Quantitative Fault-Tolerance for Reliable Workflows on Heterogeneous IaaS Clouds

Guoqi Xie, *Member, IEEE*, Gang Zeng, *Member, IEEE*, Renfa Li, *Senior Member, IEEE*, and Keqin Li, *Fellow, IEEE*

Abstract—Reliability requirement is one of the most important quality of services (QoS) and should be satisfied for a reliable workflow in cloud computing. Primary-backup replication is an important software fault-tolerant technique used to satisfy reliability requirement. Recent works studied quantitative fault-tolerant scheduling to reduce execution cost by minimizing the number of replicas while satisfying the reliability requirement of a workflow on heterogeneous infrastructure as a service (IaaS) clouds. However, a minimum number of replicas does not necessarily lead to the minimum execution cost and shortest schedule length in a heterogeneous IaaS cloud. In this study, we propose the quantitative fault-tolerant scheduling algorithms QFEC and QFEC+ with minimum execution costs and QFSL and QFSL+ with shortest schedule lengths while satisfing the reliability requirements of workflows. Extensive experimental results show that (1) compared with the state-of-the-art algorithms, the proposed algorithms achieve less execution cost and shorter schedule length, although the number of replicas are not minimum; (2) QFEC and QFEC+ are designed to reduce execution cost, and QFEC+ is better than QFEC for all low-parallelism and high-parallelism workflows; and (3) QFSL and QFSL+ are designed to decrease schedule length, and QFSL+ is better than QFSL for all low-parallelism and high-parallelism workflows.

Index Terms—Infrastructure as a service (IaaS), quantitative fault-tolerance, reliability requirement, execution cost, schedule length

1 Introduction

1.1 Background

Cloud computing assembles large networks of virtualized information and communication technology (ICT) services such as hardware resources (e.g., CPU, storage, and network), software resources (e.g., databases, application servers, and web servers) and applications [1], [2]. These services are referred to as infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS) in industry [1]. Workflows have been frequently used to model large-scale scientific problems in areas such as bioinformatics, astronomy, and physics [3]. Cloud computing has shown a great deal of promise as a cost-effective computing model for supporting scientific workflows [4]. With old, slow machines being replaced with new, fast machines continuously, cloud computing systems are believed to become more heterogeneous [5], [6]. IaaS clouds provide virtualized machines (VMs) for users to deploy their own applications, and therefore are most suitable for executing scientific workflows [7], [8]. Real-world IaaS cloud services such as Amazon EC2, provide VM instances with different CPU capacities to meet different demands of various applications [7]. Meanwhile, the frequency of transient failures has increased dramatically in executing workflows in IaaS clouds [9], [10]. As the scale and complexity of IaaS clouds increase, failures occur frequently and adversely affect resource management and scheduling [11]. Transient failures of machines have caused serious problems in quality of service (QoS) [10], [12], particularly in reliability requirement. As indicated by [10], in practice, many cloud-based services failed to fulfill their reliability requirements. However, reliability requirement is one of the most important QoS [13], [14] and should be satisfied for reliable workflow in heterogeneous IaaS clouds.

1.2 Motivation

Cloud computing offers elastic computing capacity, visualized resources, and pay-as-you-go billing models [4], [15]. These capabilities enable users to do so by paying only for the resources they used rather than requiring large upfront investments. Therefore, cost is one major criterion considered in cloud services, and high cost has an adverse impact on the system performance, especially when the resources are limited. Moreover, for the economic attributes of cloud services, more resource consumption comes with higher economic cost. Therefore, cost should be reduced as far as possible while satisfying the reliability requirement.

Scientific workflows demand massive resources from various computing infrastructures to process massive amount of big data on clouds [16]. Many workflows are commonly modeled as a set of tasks interconnected via data

- *G. Xie and R. Li are with the College of Computer Science and Electronic Engineering, Key Laboratory for Embedded and Network Computing of Hunan Province, Hunan University, Changsha, Hunan 410082, China. E-mail: {xgqman, lirenfa}@hnu.edu.cn.*
- *G. Zeng is with the Graduate School of Engineering, Nagoya University, Aichi 4648603, Japan. E-mail: sogo@ertl.jp.*
- *K. Li is with the College of Computer Science and Electronic Engineering, Hunan University, Hunan, Sheng 410006, China and also with the Department of Computer Science, State University of New York, New Paltz, NY 12561. E-mail: lik@newpaltz.edu.*

Manuscript received 23 Nov. 2016; revised 18 Sept. 2017; accepted 1 Dec. 2017. Date of publication 0 . 0000; date of current version 0 . 0000.
(Corresponding author: Renfa Li.)
Recommended for acceptance by M. Demirbas.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TCC.2017.2780098

or computing dependencies [3]. A workflow with precedence constrained tasks is described as a directed acyclic graph (DAG) [3], [7], in which the nodes represent the tasks and the edges represent the communication messages between tasks. The problem of scheduling tasks on multiprocessors is NP-hard [17], and the scheduling of workflows on clouds is an NP-hard optimization problem [3], [7], [16]. Similarly, scheduling a workflow while satisfying the reliability requirement on heterogeneous IaaS clouds is also an NP-hard optimization problem.

Fault-tolerant scheduling is an effective method to enhance the reliability of a workflow, and primary-backup replication is an important software fault-tolerant technique used to satisfy the reliability requirement. Existing fault-tolerant scheduling algorithms either use one backup for each primary to tolerate one failure based on the passive replication scheme [18], [19], [20], which cannot tolerate potential multiple failures, or use fixed ε backups for each primary to tolerate ε failures in the same time based on active replication scheme, which can satisfy the reliability requirement, but can cause high redundancy and cost [21], [22], [23], [24]. Recent studies presented the quantitative fault-tolerant scheduling algorithms MaxRe [25] and RR [26] by exploring minimum numbers of replicas (including primary and backups) to reduce cost while satisfying the reliability requirement of a workflow in heterogeneous IaaS clouds. The main difference between MaxRe and RR is the methods of calculating the sub-reliability requirement of each task (refer to Section 4.2 for more details). Quantitative fault-tolerant scheduling means that different tasks may have different numbers of replicas and could generate less cost than the previous active replication scheme, in which all the tasks have equal and fixed ε+1 replicas, as indicated by [25], [26]. However, a major limitation of MaxRe and RR is that the minimum number of replicas does not mean minimum execution cost and shortest schedule length in heterogeneous IaaS clouds because the same task has different execution times on different VMs.

1.3 Our Contributions

The main contributions of this study are as follows.

(1) We propose the quantitative fault-tolerance with minimum execution cost (QFEC) and QFEC+ algorithms for a workflow. QFEC is implemented by iteratively selecting available replicas and VMs with the minimum execution time for each task until its sub-reliability requirement is satisfied. QFEC+ is implemented by filtering out partial QFEC-selected replicas and VMs for each task with less redundancy while still satisfying its sub-reliability requirement.

(2) We propose the quantitative fault-tolerance with shortest schedule length (QFSL) and QFSL+ algorithms for a workflow. QFSL is implemented by iteratively selecting available replicas and VMs with the minimum earliest finish time (EFT) for each task until its sub-reliability requirement is satisfied. QFSL+ is implemented by filtering out partial QFSL-selected replicas and VMs for each task with less redundancy while still satisfying its sub-reliability requirement.

(3) Extensive experiments on five real workflows, including linear algebra, Gaussian elimination, diamond graph, complete binary tree, and fast Fourier transform, were conducted. Experimental results verify that the effectiveness of the proposed algorithms in reducing execution cost and schedule length.

The rest of this paper is organized as follows. Section 2 reviews related research. Section 3 presents the models. Section 4 presents quantitative fault-tolerance with minimum execution cost. Sections 5 presents quantitative fault-tolerance with shortest schedule length. Section 6 verifies all the presented algorithms. Section 7 concludes this study.

2 Related Work

Given that this study focuses on the fault-tolerance of workflows on heterogeneous IaaS clouds, this section reviews related fault-tolerant scheduling of the DAG-based workflow.

The widely accepted reliability model was presented by Shatz and Wang [27], in which the transient failure of each VM is characterized by a constant failure rate per time unit λ. The reliability during the interval of time t is $e^{-\lambda t}$. That is, the failure occurrence follows a constant parameter Poisson law [11], [12], [25], [26], [27]. In [28], [29], Benoit et al. proved that evaluating the reliability of a DAG-based workflow belongs to an NP-complete problem.

Intuitively, a higher reliability could result in a longer schedule length of a workflow and the problem of optimizing schedule length and reliability is considered a typical bi-criteria optima or Pareto optima problem [30], [31], [32], [33]. Active replication scheme [21], [22], [23], [24], [25], [26] and passive replication (i.e., backup/restart) scheme [11], [18], [19], [20], which correspond to resource and time redundancy, respectively, are widely applied in scheduling to provide high reliability. Replication on the same processor is a restart scheme and thus is considered as an improved version of the passive replication scheme [21], [25], [26]. The reason is that the system is subsequently restarted when a processor crashes to continue just as if no failure had occurred.

For the passive replication scheme, whenever a VM fails, the task will be rescheduled to proceed on a backup VM. The main representative methods include efficient fault-tolerant reliability cost driven [18], efficient fault-tolerant reliability driven [19], and minimum completion time with less replication cost [20]. With regard to their limitations, first, these approaches assume that no more than one failure occurs at one moment; they are too ideal to tolerate potential multiple failures. Second, passive replication also supports multiple backups for each primary [11], but is unsuitable for a workflow that must satisfy the reliability requirement. The reason is that, once a VM failure is detected, the scheduler should reschedule the task located on the failed VM and reassign it to a new VMs and generate randomized numbers of replicas, which will lead to unpredictable execution cost and schedule length as pointed out in [26]. Problems with the backup/restart scheme become even more complex when a randomized number is used [26].

