IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received December 4, 2017, accepted January 30, 2018, date of publication February 27, 2018, date of current version April 23, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2810214

Fault-Tolerant Scheduling for Hybrid Real-Time Tasks Based on CPB Model in Cloud

HAORAN HAN, WEIDONG BAO, XIAOMIN ZHU, (Member, IEEE), XIAOSHENG FENG, AND WEN ZHOU
College of Systems Engineering, National University of Defense Technology, Changsha 410073, China

Corresponding author: Haoran Han (hanhaoran16@nudt.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61572511 and Grant 71702186 and in part by the China Postdoctoral Science Foundation under Grant 2016M602960 and Grant 2017T100796.

ABSTRACT Clouds are becoming a very important platform for hybrid real-time tasks. To enhance the reliability of cloud, fault tolerance of cloud becomes a critical issue. However, the complexities and specialties of traditional fault-tolerant mechanisms cannot meet the fault-tolerant requirements of clouds. To address this issue, we propose a novel fault-tolerant scheduling algorithm named ARCHER for hybrid tasks in cloud. ARCHER has three significant characteristics: 1) it integrates the traditional **p**rimary/**b**ackup model and **c**heckpoint technology which can flexibly determine the execution time of the backup copies of tasks, so it greatly enhances the resource utilization and produces more time slots to execute tasks as many as possible; 2) it employs task classification mechanism to realize precise scheduling for different types of tasks and virtual machines, which reduces the response time of clouds; and 3) it uses time slot exploiting mechanism, task forward mechanism, and task transform mechanism to achieve high-resource utilization. We conduct extensive simulations to evaluate the performance of ARCHER by comparing it with four baseline algorithms. The experimental results show that ARCHER can effectively improve the resource utilization of cloud while guaranteeing fault tolerance.

INDEX TERMS Clouds, fault-tolerant scheduling, hybrid real-time tasks, primary-backup model, CPB model.

I. INTRODUCTION

Hybrid real-time tasks such as experimental analysis of nuclear physics, emergency response and traffic stream analysis, which includes data-intensive real-time tasks and computing-intensive real-time tasks, become an important part of scientific and industrial circles. And most of these hybrid real-time tasks choose clouds as the computing platform because clouds can provide on-demand resource base on "pay-as-you-go" model. In general, these hybrid real-time tasks have high fault tolerance requirement, just like scientific real-time tasks of astronomy, physics and biology. But we need to note that the faults in clouds are inevitable. It was reported that 8% of the virtual machines encounter errors at run time [4]. Thereby, fault-tolerance becomes a challenge in cloud computing [5].

Hybrid real-time tasks in scientific and industrial applications not only require the correctness of calculation results but also require fault tolerance of these tasks [3]. For instance, nuclear physics simulation is an application which has strict fault tolerance requirements. Since nuclear physics simulations' calculation process is very complex and requires high security, cloud needs to provide not only computing resource but also high fault-tolerant capability. As a consequence, fault tolerance becomes the most critical part in cloud.

For hybrid real-time tasks on clouds, scheduling plays an indispensable role to achieve fault tolerance. Fault-tolerant scheduling is widely studied, and there are many researches focus on fault-tolerant scheduling models [1], [2]. These models can be grouped into several categories [6]. In terms of time, important methods are resubmission and checkpoint. When a task fails, it will be resubmitted or rolled back to achieve fault tolerance. In terms of space, task duplication is commonly used to realize fault tolerance [7]. There also exist some researchers who investigate how to backup the data needed for tasks, including local backup storage and online backup storage, to achieve fast task-error recovery [8], [9].

We also need to note that hybrid real-time tasks in these applications are not in one type. Different types of tasks have different requirements for virtual machines.

Computing-intensive tasks require powerful computing capabilities of virtual machines. Similarly, data-intensive tasks require high bandwidth. However clouds will have long response time.

In order to achieve fault tolerance, most researches use multiple copies technology [15]. Backup copy of tasks will consume a large amount of cloud resources, which cause low utilization of cloud resources [10].

To the best of our knowledge, most work about fault-tolerant scheduling lacks consideration about multiple kinds of tasks. And no work has been down on fault-tolerant scheduling for hybrid real-time tasks in clouds. In this paper, we propose a novel fault-tolerant scheduling model named ARCHER for hybrid real-time tasks in clouds. The major contributions of this paper are summarized as follows.

- We design a novel classification mechanism for hybrid real-time tasks, which can divide tasks and virtual machines into different groups. Classification mechanism guarantees that different types of tasks can be scheduled to appropriate virtual machines.
- We propose a CPB model which combine the advantages of **C**heckpoint and traditional **PB** model. It not only guarantees cloud's fault tolerance, but also makes full use of the idle time slots to improve the resource utilization of cloud.
- We devise three mechanisms including time slot exploiting mechanism, task forward and transform mechanism. The time slot exploiting mechanism is able to efficiently utilize time slots by scheduling suitable tasks to these time slots. If there exist time slots that have no suitable tasks, the task forward and transform mechanism will be employed to forward task and take full advantage of virtual machines in different groups by transforming these tasks' type.
- Based on the CPB model and these three mechanisms, we propose a novel fault-tolerant scheduling algorithm named ARCHER. ARCHER can precisely determine the execution time of backup copies and provide more resources for critical tasks, which greatly improves the resource utilization of cloud while guaranteeing the fault tolerance.

The remainder of this paper is organized as follows. The next section reviews related work. Section III gives the system design of our study. Section IV introduces the proposed CPB model with its advantages. The ARCHER algorithm and its analysis are detailed in Section V. The performance evaluation of ARCHER is presented in Section VI. Section VII concludes this paper and give our future work.

II. RELATED WORK

Cloud experiences failures and performance fluctuations when executing tasks, which requires high fault tolerance of the cloud. Mao *et al.* proved the dynamic variability of virtual machines in clouds, and showed that about 8% of the virtual machines will fail when running [4]. Plankensteiner *et al.* [11] summarized the reasons for the failure of clouds, and further emphasized on the importance of fault tolerance in clouds.

In large-scale distributed systems such as cloud, failures are almost inevitable [12]. The failures can occur in varieties of levels, e.g., hardware level, operating system level, software level and so on [13]. In this paper, based on our fault-tolerant scheduling algorithm ARCHER, clouds can tolerate all the faults occur in one host. Generally, exist researches can be mainly classified into two categories: fault tolerance in time and fault tolerance in space.

Fault-tolerant researches in space are replicating multiple copies of tasks on different virtual machines. When a copy of a task fails, the other copies will execute in order to complete this task before its deadline [6]. The more copies are replicated, the higher fault-tolerant capability of cloud. However, these backup copies will occupy cloud resources and cause low resource utilization of cloud [14]. It is noteworthy that two-copies replication (e.g., PB model) can realize the balance between fault tolerance and resource utilization [16]. Dukowicz and Baumgardner [17] proposed an active-copy PB model, that is, the primary copy and backup copy can be executed at the same time to achieve fault tolerance. Qin *et al.* studied the passive-copy of PB model, if the primary copy fails, the backup copy will start to execute. In this mode, a task must have enough slack time for backup copy execution [18]. Further, Amoon [19] combined the ideas of the aforementioned two studies, proposing an adaptive sub-copy algorithm that is able to independently decide whether the backup copy execute with primary copy or not according to the task' slack time.

On the aspect of time, the main researches are focus on resubmission and checkpoint. Resubmission technology resubmits tasks to cloud when tasks fail so that tasks will be re-executed to meet the fault tolerance requirements. Plankensteiner *et al.* [11] pointed out that resubmission is a powerful fault-tolerant technology that can tolerate most of fails. However, resubmission technology needs tasks to have enough slack times, which allow tasks to execute several times, and it will reduce the resource utilization of cloud. Checkpoint is another wildly used technology to achieve fault tolerance. Checkpoint allows system to periodically record the running status of tasks and save it. If tasks fail, system will automatically check the record point close to the fail point, and then the task quickly rolls back to the record point and begins to re-execute. Cao *et al.* [20] studied the checkpoint technology in a heterogeneous cloud environment, which ensures fault tolerance for tasks with longer execution time and provides priority execution strategies for high priority tasks. Compared with resubmission technology, checkpoint technology requires less on task slack time. When a task fails, it will execute from the record point. And checkpoint technology improves the resource utilization of cloud.

In addition, Zhu *et al.* [21] proposed a fault-tolerant scheduling algorithm FASTER based on PB model and improved the resource utilization of cloud by using overlapping technologies. Poola *et al.* [22] proposed a fault-tolerant

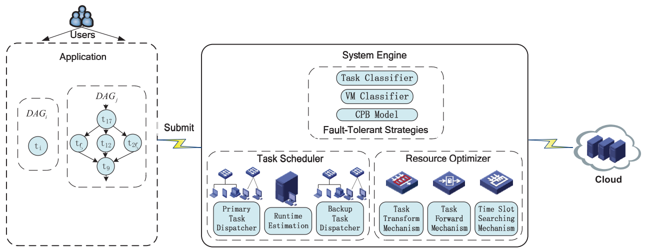


FIGURE 1. Fault-tolerant system architecture.

algorithm which can intelligently select the virtual machine for tasks. Zikos and Karatza [23] proposed a scheduling model for computing-intensive tasks based on the DVFS technology. Qian *et al.* [24] proposed a distributed framework with high fault tolerance. Zhou *et al.* [25] suggested to optimize the virtual machine locations to improve the fault tolerance of cloud.

