

Received August 6, 2018, accepted September 19, 2018, date of publication September 24, 2018, date of current version October 17, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2871821

Energy-Efficient Fault-Tolerant Scheduling Algorithm for Real-Time Tasks in Cloud-Based 5G Networks

PENGZE GUO¹, MING LIU^{1,2}, JUN WU¹, (Member, IEEE), ZHI XUE¹,
AND XIANGJIAN HE², (Senior Member, IEEE)

¹Shanghai Key Laboratory of Integrated Administration Technologies for Information Security, Department of Electronic Engineering, Shanghai Jiao Tong University, Shanghai 200240, China

²School of Electrical and Data Engineering, University of Technology Sydney, Ultimo, NSW 2007, Australia

Corresponding author: Zhi Xue (zxue@sjtu.edu.cn)

This work was supported by the National Natural Science Foundation of China under Grant 61332010.

ABSTRACT Green computing has become a hot issue for both academia and industry. The fifth-generation (5G) mobile networks put forward a high request for energy efficiency and low latency. The cloud radio access network provides efficient resource use, high performance, and high availability for 5G systems. However, hardware and software faults of cloud systems may lead to failure in providing real-time services. Developing fault tolerance technique can efficiently enhance the reliability and availability of real-time cloud services. The core idea of fault-tolerant scheduling algorithm is introducing redundancy to ensure that the tasks can be finished in the case of permanent or transient system failure. Nevertheless, the redundancy incurs extra overhead for cloud systems, which results in considerable energy consumption. In this paper, we focus on the problem of how to reduce the energy consumption when providing fault tolerance. We first propose a novel primary-backup-based fault-tolerant scheduling architecture for real-time tasks in the cloud environment. Based on the architecture, we present an energy-efficient fault-tolerant scheduling algorithm for real-time tasks (EFTR). EFTR adopts a proactive strategy to increase the system processing capacity and employs a rearrangement mechanism to improve the resource utilization. Simulation experiments are conducted on the CloudSim platform to evaluate the feasibility and effectiveness of EFTR. Compared with the existing fault-tolerant scheduling algorithms, EFTR shows excellent performance in energy conservation and task schedulability.

INDEX TERMS Energy efficiency, fault tolerance, real-time, scheduling, cloud, 5G.

I. INTRODUCTION

The increasing requirements of communication quality have promoted the evolution of mobile communication technologies. 5G networks are expected to provide ubiquitous connectivity and real-time interaction. It is forecasted that there will be more than 50 billion connected devices in 2020 [1]. The mobile communication data in 2020 will be approximately 1000 times more than that in 2010 [2]. To achieve similar energy consumption, the energy efficiency (usually measured in bits/Joule) should be increased by 100x times [3]. According to the statistics, the power consumption of traditional base stations (BSs) account for 72% of the total power consumption of radio access networks (RAN), but the energy efficiency of BS is only about 50% [4]. BSs usually have excessive computing capacity to deal with the traffic

during peak hours. However, the resource utilization is low during off-peak hours. Besides, the ancillary equipments in distributed BSs consume large amount of energy. Spurred by both economic and environmental considerations, green computing has become a research priority in the design of information systems [5]–[7].

Cloud radio access network (C-RAN), which centralizes the baseband processing into cloud data centers, is a candidate solution for 5G [8]. In C-RAN, baseband unit (BBU) pool is responsible for the signal processing, and remote radio heads (RRHs) are distributed to receive and transmit the data from/to user equipments (UEs). BBU and RRHs are connected via high-speed optical fronthaul. Centralized signal processing considerably reduces the energy consumption of cooling devices and other supporting equipments

in BSs. Scalable computing can dynamically adjust the system resources according to the workload demands in the cloud environment [9]. Centralized deployment can improve the BS utilization by sharing resources under dynamic traffic load.

As a new paradigm of delivering computing services, cloud computing has the features of dynamic scalability, measured service and on-demand resource provisioning [10]–[12]. These features largely depend on virtualization. With virtualization, physical hosts can be divided into several virtual machines (VMs) [13]. As the mainstay of computing resources, cloud-based BBU pool takes charge of most task processing. It is of paramount importance to improve the energy efficiency of BBU servers. The elasticity of cloud computing raises challenges for C-RAN to increase the resource utilization. In addition, with the tremendous increase of network traffic, it is another tricky problem of how to meet the deadline constraints of real-time user requests.

Nowadays more and more real-time services are realized through wireless communication systems, e.g., Internet of Things (IoT), vehicular networks [14], and video streaming [15]. The timeliness of services should be guaranteed. In real-time systems, the computational results should be produced not only correctly but also timely [16]–[18]. The consequences of missing deadlines are different for different real-time systems [19]. For hard real-time systems, missing deadlines can result in catastrophe consequence. While for soft real-time systems, violation of time constraints usually results in service quality degradation, but the system can continue running [20].

In large-scale cloud data centers, node failures are common [21]. Therefore, fault tolerance is a mandatory mechanism of wireless networks. Since one computing instance's failure may cause some tasks to violate the deadline constraints, C-RAN should ensure the timeliness of real-time tasks even in case of failure. Fault-tolerant scheduling is an effective method to increase the system reliability. Primary-backup (PB) model is a popular fault-tolerant scheme. In the PB model, each task is duplicated and the two copies (i.e., primary copy and backup copy) are sent to different computing nodes for fault tolerance. Fundamentally, PB model utilizes the redundancy technology to improve the reliability of the system [22].

To the best of our knowledge, no previous work has been done on dynamic energy-efficient fault-tolerant scheduling for real-time tasks in cloud-based 5G networks. In this paper, the UEs' tasks that we concern are independent, aperiodic, and non-preemptive. Both energy conservation and fault tolerance is considered while meeting the real-time requirements. We first analyze the schedulability of real-time tasks and then try to reduce the energy consumption. In addition, we sufficiently consider the dynamics and elasticity of cloud computing, e.g., VM migration and VM creation. When the current system cannot guarantee the timing requirements, new VMs are added. Proactive strategy is adopted to select proper new VMs. Simulation results show that proactive

strategy and rearrangement mechanism bring substantial improvement in energy efficiency and tasks guarantee ratio.

The rest of this paper is organized as follows. Related work is described in Section II. Section III presents the system model, including architecture framework, power model, and VM migration. Scheduling criteria are also analyzed here. Section IV describes the EFTR algorithm in detail. Section V demonstrates the experiments to evaluate the performance of EFTR. Finally, we make conclusions in Section VI.

II. RELATED WORK

Energy-efficient techniques of wireless networks have been studied from the aspects of mobile devices, communication infrastructures, and cloud data centers. Many researches are focused on the former two cases [23]–[26]. This work focuses on energy saving in cloud data centers through task scheduling.

Since finding the optimal allocation of tasks in uniprocessor and multiprocessor systems is an NP-complete problem [27], many varieties of heuristics for scheduling tasks have been devised. For scheduling preemptive periodic tasks in uniprocessor systems, Liu and Layland [28] proposed the Rate-Monotonic (RM) algorithm, which prioritizes tasks in proportion to their frequency and is proved to be the optimal fixed-priority algorithm. To precisely judge the schedulability of tasks with priorities on uniprocessor, Joseph and Pandya [29] put forward the sufficient and necessary condition, called the Completion Time Test (CTT). Rate-Monotonic First-Fit (RMFF), which extends the RM algorithm from uniprocessor to multiprocessor with first-fit bin-packing heuristic, was designed by Dhall and Liu [30]. Some works have been done on task scheduling in C-RAN. Xia *et al.* [7] proposed an iterative coordinate descent algorithm to find the scheduling solution for minimizing the network power consumption of downlink C-RAN. Wang and Cen [31] proposed a real-time scheduling algorithm for periodic preemptive tasks in C-RAN. Zhang *et al.* [32] put forward the near-far C-RAN (NFC-RAN) architecture composed of near edge computing (NEC) and far edge computing (FEC). Task assignment between NEC and FEC is elaborated to increase the task completion rate. However, fault tolerance has not been studied in these algorithms.

Bertossi *et al.* [33] put forward a multiprocessor-based fault-tolerant algorithm FTRMFF using PB model. The FTRMFF algorithm considers both backup overbooking and deallocation [34] to reduce system overhead. Guo and Xue [35] proposed the QFTRMFF algorithm, which strives to optimize the QoS levels of tasks after the FTRMFF scheduling. Unfortunately, these works are designed for homogenous systems and not suitable for heterogeneous systems. The computing resources in C-RAN have various processing capabilities [36]. In addition, the tasks in above works are preemptive, i.e., a task can take the place of another executing task if their execution time overlaps. However, we consider non-preemptive tasks in this paper. Besides, some scheduling algorithms consider tasks with

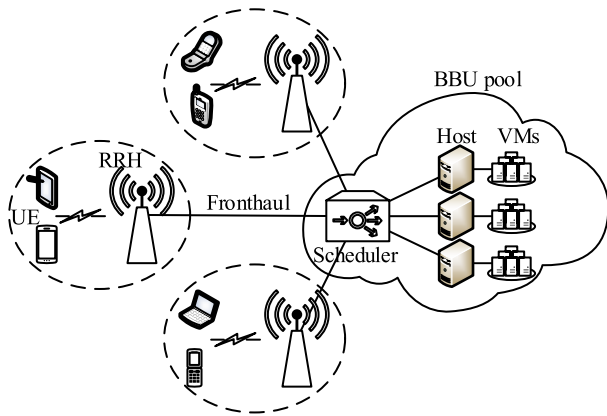


FIGURE 1. Structure of C-RAN.

inter-dependence [37]–[39], but we focus on independent tasks. Dependent tasks can be transformed to independent tasks by setting new start times and deadlines [40].

