FESTAL: Fault-Tolerant Elastic Scheduling Algorithm for Real-Time Tasks in Virtualized Clouds

Ji Wang, Weidong Bao, Xiaomin Zhu, *Member, IEEE,* Laurence T. Yang, *Senior Member, IEEE,* and Yang Xiang, *Senior Member, IEEE*

Abstract—As clouds have been deployed widely in various fields, the reliability and availability of clouds become the major concern of cloud service providers and users. Thereby, fault tolerance in clouds receives a great deal of attention in both industry and academia, especially for real-time applications due to their safety critical nature. Large amounts of researches have been conducted to realize fault tolerance in distributed systems, among which fault-tolerant scheduling plays a significant role. However, few researches on the fault-tolerant scheduling study the virtualization and the elasticity, two key features of clouds, sufficiently. To address this issue, this paper presents a fault-tolerant mechanism which extends the Primary-Backup (PB) model to incorporate the features of clouds. Meanwhile, for the first time, we propose an elastic resource provisioning mechanism in the fault-tolerant context to improve the resource utilization. On the basis of the fault-tolerant mechanism and the elastic resource provisioning mechanism, we design novel fault-tolerant elastic scheduling algorithms for real-time tasks in clouds named FESTAL, aiming at achieving both fault tolerance and high resource utilization in clouds. Extensive experiments injecting with random synthetic workloads as well as the workload from the latest version of the Google cloud tracelogs are conducted by CloudSim to compare FESTAL with three baseline algorithms, i.e., Non-Migration-FESTAL (NMFESTAL), Non-Overlapping-FESTAL (NOFESTAL), and Elastic First Fit (EFF). The experimental results demonstrate that FESTAL is able to effectively enhance the performance of virtualized clouds.

Index Terms—cloud, fault-tolerant scheduling, elasticity, primary-backup model.

1 Introduction

The cloud, usually supported by virtualized data centers, has become a revolution paradigm for the on-demand provisioning of applications, platforms, and computing resources in a "pay-as-you-go" manner. Empowered by the large-scale resources in data centers and the virtualization technology, the cloud gives users the illusion of unlimited computing resources whose provisioning is elastic according to the users' demand [1]. While cloud computing enjoys numerous advantages, the large cloud data centers incur a high resources failure probability due to the increased functionality and complexity of the large systems [2]. This phenomenon is even more noticeable in most cloud service providers like Google, since they are built on inexpensive commodity hardware whose failure probability is expected to be much higher.

On the other hand, an increasing number of enterprises and research institutes have deployed their applications in clouds. Noticeably, many applications, e.g., financial transactions and scientific computing, are with real-time nature, where the correctness depends not only on the computation results, but also on the time instants at which they become available [3]. On account of the deadline-bound requirements of applications, it is essential to preserve the ability to provision computing services despite the occurrence of failures. Therefore, Fault tolerance for real-time tasks becomes a common challenge for clouds.

Scheduling is an efficient approach to achieving fault tolerance by allocating multiple copies of tasks on different computing instances. Among the researches on fault-tolerant scheduling, the Primary-Backup (PB) model is a popular scheme in which each task has two copies, namely, the primary and the backup, executing on two different computing instances for fault tolerance [4]. Extensive researches have been conducted to devise effective fault-tolerant scheduling algorithms based on the PB model in the traditional distributed systems such as grids and clusters [3], [5], [6], [7], [8], [9]. However, differing from the traditional distributed systems, clouds have distinct features:

- One of the vital technologies in cloud computing is virtualization which makes VMs become the basic computing instances, and enables VMs to migrate across the multiple-hosts;
- The scale of cloud is elastic according to the users' demand. It can be scaled up to satisfy the increased resource requests while scaled down to improve the resource utilization when the demand is relatively low.

The above features incur more difficulties for the research in clouds. The virtualization technology splits the system into

- *Ji Wang, Weidong Bao and Xiaomin Zhu are with the School of Information System and Management, National University of Defense Technology, Changsha, Hunan, P. R. China, 410073. E-mail:{wangji, wdbao, xmzhu}@nudt.edu.cn*
- *Laurence T. Yang is with the Department of Computer Science, St. Francis Xavier University, Antigonish, NS, B2G 2W5, Canada. E-mail: ltyang@stfx.ca*
- *Yang Xiang is with the School of Information Technology, Deakin University 221, Burwood, VIC 3125, Australia. E-mail: yang.xiang@deakin.edu.au*

two layers: hosts and VMs. One host's collapse results in multiple computing instances (i.e., VMs) failures, which is much tougher to tackle. Meanwhile, the VM migration as well as an elastic resource provisioning mechanism should be developed when designing the fault-tolerant scheduling algorithm to improve the system resource utilization, which makes the research further complex.

To the best of our knowledge, few researches have studied the above problems profoundly. As a result, it is hard for the existing fault-tolerant scheduling algorithms to fully benefit from clouds. Motivated by the need of high efficient fault-tolerant scheduling for real-time tasks in virtualized clouds, we gives substantial considerations to the virtualization and the elasticity, and then develops FESTAL, i.e., **F**ault-tolerant **E**lastic **S**cheduling algorithms for real-time **TA**sks in c**L**ouds. The main contributions of this work are as follows:

- We developed a novel fault-tolerant mechanism that extends the conventional PB model to incorporate the features of clouds;
- For the first time, we propose an elastic resource provisioning mechanism in the fault-tolerant context to optimize resource utilization while supporting fault tolerance in clouds;
- Based on the fault-tolerant mechanism and the elastic resource provisioning mechanism, we designed an efficient fault-tolerant scheduling algorithm − FESTAL;
- Extensive simulation experiments injecting with random synthetic workloads as well as the workloads form the Google tracelogs are conducted to validate the superiority of FESTAL.

The rest of this paper is organized as follows: Section 2 reviews related work. The problem formulation is presented in Section 3. Section 4 proposes the fault-tolerant mechanism. Based on the analysis in Section 4, the scheduling algorithms as well as the elasticity mechanism are developed in Section 5. Section 6 demonstrates the simulation experiments and the performance analysis. Finally, the paper is concluded by a summary and future work in Section 7.

2 Related Work

It is a challenging work to realize fault tolerance in distributed systems while optimizing resource utilization and task performance. To accomplish it, two fundamental technologies are widely applied: resubmission and replication [10]. Resubmission strives to re-execute the tasks on other normal computing instances after a failure. In MapReduce [11], to handle worker failures, the tasks on a failed worker are reset back to their initial states, and re-execute on other workers. Plankensteiner *et al*. proposed a new heuristic named Resubmission Impact to achieve fault tolerance in distributed systems [12]. Nonetheless, the resubmission may significantly delay the finish time of tasks after failures, which is catastrophic for real-time tasks due to their strict timing requirements. The system using replication technology executes several copies of the same task to support fault tolerance while guaranteeing finishing tasks before their deadlines. However, replication suffers from relatively large resource consumption, and results in a poor performance of the system.