There are two main distinctions between existing researches and ours. First, they do not consider hybrid real-time tasks characteristics in applications, these models cannot schedule the different types of tasks to appropriate virtual machines. Second, we combine the advantages of traditional PB (primary/backup) model and checkpoint technology, and propose a novel fault-tolerant model named ARCHER which achieves the balance between fault tolerance and resource utilization of cloud.

III. SYSTEM DESIGN

In this section, we give the system architecture, model and definitions used throughout this paper. And we analyze the classification model of hybrid tasks and VMs (virtual machines).

The fault-tolerant system architecture is presented in Fig. 1. The system engine is between users and cloud, it provides fault-tolerant mechanisms for tasks and schedules them to cloud.

In this paper, We use **D**irected **A**cyclic **G**raphs (DAGs) to represent applications. When users submit applications to cloud, task classifier will classify the tasks in DAGs, then the backup copies of task will be created. Task scheduler will schedule primary copies to VMs within the constraints, then computes tasks' makespan and DAGs' slack time, schedules backup copies according CPB model. The resource optimizer realizes the efficient use of resources through time slot exploiting mechanism, task forward mechanism and task transform mechanism.

CPB model will be introduced in Section IV, which integrates the advantages of traditional PB model and checkpoint technology, and provides a better fault-tolerant mechanism for tasks. Task scheduler and resource optimizer will be introduced in Section V.

A. TASK MODEL

An application can be denoted as a DAG $G = \{V(t), E(e)\}$, where V represents a set of tasks $V(t) = \{t_1, t_2 \ldots, t_n\}$, and t_n represents the n-th subtask of $V(t)$. E represents the edge set among tasks, $E(e) = \{e_{1,2}, e_{1,3} \ldots, e_{n-1,n}\} \subset V(t) \times V(t)$ defines the edge set among all tasks $e_{i,j} \subset t_i \times t_j(i \neq j \cap i < j)$.

Task t_i can be described as $t_i = (id_{t_i}, c_{t_i}, d_{t_i}, p_{t_i}, a_{t_i}, dl_{t_i}, i_{t_i}, o_{t_i})$, where id_{t_i} represents the identification of task, c_{t_i} represents the amount of calculation [26], d_{t_i} represents the amount of task' data, p_{t_i} indicates the priority of a task t_i, a_{t_i} is the arrival time of task t_i, dl_{t_i} represents the deadline of task t_i. i_{t_i} is the input required by the task, and o_{t_i} denotes the output result after the execution of task t_i.

$e_{i,j}$ in edge set $E(e)$ represents the relation between task t_i and task t_j. Edge $e_{i,j}$ can be described as $e_{i,j} = (id_{e_{i,j}}, c_{e_{i,j}}, d_{e_{i,j}})$, where $id_{e_{i,j}}$ represents the identification of edge $e_{i,j}$, $c_{e_{i,j}}$ represents the amount of data transferred between tasks, and $d_{e_{i,j}}$ denotes the connection between tasks t_i and t_j. $d_{e_{i,j}}$ is 0 if there is no relation between t_i and t_j, otherwise, $d_{e_{i,j}}$ is 1.

B. RESOURCE MODEL

We define $H = \{h_1, h_2, \ldots, h_n\}$ as a group of physical machines, where node h_i represents the i-th physical machine. h_i can be described as $h_i = (id_{h_i}, b_{h_i}, s_{h_i}, p_{h_i})$ where id_{h_i} represents the ID of physical machine node h_i. b_{h_i} represents the bandwidth of h_i, s_{h_i} represents the storage capability of h_i. p_{h_i} represents the CPU processing power of h_i, which is measured by MIPS.

Each physical machine h_i can host multiple virtual machines, denoted as $h_i = \{v_{h_i1}, v_{h_i2}, \ldots, v_{h_in}\}$, where v_{h_ij} represents the j-th virtual machine on the i-th physical machine, $v_{h_ij} = \{id_{v_{h_ij}}, b_{v_{h_ij}}, s_{v_{h_ij}}, p_{v_{h_ij}}\}$. Note that $\sum_{j=1}^{n} b_{v_{h_ij}} \leq b_{h_i}$, $\sum_{j=1}^{n} s_{v_{h_ij}} \leq s_{h_i}$.

After giving the task model and resource model. To facilitate the analysis, we propose some definitions needed in this paper.

Definition 1: Data Transmission Time (DTT): The time required to download all the data which are needed to execute a task. It depends on the amount of data and the transmission speed.

$$DTT(t_j) = d_{t_j}/ts(st, h(t_j)) + o_{P(t_i)}/ts(h(t_i), h(t_j)), \quad (1)$$

where d_{t_j} is the data needed for task t_j, $ts(st, h(t_j))$ is the data transmission speed, $o_{P(t_i)}$ is the result of task t_j's parent task, and $ts(h(t_i), h(t_j))$ is the transmission speed of data between two hosts.

Definition 2: Task Processing Time (TPT): The time to execute a task.

$$TPT(t_i) = c_{t_i}/p_{v_{h_ij}}, \quad (2)$$

where c_{t_i} is the amount of computation for task t_i (using MI as a measure), $p_{v_{h_ij}}$ is the CPU capability of the j-th virtual machine in the i-th host, which is measured by MIPS (Million of Instructions Per Second).

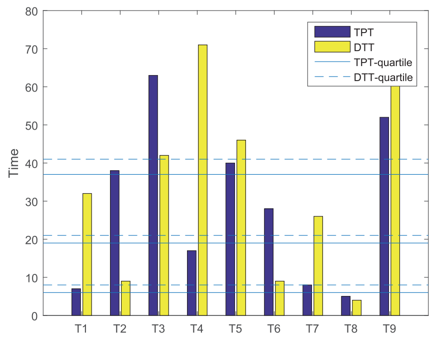


FIGURE 2. Task classification.

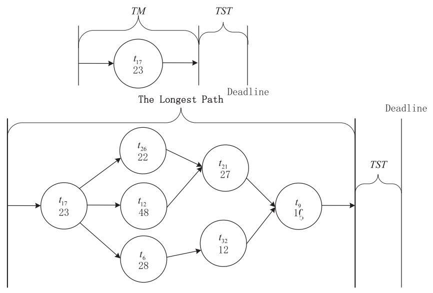


FIGURE 3. Schematic diagram of critical tasks.

Definition 3: Task Slack Time (TST): The time between a DAG's finish time and its deadline.

$$TST(DAG_i)=dl_{DAG_i}-max(\sum_{i=X}(DTT(t_i)+TPT(t_i))), \qquad (3)$$

where X involves every task ID in DAG' task chain which is the longest tasks execution sequence.

C. CLASSIFICATION OF TASKS AND VIRTUAL MACHINES

In order to improve the accuracy of task scheduling in this paper, we devise the task and virtual machine classification mechanism that improves resource utilization of cloud and reduces task execute time while guaranteeing fault tolerance. After classification, the task scheduler can accurately schedule tasks to virtual machines.

1) THE CLASSIFICATION OF TASKS

When tasks are submitted to a cloud, the task classifier will firstly divide these tasks into three types: computing-intensive tasks (CIT), data-intensive tasks (DIT) and balanced tasks (BT).

The classification of tasks is mainly based on the data transmission time and task processing time. As shown in Fig. 2, we choose 9 types of representative tasks and extract the quartiles of their DTT and TPT. We can clearly see that tasks t_3, t_5, and t_8 are the balanced tasks, whose DTT and TPT are similar. t_1, t_4, and t_7 are the data-intensive tasks, whose DTT are much larger than TPT. t_2, t_6, and t_9 are the computing-intensive tasks, whose DTT are smaller TPT.

2) THE CLASSIFICATION OF VIRTUAL MACHINE

The classification of virtual machines is similar to tasks. According to the data transmission speed (bandwidth) and CPU processing capability of virtual machines, the virtual machine can be divided into the following types: 1) DT virtual machines whose data transmission speed is higher than CPU manipulating ability. 2) CM virtual machines whose data transmission speed is lower than CPU manipulating ability. 3) Balanced virtual machines whose data transmission speed and CPU manipulating ability are balanced.

3) THE DIVISION OF CRITICAL TASK

In DAGs, there are some tasks whose makespans are longer than DAGs' slack time. So those tasks cannot execute two times before deadlines. Whether those tasks can be completed before their deadlines become the key to cloud.

Critical tasks are the tasks whose makespans are longer than DAGs' slack time, $TM(t_i) = DTT(t_i) + TPT(t_i) > TST(DAG)$. As shown in Fig. 3, critical tasks can be divided into two types: independent critical tasks and dependent critical tasks.

As shown in Fig. 3, $dl(DAG) - TM(t_{17}) = 20 = TST(DAG) < TM(t_{17})$, task t_{17} is an independent critical task. The Fig. 3 also shows the dependent key tasks in DAG. If $TST(DAG) = dl(DAG) - \sum_{i=X}^{17,12,21,9}(DTT(t_i) + TPT(t_i)) = 25$, t_6, t_{12} and t_{21} are dependent critical tasks in this case.

IV. FAULT-TOLERANT SCHEDULING MODEL

In this paper, we assume that at most one host fails at one time instant. Based on this hypothesis, we propose a novel fault-tolerant scheduling model CPB that combines the **C**heckpoint technology with traditional **P**rimary/**B**ackup (PB) model for hybrid tasks.