For the active replication scheme, each task is simultaneously replicated on several VMs, and the task will succeed if at least one of the VMs does not fail. Each task uses fixed ε backups for each primary to tolerate ε failures [21], [22], [23], [24], [34]. The active replication scheme is suitable

TABLE 1
Important Notations in This Study

Notation	Definition
$c_{i,j}$	Communication time between the tasks n_i and n_j
$w_{i,k}$	Execution time of the task n_i on the VM u_k
$\overline{w_i}$	Average execution time of the task n_i
$rank_{\mathrm{u}}(n_i)$	Upward rank value of the task n_i
$\lvert X \rvert$	Size of the set X
λ_k	Constant failure rate per time unit of the VM u_k
num_i	Number of replicas of the task n_i
$NR(G)$	Total number of the replicas of the workflow G
$cost(G)$	Total execution cost of the workflow G
$SL(G)$	Total schedule length of the workflow G
n_i^x	xth replica of the task n_i
$u_{pr(n_i^x)}$	Assigned VM of the replica n_i^x
$R(n_i, u_k)$	Reliability of the task n_i on the VM u_k
$R(n_i)$	Reliability of the task n_i
$R(G)$	Reliability of the workflow G
$R_{\mathrm{seq}}(G)$	Reliability requirement of the workflow G
$R_{\mathrm{seq}}(n_i)$	Sub-reliability requirement of the task n_i
$R_{\mathrm{up\_seq}}(n_i)$	Upper bound on reliability requirement of the task n_i

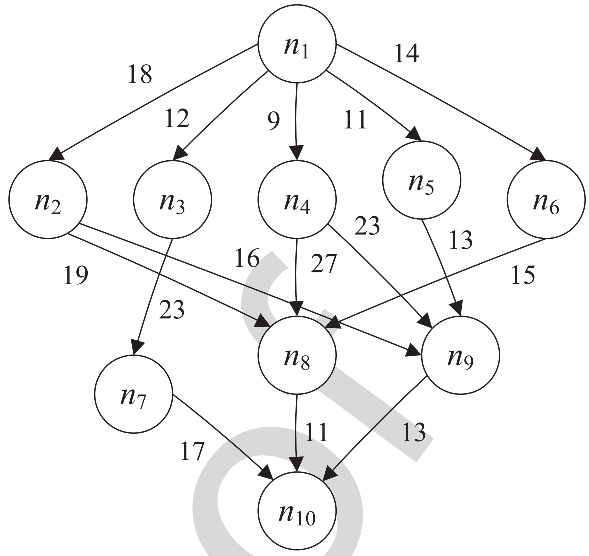


Fig. 1. Motivating example of a DAG-based workflow with ten tasks [35], [36], [37].

for a workflow that must satisfy reliability requirements because adding any one replica can provide enhancement of reliability for the workflow. The main problem with this approach is that it must tolerate ε failures with high redundancy to satisfy the reliability requirement of the workflow, as indicated by [25]. Although the reliability requirement can be satisfied, high redundancy causes high execution cost and long schedule length.

Given the problems of active and passive replication schemes, recent studies began to explore quantitative backups for each task approach to satisfy the reliability requirement of a workflow [25], [26]. In [25] and [26], the authors proposed the fault-tolerant scheduling algorithms MaxRe and RR, which incorporate reliability analysis into the active replication scheme and exploit a minimum number of backups for different tasks by considering the sub-reliability requirement of each task. However, as discussed in Section 1.2, in heterogeneous IaaS clouds, a minimum number of replicas does not mean minimum execution cost and shortest schedule length.

3 Models and Preliminaries

Table 1 lists important notations and their definitions that are used in this study.

3.1 Workflow Model

Let $U = \{u_1, u_2, \ldots, u_{|U|}\}$ represent a set of heterogeneous VMs on IaaS clouds, where $|U|$ is the size of set U. In this study, for any set X, $|X|$ is used to denote size. Similar to [25], [35], [36], [37], we also presume that communication can be overlapped with computation, which means data can be transmitted from one VM to another while a task is being executed on the recipient VM.

A workflow running on VMs is represented by a DAG $G = (N, W, M, C)$ with known values [3], [7], [8], [25], [26], [35], [36], [37]. (1) N represents a set of nodes in G, and each node $n_i \in N$ is a task with different execution times on different VMs. $pred(n_i)$ is the set of immediate predecessor tasks of n_i, while $succ(n_i)$ is the set of immediate successor tasks of n_i. Tasks without predecessor tasks are denoted by n_{entry}; and tasks with no successor tasks are denoted by n_{exit}. If a workflow has multiple entry or multiple exit tasks, then a dummy entry or exit task with zero-weight dependencies is added to the graph. W is a $|N| \times |U|$ matrix in which $w_{i,k}$ denotes the execution time of n_i running on u_k. In addition, task executions of a given workflow are assumed to be non-preemptive which is possible in many systems [25], [26], [35], [36], [37].

(2) Two tasks with immediate precedence constraints need to exchange messages. M is a set of communication edges, and each edge $m_{i,j} \in M$ represents a communication from n_i to n_j. C represents the corresponding communication time set of M. Accordingly, $c_{i,j} \in C$ represents the communication time of $m_{i,j}$ if n_i and n_j are assigned to different VMs. If both tasks n_i to n_j are allocated to the same VM, $c_{i,j}$ becomes zero because we assume that the intra-VM communication cost is negligible [25], [26], [35], [36], [37]. The execution time is also neglected if tasks are mapped to different VMs on the same physical machine because these VMs have the same shared memory. In this study, we assume each physical machine only contains one VM for better explaining the proposed algorithms.

Fig. 1 shows a motivating workflow with tasks and messages [35], [36], [37]. Table 2 is a matrix of the execution

TABLE 2
Execution Times of Tasks on Different VMs of the Motivating Workflow [35], [36], [37]

Task	u_1	u_2	u_3
n_1	14	16	9
n_2	13	19	18
n_3	11	13	19
n_4	13	8	17
n_5	12	13	10
n_6	13	16	9
n_7	7	15	11
n_8	5	11	14
n_9	18	12	20
n_{10}	21	7	16

times shown in Fig. 1. The example shows 10 tasks executed on 3 VMs $\{u_1, u_2, u_3\}$. The weight 14 of n_1 and u_1 in Table 2 represents execution time of n_1 on u_1, denoted by $w_{1,1} = 14$. Clearly, the same task has different execution times on different VMs due to the heterogeneity of the VMs. The weight 18 of the edge between n_1 and n_2 represents communication time, denoted by $c_{1,2}$ if n_1 and n_2 are not assigned to the same VM. For simplicity, all the units of all parameters are ignored in the example.

3.2 Reliability Model

Two major types of failures exist, that is, transient failure and permanent failure; this study considers the transient failure of VMs. In general, the occurrence of transient failure for a task in a DAG-based workflow follows a Poisson distribution [25], [26], [27], [28], [32]. The reliability of an event in unit time t is denoted by

$$R(t) = e^{-\lambda t},$$

where λ is the *constant failure rate per time unit* for a VM. We use λ_k to represent the constant failure rate per time unit of the VM u_k. The reliability of n_i executed on u_k in its execution time is denoted by

$$R(n_i, u_k) = e^{-\lambda_k w_{i,k}}, \tag{1}$$

and the failure probability for n_i without using the active replication scheme is

$$1 - R(n_i, u_k) = 1 - e^{-\lambda_k w_{i,k}}. \tag{2}$$

Similar to [26], we also use the active replication scheme to implement fault-tolerance in this study. The reason has been explained in Section 2. Considering that each task has a certain number of replicas with the active replication scheme, we define num_i ($num_i \leqslant |U|$) as the number of replicas of n_i. Thus, the replica set of n_i is $\{n_i^1, n_i^2, \ldots, n_i^{num_i}\}$, where n_i^1 is the primary and the others are the backups. Then, the total number of replicas for the workflow is

$$NR(G) = \sum_{i=1}^{|N|} num_i. \tag{3}$$

As long as one replica of n_i is successfully completed, then we can recognize that no failure occurs for n_i, and the reliability of n_i is updated to

$$R(n_i) = 1 - \prod_{x=1}^{num_i} \left(1 - R\Big(n_i^x, u_{pr(n_i^x)}\Big)\right), \tag{4}$$

where $u_{pr(n_i^x)}$ represents the assigned VM of n_i^x. Note that replication on the same processor is not allowed because it is an improved version of the passive replication scheme as pointed out earlier. Then, the reliability of the workflow with precedence-constrained tasks should be

$$R(G) = \prod_{n_i \in N} R(n_i). \tag{5}$$

In [26], communication and computation failures are considered; however, some communication networks themselves provide fault-tolerance. For instance, routing information protocol and open shortest path first are designed to reroute packets to ensure that they reach their destination [38]. Therefore, similar to [20], [25], [39], this study only considers VM failure and assumes reliable communication.

TABLE 3
Upward Rank Values for Tasks of the Motivating Workflow

Task	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9	n_{10}
$rank_{\rm u}(n_i)$	108	77	80	80	69	63.3	42.7	35.7	44.3	14.7

3.3 Cost Model

The cost model in this study is based on a pay-as-you-go condition, and the users are charged according to the amount of time that they have used processors according to the current commercial clouds [40]. Each processor has an individual unit price because processors in the system are completely heterogeneous [41], [42]. Therefore, the computation execution cost of the workflow is the sum of the execution time values of all replicas of tasks and the corresponding execution cost unit prices of VMs; that is,

$$cost(G) = \sum_{n_i \in N} cost(n_i) = \sum_{n_i \in N} \left(\sum_{y=1}^{num_i} w_{i,pr(n_i^y)} \times \gamma_{pr(n_i^y)} \right), \tag{6}$$

where $\gamma_{pr(n_i^y)}$ represents the execution cost unit price of the VM $u_{pr(n_i^y)}$.

3.4 Fault-Tolerant Scheduling

Scheduling tasks for a DAG-based workflow with fastest execution is a well-known NP-hard optimization problem and heterogeneous earliest finish time (HEFT) is one of the most famous scheduling algorithms [35]. List scheduling is the most well-known method for a DAG-based workflow and includes two phases: the first phase orders tasks based on the descending order of priorities (task prioritization), whereas the second phase allocates each task to the appropriate VM (task allocation). Similarly, fault-tolerant scheduling for a DAG-based workflow is also an NP-hard problem [28], [29], and fault-tolerant list scheduling also contains the following two phases.

(1) Task prioritization. Similar to HEFT [35] and state-of-the-art MaxRe [25] and RR algorithms [26], this study also uses the well-known upward rank value ($rank_{\rm u}$) of a task (Eq. (7)) as the task priority standard. In this case, the tasks are ordered by descending order of $rank_{\rm u}$, which are obtained by Eq. (7) [35], as follows:

$$rank_{\rm u}(n_i) = \overline{w_i} + \max_{n_j \in succ(n_i)} \{c_{i,j} + rank_{\rm u}(n_j)\}, \tag{7}$$

in which $\overline{w_i}$ represents the average execution times of task n_i and is calculated by $\overline{w_i} = (\sum_{k=1}^{|U|} w_{i,k})/|U|$. Table 3 shows the upward rank values of all the tasks of the motivating example. n_i can be allocated to VM only if all the predecessors of n_i have been assigned. We assume that two tasks n_i and n_j satisfy $rank_{\rm u}(n_i) > rank_{\rm u}(n_j)$; if there is no precedence constraint between n_i and n_j, n_i does not necessarily take precedence for n_j to be assigned. Finally, the task assignment order in the motivating example G is $\{n_1, n_3, n_4, n_2, n_5, n_6, n_9, n_7, n_8, n_{10}\}$.

(2) Task allocation. Two types of fault-tolerant scheduling exist for workflow, namely, the strict schedule and the general schedule [28], [43]. In the strict schedule, each task should wait for the completion (including success and fail) of all the replicas of its predecessors before starting its execution. In the general schedule, the execution of each task can start as soon as one replica of each predecessor has successfully completed. In other words, the strict schedule is equivalent to a compile-time static scheduling, whereas the general schedule is equivalent to a run-time dynamic scheduling. In this study, we only discuss the strict schedule for predictable schedule result during the design phase.

