



Real-time and dynamic fault-tolerant scheduling for scientific workflows in clouds

Zhongjin Li ^a, Victor Chang ^{b,*}, Haiyang Hu ^a, Hua Hu ^c, Chuanyi Li ^d, Jidong Ge ^d

^a School of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou, China

^b Artificial Intelligence and Information Systems Research Group, School of Computing, Engineering and Digital Technologies, Teesside University, Middlesbrough, UK

^c School of Information Science and Engineering, Hangzhou Normal University, Hangzhou, China

^d State Key Laboratory for Novel Software Technology, Software Institute, Nanjing University, Nanjing, China



ARTICLE INFO

Article history:

Received 20 June 2020

Received in revised form 28 February 2021

Accepted 2 March 2021

Available online 9 March 2021

Keywords:

Cloud computing

Fault-tolerant workflow scheduling

Replication

Checkpointing

Rescheduling

Delay execution

ABSTRACT

Cloud computing has become a popular technology for executing scientific workflows. However, with a large number of hosts and virtual machines (VMs) being deployed, the cloud resource failures, such as the permanent failure of hosts (HPF), the transient failure of hosts (HTF), and the transient failure of VMs (VMTF), bring the service reliability problem. Therefore, fault tolerance for time-consuming scientific workflows is highly essential in the cloud. However, existing fault-tolerant (FT) approaches consider only one or two above failure types and easily neglect the others, especially for the HTF. This paper proposes a Real-time and dynamic Fault-tolerant Scheduling (ReadyFS) algorithm for scientific workflow execution in a cloud, which guarantees deadline constraints and improves resource utilization even in the presence of any resource failure. Specifically, we first introduce two FT mechanisms, i.e., the replication with delay execution (RDE) and the checkpointing with delay execution (CDE), to cope with HPF and VMTF, simultaneously. Additionally, the rescheduling (ReSC) is devised to tackle the HTF that affects the resource availability of the entire cloud datacenter. Then, the resource adjustment (RA) strategy, including the resource scaling-up (RS-Up) and the resource scaling-down (RS-Down), is used to adjust resource demands and improve resource utilization dynamically. Finally, the ReadyFS algorithm is presented to schedule real-time scientific workflows by combining all the above FT mechanisms with RA strategy. We conduct the performance evaluation with real-world scientific workflows and compare ReadyFS with five vertical comparison algorithms and three horizontal comparison algorithms. Simulation results confirm that ReadyFS is indeed able to guarantee the fault tolerance of scientific workflow execution and improve cloud resource utilization.

© 2021 Elsevier Inc. All rights reserved.

1. Introduction

Cloud computing platform leverages virtualization technology to provide flexible, scalable, and unlimited virtual machine (VM) resources for external customers [1,2]. Hence, more and more institutes and enterprises conduct their large-scale applications in the cloud platforms to produce high-quality service and low cost. For example, scientific workflow applications (e.g., physics, bioinformatics, astronomy, numerical weather forecast, etc.) prefer to be deployed on the clouds to decrease execution time and cost [3–5].

* Corresponding author.

E-mail address: victorchang.research@gmail.com (V. Chang).

Although cloud computing brings great benefits for executing scientific workflows, the cloud suffers from multiple types of resource failure, such as the permanent failure of hosts (HPF) [6,7], the transient failure of hosts (HTF) [8,9], and the transient failure of VM (VMTF) [10,11]. It is reported that 0.01% of reliable hosts will be failed every day, and about 1–5% of hard disks die and 2–4% of physical servers crash each year [6]. The above hardware problems cause the HPF. Due to human errors, all the cloud hosts may experience HTF. As an example, the power failure of the datacenter level impacts the power supply of the hosts [9]. In addition, the participant faults, i.e., the conflict between cloud participants (e.g., resource provider and administrator), influence the host availability [8]. VMTF mainly results from the software problem, since VM is a type of middleware and hence has a probability of arising failure [12].

All the failures can trigger the premature termination of task execution, directly impacting the makespan of the scientific workflow. This is because the task failure inevitably elongates its execution time, causing the delay of the entire workflow. However, some workflows are real-time scientific applications that demand an effective completion time. Consequently, developing fault-tolerant (FT) mechanisms for scientific workflow execution is essential in a cloud.

Resubmission and replication are two fundamental FT mechanisms that have been effectively applied in the cluster [13,14], grid [15–17], and cloud [6,18,19]. Resubmission resubmits a task after a failure happens, which can effectively reduce resource consumption. However, it introduces a longer task execution time, resulting in hardly satisfying the deadline constraint [19]. Alternatively, replication duplicates multiple task copies and allocates them to different computational units [18]. For example, the primary-backup (PB) model is one of the replication methods, where only two task copies are scheduled [6,13,20]. Essentially, the more task copies, the higher FT ability but, the lower resource utilization.

Rescheduling is usually applied in the service delay caused by resource dynamics and failures [21,22]. The key idea of rescheduling is to readjust sub-makespans and recompute optimal mapping strategies for unexecuted tasks. For example, if HTF occurs, rescheduling is utilized to adjust the original scheduling scheme for unfinished tasks to compensate for the host and VM downtime. Thus, it has the potential to guarantee the completion time of workflow execution.

HPF, HTF, and VMTF are the three main and common resource failure types. To the best of our knowledge, none of the related work considers all the above resource failures for scientific workflows scheduling in the cloud simultaneously. In order to overcome the resource failure problem, this work presents a Real-time and dynamic Fault-tolerant Scheduling algorithm, called ReadyFS, for scientific workflows in the cloud. ReadyFS is devised to guarantee workflow completion time and improve cloud resource utilization even in the presence of host and VM failures. More specifically, the replication with delay execution (RDE) and checkpointing with delay execution (CDE) are devised to tolerate HPF and VMTF. The rescheduling FT mechanism, ReSC, is introduced to cope with HTF when all the cloud resources are unavailable. In addition, the resource adjustment (RA) strategy, including resource scaling-up (RS-Up) and resource scaling-down (RS-Down), is developed to adjust resource demands and improve resource utilization. By integrating the above FT mechanisms with RA strategy, ReadyFS is presented to implement the real-time FT workflow scheduling. We evaluate the ReadyFS with real-world scientific workflows and compare it with five vertical comparison algorithms and three horizontal comparison algorithms. Simulation results show that ReadyFS is indeed able to guarantee fault-tolerance and improve resource utilization. The main contributions of this paper are listed as follows:

- By incorporating the cloud characteristics and scientific workflow, FT scheduling architecture is established to guarantee fault tolerance and improve resource utilization.
- A sub-makespan allocation is designed to divide the workflow deadline into multiple sub-makespans. Based on the allocated sub-makespan, we propose three FT mechanisms, i.e., RDE, CDE, and ReSC, to guarantee the fault tolerance of scientific workflow execution. Moreover, according to the theoretical proof, RDE and CDE outperform the traditional replication and resubmission methods, respectively.
- We design a dynamic RA strategy, including the RS-Up and RS-Down, to adjust resource demands and achieve high cloud resource utilization, which is applied during the process of workflow execution and scheduling.
- We compare ReadyFS with five vertical and three horizontal comparison algorithms based on real-world scientific workflows. The corresponding experimental results prove the feasibility and effectiveness of our ReadyFS.

The rest of this paper is organized as follows: Section 2 reviews the related work. Section 3 describes scheduling architecture and system models. Section 4 introduces the implementation of the proposed ReadyFS algorithm. Section 5 discusses performance evaluation. Finally, Section 6 gives the conclusions and future work.

2. Related work

Cloud workflow scheduling has been extensively studied. Zhao et al. [3] propose a cloud workflow management system by bringing cloud and workflow together. Wang et al. [23] introduce an attack-defense game model-based scientific workflow scheduling method to improve the security of workflow execution. Rodriguez and Buyya [24] present a resource provisioning method to minimize the workflow execution cost under the pre-defined deadline constraint. Based on the VM pre-allocation scheme and three resource optimization strategies, Li et al. [25] introduce a cost and energy-aware scheduling (CEAS) algorithm for cloud scientific workflows. Tong et al. [26] propose a deep Q-learning-based workflow scheduling algorithm in the cloud. Zhang et al. [5] devise an efficient priority and relative distance (EPRD) algorithm for workflow scheduling to minimize the finish time with the deadline constraint. Many multi-objective algorithms for workflow scheduling have

been proposed, as well. For example, Durillo et al. [27] present a Pareto-based list multi-objective workflow scheduling algorithm. Zhu et al. [28] propose an evolutionary algorithm-based multi-objective scheduling method. Kalra and Singh [29] apply the Intelligent Water Drops Algorithm and Genetic Algorithm jointly to implement multi-objective workflow scheduling, i.e., optimizing makespan, cost, and energy consumption. However, all the above algorithms neglect the FT problem that has a direct impact on the quality of service (QoS) of the workflow execution.

Resource failure is unpredictable, and many FT methods have been developed due to their importance for time-critical applications. Resubmission and replication are two basic FT mechanisms in the distributed computing environment [14]. Replication submits multiple task copies to different processing units simultaneously to guarantee that at least one task copy executes successfully. This mechanism can obtain the minimum task completion time. For example, Plankensteiner and Prodan [14] propose a heuristic method for scheduling scientific workflows to tolerate resource failures in the grid. However, this scheduling method cannot be applied in the cloud environment since the resource paradigm of the cloud is distinct from that of the grid. Pandey et al. [30] use the task replication method to enhance the robustness of mobile device clouds when orchestrating mobile workflow applications. However, it considers the failure of the mobile environment that is different from the cloud resource failures. Marahatta et al. [31] propose an energy-aware fault-tolerant dynamic scheduling scheme, called EFDTs, to improve resource utilization and reduce energy consumption for the cloud data center. However, EFDTs is devised for parallel task scheduling and cannot be applied to schedule workflow applications. To make the best use of idle time and surplus budget, Calheiros and Buyya [32] introduce an enhanced IC-PCP with a replication (EIPR) algorithm to improve the likelihood of workflow execution. Nevertheless, EIPR considers only VMTF and cannot tackle other cloud resource failures.

The PB model, using only the primary copy and backup copy, has gained its popularity. Zhu et al. [13] present a QoS-aware fault-tolerant (QAFT) strategy to tolerate the permanent failure of the heterogeneous cluster. Zheng and Veeravalli [15] design two FT scheduling methods to prevent the impact of processor failure and communication delay in the grid. Nevertheless, the above methods cannot be directly adopted in the cloud environment as the cloud has different resource characteristics from the cluster and grid. Considering the performance volatility of cloud resources, Yan et al. [7] propose a PB-based dynamic fault-tolerant elastic scheduling (DEFT) algorithm for real-time tasks. However, DEFT is devised for parallel tasks, not for workflow tasks. Ding et al. [33] propose a fault-tolerant elastic scheduling algorithm, called FTESW, for scientific workflows in the cloud. The PB model is used in FTESW to achieve both the fault tolerance of scientific workflows and the high resource utilization of the cloud system. However, FTESW considers only the host failure, neglecting the other resource failures in the cloud. Zhu et al. [6] construct a real-time FT scheduling framework for scientific workflow by incorporating the traditional PB model and cloud features. However, this FT framework takes only the host failure into consideration. Fan et al. [34] develop a PB-based dynamic fault-tolerant strategy for workflow scheduling with the deadline constraint, where a Petri nets-based technique is applied to analyze and validate the feasibility of the proposed FT method. However, it focuses only on the VM failure, neglecting the host failure.

The resubmission method resubmits or re-executes the failed tasks on the same or another processing unit. Olteanu et al. [35] design a FT algorithm with resubmission and exception handling in large-scale distributed systems. However, this FT algorithm considers the task level failure, not resource level failure, and it is orthogonal to our FT scheduling approach. Chen et al. [36] introduce an efficient resubmission heuristic method to address multi-workflow scheduling problems by task rearrangement and rescheduling. However, this method is used for service delays caused by resource contention and performance prediction instead of resource failure. Yao et al. [11] present a FT scheduling algorithm, called ICFWS, with resubmission and replication methods for cloud scientific workflow. However, ICFWS does not consider the host failure aspects. Wu et al. [37] propose a spatial and temporal re-execution-based dynamic FT workflow scheduling scheme, DFTWS, to cope with resource failures. However, the transient failure of all the hosts in the cloud is ignored in DFTWS.

The checkpoint rollback recovery is another form of resubmission by saving the task execution state at different stages. When a failure occurs, it resumes the failed task from the latest checkpointing state. Bougeret et al. [38] provide a checkpointing-based FT scheduling strategy for parallel applications to minimize the job execution time in large-scale and failure-prone computing systems. Wang et al. [39] devise a temporal checkpointing selection method for business and scientific workflow to verify the temporal property. Aupy et al. [40] use the rollback and recovery mechanism to optimize the workflow completion time. Setlur et al. [41] introduce a FT workflow scheduling approach using heuristic replication and checkpointing mechanisms. However, this approach takes the task-level or workflow level failure into account, not the resource-level. Considering VM coordination for executing the parallel application in the cloud, Liu et al. [42] propose a proactively coordinated fault tolerance (PCFT) algorithm to tolerate the physical machine failure in the cloud, reducing network resource usage and energy consumption. However, PCFT is only suitable for parallel applications, not for workflows. In a word, the checkpointing mechanism is indeed able to address the transient failure, but it is not amenable to permanent resource failure, such as HPF.

The VM faults should be concerned as well. Mondal and Muppala [43] apply the checkpointing for scientific workflows to eliminate the impact of VMTF. Sobhanayak et al. [12] propose a VMTF analysis model to achieve load balancing and high resource utilization. Due to the threat of deliberate security intrusions and malicious attacks, Zheng et al. [44] present a queuing theory analysis for quantifying a security attribute in the VM-based FT architecture. Das et al. [45] introduce a hierarchical model by considering the performance degradation of web applications to cope with the VM failures. However, these FT algorithms consider only the VMTF, irrespective of other resource failures.

There also exist several rescheduling efforts for workflow scheduling in grid and cloud [21,22,46–48]. In [21], a runtime rescheduling approach is proposed for dynamic service delay and failure by recomputing the optimal scheduling schemes. In

[46], rescheduling is applied to increase the job completion reliability when the resource performance fluctuates. In [47], a workflow rescheduling is proposed to merge the processors as much as possible to realize energy efficiency. In [48], a delay minimization algorithm for scientific workflow execution in the cloud is introduced, which uses the rescheduling to optimize the execution cost and time. Although rescheduling has been used for improving reliability and reducing cost and time, this work is the first attempt to apply the core idea of rescheduling to cope with the HTF.

The main distinctions between the above existing works and our work are two-fold: 1) the existing approaches consider only one or two failure types at most. For example, the works in [6,7,33] consider only the HPF, the works in [11,32] take only the VMTF into account, and the work in [10] focuses on the HPF and VMTF, 2) as far as we know, none of the existing works concentrates on the HTF of cloud resources. In order to overcome the VMTF, HPF, and HTF simultaneously, this paper brings RDE, CED, and ReSC together to form an effective FT scheduling mechanism to guarantee the reliability of scientific workflow execution. Moreover, we devise a RA strategy, including RS-Up and RS-Down, to adjust resource demands and improve resource utilization dynamically.