Based on the replication technology, fault-tolerant scheduling tries to schedule the copies of tasks in an efficient manner that can enhance the schedulability, reliability and flexibility of computing systems effectively. Large amounts of researches have been conducted to develop efficient fault-tolerant scheduling algorithms, among which the Primary-Backup (PB) model is a widely used model where each task is replicated into two copies, i.e., the primary and the backup, scheduled on two different computing instances for fault tolerance [5], [6]. In order to reduce the extra resource consumption of backups, an effective technique called overlapping is investigated. By the overlapping technique, the backups are able to overlap in the same time slot on the same computing instances [4]. Qin and Jiang studied the backups overlapping of dependent tasks and designed a fault-tolerant scheduling algorithm in heterogeneous systems [3]. Nonetheless, the algorithm is static in nature, and not suitable for dynamic scheduling. In many scenarios, tasks are aperiodic, the arrival time and the deadline of which are unknown in advance, requiring dynamic scheduling algorithm. Based on the overlapping technique, Zheng *et al*. developed two dynamic algorithms named MRC-ECT and MCT-LRC to schedule backups of independent and dependent tasks, respectively [7]. In [9], the load-balancing technique as well as the overlapping technique were integrated with fault-tolerant scheduling to improve the performance of grids. Note that the backups in these literatures belong to the passive backup scheme, i.e., a backup is allowed to execute only if its primary fails. This scheme requires that the task has enough time laxity for backups, which may be unrealistic in practice. On the contrary, the active backup scheme allows backups to execute concurrently with its primaries, that is to say a backup can begin to execute even though its primary does not fail. Al-Omari *et al*. investigated how to adaptively control the backup scheme based on the task's time laxity [13]. Zhu *et al*. analyzed the overlapping technique with both the passive and active backup scheme, and proposed a QoS-aware fault-tolerant scheduling algorithm in heterogeneous clusters [8]. Although these algorithms exhibit good performances in traditional distributed systems, they have two common drawbacks when taking the features of clouds into account: (1) the virtualization technology is not considered, and the constraints of VM migration are not analyzed when using the PB model; (2) the elasticity of the system is not studied in the above literatures. Therefore, the existing fault-tolerant scheduling algorithms for the traditional systems are not suitable when applied to clouds. Recently, Antony *et al*. managed to devise a fault-tolerant scheduling algorithm based on the PB model in clouds to achieve fault tolerance as well as load-balancing [14]. However, neither the deadline restriction of real-time tasks nor the methods of improving the resource utilization is studied. In addition, none of the above literatures considers the resource management while scheduling tasks.

Nowadays, the resource management in clouds is a research hotsopt where large numbers of literatures have been produced. Samimi *et al*. proposed a new market model CDARA to optimize the resource allocation in clouds, which can generate

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2014.2366751, IEEE Transactions on Computers

higher revenues for both users and providers [15]. Warneke *et al*. developed a dynamic resource allocation framework for the parallel data processing in clouds; it enables the dynamic allocation and adjustment of resources during the execution of tasks [16]. Hsu *et al*. presented an energy-aware resource management technique by consolidating VMs among virtual clusters to restrict CPU use below a specified threshold [17]. These literatures focused on how to maximize revenue and minimize cost by optimizing the resource allocation in clouds, while no elastic resource provisioning was discussed. Sharma *et al*. proposed Kingfisher, a cost-aware system that efficiently supports for elasticity in clouds [18]. Beloglazov *et al*. developed some heuristics for real-time dynamic allocation of VMs to improve the system resource utilization by applying live migration, and switching idle hosts to sleep mode [19]. Vasic *et al* devised a framework DejaVu that enables the system to quickly adapt to workload changes [20]. Graubner *et al*. proposed an elastic scheduling algorithm that was based on VM consolidation to save energy [21]. Unfortunately, these works do not consider the task scheduling when adjusting the resource provisioning. Recently, Le *et al*. studied the adaptive resource management architecture as well as the task scheduling policy to dynamically provision resource according to the request workloads [22]. Zhang *et al*. proposed a heterogeneity-aware resource management system, HARMONY, that consists of dynamic resource provisioning, resource allocation and task scheduling [23]. Nonetheless, all these literatures do not take fault tolerance into consideration, and hence they cannot support fault tolerance while optimizing the resource management.

To the best of our knowledge, few previous literatures has studied the fault-tolerant scheduling and the elastic resource provisioning collectively. Thus, we design FESTAL that can achieve both fault tolerance and high performance in terms of resource utilization.

3 Problem Formulation

In this section, we introduce the models and notions used in this paper.

3.1 System model

The cloud in this work is characterized by an infinite set $H = \{h_1, h_2, ...\}$ of hosts with different processing power P_i that is measured by Million Instructions per Second (MIPS, a widely used metric [15], [24], [25], [26]), representing the infinite heterogeneous computing resources in the cloud. Note that not all the hosts in the cloud is active, the count of active hosts is actually limited in practice. So the active hosts set is modeled by $H_a = \{h_1, h_2, ..., h_{|H_a|}\}$, $H_a \subseteq H$, while the items in $H - H_a$ represent the hosts in sleep status. The hosts can be dynamically switched between the active and the sleep status according to the system workload. Each host h_i contains multiple VMs denoted by a set $V_i = \{v_{i1}, v_{i2}, ..., v_{i|V_i|}\}$ of VMs with different processing power P_{ij}. Also, the VMs can be dynamically created and cancelled based on the system workload. Meanwhile, the VMs can migrate across the active hosts in order to consolidate resources and improve the resource utilization.

We use a set $T = \{t_1, t_2, ..., t_n\}$ of independent and non-preemptive tasks. Each task t_i has three attributes: arrival time a_i, deadline d_i, and task size ts_i that is measured by Million of Instructions (MI) similar to [25], [26], [27], [28]. Each task t_i has two copies denoted as t_i^P and t_i^B executed on two different hosts for fault tolerance. In clouds, the copies of tasks are allocated to VMs rather than hosts directly. $vm(t_i^P)$ and $vm(t_i^B)$ denote the VMs where t_i^P and t_i^B are allocated respectively, while $host(t_i^P)$ and $host(t_i^B)$ represent the corresponding hosts. In a virtualized cloud, it usually takes seconds to deliver the task to a VM, and deploy it. Then, the task is ready for execution. $r_{kl}(t_i^P)$ and $r_{kl}(t_i^B)$ denote the ready time of t_i^P and t_i^B on v_{kl}. The execution time of tasks' copies is assumed to be estimated a priori, which is a common assumption in scheduling research [29]. In this work, we apply a widely used approach that is based on the task size and the processing power to estimate the execution time [24], [28], [30], [31]: on the VM v_{kl}, the execution time $e_{kl}(t_i)$ of task t_i's copy is the ratio of its task size over the processing power of this VM. For instance, $vm(t_i^P) = v_{kl}$ means that the primary t_i^P of t_i is scheduled on v_{kl} that is on h_k, and $host(t_i^P) = h_k$, $e_{kl}(t_i) = ts_i/P_{kl}$.

Both the active backup and the passive backup schemes are adopted in this work. When a task arrives at the system, its primary is scheduled before its backup. The status of backup scheduled on v_{kl}, denoted by $st(t_i^B)$, is adaptively determined by the following expression:

$$st(t_i^B) = \begin{cases} passive & \text{if } f_i^P + e_{kl}(t_i) \le d_i, \\ active & \text{if } f_i^P + e_{kl}(t_i) > d_i. \end{cases} \quad (1)$$

To diminish the actual execution time of backups, we introduce the technique developed in [32] to terminate the execution of backups and reclaim the resource when the backups' corresponding primaries finish successfully.

