheduled first. In this paper, four candidates are proposed for the TES procedure.

• EDF (earliest deadline first). All tenants are sorted in non-decreasing order of their deadlines D_i.

• MBF (minimum budget first). All tenants are sorted in non-decreasing order of their total budgets $\sum_{j=1}^{r_i} B_{i,j}$.

• MIF (minimum number of instances first). All tenants are sorted in non-decreasing order of their number of instances r_i.

• MWF (minimum workload first). All tenants are sorted in non-decreasing order of their total computational burdens $\sum_{j=1}^{r_i} \sum_{k=1}^{n_i} \tau_{i,j,k}$.

The computational time complexities of EDF and MIF are $O(n \log n)$. The computational time complexity of MBF is $O(\max\{n \log n, \max\{nr_i\}\})$ and that of MWF is $O(\max\{n \log n, \max\{nr_i n_i\}\})$.

4.4 TAS

During each time window, ready tasks are scheduled which include the unscheduled ready tasks in the previous time windows and the immediate successors of the executing tasks. Because of the deadline and budget constraints of tenants, each task is constrained by the sub-deadline and a sub-budget which are determined using the methods presented in the previous sub-section. Since the two sub-constraints are only estimated, they are soft, i.e., they might be violated to some extent. However, it is necessary to sequence all ready tasks to leave more time and budget for remaining tasks by which the success entropy can be minimized. In this paper, two task scheduling rules are developed.

• ESDF (earliest sub-deadline first). All ready tasks are sorted in non-decreasing order of their sub-deadlines determined by (9). Tie-breaking is done randomly.

• SSBF (smallest sub-budget first). All ready tasks are sorted in non-decreasing order of their sub-budgets determined by (10). Tie-breaking is done randomly.

Any available VMs can be allocated to ready tasks. In practice, tenants want their tasks to be allocated to reliable and cheap resources. In other words, all VMs are sorted by their resource reliability and prices. Generally, tenants care more about resource reliability than prices. Resource reliability of the past $\ell - 1$ time windows can be obtained by (1). Therefore, we define the resource priority P^R_l for M_l as

$$P^R_l = wp_l + m(1 - \hat{\theta}_l(\xi_l, \ell - 1; N_l, u_l)). \tag{11}$$

w is a normalization weight (which takes 1 in the following experimental results). Eq. (11) implies that a very small decrease in reliability would lead to a big increase in priority since m is usually very big. All VMs are sorted in non-decreasing order of their resource priorities computed by (11). The sorted VM sequence is denoted as $\pi^M = \{M_{[1]}, \ldots, M_{[m]}\}$.

For each VM $M_{[l]}$ in π^M, we try to allocate as many tasks as possible from the ready task sequence π^R to it within the current time window. In other words, the ready task $\pi^R_{[1]}$ is allocated to a VM $M_{[l]}$ if and only if $M_{[l]}$ works well ($\eta_{[l]}=1$) and the available time $T^{\text{avail}}_{[l]}$ does not exceed the next time window (the $(\ell+1)$-th time window). There are two cases for the allocation:

(i) $\pi^R_{[1]}$ is successfully allocated to $M_{[l]}$. $\pi^R_{[1]}$ is appended to the end of $\text{List}_l^{(\ell+1)}$ and is removed from π^R and $T^{\text{avail}}_{[l]}$ of $M_{[l]}$ is updated. All immediate successors of $\pi^R_{[1]}$ are added to $\mathbb{S}^R$.

(ii) No VM meeting the requirements of $\pi^R_{[1]}$ means that $\pi^R_{[1]}$ cannot be scheduled in the $(\ell+1)$-th time window. $\pi^R_{[1]}$ is rolled back to the ready set $\mathbb{S}^R$.

The TAS procedure is described in Algorithm 5. The computational time complexity of Algorithm 5 is hard to determine because the number of tasks in π^R always changes, it is polynomial since each task is checked, at most, m times.

