Using imbalance characteristic for fault-tolerant workflow scheduling in Cloud systems

Guangshun Yao, Yongsheng Ding, *Senior Member, IEEE*, Kuangrong Hao

***Abstract*—Resubmission and replication are two fundamental and widely recognized techniques in distributed computing systems for fault tolerance. The resubmission based strategy has an advantage in resource utilization, while the replication based strategy can reduce the task completed time in the context of fault. However, few researches take these two techniques together for fault-tolerant workflow scheduling, especially in Cloud systems. In this paper, we present a novel fault-tolerant workflow scheduling (ICFWS) algorithm for Cloud systems by combining the aforementioned two strategies together to play their respective advantages for fault tolerance while trying to meet the soft deadline of workflow. Firstly, it divides the soft deadline of workflow into multiple sub-deadlines for all tasks. Then, it selects a reasonable fault-tolerant strategy and reserves suitable resource for each task by taking the imbalance sub-deadlines among tasks and on-demand resource provisioning of Cloud systems into consideration. Finally, an online scheduling and reservation adjustment scheme is designed to select a suitable resource for the task with resubmission strategy and adjust the sub-deadlines as well as fault-tolerant strategies of some unexecuted tasks during the task execution process, respectively. The proposed algorithm is evaluated on both real-world and randomly generated workflows. The results demonstrate that the ICFWS outperforms some well-known approaches on corresponding metrics.**

***Index Terms*—fault-tolerant, workflow, Cloud systems, resubmission, replication, imbalance**

I. Introduction

Cloud computing has emerged as an attractive platform for diverse tasks, as it allows for low-entry costs, reduced cost of maintaining IT infrastructure and on demand heterogeneous resources provisioning in a pay-as-you-go model [1]. Empowered by the flexibility and elasticity of using computing resources, Cloud systems give customers the illusion of unlimited resources and have been widely adopted by increasing customers for their tasks, such as data storage [2,3] and scientific computing [4,5].

While the benefits are immense, the Cloud systems have a high resources failure probability due to the increased functionality and complexity [6,7]. Such faults can cause a devastating influence to the execution of submitted tasks, especially for the tasks with soft deadline, because the performance depends not only on the correctness of computation results, but also on the time instants at which these results become available [8]. Although the results are still useful if the soft deadline is not met, the Quality of Service (QoS) provided is greatly reduced. If a system regularly fails to meet soft deadlines by large amounts, customers will be dissatisfied with the offered QoS [9]. In the context of workflow with dependent relationship among tasks, it has become even more difficult since a delay in finishing one task caused by fault can propagate widely and then cause delays among many tasks on the other related resources. Therefore, providing an effective fault-tolerant mechanism is mandatory for the soft deadline-constrained workflow in Cloud systems.

Among the multiple fault-tolerant strategies, replication and resubmission are two fundamental and widely recognized techniques in distributed environments [9,10]. Replication submits the primary copy and some backup copies of the same task to different process units simultaneously to achieve fault tolerance. Resubmission tries to find another suitable process unit to reexecute the task after a fault happened. Moreover, resubmission is mostly applicable during execution process and can strength the resource utilization of systems, while replication is a method suited to the task scheduling phase and has an advantage in saving the task execution time. Based on replication or resubmission, many algorithms have been proposed to design fault-tolerant strategy in distributed systems in last decades [7-9,15-18,20-24]. However, few of them try to combine the above techniques together to play their respective advantages for fault-tolerant workflow scheduling. Thus, the proposed resubmission and replication based strategies usually spend lots of time and resources, respectively, to complete the submitted workflows. For the soft deadline-constrained workflow in Cloud systems, taking fewer resources to complete more tasks is valuable for both users and resource providers as the users can reduce their cost for the submitted workflow while the resource providers can offer service to more users under the same resources and thus to increase their revenues.

In this paper, we propose a novel fault-tolerant workflow scheduling algorithm, called ICFWS, by combining replication and resubmission together to play their respective advantages for fault tolerance while trying to meet the soft deadline of workflow in Cloud systems. It divides the whole soft deadline of workflow into multiple sub-deadlines for all tasks. Based on

This work was supported in part by the Key Project of the National Natural Science Foundation of China (no. 61134009), the National Natural Science Foundation of China (nos. 61473078, 61473077), Program for Changjiang Scholars from the Ministry of Education (2015-2019), and International Collaborative Project of the Shanghai Committee of Science and Technology (no. 16510711100).

All authors are with the Engineering Research Center of Digitized Textile and Apparel Technology, Ministry of Education, College of Information Science and Technology, Donghua University, Shanghai 201620, China. Guangshun Yao is also with the College of Computer and Information Engineering, Chuzhou University, Chuzhou 239000, Anhui, China. (corresponding author: Prof. Y.S. Ding: ysding@dhu.edu.cn).

the assigned sub-deadline, each task selects a corresponding fault-tolerant strategy from the aforementioned two strategies and reserve suitable resource through on-demand resource provisioning model of Cloud systems. For the task with resubmission strategy, it also adopts online rescheduling for reexecution when it encounters a fault during its previous execution process. Furthermore, in order to make full advantage of time slot, it also takes an online reservation adjustment scheme to adjust the sub-deadlines of some unexecuted tasks and then adjust their fault-tolerant strategies during the execution process. The ICFWS can be immediately applied in any Cloud computing platform, even if no historic trace data to build a platform-specific fault model is available.

The main contributions of this paper are as follows. Firstly, a deadline division scheme is designed to divide the soft deadline of workflow into multiple sub-deadlines for all tasks. Secondly, a novel fault-tolerant workflow scheduling algorithm called ICFWS is proposed to combine resubmission and replication together to play their respective advantages in the context of fault. Thirdly, an online reservation adjustment scheme is presented to adjust the sub-deadlines and fault-tolerant strategies of some unexecuted tasks during the task execution process. Finally, an on-demand resource provisioning strategy is designed during the resource reserving process in the context of combining the above two fault-tolerant strategies together.

The rest of this paper is organized as follows. The related work is discussed in Section II. In Section III, we describe the Cloud systems and fault model used in this paper. After introducing the used workflow model, the ICFWS algorithm is presented in Section IV. In Section V, we evaluate the proposed algorithm with some other competitors and analyze the obtained results. Finally, we present the main conclusions and future work in Section VI.

II. Related work

In last decades, lots of fault-tolerant algorithms have been proposed to decrease the adverse impact caused by fault in distributed systems from different perspectives. Javadi et al. [11] started to investigate how to manage and predict failures in complex infrastructures for the fault in these systems. However, building the model used in this work can be a very difficult task that often requires the traces of failure data about the specific target environment. Jhawar et al. [12] advocated a new dimension where applications deployed in Cloud systems can obtain required fault tolerance properties from a third party. Nevertheless, selecting the suitable third party is difficult and impractical for normal users. Zheng [13] and Qiu [14] proposed a ranking-based method in which all components in Cloud systems were ranked according to their invocation structures and invocation frequencies. Based on the ranking results, an optimal algorithm was derived to determine the fault tolerance strategies for different components. However, the precise ranking is very hard to achieve and requires not only a deep knowledge of the behavior of the target infrastructure, but also years of trace data of the specific system.

Besides the aforementioned fault-tolerant mechanisms, resubmission, which tries to submit and execute the task again on the same or another process unit after a failure, is a widely used strategy for fault-tolerant workflow scheduling in distributed systems. Inspired by the immunological mechanism, Yao et al. [7] proposed a novel rescheduling algorithm called IRW for workflow in Cloud systems. It was consisted by the surveillance unit, the response unit, the learning unit and the memory unit. All these units worked cooperatively to imitate the immune system for resubmission and rescheduling. Chen et al. [15] proposed an efficient resubmission heuristic with the support of reservation adjustment to alleviate the delay caused by resource contention for multiple workflows submitted at different time. Olteanu et al. [16] designed a rescheduling algorithm by combining a wide variety of scheduling heuristics, including retry, alternate resource, rescue file and user-defined exception handing, together for workflow scheduling. Cao et al. [17] designed three schemes for tasks rescheduling in Cloud systems. After detecting a virtual machine (VM) was crashed, these schemes firstly stored the executing tasks on this VM and waited for a certain time to check whether the VM could be repaired. If it could not be repaired, the stored tasks were resubmitted to another VM. Sakellariou et al. [18] considered resubmission at some carefully selected points along execution. After the initial schedule is obtained, if the run time performance variation exceeded a predefined threshold, it selected a set of unfinished tasks for resubmission. Although the above resubmission based algorithms can provide corresponding fault-tolerant strategy for workflow and have advantage about resource utilization in the context of fault tolerance, the overall completion time of workflow is significantly delayed. Moreover, the soft deadline of workflow is not considered in the above resubmission based algorithms.

Besides resubmission, replication, which generates one or many backup copies for one task besides the primary copy and allocates these copies to different process units, is another fundamental technique for fault in distributed systems [19]. If the primary copy of one task is suspended by the failure of process unit, the backup copy can still be executed on a different unit to guarantee the successful completion of the task. Based on replication, Qin and Jiang [8] proposed a new overlapping scheme, which allowed the backup copy of a task to overlap with the primary copies of its successors for deadline constrained workflow. After identifying two crucial limitations of scheduling backups for dependent tasks in Grid systems, Zheng et al. [20] proposed two fault-tolerant scheduling algorithms, namely MRC-ECT and MCT-LRC, to schedule backups of independent and dependent tasks, respectively. For the Grid systems with dedicated communication devices, Zheng and Veeravalli [21] designed two communication-aware fault-tolerant workflow scheduling algorithms based on replication to avoid the influence caused by the processors faults and communication delays. However, the above replication based fault-tolerant algorithms are designed for Grid systems and cannot be used in Cloud systems directly. Recently, Jing and Liu [22] designed a replication based fault-tolerant scheduling algorithm called CCRH for workflow in Cloud systems. However, the on-demand resource provisioning model of Cloud systems is not considered.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2687923, IEEE Transactions on Parallel and Distributed Systems

Although the above replication based algorithms can significantly improve the system fault-tolerant capacity and the workflow execution time under these algorithms is also better than that under resubmission, they have two common drawbacks: (1) the deadline of workflow is also not considered except [8], (2) the resource consumption is deteriorated caused by replication.