The aforementioned methods fall into the category of static scheduling. 5G systems allow users to transmit and receive data in a timely manner and require on-line scheduling [41]–[43]. Such dynamic processing cases raise higher demands on scheduling since tasks are independent and no priori knowledge about the upcoming tasks is given [44]. Luo et al. [45] proposed DYFARS, which leverages PB model to provide fault tolerance and enhances the reliability without additional costs. Zhu et al. [46] proposed a QoS-aware fault-tolerant scheduling algorithm (QAFT) to increase the QoS levels of real-time tasks. QAFT reduces system overhead by advancing primary copies and delaying backup copies. However, above fault-tolerant algorithms do not consider virtualization, which is the fundamental technique of cloud computing.

Recently, Wang et al. [47] put forward a fault-tolerant elastic scheduling algorithm for real-time tasks in clouds

called FESTAL. FESTAL takes virtualization into account, and uses backup overlapping to realize high system utilization. However, the FESTAL algorithm fails to take energy saving into account. Nonetheless, it provides a general method for task scheduling in the fault-tolerant context.

III. SCHEDULING MODEL

A. ARCHITECTURE FRAMEWORK

C-RAN is composed of BBU pool, RRHs, and fronthaul links, as shown in Fig. 1. After receiving requests from UEs, RRHs send pre-processed baseband signals to BBU pool for further processing. Each RRH is connected with a BBU pool via fronthaul link. BBU pool takes over most of the signal processing previously done in BSs. BBU pool consists of computing servers or physical hosts. Different from traditional distributed system, UEs’ tasks are executed by VMs rather than by physical hosts. Each host contains several VMs which are responsible for executing tasks.

Fig. 2 shows the architecture framework of fault-tolerant scheduling in the BBU pool. Multiple users submit their tasks to the system. When a new task arrives, firstly it is sent to Global Scheduler. After analyzing the information gathered from all computing nodes, Global Scheduler makes decisions according to the scheduling algorithm and sends the primary and backup copies of the task to different VMs. Then the primary copy is executed if the VM is idle, or waits in the local queue if the VM is busy. When the primary copy is finished successfully, the backup copy is deleted and the resource occupied by the backup copy is reclaimed. Local Scheduler is in charge of rearranging the order of the local queue if any backup copy is deallocated from the VM. Resource Manager decides how VMs should be added or migrated if the current processing capacity is unable to meet the timing requirements.

The power consumption of C-RAN is typically composed of three parts: RRH power consumption, fronthaul power

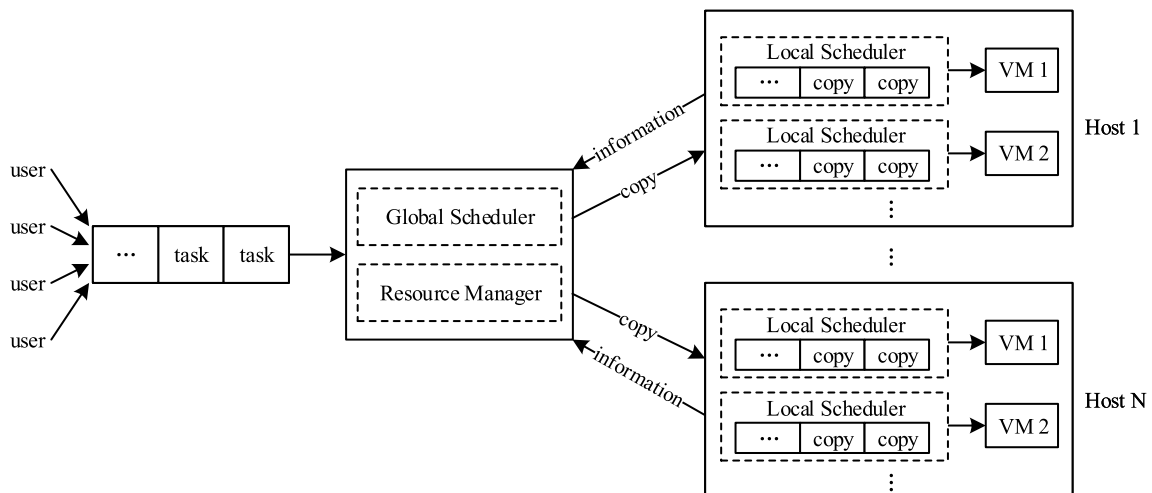


FIGURE 2. Fault-tolerant scheduling architecture of BBU pool.

consumption, and BBU pool power consumption [48]. This work focuses on energy saving in virtualized BBU pool through task scheduling.

B. SCHEDULING CRITERIA

In the BBU pool, each task from UE is sent to a physical host and executed by a VM on the host. The k -th VM on j -th host h_j is denoted by vm_{jk} . C_j and C_{jk} , which are measured by Million Instructions per Second (MIPS), denote the processing capacity of h_j and vm_{jk} respectively. In this paper, primary-backup model is employed to realize fault tolerance. In this model, each task t_i has two copies: primary copy t_i^P and backup copy t_i^B . The VMs that accommodate t_i^P and t_i^B are denoted by $vm(t_i^P)$ and $vm(t_i^B)$ respectively. t_i^P is executed before t_i^B . When t_i^P is finished, the task is executed successfully and t_i^B is removed from $vm(t_i^B)$. Task t_i arrives at a_i and must finish before its deadline d_i . l_i is the length of the task and is measured by Million Instructions (MI). The execution time of task t_i on vm_{jk} is denoted by $e_{jk}(t_i) = l_i/C_{jk}$. s_i^P and f_i^P denote the start time and finish time of t_i^P . s_i^B and f_i^B denote the start time and finish time of t_i^B . The start and finish times of copies are decided by the scheduling algorithm. The backup copy has two statuses: active and passive. A passive backup copy is executed only when the system encounters failure, whereas an active backup copy is always executed even without system faults. The status of t_i^B is decided by:

$$st(t_i^B) = \begin{cases} active, & \text{if } f_i^P > s_i^B; \\ passive, & \text{otherwise.} \end{cases} \quad (1)$$

Fig. 3 gives an example. The horizontal axis represents time. t_1^B and t_2^B adopt passive backup scheme while t_3^B adopts active backup scheme because $f_1^P < s_1^B$, $f_2^P < s_2^B$ and $f_3^P > s_3^B$. The active backup copy t_3^B is composed of redundant part t_3^{BR} (shaded area) and non-redundant part t_3^{BN} (unshaded area). The redundant part t_3^{BR} overlaps with the primary copy t_3^P in the time axis.

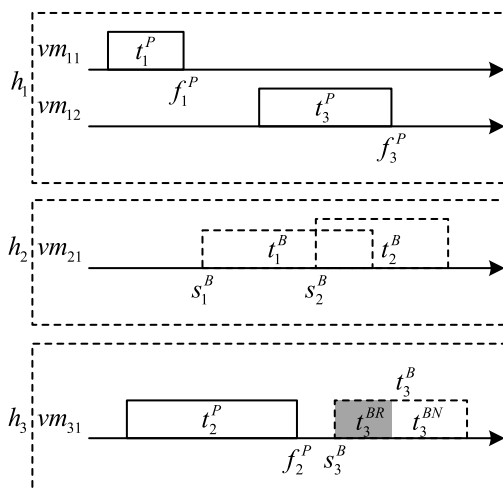


FIGURE 3. Illustration of primary and backup copies.

It is assumed that if the host $host(t_i^P)$ that accommodates t_i^P encounters failure, all VMs on the host, including $vm(t_i^P)$, break down. The fault-tolerant scheduling algorithm should guarantee that if a primary copy fails, its corresponding backup copy can still finish before deadlines. The proposed algorithm can tolerate at most one fault at any point of time. If tolerating multiple faults at one time instant is required, we can divide the hosts into several isolated groups, and apply the algorithm to each group [49].

PB approach is accomplished by introducing redundancy, i.e., the computing resource occupied by backup copies. Except for resource reclaiming, backup-backup (BB) overlapping [33] is adopted in this work to reduce system overhead. BB overlapping allows backup copies on the same VM to share the same time interval, thus reducing the VMs needed. Fig. 3 illustrates BB overlapping between t_1^B and t_2^B . t_1^B and t_2^B are both passive backup copies. If host h_1 fails, t_1^B is invoked. There is no conflict between t_1^B and t_2^B because when t_2^P finishes at f_2 , t_2^B is deallocated.

The overlapping criteria and scheduling principles are analyzed below. It is assumed that t_i is a newly arrived task. All other copies have been scheduled before its arrival.

1) PRIMARY COPY

The only criterion for scheduling primary copy is that no overlapping is allowed. Because if two primary copies overlap, there must be execution time conflict; if a primary copy overlaps with a backup copy t_j^B , they also have conflict when $host(t_j^P)$ fails and t_j^B gets invoked.

2) BACKUP COPY

Theorem 1: Backup copy cannot be scheduled on the host where the corresponding primary copy is located, i.e., $host(t_i^B) \neq host(t_i^P)$.

Proof: Prove by contradiction. Suppose that $host(t_i^B) = host(t_i^P)$. Regardless of whether $vm(t_i^B) = vm(t_i^P)$ or not, if $host(t_i^P)$ fails, both $vm(t_i^P)$ and $vm(t_i^B)$ break down. Thus, t_i^B cannot be invoked and fault tolerance is infeasible. \square

Theorem 2: Backup copy cannot overlap any primary copy.