3. Architecture and models

This section first describes the FT scheduling architecture. Then, some models are presented. The basic notations used in this paper are given in [Table 1](#).

3.1. Scheduling architecture

The FT scheduling architecture is depicted in [Fig. 1](#). The scheduler mainly performs deadline analysis, RDE, CDE, ReSC, and RA. Moreover, it is also responsible for monitoring resource status and the running status of hosts and VMs. It is worth pointing out that the status information is the small data; hence, the communication delay is neglected [7].

When a batch of scientific workflows submitted by cloud users arrives at the cloud, all of them first enter the workflow queue and will be processed by the first-come-first-service (FCFS) policy. In the deadline analysis, the workflow structure, parameters, and QoS requirements are analyzed. If the deadline can be satisfied, the sub-makespan of each task is computed. Otherwise, this workflow will be rejected. RDE and CDE are then used to compute each task's theoretical resource requirement according to the assigned sub-makespan. When the cloud system's workload is light, some hosts will be scaled down to improve resource utilization. While the workload is heavy, the scheduler scales up new hosts and creates new VMs to meet resource demands. This is the RA strategy that dynamically provides computing resources based on the resource status information. According to the delay execution mechanism, if one of the primary copies of a task finishes successfully, the VM resources occupied by the backup copies will be reclaimed. Moreover, if the HTF happens, ReSC is triggered to reschedule all the unfinished and unexecuted workflows.

3.2. Workflow model

A workflow is defined as $WF = (\mathcal{T}, \mathcal{E})$, where $\mathcal{T} = \{t_0, t_1, \dots, t_i, \dots, t_{n-1}\}$ is the task set, and $\mathcal{E} = \{(t_i, t_j) | t_i, t_j \in \mathcal{T}\}$ is the edge set that represents the dependency structure of all the tasks. Let $pre(t_i)$ and $suc(t_i)$ denote the predecessor set and the successor set of task t_i , respectively. A successor task can be executed if and only if all of its predecessors have been finished. If a task has no predecessor, we call it entry task t_{entry} ; on the contrary, if a task has no successor, we name it exit task t_{exit} . $D^{in}(t_i)$ and $D^{out}(t_i)$ are regarded as the size of input data and output data, respectively. Each task has a workload $W(t_i)$, generally, which consists of sequential workload $W^s(t_i)$ and parallel workload $W^p(t_i)$. Note that $W^s(t_i)$ and $W^p(t_i)$ can be processed only by the single-core CPU and by the multi-core CPU, respectively. Let η denote the parallel workload ratio. Then, $W^p(t_i) = \eta W(t_i)$, and $W^s(t_i) = (1 - \eta)W(t_i)$.

Table 1
Major notations.

Symbols	Semantics
t_i	The task t_i of workflow
n	The number of workflow tasks
$VM(k)$	The type of VM k in the cloud
$C(k)$	The processing capacity of $VM(k)$
$T(t_i, VM(k))$	The service time of t_i on $VM(k)$
$T_{ex}(t_i, VM(k))$	The execution time of task t_i on $VM(k)$
$T_{rx}(t_i)$	The data receiving time of task t_i
$T_{tx}(t_i)$	The data transmission time of task t_i
T_{MS}	The makespan of workflow
$T_{subMS}(t_i)$	The sub-makespan of task t_i
T_{DL}	The deadline constraint of workflow
λ_{WF}	The arrival rate of scientific workflows
λ_{VM}	The arrival rate of VMTF
λ_H	The arrival rate of HTF

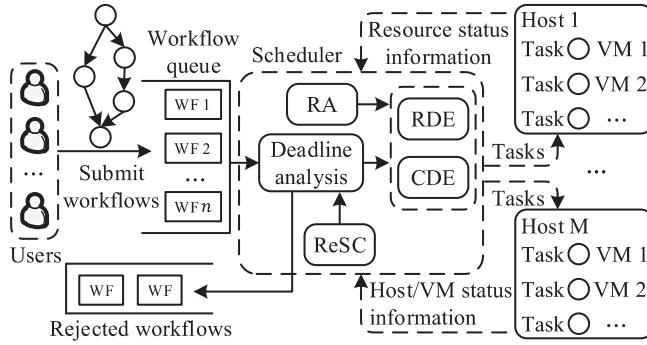


Fig. 1. The FT scheduling architecture.

3.3. Cloud model

A commercial cloud usually possesses thousands of physical hosts, and we assume that the scale of cloud computing resources is infinite [6,25]. Let $H = \{H(1), \dots, H(m), \dots, H(M)\}$ define the set of host types in cloud. Actually, a host can be abstracted into several heterogeneous VMs by virtualization technology. Thus, VM is the fundamental processing unit in the cloud. Let $VM = \{VM(1), \dots, VM(k), \dots, VM(K)\}$ represent the set of VM types. For example, as shown in Table 2, Amazon EC2 offers six types of m4 series VM instances [49]. Specifically, the $VM(k)$ has the computation capacity $C(k)$ and the price per hour. However, as stated in Section 3.2, only the parallel workload $W^p(t_i)$ can be processed by a multi-core CPU. Hence, the execution time of task t_i on $VM(k)$ can be expressed as

$$T_{ex}(t_i, VM(k)) = W^s(t_i)/C + W^p(t_i)/C(k) \quad (1)$$

where C represents the computing capacity of the single-core CPU.

3.4. Fault model

Let λ_H denote the arrival rate of HTF, which follows the Poisson distribution. Then, the reliable probability follows the Exponential distribution, which is represented by

$$P_H(T) = \exp(-\lambda_H T) \quad (2)$$

where T is the task service time.

Moreover, the cloud will suffer from HPF. Let λ'_H be the failure arrival rate, and then the reliability is measured by

$$P'_H(T) = \exp(-\lambda'_H T) \quad (3)$$

The minimum MTTF (mean time to failure) of the cloud hosts is always greater than or equal to the maximum service time of all the tasks [6,11,13]. Hence, the probability that two hosts happen HPF simultaneously is close to zero, i.e., $(1 - P'_H(T))^N \rightarrow 0, N \geq 2$. This explains the reason why the PB model can be used for FT scheduling.

Similar to the host reliability, the VM reliability is also modeled by the Exponential distribution.

$$P_{VM}(T) = \exp(-\lambda_{VM} T) \quad (4)$$

where λ_{VM} means the arrival rate of VMTF.

The features of fault tolerance are summarized as follows [6,10,13].

- HPF and HTF both can cause VM and task failures, i.e., all the cloud services are lost.
- Host failures are independent and follow the Poisson distribution. The repair process is initiated immediately when HTF happens. However, HPF is viewed as unrecoverable in a short time.

Table 2
The m4 series of VMs in Amazon EC2.

Number: k	VM Type: $VM(k)$	CPU: $C(k)$	Cost per Hour (\$)
1	m4.large	2	0.1
2	m4.xlarge	4	0.2
3	m4.2xlarge	8	0.4
4	m4.4xlarge	16	0.8
5	m4.10xlarge	40	2.0
6	m4.16xlarge	64	3.2

- VMTF is independent and obeys the Poisson distribution as well. Moreover, all the VMs can be recovered in a short time when VMTF occurs.
- All the VM and host failures can be detected immediately by the available fail-signal test.

4. Fault-tolerant scheduling

This section first presents the RDE, CDE, and ReSC mechanisms, respectively. Then, the RA strategy is introduced, which includes the RS-Up and RS-Down. Finally, the ReadyFS algorithm is summarized in terms of the above FT scheduling mechanisms and RA strategy. Regardless of any failure, the computing resources required by each task and the sub-makespan of each task are computed through the following steps.

Let $T_{start}(t_i)$ and $T_{end}(t_i)$ denote the start time and the end time of task t_i , respectively, and $T_{start}(t_i)$ is calculated by

$$T_{start}(t_i) = \max_{j \in pre(t_i)} \{T_{end}(t_j)\} \quad (5)$$

If $t_i = t_{entry}$, then $T_{start}(t_i) = 0$. The task t_i cannot start only when receiving all the predecessors' output data. Thus, the data receiving time is given by

$$T_{rx}(t_i) = D^{in}(t_i)/B \quad (6)$$

where $D^{in}(t_i) = \sum_{j \in pre(t_i)} D^{out}(t_j)$, and B is the network bandwidth between VMs. After that, the execution time of task t_i mapped on $VM(k)$ is $T_{ex}(t_i, VM(k))$, which has been given in Eq. (1). The transmission time, i.e., sending output data to the successors, is computed by

$$T_{tx}(t_i) = |suc(t_i)|D^{out}(t_i)/B \quad (7)$$

where $|suc(t_i)|$ is the number of the successors of task t_i . Then, the service time of task t_i , including data receiving time, task execution time, and data transmission time, is given by

$$T(t_i, VM(k)) = T_{rx}(t_i) + T_{ex}(t_i, VM(k)) + T_{tx}(t_i) \quad (8)$$

Thus, the end time of task t_i is calculated by

$$T_{end}(t_i) = T_{start}(t_i) + T(t_i, VM(k)) \quad (9)$$

The makespan of a workflow is the end time of the exit task, i.e.,

$$T_{MS} = T_{end}(t_{exit}) \quad (10)$$

We define the sub-makespan of task t_i as $T_{subMS}(t_i)$ [11,25], representing the pre-assigned service time of each task. The sub-makespans of all the workflow tasks can be obtained by dividing the makespan of workflow into multiple time spans. Obviously, if every task can be completed within its individual sub-makespan, then the whole workflow will be finished within the deadline constraint.

The minimum service time of task t_i can be obtained from Eq. (8) by mapping this task to $VM(K)$. In this case, the minimum makespan T_{MS}^{min} of the workflow can be computed according to Eq. (10). Thus, the sub-makespan of task t_i is calculated by

$$T_{subMS}(t_i) = T(t_i, VM(K))T_{DL}/T_{MS}^{min} \quad (11)$$

where T_{DL} is the deadline constraint. Generally, the specified deadline T_{DL} must be not less than the minimum makespan T_{MS}^{min} , i.e., $T_{DL} \geq T_{MS}^{min}$.

4.1. Replication with delay execution (RDE)

This section presents the RDE to cope with VMTF and HPF. Let a small positive number ε denote the replication coefficient for RDE. Then, we have

$$(1 - P_{VM}(T))^{N_{rep}(t_i)} \leq \varepsilon \quad (12)$$

where $N_{rep}(t_i)$ is the number of the copies duplicated by task t_i . Also, the execution of a task may fail because of the HPF, and hence an extra copy is required. Then, the total number of the copies of task t_i is required as follows:

$$N(t_i) = N_{rep}(t_i) + 1 \quad (13)$$

Eqs. (12) and (13) can be explained as follows: $N_{rep}(t_i)$ copies of task t_i are used for the VMTF, and an extra copy is added to cope with HPF. Thus, the probability of at least one task copy executes successfully is $1 - \varepsilon$.

Unlike the traditional replication method that executes all task copies simultaneously, RDE divides $N(t_i)$ task copies into two groups: primary copies and backup copies. In this case, primary copies execute first, and then backup copies execute, which is called the delay execution. The number of primary copies of task t_i is represented by $N_p(t_i)$, i.e.,

$$N_p(t_i) = N(t_i)/2 \quad (14)$$

The number of backup copies is represented by $N_b(t_i)$, i.e.,

$$N_b(t_i) = N(t_i) - N_p(t_i) \quad (15)$$

Fig. 2 shows the two cases of RDE. Case 1: $T(t_i, VM(k)) \leq T_{subMS}(t_i) < 2T(t_i, VM(k))$, the execution of backup copies start during the execution process of primary copies and stop immediately if one of the primary copies executes successfully; Case 2: $T_{subMS}(t_i) \geq 2T(t_i, VM(k))$, the execution of backup copies start only when all the primary copies have failed.

Intuitively, RDE requires fewer resources than the traditional replication method. This is because we do not need to execute the backup copies entirely. In general, the number of VM resources occupied by one task copy is computed by the form “time $\times C(k)$ ”, i.e.,

$$R(t_i) = T(t_i, VM(k))C(k) \quad (16)$$

Thus, the number of resources required by the traditional replication method is

$$R_{TR}(t_i) = N(t_i)R(t_i) \quad (17)$$

Next, we compute the expected number of resources required by RDE and prove that the traditional replication method will consume more resources than RDE.

Case 1 has two scenarios:

(1.1) At least one of the primary copies executes successfully, the probability of which is given by

$$P^{11} = 1 - [1 - P_{VM}(T(t_i, VM(k)))]^{N_p(t_i)} \quad (18)$$

Hence, there is no need to execute backup copies, and the resource required in this case is calculated by

$$R_{RDE}^{11}(t_i) = N_p(t_i)T(t_i, VM(k))C(k) + N_b(t_i)[2T(t_i, VM(k)) - T_{subMS}(t_i))]C(k) \quad (19)$$

(1.2) If all the primary copies fail, the probability is

$$P^{12} = [1 - P_{VM}(T(t_i, VM(k)))]^{N_p(t_i)} \quad (20)$$

Then, backup copies need to be executed completely. The number of resources is computed by

$$R_{RDE}^{12}(t_i) = N(t_i)T(t_i, VM(k))C(k) \quad (21)$$

Therefore, the total resources required for fault tolerance is represented by

$$\mathbb{E}\{R_{RDE}^1(t_i)\} = P^{11}R_{RDE}^{11}(t_i) + P^{12}R_{RDE}^{12}(t_i) \quad (22)$$

Case 2 also has two scenarios:

(2.1) At least one of the primary copies executes successfully, and the corresponding probability is $P^{21} = P^{11}$. So, we do not need to execute backup copies, and the number of resources is calculated by

$$R_{RDE}^{21}(t_i) = N_p(t_i)T(t_i, VM(k))C(k) \quad (23)$$

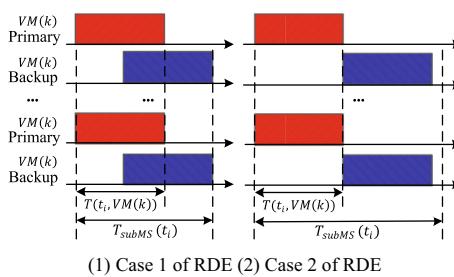


Fig. 2. Two cases of RDE.

(2.2) If all the primary copies fail, the probability is $P^{22} = P^{12}$. Then, backup copies are required to be executed completely. The number of computing resources required in this case is computed by

$$R_{RDE}^{12}(t_i) = N(t_i)T(t_i, VM(k))C(k) \quad (24)$$

Therefore, the total number of resources is represented as

$$\mathbb{E}\{R_{RDE}^2(t_i)\} = P^{21}R_{RDE}^{21}(t_i) + P^{22}R_{RDE}^{22}(t_i) \quad (25)$$

We give **Theorem 1** to prove that RDE outperforms the traditional replication method on resource consumption.