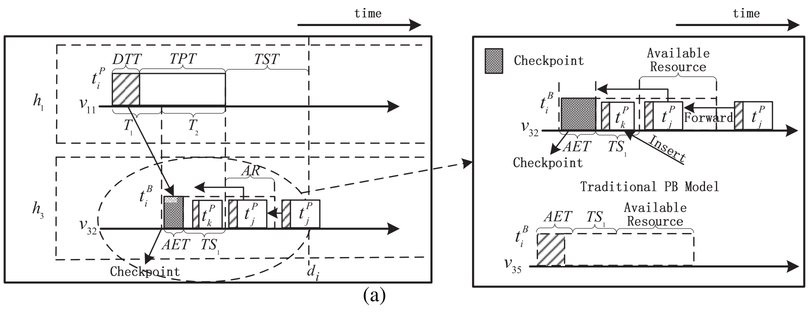
We divide the tasks scheduling of CPB model into two cases, according to tasks' makespan and its slack time. Then we give the general constraints of CPB model, we also analyze the detailed constraints of task scheduling according to the number of tasks in DAG.

Compared with the traditional fault-tolerant scheduling model, CPB model can control the execution times of backup copies and record this point. CPB model not only reduces consume of cloud resource but also creates more time slots for unscheduled tasks. Tasks scheduling in CPB model can be divided into two cases according to the tasks' makespans and task slack times.

A. WHEN $TM(t_i) > TST(t_i)$ IN CPB MODEL

Fig. 4 shows the case $TM(t_i) > TST(t_i)$. $TM(t_i)$ is task t_i's makespan, and $TM(t_i) = DTT(t_i) + TPT(t_i)$. Task t_i's slack time is $TST(t_i) = dl(DAG) - (DTT(t_i) + TPT(t_i))$.

In traditional PB model, backup copy t_i^B will be executed until its primary copy t_i^P completes. And primary copy t_j^P will be scheduled after t_i^B. A large amount of resources are



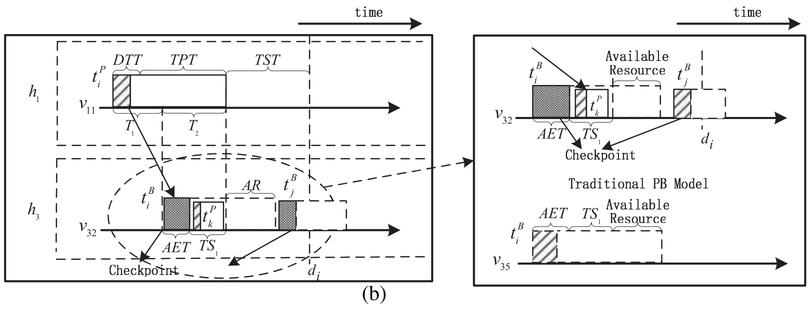


FIGURE 4. Examples of $TM(t_i) > TST(t_i)$ in CPB model.

occupied by backup copies whose primary copies are executed successfully. Importantly, these resources have not been effectively utilized.

Fig. 4 shows the same situation in CPB model. We can divide this situation into two cases. As shown in Fig. 4(a), backup copy t_i^B will only execute AET (Advance Execution Time) time, $AET(t_i) = TM(t_i) - TST(t_i)$, and then will be recorded as a point. The resource AR (Available Resource), which is occupied by backup copy in traditional PB model, becomes available in CPB model. So CPB model can take full use of the occupied resources in traditional PB model. If primary copy t_j^P satisfies Constraint 4 (we will explicate it on section 4.3.2), it will be scheduled forward. Primary copy t_j^P will start execute behind of t_i^P's finish time. There also exists an available time slot $TS = TST(t_i) - DT$, where DT is the system delay time. In this case, if primary copy t_i^P fails in T_1, backup copy t_i^B will execute and t_j^P will be re-scheduled to original position. If primary copy t_i^P fails in T_2, t_j^P will be recorded and the backup copy t_i^B will execute from the recorded point. If t_j^P does not satisfy Constraint 4, CPB model will find a unscheduled task which can complete in this time slot, and schedules it.

Fig. 4(b) shows the case that the next task is backup copy t_j^B in CPB model. In this case, t_j^B cannot execute forward, and CPB model will schedule a primary copy t_k^P to the time slot. If primary copy t_i^P fails in T_1, t_k^P will be re-scheduled to other VM and backup copy t_i^B will execute to achieve fault tolerance. If primary copy t_i^P fails in T_2, task t_k^P will be recorded and t_i^P will execute.

B. WHEN $TM(t_i) < TST(t_i)$ IN CPB MODEL

Fig. 5 shows the case that $TM(t_i) \leq TST(t_i)$.

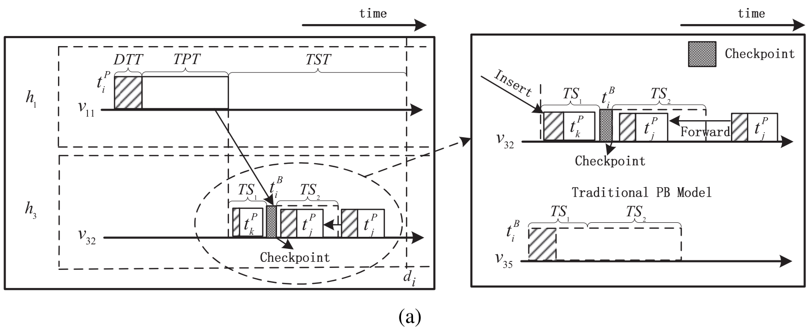
As shown in Fig. 5(a), compared with traditional PB model, in CPB model backup copy t_i^B will be scheduled after t_i^{P}' finish time and will be recorded as a point by employing checkpoint technology. There exist a time slot TS_1, CPB model will find a task that can complete in TS_1, and schedule it to this time slot. Primary copy t_j^P will execute forward if it satisfies Constraint 4, or CPB model will find a task that can complete in TS_2, and schedule it to TS_2. If primary copy fails, t_j^P will be re-scheduled to other VM, and backup copy t_i^B will execute.

Fig. 5(b) shows the case that the next task is backup copy t_j^B. Since backup copy t_j^B does not need to execute forward. CPB model will find an appropriate unscheduled task t_m^P and schedule it to this time slot. If primary copy t_i^P fails, t_m^P will be re-scheduled to other VM, and backup copy t_i^B will execute to achieve fault tolerance. As we presume that at most one host fails at the same time, according to Constraint 5 (we will explicate it on section 4.3.2). If primary copy t_i^P fails, primary copy t_j^P will execute successfully, and t_j^B will not execute so t_i^B will have enough execution time.

C. FAULT-TOLERANT SCHEDULING CONSTRAINTS FOR CPB MODEL

We analyze the constraints of tasks scheduling including primary and backup copies in CPB model to achieve fault tolerance and high resource utilization. In order to facilitate the subsequent analysis, we firstly introduce the definitions and basic constraints.

Definition 4: Earliest Start Time (EST): The earliest start time of task t_i depends on the performances of virtual machine where task t_i is scheduled to and t_i's parent tasks' parameters.



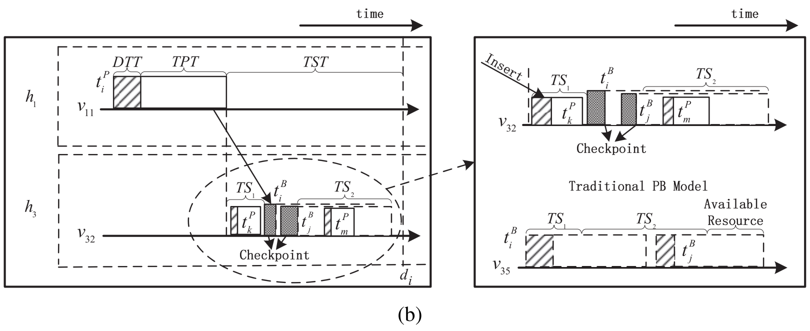


FIGURE 5. **Examples of $TM(t_i) \le TST(t_i)$ in CPB model.**

The earliest start time of primary copy t_i^P is:

$$est(t_i^P) = min\{eft(t_k) + SD(vm(t_i^P)) + DTT(t_i^P) - TF(vm(t_i^P))\}, \quad (4)$$

and if the virtual machine $vm(t_i^P)$ of primary copy t_i^P is failed, the earliest start time of backup copy t_i^B is:

$$est(t_i^B) = min\{TFT(t_i^P) + SD(vm(t_i^B)) + DTT(t_i^B)\}, \quad (5)$$

where $eft(t_k)$ is the earliest finish time of t_i's parent task t_k; If task t_i has no parent task, $eft(t_k) = 0$; $TFT(t_i^P)$ is the fail time of the primary copy t_i^P, and $SD(vm(t_i^B))$ is the delay of virtual machine $vm(t_i^B)$. $TF(vm(t_i^P))$ is the primary copy t_i^P's forward time if there exists an available time slot before t_i^P, if not, $TF(vm(t_i^B)) = 0$.

Definition 5: Earliest Finish Time (EFT): The earliest finish time of task t_i is defined as follows:

$$eft(t_i) = min\{est(t_i) + TPT(t_i)\}, \quad (6)$$

where $est(t_i)$ is the earliest start time of task t_i and $TPT(t_i)$ is the processing time of t_i.