Proof: Prove by contradiction. Suppose that t_i^B overlaps a primary copy t_j^P . When $host(t_i^P)$ fails, t_i^B must execute. No matter when the failure happens, t_i^B is bound to execute the entire task, thus incurring time conflict between t_i^B and t_j^P . Therefore, t_i^B cannot overlap t_j^P . \square

Theorem 3: Backup copy cannot overlap backup copy if their primary copies are on the same host, i.e., t_i^B cannot overlap t_j^B if $host(t_i^P) = host(t_j^P)$.

Proof: Prove by contradiction. Suppose that t_i^B overlaps t_j^B and $host(t_i^P) = host(t_j^P)$. When $host(t_i^P)$ fails, both t_i^P and t_j^P fail. t_i^B and t_j^B must execute, and execution time conflict between t_i^B and t_j^B is inevitable. Therefore, t_i^B cannot overlap t_j^B . \square

Theorem 4: Redundant part of active backup copy cannot overlap any backup copy, i.e., t_i^{BR} cannot overlap t_j^B .

Proof: Prove by contradiction. Suppose that t_i^{BR} overlaps t_j^B . When $host(t_j^P)$ fails, t_j^B is invoked. But t_i^{BR} always executes, thus resulting in time conflict between t_i^{BR} and t_j^B . Therefore, t_i^{BR} cannot overlap t_j^B . \square

Theorem 5: Active backup copies cannot overlap with each other.

Proof: Prove by contradiction. Suppose that t_i^B overlaps t_j^B , and both t_i^B and t_j^B are active. Then there must be overlapping between t_i^{BR} and t_j^B , or between t_i^B and t_j^{BR} , which violates Theorem 4. Therefore, t_i^{BR} cannot overlap t_j^B . \square

C. POWER MODEL

SPECpower benchmark [50] measures the power and performance characteristics of server-class computer equipment. For most servers, the data reflects linear relationship between power consumption and processor utilization. Therefore, the power model raised by Beloglazov *et al.* [51] is adopted in this paper. The power consumption is defined in the following equation:

$$P(t) = \alpha \cdot P_{max} + (1 - \alpha) \cdot P_{max} \cdot u(t), \quad (2)$$

where P_{max} is the power consumed when the server is 100% loaded; α is the fraction of power consumed when the server is idle; and $u(t)$ is the CPU utilization which varies with time due to different workloads.

Suppose there are N VMs on host h_j , the CPU utilization of h_j is:

$$u(t) = \sum_{k=1}^N \frac{C_{jk}(t)}{C_j}, \quad (3)$$

where $C_{jk}(t)$ equals C_{jk} if vm_{jk} is busy, or 0 if vm_{jk} is idle.

The idle power $\alpha \cdot P_{max}$ includes the power consumed by disk, memory, network interface, etc. For example, for PowerEdge R710 (Intel Xeon X5570, 16 cores \times 2.93 GHz, 8 GB), the idle power is 65 W and 100% active power is 220 W. The fraction of power consumed when the server is idle is 29.55%. We analyzed the data on SPECpower benchmark and calculated the ratios between idle power and 100% active power (i.e., the power consumption when the server is idle divided by the power consumption when the server is fully utilized). α is set to the average ratio 30% in this paper. Thus, the energy consumption for each host over a period of time is defined as:

$$E = \int_{t_1}^{t_2} P(t)dt = 0.3P_{max}(t_2 - t_1) + 0.7P_{max} \int_{t_1}^{t_2} u(t)dt. \quad (4)$$

The energy consumption of vm_{jk} to execute primary copy t_i^P is:

$$E_{jk}(t_i^P) = 0.7P_{max} \frac{C_{jk}}{C_j} \frac{l_i}{C_{jk}} = 0.7P_{max} \frac{l_i}{C_j}. \quad (5)$$

The energy consumption of vm_{jk} to execute backup copy t_i^B is:

$$E_{jk}(t_i^B) = \begin{cases} 0.7P_{max} \frac{C_{jk}}{C_j} (f_i^P - s_i^B), & \text{if } st(t_i^B) = active; \\ 0, & \text{otherwise.} \end{cases} \quad (6)$$

D. VIRTUAL MACHINE MIGRATION

Live migration of virtual machines refers to the technique of moving running VMs between physical hosts with negligible downtime [52]. An important motivation of VM migration is to consolidate the computing resource, thus increasing system utilization. The existing algorithms for dynamic VM consolidation which aim to reduce energy consumption rarely consider the cost of live migration.

It is assumed that the data of VMs is stored on network attached storage (NAS) and only memory migration is considered in this paper. Pre-copy [53] is a widely used approach for VM migration. In this approach, all memory is transmitted from the source host to the destination host at the first stage, and dirty pages of memory are iteratively transferred until the memory dirtying rate exceeds a threshold or the remaining dirty memory is small enough. The performance of VM live migration mainly depends on the memory size, memory dirtying rate, and network transmission rate. According to [54], the total network traffic (Megabyte) of migration is:

$$V_{mig} = \sum_{i=0}^n V_i = \sum_{i=0}^n V_{mem} \lambda^i = V_{mem} \frac{1 - \lambda^{n+1}}{1 - \lambda}, \quad (7)$$

where n is the total number of iterations, V_i is the data transferred at each round, V_{mem} is the size of VM memory, and λ is the ratio of memory dirtying rate to network transmission rate. In this paper, V_{mig} is set to the typical value $1.3V_{mem}$.

Some works have been done on the cost of VM migration [54]–[56]. The results show that the energy consumption indicates linear relationship with the data volume transferred. Liu *et al.* [54] found that the energy consumption of live migration is largely independent of the data transmission rate in a wired network. They concluded that the energy consumption increases linearly with the network traffic of VM migration and gave the energy cost (Joule) model:

$$E_{mig} = 0.512V_{mig} + 20.165. \quad (8)$$

In order to meet the requirement of fault tolerance, there are some constraints of VM migration [47]. We briefly list the constraints here:

- VM migration should not cause any task's primary and backup copies to be located on the same host;
- VM migration should not cause two primary copies to be located on the same host if their backup copies overlap.

Actually, these constraints are essentially identical to the first and third restricted conditions for backup copies.

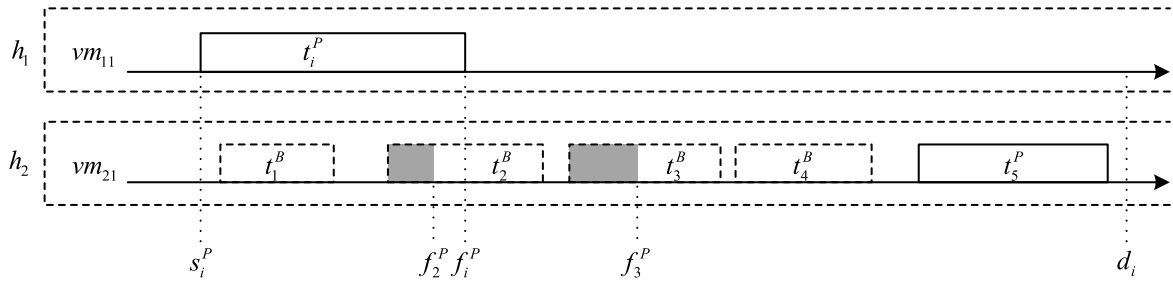


FIGURE 4. Illustration of backup schedulability test.

IV. ENERGY-EFFICIENT FAULT-TOLERANT SCHEDULING ALGORITHM

A. SCHEDULABILITY TEST

Suppose that t_i is a newly arrived task. Before scheduling, we need to perform the schedulability test to check whether t_i^P and t_i^B are schedulable on some VMs.

To check schedulability of t_i^P on vm_{jk} , we need to permute the primary and backup copies on vm_{jk} with the increasing of their start times. Suppose N primary and backup copies are assigned on vm_{jk} and the time slots occupied by them within interval $[a_i, d_i]$ are: $[s_1, f_1], [s_2, f_2], \dots, [s_N, f_N]$. s_n and f_n are the start and finish times of the n -th copy ($1 \leq n \leq N$), and $a_i \leq s_1 \leq s_2 \leq \dots \leq s_N < f_N \leq d_i$. Since primary copy should not overlap any copy, these time slots cannot be utilized by t_i^P . For the purpose of computational completeness, we extend the time slots to $[s_0, f_0], [s_1, f_1], [s_2, f_2], \dots, [s_N, f_N], [s_{N+1}, f_{N+1}]$ where $s_0 = -\infty, f_0 = a_i, s_{N+1} = d_i$ and $f_{N+1} = +\infty$. We need to scan these time slots from left to right to locate the minimum index n ($0 \leq n \leq N$) satisfying the following condition:

$$s_{n+1} - f_n \geq e_{jk}(t_i). \tag{9}$$

If such index n exists, then t_i^P is schedulable on vm_{jk} and the earliest finish time of t_i^P on vm_{jk} is $f_n + e_{jk}(t_i)$.

To check schedulability of t_i^B , BB overlapping should be considered. The unavailable time slots for t_i^B include all parts of primary copies, redundant parts of active backup copies, and backup parts located before f_i^P . Fig. 4 gives an example about schedulability test for backup copy t_i^B . The schedulability of t_i^B is checked on vm_{21} . Suppose that t_i^P is assigned to vm_{11} and four backup copies and one primary copy are located on vm_{21} . t_1^B and t_4^B are passive and t_2^B and t_3^B are active. The shaded areas are not available for t_i^B because redundant parts of active backup copies cannot overlap any copy. The backup parts located before f_i^P (i.e., $[s_1^B, f_1^B]$ and $[s_2^B, f_2^B]$) are not available for t_i^B either. Because if t_i^B occupies these time slots, its status must be active, incurring t_i^{BR} overlaps t_1^B and t_2^B , which violates the scheduling principle. Besides, the time slot occupied by t_4^B is available because it is passive. The area occupied by t_5^P is unavailable because no overlapping is allowed for primary copy. So the unavailable time slots for t_i^B on vm_{21} are $[s_1^B, f_1^B], [s_2^B, f_2^B], [s_3^B, f_3^B]$ and $[s_5^P, f_5^P]$.