We let the attributes $EST\left(n_i^x, u_k\right)$ and $EFT\left(n_i^x, u_k\right)$ represent the earliest start time (EST) and the earliest finish time, respectively, of the replica n_i^x on the VM u_k. We let $EFT(n_i^x, u_k)$ be the task allocation criterion in this study because it satisfies the local optimum of each precedence-constrained task by using the greedy policy. Given that the strict schedule is used, the aforementioned attributes are calculated as follows:

$$\begin{cases} EST\left(n_{\text{entry}}^x, u_k\right) = 0 \\ EST\left(n_i^x, u_k\right) = \max\left\{ \begin{array}{l} avail[k], \\ \max\limits_{n_h \in pred(n_i), v \in [1, num_h)]} \left\{ AFT(n_h^v) + c'_{h,i} \right\} \end{array} \right\}, \end{cases} \tag{8}$$

and

$$EFT\left(n_i^x, u_k\right) = EST\left(n_i^x, u_k\right) + w_{i,k}. \tag{9}$$

$avail[k]$ is the earliest available time when VM u_k is ready for task execution. $AFT(n_h^v)$ is the actual finish time of the replica n_h^v and is calculated by

$$AFT(n_h^v) = EFT(n_h^v, u_{pr(n_h^v)}). \tag{10}$$

$c'_{h,i}$ represents the communication time between n_h^v and n_i^x. If n_h^v and n_i^x are allocated to the same VM, then $c'_{h,i} = 0$; otherwise, $c'_{h,i} = c_{h,i}$. n_i^x is allocated to the VM with the minimum EFT by using the insertion-based scheduling policy that n_i^x can be inserted into the slack with the minimum EFT.

The final schedule length of the workflow is the AFT of the replica of the exit task n_{exit}; this replica has the maximum AFT among all replicas of n_{exit}. That is, we have

$$SL(G) = \max_{y \in [1, num_{\text{exit}}]} \{EFT(n_{\text{exit}}^y)\}. \tag{11}$$

4 Quantitative Fault-Tolerance with Minimum Execution Cost

4.1 Problem Description

The problem of minimizing execution cost with reliability requirement can be formally described as follows: We assume that we are given a workflow G and a heterogeneous VM set U. The problem is to assign replicas and corresponding VMs for each task; at the same time, we must minimize the execution cost of the workflow and ensure that the obtained reliability value $R(G)$ satisfies the reliability requirement $R_{\text{seq}}(G)$. The formal description is to find the replicas and VM assignments of all tasks to minimize execution cost

$$cost(G) = \sum_{n_i \in N} cost(n_i) = \sum_{n_i \in N} \left(\sum_{y=1}^{num_i} w_{i,pr(n_i^y)} \times \gamma_{pr(n_i^y)} \right),$$

subject to reliability requirement:

$$R(G) = \prod_{n_i \in N} (R(n_i)) \geqslant R_{\text{req}}(G),$$

for $\forall i : 1 \leqslant i \leqslant |N|$.

4.2 Satisfying Reliability Requirement

The heuristic MaxRe [25] and RR [26] algorithms was presented to transfer the reliability requirement of the workflow to the sub-reliability requirement of each task. However, there are two issues should be concerned to improve execution cost.

(1) Calculate sub-reliability requirements of all tasks. In MaxRe, the sub-reliability requirement of each task is still calculated by $R_{\text{req}}(n_i) = \sqrt[|N|]{R_{\text{req}}(G)}$. Such calculation was improved in RR, where the sub-reliability requirement of the entry task is still calculated by $R_{\text{req}}(n_1) = \sqrt[|N|]{R_{\text{req}}(G)}$, and the sub-reliability requirements of the remainder of tasks (i.e., non-entry tasks) are calculated continuously based on the actual reliability achieved by previous allocations

$$R_{\text{req}}(n_{seq(j)}) = \sqrt[|N|-j+1]{\frac{R_{\text{req}}(G)}{\prod_{x=1}^{j-1} R(n_{seq(x)})}}, \tag{12}$$

where $n_{seq(j)}$ represents the jth assigned task. However, RR merely recalculates the sub-reliability requirement (Eq. (12)) of the task $n_{seq(x)}$ based on the actual reliability achieved by previous allocations of $n_{seq(x)}$, not based on succeeding tasks of $n_{seq(j)}$.

(2) Satisfy sub-reliability requirements of all tasks. Both MaxRe and RR iteratively select available replicas and VMs with the maximum reliability value for each task to minimize the number of replicas, and thereby to reduce execution cost, until the sub-reliability of the task is satisfied. However, the minimum number of replicas does not mean minimum execution cost and shortest schedule length because of the heterogeneity of VMs.

We make the following improvement to solve the aforementioned two problems:

(1) In calculating sub-reliability requirements of all tasks, we let $\sqrt[|N|]{R_{\text{req}}(G)}$ be the upper bound on the reliability requirement of the task n_i, that is,

$$R_{\text{up\_req}}(n_i) = \sqrt[|N|]{R_{\text{req}}(G)}. \tag{13}$$

Then, we have the following strategy: we assume that the task to be assigned is $n_{seq(j)}$, where $n_{seq(j)}$ represents the jth assigned task, then $\{n_{seq(1)}, n_{seq(2)}, \ldots, n_{seq(j-1)}\}$ represents the task set with assigned tasks and $\{n_{seq(j+1)}, n_{seq(j+2)}, \ldots, n_{seq(|N|)}\}$ represents the task set with unassigned tasks. We presuppose that each task in $\{n_{seq(j+1)}, n_{seq(j+2)}, \ldots, n_{seq(|N|)}\}$ is assigned to the VM with reliability value on the upper bound (Eq. (13)) to ensure that the reliability of the workflow is satisfied at each task assignment. Thus, when assigning $n_{seq(j)}$, the reliability requirement of G is bound (Eq. (13)). Hence, when assigning $n_{seq(j)}$, the reliability requirement of G is

$$R_{\mathrm{req}}(G) = \prod_{x=1}^{j-1} R(n_{seq(x)}) \times R_{\mathrm{req}}(n_{seq(j)}) \times \prod_{y=j+1}^{|N|} R_{\mathrm{up\_req}}(n_{seq(y)}).$$

Then, the sub-reliability requirement of the task $n_{seq(j)}$ should be

$$R_{\mathrm{req}}(n_{seq(j)}) = \frac{R_{\mathrm{req}}(G)}{\prod_{x=1}^{j-1} R(n_{seq(x)}) \times \prod_{y=j+1}^{|N|} R_{\mathrm{up\_req}}(n_{seq(y)})}. \quad (14)$$

(2) In satisfying the sub-reliability requirements of all tasks, we iteratively select available replicas and VMs that have the minimum execution time for each task to reduce its execution cost, rather than the minimum number of replicas, until its sub-reliability requirement is satisfied.

4.3 The QFEC Algorithm

On the basis of the aforementioned optimizations, we present the heuristic algorithm QFEC described in Algorithm 1 to reduce execution cost while satisfying the reliability requirement of the workflow.

Algorithm 1. The QFEC Algorithm

```
Input: G = (N, W, M, C), U, R_req(G)
Output: NR(G), cost(G), SL(G), R(G) and related values
1: Order tasks according to a descending order of
   rank_u(n_i, u_k) using Eq. (7);
2: for (j ← 1; j ⩽ |N|; j++) do
3:    Calculate R_req(n_seq(j)) using Eq. (14);
4:    num_seq(j) ← 0;
5:    R(n_seq(j)) ← 0; // initial value is 0
6:    for (k ← 1; k ⩽ |U|; k++) do
7:       Calculate R(n_seq(j), u_k) for the task n_seq(j) using Eq. (1);
8:       Calculate EFT(n_seq(j), u_k) for the task n_seq(j) using
         Eq. (9);
9:    end for
10:   while (R(n_seq(j)) < R_req(n_seq(j))) do
11:      Select available replica n^x_seq(j) and VM u_pr(n^x_seq(j)) with the
         minimum execution time w_seq(j),pr(n^x_seq(j));
12:      num_seq(j)++;
13:      Calculate AFT(n^x_seq(j)) ← EFT(n^x_seq(j), u_pr(n^x_seq(j))) using
         Eq. (10);
14:      Calculate R(n_seq(j)) using Eq. (4);
15:   end while
16: end for
17: Calculate NR(G) using Eq. (3);
18: Calculate cost(G) using Eq. (6);
19: Calculate SL(G) using Eq. (11);
20: Calculate R(G) using Eq. (5);
```

The main idea of QFEC is that the reliability requirement of the workflow is transferred to the sub-reliability requirement of each task. Then, QFEC simply iteratively selects available replicas and VMs with the minimum execution time for each task until its sub-reliability requirement is satisfied. The main steps are explained as follows:

(1) In Line 1, QFEC orders task based on a descending order of $rank_{\mathrm{u}}(n_i, u_k)$ using Eq. (7).

(2) In Lines 2-16, QFEC iteratively selects available replicas and VMs with the minimum execution time for each task until its sub-reliability requirement is satisfied. In particular, the sub-reliability requirement of each task is obtained in Line 3. Then, QFEC selects available replicas $n^x_{seq(j)}$ and VMs $u_{pr(n^x_{seq(j)})}$ with the minimum execution time $w_{seq(j),pr(n^x_{seq(j)})}$ in the iterative process in Line 11.

(3) In Lines 17-20, QFEC calculates the number of replicas $NR(G)$, execution cost $cost(G)$, schedule length $SL(G)$, and actual reliability value $R(G)$ of the workflow.

Compared with the RR algorithm [26], the main improvements of the presented QFEC algorithm are as follows:

(1) QFEC recalculates the sub-reliability requirement of each task based not only on its previous assignments ($\{n_{seq(1)}, n_{seq(2)}, \ldots, n_{seq(j-1)}\}$) but also on succeeding pre-assignments $\{n_{seq(j+1)}, n_{seq(j+2)}, \ldots, n_{seq(|N|)}\}$, whereas RR is merely based on previous assignments.

(2) QFEC iteratively selects available replicas and VMs with the minimum execution time to reduce its execution cost until its sub-reliability requirement is satisfied, whereas RR iteratively selects available replicas and VMs with the maximum reliability value to reduce the number of replicas, and thereby to reduce execution cost. A minimum number of replicas does not mean minimum execution cost and shortest schedule length in heterogeneous IaaS clouds.

The time complexity of the QFEC algorithm is analyzed as follows: All tasks should be traversed once, which can be conducted in O($|N|$) time. The number of replicas should be lower or equal to the number of VMs, which can be completed in O($|U|$) time. Calculating the AFT of each replica should be conducted in O($|N| \times |U|$) time. Thus, the time complexity of the QFEC algorithm is O($|N|^2 \times |U|^2$), which is similar to that of the RR algorithm. Thus, QFEC implements efficient fault-tolerance without increasing the time complexity.