Now, we analyze the basic constraints for CPB model.

Constraint 1: Primary copy t_i^P and backup copy t_i^B of task t_i cannot be scheduled to the same host h_i.

If primary copy t_i^P and backup copy t_i^B are scheduled to the same node h_i, when h_i fails, primary copy t_i^P and backup copy t_i^B will fail at the same time, so the system will not be fault tolerant.

Based on Constraint 1, according to the characteristics of CPB model, we analyze the scheduling constraints for DAGs.

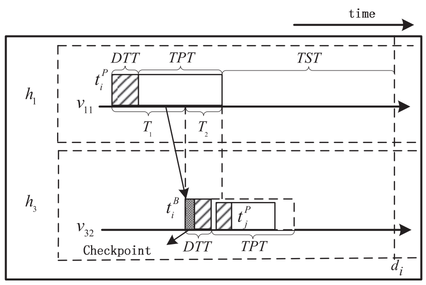


FIGURE 6. **Examples of common task in one task DAG scheduling constraints.**

1) CONSTRAINTS FOR DAG WITH ONE TASK

In this paper, we use **D**irected **A**cyclic **G**raphs (DAGs) to represent applications. There are some applications which have only one task, such as traffic stream analysis. We divide this situation into two cases.

Case 1: Task t_i is a critical task in an one-task DAG, Fig. 4 shows an example of this case.

Constraint 2: In this case, backup copy t_i^B's earliest start time needs to meet $est(t_i^P) < est(t_i^B) < eft(t_i^P)$.

As shown in Fig. 4, task t_i is a critical task satisfying $TM(t_i) > TST(t_i)$. If primary copy t_i^P fails and backup copy t_i^B's earliest start time $est(t_i^B) > eft(t_i^P)$, there is no enough time to execute t_i^B successfully before its deadline, it must ensure that t_i^B's earliest start time satisfies $est(t_i^B) < eft(t_i^P)$. In order to make effective use of resources of cloud, t_i^B's earliest start time must satisfy $est(t_i^P) < est(t_i^B)$.

Case 2: Task t_i is a task in an one-task DAG, Fig. 6 shows the example of this case.

Constraint 3: In this case, backup copy t_i^B's earliest start time needs to satisfy $eft(t_i^P) < est(t_i^B)$.

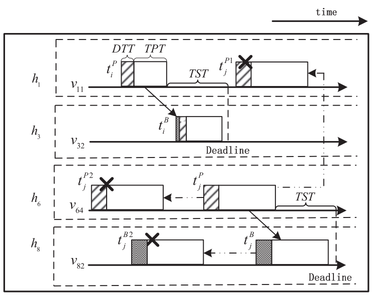


FIGURE 7. Examples of serial dependent task scheduling constraints.

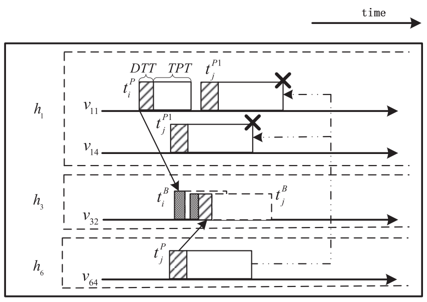


FIGURE 8. Examples of parallel dependent task scheduling constraints.

Since $TM(t_i) < TST(t_i)$, backup copy t_i^B will be recorded as a point to enhance resource utilization of cloud. As shown in Fig. 6, if t_i^B's earliest start time $eft(t_i^P) > est(t_i^B)$ and primary copy t_i^P fails, backup copy t_i^B will execute from the record point to achieve fault tolerance. And if primary copy t_j^P is scheduled behind t_i^B, t_j^P will be recorded until backup copy t_i^B completes, t_j^P may not complete before its deadline. So backup copy t_i^B's earliest start time needs to satisfy $eft(t_i^P) < est(t_i^B)$.

2) CONSTRAINTS FOR DAGs WITH DEPENDENT TASKS

The constraints of task scheduling in DAGs with dependent tasks are complicated. It not only needs to meet one task DAGs' scheduling constraints, but also needs to meet other scheduling constraints. Dependent tasks in DAGs can be divided into serial tasks and parallel tasks. In order to facilitate analysis, we introduce serial tasks and parallel tasks at first.

Serial tasks are those tasks that consist of parent tasks and children tasks in a DAG, e.g., t_{17}, t_{26}, t_{21}, and t_9 in Fig. 3. Parallel tasks are the tasks in different tasks chains, e.g., t_{26}, t_{12}, and t_6 in Fig. 3.

Case 1: Task t_i and task t_j are serial tasks, and task t_i is the parent task of t_j.

Constraint 4: Primary copies of task t_i and task t_j cannot be scheduled to the same host. Task t_i' primary copy and backup copy need to satisfy $eft(t_i^P) < est(t_j^P)$, $eft(t_i^B) < est(t_j^B)$.

As shown in Fig. 7, t_i is the parent task of t_j, when primary copies of t_i and t_j are scheduled to the same host h_l. If h_l fails, t_i^B and t_j^B will execute to ensure the finish of DAG. In CPB model, the execution time of backup copy is determined by the slack time of DAG. As shown in Fig. 7, task t_i is not a critical task, so t_i^B is recorded by checkpoint technology, t_j is a critical task and t_j^B will be recorded after executing $TM(t_j) - TST$. Since the execution time of a backup copy is determined by the slack time of DAG. So after t_i^B completes, the slack time of DAG has been used. If t_j^B needs to execute after t_i^B, there is no enough slack time.

Case 2: Task t_i and task t_j are parallel tasks in DAG.

Constraint 5: If primary copies t_i^P and t_j^P are scheduled to the same host, backup copies t_i^B and t_j^B cannot be scheduled to the same VM.

As shown in Fig. 8, if primary copies t_i^P, t_j^P are scheduled to the same host and these backup copies t_i^B, t_j^B are also scheduled to the same VM. When host h_1 fails, backup copy t_i^B and t_j^B will execute at the same time, and this will lead to time conflict.

V. FAULT-TOLERANT SCHEDULING ALGORITHM ARCHER

Based on the CPB model and aforementioned constraints, we propose a novel f**a**ult-tole**r**ant s**che**duling algorithm for hyb**r**id real-time tasks named ARCHER. ARCHER can be divided into two parts: one is task fault-tolerant scheduling based on CPB model, and the other is resource utilization improvement.

A. DAG FAULT-TOLERANT SCHEDULING

When users submit applications to cloud, all applications will be formed as DAGs, and hybrid real-time tasks in these DAGs will be scheduled to VMs.

Algorithm 1 Tasks Scheduling Algorithm

```
1  Classify all available virtual machines and tasks in
   DAG;
2  Monitor all virtual machines and DAG'deadline;
3  missTask← ∅;
4  while the task $t_i \in T$ not been scheduled do
5      success←schedulingPrimary($t_i^P$);
6      success←schedulingBackup($t_i^B$);
7      if !success then
8          if !success then
9              if there exit a task $t_k$, $t_k \in P(t_i)\&t_k \in$
               scheduedSuccess then
10                 if there exits a task $t_j$, $t_j \in c(t_i)$ then
11                     $c(t_i) \cup t_i \in$ missTask;
12                 else
13                     $t_i \in$ missTask;
14             else
15                 $t_k \in$ missTask;
16     Recycle all the reserved resources;
17     Re-calculate related parameters;
18     Re-scheduling the task ∈missTask;
```

The scheduling algorithm ARCHER firstly classifies virtual machines and tasks, monitors the deadlines of all tasks in DAGs. When the primary copy or backup copy of task t_i fails, it judges whether the task has parent tasks and children tasks or not. If t_i's parent tasks are scheduled successfully, t_i and its children tasks will be reclaimed (see Lines 4-12). The system will reclaim all the resources allocated to the tasks, recalculate the relevant parameters, and reschedule the tasks (see Lines 13-15).

ARCHER minimizes the waste of resources caused by task scheduling and decreases system response time. Considering that task scheduling is a NP-complete problem in real system, we use the heuristic algorithm to solve this problem.

we will show classification mechanism, primary copies pre-scheduling algorithm and backup copies pre-scheduling used in tasks scheduling algorithm.

1) TASKS AND VIRTUAL MACHINES CLASSIFY

After receiving DAGs submitted by users, ARCHER will classify all the tasks and available virtual machines, to ensure that tasks can accurately match with virtual machines. Algorithm 2 gives the pseudocode for task classification.

The first line calculates the data transmission time $DTT(t_i)$ and the task processing time $TPT(t_i)$.

$$DTT(t_i) = c_{t_i}/ts(st, h(t_i)) + r_{P(t_i)}/ts(h(t_i), h(t_j)), \quad (7)$$

$$TPT(t_i) = c_{t_i}/p_{v_{h_{ij}}}. \quad (8)$$

Tasks' data transmission time DTT and processing time TPT will be sorted, and system will take out the quartiles, $DTT(Q_1, Q_2, Q_3)$ and $TPT(Q_1, Q_2, Q_3)$. Tasks are classified according to the quartiles of DTT and TPT, and are divided into three types BT, CIT, and DIT (see Lines 2-12). Then, algorithm scans all tasks, puts the tasks that do not have parent tasks in $TDAG(S)$ and puts tasks that do not have children tasks in $TDAG(E)$ (see Lines 14-23). The classification method of virtual machines is similar to tasks, and the available virtual machines are divided into three types.