Without loss of generality, suppose the unavailable time slots for t_i^B on vm_{jk} within interval $[s_i^P, d_i]$ are $[s_1, f_1], [s_2, f_2], \dots, [s_N, f_N]$ and $s_i^P \leq s_1 \leq s_2 \leq \dots \leq s_N < f_N \leq d_i$. For the purpose of computational completeness, we extend the time slots to $[s_0, f_0], [s_1, f_1], [s_2, f_2], \dots, [s_N, f_N], [s_{N+1}, f_{N+1}]$ where $s_0 = -\infty, f_0 = s_i^P, s_{N+1} = d_i$ and $f_{N+1} = +\infty$. To find the latest start time of t_i^B , these time slots should be scanned from right to left to seek the maximum index n ($0 \leq n \leq N$) satisfying condition (9). If such index n exists, then t_i^B is schedulable on vm_{jk} and the latest start time of t_i^B on vm_{jk} is $s_{n+1} - e_{jk}(t_i)$.

We summarize the above process into function $Schedulability(t_i^*, vm_{jk})$. In short, function $Schedulability(t_i^P, vm_{jk})$ returns the earliest finish time of t_i^P if t_i^P is schedulable on vm_{jk} , and returns $+\infty$ otherwise. Function $Schedulability(t_i^B, vm_{jk})$ returns the latest start time of t_i^B if t_i^B is schedulable on vm_{jk} , and returns $-\infty$ otherwise.

B. REARRANGEMENT MECHANISM

In this work, we take the idea in [57] to adjust the execution sequence of copies. In [57], the backup copy is deallocated after its corresponding primary copy finishes and the idle time slot left by the backup copy can be utilized by the primary copies located on the same VM. This rearrangement mechanism helps advance the start time of primary copies and reduce the redundant parts of active backup copies. In order to further reduce the system overhead introduced by backup copies, we make improvements on the mechanism by rearranging the backup copies. The pseudocode of rearrangement is shown in Algorithm 1. When a primary copy finishes and the corresponding backup copy is deleted, the rearrangement process gets invoked on the VM which deallocates the backup copy. Firstly, all primary copies waiting on the VM are checked if they can move forward (see lines 1-13). Then all backup copies are checked if they can move backward (see lines 14-26). It should be noted that such change of execution order does not violate the scheduling constraints listed in Section III.

An example of rearrangement is shown in Fig. 5. When t_1^P finishes at f_1^P , t_1^B is deallocated from vm_{21} . The idle time interval left by t_1^B is utilized by t_3^P . After checking schedulability, t_3^P moves forward. Besides, the status of t_3^B becomes passive

Algorithm 1 Pseudocode of Rearrangement

```

1 foreach  $t_i^P$  on  $vm_{jk}$  do
2    $startTimeOrigin \leftarrow s_i^P$ ;
3    $finishTimeOrigin \leftarrow f_i^P$ ;
4   Deallocate  $t_i^P$  from  $vm_{jk}$ ;
5    $finishTimeNew \leftarrow Schedulability(t_i^P, vm_{jk})$ ;
6   Allocate  $t_i^P$  on  $vm_{jk}$ ;
7   if  $finishTimeNew < finishTimeOrigin$  then
8      $s_i^P \leftarrow finishTimeNew - e_{jk}(t_i)$ ;
9      $f_i^P \leftarrow finishTimeNew$ ;
10    Update the status of  $t_i^B$  according to (1);
11  else
12     $s_i^P \leftarrow startTimeOrigin$ ;
13     $f_i^P \leftarrow finishTimeOrigin$ ;
14 foreach  $t_i^B$  on  $vm_{jk}$  do
15    $startTimeOrigin \leftarrow s_i^B$ ;
16    $finishTimeOrigin \leftarrow f_i^B$ ;
17   Deallocate  $t_i^B$  from  $vm_{jk}$ ;
18    $startTimeNew \leftarrow Schedulability(t_i^B, vm_{jk})$ ;
19   Allocate  $t_i^B$  on  $vm_{jk}$ ;
20   if  $startTimeNew > startTimeOrigin$  then
21      $s_i^P \leftarrow startTimeNew$ ;
22      $f_i^P \leftarrow startTimeNew + e_{jk}(t_i)$ ;
23     Update the status of  $t_i^B$  according to (1);
24  else
25     $s_i^P \leftarrow startTimeOrigin$ ;
26     $f_i^P \leftarrow finishTimeOrigin$ ;

```

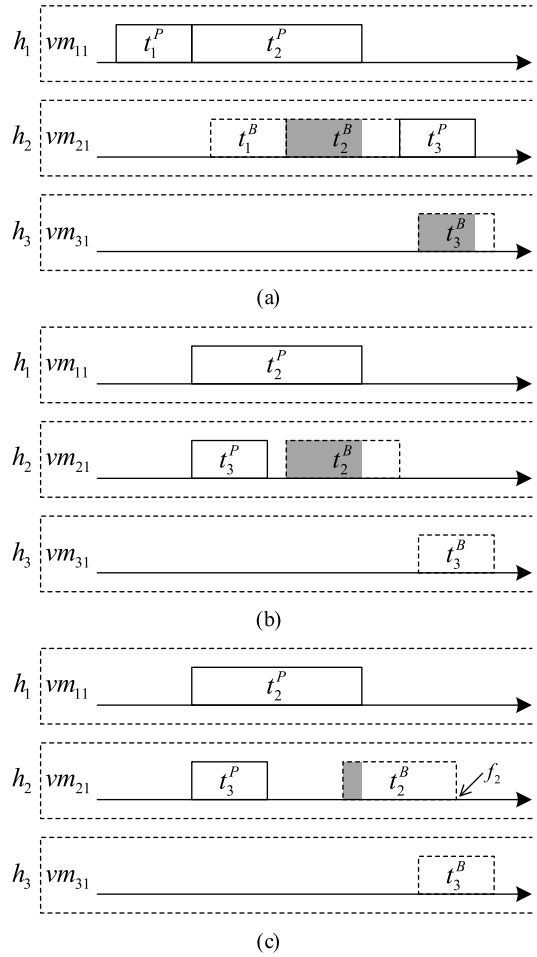


FIGURE 5. Illustration of rearrangement. (a) Before t_1^P finishes. (b) After t_1^P finishes, t_3^P moves forward. (c) After t_3^P moves forward, t_2^B moves backward.

due to the earlier finish time of t_3^P . Then t_2^B moves backward utilizing the time interval left by t_3^P . This mechanism brings two benefits. Firstly, primary copies get the chance to finish earlier. Without the arrangement mechanism, t_3^P cannot start execute before f_2^B . Secondly, the energy consumption of redundant parts is reduced. t_3^B does not consume energy after its status changes from active to passive, and t_2^B consumes less energy.

C. PRIMARY AND BACKUP SCHEDULING

For primary copies, they should be executed as early as possible. While for backup copies, they should be scheduled as late as possible. More precisely, primary copies should be assigned to VMs with earliest finish time (EFT) and backup copies should be assigned to VMs with latest start time (LST). Besides, both primary and backup copies must comply with the policy of minimum energy cost (MEC). In this work, MEC has higher priority than EFT and LST.

The pseudocode of primary and backup scheduling algorithm is presented in Algorithm 2. Suppose t_i is a newly arrived task. The system firstly checks the schedulability of the primary copy t_i^P on all VMs (see lines 1-5). If t_i^P passes the schedulability test, the energy consumption of t_i^P on each VM is calculated (see lines 6-7). If assigning t_i^P to a VM can

decrease the energy consumption or advance the finish time on the premise of equal energy consumption, t_i^P is assigned to the VM (see lines 8-11). If it fails to find a VM to accommodate t_i^P , then function $ScaleUp(t_i^P)$ (described below) is called to scale up the system by creating new VMs. If the system expansion remains unable to satisfy the scheduling requirements, t_i will be given up (see lines 12-17). If t_i^P is successfully allocated, then the algorithm manages to schedule the backup copy t_i^B . All VMs except for those on $h(t_i^P)$ are checked if t_i^B can be executed on them while meeting the real-time requirement. The VM where t_i^B gets the latest start time is selected (see lines 18-23). If assigning t_i^B to existing VMs fails, $ScaleUp(t_i^B)$ is invoked. If the system can accommodate t_i^B after calling $ScaleUp(t_i^B)$, then t_i^B is allocated to the new VM; else t_i is rejected (see lines 24-29).