Example 1. We assume that the constant failure rates for three VMs are $\lambda_1 = 0.001$, $\lambda_2 = 0.002$, and $\lambda_3 = 0.003$. We assume that the execution cost for three VMs are $\gamma_1 = 2$, $\gamma_2 = 1.5$, and $\gamma_3 = 1$. Moreover, we assume that the reliability requirement of the motivating workflow in Fig. 1 is $R_{\mathrm{seq}}(G) = 0.9$. Table 4 lists the replicas, selected VM, and reliability value of each task using the QFEC algorithm. Each row shows the selected VMs (denoted with boxed) and corresponding reliability values. For example, the sub-reliability requirement of n_1 is $R_{\mathrm{req}}(n_1) = \sqrt[10]{0.9} = 0.98951926$; QFEC selects the VMs u_3 and u_2 with the minimum and second minimum execution costs of 9 and 24, respectively, to satisfy the sub-reliability requirement. Then, the actual reliability value of n_1 is $R(n_1) = 0.99916105$, which is calculated by Eq. (4) and is larger than $R_{\mathrm{req}}(n_1) = 0.98951926$. The remaining tasks use the same pattern with n_1. Finally, the number of replicas is $NR(G) = 15$, the execution cost is $cost(G) = 240$, and the actual reliability value of the workflow G is $R(G) = 0.91295642$, which are calculated by Eqs. (3), (6), and (5), respectively.

TABLE 4
Task Assignment of the Motivating Workflow Using the QFEC Algorithm

n_i	$R_{\text{req}}(n_i)$	$w_{i,1}\times\gamma_1$	$w_{i,2}\times\gamma_2$	$w_{i,3}\times\gamma_3$	num_i	$R(n_i)$
n_1	0.98951926	28	$\boxed{24}$	$\boxed{9}$	2	0.99916105
n_3	0.97997050	22	$\boxed{19.5}$	$\boxed{19}$	2	0.99857801
n_4	0.97108055	26	$\boxed{12}$	17	1	0.98412732
n_2	0.97640101	$\boxed{26}$	28.5	$\boxed{18}$	2	0.99932104
n_5	0.97044553	24	19.5	$\boxed{10}$	1	0.97044553
n_6	0.98582658	26	$\boxed{24}$	$\boxed{9}$	2	0.99916105
n_9	0.97631346	36	$\boxed{18}$	$\boxed{20}$	2	0.99861899
n_7	0.96753856	14	22.5	$\boxed{11}$	1	0.96753856
n_8	0.98939492	$\boxed{10}$	16.5	14	1	0.99501248
n_{10}	0.97210312	42	$\boxed{10.5}$	16	1	0.98393272
$NR(G)=15$, $cost(G)=240$, $R(G)=0.90198016$						

4.4 The QFEC+ Algorithm

On one hand, although the QFEC algorithm can reduce execution cost by iteratively selecting available replicas and VMs with the minimum execution times until its sub-reliability requirement is satisfied, we find that such a process may still cause additional redundancy for some tasks. Given that minimum execution time does not mean maximum reliability value for a replica, we find that some redundant replicas for a task can be removed while still satisfying its sub-reliability requirement. For example, as shown in Table 4, when selecting replicas and VMs for n_5, QFEC first selects u_3 with minimum execution time 10 and then selects u_1 with second minimum execution time 12 to satisfy its sub-reliability requirement 0.98776561. However, if we merely select u_1 with execution time 12, then its actual sub-reliability value is 0.98807171, which can also satisfy its sub-reliability requirement 0.98776561. Such a fact reveals the necessity of filtering out partial QFEC-selected replicas and VMs by selecting the VM with the maximum reliability value to reduce redundancy. We consider the example that n_5 selects u_3 ($R(n_5,u_3)=0.97044553$) and u_1 ($R(n_5,u_1)=0.98807171$) using QFEC; then, u_3 can be removed. Therefore, in this study, we call the filter process as the QFEC+ algorithm.

On the other hand, although the QFEC+ algorithm can filter out partial replicas and VMs for a task n_i with less redundancy, the actual obtained reliability value for n_i is decreased. Given that the total reliability requirement of the workflow is fixed, such an operation may result in higher sub-reliability requirements for its succeeding tasks. Therefore, more replicas may be generated for succeeding tasks.

Considering the aforementioned possible contradictory results using QFEC+, we cannot determine which is superior between QFEC and QFEC+. Therefore, extensive experiments are needed (please refer to Section 6 for more experimental details on QFEC and QFEC+).

The description of the QFEC+ algorithm is shown in Algorithm 2 and its time complexity is also O($|N|^2\times|U|^2$), which is the same as that of the QFEC algorithm. That is, QFEC+ also does not increase time complexity.

The main idea of QFEC+ is described as follows: 1) similar to QFEC, QFEC+ first iteratively selects available replicas and VMs with the minimum execution times for each task until its sub-reliability requirement is satisfied; 2) QFEC+ reserves the selected VMs and clears the previous allocations of the task and then iteratively selects available replicas and VMs with the maximum reliability values for each task in the reserved VMs until its sub-reliability requirement is satisfied. The main steps are explained as follows:

Algorithm 2. The QFEC+ Algorithm

Input: $G=(N,W,M,C)$, U, $R_{\text{req}}(G)$
Output: $NR(G)$, $cost(G)$, $SL(G)$, $R(G)$ and related values
1: Order tasks according to a descending order of $rank_{\text{u}}(n_i,u_k)$ using Eq. (7);
2: **for** $(j\leftarrow 1; j\leqslant|N|; j{+}{+})$ **do**
3: Calculate $R_{\text{req}}(n_{seq(j)})$ using Eq. (14);
4: $R(n_{seq(j)})=0$; // initial value is 0
5: Define a list $replicas\_reliability\_list(n_{seq(j)})$ to store the replicas of $n_{seq(j)}$;
6: **for** $(k\leftarrow 1; k\leqslant|U|; k{+}{+})$ **do**
7: Calculate $R(n_{seq(j)},u_k)$ for the task $n_{seq(j)}$ using Eq. (1);
8: Calculate $EFT(n_{seq(j)},u_k)$ for the task $n_{seq(j)}$ using Eq. (9);
9: **end for**
10: **while** $(R(n_{seq(j)})<R_{\text{req}}(n_{seq(j)}))$ **do**
11: Select available replica $n^x_{seq(j)}$ and VM $u_{pr(n^x_{seq(j)})}$ with the minimum execution time $w_{seq(j),pr(n^x_{seq(j)})}$;
12: Put $n^x_{seq(j)}$ into the list $replicas\_reliability\_list(n_{seq(j)})$;
13: Calculate $R(n_{seq(j)})$ using Eq. (4);
14: **end while**
15: Sort the replicas in the list $replicas\_reliability\_list(n_{seq(j)})$ by descending order of reliability values of replicas.
16: Clear the previous allocations of n_i in Lines 10-14;
17: $num_{seq(j)}\leftarrow 0$;
18: $R(n_{seq(j)})\leftarrow 0$; // reset the reliability value of $n_{seq(j)}$ to 0;
19: **while** $(R(n_{seq(j)})<R_{\text{req}}(n_{seq(j)}))$ **do**
20: Select available replica $n^x_{seq(j)}$ and VM $u_{pr(n^x_{seq(j)})}$ with the maximum reliability value $R\left(n^x_{seq(j)},u_{pr(n^x_{seq(j)})}\right)$ in the list $replicas\_reliability\_list(n_{seq(j)})$;
21: Remove the replica $n^x_{seq(j)}$ from the list $replicas\_reliability\_list(n_{seq(j)})$;
22: $num_{seq(j)}{+}{+}$;
23: Calculate $AFT(n^x_{seq(j)})\leftarrow EFT(n^x_{seq(j)},u_{pr(n^x_{seq(j)})})$ using Eq. (10);
24: Calculate $R(n_{seq(j)})$ using Eq. (4);
25: **end while**
26: **end for**
27: Calculate $NR(G)$ using Eq. (3);
28: Calculate $cost(G)$ using Eq. (6);
29: Calculate $SL(G)$ using Eq. (11);
30: Calculate $R(G)$ using Eq. (5);

(1) In Line 1, similar to QFEC, QFEC+ orders tasks based on a descending order of $rank_{\text{u}}(n_i,u_k)$ using Eq. (7).
(2) In Lines 2-14, similar to QFEC, QFEC+ iteratively selects available replicas and VMs with the minimum execution times for each task until its sub-reliability requirement is satisfied.
(3) In Line 15, QFEC+ reserves the selected VMs and sorts the replicas in the list $replicas\_reliability\_list(n_{seq(j)})$ by descending order of reliability values of the replicas.

TABLE 5
Task Assignment of the Motivating Workflow Using the QFEC+ Algorithm

n_i	$R_{\mathrm{req}}(n_i)$	$w_{i,1} \times \gamma_1$	$w_{i,2} \times \gamma_2$	$w_{i,3} \times \gamma_3$	num_i	$R(n_i)$
n_1	0.98951926	28	$\boxed{24}$	$\boxed{9}$	2	0.99916105
n_3	0.97997050	22	$\boxed{19.5}$	$\boxed{19}$	2	0.99857801
n_4	0.97108055	26	$\boxed{12}$	17	1	0.98412732
n_2	0.97640101	$\boxed{26}$	28.5	$\boxed{\cancel{18}}$	1	0.98708414
n_5	0.97880978	24	$\boxed{19.5}$	$\boxed{10}$	2	0.99924149
n_6	0.96928634	26	24	$\boxed{9}$	1	0.97336124
n_9	0.98537671	36	$\boxed{18}$	$\boxed{20}$	2	0.99861899
n_7	0.97639765	$\boxed{14}$	22.5	$\boxed{\cancel{11}}$	1	0.99302444
n_8	0.97295116	$\boxed{10}$	16.5	14	1	0.99501248
n_{10}	0.96757973	42	$\boxed{10.5}$	16	1	0.98609754

$NR(G) = 14, cost(G) = 220.5, R(G) = 0.91722446$

(4) In Lines 16-18, QFEC+ clears the previous allocations (Lines 10-14) of n_i. The objective of the above two steps is to prepare reassignment for the replicas.

(5) In Lines 19-25, QFEC+ iteratively selects available replicas and VMs with the maximum reliability values for each task in the reserved VMs until its sub-reliability requirement is satisfied.

(6) In Lines 26-29, QFEC+ calculates the number of replicas $NR(G)$, execution cost $cost(G)$, schedule length $SL(G)$, and actual reliability value $R(G)$ of the workflow.

Example 2. The same parameter values ($\lambda_1 = 0.001$, $\lambda_2 = 0.002$, $\lambda_3 = 0.003$, $\gamma_1 = 2$, $\gamma_2 = 1.5$, $\gamma_3 = 1$, and $R_{\mathrm{seq}}(G) = 0.9$) as the aforementioned examples are used. Table 5 shows the task assignment for each task of the motivating workflow using the QFEC+ algorithm. Each row shows the selected VMs (in boxed), the removed VMs (in boxed strikeout), and the actual reliability value of the workflow. For example, when QFEC+ filters out u_3 with minimum execution cost 18 for n_2 and u_3 with minimum execution cost 11 for n_7, the sub-reliability requirements of n_5 and n_9 remain satisfied. Finally, the number of replicas is $NR(G) = 14$, the execution cost is $cost(G) = 220.5$, and the actual reliability value of the workflow G is $R(G) = 0.91722446$; these values are calculated by Eqs. (3), (6), and (5), respectively.