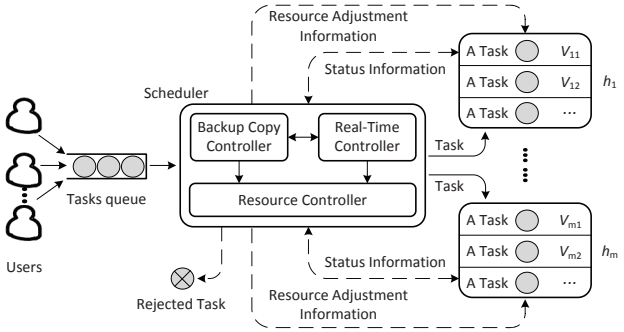


Fig. 1. Scheduling architecture.

The scheduling architecture is shown in Fig. 1. The scheduler consists of a backup copy controller, a real-time controller, and a resource controller. This architecture adopts the star-topology communication model that is widely studied in clouds [14], [19], [26] where the scheduler is responsible for scheduling all the tasks to the hosts, and monitoring

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2014.2366751, IEEE Transactions on Computers

the status of all the hosts. The hosts in the cloud report their status information such as tasks' execution status to the scheduler directly. Considering our work focuses on the computing-extensive tasks, both the results of tasks and the status information of hosts are assumed to be small-data. Hence, the communication delay is fixed and small.

When a task arrives, its backup copy is produced by the backup copy controller. Then, the backup copy controller delivers the two copies of the task to the real-time controller that is responsible for determining whether the two copies can be finished before its deadline. If not, the real-time controller informs the resource controller to add new resources. If no schedule can be found to satisfy the tasks' timing constraint although new resources have been added, the task will be rejected. In addition, the resource controller monitors the status of resources. When the system is in light-workload where some VMs keep idle for a long time, the resource controller decides whether some VMs should be cancelled to improve the resource utilization.

If the primary of a task finishes successfully, the success information will be sent to the backup copy controller. Then, the backup copy controller informs the VM where the task's backup is allocated to cancel the execution of the backup. On the other hand, when a host fails the fault detection mechanism will detect the failure whose information then will be reported to the backup copy controller. In this situation, the backup copy controller does not inform the VM to cancel the execution of backups. And backups will execute based on the schedule normally for fault tolerance.

3.2 Fault model

In this work, we focus on the failure of hosts. If a host fails, the copies of tasks on this host will fail to finish. At one time instant, at most one host may encounter a failure, that is to say, if the primaries of tasks fail, the backups can always finish successfully before another host fails. Failures can be either transient or permanent, and are independent, affecting only a single host.

In addition, there exists a failure detection mechanism such as fail-signal and acceptance test [4], [5], providing information of failures. New tasks will not be scheduled to a known failed host.

Note that it is straightforward to extend this model to tolerate multiple host failures. First of all, we divide the hosts into several small groups. Then, the above fault model is applied in each small group. Finally, the proposed fault-tolerant mechanism is employed in each small group like that in [4] to handle multiple host failures.

3.3 Scheduling objectives

The main objective in this study is to accommodate as many tasks as possible, and enhance the system resource utilization while adopting the PB fault-tolerant model. As a result, the first scheduling objective is to maximize the number of accepted tasks under timing constraints.

In order to enhance the resource utilization, the active time of hosts (i.e., the time duration of hosts in active status) should be fully utilized. Hence, another scheduling objectives, maximizing the resource utilization, can be expressed as follows:

$$RTH = \max_{t_i \in T, h_k \in H} \{TET/HAT\}, \quad (2)$$

where TET denotes the total execution time of all tasks, TET is equal to $\sum_{t_i \in T} \sum_{h_k \in H} \sum_{v_{kl} \in V_k} e_{kl}(t_i^P) x_{ikl}^P + e_{kl}(t_i^B) x_{ikl}^B$, x_{ikl}^P (or, x_{ikl}^B) equals 1 if and only if t_i^P (or, t_i^B) is allocated to vm_{kl}, otherwise $x_{ikl}^P = 0$ (or, $x_{ikl}^B = 0$); HAT represents the total active time of all the hosts in a cloud, HAT equals $\sum_{h_k \in H} at_k$, at_k denotes the active time of h_k.

The above objectives suggest that our fault-tolerant scheduling strategy strives to accommodate more tasks within timing constraints and decrease the total active time of hosts.

4 Fault-Tolerant Mechanism

The PB model is employed in our work for fault tolerance. Since one host's failure will result in the failure of all the VMs on the host, two copies of one task is not allowed to be scheduled on the VMs that are in the same host. The primary and the backup of one task must be allocated on different hosts for fault tolerance.

The conventional PB model consumes at least double resources for the sake of fault tolerance. To efficiently boost the resources utilization and thereby to improve the system performance, we attempt to extend the PB model by introducing the BB overlapping technique and the VM migration technique in the context of virtualized clouds. In this section, we will analyze the constraints of the BB overlapping and the VM migration to achieve fault tolerance in a virtualized cloud.

4.1 BB overlapping

According to the analysis in [7], backups can overlap with each other when their primaries are on different computing instances in the traditional distributed systems. However, the situation becomes much more complex in virtualized clouds.

Because the basic computing instances in virtualized clouds are VMs rather than hosts. It is the fundamental premise of BB overlapping that the primaries are not scheduled on the VMs that are on the same host.

Theorem 1: If $host(t_i^P) = host(t_j^P)$, then t_i^B cannot overlap t_j^B, regardless of whether $vm(t_i^P) = vm(t_j^P)$ or not.

Proof: Prove by contradiction. Suppose that $host(t_i^P) = host(t_j^P) = h_1$, and t_i^B overlaps t_j^B. When h_1 fails, all the VMs in h_1 fail, incurring the failures of the copies on these VMs, including t_i^P and t_j^P. Thus, t_i^B and t_j^B must execute. Since t_i^B and t_j^B execute simultaneously in the same VM, a timing conflict happens. Contradiction happens. Therefore, t_i^B cannot overlap t_j^B. □

Without loss of generality, we assume that t_j is a newly-arrived task to be scheduled and two copies of task t_i have been scheduled. In terms of the scheme of t_i^B, two cases should be discussed respectively, that is $st(t_i^B) = passive$ and $st(t_i^B) = active$.

Case 1. $st(t_i^B) = passive$, $host(t_i^P) \neq host(t_j^P)$, $vm(t_i^B) = vm(t_j^B)$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2014.2366751, IEEE Transactions on Computers

Fig. 2 depicts two scenarios of this case. In Fig. 2(a), t_j^B adopts passive backup scheme. According to the assumptions in Section 3.2, at most one host may encounter a failure at one time instant, thus t_i^B can overlap t_j^B. Let $LST_{kl}(t_j^B)$ denote the latest start time of t_j^B on v_{kl}. To satisfy the timing constraint of task t_j, we can derive the following equation:

$$LST_{kl}(t_j^B) = d_j - e_{kl}(t_j). \quad (3)$$

t_j^B adopts active backup scheme in Fig 2.(b). The execution of t_j^B is divided into two parts: t_j^{BY} that executes with t_j^P concurrently and t_j^{BN} that executes only if t_j^P fails and can be terminated when t_j^P finishes successfully. The overlapping between t_i^B and t_j^B is infeasible in this scenario.

Theorem 2: If $st(t_i^B) = passive$, $st(t_j^B) = active$, then t_i^B cannot overlap t_j^B.