Theorem 1. *The total number of resources required by RDE is less than that of the traditional replication method, i.e.,*

$$R_{TR}(t_i) > \mathbb{E}\{R_{RDE}^1(t_i)\} \quad (26)$$

$$R_{TR}(t_i) > \mathbb{E}\{R_{RDE}^2(t_i)\} \quad (27)$$

Proof. According to Eq. (22) and using the fact that $P^{11} + P^{12} = 1$, we have

$$\begin{aligned} \mathbb{E}\{R_{RDE}^1(t_i)\} &= P^{11}R_{RDE}^{11}(t_i) + P^{12}R_{RDE}^{12}(t_i) \\ &= N(t_i)T(t_i, VM(k))C(k) + N_b(t_i)[T(t_i, VM(k)) - T_{subMS}(t_i)]P^{11}C(k) \end{aligned} \quad (28)$$

We know that $T(t_i, VM(k)) < T_{subMS}(t_i)$, and $P^{11} > 0$, and hence we get

$$N_b(t_i)[T(t_i, VM(k)) - T_{subMS}(t_i)]P^{11}C(k) < 0 \quad (29)$$

Then, we get

$$\mathbb{E}\{R_{RDE}^1(t_i)\} < N(t_i)T(t_i, VM(k))C(k) = R_{TR}(t_i) \quad (30)$$

This proves Eq. (26). Using the same analysis process can prove Eq. (27).

The pseudocode of RDE is outlined in Algorithm 1. Let $VM_{RDE}^{opt}(k)$ and $R_{RDE}^{opt}(t_i)$ denote the optimal VM type and the optimal number of resources for task t_i , respectively (lines 1–2). Then, we go through all the VM types and select some of them that meet the sub-makespan constraint (lines 3–5). If Case 1 is satisfied, the number of resources $\mathbb{E}\{R_{RDE}^1(t_i)\}$ is calculated (lines 6–8); or $\mathbb{E}\{R_{RDE}^2(t_i)\}$ is calculated (lines 9–11). Finally, $R_{RDE}^{opt}(t_i)$ and $VM_{RDE}^{opt}(k)$ are updated (lines 12–14). The time complexity of RDE is $O(K)$, which mainly depends on the number of VM types.

Algorithm 1. Replication with Delay Execution (RDE)

1. $VM_{RDE}^{opt}(k) \leftarrow 0$;
 2. $R_{RDE}^{opt}(t_i) \leftarrow \infty$;
 3. **foreach** $VM(k) \in VM$ **do**
 4. **if** $T_{subMS}(t_i) \geq T(t_i, VM(k))$ **then**
 5. Calculate $N(t_i)$;
 6. **if** Case 1 **then**
 7. Calculate $\mathbb{E}\{R_{RDE}^1(t_i)\}$;
 8. $R(t_i) \leftarrow \mathbb{E}\{R_{RDE}^1(t_i)\}$;
 9. **else if** Case 2 **then**
 10. Calculate $\mathbb{E}\{R_{RDE}^2(t_i)\}$;
 11. $R(t_i) \leftarrow \mathbb{E}\{R_{RDE}^2(t_i)\}$;
 12. **if** $R(t_i) < R_{RDE}^{opt}(t_i)$ **then**
 13. $R_{RDE}^{opt}(t_i) \leftarrow R(t_i)$;
 14. $VM_{RDE}^{opt}(k) \leftarrow VM(k)$;
-

4.2. Checkpointing with delay execution (CDE)

Checkpointing is the technique that stores task execution information on the disk periodically. This section introduces the CDE for coping with VMTF and HPF. The implementation process is as follows: we split a task into several chunks, and the status of each chunk is preserved after executing successfully. If VMTF happens during the chunk execution process, this chunk needs to be re-executed.

The CDE also has two cases, which are shown in Fig. 3. Case 1: $T_{\mathbb{E}}(t_i, VM(k)) \leq T_{subMS}(t_i) < 2T_{\mathbb{E}}(t_i, VM(k))$; Case 2: $T_{subMS}(t_i) \geq 2T_{\mathbb{E}}(t_i, VM(k))$, where $T_{\mathbb{E}}(t_i, VM(k))$ is the expected service time of task t_i by using checkpointing approach. Similar to the well-known PB model, CDE needs only two task copies. Note that both the primary copy and backup copy can tackle the VMTF, and the backup copy is also used to cope with the HPF.

Let $T(W)$ and $T(W_i)$ denote the execution time of task t_i and the chunk execution time of task t_i , respectively. Hence, we have

$$T(W) = \sum_{l=1}^L T(W_l) = T_{ex}(t_i, VM(k)) \quad (31)$$

where W and W_l represent the task workload and chunk workload, respectively. Let $T(W|\tau)$ represent the random variable that measures the time required for the successful execution of workload W , and let W_1 denote the first chunk. Then, the recursion equation can be written by

$$T(W|\tau) = \begin{cases} T(W_1) + T_{chk} + T(W - W_1|\tau + T(W_1) + T_{chk}), P_{chk} \\ E\{T(W_1) + T_{chk}\} + T_{wst}^{VM} + T(W|T_{wst}^{VM}), 1 - P_{chk} \end{cases} \quad (32)$$

where $P_{chk} = P_{VM}(T(W_1) + T_{chk})$, and T_{chk} denotes the time overhead to perform a checkpointing. $T_{wst}^{VM} = T_{dn}^{VM} + T_{rec}^{VM}$ indicates the time wasted by VM failure, including VM downtime T_{dn}^{VM} and recovery time T_{rec}^{VM} . The explanation of Eq. (32) is that: if the VM normally runs during the time $T(W_1) + T_{chk}$ with probability P_{chk} , there remains to execute a workload of size $W - W_1$, and then the start time of the next chunk is $\tau + T(W_1) + T_{chk}$; 2) If the VM fails during the execution process of the first chunk with probability $1 - P_{chk}$, then the extra time $E\{T(W_1) + T_{chk}\}$ is produced and there still remains W units of workload to be processed, and hence the start time of current chunk is T_{wst}^{VM} . Next, we analyze the task execution time under the assumption that all the resource failure probabilities follow the Arbitrary distribution and Exponential distribution, respectively.

4.2.1. Execution time on arbitrary distribution

The goal is to minimize the theoretical execution time $E\{T(W|\tau)\}$ by setting the optimal number of task chunks. In this case, the execution time minimization problem is equivalent to finding a solution to minimize the following formula:

$$\min E\{T(W|\tau)\} = P_{chk}(T(W_1) + T_{chk} + E\{T(W - W_1|\tau + T(W_1) + T_{chk})\}) + (1 - P_{chk})(E\{T(W_1) + T_{chk}\} + T_{wst}^{VM} + E\{T(W|T_{wst}^{VM})\}) \quad (33)$$

where the probabilities of all the resource failures follow the arbitrary distributions (e.g., Exponential, Gamma, Weibull, or Lognormal distributions). Solving $\min E\{T(W|\tau)\}$ for an arbitrary distribution is difficult because there is no specific distribution function. However, we know that the optimal solution is to find the number of chunks and the size of each chunk. In other words, to minimize $E\{T(W|\tau)\}$, we need to determine how many chunks are associated with each task and the size of each chunk.

We propose a dynamic programming (DP) based approximation algorithm, called DPminExTime, to minimize $E\{T(W|\tau)\}$ [38]. The pseudocode of DPminExTime is shown in Algorithm 2. Let w denote the unit workload, meaning that each chunk

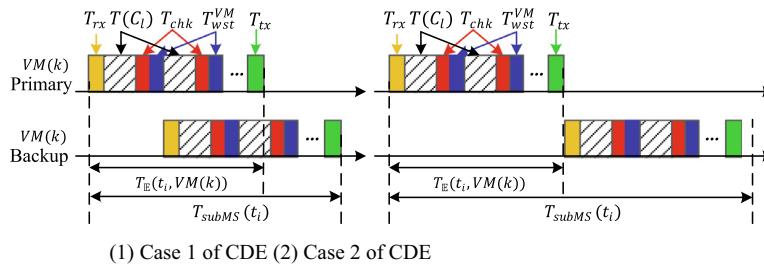


Fig. 3. Two cases of CDE.

size W_l is the integer multiples of w . The execution time for this unit workload w is represented by T_w . To minimize $\mathbb{E}\{T(W|\tau)\}$, we define a function $DPminExTime(x, \tau)$, where x is initialized to W/w . This function can be viewed as the remaining number of unit workload needed to be processed (line 1). We also define two symbols $chunkSize$ and T_{ex} , denoting the chunk size and task execution time, respectively (line 2). We use the ergodic search method to find the optimal chunk size (lines 3–11). We can see from line 4 that if the task chunk with the size kw is executed successfully, the remaining workload is $(x - k)w$, and the time since the last failure is $\tau + kT_w + T_{chk}$. Thus, we can compute $\mathbb{E}\{T(xw - kw|\tau + kT_w + T_{chk})\}$ (lines 4–5). Otherwise, we can obtain $\mathbb{E}\{T(xw|T_{wst}^{\text{VM}})\}$, where the remaining workload is still xw , and the next failure time starts from T_{wst}^{VM} (lines 6–7). Based on the above analysis, the current execution time T can be derived (line 8), which is used to compare with T_{ex} and record the optimal execution time and chunk size (9–11). The time complexity of $DPminExTime$ largely depends on the granularity of unit workload, i.e., a smaller w can get a more accurate solution but incurs higher time complexity.

$DPminExTime$ is the general algorithm for the arbitrary distributions to find the minimum $\mathbb{E}\{T(W|\tau)\}$. In the next section, we assume the resource failure probabilities follow the Exponential distribution, where the numerical solutions can be obtained by directly deducing.

Algorithm 2. Minimize Execution Time by DP ($DPminExTime$)

1. Define the function $DPminExTime(x, \tau)$;
 2. $chunkSize \leftarrow 0$, $T_{ex} \leftarrow \infty$;
 3. **for** $k = 1$ to x **do**
 4. **if** chunk kw is successful **then**
 5. $\mathbb{E}\{T(xw - kw|\tau + kT_w + T_{chk})\} \leftarrow DPminExTime(x - k, \tau + kT_w + T_{chk})$;
 6. **else**
 7. $\mathbb{E}\{T(xw|T_{wst}^{\text{VM}})\} \leftarrow DPminExTime(x, T_{wst}^{\text{VM}})$;
 8. $T \leftarrow \mathbb{E}\{T(W|\tau)\}$;
 9. **if** $T < T_{ex}$ **then**
 10. $T_{ex} \leftarrow T$;
 11. $chunkSize \leftarrow kw$;
-

4.2.2. Execution time on Exponential distribution

One property of the Exponential distribution is “memoryless”. So, we can write $T(W - W_1)$ instead of $T(W - W_1|\tau + T(W_1) + T_{chk})$ and $T(W)$ instead of $T(W|\tau)$ and $T(W|T_{wst}^{\text{VM}})$. According to Eq. (33), the expected execution time is computed by

$$\mathbb{E}\{T(W)\} = P_{chk}(T(W_1) + T_{chk} + \mathbb{E}\{T(W - W_1)\}) + (1 - P_{chk})(\mathbb{E}\{T(W_1) + T_{chk}\} + T_{wst}^{\text{VM}} + \mathbb{E}\{T(W)\}) \quad (34)$$

Rearranging terms yields the following equation:

$$\mathbb{E}\{T(W)\} = \mathbb{E}\{T(W - W_1)\} + T(W_1) + T_{chk} + (\mathbb{E}\{T(W_1) + T_{chk}\} + T_{wst}^{\text{VM}}) \frac{1 - P_{VM}(W_1 + T_{chk})}{P_{VM}(W_1 + T_{chk})} \quad (35)$$

where $P_{VM}(T(W_1) + T_{chk}) = \exp(-\lambda_{VM}(T(W_1) + T_{chk}))$. According to Lemma 1 in [38], we get

$$\mathbb{E}\{T(W_1) + T_{chk}\} = \frac{1}{\lambda_{VM}} - \frac{T(W_1) + T_{chk}}{\exp(\lambda_{VM}(T(W_1) + T_{chk})) - 1} \quad (36)$$

Then, we have

$$\mathbb{E}\{T(W)\} = \mathbb{E}\{T(W - W_1)\} + \left(\frac{1}{\lambda_{VM}} + T_{wst}^{\text{VM}}\right)(\exp(\lambda_{VM}(T(W_1) + T_{chk})) - 1) \quad (37)$$

Summing the above over all the chunks yields

$$\sum_{l=1}^L \mathbb{E}\{T(W)\} = \sum_{l=1}^L \mathbb{E}\{T(W - W_l)\} + \sum_{l=1}^L \left[\left(\frac{1}{\lambda_{VM}} + T_{wst}^{\text{VM}}\right)(\exp(\lambda_{VM}(T(W_l) + T_{chk})) - 1)\right] \quad (38)$$

where $\sum_{l=1}^L \mathbb{E}\{T(W - W_l)\} = (L - 1)\mathbb{E}\{T(W)\}$, and plunge it into Eq. (38) yields

$$\mathbb{E}\{T(W)\} = \sum_{l=1}^L \left[\left(\frac{1}{\lambda_{VM}} + T_{wst}^{\text{VM}}\right)(\exp(\lambda_{VM}(T(W_l) + T_{chk})) - 1)\right] \quad (39)$$

Define $f(z) = \exp(\lambda_{VM}z)$, where $z = T(W_1) + T_{chk}$. As $0 < W_l \leq W$ and $T_{chk} > 0$, and hence $z > 0$. Then, we have the first derivative and second derivative of $f(z)$.

$$f'(z) = \lambda_{VM} \exp(\lambda_{VM}z) \quad (40)$$

$$f''(z) = \lambda_{VM}^2 \exp(\lambda_{VM}z) \quad (41)$$

As $\lambda_{VM} > 0$, the second derivative is nonnegative, i.e., $f''(z) > 0$. So, $f(z)$ is a convex function, and then we have $\mathbb{E}\{f(z)\} \geq f\{\mathbb{E}(z)\}$. Applying this inequality yields

$$\begin{aligned} \mathbb{E}\{T(W)\} &= \sum_{l=1}^n \left[\left(\frac{1}{\lambda_{VM}} + T_{wst}^{VM} \right) (\exp(\lambda_{VM}(T(W_l) + T_{chk})) - 1) \right] \\ &\geq L \left(\frac{1}{\lambda_{VM}} + T_{wst}^{VM} \right) \left(\exp\left(\lambda_{VM} \frac{\sum_{l=1}^L (T(W_l) + T_{chk})}{L}\right) - 1 \right) \end{aligned} \quad (42)$$

We know that $T(W) = \sum_{l=1}^L T(W_l)$. In order to minimize $\mathbb{E}\{T(W)\}$, all the chunks should have the same size, i.e., $T(W_l) = T(W)/L$. Then, we get

$$\mathbb{E}\{T(W)\} = L \left(\frac{1}{\lambda_{VM}} + T_{wst}^{VM} \right) \left(\exp\left(\lambda_{VM} \frac{T(W)}{L} + T_{chk}\right) - 1 \right) \quad (43)$$

Next, we find the value of L to minimize $\mathbb{E}\{T(W)\}$. We need to solve the following equation

$$\mathbb{E}\{T(W)\}' = \exp\left(\lambda_{VM} \left(\frac{T(W)}{L} + T_{chk} \right)\right) \left(1 - \frac{\lambda_{VM} T(W)}{L} \right) = 0 \quad (44)$$