By employing classification algorithm, tasks and virtual machines are grouped in different types, which provides the basis for accurately matching tasks to virtual machines. It improves resource utilization of cloud and decrease response time.

2) PRIMARY COPY PRE-SCHEDULE

In order to ensure that all tasks can successfully complete before their deadlines, ARCHER will minimize the execution time of backup copy to provide more resources for tasks. Algorithm 3 shows the detailed process of pre-scheduling of the primary copy.

Algorithm 3 scans tasks and available virtual machines in different groups (see Lines 1-3). Based on Constraint 4, algorithm sorts tasks and matches them with virtual machine v_{kl}, and calculates the earliest finish time $eft^P_{i_{kl}}$. The primary copy t_i^P of task t_i will be scheduled to the virtual machine v_{kl} that has the earliest finish time eft_i^P (see Lines 4-13) to ensure that DAGs can be completed in the earliest time. After all the tasks in the $TDAG(s)$ are scheduled,

Algorithm 2 Tasks and Virtual Machines Classification

```
1  Statistics of all tasks' data transmission time (DTT) and
   task Processing time (TPT);
2  Sorting DTT and TPT then find out quartiles of them;
3  missTask ← ∅;
4  for $t_i \in T$ do
5      if $DTT(t_i) > DTT(Q_1) \& TPT(t_i) > TPT(Q_1)$ then
6          $T(Bt) \leftarrow t_i$;
7      if $DTT(t_i) > DTT(Q_1) \& TPT(t_i) < TPT(Q_2)$ then
8          $T(DIt) \leftarrow t_i$;
9      if $DTT(t_i) < DTT(Q_2) \& TPT(t_i) < TPT(Q_2)$ then
10         $T(CIt) \leftarrow t_i$;
11     else
12         $T(Bt) \leftarrow t_i$;
13         Return $T(BT) \& T(CIT) \& T(DIT)$;
14 Classify virtual machines by its compute capacity and
   bandwidth;
15 Sorting all the tasks belong to $T(BT)T(CIT)T(DIT)$;
16 while $!all\ tasks\ t_a \in T\ have\ been\ scanned$ do
17     for $t_a \in T$ do
18         if $t_a$ have $e_{a,b}$ satisfies $d_{e_{a,b}} > 0$ then
19             $TDAG \leftarrow TDAG \cup \{t_a\}$;
20             for $t_i \in TDAG$ do
21                 if $t_i$ have $e_{a,b}$ satisfies $d_{e_{i,j}} > 0 \&\& t_i$
                   have not $e_{k,i}$ then
22                     $TDAG(S) \leftarrow TDAG \cap \{t_i\}$;
23                 if $t_m$ have $e_{m,n}$ satisfies $d_{e_{m,n}} > 0 \&\& t_n$
                   have not $e_{n,p}$ then
24                     $TDAG(E) \leftarrow TDAG \cap \{t_m\}$;
25                 else
26                     $TDAG(T) \leftarrow TDAG \cap \{t_i\}$;
27 Return all task sets:$T(Bt)$, $T(CIt)$, $T(DIt)$, $TDAG(S)$,
   $TDAG(E)$, $TDAG(T)$;
```

tasks in $TDAG(T)$ will be scheduled to virtual machines and calculate $eft^P_{j_{mn}}$ (see Lines 14-22). Finally, Algorithm 3 checks whether each DAG can be completed before its deadline. If not, the task forward and transform mechanism will be employed (see Lines 23-28).

Through primary copy pre-scheduling, it will not only ensure the completion of tasks, but also improve the utilization of resources.

3) BACKUP COPY PRE-SCHEDULE

The scheduling of backup copies has more constraints than primary copies. However, the scheduling of backup copy has a strong connection with primary copy, so ARCHER will schedule backup copy according to the scheduling of primary copy. Algorithm 4 gives the pseudocode for backup copies scheduling.

Algorithm 3 Primary Copy (t_i^P) Schedule

```
1  Sorting all the tasks DAG belong to $T(BT)T(CIT)T(DIT)$;
2  Search and sort all the hosts $\in H_a$ which are available;
3  $H^p_{available} \leftarrow$top $\alpha\%$ available hosts $\in H_a$;
4  missTask$\leftarrow \emptyset$;
5  while !all task$\in$ TDAG(s) or independent task have been scheduled do
6      According to the sorting task out the host $h_k$ and task $t_i$ from different groups;
7      if $h_k \in H^p_{available}$ satisfies $t_i^{p'}$ s scheduling constraints of Constraint1 and Constraint4 then
8          for $v_{kl}$ in $h_k$'VmList do
9              Calculate the earliest finish time $eft^P_{i_{kl}}$ based on Eq.(6),Definition2;
10             if $v_{kl}$ have the earliest finish time of $v_{kl}$'VmList then
11                 Pre-Schedule $t_i^P$ to $v_{kl}$;
12                 Update the $H_{available}$ and $TDAG(s)$;
13                 Return $eft^P_{i_{kl}}$;
14 while !all task$\in TDAG(T)$ have been scheduled do
15     for $v_{mn}$ in $h_m$'VmList do
16         Calculate the earliest start time $est^P_{j_{mn}}$ based on Eq.(4),Definition1;
17         $est^P_{j_{mn}} = eft^P_{i_{kl}} + DTT(t_j^P) + SD$;
18         $eft^P_{j_{mn}} = est^P_{j_{mn}} + TPT(t_j^P) + SD$;
19         if $v_{mn}$ have the earliest finish time of $h_m$'VmList then
20             Pre-Schedule $t_j^P$ to $v_{mn}$;
21             Update the $H_{available}$ and $TDAG(T)$;
22             Return $eft^P_{j_{mn}}$;
23 for task $t_m \in TDAG(E)$ do
24     if $eft^P_{m_{xy}}$ >deadline(TDAG($t_i$)) then
25         Using the Task Forward Mechanism;
26     else
27         Schedule all tasks $t_i \in TDAG$;
28         Calculate task slack time $TST(t_i)$ based on Definition3;
29 Return $est^P_{j_{mn}}$, $eft^P_{j_{mn}}$, $TST(t_i)$;
30 Return true;
```

Algorithm 4 Backup Copy Scheduling

```
1  while !all $t_i^B \in$ TDAG or independent task $t_i^B$ have been scheduled do
2      Search and sort all the hosts$\in H_a$ which after $eft^P_{i_{mn}}$ are available;
3      $H^B_{available} \leftarrow$top $\beta\%$ available hosts $\in H_a$;
4      According to the sorting take out the host $h_k$ and tasks $t_i^B$ from different groups;
5      if $h_k \in H^B_{available}$ satisfies $t_i^{B'}$ s scheduling constraints of Contraint1, Contraint3 and Contraint4 then
6          for $v_{kl} \in h_k$ do
7              Calculate the earliest finish time $eft^B_{i_{kl}}$ based on Eq.(6),Definition6;
8              if $v_{kl}$ have the earliest finish time then
9                  Pre-Schedule $t_i^B$ to $v_{kl}$;
10                 Update the $H^B_{available}$ and TDAG;
11                 Calculate data transmission time($DTT$) based on Definition1;
12                 Calculate task process time($TPT$) based on Definition2;
13                 $TM(t^B_{i_{kl}}) = DTT(t_i^B) + TPT(t_i^B)$;
14                 if $TM(t^B_{i_{kl}}) > TST(t_i)$ then
15                     Task $t^B_{i_{kl}}$ will be scheduled based on constraints for critical tasks in CPB model;
16                 else
17                     Task $t^B_{i_{kl}}$ will be scheduled based on constraints for common tasks in CPB model;
18                 Task $t^B_{i_{kl}}$ will be executed based on CPB model;
```

Similar to Algorithm 3, Algorithm 4 scans and finds available virtual machines after the primary copy' the earliest start time $est^P_{i_{mn}}$ (see Lines 2-3). Then Algorithm 4 finds the virtual machine that has the earliest finish time $eft^B_{i_{kl}}$, and pre-schedule t_i^B on this virtual machine v_{kl} (see Lines 4-9). Finally it calculates task makespan $TM(t^B_{i_{kl}})$ of backup copy and compares it with the task slack time calculated in Algorithm 3, task $t^B_{i_{kl}}$ will execute according to CPB model (see Lines 11-17).

B. RESOURCE UTILIZATION IMPROVEMENT

In order to effectively utilize time slots which are created by CPB model, we design time slot exploiting mechanism and task forward and transform mechanism to improve the resource utilization of cloud.

1) TIME SLOT EXPLOITING MECHANISM

By employing CPB model, ARCHER can provide more available time slots. Efficient use of those time slots can greatly improve resource utilization of cloud while guaranteeing fault tolerance. Algorithm 5 is the pseudocode for the time slot exploiting mechanism.