Algorithm 5 TAS

Input: Resource reliability vector ($\hat{\boldsymbol{\theta}}$) of the $\ell-1$ time windows, the ready task set $\mathbb{S}^R$, the available time vector $\boldsymbol{T}^{\text{avail}}$.

Construct the ready task sequence π^R by a task scheduling rule (ESDF or SSBF);
for $l=1$ to m **do**
  $P_l^R \leftarrow wp_l + m(1-\hat{\theta}_l(\xi,\ell-1;N,u))$;
end for
Construct $\pi^M=\{M_{[1]},\ldots,M_{[m]}\}$ in non-decreasing order of priorities P_l^R;
while $\pi^R \neq$ NULL **do**
  Find the task $v_{i,j,k}$ corresponding to $\pi^R_{[1]}$;
  $T^s_{i,j,k} \leftarrow \max_{v_{i,j,k'}\in \text{PR}_{i,j,k}}\{T^f_{i,j,k'}\}$;
  $l \leftarrow 1$;
  while ($\eta_{[l]}=1$ and $l \leqslant m$) **do**
    if $T^{\text{avail}}_{[l]} \leqslant (\ell+1)\times T^w$ **then**
      $\pi^R_{[1]}$ is moved from π^R to the end of $\text{List}_l^{(\ell+1)}$;
      $T^{\text{avail}}_{[l]} \leftarrow T^{\text{avail}}_{[l]} + \frac{\tau_{i,j,k}}{S_l}$;
      $T^f_{i,j,k} \leftarrow T^s_{i,j,k} + \frac{\tau_{i,j,k}}{S_l}$;
      Add all immediate successors of $\pi^R_{[1]}$ to $\mathbb{S}^R$;
      Break;
    else
      $l \leftarrow l+1$;
    end if
  end while
  if $l>m$ **then**
    $\pi^R_{[1]}$ is moved from π^R to $\mathbb{S}^R$;
  end if
end while
return

4.5 RRU

For each VM M_l, the priority is computed by (11) in terms of the resource reliability $\hat{\theta}_l(\xi_l,\ell-1;N_l,u_l)$ for the past $\ell-1$ time windows. $\hat{\theta}_l(\xi_l,\ell-1;N_l,u_l)$ is updated in terms of the collected status $\eta_l\in\{0,1\}$ at the beginning of the current (the ℓ-th) time window.

- $\eta_l=1$ means that M_l works well. ξ_l is increased by 1. Both N_l and u_l are increased by $|\text{List}_l^{(\ell-1)}|$.
- $\eta_l=0$ means that M_l breaks down. ξ_l is set as 0. N_l is increased by $|\text{List}_l^{(\ell-1)}|$. No increase is made on u_l since it is unknown when M_l broke down in the past time window. For each task $v_{i,j,k}\in\text{List}_l^{(\ell-1)}$, $y_{i,j}$ is set as 0 which means that the successive tasks of $I_{i,j}$ are not executed any more. $I_{i,j}$ fails (because of the instance-intensive environment).

The resource reliability updating method is formally depicted in Algorithm 6 with the computational time complexity being $O(m)$.

5 Computational experiments

There are several candidates for each component or each parameter of the proposed WSRR framework. We calibrate the parameters and components statistically over a comprehensive set of random instances

Algorithm 6 RRU

for $l = 1$ to m **do**
  Collect the status η_l of M_l;
  if $\eta_l = 1$ **then**
    $\xi_l \leftarrow \xi_l + 1$;
    $N_l \leftarrow N_l + |\mathrm{List}_l^{(\ell-1)}|$;
    $u_l \leftarrow u_l + |\mathrm{List}_l^{(\ell-1)}|$;
  else
    $\xi_l \leftarrow 0$;
    $N_l \leftarrow N_l + |\mathrm{List}_l^{(\ell-1)}|$;
    for $v_{i,j,k} \in \mathrm{List}_l^{(\ell-1)}$ **do**
      $y_{i,j} \leftarrow 0$;
    end for
  end if
  Compute $\hat{\theta}_l(\xi_l, \ell-1; N_l, u_l)$ using Eq. (1);
end for
return