According to [Theorem 1](#) in [38], we have

$$L = \frac{\lambda_{VM} T(W)}{1 + \mathbb{L}(-\exp(-\lambda_{VM} T_{chk} - 1))} \quad (45)$$

where \mathbb{L} is the Lambert function, i.e., $\mathbb{L}(z) \exp(\mathbb{L}(z)) = z$. Moreover, the optimal value is obtained by one of the two integers surrounding the L , and $L \geq 1$. Hence, we have

$$L^* = \max\{1, L\} \text{ or } \lceil L \rceil \quad (46)$$

Thus, the expected service time $T_{\mathbb{E}}(t_i, VM(k))$ of task t_i is given as below.

$$T_{\mathbb{E}}(t_i, VM(k)) = \mathbb{E}\{T(W)\} + T_{rx}(t_i) + T_{tx}(t_i) \quad (47)$$

4.2.3. CDE on Exponential distribution

Here we discuss the number of resources required by CDE under the Exponential distribution assumption, which can be calculated as follows:

Case 1 has two scenarios:

(1.1) If the primary copy of the task t_i executes successfully, the probability is given by

$$P_H^{11} = P'_H(T_{\mathbb{E}}(t_i, VM(k))) = \exp(-\lambda'_H T_{\mathbb{E}}(t_i, VM(k))) \quad (48)$$

Yielding the resource requirement, i.e.,

$$R_{CDE}^{11}(t_i) = T_{\mathbb{E}}(t_i, VM(k))C(k) + (2T_{\mathbb{E}}(t_i, VM(k)) - T_{subMS}(t_i))C(k) \quad (49)$$

(1.2) If the primary copy fails with probability $P_H^{12} = 1 - P_H^{11}$, the backup copy should be executed completely.

$$R_{CDE}^{12}(t_i) = 2T_{\mathbb{E}}(t_i, VM(k))C(k) \quad (50)$$

The number of resources consumed by CDE in Case 1 is represented by

$$\mathbb{E}\{R_{CDE}^1(t_i)\} = P_H^{11} R_{CDE}^{11}(t_i) + P_H^{12} R_{CDE}^{12}(t_i) \quad (51)$$

Case 2 has two scenarios as well:

- (1) Similarly, if the primary copy of the task t_i executes successfully, the probability is $P_H^{21} = P_H^{11}$. the number of resources is

$$R_{CDE}^{21}(t_i) = T_{\mathbb{E}}(t_i, \text{VM}(k))C(k) \quad (52)$$

- (2) If the primary copy fails with probability $P_H^{22} = 1 - P_H^{21}$, the backup copy should be executed completely. In this case, the number of resources is $R_{CDE}^{22}(t_i) = R_{CDE}^{12}(t_i)$.

The number of resources consumed by CDE in Case 2 is expressed as

$$\mathbb{E}\{R_{CDE}^2(t_i)\} = P_H^{21}R_{CDE}^{21}(t_i) + P_H^{22}R_{CDE}^{22}(t_i) \quad (53)$$

Algorithm 3. Checkpointing with Delay Execution (CDE)

```

1.  $\text{VM}_{CDE}^{opt}(k) \leftarrow 0$ 
2.  $R_{CDE}^{opt}(t_i) \leftarrow \infty$ 
3. foreach  $\text{VM}(k) \in \text{VM}$  do
4.   Calculate  $T_{\mathbb{E}}(t_i, \text{VM}(k))$ ;
5.   if  $T_{subMS}(t_i) \geq T_{\mathbb{E}}(t_i, \text{VM}(k))$  then
6.     if Case 1 then
7.       Calculate  $\mathbb{E}\{R_{CDE}^1(t_i)\}$ ;
8.        $R(t_i) \leftarrow \mathbb{E}\{R_{CDE}^1(t_i)\}$ ;
9.     else if Case 2 then
10.    Calculate  $\mathbb{E}\{R_{CDE}^2(t_i)\}$ ;
11.     $R(t_i) \leftarrow \mathbb{E}\{R_{CDE}^2(t_i)\}$ ;
12.    if  $R(t_i) < R_{CDE}^{opt}(t_i)$  then
13.       $R_{CDE}^{opt}(t_i) \leftarrow R(t_i)$ ;
14.       $\text{VM}_{CDE}^{opt}(k) \leftarrow \text{VM}(k)$ ;

```

We can also prove that CDE needs less resources than the traditional replication method. Similar to [Theorem 1](#), the proof process is omitted in this section. The pseudocode of CDE is shown in Algorithm 3. Let $\text{VM}_{CDE}^{opt}(k)$ and $R_{CDE}^{opt}(t_i)$ denote the optimal VM type and the optimal number of resources of task t_i , respectively (lines 1–2). Then, we go through all the VM types and select some of them that meet the sub-makespan constraint (lines 3–5). If Case 1 is satisfied, the number of resources $\mathbb{E}\{R_{CDE}^1(t_i)\}$ is calculated (lines 6–8); or $\mathbb{E}\{R_{CDE}^2(t_i)\}$ is obtained (lines 9–11). Finally, $R_{CDE}^{opt}(t_i)$ and $\text{VM}_{CDE}^{opt}(k)$ are updated (lines 12–14). The time complexity of CDE is $O(K)$, depending on the number of VM types.

4.3. Rescheduling (ReSC) for HTF

All the task execution and cloud services will be interrupted by HTF. Thus, re-executing the unfinished and unexecuted tasks by the original scheduling scheme may fail to satisfy the deadline constraint. Therefore, rescheduling is essentially deployed for this service interruption.

[Fig. 4](#) depicts the ReSC process applied to a simple workflow WF . When HTF happens, the cloud needs to take a certain amount of wasted time, $T_{wst}^H + T_{wst}^{\text{VM}}$, where T_{wst}^H is the host waste time, consisting of host downtime T_{dn}^H and recovery time T_{rec}^H . We can recalculate the residual deadline $T_{DL}^{\text{res}} = T_{DL} - T_{wst}^H - T_{wst}^{\text{VM}} - T_{DL}^{\text{used}}$, where T_{DL}^{used} is the already used time for workflow execution. Then, for this unfinished workflow, a dummy task t_0 is connected to the unfinished and executable tasks, forming a new workflow WF' . The following process is equivalent to the cloud accepts a workflow WF' with deadline constraint T_{DL}^{res} and schedule this workflow with fault tolerance guarantee.

The pseudocode of ReSC is given in Algorithm 4. First, the original scheduling scheme for all the workflows is cleared up (line 1). Then, we construct a new workflow WF' for each uncompleted workflow and calculate the corresponding residual deadline T_{DL}^{res} (lines 2–4). However, if the deadline is larger than its minimal execution time T_{MS}^{\min} , i.e., $T_{DL}^{\text{res}} < T_{MS}^{\min}$, WF' will be

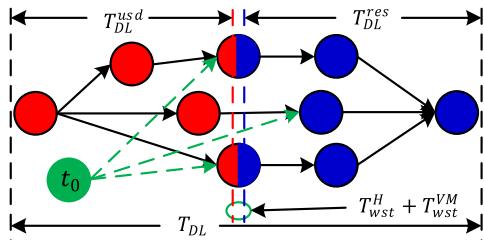


Fig. 4. The ReSC process.

rejected (lines 5–6). Otherwise, T_{DL}^{res} is divided into multiple sub-makespans (line 7). Next, RDE and CDE are used to ensure the successful execution of the tasks and get the optimal VM type for each task (lines 8–11). If the resources of the current active hosts and VMs are enough, we map all the task copies to the optimal VMs. Otherwise, RS-Up is applied to start hosts for resource demand. The RS-Up will be introduced in the next section. The worst time complexities of RDE, CDE, and RS-Up are $O(K)$, $O(K)$, and $O(|H_{act}| \log |H_{act}|)$, respectively. So, the worst time complexity of ReSC is $O(\sum_{WF'} n_{WF'} |H_{act}| \log |H_{act}|)$, where $|H_{act}|$ is the number of active hosts in cloud, and $n_{WF'}$ is the number of tasks in WF' .

Algorithm 4. Rescheduling (ReSC)

1. Clear up the original scheduling scheme;
 2. **foreach** WF in Cloud **do**
 3. Construct a new workflow WF' ;
 4. Calculate the residual deadline T_{DL}^{res} ;
 5. **if** $T_{DL}^{res} < T_{MS}^{min}$ **then**
 6. Reject WF' and Continue;
 7. Divide T_{DL}^{res} into sub-makespans;
 8. **foreach** t_i in WF' **do**
 9. $R_{RDE}^{opt}(t_i) \leftarrow$ Call RDE;
 10. $R_{CDE}^{opt}(t_i) \leftarrow$ Call CDE;
 11. $R^{opt}(t_i) \leftarrow \min\{R_{RDE}^{opt}(t_i), R_{CDE}^{opt}(t_i)\};$
 12. **if** VM resources are enough **then**
 13. Map copies of tasks to $VM^{opt}(k)$;
 14. **else**
 15. Call RS-Up;

4.4. Resource adjustment

The dynamic RA strategy is presented in this section to adjust computing resources and improve resource utilization, consisting of RS-Up and RS-Down.

4.4.1. Resource scaling up (RS-Up)

If primary copies or backup copies of a task cannot be mapped to the existing active VMs, the corresponding RS-Up strategy is applied by creating new VMs from current active hosts. If no such hosts can satisfy the resource requirement, a sleeping host will be started.

The pseudocode of RS-Up is shown in Algorithm 5. Let H_{act} represent the set of active hosts in the cloud, and let H_{ins} denote the host instance in H_{act} . All the hosts in H_{act} are ordered by the number of remaining CPUs, i.e., C^{res} (line 1). If $H_{ins}.C^{res} \geq C(k)$, a new VM is created on H_{ins} , and one task copy is mapped to this VM (lines 2–6). Otherwise, a new host should be started, and the new VM tries to be created on this new host (lines 7–11). Note that the cloud has multiple host types. For improving resource utilization, it is preferable to start a host with low performance. The time complexity of RS-Up is $O(|H_{act}| \log |H_{act}|)$, which mainly depends on sorting H_{act} .

Algorithm 5. Resource Scaling-Up (RS-Up)

```

1. Sort  $H_{act}$  in increasing order by residual CPUs;
2. foreach  $H_{ins}$  in  $H_{act}$  do
3.   if  $H_{ins}.C^{res} \geq C(k)$  then
4.     Create a new VM on  $H_{ins}$ ;
5.     Allocate the task copy to this new VM;
6.   Return true;
7. foreach  $H(m)$  in  $H$  do
8.   if  $H(m).C^{res} \geq C(k)$  then
9.     Create a new VM on  $H(m)$ ;
10.     $H_{act} \leftarrow H_{act} + \{H(m)\}$ ;
11.  Return true;
```

4.4.2. Resource scaling down (RS-Down)

The RS-Down is devised to reduce the idle VMs when the cloud workload is light. For example, when a VM instance is idle for a short time, the scheduler can decrease its CPU frequency by the DVFS technique [25]. However, if a VM will be idle for a relatively long time, it will be canceled. Moreover, if more than one primary copy executes successfully, the VM resources assigned for backup copies will be recycled. All the above methods, i.e., decreasing CPU frequency, removing idle VMs, and reclaiming VM resources, can improve cloud resource utilization.

Algorithm 6 gives the pseudocode of RS-Down. Let T_{th} denote the time threshold. When the idle time of a VM is less than T_{th} , its working frequency will be decreased to the minimum value (lines 1–3). Otherwise, this VM will be canceled (lines 4–5). Then, we check the resource utilization for current active hosts. If the utilization of a host is less than the self-defined threshold U_{th} , the scheduler tries to migrate its VMs to the other hosts (lines 6–11). When all the VMs of a host are migrated successfully, this host will be shut down (lines 12–14). The worst time complexity of RS-Down is $O(n_{VM(k)}n_{VM(j)}|H_{act}|)$, which depends on three-loop statements in line 1, line 7, and line 8, where $n_{VM(k)}$, $n_{VM(j)}$, and $|H_{act}|$ represent the number of VMs in the cloud, the number of VMs in host $H(m)$, and the number of active hosts in the cloud, respectively.

Algorithm 6. Resource Scaling Down (RS-Down)

```

1. foreach  $VM(k)$  in Cloud do
2.   if  $T_{idle}(VM(k)) < T_{th}$  then
3.     Decrease CPU frequency to the lowest level;
4.   else if  $T_{idle}(VM(k)) \geq T_{th}$  then
5.     Cancel  $VM(k)$  from its host  $H_{ins}$ ;
6.   if  $H_{ins}.U \leq U_{th}$  then
7.     foreach  $VM(j)$  in  $H_{ins}$  do
8.       foreach  $H'_{ins}$  in  $H_{act} - \{H_{ins} | H_{ins}.U \leq U_{th}\}$  do
9.         if  $H'_{ins}.C^{res} \geq C(j)$  then
10.           Migrate  $VM(j)$  to  $H'_{ins}$ ;
11.           Cancel  $VM(j)$  from  $H_{ins}$ ;
12.   if  $H_{ins}$  is idle then
13.     Shut down  $H_{ins}$ ;
14.    $H_{act} \leftarrow H_{act} - \{H_{ins}\}$ ;
```

4.5. ReadyFS algorithm

Algorithm 7 gives the pseudocode of ReadyFS. Let $WF(\tau)$ denote the set of scientific workflows that arrived at time slot τ , which follows the Poisson distribution with arrival rate λ_{WF} . These workflows will be scheduled based on the FCFS policy. For each workflow, if its deadline constraint cannot be met, it will be rejected (lines 1–3). Otherwise, the sub-makespan of each task is calculated (line 4). Then, the RDE and CDE are used to ensure the successful execution of the task (lines 5–8). If the resource demand can be met, all the task copies are mapped to the optimal VMs. Otherwise, RS-Up is applied to start new hosts (lines 9–12). Finally, if HTF happens, ReSC is applied to reschedule the unfinished workflows (lines 13–14). If VM resources are idle, RS-Down is applied to improve resource utilization (lines 15–16). Note that all the task copies are executed on different VMs and hosts.

As discussed above, the time complexities of RDE, CDE, ReSC, RS-Up, and RS-Down are $O(K)$, $O(K)$, $O(\sum_{WF'} n_{WF'} |H_{act}| \log |H_{act}|)$, $O(|H_{act}| \log |H_{act}|)$, and $O(n_{VM(k)}n_{VM(j)}|H_{act}|)$, respectively. Therefore, the worst time complexity of

Algorithm 7 from lines 1 to 12 is $O(\sum_{WF} n_{WF} |H_{act}| \log |H_{act}|)$, where n_{WF} is the number of tasks in WF , and we assume that $K < |H_{act}| \log |H_{act}|$. Let N_{WF}^{\max} and n_{WF}^{\max} represent the maximum number of workflows and the maximum number of tasks in $WF(\tau)$ and WF , respectively. Thus, the time complexity of ReadyFS is $\max\{O(N_{WF}^{\max} n_{WF}^{\max} |H_a| \log |H_a|), O(n_{VM(k)} n_{VM(j)} |H_{act}|)\}$.