Different from the above researches adopted a single fault-tolerant strategy, Plankensteiner et al. [9] took both replication and resubmission simultaneously for each task in the workflow and used the impact of resubmission of each task to adjust the replication size of this task to balance resubmission and replication. However, the characteristics about resource provision in Cloud systems are not considered in this work. Recently, Jayadivya [23] and Patra [24] proposed FTWS and RRADFTRC, respectively, for fault-tolerant workflow scheduling in Cloud systems by taking resubmission and replication simultaneously for each task in the workflow. Both FTWS and RRADFTRC are implemented based on the provided maximum replication and resubmission count from users. The difference is that the FTWS adopted the impact of resubmission, out degree and deadline of each task to adjust the replication size of this task while the RRADFTRC did not. When a task encountered a fault, both FTWS and RRADFTRC took the designed backup copies at first. If all of these backups were not completed successfully, the resubmission was adopted. In other words, the resubmission used was only passive adopted when all backup copies of this task had failed in FTWS and RRADFTRC. As a result, more resources were needed by the backups and the advantage of resubmission was missed. Different from the above algorithms, the proposed ICFWS in this paper does not need any parameters from users and can be immediately applied in any Cloud computing platform. Moreover, each task in ICFWS only chooses one fault-tolerant strategy from replication and resubmission at any instant and the selected fault-tolerant strategy can be changed during the execution process if necessary. Furthermore, in the proposed ICFWS, if one task selects resubmission as its fault-tolerant strategy and encounters a fault during its execution process, the resubmission can be adopted immediately without any waiting.

III. Cloud Systems and Fault Model

In this section, we describe the models used in this paper, including the Cloud systems model and fault model.

A. Cloud systems model

Through virtualization technology, each host in Cloud systems can be virtualized to a set of heterogeneous VMs. So the VM is the basic processor unit rather than host. In our model, the Cloud systems offer a set of M types of virtualized resources in the form of VMs $VM = \{vmt_1, vmt_2, \cdots, vmt_M\}$ to customers in a pay-as-you-go model. For example, Amazon EC2 provides five types of computing optimized virtualized resources as shown in Table I. All VMs with the same type vmt_m have the identical processing capacity vmp_m. Furthermore, all VMs are assumed to be located in the same data center or region so that the average bandwidth between VMs is rough equal [4,5]. Empowered by the virtualization technology, an infinite amount of resources can be accessed in Clouds and there is no limitation on the number of VMs leased by an application. However, it is worth to note that not all the VMs are active. The VMs can be dynamically switched between the active status and the sleep one according to the system workload. When a VM is leased, it requires an initial boot time $delay$ for its proper initialization. So the $delay$ should be considered when designing the scheduling strategy. The reserved VM is only active during its leased time interval. After that, the VM is shut down immediately to improve the energy consumption and resource utilization.

The Cloud systems offer services to customers through Internet via the Scheduler, which accepts and schedules workflow submitted from customer. The Scheduler architecture used in this paper as shown in Fig. 1 is an improved model for workflow based on the architecture proposed in [7,25].

The Scheduler consists of a Workflow Analyzer, a Resubmission Controller, a Replication Controller, and a Resource Manager. When a workflow arrives, the Scheduler will determine whether accept the submitted workflow by the cooperation of Workflow Analyzer and Resource Manager at first. If the Scheduler cannot find a feasible scheduling strategy for the submitted workflow under the given QoS, such as deadline, it will reject the workflow.

For the accepted workflow, the Workflow Analyzer will analyze the relationship among tasks and then divide the whole soft deadline of workflow into multiple sub-deadlines for all tasks in this workflow. Then, the Scheduler will determine the selection of fault-tolerant strategy for each task from resubmission and replication according to the imbalance sub-deadlines of tasks and performance of Cloud resources. The task with enough time to execute again for the fault will select resubmission for fault tolerance. Otherwise, it will select replication. In this way, resubmission and replication are comb-

TABLE I. Compting optimized VMs in Amazon EC2

	CPU	Processing capacity (MFLOPS)	Memory (GB)	Storage (GB)
c3.large	2	8800	3.75	2×16 SSD
c3.xlarge	4	17600	7.5	2×40 SSD
c3.2xlarge	8	35200	15	2×80 SSD
c3.4xlarge	16	70400	30	2×160 SSD
c3.8xlarge	32	140800	60	2×320 SSD

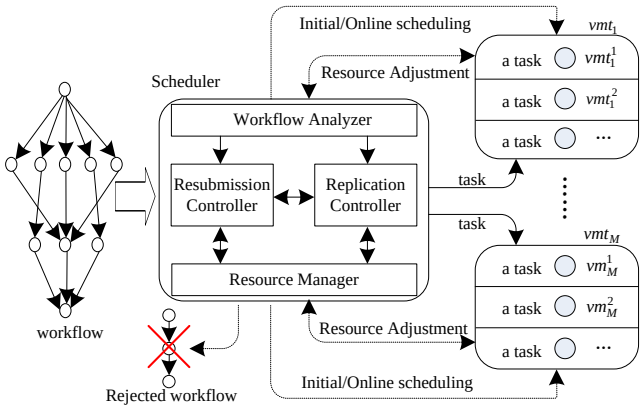


Fig. 1. Scheduling architecture for Cloud systems model

ined together and each strategy plays its respective advantage for fault tolerance. Then, through the cooperation of Resubmission Controller and Replication Controller, the Scheduler adopts an initial scheduling and online scheduling before and during the task execution process, respectively. In order to make the full advantage of time slot, the Scheduler also adopts an online reservation adjustment scheme to adjust the sub-deadlines and fault-tolerant strategies of some other unexecuted tasks if possible. Moreover, on-demand resource provisioning is adopted by Scheduler to select suitable VMs for tasks in both initial scheduling and online scheduling. The active resources are adjusted dynamically by Resource Manager according to the system workload.

In this work, the resubmission is performed immediately when the execution process encounters a fault for the task with resubmission strategy, and the primary and backup copy are executed on two different VMs simultaneously for the task with replication strategy. Through virtualization technology and on-demand resource provisioning, the proposed ICFWS can be extended easily to large systems.

B. Fault model

It is supposed that there exists a fault-detection mechanism, such as fail-signal and acceptance test, to detect failures [26]. Tasks will fail when the processor units where they are located fail. In this work, we focus on the failure of VM, which is the basic processor unit in Cloud systems. The faults can be transient or permanent and are assumed to be independent, affecting only a single VM. We also assume the minimum required value of the mean time to failure (MTTF) is always greater than or equal to the maximum task execution time [20].

IV. Algorithm Implementation

In this section, we elaborate the proposed ICFWS. In order to make it clearly, the workflow application model is introduced at first and the major symbols and notations used throughout of this paper are summarized in Table II.

A. Workflow application model

In this paper, we use the Directed Acyclic Graph (DAG) to represent the workflow submitted from customer. A $DAG = (T, E)$ consists of R tasks $T = \{t_1, t_2, \cdots, t_R\}$, which are interconnected through data and control flow such as

$$E = \{(t_i, t_j, e_{ij}, contr_{ij}) \big| (t_i, t_j) \in T \times T, i \neq j\},$$

where e_{ij} represents the size of data that needs to be transferred from t_i to t_j and $contr_{ij}$ denotes the control relationship between t_i and t_j. Because the bandwidth among VMs is rough equal, we also use e_{ij} to represent the data transfer time from t_i to t_j. The size of each task t_i is measured by Million of Instructions MI_i, so the execution times of tasks in the accepted workflow can be denoted as

$$et_i^m = \begin{vmatrix} \frac{MI_1}{vmp_1} & \cdots & \frac{MI_1}{vmp_M} \\ \vdots & \ddots & \vdots \\ \frac{MI_R}{vmp_1} & \cdots & \frac{MI_R}{vmp_M} \end{vmatrix} \quad (1 \le i \le R, 1 \le m \le M),$$

where et_i^m is the execution time of t_i on the m-th type VM vmt_m with the processing capacity vmp_m. The set of predecessors and successors of task t_i are denoted as $pred(t_i)$ and $succ(t_i)$, respectively. In a given DAG, a task without any predecessor is called the entry task t_{entry} and a task without any successor is called the exit task t_{exit}. In this paper, we always add two dummy tasks t_{entry} and t_{exit} to the beginning and the end of the graph, respectively, to ensure the given DAG has a single entry and a single exit task. The execution times of these dummy tasks are zero and they are connected with zero-weight arcs to the real entry and exit tasks, respectively. In addition, each workflow has a soft deadline $DeadL$, which is given by customer.