Table 2 VMs attributes for the instances and experiments

VM type	Processing speed	Unit price
1	0.5	3
2	1	6
3	2	12
4	4	24

first. Two algorithms are compared over real applications. All tested algorithms are coded in Java and run on an Intel(R) Core(TM) i7-4770 CPU @3.40 GHz computer with 8 GBytes of RAM.

CloudSim is used for the experiments which supports modeling and simulations of large-scale cloud computing environments on a single computing node, including service brokers, data centers, VMs and resource provisioning. There are many workflow instances and generally each instance has several tasks for the instance-intensive workflows. The number of tenants n takes values from {20, 40, 60, 80, 100}. Similarly to [31], the number of workflow instances r_i follows a uniform distribution $U(100, 180)$, the number of tasks n_i in each instance follows a uniform distribution $U(3, 9)$ and the computation burden of each task $\tau_{i,j,k}$ also follows a uniform distribution $U(10, 120)$. Similar to the parameter settings in [22], the sampling error ε takes a value of 0.1 and the confidence level threshold γ_0 is 0.95. Therefore, the sample size N_0 is set to 185. For each n, 10 experimental instances are randomly generated, i.e., 50 random experimental instances in total are conducted for parameter and algorithm component calibration. The cloud computing platform provides different types of resources with VM types, processing speeds and unit prices as is shown in Table 2 and which is identical to that in [27]. The number of resources of each type is 500. The initial resource failure rate of VMs follows uniform distribution $U(1, 10) \times 10^{-5}$.

Let $Z(A)$ be the AISE value obtained by algorithm A and Z^* be the best AISE value obtained by all algorithms conducted on all treatments in the same instance. According to Theorem 1, $Z(A) \in [0, \frac{n}{e \ln 2}]$ and $Z^* \in [0, \frac{n}{e \ln 2}]$. In this paper, the RPD (relative percentage deviation) is adopted to evaluate the performance of algorithms which is defined as

$$\mathrm{RPD} = \frac{e \ln 2 (Z(A) - Z^*)}{n} \times 100\%. \tag{12}$$

5.1 Parameters and components calibration

In the proposed WSRR framework, there is one parameter T_w and two algorithm components (tenant sequencing and task scheduling) to be calibrated. The time window T_w takes values from {60, 90, 120, 150, 180, 210} s. There are four candidates for tenant sequencing: EDF, MBF, MIF, and MWF. Two candidates are involved in task scheduling: ESDF and SSBF. Therefore, there are $6 \times 4 \times 2 = 48$ combinations in total for WSRR in each instance replicate. Because of stochastic unreliable resources and random tie-breaking in task scheduling, 5 replicates are conducted for each combination and $50 \times 48 \times 5 = 12000$ tests are performed in total for calibration. In addition, the deadline for every tenant takes a random value from the interval $[D_{\max}/2,\ D_{\max}]$ in terms of [33] where $D_{\max}$ is the maximum deadline determined by $D_{\max} = \sum_{I_{i,j} \in G_i} \overline{T}^f_{i,j,k}$.