Proof: Prove by contradiction. Suppose t_i^B overlaps with t_j^B. If $host(t_i^P)$ fails, t_i^B is invoked. However, $st(t_j^B) = active$, t_j^{BY} also executes. In the same VM, t_j^{BY} and t_i^B execute simultaneously. Contradiction happens. Thus, t_i^B cannot overlap t_j^B. □

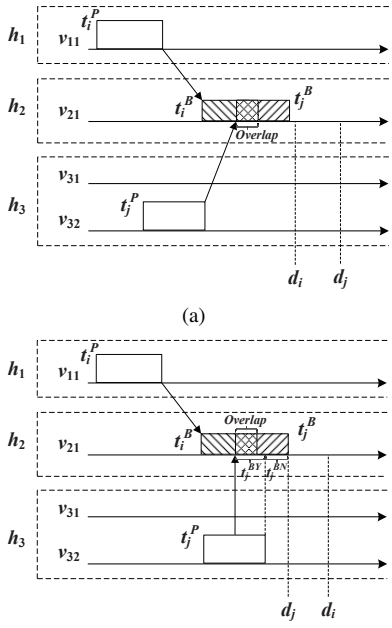


Fig. 2. Illustration examples of $st(t_i^B) = passive$.

Case 2. $st(t_i^B) = active$, $host(t_i^P) \neq host(t_j^P)$, $vm(t_i^B) = vm(t_j^B)$.

Fig. 3 depicts two scenarios of the case of t_i^B adopting active backup scheme where t_i^B is composed of t_i^{BY} and t_i^{BN}. In Fig. 3(a), t_j^B is able to overlap t_i^B when the following theorem is satisfied.

Theorem 3: Let $EST_{kl}(t_j^B)$ denote the earliest start time of t_j^B. On the VM v_{kl}, $vm(t_i^B) = vm(t_j^B) = v_{kl}$, If $st(t_i^B) = active$, $st(t_j^B) = passive$, and t_i^B overlaps t_j^B, then

$$EST_{kl}(t_j^B) \geq f_i^P + \varepsilon, \quad (4)$$

where ε is the time to cancel the execution of t_i^B if t_i^P finishes successfully.

Proof: Prove by contradiction. Suppose that $s_j^B < f_i^P$, and t_i^B overlaps t_j^B, so t_i^{BY} overlaps t_j^B. When $host(t_j^P)$ fails, t_j^B is invoked. But t_i^{BY} is executing, and cannot be interrupted, which results in a timing conflict between t_i^{BY} and t_j^B. Contradiction happens. Therefore, $EST_{kl}(t_j^B)$ must be later than $f_i^P + \varepsilon$. □

t_j^B adopts active backup scheme in Fig. 3(b). But t_i^B is not allowed to overlap t_j^B in this scenario, the proof of which is similar to Theorem 2. If $host(t_i^P)$ fails, the overlapping between t_i^B and t_j^B will result in a timing conflict between t_i^B and t_j^{BY}.

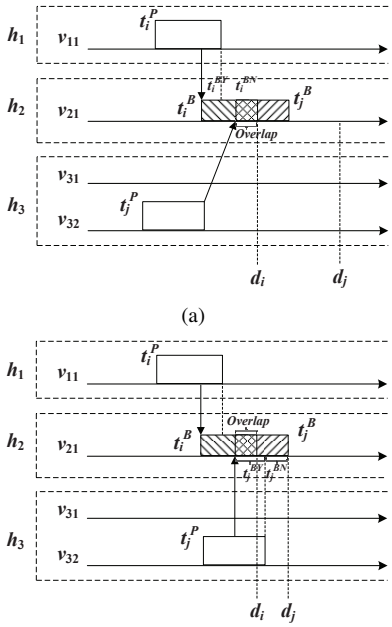


Fig. 3. Illustration examples of $st(t_i^B) = active$.

4.2 VM migration

The overlapping technique improves the resource utilization by reducing the backup's usage of available processor time, which nonetheless only functions at the VM level. Further improvement is supposed to be achieved at the host level. The VM migration, a widely used technique in the cloud resource management, is able to further optimize the resource utilization at the host level. In order to guarantee fault tolerance when VMs migrate across hosts, the constraints of VM migration are analyzed as below.

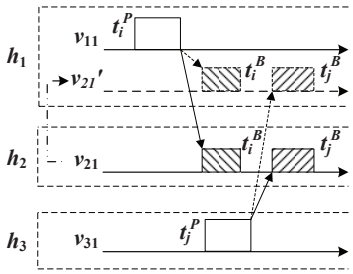
Theorem 4: Let fH_{kl} be a set of hosts. $fH_{kl} = \{host(t_i^P) | \forall t_i \in T, vm(t_i^B) = v_{kl}\} \cup \{host(t_i^B) | \forall t_i \in T, vm(t_i^P) = v_{kl}\}$. v_{kl} cannot migrate to $h_j, \forall h_j \in fH_{kl}$.

Proof: Prove by contradiction. Suppose that v_{kl} migrates to $h_j, \forall h_j \in fH_{kl}$. Then, the primary and the backup of one task are allocated on the same host. If this host fails, both the primary and the backup fail to finish successfully. Obviously, fault tolerance cannot be realized. Contradiction happens. As a result, the migration is infeasible. An illustration example is shown in Fig. 4(a). □

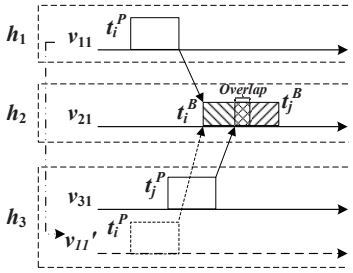
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2014.2366751, IEEE Transactions on Computers

Theorem 5: $\forall t_i \in T$, $vm(t_i^P) = v_{kl}$, if t_i^B overlaps t_j^B, then v_{kl} cannot migrate to $host(t_j^P)$.

Proof: Prove by contradiction. Suppose that v_{kl} migrates to $host(t_j^P)$. Then, t_i^P and t_j^P are on the same host while t_i^B overlaps t_j^B, which violates Theorem 1. Contradiction happens. As a result, the migration is infeasible. An illustration example is shown in Fig. 4(b). □



(a) An example of Theorem 4.



(b) An example of Theorem 5.

Fig. 4. Illustration examples of the constraints of the VM migration.

5 Fault-Tolerant Elastic Scheduling Algorithms—FESTAL

In this section, we develop an elastic resource provisioning mechanism in the context of fault tolerance. Then, based on the fault-tolerant mechanism, we devise a novel fault-tolerant scheduling algorithm FESTAL that incorporates the elastic resource provisioning mechanism. The main goal of FESTAL is to enhance the system schedulability and resource utilization so long as achieving fault tolerance.

5.1 Elastic Resource Provisioning Mechanism

FESTAL incorporates an elastic resource provisioning mechanism that is able to dynamically adjust the resource provisioning based on the resource request. The elastic resource provisioning mechanism consists of the resource scaling-up mechanism and the resource scaling-down mechanism. When copies of tasks cannot be scheduled on the existing resources, the resource scaling-up mechanism is called to augment the resources. Meanwhile, if some resources are idle for a long time during the operation of the system, the resource scaling-down mechanism will work to eliminate the idle resources, and so improve the resource utilization.

If a copy of task t_i cannot be scheduled on any existing VM, the resource scaling-up mechanism will be called to create a new VM and add it into the system. The processing power P_{new} of the new VM satisfies the following expression:

$$r_{new}(t_i) + ts_i/P_{new} + delay < d_i, \tag{5}$$

where $r_{new}(t_i)$ denotes the ready time of t_i on the new VM, $delay$ denotes the time delay coming from the creation time of a new VM, the time delay of VM migration and the boot time of new active hosts. The pseudocode for the scaling-up mechanism is presented in Algorithm 1. The mechanism first tries to allocate the new VM on the existing active hosts (see lines 3-6). If there is no suitable active host, the system migrates some VMs among the active hosts to make room for the new VM (see lines 7-11). If it still fails, a host in sleep status will be turned on, and then the new VM is allocated on it (see lines 12-17).