TABLE II. Major symbols and notations in ICFWS

Symbol	Definition
vmt_m	The VM of type m, $1 \le m \le M$
vmp_m	The processing capacity of vmt_m
$delay$	The initial boot time of a created new VM
e_{ij}	The data transfer time from t_i to t_j
MI_i	The size of task t_i
et_i^m	The execution time of t_i on vmt_m
$pred(t_i)$	The predecessor set of task t_i
$succ(t_i)$	The successor set of task t_i
$EST(t_i)$	The earliest start time of task t_i
$MEST(t_i)$	The modified earliest start time of task t_i
$MET(t_i)$	The minimum execution time of task t_i
$AET(t_i)$	The average execution time of t_i among all VM types
$sdl(t_i)$	The sub-deadline of task t_i
$reex(t_i)$	The minimum time for reexecution of task t_i
$fst(t_i)$	The selected fault-tolerant strategy of task t_i
$pft(t_i)$	The planed finish time of task t_i
$aft(t_i)$	The actual finish time of task t_i
$CPr(t_i)$	The critical predecessor of task t_i
$Active\_VMs$	The set of already reserved VMs
rvm_m^n	The n-th reserved VM with type vmt_m
$[rvm_m^n, rst, rft]$	The reserved start time rst and reserved finish time rft of rvm_m^n
$fault(t_i)$	The time of detecting the first fault of task t_i
$fault'(t_i)$	The time of detecting another fault (not the first fault) of task t_i

In this work, we also suppose that each task cannot be interrupted during execution and must be finished except fault. It cannot be divided further for parallel processing, and thus must be scheduled in its entirety on a process unit.

B. The scheme of the ICFWS algorithm

The proposed ICFWS has three main phases: Deadline Division, Initial Scheduling as well as Online Scheduling and Reservation Adjustment. The Deadline Division is used to divide the soft deadline $DeadL$ into multiple sub-deadlines for all tasks. Based on the assigned sub-deadline and available resources, the Initial Scheduling is used to select the fault-tolerant strategy for each task from replication and resubmission and schedule all tasks for their first execution as well as the backup copies of the tasks with replication strategy. Thus, the tasks with replication strategy can be completed successfully even under fault. As for the task with resubmission,

if it encounters a fault during its execution process, the Online Scheduling scheme is used to select suitable VM for executing it again. In this way, the replication and resubmission can be combined together for fault tolerance and the workflow can be completed under the constrained soft deadline. In order to make full advantage of time slot, the Online Reservation Adjustment is used to adjust the sub-deadlines and fault-tolerant strategies of some unexecuted tasks when all of their predecessors have been completed before their sub-deadlines.

The main pseudo code of the proposed ICFWS is shown in Algorithm1. After receiving the submitted workflow, the scheduler takes a simulated scheduling by Heterogeneous Earliest-Finish-Time (HEFT) [27] and then gets the makespan ($makespan_{HEFT}$) in an ideal state before real scheduling (line 1). For a given workflow with deadline $DeadL$, if $makespan_{HEFT}$ is greater than $DeadL$, the workflow cannot be completed by Cloud systems and therefore is returned to customer for revising the deadline (lines 2-3). Otherwise, it is accepted. For the accepted workflow, the scheduler assigns the sub-deadline to t_{entry} and t_{exit} as zero and $DeadL$, respectively (line 6). Then, the Deadline Division is executed to divide the whole deadline of workflow into multiple sub-deadlines for all tasks (line 7). Based on these sub-deadlines and available resources, the Initial Scheduling is adopted before task execution to determine the fault-tolerant strategy and reserve corresponding resource for each task (line 8). During the task execution process, the Online Scheduling is used for task with resubmission strategy and the Reservation Adjustment is designed for adjusting sub-deadlines, fault-tolerant strategies and reserved resources for some unexecuted tasks (line 10). In the following sections, we elaborate on the details of these sub-algorithms.

C. Deadline Division

It is obviously that if the deadline of workflow can be divided into multiple sub-deadlines for all tasks and each task can be completed in/near its individual sub-deadline, then the whole workflow will be finished within/near its deadline. In this paper, the workflow deadline is distributed to each task based on a critical path (CP) heuristic. The CP heuristics are widely applied in workflow scheduling and the CP of a workflow is the longest execution path between the entry and the exit tasks of the workflow [28,29]. In this section, the whole deadline of workflow is divided to multiple sub-deadlines for all tasks in the CP based on the task execution time and communication time among tasks at first. In this way, each task in the critical path has been assigned a sub-deadline. Then, by taking two tasks in CP as an entry node and an exit node in turn, the local CP and sub-deadlines of tasks in local CP can be computed under the same procedure. The procedure continues recursively until all tasks have been assigned sub-deadlines. To facilitate the description, some concepts are introduced as follows.

Algorithm1: ICFWS()

```
1   schedule the DAG by HEFT in ideal state and get makespan_HEFT;
2   if (DeadL < makespan_HEFT)
3       prompt user to specify a deadline above makespan_HEFT ;
4   else
5       // before task execution process
6       sdl(t_entry) ← 0; sdl(t_exit) ← DeadL;
7       call DeadDivi(t_entry, t_exit);
8       call IniScheduling();
9       // during task execution process
10      call OnlSchResAdj()
```

Similar to [29,30], the Earliest Start Time of task t_i, $EST(t_i)$, is defined as follow:

$$EST(t_i) = \begin{cases} 0, & if\ t_i = t_{entry} \\ min_{t_p \in pred(t_i)} EST(t_p) + MET(t_p) + e_{p,i}, & otherwise \end{cases}, \quad (1)$$

where $MET(t_i)$, the Minimum Execution Time of task t_i, is the execution time of task t_i on one type of VM which has the minimum et_i^m among all VM types. Note that both $MET(t_{entry})$ and $MET(t_{exit})$ are equal to zero if t_{entry} and t_{exit} are two dummy tasks.

Definition 1: The Critical Predecessor of a task t_i,$CPr(t_i)$, is the predecessor of t_i that has the latest data arrival time at t_i, that is, it is the predecessor t_p of t_i for which $EST(t_p) + MET(t_p) + e_{p,i}$ is maximal.

Having this definition, we can get the critical path, CP, from t_{exit} to t_{entry} in a recursive manner. Then, the overall deadline of workflow can be distributed to each task in CP according to the strategy presented as follows.

For a serial and deadline constrained path, $P = \{t_1, t_2, \cdots, t_k\}$, if the sub-deadlines of t_1 and t_k are known as $sdl(t_1)$ and $sdl(t_k)$, respectively. Then, we can distribute the path deadline to all tasks of this path in proportion to their average execution time and communication time. That is

$$sdl(t_i) = (sdl(t_k) - sdl(t_1)) \times \frac{\max_{t_p \in pred(t_i)} e_{p,i} + AET(t_i)}{\sum_{i=1}^{k} (\max_{t_p \in pred(t_i)} e_{p,i} + AET(t_i))}. \quad (2)$$

where $AET(t_i)$ is the average execution time of t_i among all VM types.

Then, by setting $sdl(t_{entry})$ and $sdl(t_{exit})$ to zero and $DeadL$, respectively, the overall deadline of workflow can be distributed to each task in CP based on the task execution time and communication time among tasks according to Eq.(2).

After that, by taking two tasks in CP as the entry task and exit task in turn, a local workflow can be formed. Then, the local CP and sub-deadline of each task, which belongs to this local CP, can be calculated under the same procedure. The procedure continues recursively until all tasks have been assigned a sub-deadline. Moreover, the sub-deadlines are imbalance among tasks caused by the uneven size of tasks and transferred times between tasks.

The pseudo code for the Deadline Division is presented in Algorithm2. The first execution of the Deadline Division scheme is called by t_{entry} and t_{exit} of the accepted workflow as shown in Algorithm1. Then, it gets the CP of workflow by finding CPr from t_{exit} to t_{entry} (lines 2-5). After that, each task in CP is assigned a sub-deadline according to Eq.(2) (lines 6-7). Then, the same procedure is adopted by taking two nodes

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2687923, IEEE Transactions on Parallel and Distributed Systems

Algorithm2 DeadDivi (t_{entry}, t_{exit})

Input: a DAG $V = (T, E)$ with a soft deadline $DeadL$; the VM type $vmt_m (1 \le m \le M)$;
Output: sub-deadline for each task;

```
1  CP ← NULL, t_i ← t_exit;
2  while (pred(t_i) ≠ t_entry)
3      select CPr(t_i) according to Definition 1;
4      add CPr(t_i) to the beginning of CP;
5      t_i ← CPr(t_i);
6  for (i = 1 to |CP|)
7      calculate sdl(t_i) according to Eq.(2);
8  for (i = 1 to |CP − 1|)
9      for (j = 2 to |CP|)
10         t_entry ← t_i, t_exit ← t_j;
11         sdl(t_entry) ← sdl(t_i), sdl(t_exit) ← sdl(t_j);
12         Call DeadDivi(t_entry, t_exit);
13 update the Earliest Start Time of each task;
```

in CP as the entry node and exit node in turn to assign one sub-deadline to each task in the local CP. The procedure is executed in a recursive manner until each tasks has been assigned a sub-deadline (lines 8-12). Through the above procedure, each task has assigned an individual sub-deadline and the sub-deadlines among tasks are imbalance. Then, the Earliest Start Time of each task is updated based on the sub-deadlines of its predecessors and the communications between it and its predecessors (line 13).

D. Initial Scheduling

In this section, we present how to select fault-tolerant strategy from resubmission and replication for each task based on the results of the Deadline Division at first. We also introduce how to schedule all tasks for their first execution as well as the backup copies of the tasks with replication strategy.