The budget of an instance takes a random value from $[B_{\max},\ 2 \times B_{\max}]$ where $B_{\max}$ is the maximum

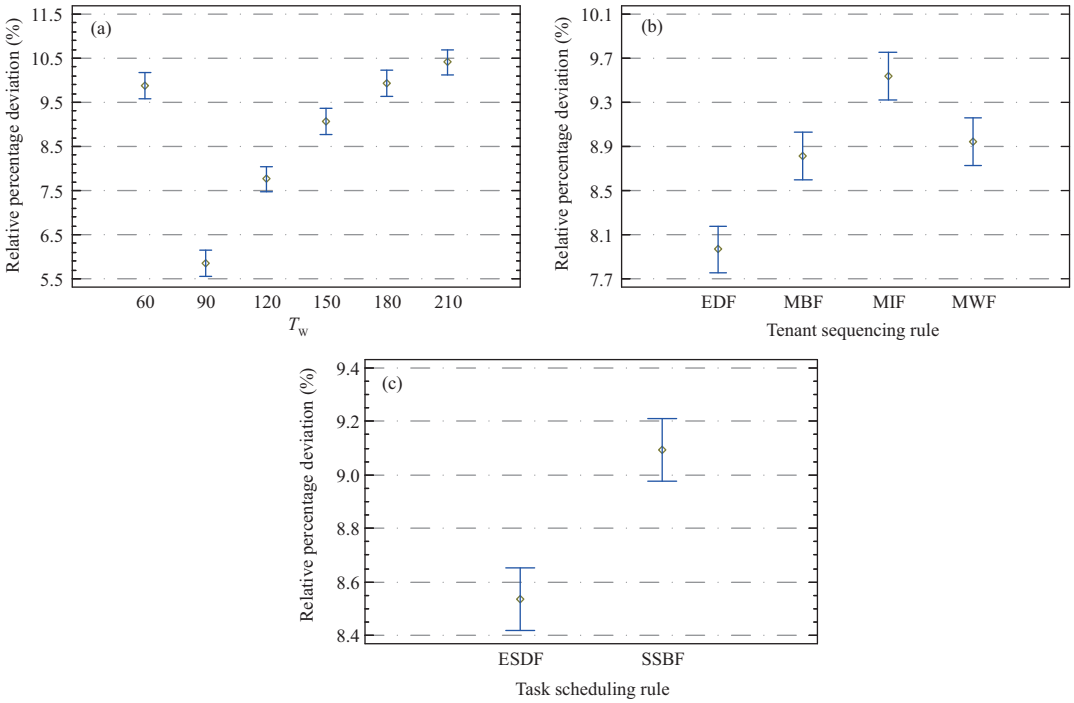

Figure 3 (Color online) (a) Mean plot of T_w with 95% confidence Tukey HSD intervals; (b) mean plot with 95% confidence Tukey HSD intervals for the tenant sequencing rule; (c) mean plot with 95% confidence Tukey HSD intervals for the task scheduling rule.

Table 3 Workflow instances with a different number of tasks

n	3	4	5	6	7	8	9	10	11
Ins.	84222	48225	37909	24454	25481	12815	10471	8380	6979

budget calculated by $\sum_{k=1}^{n_i} c_{i,j,k}$.

Experimental results are analyzed by the multi-factor analysis of variance (ANOVA) statistical technique. The three main hypotheses (normality, homoscedasticity, and independence of the residuals) are checked from the residuals of the experiments. All three hypotheses are acceptable from the analysis. The p-values are lower than 0.05 indicating that all the factors studied have a statistically significant effect on the RPD response variable with a 95% confidence level.

Figure 3(a) shows the means plot of T_w with 95.0% confidence Tukey HSD intervals which implies that the parameter T_w has a statistically significant effect at different values, i.e., T_w has a significant effect on the performance of the algorithm. When T_w is greater than 90 s, RPD increases with an increase in T_w. The RPD of $T_w = 60$ is greater than that of $T_w = 150$. Therefore, the algorithm obtains the best objective value when $T_w = 90$. When the time window size is close to the execution time of a task, the failure of a VM only affects the tasks in that window and only one task is affected in most cases. With an increase in the time window, the number of affected tasks increases which results in worse experimental results. T_w takes 90 s in this paper.