Algorithm 7. ReadyFS Algorithm

1. **foreach** WF in $WF(\tau)$ **do**
 2. **if** $T_{DL} < T_{MS}^{\min}$ **then**
 3. Reject WF and Continue;
 4. Divide the deadline into sub-makespans;
 5. **foreach** t_i in WF **do**
 6. $R_{RDE}^{opt}(t_i) \leftarrow$ Call RDE;
 7. $R_{CDE}^{opt}(t_i) \leftarrow$ Call CDE;
 8. $R^{opt}(t_i) \leftarrow \min\{R_{RDE}^{opt}(t_i), R_{CDE}^{opt}(t_i)\}$;
 9. **if** VM resources are enough **then**
 10. Map copies of tasks to $VM^{opt}(k)$;
 11. **else**
 12. Call RS-Up;
 13. **if** HTF **then**
 14. Call ReSC;
 15. **if** Reclaim $VM(k)$ **then**
 16. Call RS-Down;
-

The performance comparisons of RDE and CDE are shown in Fig. 5. All of them can guarantee the successful execution of tasks in the presence of VMTF and HPF. The advantage of RDE is that it can reduce the task execution time but cause more resources. In contrast, CDE can enhance resource utilization but incur longer task execution time. In the real-world scenario, the cloud users submit scientific workflow applications with different structures, parameters, and deadline constraints. Hence, RDE and CDE should be combined to ensure fault tolerance and improve resource utilization. For example, if a workflow deadline is very tight, RDE will be selected because CDE may not meet the deadline constraint; while the deadline becomes loose, CDE can be adopted to save resources.

5. Performance evaluation

To demonstrate the feasibility and effectiveness of our ReadyFS, we simulate five vertical and three horizontal comparison algorithms to compare with ReadyFS on three performance metrics, namely workflow guarantee rate, resource consumption of tasks, and cloud resource utilization.

5.1. Simulation setup

An open-source workflow scheduling simulator, WorkflowSim toolkit [50], is adopted and extended to simulate our FT scheduling experiments, which has been widely used in many works [11,18]. Hosts in WorkflowSim are modeled with 8,

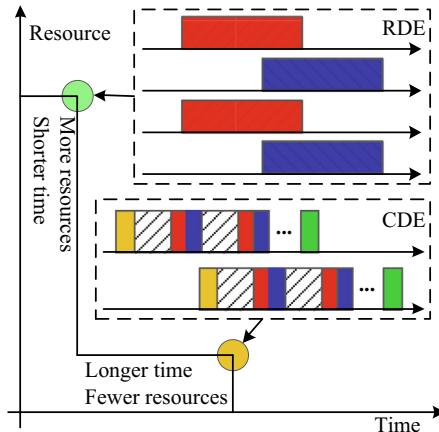


Fig. 5. The comparison of RDE and CDE.

16, 32, and 64-core CPUs connected to 10Gbps Ethernet. The VM types are listed in Table 2. We simulate $N = 2000$ batches of scientific workflows arrived at the cloud, and the length of each time slot τ is 1 h. The deadline of each workflow is a random variable in $[T_{MS}^{min}, T_{MS}^{max}]$ follows the uniform distribution, where T_{MS}^{min} and T_{MS}^{max} are the minimum and maximum workflow make-span, respectively. Then, the workflow deadline is $T_{DL} = T_{MS}^{min} + \rho(T_{MS}^{max} - T_{MS}^{min})$, where $\rho \in [0, 1]$ is called the deadline ratio. For saving the idle resources, T_{th} is set to 15 min, and $U_{th} = 25\%$. We set $T_{dn}^H = 5$ and $T_{rec}^H = 3$ minutes and $T_{dn}^{VM} = 3$ and $T_{rec}^{VM} = 1$ minutes. The Poisson arrival rate of HPF is $\lambda'_H = 0.005$. Like [40], the checkpointing time is $T_{chk} = 0.1T(t_i, VM(k))$.

We conduct the simulations on five real-world scientific workflows, i.e., Montage, CyberShake, Epigenomics, LIGO, and SIPHT, which have been widely applied in the performance evaluation of workflow scheduling. The structures of these workflows can be found in Fig. 6. Moreover, we adopt the trace-based workflow datasets, i.e., the characteristics of the above scientific workflows (such as task count, average data size and average task runtime), which are given in Table 3. These traces are collected from the real running processes of five scientific workflows [50].

The performance metrics are as follows:

- Workflow Guarantee Ratio (WGR): the percentage of workflows finished successfully with the deadline constraints among all the arrived workflows, i.e., $WGR = \sum_{\tau=1}^N n'_{WF(\tau)} / \sum_{\tau=1}^N n_{WF(\tau)}$, where $n'_{WF(\tau)}$ and $n_{WF(\tau)}$ are the number of finished workflows and the number of arrived workflows, respectively.
- Resource Consumption of Tasks (RCT): the total resources used for processing tasks, i.e., $RCT = \sum_{\tau=1}^N \sum_{n=1}^{n_{WF(\tau)}} \sum_{i=1}^{n_{WF(n)}} [T(t_i) + T_{idle}(t_i)]C(k)$, where the time terms include the actual task service time $T(t_i)$ and the idle time $T_{idle}(t_i)$. RCT is measured by counting “time $\times C$ ”. For example, if a task runs on the VM with $C(k) = 8$ (CPUs) for 1 h, the RCT is $8 \times 1 = 8$ CPU-hour.
- Cloud Resource Utilization (CRU): the ratio of RCT over the total computing resources, reflecting the resource utilization of the cloud, i.e., $CRU = \sum_{\tau=1}^N \sum_{n=1}^{n_{WF(\tau)}} \sum_{i=1}^{n_{WF(n)}} T(t_i)C(k) / RCT$.

To show the effects of ReadyFS, in vertical comparison, ReadyFS is compared with its five stripped versions (see Section 5.2). Moreover, in horizontal comparison, three state-of-the-art FT scheduling algorithms are performed to compare with ReadyFS (see Section 5.3).

5.2. Vertical comparison

This section shows the evaluation results on three performance metrics with a variety of different parameters, such as ρ , λ_{WF} , λ_{VM} , λ_H , and ε . Note that we fix $\eta = 0.7$ in this section. Five vertical comparison algorithms are introduced as follows:

- Only-RDE: this algorithm applies only the RDE to guarantee the VMTF and HPF, i.e., removing the CDE from ReadyFS.
- Only-CDE: this algorithm contains only the CDE to ensure the fault tolerance of VMTF and HPF, i.e., removing the RDE from ReadyFS.

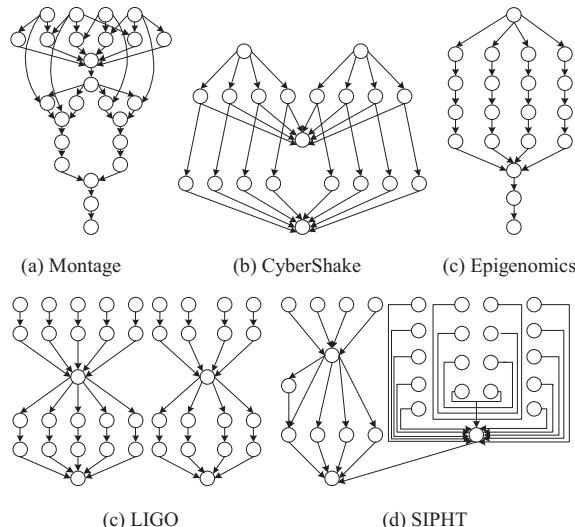


Fig. 6. The structures of five real-world scientific workflows.

Table 3
The characteristics of five scientific workflows.

Workflows (Task count)	Average data size (MB)	Average task runtime (s)
Montage (25)	5.77	9.11
Montage (50)	8.64	10.17
Montage (100)	8.61	10.24
Montage (1000)	8.83	11.38
Epigenomics (24)	146.72	738.34
Epigenomics (46)	188.22	880.89
Epigenomics (100)	582.72	4034.26
Epigenomics (997)	648.94	3866.37
CyberShake (30)	467.77	27.62
CyberShake (50)	481.08	26.46
CyberShake (100)	488.11	22.37
CyberShake (1000)	242.68	23.32
SIPHT (30)	3.42	191.26
SIPHT (60)	6.65	201.19
SIPHT (100)	8.54	179.17
SIPHT (1000)	10.52	179.42
LIGO (30)	14.61	220.57
LIGO (50)	18.35	235.24
LIGO (100)	12.15	210.24
LIGO (1000)	16.71	227.70

- No-DE: this algorithm does not adopt the delay execution mechanism, i.e., all the primary copies or backup copies are executed simultaneously.
- No-ReSC: this algorithm does not employ the ReSC when the HTF happens. All the workflow tasks continue using the original scheduling scheme after the hosts and VMs recover from HTF.
- No-FT: it does not have any FT scheduling mechanism.

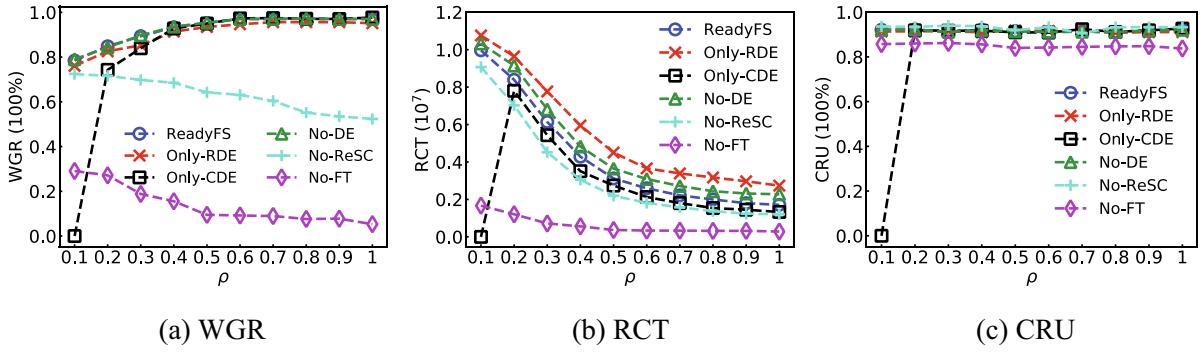
5.2.1. Performance impact of deadline ratio ρ

The simulation results on three performance metrics, such as WGR, RCT, and CRU, by varying the deadline ratio ρ from 0.1 to 1 are given in Fig. 7, where we keep the other parameters fixed, i.e., $\varepsilon = 0.0001$, $\lambda_H = 0.05$, $\lambda_{VM} = 0.05$, and $\lambda_{WF} = 3$. Note that the larger ρ means the larger deadline T_{DL} and the larger sub-makespan $T_{subMS}(t_i)$.

From Fig. 7a, we can observe that the WGRs of ReadyFS, No-DE, Only-RDE, and Only-CDE increase with ρ . In particular, when $\rho = 0.1$, the WGR of Only-CDE is 0%. This is because, in this case, the deadline constraint is hardly satisfied, i.e., Only-CDE cannot be implemented due to the tight deadline, resulting in rejecting all the workflows. When ρ goes up, more workflows can be scheduled by CDE, and then the WGR of Only-CDE becomes larger. We notice that ReadyFS and No-DE have the same and highest WGR, since both of them adopt the same FT scheduling mechanisms. Moreover, ReadyFS and No-DE keep the same pace with Only-CDE when ρ increases. Only-RDE has a slightly lower WGR than ReadyFS. The rationale is that when the HTF happens, the replication strategy needs extra time to re-execute the failed tasks, leading to a small number of workflows that cannot be finished with the deadline constraints. We also find that No-FT and No-ReSC have the least and second least WGR, respectively. Moreover, when ρ increases, their WGRs decrease. This is because, without the ReSC, more workflows will be rejected by No-ReSC. As to No-FT, due to lack of the FT scheduling mechanism, a task with a longer execution time is much more likely to suffer from resource failures.

Fig. 7b shows that the RCTs of all the algorithms decrease when ρ increases (except Only-CDE at $\rho = 0.1$). For ReadyFS, No-DE, Only-RDE, and Only-CDE, the main reason behind this phenomenon is that the workflow tasks can be scheduled to the lower performance VMs when the deadline is larger, resulting in saving VM resources. For Only-ReSC, the above explanation is one of the reasons, and another reason is that the lower WGR also leads to less resource usage. Due to the impact of WGR, the RCT of No-FT goes down when ρ goes up. We can also observe that Only-RDE has the highest RCT, which indicates that the replication indeed requires more resources. No-DE has more RCT than ReadyFS. This is reasonable since we have already proved in Theorem 1, i.e., delay execution mechanism demands fewer resources than No-DE. Because of adopting only the checkpointing mechanism, Only-CDE exhibits lower RCT than ReadyFS. No-FT has the lowest RCT among all the algorithms. This is due to the fact that all the workflow tasks in No-FT are executed only once. No-ReSC has the second-lowest RCT, as a large number of workflows are rejected due to deadline constraints, leading to less resource usage.

Fig. 7c shows the results of CRU. We can see that except No-FT, all the algorithms almost have the same results (except Only-CDE at $\rho = 0.1$), because they use the same RA strategy. No-FT has a slightly lower CRU than the other algorithms. The rationale is that with more failed tasks, No-FT will generate more idle resources. Moreover, some idle resources cannot be utilized and consolidated efficiently. All the curves are almost flat, indicating that the CRU is independent of ρ .

Fig. 7. Performance impact of ρ on the vertical comparison.

5.2.2. Performance impact of replication coefficient ε

This section discusses the performance impact of the replication coefficient ε . Fig. 8 shows the corresponding results by varying ε from 10^{-5} to 10^{-2} , where $\rho \in [0, 1]$, $\lambda_H = 0.05$, $\lambda_{VM} = 0.05$, and $\lambda_{WF} = 3$. Note that the larger ε , the fewer task copies needed by RDE.

As shown in Fig. 8a, the WGRs of ReadyFS, No-DE, Only-RDE, and No-ReSC decrease with the increase of ε . This is because, under the fixed VM and host failure probabilities, these algorithms execute fewer task copies, leading to more workflows will be failed. We also notice that the WGR of Only-RDE decreases rapidly when ε increases, which suggests that the variation of ε has a great impact on Only-RDE. The WGRs of No-FT and Only-CDE are independent of ε . This is because varying ε affects only the RDE. However, No-FT and Only-CDE do not adopt the RDE.

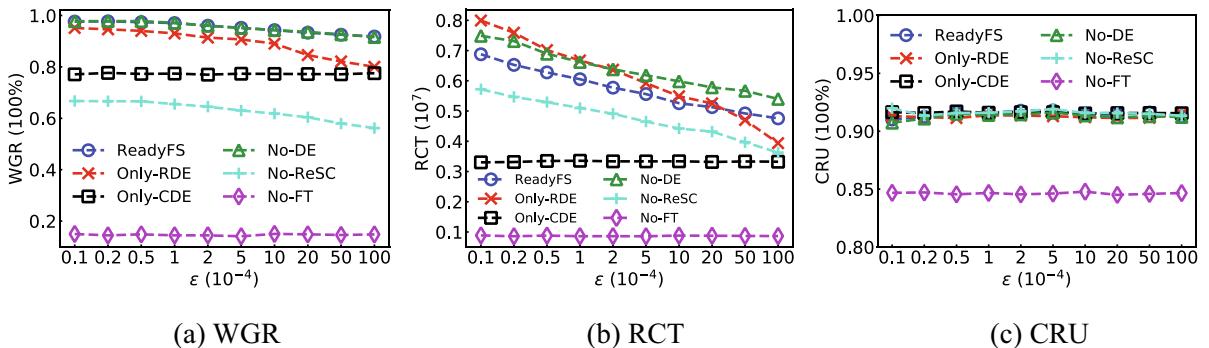
Fig. 8b shows the RCTs of all the algorithms. We find that as ε increases, the RCTs of ReadyFS, No-DE, Only-RDE, and No-ReSC decrease. This is because the larger ε , the fewer task copies are required, resulting in less resource consumption. We also see that the RCT of Only-RDE reduces rapidly when ε goes up, which indicates that the ε causes a significant impact on Only-RDE. In contrast, the curves of No-FT and Only-CDE are flat because they are independent of ε on RCT.