Suppose task t_i is allocated on one VM, the time that needed for t_i executing successfully on this VM should contain the maximum data transfer time from its predecessors and the execution time on this VM. If task t_i selects resubmission as its fault-tolerant strategy, its sub-deadline needs to satisfy the requirement of executing this task at least twice on different VMs in the best situation. When taken the initial boot time of VM $delay$ into consideration, the required time for task selecting resubmission as its fault-tolerant strategy can be denoted as

$$sdl(t_i) - EST(t_i) \ge 2 \times (delay + \max_{t_p \in pred(t_i)}\{e_{p,i}\} + MET(t_i)) . \quad (3)$$

If Eq.(3) is satisfied, the task will select resubmission as its fault-tolerant strategy and set the fault-tolerant strategy type $fts(t_i)$ as $resubmission$. Otherwise, it selects replication and set $fts(t_i)$ as $replication$. In this way, task t_i can determine its fault-tolerant strategy according to its available sub-deadline and the resources performance without any historic trace data. Furthermore, resubmission and replication are combined together for fault tolerance to play their respective advantages.

Then, how to select the VM type, reserve corresponding serve time interval for each task and allocate the reserved serve time interval on which VM in the selected VM type are needed to be considered. For the task with resubmission strategy, to facilitate the description, we define the execution before and after fault as initial execution and reexecution, respectively. We also define the execution for the task with replication strategy as initial execution.

As mentioned above, the initial execution is decided during the scheduling process and the reexecution is designed during the task execution process. So it is necessary to arrange the initial execution at first. For the task with resubmission strategy, if a fault is happed during the initial execution process of t_i, the reexecution is required to be completed before its sub-deadline. The minimum time for the reexecution of t_i (denoted as $reex(t_i)$) should contain the initial boot time of VM $delay$, the maximum data transfer time from its predecessors and the minimum execution time on all VM types at least. That is

$$reex(t_i) = dealy + \max_{t_p \in pred(t_i)}\{e_{p,i}\} + MET(t_i) \quad (4)$$

So the maximum time for the initial execution of t_i with resubmission is $sdl(t_i) - EST(t_i) - reex(t_i)$. Then, the VM type vmt_x for the initial execution of t_i can be determined by

$$\begin{cases} vmt_x \in \{vmt_1, vmt_2, \cdots, vmt_m, \cdots, vmt_M\} \\ s.t.\ et_i^x \le sdl(t_i) - EST(t_i) - reex(t_i) - . \\ \qquad delay - \max_{t_p \in pred(t_i)} e_{p,i} \end{cases} \quad (5)$$

If multiple VM types can satisfy Eq.(5), the VM type with the minimum value of et_i^x (not $MET(t_i)$) is selected as the elected VM type to reserve enough time for the reexecution. If no VM type can satisfy Eq.(5), the VM type with the closest value to $sdl(t_i) - EST(t_i) - reex(t_i) - delay - \max_{t_p \in pred(t_i)} e_{p,i}$ is selected.

For the task with replication strategy, two VMs are selected to execute the primary and backup copy simultaneously. So the task t_i always can be completed successfully, even in the context of fault. The VM type for t_i with replication strategy can be decided by the following equation.

$$\begin{cases} vmt_x \in \{vmt_1, vmt_2, \cdots, vmt_m, \cdots, vmt_M\} \\ s.t.\ et_i^x \le sdl(t_i) - EST(t_i) - delay \\ \qquad - \max_{t_p \in pred(t_i)} e_{p,i} \end{cases} \quad (6)$$

If multiple VM types can satisfy Eq.(6), the VM type with the maximum value of et_i^x is selected as the elected VM type to reduce the cost of customer as the VM with higher performance is usually more expensive and the economic cost is also needed to be considered when using resources in Cloud systems, which is similar to the economic cost in Grid systems [31,32]. If no VM type can satisfy Eq.(6), the VM type with the closest value to $sdl(t_i) - EST(t_i) - delay - \max_{t_p \in pred(t_i)} e_{p,i}$ is selected.

Then, the planned finish time $pft(t_i)$ of t_i can be denoted as

$$pft(t_i) = delay + EST(t_i) + \max_{t_p \in pred(t_i)} e_{p,i} + et_i^x . \quad (7)$$

For the task t_i with resubmission, if its initial execution is completed successfully, the actual finish time $aft(t_i)$ of task t_i is equal to $pft(t_i)$. Otherwise, $aft(t_i)$ is determined by the reexecution of t_i. For the task t_i with replication, $pft(t_i)$ is always equal to $aft(t_i)$. Obviously, for task with either resubmission or replication, $sdl(t_i) \ge aft(t_i) \ge pft(t_i)$.

After t_i is completed successfully, the VM where t_i is executed cannot be shut down immediately as it needs to transfer corresponding output to its successors. So the serve

time interval on one selected VM with type vmt_x for the initial execution of t_i can be denoted as $[EST(t_i), pft(t_i) + \max_{t_s \in succ(t_i)}\{e_{i,s}\}]$.

Then, the serve time interval $[EST(t_i), pft(t_i) + \max_{t_s \in succ(t_i)}\{e_{i,s}\}]$ reserved on which VM with type vmt_x is need to be considered as each type of VM has infinite amount and lots of VMs can be in active status simultaneously. Let $Active\_VMs$ and rvm_m^n denote the set of already reserved VMs by other tasks and the *n*-th reserved VM with type vmt_m, respectively. Let $[rvm_m^n, rst, rft]$ denote the already reserved time interval of rvm_m^n, where rst and rft are the reserved start time and reserved finish time, respectively. If multiple tasks can be allocated on the same VM in the context of fault-tolerant and deadline-constrained, the resource utilization of Cloud systems can be improved greatly [4]. So we try to find one VM in the set of already reserved VMs with the same type vmt_x at first. That is

$$\begin{cases} \forall rvm \in Active\_VMs \\ s.t.\ (rvm.vmt = vmt_x)\ and\ ((pft(t_i) + \\ \max_{t_s \in succ(t_i)}\{e_{i,s}\} + dealy \geq rst \geq pft(t_i) + \\ \max_{t_s \in succ(t_i)}\{e_{i,s}\})\ or\ (rft + delay \geq EST(t_i) \geq rft) \end{cases} . \quad (8)$$

If there is one VM in $Active\_VMs$ can satisfy Eq. (8), this VM is selected for the execution of t_i and the reserved serve time interval of this VM is updated. Otherwise, a new VM with type vmt_x is created for the execution of t_i and added to $Active\_VMs$ with reserved serve time interval as $[EST(t_i), pft(t_i) + \max_{t_s \in succ(t_i)}\{e_{i,s}\}]$.

The pseudo code for the initial scheduling is presented in Algorithm3. Through initial scheduling, both the VM type and corresponding reserved serve time interval on suitable VM are decided for the initial execution of each task. However, the reexecution for the task with resubmission strategy is designed during the execution process and introduced in the next section.

E. Online Scheduling and Reservation Adjustment

As for the task with replication strategy, it always can be finished successfully even in the context of fault as two VMs are selected to execute the primary and backup copy simultaneously during the initial scheduling process. As for the task with resubmission strategy, if a fault happens during its initial execution process, its sub-deadline can allow it to execute at least twice as mentioned in the previous section. So it is necessary to find another VM for its first reexecution.

Let $fault(t_i)$ denote the time when the fault is detected during the initial execution process of t_i with resubmission. Obviously, $fault(t_i) > EST(t_i)$. Thus, the left time for reexecution is $sdl(t_i) - fault(t_i)$. So the VM type for the first reexecution can be determined by the following equation.

$$\begin{cases} vmt_x \in \{vmt_1, vmt_2, \cdots, vmt_m, \cdots, vmt_M\} \\ s.t.\ et_i^x \leq sdl(t_i) - fault(t_i) - delay \\ \quad - \max_{t_p \in pred(t_i)} e_{p,i} \end{cases} \quad (9)$$

If multiple VM types can satisfy Eq.(9), the VM type with the maximum value of et_i^x is selected as the elected VM type to reduce the workflow execution cost. If no VM type can satisfy Eq.(9), the VM type with the closest value to $sdl(t_i) - fault(t_i) - delay - \max_{t_p \in pred(t_i)} e_{p,i}$ is selected.

Algorithm3: IniScheduling()

Input: the result of Algorithm 2;
the VM type $vmt_m (1 \leq m \leq M)$;
Output: the initial scheduling strategy for each task;

```
for i = 1 to n
    if (Eq.(3) is satisfied )
        fts(t_i) ← resubmission;
        if (Eq.(5) is satisfied )
            select the VM type vmt_x with the minimum value of et_i^x;
        else
            select the VM type vmt_x with the closest value to sdl(t_i) − EST(t_i) − reex(t_i) − delay − max_{t_p∈pred(t_i)} e_{p,i};
    else
        fts(t_i) ← replication;
        if (Eq.(6) is satisfied )
            select the VM type vmt_x with the maximum value of et_i^x;
        else
            select the VM type vmt_x with the closest value to
            sdl(t_i) − EST(t_i) − delay − max_{t_p∈pred(t_i)} e_{p,i};
    calculate the planed finish time pft(t_i) according to Eq.(7);
    if (exist one VM belong to Active_VMs and santify Eq.(8))
        reserve [EST(t_i), pft(t_i) + max_{t_s∈succ(t_i)}{e_{i,s}}] on this VM for t_i;
        update [rvm_m^n, rst, rft];
    else
        create a new VM and reserve [EST(t_i), pft(t_i) + max_{t_s∈succ(t_i)}{e_{i,s}}] for t_i;
```

Then, after the first reexecution after fault for t_i with resubmission strategy, the planed finish time $pft(t_i)$ can be denoted as

$$pft(t_i) = fault(t_i) + delay + \max_{t_p \in pred(t_i)} e_{p,i} + et_i^x. \quad (10)$$

So the serve time interval on one selected VM with type vmt_x for the reexecution of t_i is $[fault(t_i), pft(t_i) + \max_{t_s \in succ(t_i)}\{e_{i,s}\}]$. The next process for reserving the serve time interval on one VM for the first reexecution of t_i is similar to the corresponding process in the Initial Scheduling.