The Tukey HSD interval with 95% confidence intervals for the four tenant sequencing rules is shown in Figure 3(b). Figure 3(b) shows that the RPD of EDF is lower than those of MBF, MIF, and MWF. In other words, the rule which considers the earliest deadline is more effective than the other three tenant sequencing rules in the considered problem. Therefore, we select the EDF rule in this paper.

The Tukey HSD interval with 95% confidence intervals of the two task scheduling rules is shown in Figure 3(c). From Figure 3(c), we can observe that the RPDs of ESDF and SSBF are statistically different. The RPD of ESDF is lower than that of SSBF, which means that the task scheduling which considers the earliest sub-deadline is more effective for the problem under study. The ESDF rule is adopted in this paper.

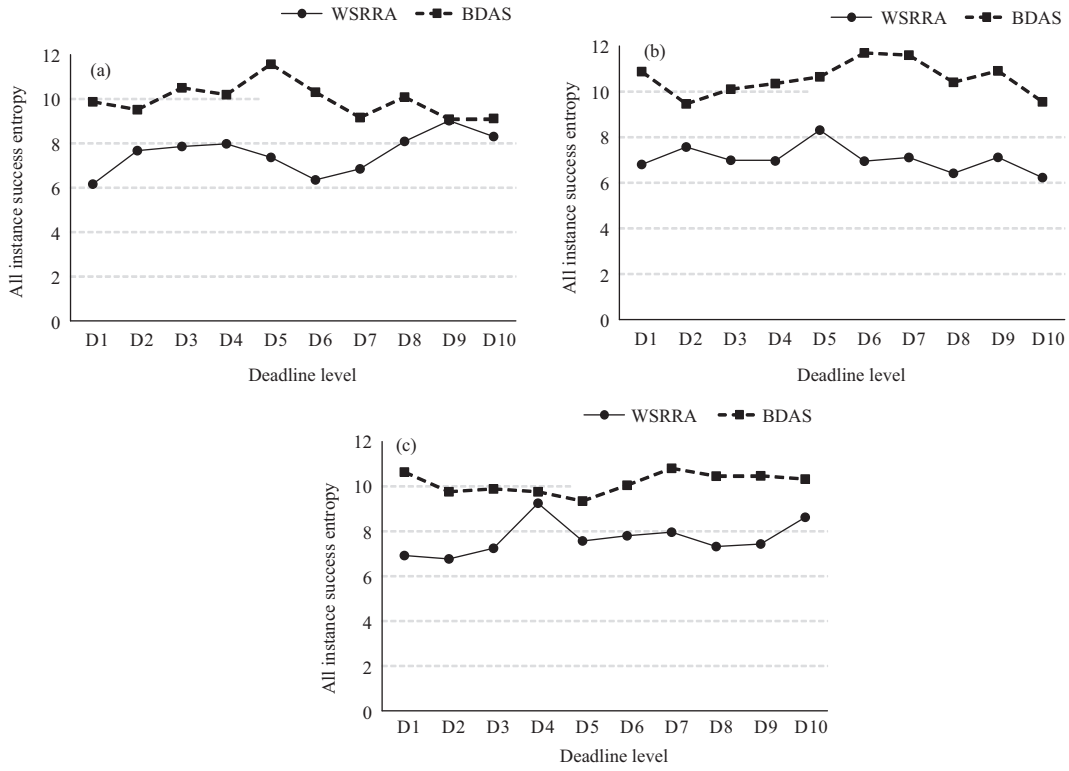

Figure 4 Algorithm comparisons on (a) B_s, (b) B_n, and (c) B_r.

5.2 Algorithm comparisons

The calibrated algorithm is evaluated over a set of practical Alibaba Cluster Data V2018 instances[1)]. We only consider the first 3×10^6 items (or tasks) which are included in 429939 workflow instances. The number of tasks in each workflow instance is a variable with a non-uniform distribution in [1,199]. 90% of the workflow instances have no more than 11 tasks. In addition, the instances with only one and two tasks are special cases. We only consider the workflow data set with tasks in [3,11] as shown in Table 3. Since the workflow model with n ($n = 3, \ldots, 11$) activities is randomly distributed in the considered data set, we randomly select instances. For a workflow with n tasks, we only consider n tenants for simplicity. Therefore $\frac{(3+11)\times 9}{2} = 63$ tenants in total are considered in the evaluation. The number of workflow instances of each tenant r_i follows a uniform distribution $U(20, 30)$.