5 Quantitative Fault-Tolerance with Shortest Schedule Length

5.1 Problem Description

The problem of minimizing schedule length with reliability requirement can be formally described as follows: We assume that we are given a workflow G and a heterogeneous VM set U. The problem is to assign replicas and corresponding VMs for each task; at the same time, we must minimize the schedule length of the workflow and ensure that the obtained reliability value $R(G)$ satisfies the reliability requirement $R_{\mathrm{seq}}(G)$. The formal description is to find the replicas and VM assignments of all tasks to minimize schedule length

$$SL(G) = \max_{x \in [1, num_{\mathrm{exit}}]} (AFT(n^x_{\mathrm{exit}})),$$

subject to reliability requirement:

$$R(G) = \prod_{n_i \in N} (R(n_i)) \geqslant R_{\mathrm{req}}(G),$$

for $\forall i : 1 \leqslant i \leqslant |N|$.

5.2 The QFSL Algorithm

Iteratively selecting available replicas and VMs with the minimum execution times can achieve minimum execution cost using QFEC. Correspondingly, selecting available replicas and VMs with the minimum EFTs could achieve the shortest schedule length. Algorithm 3 describes the QFSL algorithm to minimize schedule length while satisfying the reliability requirement of the workflow.

Algorithm 3. The QFSL Algorithm

Input: $G = (N, W, M, C)$, U, $R_{\mathrm{req}}(G)$
Output: $NR(G)$, $cost(G)$, $SL(G)$, $R(G)$ and related values

1: Order tasks according to a descending order of $rank_{\mathrm{u}}(n_i, u_k)$ using Eq. (7);
2: **for** ($j \leftarrow 1; j \leqslant |N|; j$++) **do**
3: Calculate $R_{\mathrm{req}}(n_{seq(j)})$ using Eq. (14);
4: $num_{seq(j)} \leftarrow 0$;
5: $R(n_{seq(j)}) \leftarrow 0$; // initial value is 0
6: **for** ($k \leftarrow 1; k \leqslant |U|; k$++) **do**
7: Calculate $R(n_{seq(j)}, u_k)$ for the task $n_{seq(j)}$ using Eq. (1);
8: Calculate $EFT(n_{seq(j)}, u_k)$ for the task $n_{seq(j)}$ using Eq. (9);
9: **end for**
10: **while** ($R(n_{seq(j)}) < R_{\mathrm{req}}(n_{seq(j)})$) **do**
11: Select available replica $n^x_{seq(j)}$ and VM $u_{pr(n^x_{seq(j)})}$ with the minimum EFT;
12: $num_{seq(j)}$++;
13: Calculate $AFT(n^x_{seq(j)}) \leftarrow EFT(n^x_{seq(j)}, u_{pr(n^x_{seq(j)})})$ using Eq. (10);
14: Calculate $R(n_{seq(j)})$ using Eq. (4);
15: **end while**
16: **end for**
17: Calculate $NR(G)$ using Eq. (3);
18: Calculate $cost(G)$ using Eq. (6);
19: Calculate $SL(G)$ using Eq. (11);
20: Calculate $R(G)$ using Eq. (5);

Compared with Algorithm 1 and Algorithm 3, the sole change between QFEC and QFSL is that "Select available replica $n^x_{seq(j)}$ and VM $u_{pr(n^x_{seq(j)})}$ with the minimum execution time $w_{seq(j),pr(n^x_{seq(j)})}$" in Line 11 in QFEC is changed to "Select available replica $n^x_{seq(j)}$ and VM $u_{pr(n^x_{seq(j)})}$ with the minimum EFT $w_{seq(j),pr(n^x_{seq(j)})}$" in QFSL.

Example 3. The same parameter values ($\lambda_1 = 0.001$, $\lambda_2 = 0.002$, $\lambda_3 = 0.003$, and $R_{\mathrm{seq}}(G) = 0.9$) as the aforementioned examples are used. Table 6 shows the task assignment for each task of the motivating workflow using QFSL algorithm. Each row shows the selected VMs (in boxed) and actual reliability value of the workflow. QFSL iteratively selects available replicas and VMs with minimum EFTs. For example, the sub-reliability requirement of n_5 is $R_{\mathrm{req}}(n_5) = 0.98776561$; QFSL selects the VMs u_3 and u_2 with the minimum and second minimum EFTs,

TABLE 6
Task Assignment of the Motivating Workflow Using the QFSL Algorithm

n_i	$R_{\text{req}}(n_i)$	$EFT(n_i,u_1)$	$EFT(n_i,u_2)$	$EFT(n_i,u_3)$	num_i	$R(n_i)$
n_1	0.98951926	$\boxed{14}$	16	$\boxed{9}$	2	0.99962966
n_3	0.97951111	$\boxed{32}$	39	45	1	0.98906028
n_4	0.97996567	45	$\boxed{31}$	40	1	0.98412732
n_2	0.98533480	$\boxed{45}$	51	50	1	0.98708414
n_5	0.98776561	57	$\boxed{44}$	$\boxed{35}$	2	0.99924149
n_6	0.97775779	$\boxed{58}$	60	$\boxed{44}$	2	0.99965594
n_9	0.96784316	76	$\boxed{73}$	81	1	0.97628571
n_7	0.98096227	$\boxed{65}$	68	66	1	0.99302444
n_8	0.97749966	$\boxed{70}$	84	87	1	0.99501248
n_{10}	0.97210312	107	$\boxed{89}$	102	1	0.98609754
$NR(G)=13$, $cost(G)=131$, $R(G)=0.91295642$						

respectively, to satisfy the sub-reliability requirement. Finally, the number of replicas is $NR(G)=13$, the execution cost is $cost(G)=131$, and the actual reliability value of the workflow G is $R(G)=0.91295642$.

Fig. 2 also shows the scheduling of the motivating workflow G using QFSL, where the schedule length is $SL(G)=89$.

Similar to QFEC, QFSL can also be extended to QFSL+ with the same pattern. Considering space limitations, we do not provide the description of the QFSL+ algorithm in this study. Actually, the QFSL+ algorithm is similar to the QFEC+ algorithm, and the only difference between between QFSL+ and QFEC+ is that "Select available replica $n^x_{seq(j)}$ and VM $u_{pr(n^x_{seq(j)})}$ with minimum execution times $w_{seq(j),pr(n^x_{seq(j)})}$" in Line 11 in QFEC+ is changed to "Select available replica $n^x_{seq(j)}$ and VM $u_{pr(n^x_{seq(j)})}$ with minimum EFTs $EFT(seq(j), pr(n^x_{seq(j)}))$" in QFSL+.

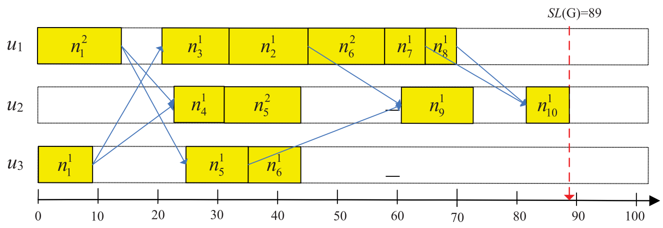


Fig. 2. Scheduling of the motivating workflow using QFSL.

Zhao implemented another version of FTSA by ordering tasks using $rank_{\text{u}}(n_i)$, and this version is called the FTSA(u) algorithm. The results show that FTSA(u) outperforms FTSA (u+d) in terms of schedule length [26]. Hence, similar to [26], we also use FTSA(u) for comparison in this study. Both MaxRe and RR study the same problem of quantitative fault-tolerance for reliable workflows. The metrics are final number of replicas, execution cost, and schedule length under the reliability requirement is satisfied.

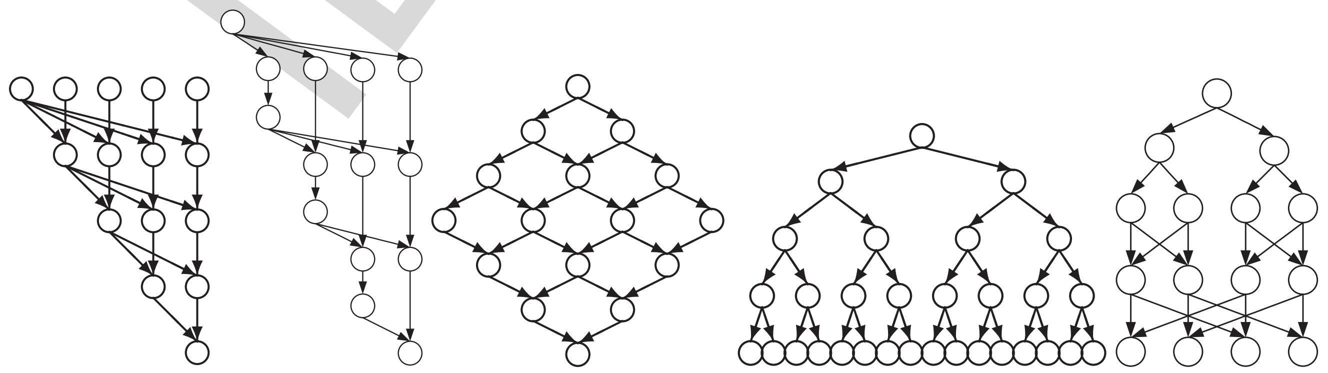
6 Experiments

6.1 Experimental Workflows and Metrics

We select fault-tolerant scheduling algorithm (FTSA) [21], MaxRe [25], and RR [26] for comparison in the experiments. FTSA is a heuristic bi-criteria approach to reduce the schedule length for a workflow in heterogeneous systems by using the active replication strategy to allocate ε+ 1 replicas of each task to ε+ 1 VMs. Note that the original FTSA orders tasks using $rank_{\text{u}}(n_i)+rank_{\text{d}}(n_i)$, and it is called FTSA(u+d) in [26].

Many cloud providers do provide the relevant information for their actual platforms, such as Amazon EC2 and Microsoft Azure et al. [8]. In this study, we use the relevant information of Amazon EC2 as test bed to do the experiments because it has been widely used in most works [3], [44], [45]. The simulated heterogeneous cloud platform contains of 64 VMs with different computing abilities and unit prices, where the prices of VMs are based on the Amazon EC2 [44]. As this study uses the VM specification of short term lease (i.e., pay-as-you-go), the prices for VMs are from \$0.095 to \$0.38 per hour [44]. In practice, the mean time between failures (MTBF, $1/\lambda$) is often reported instead of the failure rates to represent the reliability [44], [45]. The MTBF of each VM could belong to the scope of 100,000 h and 1,000,000 h. Therefore, the failure rates belongs to the scope of 10^{-7}/hour and 10^{-6}/hour. The execution time values of tasks and communication time values of messages could be the scope of: 1 h $\leq w_{i,k} \leq$ 128 h, 1 h $\leq c_{i,j} \leq$ 128 h [15].