The pseudocode of function $ScaleUp(t_i^*)$ is shown in Algorithm 3. $ScaleUp(t_i^*)$ is triggered by existing system's failing to allocate primary or backup copy t_i^* . Suppose there are N types of VM templates that can be deployed on the hosts. Their processing capacities (in MIPS) are c_1, c_2, \dots, c_N with $c_1 < c_2 < \dots < c_N$, and memories

Algorithm 2 Pseudocode of Primary and Backup Scheduling

```

1 foreach new task  $t_i$  do
2    $f_i^P \leftarrow +\infty$ ;  $E(t_i^P) \leftarrow +\infty$ ;
3   foreach host  $h_j$  do
4     foreach VM  $vm_{jk}$  on  $h_j$  do
5        $EFT \leftarrow \text{Schedulability}(t_i^P, vm_{jk})$ ;
6       if  $EFT \neq +\infty$  then
7         Calculate  $E_{jk}(t_i^P)$  according to (5);
8         if  $E(t_i^P) > E_{jk}(t_i^P) \parallel (E(t_i^P) == E_{jk}(t_i^P)$ 
          &&  $f_i^P > EFT)$  then
9            $vm(t_i^P) \leftarrow vm_{jk}$ ;
10           $f_i^P \leftarrow EFT$ ;
11           $E(t_i^P) \leftarrow E_{jk}(t_i^P)$ ;
12   if  $vm(t_i^P) == \text{Null}$  then
13      $vm_{new} \leftarrow \text{ScaleUp}(t_i^P)$ ;
14     if  $vm_{new} \neq \text{Null}$  then
15        $vm(t_i^P) \leftarrow vm_{new}$ ;
16     else
17       Reject  $t_i$ ;
18   foreach host  $h_j \neq h(t_i^P)$  do
19     foreach VM  $vm_{jk}$  on  $h_j$  do
20        $LST \leftarrow \text{Schedulability}(t_i^B, vm_{jk})$ ;
21       if  $LST \neq -\infty$  &&  $s_i^B < LST$  then
22          $vm(t_i^B) \leftarrow vm_{jk}$ ;
23          $s_i^B \leftarrow LST$ ;
24   if  $vm(t_i^B) == \text{Null}$  then
25      $vm_{new} \leftarrow \text{ScaleUp}(t_i^B)$ ;
26     if  $vm_{new} \neq \text{Null}$  then
27        $vm(t_i^B) \leftarrow vm_{new}$ ;
28     else
29       Reject  $t_i$ ;

```

(in MB) are m_1, m_2, \dots, m_N with $m_1 < m_2 < \dots < m_N$. r_j is the remaining processing capacity of host h_j . t_{VM} denotes the time for creating a VM, and t_{host} denotes the boot time of a physical host. po_j is the time when host h_j gets powered on. BW is the bandwidth of the network between different hosts. $\text{ScaleUp}(t_i^*)$ strives to create new VMs to raise the system's processing capability. Running hosts are checked first. The VM template with minimum processing capacity that satisfies both timing requirement and processing capacity constraint is selected as a candidate. If t_i^* is primary copy and the expected finish time exceeds the average of a_i and d_i , the new VM's processing capacity is increased by one level (see lines 1-7). Because if $f_i^P > (a_i + d_i)/2$, t_i^B is very likely to be active. Besides, creating VMs with excessively low processing capacities makes little sense for subsequent tasks. This proactive strategy is a trade-off between decreasing energy

Algorithm 3 Pseudocode of Function $\text{ScaleUp}(t_i^*)$

```

1 foreach running host  $h_j$  do
2   Find the minimum processing capacity  $c_k$  that
   satisfies  $a_i + t_{VM} + l_i/c_k \leq d_i$  &&  $r_j \geq c_k$ ;
3   if such  $c_k$  exists then
4     if  $t_i^*$  is primary copy &&
        $a_i + t_{VM} + l_i/c_k > (a_i + d_i)/2$  then
5        $k \leftarrow k \neq N?(k+1) : N$ ;
6       Create  $vm_{new}$  with processing capacity  $c_k$  on  $h_j$ ;
7       Return  $vm_{new}$ ;
8   foreach turning on host  $h_j$  do
9     Find the minimum processing capacity  $c_k$  that
     satisfies  $po_j + t_{host} + t_{VM} + l_i/c_k \leq d_i$  &&  $r_j \geq c_k$ ;
10    if such  $c_k$  exists then
11      if  $t_i^*$  is primary copy &&
         $po_j + t_{host} + t_{VM} + l_i/c_k > (a_i + d_i)/2$  then
12         $k \leftarrow k \neq N?(k+1) : N$ ;
13        Create  $vm_{new}$  with processing capacity  $c_k$  on  $h_j$ ;
14        Return  $vm_{new}$ ;
15  foreach host  $h_j$  do
16     $c_{min}^j \leftarrow$  minimum VM processing capacity on  $h_j$ ;
17     $m_{min}^j \leftarrow$  minimum VM memory on  $h_j$ ;
18    Find the minimum processing capacity  $c_k$  that
    satisfies  $a_i + 1.3m_{min}^j/BW + t_{VM} + l_i/c_k \leq d_i$  &&
     $r_j + c_{min}^j \geq c_k$ ;
19    if such  $c_k$  exists then
20      foreach VM  $vm_{min}^j$  with processing capacity
         $c_{min}^j$  on  $h_j$  do
21        foreach host  $h_k \neq h_j$  do
22          if  $r_k \geq c_{min}^j$  then
23            Migrate  $vm_{min}^j$  from  $h_j$  to  $h_k$  and
            create  $vm_{new}$  with processing
            capacity  $c_k$  on  $h_j$ ;
24            Return  $vm_{new}$ ;
25  Find the minimum processing capacity  $c_k$  that satisfies
     $a_i + t_{host} + t_{VM} + l_i/c_k \leq d_i$ ;
26  if such  $c_k$  exists then
27    if  $t_i^*$  is primary copy &&
       $a_i + t_{host} + t_{VM} + l_i/c_k > (a_i + d_i)/2$  then
28       $k \leftarrow k \neq N?(k+1) : N$ ;
29    Turn on a new host and create  $vm_{new}$  with
    processing capacity  $c_k$  on the new host;
30    Return  $vm_{new}$ ;
31 Return  $\text{Null}$ ;

```

consumption and improving system performance. If no suitable VM can be allocated to running hosts, then hosts which are starting up are considered. VM templates are checked if

they can be pre-allocated to hosts (see lines 8-14). If above strategies do not work, then the algorithm checks if migrating a VM from some host and consolidating the spare processing capability can meet the time and resource requirements (see lines 15-24). If creating a new VM on existing hosts is still not feasible, then a new host is turned on and a new VM with suitable processing capacity is created on it (see lines 25-30). If all attempts to allocate t_i^* fail, *Null* is returned (see line 31).

V. PERFORMANCE EVALUATION

A. PERFORMANCE METRICS

In this section, we evaluate the overall performance of EFTR. We compare it with FESTAL, which is an algorithm recently proposed by Wang et al. [47], baseline algorithm NPEFTR (non-proactive EFTR), algorithm NPEFTR (non-proactive EFTR), and NMEFTR (non-migration EFTR). The algorithms for comparison are concisely explained as follows:

- *FESTAL*. It provides a general framework for task scheduling in the cloud environment and considers both virtualization and backup overlapping. Different from EFTR, FESTAL adopts conservative policy instead of proactive policy, which means FESTAL creates new VMs with minimum processing capacities satisfying the energy and resource constraints. FESTAL does not adopt the rearrangement mechanism. Besides, it does not consider the energy problem. For ease of comparison, FESTAL is modified in such a way that it uses the same scheduling strategy in Algorithm 2.
- *NPEFTR*. Different from EFTR, NPEFTR does not adopt the proactive strategy.
- *NREFTR*. Different from EFTR, NREFTR does not employ the rearrangement mechanism.
- *NMEFTR*. Different from EFTR, NMEFTR does not employ the VM migration technique.

We compare the algorithms based on the following three metrics:

- Guarantee Ratio is defined to be the ratio of the number of successfully executed tasks over the total number of tasks.
- Energy Consumption is defined as the total power consumption.
- VM Count denotes the total number of VMs needed during the scheduling.

B. EXPERIMENT SETUP

Simulation has the advantage of providing repeatable and controllable environment. CloudSim [58] is selected as our simulation platform. CloudSim is an event driven framework for modeling cloud infrastructures and services. User defined policies and strategies for managing tasks and resources can be deployed on the platform. In this paper, three types of hosts and VMs are available in the cloud data centers. The maximum power of each host is 200, 250 or 400 W, and their corresponding processing capacities are 1000, 1500 and 2000 MIPS, respectively. The processing capacities of three types of VM templates are 200, 300 and 400 MIPS. The time

needed for creating a VM and turning on a host is 15 and 90 seconds respectively.

The characteristics of tasks, including task size, task count, interval time and baseDeadline, are shown in Table 1. The task size (MI) is uniformly distributed between 10^5 and 2×10^5 . The task arrival rate is in compliance with Poisson distribution. $1/\lambda$ (s) denotes the mean time between task arrival. The deadline of each task t_i is $d_i = a_i + U$ (*baseDeadline*, *4baseDeadline*).

TABLE 1. Parameters of tasks.

Parameter	Value (Fixed)-(Min, Max, Step)
task size ($\times 10^5$ MI)	([1, 2])
task count ($\times 10^4$)	(1)-(0.5, 3, 0.5)
interval time $1/\lambda$ (s)	(2)-(1, 7, 1)
baseDeadline (s)	(300)-(170, 350, 30)

C. PERFORMANCE IMPACT OF TASK COUNT

In this section, we conduct experiments to evaluate the performance impact of task count. Task count increases from 5000 to 30000 with step 5000, and other variables are constant.

Fig. 6(a) shows the energy consumption impact of task count. With the increase of task count, more energy is consumed, because longer execution time is needed and more VMs are created. EFTR consumes least energy while FESTAL consumes most energy. This result indicates that proactive and rearrangement policies play important roles in saving energy. The performance difference between EFTR and NPEFTR shows that although the proactive strategy increases the processing capacities of some new VMs, it does increase the energy consumption because less VMs are created (see Fig. 6(c)). The comparison between EFTR and NREFTR indicates that the rearrangement mechanism helps efficiently reduce idle power consumption by utilizing the idle time slots left by deleted backup copies. Besides, the energy consumption of NMEFTR indicates that VM migration can effectively increase the resource utilization. EFTR outperforms FESTAL by 9.02% on average in energy conservation.