Algorithm 1: Function scaleUpResources()

```
1  Select a kind of newVm satisfying the equation (5);
2  Sort H_a in a decreasing order by the remaining MIPS;
3  foreach h_k in H_a do
4      if h_k can accommodate newVm then
5          Create newVm on h_k;
6          return newVm;
7  foreach h_s in H_a do
8      Migrate the VM with minimal MIPS in h_s to other hosts
       with the fault-tolerant requirements of Theorems 4 and 5;
9      if h_s can accommodate newVm then
10         Create newVm on h_s;
11         return newVm;
12 Turn on a host h_new in H − H_a;
13 if the MIPS of h_new satisfies newVm then
14     Create newVm on h_new;
15     return newVm;
16 else
17     return NULL;
```

When some VMs are idle for a relatively long time, it means that the system is in the light workload and some resources are wasted. Thereby, the resource scaling-down mechanism functions to cancel the idle VMs and switch some active hosts to the sleep status for better resource utilization. For each VM, we set a time instant T_{cancel} at which the VM is expected to be cancelled. When a task's copy is scheduled on the VM, T_{cancel} is updated as follows:

- When a primary t_i^P is scheduled on the VM, $T_{cancel} = \max\{f_i^P + T_{idle}, T_{cancel}\}$;
- When a backup t_i^B is scheduled on the VM, $T_{cancel} = \max\{f_i^P + T_{idle}, T_{cancel}\}$; if t_i^P encounters a fault, and t_i^B is due to execute, $T_{cancel} = \max\{f_i^B + T_{idle}, T_{cancel}\}$.

Fig. 5 illustrates the updating of T_{cancel}. The VM will be cancelled if no task executes on the VM during a period of T_{idle}. In order to make full use of the resources during T_{idle} and to reduce the resource consumption of backups, backups can be scheduled to finish or even start after T_{cancel} as backups may be deallocated if primaries finish successfully. For example, in Fig. 5(a), t_i^B starts later than T_{cancel}. Because t_i^B will be deallocated if t_i^P succeeds, it is a waste of resource for v_{21} to be cancelled after the execution of t_i^B. However,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2014.2366751, IEEE Transactions on Computers

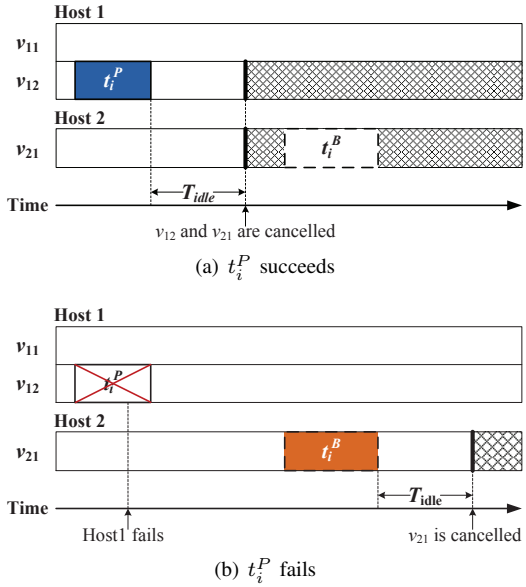


Fig. 5. Illustrations of the updating of T_{cancel}.

when t_i^P fails, t_i^B is due to execute for fault tolerance, and so the T_{cancel} is postponed (in Fig. 5(b)). Furthermore, by this approach the system can adaptively adjust the resource provisioning according to the host reliability. When the hosts are of high reliability (i.e., few hosts fail), more VMs for backups will be cancelled to save resource, and vice versa.

Algorithm 2 presents the pseudocode for the scaling-down mechanism. If there exists any VM whose idle time reaches the preestablished threshold $v_{kl}.T_{cancel}$, then this VM will be canceled (see lines 2-3). After this operation, if the processing power utilization of the host falls below the lower threshold U_{low}, the VMs on the host will try to migrate from this host (see lines 4-14). If all the VMs on the host can migrate to other hosts, the host will be switched to the sleep status to eliminate the idle resource consumption (see lines 15-17). Otherwise, the system gives up the migration operation of all the VMs on the host (see line 19).

Algorithm 2: Function scaleDownResources()

```
1  foreach VM v_kl in the cloud system do
2      if it reaches the time v_kl.T_cancel then
3          Remove v_kl from h_k and cancel it;
4          if h_k.utilization ≤ U_low then
5              offTag ← TRUE;
6              foreach v_kl in h_k do
7                  migTag ← FALSE;
8                  foreach h_i in H_a except h_k do
9                      if h_i can accommodate v_kl & the
                       migration meets fault-tolerant
                       requirements of Theorems 4 and 5 then
10                         migTag ← TRUE;
11                         break;
12                 if migTag == FALSE then
13                     offTag ← FALSE;
14                     break;
15             if offTag then
16                 Migrate VMs in h_k to destination hosts;
17                 Switch h_k to sleep status and remove it from
                   H_a;
18             else
19                 Give up the migration operation;
```

5.2 Algorithms for Scheduling Primaries and Backups

The algorithms of FESTAL for scheduling primaries and backups are in heuristic fashion. In order to reserve sufficient time laxity for backups, the primaries should finish as early as possible. Besides, according to Theorem 1, primaries should not concentrate on several hosts, which otherwise makes backups' overlapping infeasible, and consequently results in more resource consumption for backups. Hence, the primaries scheduling should reach the following two objectives:

- The finish time of primaries should be as early as possible;
- The primaries should be evenly scheduled on the active hosts.

On the basis of the two objectives, we propose an improved "As Early As Possible" (I-AEAP) strategy. The pseudocode for the I-AEAP is presented in Algorithm 3. I-AEAP first determines the candidate hosts on which fewer primaries are scheduled: the top $\alpha\%$ hosts with fewer primaries are chosen as candidate hosts (see lines 1-2). Then, the VM where the finish time of the primary is the earliest is chosen to execute the primary (see lines 5-12). If there is no VM among the VMs on the candidate hosts, the next top $\alpha\%$ hosts will be chosen for the next round of searching (see lines 13-16). With the help of I-AEAP, the newly-arrived primary is scheduled on the host with fewer primaries preferentially to make primaries evenly scheduled while it also can finish as early as possible. When it fails to find a suitable VM to finish the primary before its deadline, the scaling-up mechanism is called to add a new VM for the primary (see lines 17-18). If it still fails due to the tight deadline, the task will be rejected (see line 23).

We now evaluate the time complexity of primaries scheduling in FESTAL. Suppose we consider scheduling a primary copy of task t_i. N_a denotes the number of active hosts in the cloud. $N_{vm}(k)$ represents the number of VMs on host h_k. $N_{tw}(kl)$ represents the number of existing scheduled copies on VM v_{kl}. N_{vm} denotes the maximum number of $N_{vm}(k)$ among all the active hosts in the cloud. N_{tw} represents the maximum number of $N_{tw}(kl)$ among all the VMs on active hosts.

Theorem 6: The time complexity of primaries scheduling is $O(N_a N_{vm} N_{tw} + N_a^2)$ in the worst situation.