We use five types of workflows, namely, linear algebra [46], Gaussian elimination [12], [35], diamond graph [46], complete binary tree [46], and fast Fourier transform [12], [35], to extensively validate the effectiveness of the proposed algorithms. These workflows are also used to compare the results of all the algorithms. Figs. 3a, 3b, 3c, 3d, and 3e show the examples of linear algebra with the size ρ=5 and the total number of tasks is $|N|=\rho(\rho+1)/2$, the Gaussian elimination with the size $\rho=5$ and the total number of tasks is $|N|=\frac{\rho^2+\rho-2}{2}$, the diamond graph with the size $\rho=4$ and



(a) Linear Algebra. (b) Gaussian elimination. (c) Diamond graph. (d) Complete binary tree. (e) Fast Fourier transform.

Fig. 3. Five different types of workflows.

TABLE 7
Average Schedule Lengths (Unit: h) of Workflows Using HEFT

Workflow	Task number	Average schedule lengths (unit:h)
Linear Algebra	2,556	6,483
Gaussian elimination	2,555	9,148
Diamond graph	2,601	9,542
Complete binary tree	2,047	1,086
Fast Fourier transform	2,559	1,544

the total number of tasks is $|N| = \rho^2$, the complete binary tree with the size $\rho = 5$ and the total number of tasks is $|N| = 2^\rho - 1$, the fast Fourier transform with the size $\rho = 4$ and the total number of tasks is $|N| = (2 \times \rho - 1) + \rho \times \log_2{}^{\rho}$, respectively.

Table 7 shows the average schedule lengths of these workflows with approximate equal task numbers using the standard HEFT algorithm. The schedule lengths of linear algebra, Gaussian elimination, and diamond graph (6,483-9,542) are larger than those of complete binary tree and fast Fourier transform (1,086-1,544) in the approximate equal scales. The results indicate that linear algebra, Gaussian elimination, and diamond graph are low-parallelism workflows, whereas complete binary tree and fast Fourier transform are high-parallelism workflows. The readers can refer to [47] with regard to the parallelism degree of a DAG-based workflow.

6.2 Low-Parallelism Workflows

Experiment 1. This experiment compares the total numbers of replicas, execution costs, and schedule lengths of large-scale low-parallelism workflows (including linear algebra, Gaussian elimination, and diamond graph). $R_{\rm req}(G)$ is changed from 0.91 to 0.99 with 0.02 increments.

Table 8 shows the results of linear algebra workflow with $\rho = 71$ and $|N| = 2556$ for varying reliability requirements. The total numbers of replicas, execution costs, and schedule lengths increase with the increase in reliability requirements using all the algorithms except for FTSA(u). That is, more resources are needed to satisfy higher reliability requirements. The following observations are drawn:

(1) In all cases, RR generates the minimum number of replicas followed by MaxRe, QFEC+, QFEC, QFSL+, QFSL, FTSA(u). The results verify that RR and MaxRe implement resource reduction by exploring less resource redundancy.

(2) In all cases, QFEC+ generates minimum execution costs followed by QFEC, QFSL+, QFSL, RR, MaxRe, and FTSA(u). QFEC and QFEC+ are used to reduce execution cost, and QFEC+ is slightly better than QFEC. The results indicate that QFEC+ is more effective in reducing execution cost than QFSL+, QFSL, QFEC, RR, MaxRe, and FTSA(u), for low-parallelism linear algebra workflows.

(3) In all cases, QFSL+ generates the shortest schedule length, followed by QFSL, QFEC (or QFEC+), RR, MaxRe, and FTSA(u). The results indicate that QFSL + is slightly better than QFSL in reducing schedule length and its advantages are obvious compared with QFEC, QFEC+, RR, and MaxRe, and FTSA(u).

TABLE 8
Results of Linear Algebra with $|N| = 2556$

Reliability requirement	Numbers of replicas						
	FTSA(u)	MaxRe	RR	QFEC	QFEC+	QFSL	QFSL+
0.91	5,114	2,610	**2,568**	3,020	2,837	3,498	3,243
0.93	5,114	2,830	**2,626**	3,286	3,074	3,720	3,447
0.95	5,114	3,419	**2,906**	3,601	3,382	3,926	3,719
0.97	5,114	4,692	**3,542**	3,974	3,761	4,246	4,060
0.99	5,114	5,114	**4,574**	4,525	4,439	4,674	4,576
Reliability requirement	Execution cost (unit: $)						
	FTSA(u)	MaxRe	RR	QFEC	QFEC+	QFSL	QFSL+
0.91	40,761	27,398	27,100	9,014	**8,559**	22,003	19,629
0.93	40,761	28,963	27,507	9,902	**9,373**	23,072	20,730
0.95	40,761	32,379	29,236	10,919	**10,391**	23,573	21,834
0.97	40,761	38,930	32,882	12,162	**11,670**	24,334	23,772
0.99	40,761	40,761	38,117	13,907	**13,747**	25,036	24,218
Reliability requirement	Schedule lengths (unit: h)						
	FTSA(u)	MaxRe	RR	QFEC	QFEC+	QFSL	QFSL+
0.91	39,041	25,443	29,063	59,257	8,559	22,003	**19,629**
0.93	39,041	27,079	25,101	61,173	60,888	6,159	**5,916**
0.95	39,041	30,777	26,063	65,418	63,881	6,288	**6,161**
0.97	39,041	39,318	26,210	65,414	65,765	6,470	**6,416**
0.99	39,041	39,041	35,744	69,911	66,429	6,670	**6,641**

(4) An obvious phenomenon is that the results produced by FTSA(u) do not change with the reliability requirements. This is because FTSA(u) is a heuristic bi-criteria approach and it does not need to comply with the reliability requirement.

Table 9 shows the results of Gaussian elimination with $\rho = 71$ and $|N| = 2{,}555$ for varying reliability requirements, similar to the results of Table 8 for linear algebra. Table 9 shows that QFEC+ and QFSL+ continue to generate the minimum execution costs and shortest schedule lengths, respectively. The results of the number of replicas and

TABLE 9
Results of Gaussian Elimination with $|N| = 2555$

Reliability requirement	Numbers of replicas						
	FTSA(u)	MaxRe	RR	QFEC	QFEC+	QFSL	QFSL+
0.91	5,110	3,126	2,702	2,624	**2,613**	3,330	3,185
0.93	5,110	3,295	2,735	2,731	**2,716**	3,475	3,340
0.95	5,110	3,532	3,054	2,911	**2,882**	3,700	3,566
0.97	5,110	3,966	3,411	3,208	**3,190**	3,987	3,873
0.99	5,110	4,777	4,225	3,861	**3,852**	4,430	4,362
Reliability requirement	Execution cost (unit: $)						
	FTSA(u)	MaxRe	RR	QFEC	QFEC+	QFSL	QFSL+
0.91	33,010	28,585	25,166	8,623	**8,482**	20,930	19,344
0.93	33,010	29,445	26,222	9,668	**9,547**	22,638	20,762
0.95	33,010	30,370	27,597	11,269	**11,003**	23,401	21,820
0.97	33,010	31,582	29,246	13,253	**13,182**	24,926	23,434
0.99	33,010	32,808	31,648	15,813	**15,797**	26,781	26,423
Reliability requirement	Schedule lengths (unit: h)						
	FTSA(u)	MaxRe	RR	QFEC	QFEC+	QFSL	QFSL+
0.91	26,477	21,976	18,610	12,798	12,769	9,833	**9,657**
0.93	26,477	23,327	19,751	12,832	12,823	9,936	**9,852**
0.95	26,477	24,230	20,956	12,741	12,870	10,504	**10,282**
0.97	26,477	25,188	23,062	13,098	13,147	10,834	**10,737**
0.99	26,477	26,313	25,184	13,687	13,622	11,311	**11,210**

TABLE 10
Results of Diamond Graph with $|N| = 2{,}601$

Reliability requirement	Numbers of replicas						
	FTSA(u)	MaxRe	RR	QFEC	QFEC+	QFSL	QFSL+
0.91	5,202	3,152	2,768	2,601	**2,600**	3,179	3,045
0.93	5,202	3,387	2,886	2,702	**2,696**	3,310	3,190
0.95	5,202	3,677	3,123	2,899	**2,895**	3,544	3,396
0.97	5,202	4,085	3,505	3,252	**3,244**	3,827	3,723
0.99	8,233	4,900	4,311	3,965	**3,952**	4,390	4,333
Reliability requirement	Execution cost (unit: $)						
	FTSA(u)	MaxRe	RR	QFEC	QFEC+	QFSL	QFSL+
0.91	33,370	38,799	33,053	11,222	**11,220**	19,747	17,738
0.93	33,370	40,687	34,792	12,496	**12,442**	19,939	18,386
0.95	33,370	42,318	37,214	14,626	**14,542**	20,732	19,222
0.97	33,370	43,886	40,029	17,350	**17,319**	21,780	20,791
0.99	33,370	45,434	43,733	20,537	**20,439**	22,955	22,823
Reliability requirement	Schedule lengths (unit: h)						
	FTSA(u)	MaxRe	RR	QFEC	QFEC+	QFSL	QFSL+
0.91	33,370	29,792	25,657	12,514	12,514	9,959	**9,785**
0.93	33,370	30,972	27,061	12,646	12,701	10,033	**9,892**
0.95	33,370	31,741	28,586	12,746	12,774	10,136	**10,256**
0.97	33,370	32,733	30,317	12,988	13,121	10,412	**10,401**
0.99	33,370	33,327	32,534	13,611	13,619	10,718	**10,710**

TABLE 11
Results of Complete Binary Trees with $|N| = 2{,}047$

Reliability requirement	Numbers of replicas						
	FTSA(u)	MaxRe	RR	QFEC	QFEC+	QFSL	QFSL+
0.91	4,096	2,303	2,084	2,048	**2,046**	2,469	2,388
0.93	4,096	2,443	2,178	2,094	**2,089**	2,632	2,524
0.95	4,096	2,594	2,332	2,188	**2,182**	2,778	2,643
0.97	4,096	2,921	2,576	2,387	**2,371**	2,975	2,867
0.99	4,096	3,678	3,177	2,948	**2,927**	3,412	3,309
Reliability requirement	Execution cost (unit: $)						
	FTSA(u)	MaxRe	RR	QFEC	QFEC+	QFSL	QFSL+
0.91	32270	24,411	20,507	9,301	**9,241**	15,606	14,652
0.93	32270	26,326	22,293	10,841	**10,763**	16,331	15,572
0.95	32270	27,860	24,604	13,329	**13,311**	17,466	17,183
0.97	32270	29,789	27,088	10,841	**10,763**	16,331	15,572
0.99	32270	31,893	30,249	13,329	**13,311**	17,466	17,183
Reliability requirement	Schedule lengths (unit: h)						
	FTSA(u)	MaxRe	RR	QFEC	QFEC+	QFSL	QFSL+
0.91	16,917	11,844	9,363	4,598	4,598	1,249	**1,201**
0.93	16,917	13,793	10,148	4,598	4,598	1,363	**1,338**
0.95	16,917	14,698	12,199	4,674	4,674	1,423	**1,386**
0.97	16,917	15,915	13,905	4,743	4,743	1,482	**1,461**
0.99	16,917	16,824	15,891	4,778	4,778	1,593	**1,574**

execution costs for Gaussian elimination using all the algorithms remain approximately equal to those that use linear algebra. The main difference is that Gaussian elimination has longer schedule lengths than linear algebra. Linear algebra has merely 67 percent of the schedule lengths of Gaussian elimination. Another main difference is that QFEC+ rather than RR generates the minimum numbers of replicas for Gaussian elimination.