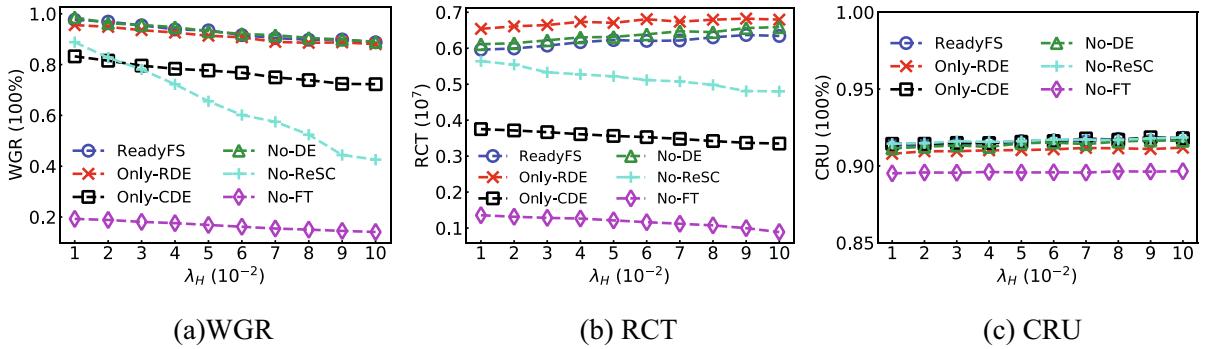
The CRUs of all the algorithms are independent with ε , which are shown in Fig. 8c. No-FT has the lowest CRU as it generates more idle resources that cannot be utilized and consolidated efficiently. The rest of the algorithms almost have the same CRU because all of them adopt the RA strategy.

5.2.3. Performance impact of HTF arrival rate λ_H

This section inspects the performance impact of HTF arrival rate λ_H . We conduct the experiments with λ_H varying from 0.01 to 0.1, where we keep the other parameters fixed, i.e., $\varepsilon = 0.0001$, $\rho \in [0, 1]$, $\lambda_{VM} = 0.05$, and $\lambda_{WF} = 3$. The related results are shown in Fig. 9. Note that the larger λ_H , the higher probability of HTF.

From Fig. 9a, we can see that the WGRs of all the algorithms decrease when λ_H goes up. Particularly, the WGR of No-ReSC goes down faster than that of the other algorithms. The reason is that when λ_H increases, the cloud will suffer from more HTFs. Without the ReSC, No-ReSC tends to reject more workflows. Although with the ReSC, the WGRs of ReadyFS, No-DE, Only-RDE, and Only-CDE still go down. The reason can be explained as follows: The deadline ratio ρ of each arrived workflow is uniformly generated in $[0, 1]$; therefore, if a workflow's deadline is too tight, it is more likely to be rejected when the HTF happens. Due to the lack of FT scheduling mechanisms, No-FT has the lowest WGR, which slightly goes down with λ_H .

Fig. 8. Performance impact of ε on the vertical comparison.

Fig. 9. Performance impact of λ_H on the vertical comparison.

From Fig. 9b, we can observe that the RCTs of ReadyFS, No-DE, and Only-RDE increase with λ_H . This is because these algorithms adopt ReSC to tackle the HTF. So, the larger λ_H , the more tasks need to be re-executed, resulting in more resource consumption. However, the RCT of No-ReSC decreases rapidly when λ_H increases, since No-ReSC has the fastest decreasing rate of WGR. The more rejected workflows, the less resource usage. The RCT of Only-CDE slightly goes down with the increase of λ_H . This is because when the HTF occurs, all the tasks in Only-CDE restart from the latest checkpointing state, leading to significant resource-saving. Furthermore, with the lowest WGR, Only-CDE consumes the least resources. Similarly, due to the factor of WGR, the RCT of No-FT decreases slowly.

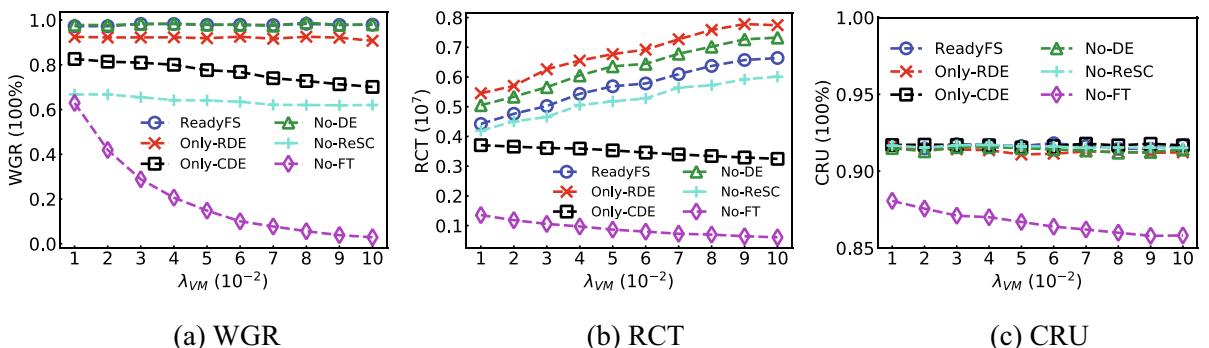
The performance impact of λ_H on CRU is shown in Fig. 9c. We find that the CRUs of all the algorithms are independent of λ_H . All the algorithms except No-FT almost have the same CRU, since all of them adopt the RA strategy. No-FT has a slightly lower CRU than the other algorithms as it produces more idle resources.

5.2.4. Performance impact of VMTF arrival rate λ_{VM}

To investigate the impact of VMTF arrival rate λ_{VM} , we perform the experiments with λ_{VM} varying from 0.01 to 0.1. The related results are shown in Fig. 10, where we keep the other parameters fixed, for example, $\varepsilon = 0.0001$, $\rho \in [0, 1]$, $\lambda_H = 0.05$, and $\lambda_{WF} = 3$. Note that the larger λ_{VM} indicates a higher VMTF probability. Thus, more task copies are required to tackle VMTF by using RDE.

As shown in Fig. 10a, except Only-CDE and No-FT, all the algorithms are not affected by λ_{VM} , i.e., the WGR curves of them are flat when λ_{VM} goes up. This indicates that these algorithms can effectively deal with the VMTF even under different λ_{VM} . However, the WGRs of Only-CDE and No-FT decrease with λ_{VM} . For Only-CDE, the reason is that with the larger λ_{VM} , more checkpoints will be set, resulting in longer task execution time. Moreover, the ρ is randomly selected in $[0, 1]$. Thus, the workflows with tighter deadline constraints are much more likely to be rejected. We can also see that with the increase of λ_{VM} , the WGR of No-FT decreases significantly, indicating that varying λ_{VM} has a great influence on No-FT.

Fig. 10b shows that with the increase of λ_{VM} , the RCTs of all the algorithms, except Only-CDE and No-FT, increase at the same pace. The reason is that although the WGRs of ReadyFS, No-DE, Only-CDE, and No-ReSC are independent of λ_{VM} , with the larger λ_{VM} , more resources will be used to cope with VMTF. The RCT of Only-CDE slightly goes down as all the tasks in Only-CDE are resumed from the latest checkpoints when the VMTF occurs, resulting in significant resource-saving. Moreover, with more workflows rejected by the deadline constraints, Only-CDE consumes fewer resources. Due to the factor of WGR, No-FT has less RCT when λ_{VM} increases.

Fig. 10. Performance impact of λ_{VM} on the vertical comparison.

We can see from Fig. 10c that the CRU curves of all the algorithms except No-FT are flat, which indicates that these algorithms are independent of λ_{VM} . Thanks to the RA strategy, these algorithms almost have the same CRU. However, the CRU of No-FT decreases with the increase of λ_{VM} . This is because, with the larger λ_{VM} , No-FT rejects more workflows due to VMTF. Then, more VMs will be idle and hardly be reused, thus resulting in lower CRU.

5.2.5. Performance impact of workflow arrival rate λ_{WF}

To reveal the performance impact of workflow arrival rate λ_{WF} , we vary λ_{WF} from 1 to 5 with an increment of 0.5. The related simulation results are given in Fig. 11, where we keep the other parameters fixed, i.e., $\varepsilon = 0.0001$, $\rho \in [0, 1]$, $\lambda_H = 0.05$, and $\lambda_{VM} = 0.05$.

We can see from Fig. 11a that all the algorithms keep the stable WGRs for different λ_{WF} because of infinite cloud resources. Moreover, when λ_{WF} increases, the RA strategy is used to deploy new VMs and hosts for arrived workflows dynamically.

With a larger λ_{WF} , more workflows arrive at the cloud. As shown in Fig. 11b, the RCTs of all the algorithms are almost linear in shape, which indicates that the RCTs grow linearly with λ_{WF} . We can also notice that Only-RDE and No-DE have the highest and second-highest RCT; ReadyFS has more RCT than No-ReSC; No-FT and Only-CDE have the lowest and second-lowest RCT.

Fig. 11c shows that the CRUs of all the algorithms slightly increase with λ_{WF} . With more workflows arrived at the cloud, more VM resources will be utilized. Thus, the idle resources have the opportunity to be allocated and utilized, leading to a higher CRU. However, No-FT has the lowest CRU. This is because No-FT produces more idle resources that cannot be utilized sufficiently.

5.3. Horizontal comparison

In this section, we compare our ReadyFS with three state-of-the-art FT scheduling algorithms, namely FASTER [6], ICFWS [11], and EIPR [32]. The comparison algorithms are introduced as follows:

- FASTER [6]: this algorithm applies the PB model to tolerate resource failures. However, the PB model can effectively tackle the HPF but cannot cope with the other failures. Note that FASTER has the RA capability.
- EIPR [32]: this algorithm uses the replication approach to deal with resource failures. However, EIPR cannot dynamically adjust resource demands. To make the comparison fair, we slightly modify EIPR by adding the RA strategy.
- ICFWS [11]: this algorithm combines resubmission with replication to cope with resource failures. A reasonable FT scheduling strategy is selected by the soft deadline constraint. Also, ICFWS is capable of dynamically adjusting resource demands.

However, the above peer algorithms generate only two task copies, and all of them fail to cope with the HPF. Note that we fix $\varepsilon = 0.0001$ in this section.

5.3.1. Performance impact of deadline ratio ρ

The simulation results on three performance metrics by varying ρ from 0.1 to 1 with the increment of 0.1 are given in Fig. 12, where we keep the other parameters fixed, for example, $\lambda_H = 0.05$, $\lambda_{VM} = 0.05$, $\lambda_{WF} = 3$, and $\eta = 0.7$.

We can see from Fig. 12a that the WGR of ReadyFS increases with ρ . This is because when ρ is too small, many workflows scheduled by CDE will be rejected due to tight deadlines. So, ReadyFS keeps the same pace with ρ . However, the WGRs of EIPR and ICFWS decrease when ρ grows. This is because the larger ρ , the larger deadline, and hence a longer task execution time. A task with a longer execution time is more likely to experience various resource failures. Moreover, ICFWS has a higher WGR than EIPR under different ρ because it combines resubmission with replication. We also find that when

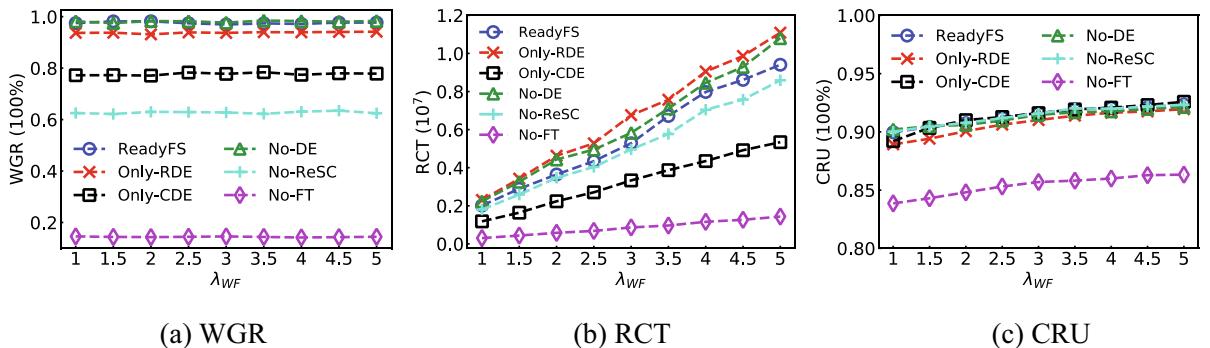
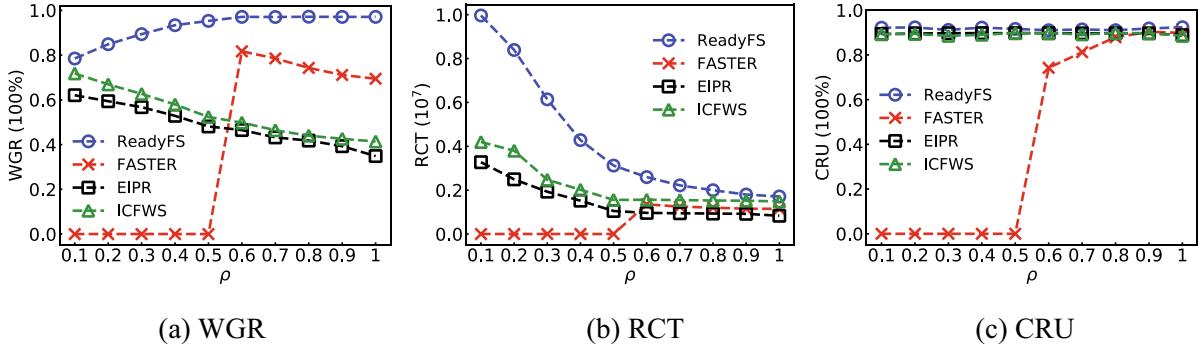


Fig. 11. Performance impact of λ_{WF} on the vertical comparison.

Fig. 12. Performance impact of ρ on the horizontal comparison.

$0.1 \leq \rho \leq 0.5$, the WGR of FASTER is 0%. The main reason is that FASTER employs the PB model to ensure the successful execution of the task. Nevertheless, In PB model, the start time of the backup copy must be greater than or equal to the end time of the primary copy. So, the smaller ρ cannot satisfy this constraint, causing all the workflows are rejected. Since FASTER, EIPR, and ICFWS produce only two task copies, the WGRs of them are less than that of ReadyFS.