Fig. 6(b) shows that the guarantee ratio impact of task count is relatively stable and the fluctuation is less than 1%. This can be ascribed to the elasticity of cloud system. When more tasks arrive, computing resources can be added from the infinite resource pool to guarantee that tasks can meet their deadlines. When the number of tasks are relatively small, the time delays caused by turning on hosts and creating VMs have a negative impact on the guarantee ratio. When the number of tasks are large enough, the system reaches a balanced state and less new hosts and VMs are needed. So the guarantee ratio gradually increases to a stable value. Besides, EFTR and NREFTR have higher guarantee ratios than other three algorithms. This can be attributed to the proactive strategy adopted by EFTR and NREFTR, which increases the system processing capacity to accommodate

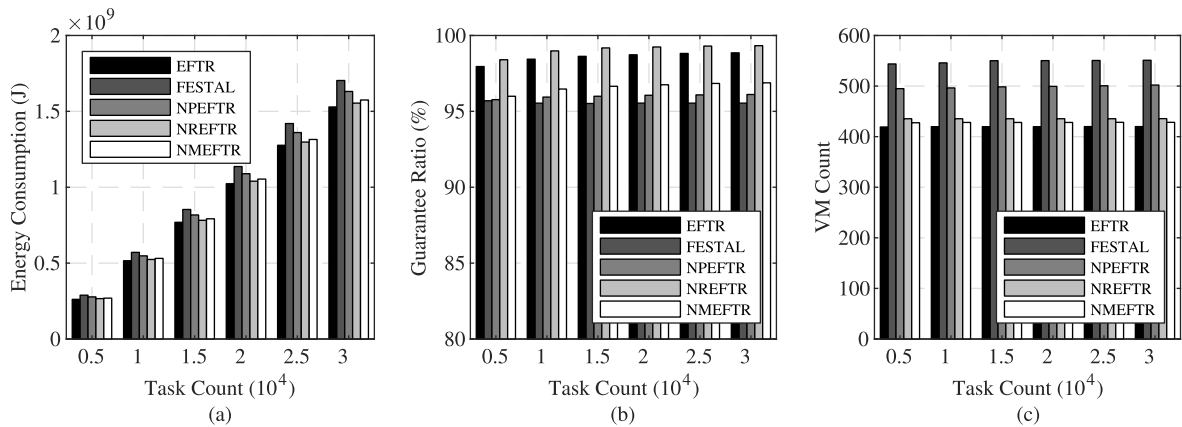


FIGURE 6. Performance impact of task count. (a) Energy consumption impact of task count. (b) Guarantee ratio impact of task count. (c) VM count impact of task count.

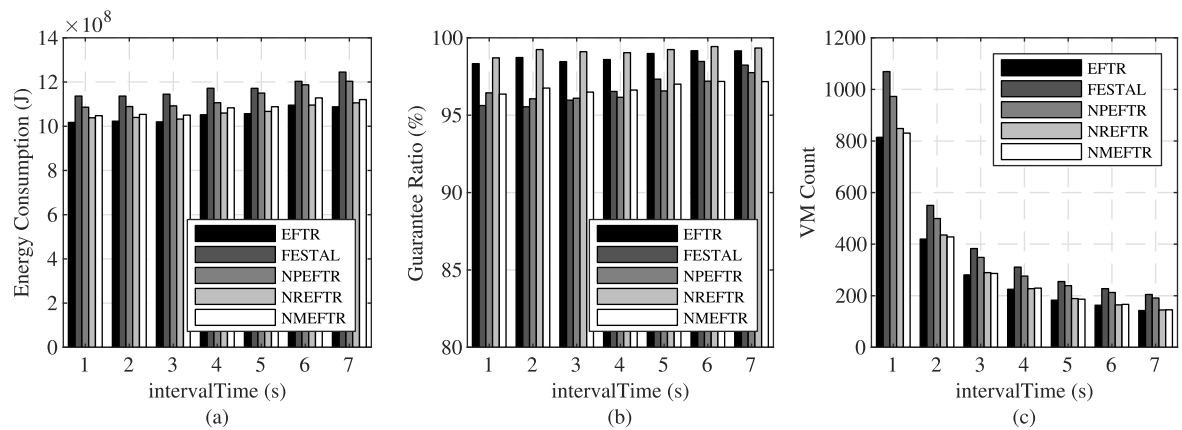


FIGURE 7. Performance impact of task arrival rate. (a) Energy consumption impact of arrival rate. (b) Guarantee ratio impact of arrival rate. (c) VM count impact of arrival rate.

more tasks. EFTR achieves 3% higher guarantee ratio than FESTAL.

Fig. 6(c) demonstrates that EFTR needs least VMs among the five algorithms. Compared with NREFTR, EFTR shows that the rearrangement mechanism plays an important role in reducing the system overhead by allowing waiting primary copies to move forward and waiting backup copies to move backwards. Compared with NPEFTR, EFTR shows that the proactive strategy makes good trade-off between increasing system processing capacity and reducing VM count. The performance of NMEFTR shows that VM migration can save about 2% VMs for EFTR. EFTR needs 23.5% less VMs than FESTAL.

D. PERFORMANCE IMPACT OF TASK ARRIVAL RATE

Parameter *intervalTime* reflects the task arrival rate. So the smaller *intervalTime* is, the more frequently tasks arrive. We vary *intervalTime* while keeping other parameters unchanged to test the influence of task arrival rate.

From Fig. 7(a), we can observe that the energy consumption increases gradually with the decrease of task arrival rate. This is because larger time intervals between two tasks leads

to longer finish times of all tasks, thus increasing the overall energy consumption. Compared with tasks’ execution times, the change of time interval is relatively small, so the rise in energy is not obvious. Basically, EFTR needs least energy. The explanation is the same as that in Fig. 6(a). On average, EFTR outperforms FESTAL by 10.43% in terms of energy conservation.

Fig. 7(b) shows that the guarantee ratios of five algorithms keep slow increasing trend with the decrease of task arrival rate. The reason is that lower arrival rate means less resource competition and less system load. Furthermore, the cloud system has enough time to expand the computing capacities by adding new hosts or VMs. EFTR and NREFTR with proactive strategy have higher guarantee ratios than FESTAL, NPEFTR and NMEFTR.

Fig. 7(c) illustrates that the VM count decreases sharply when tasks arrive more slowly. When the task load is heavy, the system has to add more VMs to accommodate the tasks and to meet the deadline requirements. When the system load becomes light, the system is capable to execute the tasks and less new resources are needed. Owing to the proactive and rearrangement policies, EFTR requires least VMs.

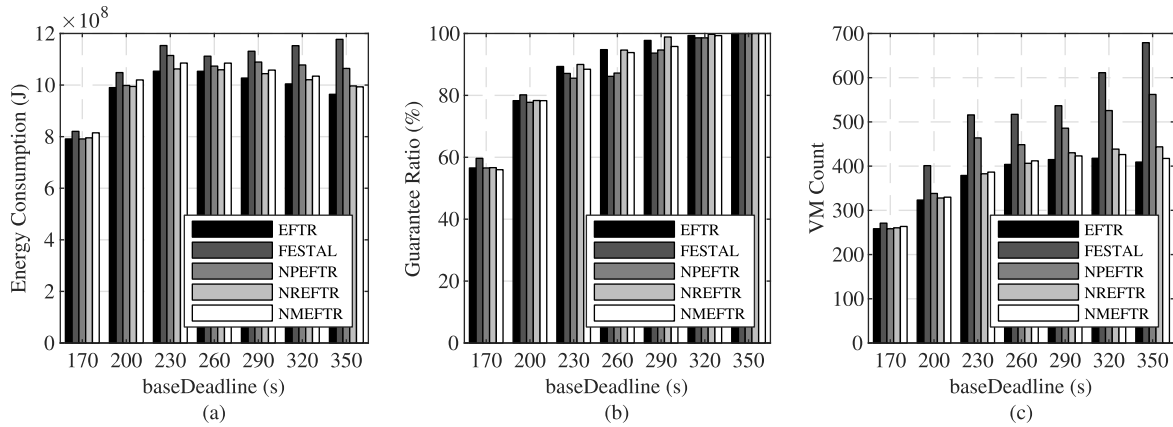


FIGURE 8. Performance impact of task deadline. (a) Energy consumption impact of deadline. (b) Guarantee ratio impact of deadline. (c) VM count impact of deadline.

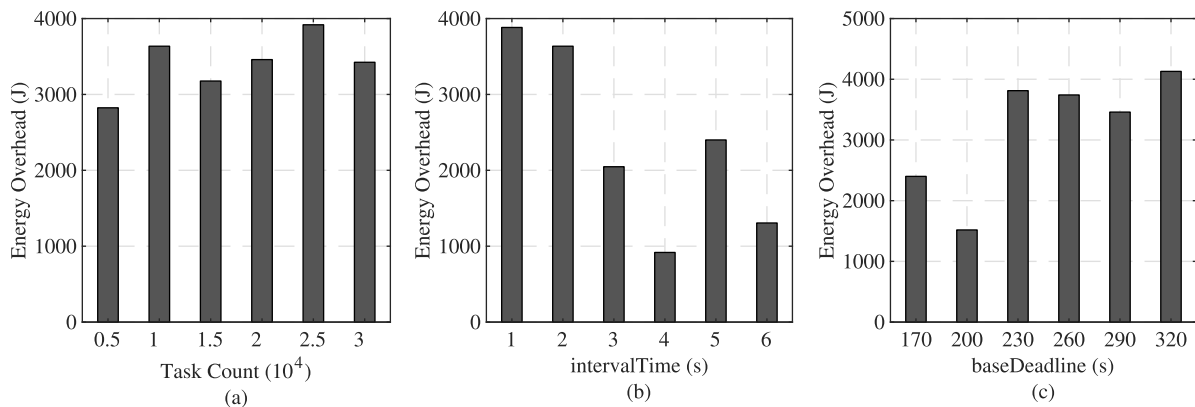


FIGURE 9. Energy overhead of VM migration. (a) Overhead impact of task count. (b) Overhead impact of arrival rate. (c) Overhead impact of deadline.