If the first reexecution of t_i is completed successfully, the actual finish time $aft(t_i)$ is equal to $pft(t_i)$. However, if the first reexecution of t_i encounters another fault at the time $fault'(t_i)$, thus the left time for t_i is $sdl(t_i) - fault'(t_i)$. Under this situation, t_i will decide its fault-tolerant strategy from resubmission and replication according to Eq.(3) by taking $fault'(t_i)$ to replace $EST(t_i)$ in Eq.(3). Then, it adopts the same procedure as mentioned in the above sections to decide its service resource and time interval and thus its actual finish time $aft(t_i)$ is decided by the successful execution.

Through the aforementioned Initial Scheduling and Online Scheduling, the Cloud systems try to complete each task in/near its sub-deadline, even in the context of fault. In the above

analysis, it can find that the Cloud systems reserve corresponding serve time interval for any possible fault of each task during the scheduling process. However, some considered faults may not be happened during the task execution process. If one task can be finished before its sub-deadline, especially for the task with resubmission when no fault happens during its initial execution, the time slot between $sdl(t_i)$ and $aft(t_i)$ can be used for other unexecuted tasks and thus alleviate the time constraints of these tasks effectively, especially in the context of fault and constraint of deadline. So we design an Online Reservation Adjustment scheme to apply these time slots and enlarge the sub-deadlines of some unexecuted tasks during the task execution process. For t_i, if all of its predecessor tasks have been completed before their sub-deadlines, its Earliest Start Time $EST(t_i)$ can be adjusted to Modified Earliest Start Time $MEST(t_i)$ as

$$MEST(t_i) = EST(t_i) - min_{t_p \in pred(t_i)}\{sdl(t_p) - aft(t_p)\}. \quad (11)$$

And the available time slot for t_i can be enlarged to $[MEST(t_i), sdl(t_i)]$.

With the modified time slot of t_i, its fault-tolerant strategy, including the selection between replication and resubmission, the reserved VM type and serve time interval, can also be changed. The change process is similar to Algorithm2. However, the criteria for determining select replication or resubmission is changed as follows.

$$sdl(t_i) - MEST(t_i) \geq 2 \times (delay + max_{t_p \in pred(t_i)}\{e_{p,i}\} + MET(t_i)) \quad (12)$$

The pseudo code for the Online Scheduling and Reservation Adjustment of task t_i is presented in Algorithm4.

Algorithm4: OnlSchResAdj()

```
1  for (each task t_p ∈ pred(t_i))
2    if (one fault is detected during its execution)
3      if (fts(t_p) == resubmission)
4        if (it is the first fault for t_p)
5          if (Eq.(9) is satisfied )
6            select the VM type vmt_x with the maximum
             value of et_p^x;
7          else
8            select the VM type vmt_x with the closest
             value to sdl(t_p) − fault(t_p) − delay −
             max_{t_p' ∈ pred(t_p)} e_{p',p};
9          calculate the planed finish time pft(t_p) for
           reexecution according to Eq.(10);
10         reserve [fault(t_p), pft(t_p) + max_{t_i ∈ succ(t_p)}{e_{p,i}}]
           on one VM as same as lines 18-23 in Algorithm
           3;
11       else
12         decide fault-tolerant strategy according to
           Eq.(3) by taking fault'(t_p) to replace
           EST(t_p) in Eq.(3);
13         take the same procedure as Algorithm 3;
14   calculate {sdl(t_p) − aft(t_p)};
15 calculate MEST(t_i) according to Eq.(11);
16 if (fts(t_i) == replication &&(Eq.(12) is satisfied))
17   fts(t_i) ← resubmission;
18   release the initial reserved resource for t_i;
19   reserve new resource for t_i as same as lines 2-19 in
     Algorithm3;
```

V. Performance Evaluation

In this section, the WorkflowSim toolkit [33], which is a modern simulation framework aimed for workflow scheduling in Cloud systems and widely used in some related researches [7,34], is used to simulate the proposed ICFWS and evaluate the results with other related works.

A. Simulation setup

We compare the performance of ICFWS with some resubmission and replication based fault-tolerant workflow scheduling algorithms, respectively. IRW [7] is selected as the representative of resubmission based algorithms while CCRH [22] is chosen to stand for the replication based algorithms because both of them are designed for fault-tolerant workflow scheduling in Cloud systems. We also compare ICFWS with FTWS [23] and RRADFTRC [24]. Both FTWS and RRADFTRC adopted replication and resubmission simultaneously for fault-tolerant workflow scheduling in Cloud systems. The difference between FTWS and RRADFTRC is that the FTWS took the impact of resubmission, out degree and deadline of each task to adjust the backup number of this task while the RRADFTRC did not. The above algorithms are compared from the following metrics:

(1) Task Completion Rate (*TCR*) is defined as the ratio of completed tasks over the total number of tasks in the tested workflow at the time of soft deadline, reflecting the task completion efficiency of the compared algorithms in the context of fault.

(2) VMs Reserve Time Rate (*VRTR*) is defined as the ratio of the total reserved serve time of all the VMs in the context of fault over the total reserved serve time of all the VMs under ideal status got by HEFT [27], reflecting the system resource consumption of the compared algorithms under fault.

Both of above metrics show the costs of the scheduling algorithms. The *TCR* reflects the cost about time while the *VRTR* shows the resource cost in the context of fault.

In order to clearly observe the performances of the compared algorithms, we conduct the simulations on both real-world and randomly generated workflows. Pegasus project[1] has published a lot of real-world workflows, including Montage, CyberShake, Epigenomics and Inspiral [35]. These workflows have been widely used to measure the performance of scheduling algorithms, and thus we include them in our simulations. The DAG characteristics of these workflows, including the numbers of nodes and edges, average data size and average task runtime, are given in Table III, and the structures of theses workflows are shown in Fig. 2 [7,35,36]. More details about these real-world workflows can be referred to [35].

Besides real-world workflows, we also conduct simulations on some randomly generated workflows. The DAG generator tool used in [36] is chosen to form the workflows for simulation. The generator tool defines the DAG shape based on five

[1] https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2687923, IEEE Transactions on Parallel and Distributed Systems

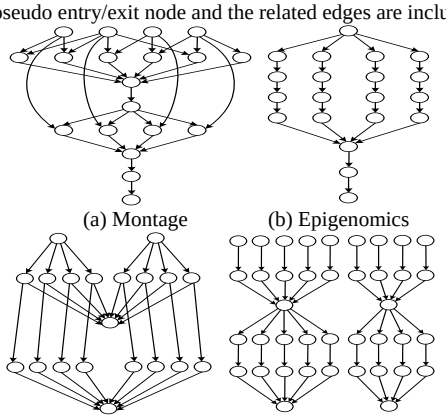
parameters: *task count*, *width*, *regularity*, *density*, and *jumps*. The *task count* denotes the total number of tasks in the generated workflow. The *width* determines the maximum number of tasks that can be executed concurrently. The *regularity* indicates the uniformity of the number of tasks in each level. The *density* denotes the number of edges between two levels of the DAG. A *jump* indicates that an edge can go from level *l* to level *l+jump*. A *jump* of one is an ordinary connection between two consecutive levels. In our simulations, the parameters for DAG generation are shown in Table IV.

The VMs in simulations are heterogeneous and modeled similar to the VMs provided by Amazon EC2 as shown in Table I. The number of usable VMs in simulations is not restricted to imitate the unlimited resources in Cloud systems. During the simulation, the maximum replication and resubmission count are set as three for both FTWS and RRADFTRC. In order to test the impact of simulation parameters to the results, we investigate the compared metrics under two different situations for each DAG. The first one is that the soft deadline of each DAG is change dynamically under the case that the failure rate of VMs is constant. In this case, the soft deadline of each test case is set from 1.1 times of makespan got by HEFT in ideal status to 1.5 times with step of 0.1 times as well as the failure rate is 10%. The other situation is that the failure rate of VMs is change dynamically but the soft deadline of each DAG is constant. Under this situation, we adopt 5%, 7.5%, 10%, 12.5% and 15% as the failure rate, respectively, along with 1.3 times of makespan got by HEFT in ideal status as soft deadline unchanged. We call the first situation as Case One and the other one as Case Two as shown in Table V.

TABLE III. CHARACTERISTICS OF THE REAL-WORD WORKFLOWS

Workflow	Number of tasks	Number of edges	Average data size (MB)	Average task runtime (s)
Montage_1000	1000	4485	3.21	11.36
Epigenomics_997	997	3228	388.59	3858.67
CyberShake_1000	1000	3988	102.29	22.71
Inspiral_1000	1000	3246	8.90	227.25

* When calculating the number of edges, average data size and average task runtime, the pseudo entry/exit node and the related edges are included.



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

Fig. 2. The structure of evaluated real-world workflows

TABLE IV. PARAMETERS FOR GENETATED WORKFLOWS

Parameter	Value (min)-(max)
task count	(1100)-(1500)
width	[0.2, 0.4, 0.8]
regularity	[0.2, 0.4, 0.8]
density	[0.2, 0.4, 0.8]
jumps	[1, 2, 3]

TABLE V. PARAMETERS FOR CASE ONE AND CASE TWO

	soft deadline (times)	failure rate (%)
Case One	1.1,1.2,1.3,1.4,1.5	10
Case Two	1.3	5, 7.5,10,12.5 15

B. Simulations on real-world workflows

1) Case One

Firstly, we compare the *TCR* of real-world workflows under Case One and the results are presented in Fig. 3. It can be found that the *TCR*s of all algorithms are increased with the growth of soft deadline. This is because more time can be used to decrease the influence caused by fault with the increase of soft deadline. The different structure as shown in Table III can explain why the identical algorithm can get the different *TCR* among different test cases under the same soft deadline.