After primary copies are pre-scheduled, some DAGs cannot complete before their deadlines. In this case, the tasks

Algorithm 5 Time Slot Exploiting Mechanism

```
1  $T^S_{available}$ ←the time slots which are available;
2  $T_{insert}$ ←Sort tasks which can use time slot by critical degree;
3  while !all time slot in $T_{available}$ have been scheduled do
4      for task $t_i \in T_{insert}$ do
5          if time slot'VM $v_{kl}$ satisfies $t_i$'s scheduling constraints then
6              Calculate the earliest start time $est_{t_i}$ based on Eq.(4), Definition4;
7              $est_{t_i} = DTT(t_i) + SD$;
8              $eft_{t_i} = est_{t_i} + TPT(t_i)$;
9              if $eft_{t_i} < t_i$'deadline then
10                 Schedule $t_i$ to $v_{kl}$;
11                 Calculate the surplus time slot $STS$;
12                 $STS = TS - eft_{t_i} + est_{t_i}$;
13                 Add surplus time slot to $T^S_{available}$;
14             else
15                 Use task transform mechanism;
16                 Re-calculate the earliest finish time $eft_{t_i}$;
17                 if $eft_{t_i} < t_i$'deadline then
18                     Schedule $t_i$ to $v_{kl}$;
19                     Calculate the surplus time slot $STS$;
20                     Add surplus time slot to $T^S_{available}$;
21         else
22             Break;
```

Algorithm 6 Task Forward and Transform Mechanism

```
1  while !all task $t_i$ complete successful do
2      if task $t_i$'primary copy fails then
3          Recycle all the reserved resources of $t^P_i$;
4          if following task $t_j$ is primary copy and satisfies constraint4 then
5              Task $t_j$ has been processed forward;
6              Re-Calculate task $t_j$'earliest start time;
7              $est_j = eft^P_i + TR(t^P_i) + DTT(t_j)$;
8              $eft_j = est_j + TPT(t_j)$;
9          else
10             Use time slot exploiting mechanism;
11         if task $t_i$'primary copy complete successful then
12             Recycle all the reserved resources of $t^P_i$;
13             Process as primary copy fails;
14 while !all task $t_i$ can be processed in its type of virtual machines do
15     if VM $v_{kl}$ processing the CIT task $t_i$ then
16         Choose DIT critical task $t_j$;
17         Calculate $DTT(t_j)$ while $t_i$ be processed;
18         if task $t_j$'earliest finish time $eft_j$ < deadline then
19             Allocation $t_i$ to $t_i$;
20         else
21             Download the next task $t_j$'data;
22             Refresh $est_j$ and $eft_j$;
```

in a DAG will be re-scheduled and critical tasks will be scheduled to available time slots to solve this problem. Before scheduling critical tasks into time slots, it is necessary to know whether the time slot satisfies the constraints for fault tolerance. The time slot exploiting mechanism will calculate whether $eft_{t_i} < dl_{t_i}$. If it satisfies the constraints, Algorithm 5 will schedule t_i to v_{kl}, calculates the remaining resources in this time slot, and adds it to $T^S_{available}$ (see Lines 5-13). If $eft_{t_i} > dl_{t_i}$, Algorithm 5 will employ task transform mechanism.

2) TASK FORWARD AND TRANSFORM MECHANISM

Task forward and transform mechanism will improve resource utilization of cloud while guaranteeing fault tolerance. Task forward mechanism will be employed to advance the execution of tasks if there exist available time slots and tasks satisfy the constraints. Task transform mechanism focuses on the transformation of different task types to shorten critical tasks' makespan.

As shown in Algorithm 6, if primary copy t^P_i fails, and the next task is a primary copy t^P_j that meets Constraint 4, t^P_j will execute forward. If the next task is a backup copy, it will be recorded as a point by using checkpoint technology, ARCHER will find a task that can complete in this time slot and schedule it to this time slot (see Lines 3-11). The task transform mechanism can transform *DIT* tasks to simple compute tasks or reduce *CIT* tasks' complexity. If there is a tight time slot behind a *CIT* task or a *DIT* task, and ARCHER cannot find an available task, task transform mechanism will be employed. For the *CIT* task t_i, ARCHER will use its compute time to download a *DIT* task t_j's data. When the data download completes, t_i will be transformed to a simply *BT* task. For the *DIT* task, ARCHER will use its data transmission time to compute a *CIT* task (see Lines 16-21). Task transform mechanism can also be used to reduce system response time for a critical task.

VI. PERFORMANCE EVALUATION

In order to verify the performance of the proposed ARCHER, extensive simulation experiments were conducted. We compare ARCHER with Non-Resource-Utilization-ARCHER (NRUARCHER), Non-CPB-Model-ARCHER (NCMARCHER), to verify the effectiveness of task classification, CPB model and resource utilization

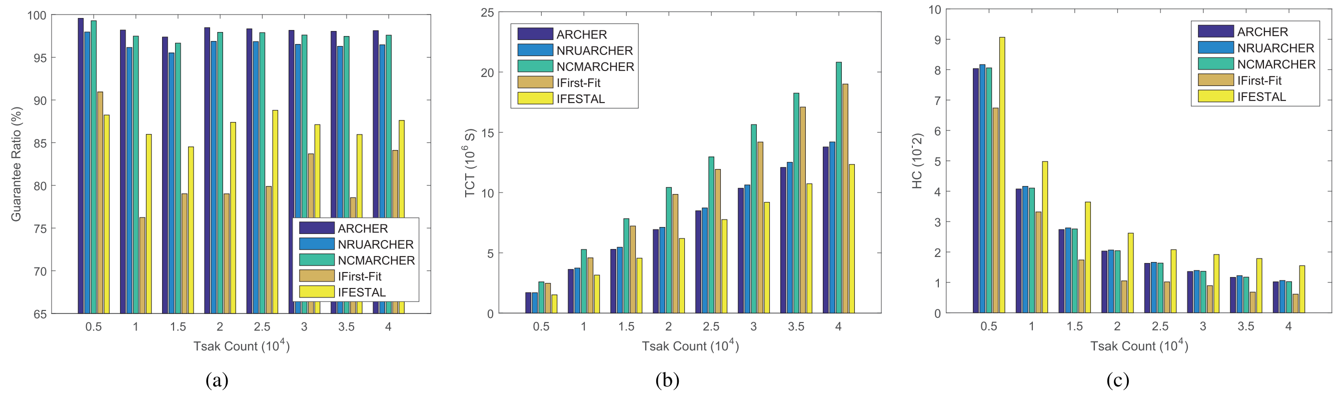


FIGURE 9. Performance impact of task count.

improvement. Also, we compare ARCHER with improved classical First-Fit (IFirst-Fit) algorithm [27] and improved FESTAL (IFESTAL) algorithm [21]. The main differences between ARCHER and other algorithms are as follows:

- NRUARCHER lacks resource utilization improvement compared with ARCHER, and the comparison of NRUARCHER can verify the effectiveness of resource utilization improvement mechanism.
- NCMARCHER employs traditional PB model compared with ARCHER. It cannot flexibly adjust the execution time of backup copies. The comparison with NCMARCHER can verify the superiorities of the CPB model.
- IFirst-Fit algorithm is an improved classic First-Fit algorithm. IFirst-Fit gives priority to schedule tasks to the nearest hosts (the nearest neighbor scheduling mechanism). The comparison between ARCHER and IFirst-Fit is used to verify the fault tolerance and resource utilization of ARCHER.
- IFESTAL algorithm is an improved fault-tolerant scheduling algorithm for real-time scientific workflows.

The following three metrics are used to verify the performance of these algorithms.

- Guarantee Ratio (GR): The percentage of DAGs that successfully complete before their deadlines.
- Task Complete Time (TCT): The time consumed by cloud to complete all DAGs.
- Host Consume (HC): The ratio of active hosts' number over the number of completed tasks. It represents the resource consumption of each task.

We choose Cloudsim as a simulation platform, which is widely used in academia and industry, and conduct experiments. CloudSim is able to provide cloud environment under almost all the necessary testing interface, thus it is a suitable platform to verify the algorithms. The detailed parameters are as follows:

Hosts' processing capacities are randomly selected from 1000 MIPS, 2000 MIPS, 3500 MIPS, 4000 MIPS (Million of Instructions Per Second), and the bandwidths are 1 Gbps, 2 Gbps or 5 Gbps. The virtual machines' processing capacities are randomly selected form 250 MIPS, 500 MIPS, 750 MIPS, to 1000 MIPS, respectively. The bandwidth are 500 Mbps, 1000 Mbps, 1300 Mbps. The arrival times of tasks are set as the Poisson distribution to simulate the real users who submit applications.

TABLE 1. Parameters of experimental variable.

Parameter	Value (Fixed)-(Min, max, step)
Task Count(10^4)	(1)-(0.5,4,0.5)
Task Size($\times 10^3$MI, Mpbs)	([20-30]-(16,40,3))
Host Count	(300)-(190,400,30)

In order to verify the performance of algorithms under different parameters, we change task count, task size and initialize host count while the other parameters are fixed. Table 1 shows the values of the parameters.

A. THE IMPACT OF TASK COUNT

In this section, we test the performances of the five algorithms with the change of task count from 5000-40000.

In Fig. 9(a), it can be seen that with the increase of task count, ARCHER, NRUARCHER and NCMARCHER maintain a high guarantee ratio (more than 95%). This is because those algorithms can continue to search for appropriate virtual machines and schedule tasks to them for high guarantee ratio. However, NCMARCHER does not employ CPB model, the high guarantee ratios are achieved by consuming a large amount of resources as Fig. 9(b) demonstrates. As shown in Fig. 9(a), the guarantee ratios of IFirst-Fit and IFESTAL are obviously lower than other three algorithms because the nearest neighbor scheduling mechanism of IFirst-Fit brings excessive tasks for hosts with increase of task count, which causes that the finish time of many tasks is later than their deadlines. IFESTAL employs task overlap mechanism, I-AEAP (improved as early as possible) strategy and I-ALAP (improved as late as possible) strategy to have a higher utilization of resources. But with the increase of task count, the complexity of task scheduling is increased, causing conflicts between scheduled tasks, so the guarantee ratio of IFESTAL is low.