The problem considered in [2] is similar to the problem under study. The BDAS (budget deadline aware scheduling) algorithm proposed in [2] is compared to the calibrated WSRR algorithm (WSRRA) in this paper. BDAS partitions tasks into different levels based on their dependencies in a workflow instance. The budget and deadline of the workflow instance is distributed to tasks. All tasks in the same level have the same level-deadline and level-budget. BDAS allocates resources to tasks by the CTTF (cost time trade-off factor) index. The resource with the highest CTTF is selected for the current task. To comprehensively evaluate the compared algorithms under different deadlines and budget levels, we divide the deadline range $[D_{\max}/2, D_{\max}]$ into 10 equal intervals $D_1, D_2, \ldots, D_{10}$ and the budget range $[B_{\max}, 2 \times B_{\max}]$ into 3 equal intervals B_s, B_n, and B_r (which represent a strict budget, a normal budget and a relaxed budget respectively).

We also employ the considered objective AISE index to evaluate the performance of the compared algorithms. AISEs on different deadline levels for the three budget scenarios (strict, normal and relaxed) are shown in Figures 4(a)–(c) respectively. It can be seen that the proposed WSRRA outperforms the BDAS across all budget and deadline levels, i.e., AISEs of WSRRA are better than those of BDAS for different budgets and deadlines. The reason lies in that WSRRA takes into account resource reliability when selecting the resources. The impact of budget constraints on task execution is alleviated, to some extent, with large budgets. On the other hand, BDAS considers execution costs or execution times when

1) https://github.com/alibaba/clusterdata.

Table 4 Computation time (ms)

Budget	Algorithm	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
B_s	WSRRA	410	359	254	289	331	321	310	583	338	406
	BDAS	575	482	469	560	642	533	726	583	560	607
B_n	WSRRA	298	264	315	410	326	280	319	264	311	385
	BDAS	581	581	588	620	580	599	570	535	593	594
B_r	WSRRA	396	292	295	291	292	354	347	320	300	414
	BDAS	603	668	627	559	649	602	631	574	617	577

selecting resources, which always leads to poorer performance when resources fail. In addition, BDAS distributes the budgets and deadlines based on the task levels more strictly than WSRRA which results in competition for crucial resources which possibly leads to worse AISEs as shown in Figures 4(a)–(c).

In total, the AISEs of WSRRA are between 6.16–9.25 and those of BDAS lie between 9.08–11.70 for different deadlines. Figure 4(a) demonstrates that the AISE of WSRRA is between 6.16–9.02 while that of the BDAS algorithm is at least 9.08% in the strict budget cases. Figure 4(b) illustrates that the AISE of WSRRA is between 6.23–8.31 whereas that of the BDAS algorithm is at least 9.46% in the normal budget cases. Figure 4(c) demonstrates that the AISE of WSRRA is between 6.77–9.25 but that of the BDAS algorithm is at least 9.76% in the relaxed budget cases.

Table 4 shows the computation time comparisons. We can observe that WSRRA employs shorter computation times than BDAS, sometimes even less than half. The reason lies in that WSRRA selects the first eligible VM when selecting resources, updates the reliability and ranks the VMs at each time window whereas BDAS traverses all VM lists to select the one with the largest CTTF when a task arrives.