E. PERFORMANCE IMPACT OF TASK DEADLINE

Task deadline is also a significant factor that affects the algorithm performance. In this section, we compare the five algorithms in terms task deadline. With the increase of parameter *baseDeadline*, the tasks have looser deadlines. Parameter *baseDeadline* varies from 170 to 350 with step size 30.

As we can see from Fig. 8(a), the energy consumption becomes larger when *baseDeadline* increases. Because more tasks are accepted when deadlines become looser, thus more VMs are created. When *baseDeadline* is larger than 230, the energy consumption of all algorithms decreases. It can be explained that when more tasks can be finished on existing VMs, the finish times of tasks become earlier and the growth of VM count slows down. Moreover, the downward trend of EFTR is more obvious because it employs both proactive and rearrangement policies to increase the system efficiency. Compared with FESTAL, EFTR conserves energy by 9.89% on average.

Fig. 8(b) shows that the guarantee ratio gets higher with the increase of deadline. The reason here is obvious – looser deadlines allow more tasks to finish before their deadlines with the same system processing capacity. When *baseDeadline* is big enough, the guarantee ratio gets close to 100%.

Besides, EFTR and NREFTR have higher guarantee ratios than FESTAL, NPEFTR and NMEFTR. This is because adopting proactive strategy can shorten the execution times of tasks, thus raising the guarantee ratio.

The slowdown trend of VM count growth is obvious in Fig. 8(c). The VM counts of EFTR, NREFTR and NMEFTR increase slowly and even decrease when *baseDeadline* is large enough. However, the VM counts of FESTAL and NPEFTR keep increasing. The effect of proactive strategy is evident here. Besides, EFTR needs the least number of VMs, which verifies the effectiveness of the rearrangement mechanism.

F. OVERHEAD OF VM MIGRATION

In this section, we evaluate the energy overhead of VM migration in EFTR. As shown in Fig. 9, the overhead caused by VM migration is generally proportional to the number of VMs. The energy overhead is negligible compared with the total energy consumption. However, the positive effect of VM migration is obvious. From Fig. 6-8, we can see that with VM migration, the algorithm needs about 2% less VMs, consumes 3% less energy, and accepts 2% more tasks. The data indicates that employing the VM migration technique in

cloud BBU pool is efficient in energy conservation and task processing.

VI. CONCLUSION

In this paper, we propose an energy efficient fault-tolerant scheduling algorithm, called EFTR, for real-time tasks in C-RAN. Fault tolerance is realized based on the primary-backup model. EFTR algorithm dynamically schedules primary and backup copies of tasks with timing requirements to different virtual machines. The scheduling criteria and backup overlapping constraints are discussed in detail. Schedulability test is designed to check whether the primary and backup copies are schedulable on some VMs. In order to increase resource utilization, we employ the rearrangement mechanism to fully utilize the idle time slots. In addition, EFTR inherits the elasticity of cloud computing and adopts proactive strategy to increase the system processing capacity. These policies significantly improve the system schedulability and reduce the energy consumption. Through theoretical analysis and simulation studies, we show that EFTR outperforms FESTAL in terms of energy conservation, guarantee ratio and VM count under different workloads. Meanwhile, we notice that our algorithm is not suitable for dependent tasks, which are common in realistic environment. This is our research focus in further studies.

REFERENCES

- [1] "More than 50 billion connected devices," Ericsson, Stockholm, Sweden, White Paper, Feb. 2011.
- [2] Q. Wang, D. Chen, N. Zhang, Z. Qin, and Z. Qin, "LACS: A lightweight label-based access control scheme in IoT-based 5G caching context," *IEEE Access*, vol. 5, pp. 4018–4027, 2017.
- [3] A. Abrol and R. K. Jha, "Power optimization in 5G networks: A step towards GrEEen communication," *IEEE Access*, vol. 4, pp. 1355–1374, 2016.
- [4] "C-RAN: The road towards green RAN, version 2.5," China Mobile Res. Inst., Beijing, China, White Paper, Oct. 2011.
- [5] Q. Xu, Z. Su, Q. Zheng, M. Luo, and B. Dong, "Secure content delivery with edge nodes to save caching resources for mobile users in green cities," *IEEE Trans. Ind. Informat.*, vol. 14, no. 6, pp. 2550–2559, Jun. 2018.
- [6] J. Wu, M. Dong, K. Ota, J. Li, and Z. Guan, "Big data analysis-based secure cluster management for optimized control plane in software-defined networks," *IEEE Trans. Netw. Service Manag.*, vol. 15, no. 1, pp. 27–38, Mar. 2018.
- [7] W. Xia, J. Zhang, T. Q. S. Quek, S. Jin, and H. Zhu, "Energy-efficient task scheduling and resource allocation in downlink C-RAN," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Apr. 2018, pp. 1–6.
- [8] J. Luo, Q. Chen, and L. Tang, "Reducing power consumption by joint sleeping strategy and power control in delay-aware C-RAN," *IEEE Access*, vol. 6, pp. 14655–14667, 2018.
- [9] Q. Liu, T. Han, N. Ansari, and G. Wu, "On designing energy-efficient heterogeneous cloud radio access networks," *IEEE Trans. Green Commun. Netw.*, vol. 2, no. 3, pp. 721–734, Sep. 2018.
- [10] A. Botta, W. de Donato, V. Persico, and A. Pescapé, "Integration of Cloud computing and Internet of Things: A survey," *Future Gener. Comput. Syst.*, vol. 56, pp. 684–700, Mar. 2016.
- [11] Y. Al-Dhuraiibi, F. Paraiso, N. Djarallah, and P. Merle, "Elasticity in cloud computing: State of the art and research challenges," *IEEE Trans. Serv. Comput.*, vol. 11, no. 2, pp. 430–447, Mar./Apr. 2018.
- [12] P. M. Mell and T. Grance, "The NIST definition of cloud computing," Nat. Inst. Standards Technol., Gaithersburg, MD, USA, Tech. Rep., 2011.
- [13] F. Zhang, G. Liu, X. Fu, and R. Yahyapour, "A survey on virtual machine migration: Challenges, techniques, and open issues," *IEEE Commun. Surv. Tuts.*, vol. 20, no. 2, pp. 1206–1243, 2nd Quart., 2018.
- [14] Z. Su, Y. Hui, Q. Xu, T. Yang, J. Liu, and Y. Jia, "An edge caching scheme to distribute content in vehicular networks," *IEEE Trans. Veh. Technol.*, vol. 67, no. 6, pp. 5346–5356, Jun. 2018.
- [15] N. Abbas, H. Hajj, Z. Abbas, K. Jahed, and S. Sharafeddine, "An optimized approach to video traffic splitting in heterogeneous wireless networks with energy and QoE considerations," *J. Netw. Comput. Appl.*, vol. 83, pp. 72–88, Apr. 2017.
- [16] S. Baruah, M. Bertogna, and G. Buttazzo, *Multiprocessor Scheduling for Real-Time Systems* (Embedded Systems). Springer, 2015.
- [17] A. A. Safaei, "Real-time processing of streaming big data," *Real-Time Syst.*, vol. 53, no. 1, pp. 1–44, Jan. 2017.
- [18] J. A. Stankovic, "Misconceptions about real-time computing: A serious problem for next-generation systems," *Computer*, vol. 21, no. 10, pp. 10–19, Oct. 1988.
- [19] J. W. S. Liu, *Real-Time Systems*. Englewood Cliffs, NJ, USA: Prentice-Hall, 2000.
- [20] A. Menychtas, D. Kyriazis, and K. Tserpes, "Real-time reconfiguration for guaranteeing QoS provisioning levels in Grid environments," *Future Gener. Comput. Syst.*, vol. 25, no. 7, pp. 779–784, Jul. 2009.
- [21] S. Sharafeddine, K. Jahed, O. Farhat, and Z. Dawy, "Failure recovery in wireless content distribution networks with device-to-device cooperation," *Comput. Netw.*, vol. 128, pp. 108–122, Dec. 2017.
- [22] Z. Su, Q. Xu, J. Luo, H. Pu, Y. Peng, and R. Lu, "A secure content caching scheme for disaster backup in fog computing enabled mobile social networks," *IEEE Trans. Ind. Informat.*, to be published.
- [23] S. Sharafeddine and A. El Arid, "An empirical energy model for secure Web browsing over mobile devices," *Secur. Commun. Netw.*, vol. 5, no. 9, pp. 1037–1048, Sep. 2012.
- [24] A. Yadav, O. A. Dobre, and N. Ansari, "Energy and traffic aware full-duplex communications for 5G systems," *IEEE Access*, vol. 5, pp. 11278–11290, 2017.
- [25] K. N. R. S. V. Prasad, E. Hossain, and V. K. Bhargava, "Energy efficiency in massive MIMO-based 5G networks: Opportunities and challenges," *IEEE Wireless Commun.*, vol. 24, no. 3, pp. 86–94, Jun. 2017.
- [26] A. Mukherjee, "Energy efficiency and delay in 5G ultra-reliable low-latency communications system architectures," *IEEE Netw.*, vol. 32, no. 2, pp. 55–61, Mar./Apr. 2018.
- [27] M. R. Garey and D. S. Johnson, "Complexity results for multiprocessor scheduling under resource constraints," *SIAM J. Comput.*, vol. 4, no. 4, pp. 397–411, Dec. 1975.
- [28] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.
- [29] M. Joseph and P. Pandya, "Finding response times in a real-time system," *Comput. J.*, vol. 29, no. 5, pp. 390–395, May 1986.
- [30] S. K. Dhall and C. L. Liu, "On a real-time scheduling problem," *Oper. Res.*, vol. 26, no. 1, pp. 127–140, Feb. 1978.
- [31] K. Wang and Y. Cen, "Real-time partitioned scheduling in cloud-RAN with hard deadline constraint," in *Proc. IEEE Wireless Commun. Netw. Conf.*, Mar. 2017, pp. 1–6.
- [32] L. Zhang, K. Wang, D. Xuan, and K. Yang, "Optimal task allocation in near-far computing enhanced C-RAN for wireless big data processing," *IEEE Wireless Commun.*, vol. 25, no. 1, pp. 50–55, Feb. 2018.
- [33] A. A. Bertossi, L. V. Mancini, and F. Rossini, "Fault-tolerant rate-monotonic first-fit scheduling in hard-real-time systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 10, no. 9, pp. 934–945, Sep. 1999.
- [34] S. Ghosh, R. Melhem, and D. Mosse, "Fault-tolerance through scheduling of aperiodic tasks in hard real-time multiprocessor systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 8, no. 3, pp. 272–284, Mar. 1997.
- [35] P. Guo and Z. Xue, "QoS-aware fault-tolerant rate-monotonic first-fit scheduling in real-time systems," in *Proc. IEEE 2nd Inf. Technol. Netw., Electron. Autom. Control Conf.*, Dec. 2017, pp. 311–315.
- [36] Y. Li, T. Jiang, K. Luo, and S. Mao, "Green heterogeneous cloud radio access networks: Potential techniques, performance trade-offs, and challenges," *IEEE Commun. Mag.*, vol. 55, no. 11, pp. 33–39, Nov. 2017.
- [37] S. Wang, K. Li, J. Mei, G. Xiao, and K. Li, "A reliability-aware task scheduling algorithm based on replication on heterogeneous computing systems," *J. Grid Comput.*, vol. 15, no. 1, pp. 23–39, Mar. 2017.
- [38] Y. Ding, G. Yao, and K. Hao, "Fault-tolerant elastic scheduling algorithm for workflow in cloud systems," *Inf. Sci.*, vol. 393, pp. 47–65, Jul. 2017.
- [39] G. Xie et al., "Minimizing redundancy to satisfy reliability requirement for a parallel application on heterogeneous service-oriented systems," *IEEE Trans. Serv. Comput.*, to be published.