Proof: For determining the earliest finish time of t_i^P on VM v_{kl}, all the existing scheduled copies are tested, so the time complexity is $O(N_{tw}(kl))$. Then, it takes $O(N_{vm}(k) \cdot \max_{v_{kl} \in V_k}\{N_{tw}(kl)\})$ to scan all the VMs on host h_k. Hence, it takes $O(N_a N_{vm} N_{tw})$ in the worst situation where all the active hosts are tested.

Then it comes to the function scaleUpResources(). Firstly, it takes $O(N_a)$ to allocate a new VM to an active host. For

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2014.2366751, IEEE Transactions on Computers

Algorithm 3: Primaries Scheduling in FESTAL

```
1  Sort H_a in an increasing order by the count of scheduled primaries;
2  H_candidate ← top α% hosts in H_a;
3  find ← FALSE; EFT ← +∞; v ← NULL;
4  while !all hosts in H_a have been scanned do
5      foreach h_k in H_candidate do
6          foreach v_kl in h_k.VmList do
7              Calculate the earliest finish time EFT_kl(t_i^P);
8              if EFT_kl(t_i^P) ≤ d_i then
9                  find ← TRUE;
10                 if EFT_kl(t_i^P) ≤ EFT then
11                     EFT ← EFT_kl(t_i^P);
12                     v ← v_kl;
13     if find == FALSE then
14         H_candidate ←next top α% hosts in H_a;
15     else
16         break;
17 if find == FALSE then
18     v ←scaleUpResources();
19 if find == TRUE then
20     Allocate t_i^P to v;
21     Update the T_cancel of v;
22 else
23     Reject t_i^P;
```

line 8 in Algorithm 1, it takes $O(N_a)$ to find a destination host. Hence, the time complexity of adding new VMs by VM migration (see lines 7-11) is $O(N_a^2)$. For allocating a new VM to a new turned-on active host, the time complexity is $O(1)$. Therefore, the complexity of scaleUpResources() is $O(N_a + N_a^2 + 1) = O(N_a^2)$ as N_a^2 is larger or at least equal to N_a and 1. As a result, the overall worst-situation time complexity of primary scheduling in FESTAL is $O(N_a N_{vm} N_{tw} + N_a^2)$. □

The backups are expected to be allocated on the hosts with fewer primaries, and so that the hosts mainly accommodate backups can be switched off timely when the system is of high reliability and few backups are due to execute. Besides, in order to make full use of the idle resources, the system should try to schedule backups on the VM with the maximum T_{cancel}. This policy can be attributed to the resource scaling-down mechanism proposed above in which the allocation of backups does not postpone the VM's T_{cancel} in most cases. Therefore, the choice of the VM with the maximum T_{cancel} can help the system fully utilize the processor time during T_{idle} by backups. In addition, the backups should try to adopt the passive backup scheme for the reason that the passive backups can be deallocated and are easier to overlap others than active backups, which can further reduce the resource consumption of backups. In conclusion, three scheduling policies are listed as follows:

- The backups should concentrate on several active hosts;
- The backups should be scheduled on the VM with the maximum T_{cancel};
- The backups should adopt passive backup scheme as much as possible.

Based on the three policies, an improved "As Late As Possible" (I-ALAP) strategy is developed. The pseudocode for the I-ALAP is presented in Algorithm 4. I-ALAP first manages to find a proper VM from the hosts on which no primaries are scheduled (see line 1). Then, I-ALAP allocates the backup on the VM with the maximum T_{cancel} and the start time of the backup is the latest among the VMs that are on the candidate hosts (see lines 5-13). If no VM is found from these hosts, I-ALAP chooses the hosts with fewer primaries as the candidate hosts (see lines 14-17). When it fails to find a suitable VM to finish the backup before its deadline, the scaling-up mechanism is called to add a new VM for the backup (see lines 18-19). If it still fails, both the primary and the backup will be rejected (see line 24).

Algorithm 4: Backups Scheduling in FESTAL

```
1  H_candidate ←the hosts on which no primaries are scheduled;
2  H_primary ←Sort H_a − H_candidate in an increasing order by the count of scheduled primaries;
3  find ← FALSE; v ← NULL; T ← MAX; LST ← 0;
4  while !all hosts in H_primary have been scanned do
5      foreach h_k in H_candidate do
6          foreach v_kl in h_k.VmList do
7              Calculate the latest start time LST_kl(t_i^B);
8              if LST_kl(t_i^B) + e_kl(t_i) ≤ d_i then
9                  find ← TRUE;
10                 if v_kl.T_cancel < T || v_kl.T_cancel == T &
                   LST_kl(t_i^B) > LST then
11                     T ← v_kl.T_cancel;
12                     LST ← LST_kl(t_i^B);
13                     v ← v_kl;
14     if find == FALSE then
15         H_candidate ←next top α% hosts in H_primary;
16     else
17         break;
18 if find == FALSE then
19     v ←scaleUpResources();
20 if find == TRUE then
21     Allocate t_i^B to v;
22     Update the T_cancel of v;
23 else
24     Reject t_i^P; Reject t_i^B;
```

Then, we analyze the time complexity of backups scheduling in FESTAL. Suppose we consider scheduling a backup copy of task t_i. The nomenclatures are the same as those in the analysis of primaries scheduling.

Theorem 7: The time complexity of backups scheduling is $O(N_a N_{vm} N_{tw} + N_a^2)$.

Proof: For determining the latest start time of t_i^B on VM v_{kl}, the time complexity is $O(N_{tw}(kl))$. So the complexity for all the active hosts is $O(N_a N_{vm} N_{tw})$. Considering the complexity of scaleUpResources(), the overall time complexity of backup allocation in FESTAL is $O(N_a N_{vm} N_{tw} + N_a^2)$. □

6 Performance Evaluation

To reveal the performance improvements of FESTAL, we compare it with three baseline algorithms, i.e.,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2014.2366751, IEEE Transactions on Computers

TABLE 1
Parameters of workloads

Parameter	Value(Fixed)−(Min,Max,Step)
task count($\times 10^4$)	(1) − (0.5,4,0.5)
task size($\times 10^5$MI)	([1,2])
Interval time $1/\lambda$	(2) − (0,14,2)
baseDeadline($\times 10^2$s)	(4) − (1,4.5,0.5)

Non-Migration-FESTAL (NMFESTAL), Non-Overlapping-FESTAL (NOFESTAL) and Elastic First Fit (EFF). The three baseline algorithms are briefly described as follows:

- NMFESTAL: NMFSTAL is a variant of FESTAL. The difference between FESTAL and NMFESTAL lies in that NEFESTAL does not employ the VM migration technique.
- NOFESTAL: NOFSTAL is a variant of FESTAL. Differing from FESTAL, the overlapping technique is not employed in the NOFSTAL.
- EFF: EFF derives from the classical First Fit algorithm that has been widely used in industry. To adapt First Fit algorithm to virtualized clouds, the elastic resource provisioning mechanism is integrated into EFF.

We compare these algorithms from the respect of the following three metrics:

- Guarantee Ratio (*GR*) is defined to be the percentage of tasks that are guaranteed to finish successfully among all the submitted tasks;
- Hosts Active Time (*HAT*) is defined to be the total active time of all the hosts in cloud, reflecting the resource consumption of the system;
- Ratio of Task time over Hosts time (*RTH*) is defined to be the ratio of the total tasks' execution time over the total active time of hosts, reflecting the resource utilization of the system.