6 Conclusion

In this paper, we have considered the instance-intensive workflow scheduling problem with resource reliability in multi-tenant clouds to maximize the QoS. This is formulated to AISE. In terms of the characteristics of the considered problem in a multi-tenant cloud environment, a scheduling architecture is proposed. A hybrid resource reliability assessment method is developed to evaluate resource reliability which takes advantages of two typical resource reliability models. The properties of AISE are analyzed and are in accordance with the QoS for many tenants. For the problem under study, the WSRR framework is proposed which mainly consists of four components: deadline and BD, tenant sequencing, task scheduling and resource reliability updating. After statistically calibrating the algorithm parameters and components, the proposed WSRRA framework is compared to the most recent effective algorithm, BDAS, over a comprehensive set of practical instances. Experimental results show that WSRRA significantly outperforms BDAS in terms of both effectiveness and efficiency.

In the future, problems considering customer experience and methods with more learned knowledge are promising avenues for investigation for multi-tenant cloud workflow scheduling.

Acknowledgements This work was supported by the National Key Research and Development Program of China (Grant No. 2017YFB1400800), National Natural Science Foundation of China (Grant Nos. 61872077, 61832004), and Collaborative Innovation Center of Wireless Communications Technology. Rubén Ruiz was partly supported by the Spanish Ministry of Science, Innovation, and Universities, under the Project "OPTEP-Port Terminal Operations Optimization" (Grant No. RTI2018-094940-B-I00) Financed with FEDER Funds.

References

1 Ghahramani M H, Zhou M C, Hon C T. Toward cloud computing QoS architecture: analysis of cloud systems and cloud services. IEEE/CAA J Autom Sin, 2017, 4: 6–18
2 Arabnejad V, Bubendorfer K, Ng B. Budget and deadline aware e-science workflow scheduling in clouds. IEEE Trans Parallel Distrib Syst, 2019, 30: 29–44
3 Jia Y H, Chen W N, Yuan H, et al. An intelligent cloud workflow scheduling system with time estimation and adaptive ant colony optimization. IEEE Trans Syst Man Cybern Syst, 2021, 51: 634–649
4 Chen L, Li X P. Cloud workflow scheduling with hybrid resource provisioning. J Supercomput, 2018, 74: 6529–6553
5 Kaur P, Mehta S. Resource provisioning and work flow scheduling in clouds using augmented shuffled frog leaping algorithm. J Parallel Distrib Comput, 2017, 101: 41–50
6 Li W L, Xia Y N, Zhou M C, et al. Fluctuation-aware and predictive workflow scheduling in cost-effective infrastructure-as-a-service clouds. IEEE Access, 2018, 6: 61488–61502
7 Wu Q, Zhou M C, Zhu Q, et al. MOELS: multiobjective evolutionary list scheduling for cloud workflows. IEEE Trans Autom Sci Eng, 2020, 17: 166–176
8 Wang S, Li X P, Ruiz R. Performance analysis for heterogeneous cloud servers using queueing theory. IEEE Trans Comput, 2020, 69: 563–576