[40] D. Natale and Stankovic, "Dynamic end-to-end guarantees in distributed real time systems," in *Proc. Real-Time Syst. Symp. (REAL)*, Dec. 1994, pp. 216–227.

[41] S. Saha, A. Sarkar, and A. Chakrabarti, "Scheduling dynamic hard real-time task sets on fully and partially reconfigurable platforms," *IEEE Embedded Syst. Lett.*, vol. 7, no. 1, pp. 23–26, Mar. 2015.

[42] 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.

[43] Y. Li, M. Chen, W. Dai, and M. Qiu, "Energy optimization with dynamic task scheduling mobile cloud computing," *IEEE Syst. J.*, vol. 11, no. 1, pp. 96–105, Mar. 2017.

[44] X. Qin and H. Jiang, "A dynamic and reliability-driven scheduling algorithm for parallel real-time jobs executing on heterogeneous clusters," *J. Parallel Distrib. Comput.*, vol. 65, no. 8, pp. 885–900, Aug. 2005.

[45] W. Luo, J. Li, F. Yang, G. Tu, L. Pang, and L. Shu, "DYFARS: Boosting reliability in fault-tolerant heterogeneous distributed systems through dynamic scheduling," in *Proc. IEEE 8th ACIS Int. Conf. Softw. Eng. Artif. Intell. Netw., Parallel/Distrib. Comput.*, vol. 1, Jul./Aug. 2007, pp. 640–645.

[46] X. Zhu, X. Qin, and M. Qiu, "QoS-aware fault-tolerant scheduling for real-time tasks on heterogeneous clusters," *IEEE Trans. Comput.*, vol. 60, no. 6, pp. 800–812, Jun. 2011.

[47] J. Wang, W. Bao, X. Zhu, L. T. Yang, and Y. Xiang, "FESTAL: Fault-tolerant elastic scheduling algorithm for real-time tasks in virtualized clouds," *IEEE Trans. Comput.*, vol. 64, no. 9, pp. 2545–2558, Sep. 2015.

[48] T. Sigwele, A. S. Alam, P. Pillai, and Y. F. Hu, "Energy-efficient cloud radio access networks by cloud based workload consolidation for 5G," *J. Netw. Comput. Appl.*, vol. 78, pp. 1–8, Jan. 2017.

[49] G. Manimaran and C. S. R. Murthy, "A fault-tolerant dynamic scheduling algorithm for multiprocessor real-time systems and its analysis," *IEEE Trans. Parallel Distrib. Syst.*, vol. 9, no. 11, pp. 1137–1152, Nov. 1998.

[50] (2018). *Specpower_ssj2008 Results*. [Online]. Available: http://www.spec.org/power_ssj2008/results/power_ssj2008.html

[51] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Generat. Comput. Syst.*, vol. 28, no. 5, pp. 755–768, 2012.

[52] A. Varasteh and M. Goudarzi, "Server consolidation techniques in virtualized data centers: A survey," *IEEE Syst. J.*, vol. 11, no. 2, pp. 772–783, Jun. 2017.

[53] C. Clark et al., "Live migration of virtual machines," in *Proc. 2nd Conf. Symp. Netw. Syst. Design Implement.*, vol. 2. Berkeley, CA, USA: USENIX Association, Jan. 2005, pp. 273–286.

[54] H. Liu, H. Jin, C.-Z. Xu, and X. Liao, "Performance and energy modeling for live migration of virtual machines," *Cluster Comput.*, vol. 16, no. 2, pp. 249–264, 2013.

[55] A. Strunk, "A lightweight model for estimating energy cost of live migration of virtual machines," in *Proc. IEEE 6th Int. Conf. Cloud Comput.*, Jun./Jul. 2013, pp. 510–517.

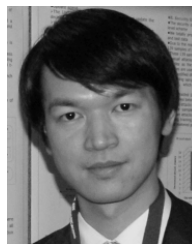
[56] V. De Maio, R. Prodan, S. Benedict, and G. Kecskemeti, "Modelling energy consumption of network transfers and virtual machine migration," *Future Gener. Comput. Syst.*, vol. 56, pp. 388–406, Mar. 2016.

[57] P. Guo and Z. Xue, "Real-time fault-tolerant scheduling algorithm with rearrangement in cloud systems," in *Proc. IEEE 2nd Inf. Technol., Electron. Autom. Control Conf.*, Dec. 2017, pp. 399–402.

[58] R. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw., Pract. Exper.*, vol. 41, no. 1, pp. 23–50, 2011.



MING LIU is currently a joint Ph.D. Student with the School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, China, and the Faculty of Engineering and Information Technologies, University of Technology Sydney, Australia. His research interests include cyber threat intelligence, intrusion detection systems, and scalable data analytics.



JUN WU (S'08–M'12) received the Ph.D. degree in information and telecommunication studies from Waseda University, Japan, in 2011. He was a Post-Doctoral Researcher with the Research Institute for Secure Systems, National Institute of Advanced Industrial Science and Technology, Japan, from 2011 to 2012. He was a Researcher with the Global Information and Telecommunication Institute, Waseda University, from 2011 to 2013. He is currently an Associate Professor with

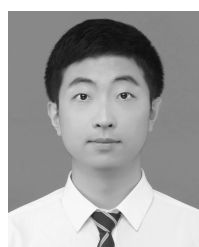
the School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, China, where he is also the Vice Director of the National Engineering Laboratory for Information Content Analysis Technology. His research interests include the advanced computing, communications and security techniques of software-defined networks, information-centric networks smart grids, Internet of Things, and fifth generation. He has authored over 100 refereed papers in these fields. He is a TPC member of more than 10 international conferences, including ICC, GLOBECOM, and WINCON. He is the Chair of the IEEE P21451-1-5 Standard Working Group. He has hosted and participated in a lot of research projects including the National Natural Science Foundation of China, the National 863 Plan and 973 Plan of China, and the Japan Society of the Promotion of Science Projects. He is a Guest Editor of the IEEE SENSORS JOURNAL. He is currently an Associate Editor of the IEEE ACCESS.



ZHI XUE received the Ph.D. degree in communication and information systems from the School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, China, in 2001. He is currently a Professor with Shanghai Jiao Tong University. His research interests include cyber security and cloud computing.



XIANGJIAN HE (M'99–SM'05) received the Ph.D. degree in computing sciences from the University of Technology Sydney, Australia, in 1999. Since 1999, he has been with the University of Technology Sydney. He is currently a Full Professor and the Director of the Computer Vision and Pattern Recognition Laboratory, Global Big Data Technologies Centre. He is a Co-Leader of the Network Security Research Team, Center for Real-Time Information Networks, University of Technology Sydney.



PENGZE GUO received the B.S. degree from the School of Electronic and Information Engineering, Xi'an Jiaotong University, China, in 2013. He is currently pursuing the Ph.D. degree in information and communication engineering with Shanghai Jiao Tong University. His research interests include real-time systems, fault tolerance, and cloud computing.