In order to ensure the repeatability of the experiments, we use the approach of simulation to evaluate the performance of aforementioned algorithms. CloudSim [24] is chosen to simulate the cloud data center in our experiments. CloudSim, supporting the seamless modeling, simulation, and experimentation of large-scale cloud infrastructures, is a widely recognized simulator in both industry and academia. The detailed setting and parameters used in CloudSim are listed as follows:

- Each host is modeled as performance equivalent to 1000, 1500, 2000 or 3000 MIPS;
- Four types of VMs with the processing power equivalent to 250, 500, 700 and 1000 MIPS are considered;
- According to [33], the time required for turning on a host and creating a VM is set as 90s and 15s, respectively.

6.1 Evaluation based on random synthetic workloads

In order to clearly observe the impacts of different parameters of workloads on performance of clouds, we conduct the experiments based on the random synthetic workloads. Tasks are assumed to arrive at the cloud following Poisson distribution with the average interval time $1/\lambda$ that is uniformly distributed in the range $[1/\lambda, 1/\lambda + 2]$. The task sizes are uniformly distributed in the range $[1 \times 10^5, 2 \times 10^5]$ MI. The deadline is designated as $d_i = a_i + deadlineTime$, and $deadlineTime$ subjects to the uniform distribution, $U(baseDeadline, 4baseDeadline)$. In each group of experiment, we change a single parameter while keeping the other parameters fixed. Table 1 gives the parameters and their values. Each experiment runs 50 times.

6.1.1 Performance impact of task count

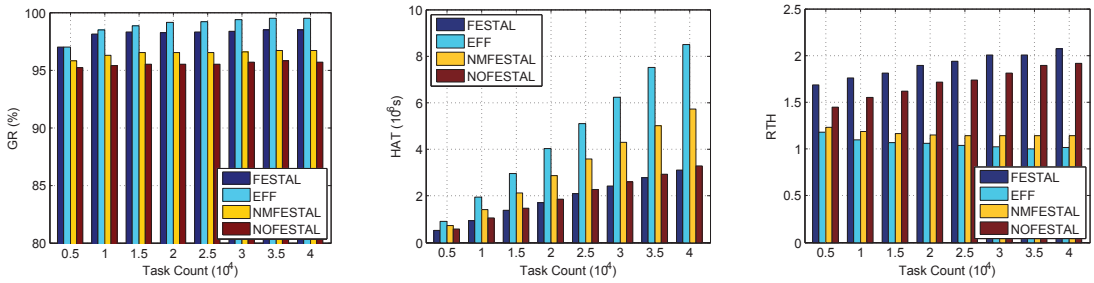
In this experiment, we investigate the performance impact of task count that increases from 5,000 to 40,000 with step of 5,000. Fig. 6 depicts the performances of FESTAL, EFF, NMFESTAL and NOFESTAL.

Fig. 6(a) shows that with the increase of task count, the four algorithms all maintain high guarantee ratios, which can be attributed to the infinite resources in the cloud. When the task count increases, the system can satisfy the new request by adding new resources. Despite of the infinite resources in the cloud, there are still some tasks submitted to the system cannot be accepted. This is because of the additional time needed to create a new VM and turn on a new host, which causes that tasks cannot start timely, and thus miss their deadlines. Besides, it can be found that FESTAL and EFF have higher guarantee ratios than NMFESTAL and NOFESTAL. However, the reasons for the two algorithms are different. For FESTAL, the comprehensive employment of the proposed techniques and mechanisms enhances the schedulability of the system, and hence it has a higher guarantee ratio. While the high value of EFF mainly comes from the much more resource consumption which is apparent in Fig. 6(b).

Fig. 6(b) demonstrates that the *HAT*s of the four algorithms keep ascending trend with the increase of task count. This is explainable in that more active hosts are needed to execute more tasks accepted by the system. Additionally, EFF and NMFESTAL perform much worse than FESTAL and NOFESTAL in terms of *HAT*. The bad performance of EFF argues that the classical scheduling algorithm is not suitable for the fault-tolerant scheduling in clouds. This is because without the I-AEAP and the I-ALAP, backups are mixed with primaries on all the VMs of active hosts. The idle time of VMs cannot be fully used by backups, and VMs cannot be cancelled timely after the deallocating of backups. Hence, the active hosts with theses VMs keep active, which definitely raises the *HAT*. Besides, the poor exhibition of NMFESTAL indicates that employing the VM migration technique is very efficient in fault-tolerant scheduling. On one hand, when the task count increases, current VMs can be consolidated to make some room for creating new VMs, which avoids the consumption caused by adding new active hosts. On the other hand, the VMs in light-load host can migrate to other hosts and then the idle hosts can be switched off, which further reduces the *HAT*.

This advantage of FESTAL and NOFESTAL is further exhibited in Fig. 6(c). The highest *RTH* of FESTAL suggests

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2014.2366751, IEEE Transactions on Computers



(a) (b) (c)

Fig. 6. Performance impact of task count.

that the comprehensive employment of the proposed methods is able to boost the resource utilization effectively. In addition, we can observe that the *RTH*s of FESTAL and NOFESTAL keep a slight ascending trend with the increase of task count, while those of EFF and NMFESTAL keep a decreasing trend. This is reasonable in that EFF and NMFESTAL cannot make full use of the resources, and have to add more active hosts when more tasks are submitted, which as a result incurs more waste of resources. On average, FESTAL outperforms EFF, NMFESTAL and NOFESTAL 79.8%, 63.6% and 11.1%, respectively.

6.1.2 Performance impact of arrival rate

A group of experiments are conducted in this section to observe the impact of arrival rate on the performance. Parameter interval time $1/\lambda$ varies from 0 to 14 with an increment of 2. The experimental results are shown in Fig. 7.

Fig. 7(a) demonstrates that when the interval time $1/\lambda$ varies, FESTAL, EFF, NMFESTAL and NOFESTAL all have relatively high guarantee ratios (above 90%), which also owes to the infinite resources in the cloud. Nonetheless, the guarantee ratio of NOFESTAL is the lowest. This is because the backups cannot overlap other copies and consumes more resources in NOFESTAL, affecting the scheduling of the following tasks.

Fig. 7(b) shows that FESTAL requires smaller *HAT* than EFF, NMFESTAL and NOFESTAL. The reason is similar to that of Fig. 6(b). An interesting phenomenon is that the *HAT* of NOFESTAL in $1/\lambda = 0$ is much larger than those in other situations. This is because tasks surge into the system almost at one time instant, which makes the system have no time to adjust existing resources. Thereby, the overlapping technique, reducing the consumption of backups, is much more effective to make full use of the existing resources to accommodate more tasks, and avoids the large augment of resources.

From Fig. 7(c), it can be found that the *RTH*s of the four algorithms decrease when the $1/\lambda$ ascends. This can be explained that the interval time of the tasks' arriving is prolonged, but the idle time of the VMs does not reach the threshold to cancel them. As a result, the VMs keep idle to wait for the next submitted task, which decreases the resource utilization. When the interval time changes, averagely, FESTAL outperforms EFF, NMFESTAL and NOFESTAL 35.5%, 47.1% and 17.8%, respectively.

6.1.3 Performance impact of deadline

The objective of this experiment is to investigate the impact of task deadlines on the performance of the four algorithms. The parameter $baseDeadline$ varies from 100 to 450 with step 50. Fig. 8 illustrates the experimental results.

In Fig. 8(a), it can be observed that the guarantee ratios of the four algorithms increase sharply when the $baseDeadline$ is prolonged. Compared with the experimental results of task count and arrival time, the impact of deadline on guarantee ratio is much greater. This is explainable in that the tight deadline makes adding new resources meaningless because of the additional boot time. When the deadline is loose enough, almost all the tasks can be scheduled in the four algorithms in virtue of the infinite resources in the cloud.