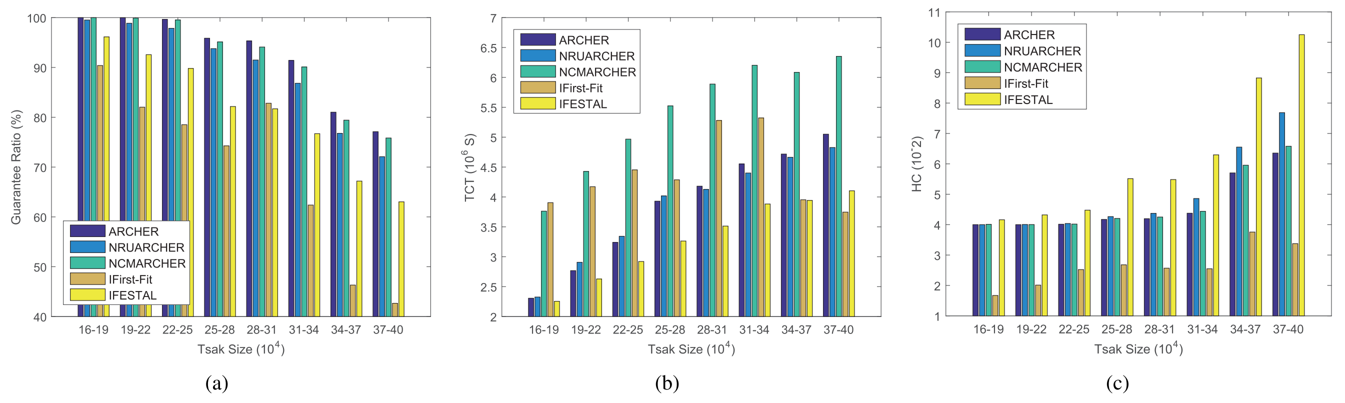


FIGURE 10. **Performance impact of task size.**

As shown in Fig. 9(b), task complete time of these five algorithms increases with the task count increases but TCTs of NCMARCHER and IFirst-Fit are significantly higher than other algorithms. Because NCMARCHER does not employ CPB model, it cannot make batter decisions on backup copies' execution time. IFirst-Fit cannot make full use of virtual machines' idle resources and cannot control the execution time of backup copies with the nearest neighbor scheduling mechanism, thus their TCTs are higher than others. TCT of IFESTAL is lower than ARCHER, there are two reasons. The first one is that IFESTAL uses task overlap mechanism, and another one is that IFESTAL' GR is much lower, a large number of tasks cannot complete, thus the actual TCT of IFESTAL will not be lower than ARCHER.

Fig. 9(c) shows the change of HC with the increase of task count. We can find that HC is significant reduced and drop to a lower level with the increase of task count. With the increase of task count, ARCHER' HC is lower than NRUARCHER and NCMARCHER. HC for IFirst-Fit is always at a low level because IFirst-Fit is more likely to schedule tasks to the nearest hosts. HC of IFESTAL is higher than other algorithms because with the increase of task count, IFESTAL will increase host number to schedule tasks.

B. PERFORMANCE IMPACT OF TASK SIZE

In this subsection, we verify the performance of algorithms with the change of task size. The results of experiment are shown in Fig. 10.

It can be seen from Fig. 10(a) that the GR of each algorithm decreases with the increase of task size. Due to the increase of task size, each DAG's slack time significantly reduces. If a task' primary copy fails, its backup copy does not have enough time to execute successfully. We can also observe that with the increase of task size, ARCHER, NRUARCHER and NCMARCHER have higher GRs than other two algorithms. However, the GR of ARCHER is significantly higher than NRUARCHER, since NRUARCHER dose not employ resource utilization improvement mechanism, so it cannot make full use of time slot. Since *TM* of tasks increases, a lot of tasks cannot complete before their deadlines by IFirst-Fit, so the GR of IFirst-Fit is low. When task size increases, because IFESTAL uses PB model, backup copies will take up a large amount of resource. IFESTAL cannot find appropriate virtual machines to schedule tasks, the only way is to start new hosts and virtual machines. However, since the start up of hosts and virtual machines needs some time, tasks that have tight slack time cannot complete before their deadlines, so the GR of IFESTAL is low.

As shown in Fig. 10(b), TCT increases with the increase of tasks size. Compared with ARCHER and NRUARCHER, since NCMARCHER employs traditional PB model, it needs to consume a large amount of resources to guarantee system fault tolerance. The TCT of NCMARCHER is significantly higher than other two algorithms. The trend of IFirst-Fit' TCT is irregular, it increases first and then decreases. With the increase of task size, TCT of IFirst-Fit increases, which is an inevitable trend. However, due to IFirst-Fit' nearest neighbor scheduling mechanism, it is not able to flexibly adjust resources. When the critical value of task size is reached, the GR of IFirst-Fit will be substantially reduced, TCT of IFirst-Fit will also reduce. For IFESTAL, its overlapping mechanisms reduces task completion time, but it should be noted that its GR is low. As shown in Fig. 10(c), IFESTAL' lower TCT needs to consume more hosts.

We can see from Fig. 10(c) that with the increase of task size, Host Consumes of algorithms continue to rise except IFirst-Fit. In those four algorithms, IFESTAL' HC increases quickly. This is because with the increase of tasks size, IFESTAL cannot maximize the use of existing resource, so it must add new hosts to satisfy task scheduling requirements. HCs of ARCHER, NCMARCHER, NRUARCHER are similar at beginning, but with the increase of tasks size, HC for NRUARCHER is significantly higher than ARCHER, because NRUARCHER adds more hosts to schedule tasks. HC of IFirst-Fit algorithm is at a low level, which is related to IFirst-Fit that likely schedules tasks to the nearest hosts, and maintains a lower level of HC.

C. PERFORMANCE IMPACT OF HOST COUNT

In this subsection, we analyze the impact of initialize host count on performance of each algorithm. The count of

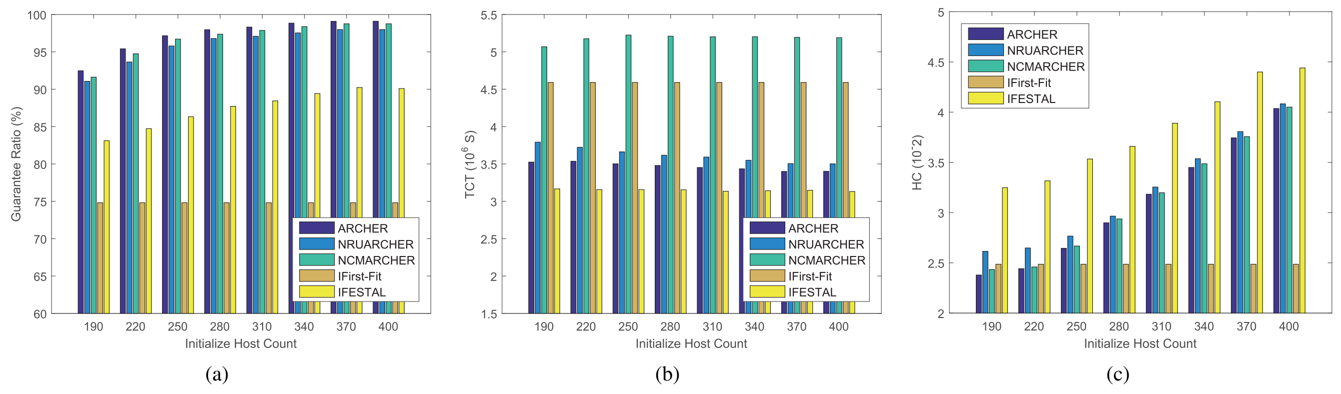


FIGURE 11. **Performance impact of host count.**

initialize host (available hosts when the tasks arrive) gradually increases from 190 to 400, and the results of the experiments are shown in Fig. 11.

Fig. 11(a) shows the effectiveness of initial host count on GR. It can be seen from the figure that the guarantee ratios of all algorithms increase steadily with the increase of initial host count except IFirst-Fit. This is because with the increase of initial host count, more virtual machines are available to ARCHER, tasks that have tight slack time can be executed on time, so tasks can be successfully completed. In those four algorithms except IFirst-Fit, it is clear that the GR of ARCHER is the highest, although GRs of NRUARCHER and NCMARCHER are close to ARCHER, their resource utilizations are low (see Fig. 11(b)). For IFirst-Fit, the count of hosts has little impact on guarantee ratio due to its nearest neighbor scheduling mechanism.