Figure 3 also shows that the ICFWS, FTWS, RRADFTRC and CCRH get the similar results of *TCR* under different soft deadlines and all of them are better than that of IRW. This is caused by the different adopted fault-tolerant strategies of these algorithms. The FTWS and RRADFTRC adopted both resubmission and replication simultaneously for each task. Although the FTWS adopted the impact of resubmission, out degree and deadline to adjust the number of backups, it still formed multiple backups for each task. And the CCRH only adopted replication as its fault-tolerant strategy. As for the proposed ICFWS, it combines replication and resubmission together and selects one approach from them for each task as its fault-tolerant strategy. Therefore, all these four algorithms adopt replication for all or parts of tasks as their fault-tolerant strategy. It is known that the procedure for finding suitable resource for fault in replication based algorithms is designed during the scheduling process. As a result, lots of time is saved and more tasks are completed at the time of soft deadline. Thus, they have high value of *TCR*s. As for the resubmission based algorithm IRW, the procedure for finding suitable VM for resubmission is performed during the task execution process and takes lots of time and then leads a delay to finish this task. Caused by the dependent relationship among tasks, the delay can also propagates to other tasks depended on it. So the *TCR* of IRW is lower than the other four algorithms. When taking the ICFWS into consideration, as some tasks select replication as their fault-tolerant strategy and the others choose resubmission, the *TCR* of ICFWS is a littler lower than those of FTWS, RRADFTRC and CCRH and higher than that of IRW. Moreover, both the replication and resubmission in ICFWS are different from that in other algorithms. As for replication, two VMs are active executed simultaneously for one task in ICFWS while the backups in FTWS, RRADFTRC and CCRH were only passive started to work when the primary or one other replication had encountered a fault. The active execution scheme for replication in ICFWS can also save parts of time. At the same time, the resubmission in ICFWS is constrained by the sub-deadline of this task, whereas no such limitation in IRW. Furthermore, the resubmission in ICFWS is also different from those in FTWS and RRADFTRC. Based on the results of Deadline Division and assigned fault-tolerant strategy of each task, the resubmission is active adopted directly if the initial execution of this task encounters a fault. However, the

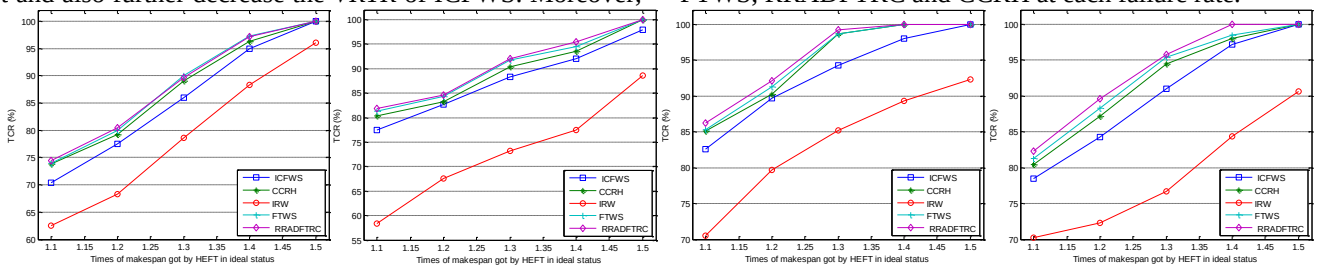
resubmissions in both FTWS and RRADFTRC were only passive adopted when all backups of this task had failed even the counter of backups is adjusted by the impact of resubmission, out degree and deadline in FTWS.

The results about *VRTR* under Case One are shown in Fig.4. It can be found no matter which fault-tolerant strategy is adopted, more resources are needed for fault. Furthermore, the *VRTR*s of RRADFTRC, FTWS and ICFWS are decreased with the increase of soft deadline. The RRADFTRC and FTWS get the top two values of *VRTR* among all algorithms. This is due to that they adopted replication and resubmission simultaneously for each task. For RRADFTRC, its *VRTR* is greater than other algorithms because each task has three backup copies. As for FTWS, the counter of backups for each task was adjusted according to the impact of resubmission, out degree and deadline and thus some tasks had less backup copies. So the *VRTR* of FTWS is less than four but still greater than three. The CCRH only adopted replication, which needed two resources at least for each task, for fault. Moreover, because the on-demand resource provisioning is not considered in CCRH and the resources for backups cannot always have the same performance for primary copies, the *VRTR* of CCRH is greater than two. As for IRW, it only chose resubmission when the task encountered a fault. So it has the minimum value of *VRTR*. Moreover, both CCRH and IRW took one single fault-tolerant strategy for all tasks, without considering the influence of soft deadline. Thus both of them get the unchanged *VRTR*s with different soft deadlines as shown in Fig.4. As for the proposed ICFWS, it reserves moderately serve time of VMs for fault and the reserved serve time is decreased with the increase of soft deadline. This is caused by that the ICFWS adopts different fault-tolerant strategy for different task based on the imbalance sub-deadlines among tasks and the performance of resources. When the soft deadline is low, more tasks choose replication as their fault-tolerant strategy. With the increased soft deadline, same tasks change their fault-tolerant strategy from replication to resubmission, and then the *VRTR* is decreased. Furthermore, the Online Reservation Adjustment can make full use of time slot and also further decrease the *VRTR* of ICFWS. Moreover, the on-demand resource provisioning is also adopted and thus more resources with same performance can be got in ICFWS. So the *VRTR* of ICFWS becomes lower and lower with the increase of soft deadline.

From the perspective of one single compared metric, the achievements of ICFWS are similar to those of other algorithms. For example, the *TCR* of ICFWS is similar to those of CCRH, FTWS and RRADFTRC. However, no matter *TCR* or *VRTR*, they only reflect one aspect of all algorithms. The *TCR* reflects the cost about time in the context of fault while the *VRTR* shows the resource cost. So it is necessary to consider *TCR* and *VRTR* simultaneously. From this perspective, it can be found that the ICFWS gets a higher *TCR* at a lower *VRTR* while the other algorithms get a higher *TCR* at a higher *VRTR* or a lower *TCR* at a lower *VRTR*. This means that only the proposed ICFWS can reduce the cost of time and resource at the same time for the soft deadline constrained-workflow in Cloud systems. As for CCRH, FTWS and RRADFTRC, they complete more tasks at the cost of lower resource utilization. On the contrary, the IRW improve the resource utilization of Cloud systems but complete fewer tasks at the soft deadline. As mentioned in the above section, completed more tasks with less resources is valuable for both users and resource providers. So the proposed ICFWS is more practical than other algorithms for the soft deadline-constrained workflow in Cloud systems.

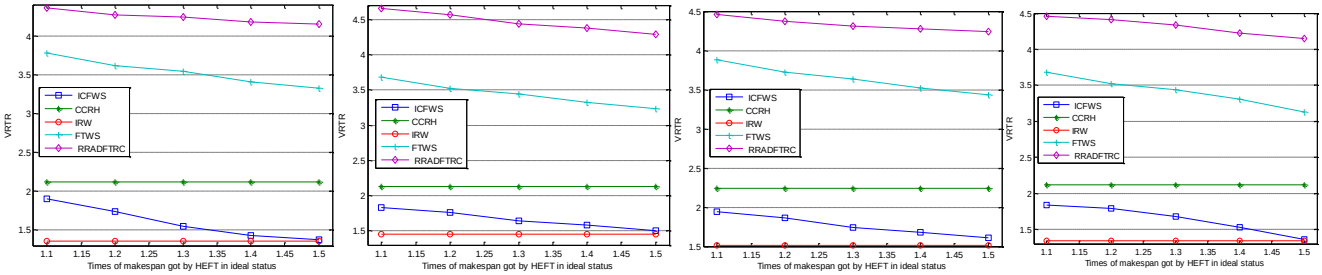
2) Case Two

The results about *TCR*s under Case Two are shown in Fig.5. It can be seen that the *TCR*s of all algorithms are decreased with the increase of failure rate. The reason for this phenomenon is that more time is needed to deal with the rose faults with the increase of failure rate and thus fewer tasks are completed at the defined soft deadline. However, the *TCR*s of ICFWS, FTWS, RRADFTRC and CCRH are still better than that of IRW. This is due to that the time required by resubmission is more than that of replication and more resubmissions are needed in IRW with the increase of failure rate. Caused by the same reason and some tasks select resubmission as their fault-tolerant strategy, the proposed ICFWS has a little smaller *TCR* than those of FTWS, RRADFTRC and CCRH at each failure rate.