Fig. 12b shows that the RCTs of all the algorithms except FASTER decrease when ρ increases. The reason is that the workflow tasks can be scheduled to the lower performance VM under the larger deadline, resulting in more VM resources are saved. For FASTER, because its WGR is 0% when $0.1 \leq \rho \leq 0.5$, the RCT is 0. However, when ρ grows from 0.6 to 1, the RCT decreases with ρ .

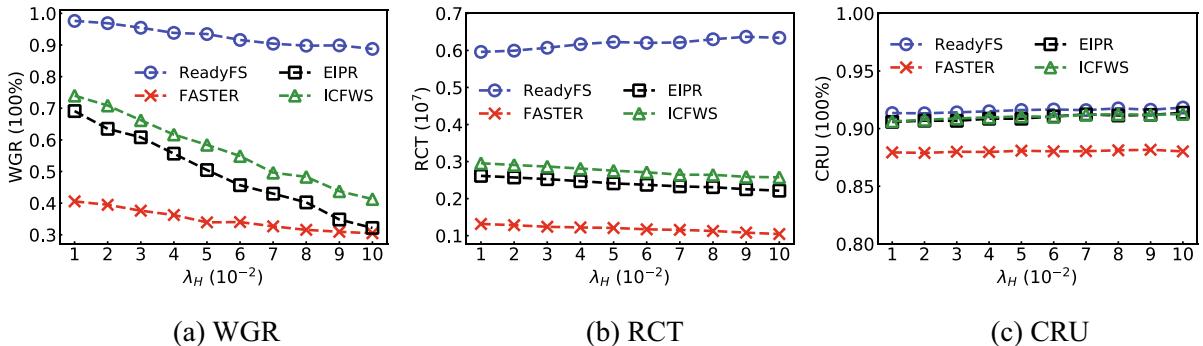
It can be observed from Fig. 12c that except FASTER, all the algorithms keep the same and stable CRU when the ρ varies. However, the CRU of FASTER increases when ρ goes up from 0.6 to 1. This can be explained that the PB model cannot be implemented until $\rho \geq 0.6$. For example, when $\rho = 0.6$, FASTER just meets the condition of PB method. In this case, each task may be mapped to the VM type with the highest performance. So, the residual VM resources cannot be utilized, resulting in a low CRU. However, when ρ grows, this resource waste phenomenon can be alleviated.

5.3.2. Performance impact of HTF arrival rate λ_H

This section discusses the performance impact of λ_H . We run experiments on the variation of λ_H from 0.01 to 0.1. The corresponding results are plotted in Fig. 13, where we keep the other parameters fixed, i.e., $\rho \in [0, 1]$, $\lambda_{VM} = 0.05$, $\lambda_{WF} = 3$, and $\eta = 0.7$.

From Fig. 13a, we find that the WGRs of all the algorithms decrease when λ_H grows. For ReadyFS, the reason is that the ρ is uniformly selected in the range $[0, 1]$; therefore, if the workflow deadline is too tight, this workflow is more likely to be rejected when the HTF happens. We know that the larger λ_H , the higher probability of HTF. Without the FT scheduling mechanism for HTF, the WGRs of FASTER, EIPR, and ICFWS go down with the increase of λ_H . We also observe that ReadyFS always has the highest WGR, ICFWS has a higher WGR than EIPR, and FASTER has the lowest WGR because it is mainly designed for coping with HPF.

As observed in Fig. 13b, the RCT of ReadyFS slightly increases when λ_H goes up. This is due to the fact that ReadyFS adopts the ReSC to handle the HTF. So, the larger λ_H , the more tasks need to be re-executed, resulting in more resource consumption. However, the RCTs of FASTER, EIPR and ICFWS decrease with λ_H . The reason for this phenomenon results from their WGRs, i.e., rejecting more workflows leads to less resource usage.

Fig. 13. Performance impact of λ_H on the horizontal comparison.

From Fig. 13c, the CRU curves of all the algorithms are flat when λ_H goes up, which indicates that the CRUs of these algorithms are independent of λ_H . ReadyFS, EIPR, and ICFWS nearly exhibit the same behavior. Specifically, EIPR and ICFWS have the same CRU, and ReadyFS achieves a slightly better CRU than EIPR and ICFWS. However, FASTER has a lower CRU than the other algorithms, resulting from its idle resources cannot be utilized efficiently.

5.3.3. Performance impact of VMTF arrival rate λ_{VM}

We examine the performance impact of λ_{VM} . The results with λ_{VM} varying from 0.01 to 0.1 are shown in Fig. 14, where we keep the other parameters fixed, for example, $\rho \in [0, 1]$, $\lambda_H = 0.05$, $\lambda_{WF} = 3$, and $\eta = 0.7$. Note that the larger λ_{VM} indicates a higher probability of VMTF.

According to Fig. 14a, the WGR curve of ReadyFS is flat when λ_{VM} increases. This is because ReadyFS can effectively deal with the VMTF under various λ_{VM} . For example, when λ_{VM} goes up, RDE and CDE guarantee the successful completion of the task by adding more task copies and setting more checkpoints, respectively. However, the WGRs of the other algorithms decrease with λ_{VM} . This is because FASTER, EIPR and ICFWS deploy only two task copies, which cannot effectively tackle the VMTF. We also find that the WGRs of EIPR and ICFWS decrease rapidly, which indicates that their WGRs are highly sensitive to the variation of λ_{VM} .

Fig. 14b shows that with the increase of λ_{VM} , the RCT of ReadyFS increases. The reason is that the larger λ_{VM} , the more task copies and the more checkpoints are required, bringing more resource consumption. However, since FASTER, EIPR, and ICFWS have only two task copies, the RCTs of them decrease with λ_{VM} .

From Fig. 14c, we can see that the CRUs of all the algorithms are stable when λ_H goes up. Specifically, EIPR and ICFWS have the same CRU, and ReadyFS achieves a slightly better CRU than EIPR and ICFWS. FASTER has a lower CRU than the others, since its idle resources cannot be utilized efficiently.

5.3.4. Performance impact of workflow arrival rate λ_{WF}

We discuss the performance impact of λ_{WF} by varying it from 1 to 5 with an increment of 0.5. The corresponding results are given in Fig. 15. Similarly, we keep the other parameters fixed, for example, $\rho \in [0, 1]$, $\lambda_H = 0.0$, $\lambda_{VM} = 0.05$, and $\eta = 0.7$.

From Fig. 15a, we can observe that the WGRs of all the algorithms are stable under different λ_{WF} . This is because the cloud can provide infinite computing resources, i.e., when λ_{WF} goes up, the RA strategy can dynamically adjust real-time resource requirements.

With the increase of λ_{WF} , more workflow requests will be received by the cloud. Therefore, the RCTs of all the algorithms are almost linear in shape and grow with λ_{WF} , which is shown in Fig. 15b. We also find that ReadyFS spends much more RCT than the other algorithms since it processes more workflows. EIPR and ICFWS have lower RCTs than ReadyFS. Due to the lowest WGR, FASTER has the lowest RCT.

Fig. 15c shows that the CRUs of all the algorithms slightly increase with λ_{WF} . With more workflows arrived at the cloud, more VM resources will be consumed. Thus, idle resources have the opportunity to be reused, leading to a higher CRU. However, FASTER has a lower CRU than the other algorithms. The reason is that the idle time between the primary copy and the backup copy is hardly utilized by the other tasks.

5.3.5. Performance impact of parallel workload ratio η

This section discusses the performance impact of parallel workload ratio η by varying it from 0 to 1. Fig. 16 shows the corresponding results, where we fix the other parameters, i.e., $\rho \in [0, 1]$, $\lambda_H = 0.05$, $\lambda_{VM} = 0.05$, and $\lambda_{WF} = 3$. Note that the larger η , the more parallel workload $W^p(t_i)$ and the less sequential workload $W^s(t_i)$. Particularly, when $\eta = 0$, $W^s(t_i) = W(t_i)$; when $\eta = 1$, $W^p(t_i) = W(t_i)$.

Fig. 16a gives the WGR results. We can observe that with the increase of η , the WGRs of all the algorithms increase. When $\eta = 0$, because of the deadline constraint, all the algorithms have the 0 WGR. For FASTER, when $\eta \in [0, 0.6]$, the WGR is also 0.

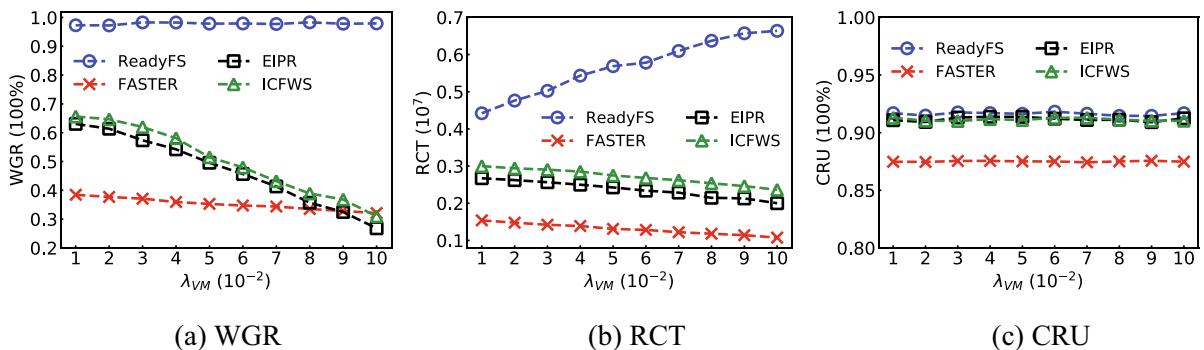
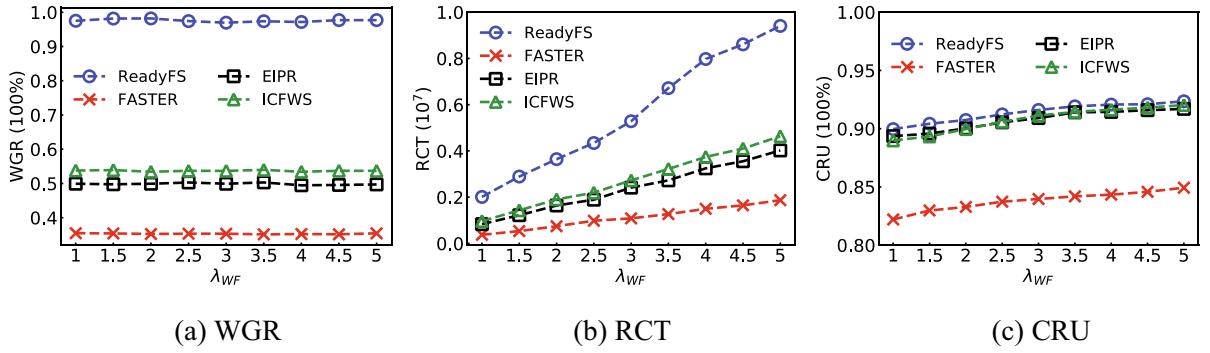


Fig. 14. Performance impact of λ_{VM} on the horizontal comparison.

Fig. 15. Performance impact of λ_{WF} on the horizontal comparison.

This is because it is very difficult for FASTER to meet the deadline constraint by using only the PB strategy. However, with larger η and hence more $W^p(t_i)$, the multi-core CPU can be sufficiently utilized, resulting in the shorter execution time of the task. Thus, more workflows can be finished under the deadline constraints, causing a larger WGR.

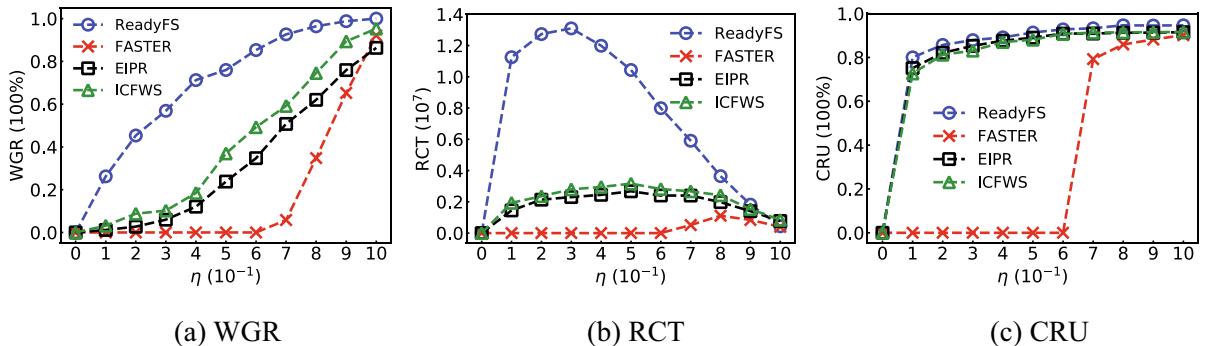
Fig. 16b shows the results of varying η on RCT. We find that each RCT first increases and then decreases with η . There are two factors that cause this phenomenon. First, with a larger η , more workflows will be processed and hence more computing resources are consumed. Second, the VMs with multi-core CPUs are fully utilized with larger η , leading to less resource waste. So, there exists a resource offset between the above two factors. Taking ReadyFS as an example, when $\eta \in [0, 0.3]$, the RCT mainly depends on the number of finished workflows; when $\eta \in [0.4, 1.0]$, the dominating factor for RCT is the parallel workload.

The results on CRU by varying η can be found in Fig. 16c. Due to the 0 WGR, the CRUs of all the algorithms are also 0 when $\eta = 0$. Nevertheless, their CRUs increase when η goes up. The reason is that when η is a small value, although the DVFS technique can save resources, the idle resources generated by sequential workload $W^s(t_i)$ are still wasted. When η goes up, we can make full use of the capability of the multi-core CPU, resulting in a higher CRU.

5.4. Results summary

We find that none of the algorithms can get the highest performance in all the metrics in the vertical comparison. ReadyFS achieves the best on WGR and CRU in all the cases. However, its RCT is easily affected by the parameters varying as it adopts the dynamic RA strategy. Compared with ReadyFS, No-DE has comparable WGR and CRU, but it always exhibits a higher RCT. No-FT almost exhibits the lowest WGR, RCT, and CRU because it cannot complete too many workflow applications successfully due to the lack of FT scheduling mechanism. Only-CDE employs only the CDE that leads to lower RCT. However, when the workflow deadline is too tight, it cannot find the proper VM to process workflows and hence exhibit the lowest WGR. In contrast, Only-RDE uses only the RDE to guarantee the successful execution of the scientific workflows. It adapts to any deadline but needs more computing resources than ReadyFS and Only-CDE. No-ReSC almost performs equally well on CRU compared with ReadyFS. However, it shows a lower WGR than all the algorithms except No-FT, and it is easily affected by λ_H .

Our proposed ReadyFS performs noticeably better than all the comparison algorithms on WGR, but with a higher RCT in the horizontal comparison. All the algorithms except the FASTER exhibit the same CRU as FASTER's idle resources are not

Fig. 16. Performance impact of η on the horizontal comparison.

easily consolidated. ICFWS always has a higher WGR and hence a higher RCT than EIPR. This is because ICFWS combines resubmission with replication, and EIPR adopts only replication. FASTER almost performs the lowest performance on all the metrics. Especially, FASTER has the worst WGR when the deadline is too tight ($\rho \leq 0.5$).