Table 10 shows the results of the diamond graph with $\rho = 51$ and $|N| = 2{,}601$ for varying reliability requirements. The workflow illustrates a similar pattern as those Gaussian elimination workflows for all the algorithms in the approximate equal scale. That is, QFEC+, QFEC+, and QFSL+ still generate the minimum numbers of replicas, minimum execution costs, and shortest schedule length, respectively, for the diamond graph.

By combining the results of Tables 8, 9, and 10, we find that QFEC+ and QFSL+ can be used to minimize execution cost and schedule length, respectively, for low-parallelism workflows. Moreover, for the approximate equal scale and reliability requirement, all the workflows obtain approximately equal numbers of replicas and execution costs.

6.3 High-Parallelism Workflows

Experiment 2. This experiment compares the total numbers of replicas, execution costs, and schedule lengths of large-scale high-parallelism workflows (including complete binary tree and fast Fourier transform). $R_{\text{req}}(G)$ is also changed from 0.91 to 0.99 with 0.02 increments.

Table 11 shows the results of the complete binary tree with $\rho = 11$ and $|N| = 2{,}047$ for varying reliability requirements. Compared with low-parallelism workflows in Tables 8–10, RR and QFEC+ still generate the minimum numbers of replicas and minimum execution costs, respectively, for the complete binary tree workflow. Moreover, for the approximate equal scale and reliability requirement, the workflow also obtains the approximate equal numbers of replicas and execution costs to low-parallelism workflows. The results also show that QFSL+ generates shorter schedule lengths than QFSL for high-parallelism workflows.

Table 12 shows the results of the fast Fourier transform with $\rho = 256$ and $|N| = 2{,}559$ workflow for varying reliability requirements. Similar to the results for the complete binary tree, QFEC+, QFEC+, and QFSL+ still generate the minimum numbers of replicas, minimum execution costs, and shortest schedule length, respectively, for fast Fourier

TABLE 12
Results of Fast Fourier Transform with $|N| = 2{,}559$

Reliability requirement	Numbers of replicas						
	FTSA(u)	MaxRe	RR	QFEC	QFEC+	QFSL	QFSL+
0.91	5,118	2,660	2,660	2,559	**2,557**	3,127	2,993
0.93	5,118	3,157	2,757	2,578	**2,572**	3,268	3,107
0.95	5,118	3,365	2,956	2,748	**2,734**	3,439	3,321
0.97	5,118	3,799	3,296	3,044	**3,024**	3,750	3,602
0.99	5,118	4,709	4,124	3,762	**3,746**	4,287	4,132
Reliability requirement	Execution cost (unit: $)						
	FTSA(u)	MaxRe	RR	QFEC	QFEC+	QFSL	QFSL+
0.91	36,382	29,745	24,975	8,931	**8,921**	18,176	16,510
0.93	36,382	30,994	26,286	98,512	**96,425**	129,982	127,260
0.95	36,382	32,334	28,568	131,071	**128,487**	174,325	170,651
0.97	36,382	34,157	31,175	139,022	**136,357**	185,401	181,020
0.99	36,382	36,110	34,488	155,967	**153,182**	208,519	203,237
Reliability requirement	Schedule lengths (unit: h)						
	FTSA(u)	MaxRe	RR	QFEC	QFEC+	QFSL	QFSL+
0.91	13,898	10,782	9,413	6,419	6,419	1,621	**1,530**
0.93	13,898	11,443	9,649	6,442	6,442	1,595	**1,567**
0.95	13,898	12,034	10,388	6,619	6,509	1,663	**1,639**
0.97	13,898	12,983	11,471	6,752	6,744	1,757	**1,742**
0.99	13,898	13,756	13,064	7,138	7,183	1,865	**1,857**

TABLE 13
Percentages Using Different Algorithms

Workflow	Percentages of obtaining minimum numbers of replicas						
	FTSA(u)	MaxRe	RR	QFEC	QFEC+	QFSL	QFSL+
Linear Algebra	0%	0%	**100%**	0%	0%	0%	0%
Gaussian elimination	0%	0%	0%	**100%**	0%	0%	0%
Diamond graph	0%	0%	0%	**100%**	0%	0%	0%
Complete binary tree	0%	0%	0%	**100%**	0%	0%	0%
Fast Fourier transform	0%	0%	0%	**100%**	0%	0%	0%
Workflow	Percentages of obtaining minimum execution costs						
	FTSA(u)	MaxRe	RR	QFEC	QFEC+	QFSL	QFSL+
Linear Algebra	0%	0%	0%	0%	**100%**	0%	0%
Gaussian elimination	0%	0%	0%	13%	**87%**	0%	0%
Diamond graph	0%	0%	0%	9%	**91%**	0%	0%
Complete binary tree	0%	0%	0%	11%	**89%**	0%	0%
Fast Fourier transform	0%	0%	0%	1%	**99%**	0%	0%
Workflow	Percentages of obtaining shortest schedule lengths						
	FTSA(u)	MaxRe	RR	QFEC	QFEC+	QFSL	QFSL+
Linear Algebra	0%	0%	0%	0%	0%	2%	**99%**
Gaussian elimination	0%	0%	0%	0%	0%	4%	**96%**
Diamond graph	0%	0%	0%	0%	0%	2%	**98%**
Complete binary tree	0%	0%	0%	0%	8%	1%	**99%**
Fast Fourier transform	0%	0%	0%	0%	0%	1%	**99%**

transform. The results further indicate that QFSL+ is better than QFSL in reducing the schedule lengths for high-parallelism workflows.

6.4 Workflows Statistics

Experiment 3. This experiment shows the percentages using different algorithms that have minimum numbers of replicas, minimum execution costs, and shortest schedule lengths of workflows. $R_{\text{req}}(G)$ is generated randomly and belongs to the scope of 0.91 and 0.99.

Table 13 shows that QFEC+ and QFSL+ generate minimum execution costs and schedule lengths, respectively, for all high-parallelism and low-parallelism workflows in most cases. Such results further indicate that QFEC+ is better than QFEC in reducing execution cost regardless of the parallelism of workflows. In other words, we can determine that QFEC+ can generate minimum execution cost among the seven algorithms. For the percentages of shortest schedule lengths, we observed that QFSL+ can generate the shortest schedule lengths in most cases. That is, we can determine that QFSL+ can generate minimum schedule lengths among the seven algorithms.

6.5 Summary of Experiments

The following summarizations are made based on the aforementioned experimental results:

(1) Compared with the state-of-the-art algorithms, all the proposed algorithms achieve less execution costs and shorter schedule lengths, although the numbers of the replicas are not necessarily the smallest.

(2) QFEC and QFEC+ are designed to reduce execution cost, whereas QFSL and QFSL+ are designed to decrease schedule length.

(3) Whatever the workflow is high-parallelism or low-parallelism, QFEC+ is consistently better than QFEC in minimizing execution cost. Therefore, QFEC+ can be used for cloud services systems where economic cost is the main concern.

(4) Whatever the workflow is high-parallelism or low-parallelism, QFSL+ is consistently better than QFSL in minimizing schedule length. Therefore, QFSL+ can be used for high-performance cloud computing systems where execution time is the main concern.

7 Conclusion

We developed quantitative fault-tolerant scheduling algorithms QFEC and QFEC+ with minimum execution costs and QFSL and QFSL+ with shortest schedule lengths for a workflow in heterogeneous IaaS clouds. QFEC and QFSL iteratively select available replicas and VMs with the minimum execution times and minimum EFTs, respectively, for each task until its sub-reliability requirement is satisfied. QFEC+ and QFSL+ filter out partial QFEC-selected and QFSL-selected replicas and VMs for each task, respectively, by selecting available replicas and processors with the maximum reliability value until the sub-reliability requirement of the task is satisfied. Extensive experimental results show that QFEC+ is the best algorithm in reducing execution cost for both high-parallelism and low-parallelism workflows, whereas QFSL+ is the best algorithm in decreasing schedule length for both high-parallelism and low-parallelism workflows.

Acknowledgments

The authors would like to express their gratitude to the associate editor and three anonymous reviewers whose constructive comments have helped to improve the manuscript. This work was partially supported by the National Key Research and Development Plan of China under Grant No. 2016YFB0200405, the National Natural Science Foundation of China with Grant Nos. 61702172, 61672217, 61432005, 61379115, and 61502405, the CCF-Venustech Open Research Fund under Grant No. CCF-VenustechRP2017012, the China Postdoctoral Science Foundation under Grant No. 2016M592422, the Fundamental Research Funds for the Central Universities, the Natural Science Foundation of Fujian Province under Grant No. 2018J01571, the CERNET Innovation Project under Grant No. NGII20161003, and the Fujian Educational Bureau under Grant No. JA15368.

References

[1] R. Ranjan, L. Wang, A. Y. Zomaya, D. Georgakopoulos, X.-H. Sun, and G. Wang, "Recent advances in autonomic provisioning of big data applications on clouds," *IEEE Trans. Cloud Comput.*, vol. 3, no. 2, pp. 101–104, Apr. 2015.

[2] Y. Kong, M. Zhang, and D. Ye, "A belief propagation-based method for task allocation in open and dynamic cloud environments," *Knowl.-Based Syst.*, vol. 115, pp. 123–132, Jan. 2017.

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

[4] K. Keahey, I. Raicu, K. Chard, and B. Nicolae, "Guest editors introduction: Special issue on scientific cloud computing," *IEEE Trans. Cloud Comput.*, vol. 4, no. 1, pp. 4–5, Jan. 2016.

[5] G. Xie, G. Zeng, Y. Chen, Y. Bai, Z. Zhou, R. Li, and K. Li, "Minimizing redundancy to satisfy reliability requirement for a parallel application on heterogeneous service-oriented systems," *IEEE Trans. Serv. Comput.*, vol. PP, no. 99, p. 1, Feb. 2016, doi: 10.1109/TSC.2017.2665552.

[6] Q. Liu, W. Cai, J. Shen, Z. Fu, X. Liu, and N. Linge, "A speculative approach to spatial-temporal efficiency with multi-objective optimization in a heterogeneous cloud environment," *Security Commun. Netw.*, vol. 9, no. 17, pp. 4002–4012, Nov. 2016.

[7] C. Q. Wu, X. Lin, D. Yu, W. Xu, and L. Li, "End-to-end delay minimization for scientific workflows in clouds under budget constraint," *IEEE Trans. Cloud Comput.*, vol. 3, no. 2, pp. 169–181, Apr. 2015.