(a) Montage (b) Epigenomics (c) Cybershake (d) Inspiral

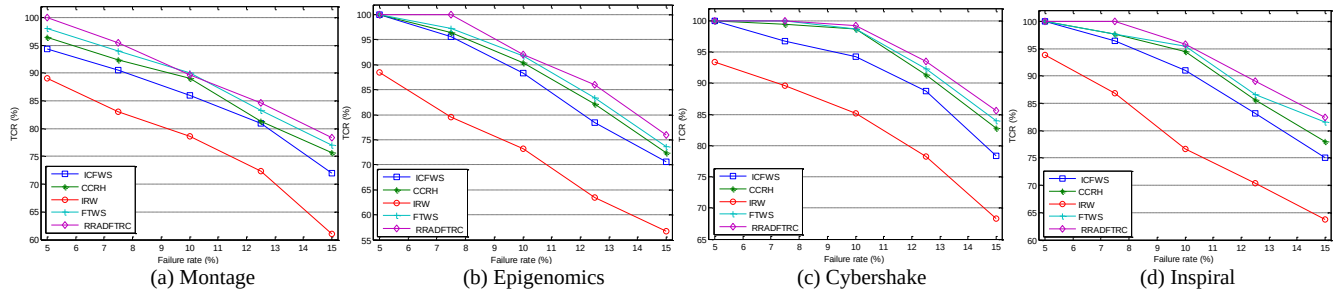
Fig. 3. Comparative *TCR* on real-world workflows under Case One



(a) Montage (b) Epigenomics (c) Cybershake (d) Inspiral

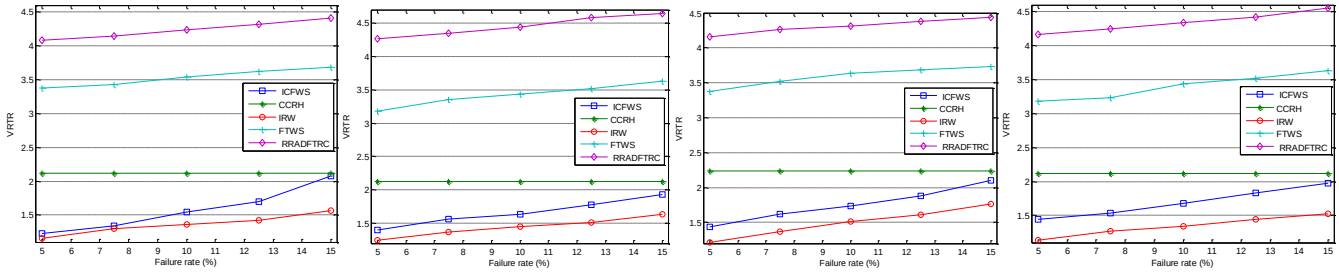
Fig. 4. Comparative *VRTR* on real-world workflows under Case One

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2687923, IEEE Transactions on Parallel and Distributed Systems



(a) Montage (b) Epigenomics (c) Cybershake (d) Inspiral

Fig. 5. Comparative *TCR* on real-world workflows under Case Two



(a) Montage (b) Epigenomics (c) Cybershake (d) Inspiral

Fig. 6. Comparative *VRTR* on real-world workflows under Case Two

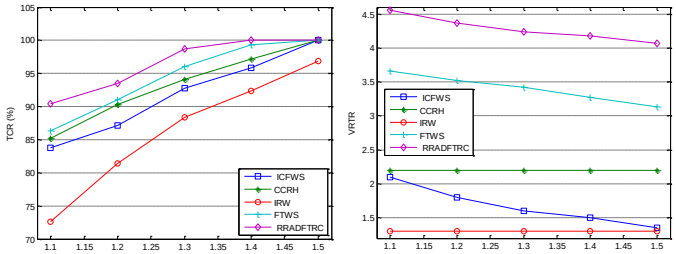
The results about *VRTR* of all algorithms under Case Two are presented in Fig.6. It indicates that the *VRTR*s are grown up with the increase of failure rate except CCTH. The reason for this phenomenon is that the CCRH only adopted replication as its fault-tolerant strategy and corresponding approach for every possible fault is designed during the scheduling process. So the increased failure rate does not cause any impact to the *VRTR* of CCRH. As for the other four algorithms, more resubmissions are needed with the increase of failure rate and thus rise up the *VRTR*s. Fig.6 also shows that the ICFWS offers the maximal increment of *VRTR* in all algorithms for each real-world workflow. This is caused by the Online Reservation Adjustment in ICFWS. With the increase of failure rate, more and more tasks change their fault-tolerant strategy type from resubmission to replication by the Online Reservation Adjustment after they encounter faults and thus the *VRTR* is increased observably. Although the resubmission is also increased in FTWS and RRADFTRC under Case Two, each task still has some backups and the resubmission is only passive adopted when all backups of this task have failed.

When taking *TCR* and *VRTR* into consideration simultaneously, it can be found that the proposed ICFWS still outperforms other competitors even with the increased failure rate.

C. Simulations on randomly generated workflows

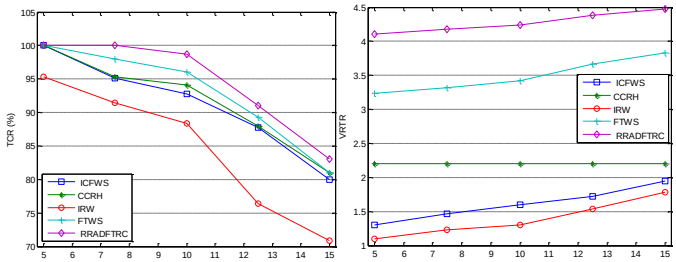
Besides real-world workflows, we also conduct simulations on some randomly generated workflows. We use the DAG generator tool to form a DAG with 1300 tasks as the tested case. The simulation results about *TCR* and *VRTR* under Case One for the generated workflow are presented in Fig. 7. The results about *TCR* and *VRTR* under Case Two are shown in Fig.8. The results on randomly generated workflow are similar to the previous simulations on real-world workflows and confirm the findings in terms of the compared metrics.

We also execute some simulations on randomly generated workflow with varied task number under fixed soft deadline (1. 3 times of makespan got by HEFT in ideal status) and failure



(a) *TCR* (b) *VRTR*

Fig. 7. Comparison on randomly generated workflow with fixed task count under Case One



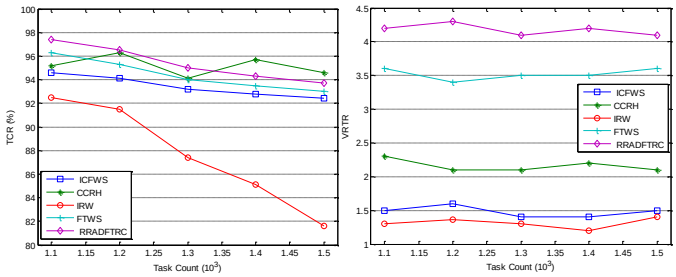
(a) *TCR* (b) *VRTR*

Fig. 8. Comparison on randomly generated workflow with fixed task count under Case Two

rate (10%) to evaluate the impact of task number. The results under this situation are presented in Fig.9.

From Fig.9(a), it can be found the TCRs are decreased with the increase of task number except CCRH. The decrease of TCR is mainly caused by the increase of resubmissions. When the task counter is increased under the same failure rate, the resubmissions in ICFWS, FTWS, RRADFTRC and IRW are also grown up. Then, the TCRs of these algorithms are decreased. However, the CCRH does not adopt any resubmission in its execution process and thus its TCR is not decreased. Fig. 9 also shows the TCRs of RRADFTRC, FTWS and ICFWS are reduced mildly with the increasing of task counter while that of IRW are decreased significantly. The rea-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2687923, IEEE Transactions on Parallel and Distributed Systems



(a) To *TCR* (b) To *VRTR*

Fig. 9. The impact of task counter under fixed soft deadline and failure rate

sons as follows can explain this phenomenon. Firstly, although the resubmission strategy is also used in RRADFTRC and FTWS, the resubmission for each task is only passive adopted after all of its backup copies has failed. As for the ICFWS, the resubmission is active adopted and constrained by the sub-deadlines of these tasks. Moreover, the Online Reservation Adjustment also changes the fault-tolerant type of some tasks from resubmission to replication in ICFWS. At the same time, only the resubmission is adopted in IRW. The little different time required for fault by replication and resubmission for one task can lead to obviously delay of the whole workflow by other unexecuted tasks, which have dependent relationship on this task. The delay caused by fault between replication and resubmission is becoming more and more obviously with the increasing of task counter.

However, different from RRADFTRC, FTWS and CCRH, the ICFWS keeps a low value of *VRTR* as shown in Fig. 9(b). This phenomenon is due to the tasks selected resubmission as its fault-tolerant strategy and the Online Reservation Adjustment scheme, which also changes the fault-tolerant strategy type of some tasks from replication to resubmission besides from resubmission to replication during the task execution process.

VI. Conclusion and Future Work

In this paper, we propose a novel fault-tolerant scheduling method, called ICFWS, for soft deadline constrained-workflow in Cloud systems by combining resubmission and replication together. The ICFWS can play the respective advantages of both resubmission and replication for fault-tolerant scheduling while trying to meet the soft deadline of workflow. It first divides the whole soft deadline of workflow into multiple sub-deadlines for all tasks in a recursive manner. Then, it selects the corresponding fault-tolerant strategy from resubmission and replication for each task based on the results of deadline division. After that, the ICFWS reserves corresponding VM and time interval for each task by taking on-demand resource provisioning model into consideration. For the task with resubmission strategy, the ICFWS also designs an online resubmission scheme when a fault is happened during its execution process. In order to take full advantage of time slot, the ICFWS also designs an online reservation adjustment scheme to enlarge the sub-deadlines of some unexecuted tasks during the task execution process and then adjust the fault-tolerant strategy of these tasks. In order to evaluate the performance of the proposed ICFWS, a series of simulations are conducted on both real-world and randomly generated workflows. The simulation results confirm that the proposed ICFWS is able to play the respective advantage of replication and resubmission for fault tolerance while trying to meet the soft deadline of workflow and outperform some corresponding competitors.

Besides fault, the performance fluctuation of VMs also plays a negative effect to the execution of workflow. So we plan to design a robust scheduling algorithm with elastic resource provisioning strategy for workflow in the Cloud systems in the future.