As shown in Fig. 11(b), the increase of initial host count has little impact on task complete time. Since various parameters of tasks are not changed, the time required to complete tasks is not reduced. It can be clearly seen from Fig. 11(b) that the task complete time of NCMARCHER and IFirst-Fit are much higher than other algorithms. For NCMARCHER, it cannot control backup copies' execution. For IFirst-Fit, because its nearest neighbor scheduling mechanism, it causes hosts to have large number of tasks and tasks cannot be executed on time. Among of ARCHER, NRUARCHER and IFESTAL, it is clear that the IFESTAL algorithm has the earliest task complete time because it consumes a large amount of host resources and has lower guarantee ratio (see Figs. 11(a)(c)).

Fig. 11(c) shows the impact of initial host count on HC. It can be seen that with the increase of initial host count, HC of those algorithms increases except IFirst-Fit. Because ARCHER, NCMARCHER, NRUARCHER and IFESTAL schedule tasks to more hosts to improve guarantee ratio. Since IFirst-Fit tends to schedule tasks to the nearest hosts, so there is no obvious changes in HC. HC of IFESTAL is higher than other algorithms, especially, under the condition of fewer hosts number. This is because IFESTAL prefers to add new hosts and virtual machines to meet the requirements of tasks.

VII. CONCLUSIONS AND FUTURE WORK

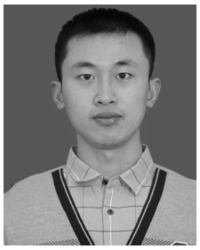
This paper investigates the problem of fault-tolerant scheduling for hybrid tasks in cloud. We establish a new fault-tolerant scheduling model, CPB, and propose a dynamic scheduling algorithm, ARCHER. The goal of this paper is to improve resource utilization of cloud while guaranteeing its fault tolerance. Through classification of tasks and virtual machines, tasks can be accurately matched to virtual machines in cloud. CPB model provides more available time slots thus the resource utilize of system is improved while guaranteeing fault tolerance. In order to verify the performance of ARCHER, we conduct extensive experiments and compare our algorithm with four baseline algorithms: NCMARCHER, NRUARCHER, IFirst-Fit and IFESTAL. The comparison results show that ARCHER is superior to others.

In our future research, we will extend our CPB model to tolerate multiple hosts' fails. A multiple of data storage methods will be considered to tolerate data corruption. We will also develop a prediction model, and analyze hosts system log to schedule tasks more accurately.

REFERENCES

[1] M. Armbrust *et al.*, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, 2010.
[2] M. Parashar and C. Stewart, "XSEDE cloud survey report," Cloud Secur., Tech. Rep., 2013.
[3] D. Sun, G. Chang, C. Miao, and X. Wang, "Analyzing, modeling and evaluating dynamic adaptive fault tolerance strategies in cloud computing environments," *J. Supercomput.*, vol. 66, no. 1, pp. 193–228, 2013.
[4] M. Mao and M. Humphrey, "A performance study on the VM startup time in the cloud," in *Proc. IEEE Int. Conf. Cloud Comput.*, Jun. 2012, pp. 423–430.
[5] S. Hwang and C. Kesselman, "Grid workflow: A flexible failure handling framework for the grid," in *Proc. IEEE Int. Symp. High Perform. Distrib. Comput.*, Jun. 2003, pp. 126–137.
[6] F. C. Gärtner, "Fundamentals of fault-tolerant distributed computing in asynchronous environments," *ACM Comput. Surv.*, vol. 31, no. 1, pp. 1–26, 1999.
[7] W. Cirne, F. Brasileiro, D. Paranhos, L. F. W. Góes, and W. Voorsluys, "On the efficacy, efficiency and emergent behavior of task replication in large distributed systems," *Parallel Comput.*, vol. 33, no. 3, pp. 213–234, 2007.
[8] J. Lu and D. Li, "Bias correction in a small sample from big data," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 11, pp. 2658–2663, Nov. 2013.
[9] J. M. Tien, "Big data: Unleashing information," *J. Syst. Sci. Syst. Eng.*, vol. 22, no. 2, pp. 127–151, 2013.

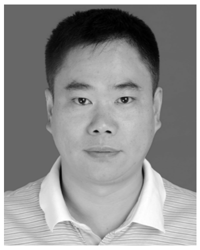
[10] A. Chervenak *et al.*, "Data placement for scientific applications in distributed environments," in *Proc. IEEE/ACM Int. Conf. Grid Comput.*, Sep. 2007, pp. 267–274.
[11] K. Plankensteiner, R. Prodan, T. Fahringer, A. Kertész, and P. Kacsuk "Fault detection, prevention and recovery in current grid workflow systems," in *Proc. Workshop Grid Services Evol.*, 2009, pp. 1–13.
[12] T. Xie and X. Qin, "Security-aware resource allocation for real-time parallel jobs on homogeneous and heterogeneous clusters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 5, pp. 682–697, May 2008.
[13] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "A performance analysis of EC2 cloud computing services for scientific computing," in *Proc. Int. Conf. Cloud Comput.*, 2009, pp. 115–131.
[14] 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. (IPDPS)*, Apr. 2008, pp. 1–8.
[15] G. Kandaswamy, A. Mandal, and D. A. Reed, "Fault tolerance and recovery of scientific workflows on computational grids," in *Proc. 8th IEEE Int. Symp. Cluster Comput. Grid*, May 2008, pp. 777–782.
[16] R. Al-Omari, A. K. Somani, and G. Manimaran, "An adaptive scheme for fault-tolerant scheduling of soft real-time tasks in multiprocessor systems," *J. Parallel Distrib. Comput.*, vol. 65, no. 5, pp. 595–608, 2001.
[17] J. K. Dukowicz and J. R. Baumgardner, "Incremental remapping as a transport/advection algorithm," *J. Comput. Phys.*, vol. 160, no. 1, pp. 318–335, 2000.
[18] X. Qin and H. Jiang, "A novel fault-tolerant scheduling algorithm for precedence constrained tasks in real-time heterogeneous systems," *Parallel Comput.*, vol. 32, nos. 5–6, pp. 331–356, 2006.
[19] M. Amoon, "A fault-tolerant scheduling system for computational grids," *Comput. Elect. Eng.*, vol. 38, no. 2, pp. 399–412, 2012.
[20] J. Cao, M. Simonin, G. Cooperman, and C. Morin, "Checkpointing as a service in heterogeneous cloud environments," in *Proc. IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, May 2015, pp. 61–70.
[21] X. Zhu, J. Wang, H. Guo, D. Zhu, L. T. Yang, and L. Liu, "Fault-tolerant scheduling for real-time scientific workflows with elastic resource provisioning in virtualized clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 12, pp. 3501–3517, Dec. 2016.
[22] D. Poola, K. Ramamohanarao, and R. Buyya, "Enhancing reliability of workflow execution using task replication and spot instances," *ACM Trans. Auto. Adapt. Syst.*, vol. 10, no. 4, 2016, Art. no. 30.
[23] S. Zikos and H. D. Karatza, "Performance and energy aware cluster-level scheduling of compute-intensive jobs with unknown service times," *Simul. Model. Pract. Theory*, vol. 19, no. 1, pp. 239–250, 2011.
[24] Z. Qian *et al.*, "TimeStream: Reliable stream computation in the cloud," in *Proc. ACM Eur. Conf. Comput. Syst.*, 2013, pp. 1–14.
[25] A. Zhou *et al.*, "Cloud service reliability enhancement via virtual machine placement optimization," *IEEE Trans. Serv. Comput.*, vol. 10, no. 6, pp. 902–913, Nov./Dec. 2017.
[26] T. Bhaskar and U. D. Kumar, "A cost model for N-version programming with imperfect debugging," *J. Oper. Res. Soc.*, vol. 57, no. 8, pp. 986–994, 2006.
[27] R. F. Freund *et al.*, "Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet," in *Proc. Heterogeneous Comput. Workshop*, Mar. 1998, pp. 184–199.



HAORAN HAN received the B.S. degree in mechanical engineering and automation from Northeastern University, Shenyang, China, in 2016. He is currently pursuing the M.S. degree with the College of Systems Engineering, National University of Defense Technology, Changsha, China. His research interests include cloud computing, information system, and edge computing.

WEIDONG BAO received the Ph.D. degree in information systems from the National University of Defense Technology, Changsha, China, in 1999. He is currently a Full Professor with the College of Systems Engineering, National University of Defense Technology. He has published over 100 research articles in refereed journals and conference proceeding such as IEEE TC, IEEE TPDS, IEEE CLOUD, and ICDCS. His recent research interests include cloud computing, information systems, and complex network. He serves on the Editorial Board of the AIMS Big Data and Information Analytics.

XIAOMIN ZHU (M'10) received the Ph.D. degree in computer science from Fudan University, Shanghai, China, in 2009. He is currently an Associate Professor with the College of Systems Engineering, National University of Defense Technology, Changsha, China. He has published over 90 research articles in refereed journals and conference proceedings such as IEEE TC, IEEE TPDS, IEEE TCC, JPDC, IEEE CLOUD, and ICDCS. His research interests include scheduling and resource management in distributed systems. He serves on the Editorial Boards of the Future Generation Computer Systems and the AIMS Big Data and Information Analytics.



XIAOSHENG FENG is currently an Associate Professor with the College of Systems Engineering, National University of Defense Technology, Changsha, China. His research interests include information resource management and big data analysis.

WEN ZHOU received the Ph.D. degree in management science and engineering from Harbin Engineering University in 2015. He is currently an Assistant Professor with the College of Systems Engineering, National University of Defense Technology, Changsha, China. His main research interests are in the field of cloud computing, information systems, and complex network.