[8] J. Li, M. Qiu, Z. Ming, G. Quan, X. Qin, and Z. Gu, "Online optimization for scheduling preemptable tasks on IaaS cloud systems," *J. Parallel Distrib. Comput.*, vol. 72, no. 5, pp. 666–77, May 2012.

[9] W. Chen, R. F. da Silva, E. Deelman, and T. Fahringer, "Dynamic and fault-tolerant clustering for scientific workflows," *IEEE Trans. Cloud Comput.*, vol. 4, no. 1, pp. 49–62, Jan. 2016.

[10] A. Zhou, S Wang, B Cheng, Z Zheng, F. Yang, R. Chang, M. Lyu, and R. Buyya, "Cloud service reliability enhancement via virtual machine placement optimization," *IEEE Trans. Serv. Comput.*, vol. 10, no. 6, pp. 902–913, Dec. 2017.

[11] J. Mei, K. Li, X. Zhou, and K. Li, "Fault-tolerant dynamic rescheduling for heterogeneous computing systems," *J. Grid Comp.*, vol. 13, no. 4, pp. 507–525, Dec. 2015.

[12] C.-Y. Chen, "Task scheduling for maximizing performance and reliability considering fault recovery in heterogeneous distributed systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 2, pp. 521–532, Feb. 2016.

[13] G. Xie, Y. Chen, Y. Liu, Y. Wei, R. Li, and K. Li, "Resource consumption cost minimization of reliable parallel applications on heterogeneous embedded systems," *IEEE Trans. Ind. Informat.*, vol. 13, no. 4, pp. 1629–1640, Aug. 2017.

[14] X. Tang, K. Li, and G. Liao, "An effective reliability-driven technique of allocating tasks on heterogeneous cluster systems," *Cluster Comput.*, vol. 17, no. 4, pp. 1413–1425, Dec. 2014.

[15] W. Chen, G. Xie, R. Li, Y. Bai, C. Fan, and K. Li, "Efficient task scheduling for budget constrained parallel applications on heterogeneous cloud computing systems," *Future Generation Comput. Syst.*, vol. 74, pp. 1–11, Sep. 2017.

[16] F. Zhang, J. Cao, K. Hwang, K. Li, and S. U. Khan, "Adaptive workflow scheduling on cloud computing platforms with iterativeordinal optimization," *IEEE Trans. Cloud Comput.*, vol. 3, no. 2, pp. 156–168, Apr. 2015.

[17] J. D. Ullman, "Np-complete scheduling problems," *J. Comput. Syst. Sci.*, vol. 10, no. 3, pp. 384–393, Jun. 1975.

[18] X. Qin, H. Jiang, and D. R. Swanson, "An efficient fault-tolerant scheduling algorithm for real-time tasks with precedence constraints in heterogeneous systems," in *Proc. Int. Conf. Parallel Process.*, 2002, pp. 360–368.

[19] X. Qin and H. Jiang, "A novel fault-tolerant scheduling algorithm for precedence constrained tasks in real-time heterogeneous systems," *Parallel Comput.*, vol. 32, no. 5, pp. 331–356, Jan. 2006.

[20] Q. Zheng, B. Veeravalli, and C.-K. Tham, "On the design of fault-tolerant scheduling strategies using primary-backup approach for computational grids with low replication costs," *IEEE Trans. Comput.*, vol. 58, no. 3, pp. 380–393, Mar. 2009.

[21] A. Benoit, M. Hakem, and Y. Robert, "Fault tolerant scheduling of precedence task graphs on heterogeneous platforms," in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, 2008, pp. 1–8.

[22] S. Gopalakrishnan and M. Caccamo, "Task partitioning with replication upon heterogeneous multiprocessor systems," in *Proc. 12th IEEE Real-Time Embedded Technol. Appl. Symp.*, Apr. 2006, pp. 199–207.

[23] A. Benoit and M. Hakem, "Optimizing the latency of streaming applications under throughput and reliability constraints," in *Proc. Int. Conf. Parallel Process.*, 2009, pp. 325–332.

[24] N. Tabbaa, R. Entezari-Maleki, and A. Movaghar, "A fault tolerant scheduling algorithm for DAG applications in cluster environments," in *Digital Information Processing and Communications*. Brelin, Germany: Springer, 2011, pp. 189–199.

[25] L. Zhao, Y. Ren, Y. Xiang, and K. Sakurai, "Fault-tolerant scheduling with dynamic number of replicas in heterogeneous systems," in *Proc. 12th IEEE Int. Conf. High Perform. Comput. Commun.*, 2010, pp. 434–441.

[26] L. Zhao, Y. Ren, and K. Sakurai, "Reliable workflow scheduling with less resource redundancy," *Parallel Comput.*, vol. 39, no. 10, pp. 567–585, Oct. 2013.

[27] S. M. Shatz and J.-P. Wang, "Models and algorithms for reliability-oriented task-allocation in redundant distributed-computer systems," *IEEE Trans. Rel.*, vol. 38, no. 1, pp. 16–27, Apr. 1989.

[28] A. Benoit, L.-C. Canon, E. Jeannot, and Y. Robert, "Reliability of task graph schedules with transient and fail-stop failures: Complexity and algorithms," *J. Scheduling*, vol. 15, pp. 1–13, Oct. 2012.

[29] A. Benoit, F. Dufossé, A. Girault, and Y. Robert, "Reliability and performance optimization of pipelined real-time systems," *J. Parallel Distrib. Comput.*, vol. 73, no. 6, pp. 851–865, Jun. 2013.

[30] A. Dogan and F. Ozguner, "Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 308–323, Mar. 2002.

[31] A. Doğan and F. Özgüner, "Biobjective scheduling algorithms for execution time–reliability trade-off in heterogeneous computing systems," *Comput. J.*, vol. 48, no. 3, pp. 300–314, Jan. 2005.

[32] A. Girault and H. Kalla, "A novel bicriteria scheduling heuristics providing a guaranteed global system failure rate," *IEEE Trans. Dependable Secure Comput.*, vol. 6, no. 4, pp. 241–254, Oct. 2009.

[33] J. J. Dongarra, E. Jeannot, E. Saule, and Z. Shi, "Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems," in *Proc. 19th Annu. ACM Symp. Parallel Algorithms Archit.*, 2007, pp. 280–288.

[34] A. Benoit, M. Hakem, and Y. Robert, "Contention awareness and fault-tolerant scheduling for precedence constrained tasks in heterogeneous systems," *Parallel Comput.*, vol. 35, no. 2, pp. 83–108, Feb. 2009.

[35] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.

[36] M. A. Khan, "Scheduling for heterogeneous systems using constrained critical paths," *Parallel Comput.*, vol. 38, no. 4, pp. 175–193, May 2012.

[37] G. Xie, R. Li, and K. Li, "Heterogeneity-driven end-to-end synchronized scheduling for precedence constrained tasks and messages on networked embedded systems," *J. Parallel Distrib. Comput.*, vol. 83, pp. 1–12, Sep. 2015.

[38] A. Verma and N. Bhardwaj, "A review on routing information protocol (RIP) and open shortest path first (OSPF) routing protocol," *Int. J. Future Generation Commun. Netw.*, vol. 9, no. 4, pp. 161–170, Apr. 2016.

[39] Q. Zheng and 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.*, vol. 69, no. 3, pp. 282–294, Mar. 2009.

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

[41] H. Arabnejad and J. G. Barbosa, "A budget constrained scheduling algorithm for workflow applications," *J. Grid Comput.*, vol. 12, no. 4, pp. 665–679, Dec. 2014.

[42] H. Arabnejad, J. G. Barbosa, and R. Prodan, "Low-time complexity budget–deadline constrained workflow scheduling on heterogeneous resources," *Future Generation Comp. Syst.*, vol. 55, pp. 29–40, Feb. 2016.

[43] A. Girault, E. Saule, and D. Trystram, "Reliability versus performance for critical applications," *J. Parallel Distrib. Comput.*, vol. 69, no. 3, pp. 326–336, Mar. 2009.

[44] G. Koslovski, W.-L. Yeow, C. Westphal, T. T. Huu, J. Montagnat, and P. Vicat-Blanc, "Reliability support in virtual infrastructures," in *Proc. IEEE 2nd Int. Conf. Cloud Comput. Technol. Sci.*, 2010, pp. 49–58.

[45] A. C. Zhou, B. He, and C. Liu, "Monetary cost optimizations for hosting workflow-as-a-service in IaaS clouds," *IEEE Trans. Cloud Comput.*, vol. 4, no. 1, pp. 34–48, Jan. 2016.

[46] K. Li, "Scheduling precedence constrained tasks with reduced processor energy on multiprocessor computers," *IEEE Trans. Comput.*, vol. 61, no. 12, pp. 1668–1681, Dec. 2012.

[47] S. Bansal, P. Kumar, and K. Singh, "An improved duplication strategy for scheduling precedence constrained graphs in multiprocessor systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 14, no. 6, pp. 533–544, Jun. 2003.

Guoqi Xie received the PhD degree in computer science and engineering from Hunan University, China, in 2014. He is an associate professor of computer science and engineering with Hunan University. He was a postdoctoral researcher with the Nagoya University, Japan, from 2014 to 2015, and with the Hunan University, from 2015 to 2017. He has received the best paper award from ISPA 2016. His major interests include embedded and cyber-physical systems, parallel and distributed systems, software engineering and methodology. He is a member of the IEEE, ACM, and CCF.

Gang Zeng received the PhD degree in information science from the Chiba University, in 2006. He is an associate professor in the Graduate School of Engineering, Nagoya University. From 2006 to 2010, he was a researcher, and then assistant professor in the Center for Embedded Computing Systems (NCES), the Graduate School of Information Science, Nagoya University. His research interests mainly include power-aware computing and real-time embedded system design. He is a member of the IEEE and IPSJ.

Renfa Li is a professor of computer science and electronic engineering, with the Hunan University, China. He is the director of the Key Laboratory for Embedded and Network Computing of Hunan Province, China. He is also an expert committee member of National Supercomputing Center in Changsha, China. His major interests include computer architectures, embedded computing systems, cyber-physical systems, and Internet of things. He is a member of the council of CCF, a senior member of the IEEE, and ACM.

Keqin Li is a SUNY Distinguished Professor of computer science in the State University of New York. He is also a Distinguished Professor of Chinese National Recruitment Program of Global Experts (1000 Plan) at Hunan University, China. He was an Intellectual Ventures endowed visiting chair professor at the National Laboratory for Information Science and Technology, Tsinghua University, Beijing, China, during 2011-2014. His current research interests include parallel computing and high-performance computing, distributed computing, energy-efficient computing and communication, heterogeneous computing systems, cloud computing, big data computing, CPU-GPU hybrid and cooperative computing, multicore computing, storage and file systems, wireless communication networks, sensor networks, peer-to-peer file sharing systems, mobile computing, service computing, Internet of things and cyber-physical systems. He has published over 530 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He is currently or has served on the editorial boards of *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, *IEEE Transactions on Cloud Computing*, *IEEE Transactions on Services Computing*, and *IEEE Transactions on Sustainable Computing*. He is a Fellow of IEEE.