References

[1] Buyya R, Yeo C S, Venugopal S, et al. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility[J]. Future Generation computer systems, 2009, 25(6): 599-616.

[2] Fu Z, Ren K, Shu J, et al. Enabling personalized search over encrypted outsourced data with efficiency improvement[J]. IEEE Transactions on Parallel and Distributed Systems, 2016, 27(9): 2546-2559.

[3] Fu Z, Sun X, Ji S, et al. Towards efficient content-aware search over encrypted outsourced data in cloud[C]//Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on. IEEE, 2016: 1-9.

[4] Sahni J, Vidyarthi D. A Cost-Effective Deadline-Constrained Dynamic Scheduling Algorithm for Scientific Workflows in a Cloud Environment[J]. IEEE Transactions on Cloud Computing, in press, http://doi.ieeecomputersociety.org/10.1109/TCC.2015.2451649.

[5] Li Z, Ge J, Hu H, et al. Cost and Energy Aware Scheduling Algorithm for Scientific Workflows with Deadline Constraint in Clouds[J]. IEEE Transactions on Services Computing, in press, http://doi.ieeecomputersociety.org/10.1109/TSC.2015.2466545.

[6] Pezoa J E, Dhakal S, Hayat M M. Maximizing service reliability in distributed computing systems with random node failures: Theory and implementation[J]. IEEE Transactions on Parallel and Distributed Systems, 2010, 21(10): 1531-1544.

[7] Yao G, Ding Y, Ren L, et al. An immune system-inspired rescheduling algorithm for workflow in Cloud systems[J]. Knowledge-Based Systems, 2016, 99: 39-50.

[8] Qin X, Jiang H. A novel fault-tolerant scheduling algorithm for precedence constrained tasks in real-time heterogeneous systems[J]. Parallel Computing, 2006, 32(5): 331-356.

[9] Plankensteiner K, Prodan R. Meeting soft deadlines in scientific workflows using resubmission impact [J]. IEEE Transactions on Parallel and Distributed Systems, 2012, 23(5): 890-901.

[10] Poola D, Ramamohanarao K, Buyya R. Enhancing Reliability of Workflow Execution Using Task Replication and Spot Instances[J]. ACM Transactions on Autonomous and Adaptive Systems , 2016, 10(4): 30.

[11] Javadi B, Kondo D, Vincent J M, et al. Discovering statistical models of availability in large distributed systems: An empirical study of seti@ home [J]. IEEE Transactions on Parallel and Distributed Systems, 2011, 22(11): 1896-1903.

[12] Jhawar R, Piuri V, Santambrogio M. Fault tolerance management in cloud computing: A system-level perspective [J]. IEEE Systems Journal, 2013, 7(2): 288-297.

[13] Zheng Z, Zhou T C, Lyu M R, et al. Component ranking for fault-tolerant cloud applications [J]. IEEE Transactions on Services Computing, 2012, 5(4): 540-550.

[14] Qiu W, Zheng Z, Wang X, et al. Reliability-Based Design Optimization for Cloud Migration [J]. IEEE Transactions on Services Computing, 2014, 7(2): 223-236.

[15] Chen W, Lee Y C, Fekete A, et al. Adaptive multiple-workflow scheduling with task rearrangement [J]. The Journal of Supercomputing, 2015, 71(4): 1297-1317.

[16] Olteanu A, Pop F, Dobre C, et al. A dynamic rescheduling algorithm for resource management in large scale dependable distributed systems [J]. Computers & Mathematics with Applications, 2012, 63(9): 1409-1423.

[17] Cao Y, Ro C W, Yin J W. Scheduling Analysis of Failure-aware VM in Cloud System [J]. International Journal of Control & Automation, 2014, 7(1): 243-250.

[18] Sakellariou R, Zhao H. A low-cost rescheduling policy for efficient mapping of workflows on grid systems [J]. Scientific Programming, 2004, 12(4): 253-262.

[19] Khan S U, Ahmad I. Comparison and analysis of ten static heuristics-based Internet data replication techniques[J]. Journal of Parallel and Distributed Computing, 2008, 68(2): 113-136.
[20] Zheng Q, Veeravalli B, Tham C K. On the design of fault-tolerant scheduling strategies using primary-backup approach for computational grids with low replication costs[J]. IEEE Transactions on Computers, 2009, 58(3): 380-393.
[21] Zheng Q, Veeravalli B. On the design of communication-aware fault-tolerant scheduling algorithms for precedence constrained tasks in grid computing systems with dedicated communication devices[J]. Journal of Parallel and Distributed Computing, 2009, 69(3): 282-294.
[22] Jing W, Liu Y. Multiple DAGs reliability model and fault-tolerant scheduling algorithm in cloud computing system[J]. Computer Modeling and New Technologies, 2014, 18(8): 22-30
[23] Jayadivya S K, Nirmala J S, Bhanu M S S. Fault tolerant workflow scheduling based on replication and resubmission of tasks in Cloud Computing[J]. International Journal on Computer Science and Engineering, 2012, 4(6): 996-1006.
[24] Patra P K, Singh H, Singh R, et al. Replication and Resubmission Based Adaptive Decision for Fault Tolerance in Real Time Cloud Computing: A New Approach[J]. International Journal of Service Science, Management, Engineering, and Technology, 2016, 7(2): 46-60.
[25] Wang J, Bao W, Zhu X, et al. FESTAL: Fault-Tolerant Elastic Scheduling Algorithm for Real-Time Tasks in Virtualized Clouds[J]. IEEE Transactions on Computers, 2015, 64(9): 2545-2558.
[26] Manimaran G, Murthy C. A fault-tolerant dynamic scheduling algorithm for multiprocessor real-time systems and its analysis[J]. IEEE Transactions on Parallel and Distributed Systems, 1998, 9(11): 1137-1152.
[27] Topcuoglu H, Hariri S, and Wu M. Performance-effective and low-complexity task scheduling for heterogeneous computing [J]. IEEE Transactions on Parallel and Distributed Systems, 2002, 13(3): 260-274.
[28] Kwok Y K, Ahmad I. Static scheduling algorithms for allocating directed task graphs to multiprocessors[J]. ACM Computing Surveys, 1999, 31(4): 406-471.
[29] Abrishami S, Naghibzadeh M, Epema D H J. Cost-driven scheduling of grid workflows using partial critical paths[J]. IEEE Transactions on Parallel and Distributed Systems, 2012, 23(8): 1400-1414.
[30] Abrishami S, Naghibzadeh M, Epema D H J. Deadline-constrained workflow scheduling algorithms for Infrastructure as a Service Clouds[J]. Future Generation Computer Systems, 2013, 29(1): 158-169.
[31] Khan S U, Ahmad I. Non-cooperative, semi-cooperative, and cooperative games-based grid resource allocation[C]//Proceedings 20th IEEE International Parallel & Distributed Processing Symposium. IEEE, 2006: 10 pp.
[32] Gao L, Ding Y, Ying H. Economics-inspired decentralized control approach for adaptive grid services and applications[J]. International Journal of Intelligent Systems, 2006, 21(12): 1269-1288.
[33] Chen W, Deelman E. Workflowsim: A toolkit for simulating scientific workflows in distributed environments[C]//E-Science (e-Science), 2012 IEEE 8th International Conference on. IEEE, 2012: 1-8.
[34] Yao G, Ding Y, Jin Y, et al. Endocrine-based coevolutionary multi-swarm for multi-objective workflow scheduling in a cloud system[J]. Soft Computing, in press, 10.1007/s00500-016-2063-8.
[35] Deelman E, Vahi K, Juve G, et al. Pegasus, a workflow management system for science automation[J]. Future Generation Computer Systems, 2015, 46: 17-35.
[36] Zhu Z, Zhang G X, Li M, et al. Evolutionary multi-objective workflow scheduling in cloud[J]. IEEE Transactions on Parallel and Distributed Systems, 2016, 27(5): 1344-1357.

Guangshun Yao is a PhD candidate at College of Information Sciences and Technology, Donghua University. He is also a lecturer at College of Computer and Information Engineering, Chuzhou University. His research interests include Cloud computing, workflow scheduling and Internet of Things.

Dr. Yongsheng Ding (M'00-SM'05) is currently a Professor at College of Information Sciences and Technology, Donghua University, Shanghai, China. He obtained the B.S. and Ph.D. degrees in Electrical Engineering from Donghua University, Shanghai, China in 1989 and 1998, respectively. He is a Changjiang Distinguished Professor appointed by the Ministry of Education, China. From 1996 to 1998, he was a Visiting Scientist at Biomedical Engineering Center, The University of Texas Medical Branch, TX, USA. From February 2005 to April 2005, he was a Visiting Professor at Department of Electrical and Computer Engineering, Wayne State University, MI, USA. From September 2007 to February 2008, he was a Visiting Professor at Harvard Medical School, Harvard University, MA, USA. He serves as Senior Member of Institute of Electrical and Electronics Engineers (IEEE). He has published more than 300 technical papers, and ten research monograph/advanced textbooks. His scientific interests include computational intelligence, network intelligence, intelligent Internet of things, big data intelligence, and intelligent robots.

Dr. Kuangrong Hao is currently a Professor at the College of Information Sciences and Technology, Donghua University, Shanghai, China. She obtained her B.S. degree in Mechanical Engineering from Hebei University of Technology, Tianjin, China in 1984, her M.S. degree from Ecole Normale Supérieur de Cachan, Paris, France in 1991, and her Ph.D. degree in Mathematics and Computer Science from Ecole Nationale des Ponts et Chaussées, Paris, France in 1995. She has published more than 100 technical papers, and three research monographs. Her scientific interests include machine vision, image processing, robot control, intelligent control, and digitized textile technology.