In a word, our proposed ReadyFS is more stable, especially on WGR, which indicates that it can effectively cope with all the resource failures in the cloud computing environment.

6. Conclusions and future work

This paper proposes the ReadyFS for scientific workflow execution in the cloud. This algorithm aims to ensure the successful execution of real-time arrived scientific workflows while tolerating the HPF, HTF, and VMTF. Specifically, three FT scheduling mechanisms, namely RDE, CDE, and ReSC, are proposed to guarantee the fault tolerance of workflow scheduling. We also develop a RA strategy (i.e., including the RS-Up and RS-Down) to adjust resource requirements and improve resource utilization.

Five vertical comparison algorithms and three horizontal comparison algorithms are simulated to compare with our ReadyFS. The simulation results show that ReadyFS has the highest WGR and CRU performance and has the acceptable RCT performance.

In the future, we first conduct workflow scheduling research by executing practical use-cases on the real-world cloud platform. Moreover, we will research how to deploy security services to address the secure FT scheduling problem due to malicious attacks in the cloud.

CRediT authorship contribution statement

Zhongjin Li: Conceptualization, Methodology, Formal analysis, Software, Investigation, Writing - original draft. **Victor Chang:** Validation, Formal analysis, Data curation, Supervision, Funding acquisition, Writing - review & editing. **Haiyang Hu:** Validation, Formal analysis, Funding acquisition. **Hua Hu:** Formal analysis, Visualization, Data curation. **Chuanyi Li:** Methodology, Visualization, Writing - review & editing. **Jidong Ge:** Writing - review & editing, Funding acquisition.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (No. 61802095, 61802167, 61572162), the Zhejiang Provincial Key Science and Technology Project Foundation (No. 2018C01012), the Key Program of Research and Development of China (2016YFC0800803), and the VC Research (No. VCR 0000057). Prof. Victor Chang is the corresponding author.

References

- [1] A. Rista, J. Ajdari, X. Zenuni, Cloud computing virtualization: a comprehensive survey, in: 43rd International Convention on Information, Communication and Electronic Technology (MIPRO), 2020, pp. 462–472.
- [2] J. Kumar, A.K. Singh, R. Buyya, Self directed learning based workload forecasting model for cloud resource management, *Inf. Sci.* 543 (2021) 345–366.
- [3] Y. Zhao, X. Fei, I. Raicu, S. Lu, Opportunities and challenges in running scientific workflows on the cloud, in: 2011 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, 2011, pp. 455–462.
- [4] H. Huang, Z. Lu, R. Peng, Z. Feng, X. Xuan, P.C.K. Hung, S.-C. Huang, Efficiently querying large process model repositories in smart city cloud workflow systems based on quantitative ordering relations, *Inf. Sci.* 495 (2019) 100–115.
- [5] L. Zhang, L. Zhou, A. Salah, Efficient scientific workflow scheduling for deadline-constrained parallel tasks in cloud computing environments, *Inf. Sci.* 531 (2020) 31–46.
- [6] X. Zhu, J.i. Wang, H. Guo, D. Zhu, L.T. Yang, L. Liu, Fault-Tolerant Scheduling for Real-Time Scientific Workflows with Elastic Resource Provisioning in Virtualized Clouds, *IEEE Trans. Parallel Distrib. Syst.* 27 (12) (2016) 3501–3517.
- [7] H. Yan, X. Zhu, H. Chen, H. Guo, W. Zhou, W. Bao, DEFT: Dynamic fault-tolerant elastic scheduling for tasks with uncertain runtime in cloud, *Inf. Sci.* 477 (2019) 30–46.
- [8] M. Hasan, M.S. Goraya, Fault tolerance in cloud computing environment: A systematic survey, *Comput. Ind.* 99 (2018) 156–172.
- [9] R. Jhawar, V. Piuri, Fault tolerance and resilience in cloud computing environments, *Computer and Information Security Handbook*, second ed., 2013
- [10] X. Qiu, Y. Dai, Y. Xiang, L. Xing, Correlation modeling and resource optimization for cloud service with fault recovery, *IEEE Trans. Cloud Comput.* 7 (3) (2019) 693–704.
- [11] G. Yao, Y. Ding, K. Hao, Using Imbalance Characteristic for Fault-Tolerant Workflow Scheduling in Cloud Systems, *IEEE Trans. Parallel Distrib. Syst.* 28 (12) (2017) 3671–3683.
- [12] S. Sobhanayak, A.K. Turuk, B. Sahoo, Analytic modeling of VM failure and repair in cloud datacenter, in: Tenccon IEEE Region Conference, 2015, pp. 1–6
- [13] X. Zhu, X. Qin, M. Qiu, QoS-aware fault-tolerant scheduling for real-time tasks on heterogeneous clusters, *IEEE Trans. Comput.* 60 (6) (2011) 800–812.

- [14] K. Plankenstein, R. Prodan, Meeting soft deadlines in scientific workflows using resubmission impact, *IEEE Trans. Parallel Distrib. Syst.* 23 (5) (2012) 890–901.
- [15] Q. Zheng, B. Veeravalli, On the design of communication-aware fault-tolerant scheduling algorithms for precedence constrained tasks in grid computing systems with dedicated communication devices, *J. Parallel Distrib. Comput.* 69 (3) (2009) 282–294.
- [16] J. Balasangameshwara, N. Raju, Performance-driven load balancing with a primary-backup approach for computational grids with low communication cost and replication cost, *IEEE Trans. Comput.* 62 (5) (2013) 990–1003.
- [17] A. Matani, H.R. Naji, H. Motallebi, A fault-tolerant workflow scheduling algorithm for grid with near-optimal redundancy, *J Grid Comput.* 18 (3) (2020) 377–394.
- [18] W. Chen, R.F. da Silva, E. Deelman, T. Fahringer, Dynamic and fault-tolerant clustering for scientific workflows, *IEEE Trans. Cloud Comput.* 4 (1) (2016) 49–62.
- [19] K. Vinay, S.M. Dilip Kumar, Fault-tolerant scheduling for scientific workflows in cloud environments, in: IEEE 7th International Advance Computing Conference (IACC), 2017, pp. 150–155.
- [20] G. Sun, V. Chang, G. Yang, D. Liao, The cost-efficient deployment of replica servers in virtual content distribution networks for data fusion, *Inf. Sci.* 432 (2018) 495–515.
- [21] J. Yu, R. Buyya, C.K. Tham, Cost-based scheduling of scientific workflow applications on utility Grids, in: International Conference on e-Science and Grid Computing, 2005, pp. 140–147.
- [22] N. Chopra, S. Singh, HEFT based workflow scheduling algorithm for cost optimization within deadline in hybrid clouds, in: International Conference on Computing, Communications and Networking Technologies (ICCCNT), 2013, pp. 1–6.
- [23] Y. Wang, Y. Guo, Z. Guo, T. Baker, W. Liu, CLOSURE: a cloud scientific workflow scheduling algorithm based on attack-defense game model, *Future Gener. Comput. Syst.* 111 (2020) 460–474.
- [24] M.A. Rodriguez, R. Buyya, Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds, *IEEE Trans. Cloud Comput.* 2 (2) (2014) 222–235.
- [25] Z. Li, J. Ge, H. Hu, W. Song, H. Hu, B. Luo, Cost and energy aware scheduling algorithm for scientific workflows with deadline constraint in clouds, *IEEE Trans. Serv. Comput.* 11 (4) (2018) 713–726.
- [26] Z. Tong, H. Chen, X. Deng, K. Li, K. Li, A scheduling scheme in the cloud computing environment using deep Q-learning, *Inf. Sci.* 512 (2020) 1170–1191.
- [27] J.J. Durillo, R. Prodan, H.M. Fard, MOHEFT: a multi-objective list-based method for workflow scheduling, in: International Conference on Cloud Computing Technology and Science Proceedings, 2012, pp. 185–192.
- [28] Z. Zhu, G. Zhang, M. Li, X. Liu, Evolutionary multi-objective workflow scheduling in cloud, *IEEE Trans. Parallel Distrib. Syst.* 27 (5) (2016) 1344–1357.
- [29] M. Kalra, S. Singh, Multi-objective energy aware scheduling of deadline constrained workflows in clouds using hybrid approach, *Wireless Pers. Commun.* 116 (3) (2021) 1743–1764.
- [30] P. Pandey, H. Viswanathan, D. Pompili, Robust orchestration of concurrent application workflows in mobile device clouds, *J. Parallel Distrib. Comput.* 120 (2018) 101–114.
- [31] A. Marahatta, Y. Wang, F. Zhang, A.K. Sangaiah, S.K. Sah Tyagi, Z. Liu, Energy-aware fault-tolerant dynamic task scheduling scheme for virtualized cloud data centers, *Mobile Netw. Appl.* 24 (3) (2019) 1063–1077, <https://doi.org/10.1007/s11036-018-1062-7>.
- [32] R.N. Calheiros, R. Buyya, Meeting Deadlines of Scientific Workflows in Public Clouds with Tasks Replication, *IEEE Trans. Parallel Distrib. Syst.* 25 (7) (2014) 1787–1796.
- [33] Y. Ding, G. Yao, K. Hao, Fault-tolerant elastic scheduling algorithm for workflow in cloud systems, *Inf. Sci.* 393 (2017) 47–65.
- [34] G. Fan, L. Chen, H. Yu, D. Liu, Modeling and analyzing dynamic fault-tolerant strategy for deadline constrained task scheduling in cloud computing, *IEEE Trans. Syst. Man Cybern., Syst.* 50 (4) (2020) 1260–1274.
- [35] A. Olteanu, F. Pop, C. Dobrev, V. Cristea, A dynamic rescheduling algorithm for resource management in large scale dependable distributed systems, *Comput. Math. Appl.* 63 (9) (2012) 1409–1423.
- [36] W. Chen, Y.C. Lee, A. Fekete, A.Y. Zomaya, Adaptive multiple-workflow scheduling with task rearrangement, *J. Supercomput.* 71 (4) (2015) 1297–1317.
- [37] N. Wu, D. Zuo, Z. Zhang, Dynamic fault-tolerant workflow scheduling with hybrid spatial-temporal re-execution in clouds, *Inf.* 10 (5) (2019) 169.
- [38] M. Bougeret, H. Casanova, M. Rabie, Y. Robert, F. Vivien, Checkpointing strategies for parallel jobs, in: International Conference on High Performance Computing Networking, Storage and Analysis, 2011, pp. 1–11.
- [39] F. Wang, X. Liu, Y. Yang, Necessary and sufficient checkpoint selection for temporal verification of high-confidence cloud workflow systems, *Inf. Sci.* 58 (5) (2015) 1–16.
- [40] G. Aupy, A. Benoit, H. Casanova, Y. Robert, Checkpointing strategies for scheduling computational workflows, *Int. J. Netw. Comput.* 6 (1) (2016) 2–26, https://doi.org/10.15803/ijnc.6.1_2.
- [41] A.R. Setlur, S.J. Nirmala, H.S. Singh, S. Khoriya, An efficient fault tolerant workflow scheduling approach using replication heuristics and checkpointing in the cloud, *J. Parallel Distrib. Comput.* 136 (2020) 14–28.
- [42] J. Liu, S. Wang, A. Zhou, S.A.P. Kumar, F. Yang, R. Buyya, Using proactive fault-tolerance approach to enhance cloud service reliability, *IEEE Trans. Cloud Comput.* 6 (4) (2018) 1191–1202.
- [43] S.K. Mondal, J.K. Muppala, Defects per million (DPM) evaluation for a cloud dealing with VM failures using checkpointing, in: IEEE/IFIP International Conference on Dependable Systems and Networks, 2014, pp. 672–677.
- [44] J. Zheng, H. Okamura, T. Dohi, Mean time to security failure of VM-based intrusion tolerant systems, in: IEEE International Conference on Distributed Computing Systems Workshops, 2016, pp. 128–133.
- [45] O. Das, A. Das, Estimating response time percentiles of cloud-based tiered web applications in presence of VM failures, in: International ACM SIGSOFT Conference on Quality of Software Architectures, 2016, pp. 1–10.
- [46] Y.C. Lee, A.Y. Zomaya, Rescheduling for reliable job completion with the support of clouds, *Future Gen. Comput. Syst.* 26 (8) (2010) 1192–1199.
- [47] Z. Tang, Z. Cheng, K. Li, K. Li, An efficient energy scheduling algorithm for workflow tasks in hybrids and DVFS-enabled cloud environment, in: International Symposium on Parallel Architectures, Algorithms and Programming, 2014, pp. 255–261.
- [48] C.Q. Wu, X. Lin, D. Yu, W. Xu, L. Li, End-to-end delay minimization for scientific workflows in clouds under budget constraint, *IEEE Trans. Cloud Comput.* 3 (2) (2015) 169–181.
- [49] Amazon EC2, <http://aws.amazon.com/ec2/>, 2021.
- [50] W. Chen, E. Deelman, WorkflowSim: A toolkit for simulating scientific workflows in distributed environments, in: IEEE International Conference on E-Science, 2012, pp. 1–8.



Zhongjin Li is a lecturer at the School of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou, China. He received the PhD degree from Software Institute, Nanjing University, Nanjing, China, in 2017. His research interests include cloud computing, workflow scheduling, and mobile edge computing.



Victor Chang is a full Professor with Teesside University, Middlesbrough, UK. He is an Editor-in-Chief of IJOCI and OJBD journals, an Editor of IEEE TII and Information Fusion, the Founding Chair of two international workshops, and the Founding Conference Chair of IoTBD and COMPLEXIS. He has given 23 keynotes in international conferences and received numerous awards. These include IEEE Outstanding Service Award 2015, the 2016 European Award: Best Project in Research, the Outstanding Young Scientist 2017, the INSTICC Service Award 2017–2020, the ICGI 2017 Special Award, top 2% Scientist 2017 & 2019 and most productive Scientist of Data Analytics between 2010 and 2019 and Best Researcher Award 2020.



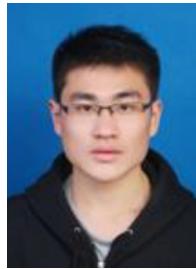
Haiyang Hu is a full Professor at School of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou, China. He received the BS, MS, and PhD degrees in computer science from Nanjing University, Nanjing, China, in 2000, 2003, and 2006, respectively. His research interests include cloud computing, computer network, workflow scheduling, mobile computing, and distributed computing.



Hua Hu is a full professor of Hangzhou Dianzi University, China. He received the Ph.D., MS and BS degree in Computer Science from Zhejiang University, China in 1998, 1992 and 1989, respectively. His research interests mainly include parallel computing and distributed system and pervasive computing.



Chuanyi Li is a PhD at the Software Institute, Nanjing University, Nanjing, China. He received the BS and PhD degrees in Software Institute from Nanjing University, Nanjing, China, in 2012 and 2017, respectively. His research interests include software engineering, systems and software quality assurance, process modeling, simulation and improvement, process mining.



Jidong Ge is an Associate Professor at Software Institute, Nanjing University, Nanjing, China. He received his PhD degree in Computer Science from Nanjing University in 2007. His current research interests include cloud computing, workflow scheduling, software engineering, workflow modeling, stochastic network optimization, process mining.