9 Sun P, Dai Y S, Qiu X W. Optimal scheduling and management on correlating reliability, performance, and energy consumption for multiagent cloud systems. IEEE Trans Rel, 2017, 66: 547–558
10 Roy P, Mahapatra G S, Dey K N. Forecasting of software reliability using neighborhood fuzzy particle swarm optimization based novel neural network. IEEE/CAA J Autom Sin, 2019, 6: 1365–1383
11 Snyder B, Ringenberg J, Green R, et al. Evaluation and design of highly reliable and highly utilized cloud computing systems. J Cloud Comput, 2015, 4: 1–16
12 Zhang P Y, Kong Y, Zhou M C. A domain partition-based trust model for unreliable clouds. IEEE Trans Inform Forensic Secur, 2018, 13: 2167–2178
13 Qiu X W, Dai Y S, Xiang Y P, et al. Correlation modeling and resource optimization for cloud service with fault recovery. IEEE Trans Cloud Comput, 2019, 7: 693–704
14 Wang L. Architecture-based reliability-sensitive criticality measure for fault-tolerance cloud applications. IEEE Trans Parallel Distrib Syst, 2019, 30: 2408–2421
15 Jøsang A, Ismail R, Boyd C. A survey of trust and reputation systems for online service provision. Decision Support Syst, 2007, 43: 618–644
16 Tang X Y, Li K L, Zeng Z, et al. A novel security-driven scheduling algorithm for precedence-constrained tasks in heterogeneous distributed systems. IEEE Trans Comput, 2011, 60: 1017–1029
17 Sonnek J, Chandra A, Weissman J. Adaptive reputation-based scheduling on unreliable distributed infrastructures. IEEE Trans Parallel Distrib Syst, 2007, 18: 1551–1564
18 Kavitha G, Sankaranarayanan V. Secure resource selection in computational grid based on quantitative execution trust. World Acad Sci Eng Technol, 2010, 72: 149–155
19 Wang X, Yeo C S, Buyya R, et al. Optimizing the makespan and reliability for workflow applications with reputation and a look-ahead genetic algorithm. Future Generation Comput Syst, 2011, 27: 1124–1134
20 Wen Z, Cała J, Watson P, et al. Cost effective, reliable and secure workflow deployment over federated clouds. IEEE Trans Serv Comput, 2016, 10: 929–941
21 Zhang L X, Li K L, Li C Y, et al. Bi-objective workflow scheduling of the energy consumption and reliability in heterogeneous computing systems. Inf Sci, 2017, 379: 241–256
22 Wang W, Zeng G S, Tang D Z, et al. Cloud-DLS: dynamic trusted scheduling for cloud computing. Expert Syst Appl, 2012, 39: 2321–2329
23 Tao Y C, Jin H, Wu S, et al. Dependable grid workflow scheduling based on resource availability. J Grid Comput, 2013, 11: 47–61
24 Wang Z B, Wen Y P, Chen J J, et al. Towards energy-efficient scheduling with batch processing for instance-intensive cloud workflows. In: Proceedings of the 16th IEEE International Symposium on Parallel and Distributed Processing with Applications, 2018. 590–596
25 Wen Y P, Wang Z B, Zhang Y, et al. Energy and cost aware scheduling with batch processing for instance-intensive IoT workflows in clouds. Future Generation Comput Syst, 2019, 101: 39–50
26 Liu K, Chen J J, Yang Y, et al. A throughput maximization strategy for scheduling transaction-intensive workflows on SwinDeW-G. Concurr Comput-Pract Exper, 2008, 20: 1807–1820
27 Liu K, Jin H, Chen J J, et al. A compromised-time-cost scheduling algorithm in SwinDeW-C for instance-intensive cost-constrained workflows on a cloud computing platform. Int J High Perform Comput Appl, 2010, 24: 445–456
28 Yang Y, Liu K, Chen J J, et al. An algorithm in SwinDeW-C for scheduling transaction-intensive cost-constrained cloud workflows. In: Proceedings of the 4th International Conference on eScience, 2008. 374–375
29 Li W J, Wu J Y, Zhang Q F, et al. Trust-driven and QoS demand clustering analysis based cloud workflow scheduling strategies. Cluster Comput, 2014, 17: 1013–1030
30 Rodriguez M A, Buyya R. Scheduling dynamic workloads in multi-tenant scientific workflow as a service platforms. Future Generation Comput Syst, 2018, 79: 739–750
31 Cui L Z, Zhang T T, Xu G Q, et al. A scheduling algorithm for multi-tenants instance-intensive workflows. Appl Math Inf Sci, 2013, 7: 99–105
32 Rimal B P, Maier M. Workflow scheduling in multi-tenant cloud computing environments. IEEE Trans Parallel Distrib Syst, 2017, 28: 290–304
33 Malawski M, Juve G, Deelman E, et al. Cost-and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds. In: Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis, 2012. 1–11