We can observe from Fig. 8(b) that when $baseTime$ increases, the HATs of the four algorithms first increase, and then keep stable values. This is because when the deadline changes from tight to loose, more tasks can be accepted by the system and thus more hosts keep active to accommodate these tasks. When almost all the tasks are accepted by the system, the HATs keep stable values. Additionally, we can find that the HATs of EFF and NMFESTAL are much higher than those of FESTAL and NOFESTAL, which can be attributed to the similar reasons discussed in Fig. 6(b). It is worth noting that the HAT of NMFESTAL is the largest when $baseDeadline$ =100, 150. We explain this observation by the fact that when the deadline is tight, few tasks can be accepted by the system, and the other algorithms using the VM migration technique can efficiently utilize the existing resource to schedule these small amounts of tasks, which thus avoids starting more hosts.

It is shown in Fig. 8(c) that the *RTH*s of all the algorithms increase when the $baseDeadline$ increases except for that of EFF. When $baseDeadline$=100,150, the RTH of EFF is much higher than those in other situations. The reason of this phenomenon lies in that when the deadline is very tight, few backups can adopt the passive backup scheme, and few backups can be cancelled during the execution. Therefore, the system using EFF can make full use of the resources even without the I-AEAP and I-ALAP. When the deadline becomes looser, some idle resources cannot be eliminated timely by EFF due to the similar reason in Fig. 6(b), and thus the *RTH* decreases. When $baseDeadline$ varies, FESTAL outperforms EFF, NMFESTAL and NOFESTAL 36.0%, 43.3% and 14.8%

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2014.2366751, IEEE Transactions on Computers

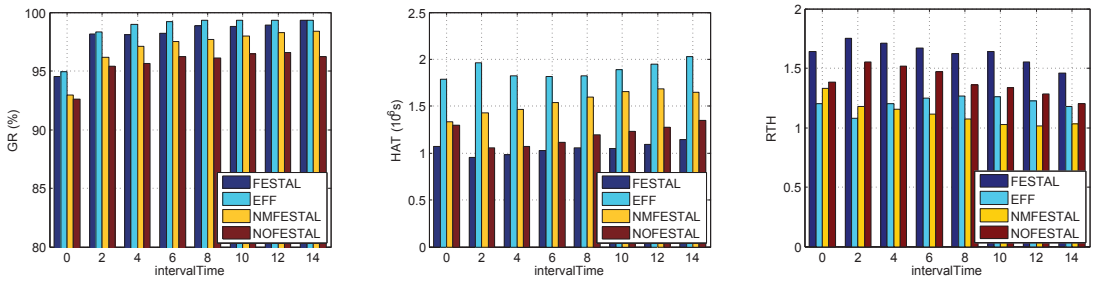


Fig. 7. Performance impact of arrival rate.

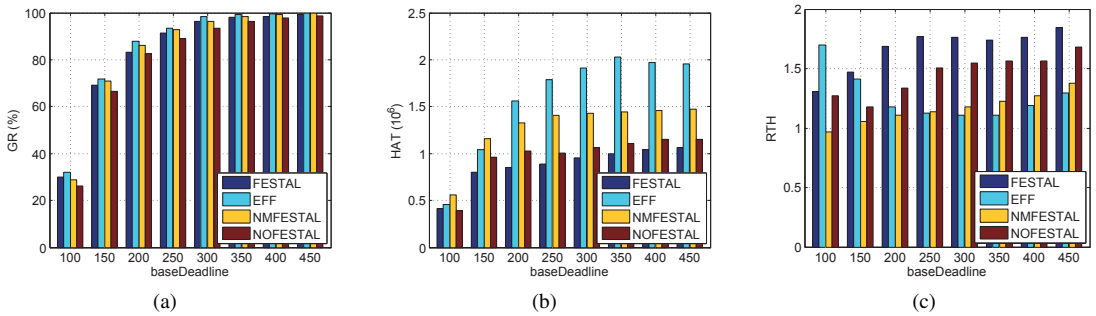


Fig. 8. Performance impact of deadline.

on average, respectively. This result indicates that deadline is an important restrictive factor of the I-AEAP and I-ALAP on which the situation of tight deadlines has a considerable adverse impact.

6.2 Evaluation based on real-world trace

The above groups of experiments demonstrate the performance improvement of FESTAL in various random synthetic workloads. In order to verify the feasibility of our FESTAL in practical use, we conduct the experiments injecting with the latest version of the Google cloud tracelogs [34].

The tracelogs contain the information of 25 million tasks grouped in 650 thousand jobs that span 29 days. It is of considerably high difficulty to conduct an experiment based on all the tasks due to the enormous count of tasks. According to the analysis in [35], day 18 is a representative day among the 29 days. Therefore, the first 5 hours in day 18 were selected as a testing sample in our experiment. About 240 thousand tasks were submitted to the cloud over this 5 hours. The count of tasks submitted in every 30 seconds is depicted in Fig. 9. It is apparent that the task count fluctuates significantly over the time. When large amounts of tasks surge into the system, the resource request is at a peak. While the resource request decreases sharply at the timestamps where few tasks were submitted into the system. Based on this observation, it is straightforward to conclude that the elastic resource provisioning mechanism is essential for a cloud.

Fig. 10 shows the Cumulative Distribution Function (CDF) of the response time (i.e., the time duration form the submission of a task to the finish of it) and the execution time for tasks. We can find that over 50% tasks' execution time is less

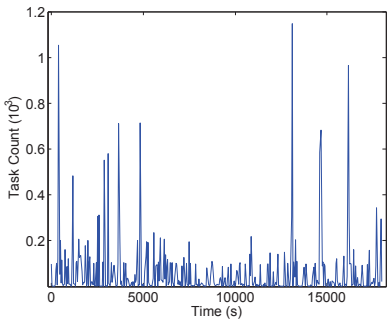


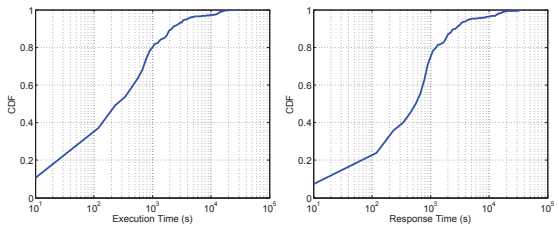
Fig. 9. The count of tasks submitted to the cloud.

than 200s, but below 40% tasks can be responded in 200s. In addition, the average ratio of the tasks' response time over the execution time is 2.89. This observation suggests that most tasks can finish in a relatively short time. But suffering from the scheduling delay and the resource failures, the response time almost triples. As a result, the cloud is in urgent need of efficient scheduling algorithms and fault-tolerant mechanisms for the improvement of its performance.

Because of lacking some detailed information and normalized data, it is necessary to make four realistic assumptions as follows:

- When a task encounters *EVICT* or *KILL* events in the tracelogs, we assume that it is reset back to the initial state. According to the statement in [11], [36], the Google cloud system attempts to restart the tasks that encounter these events from the their initial state.
- The task execution time is calculated from the last

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2014.2366751, IEEE Transactions on Computers



(a) (b)

Fig. 10. CDF of execution time and response time.

SCHEDULE event to the *FINISH* event.

- Task size ts_i is calculated based on the execution time and the average CPU utilization. As the tracelog does not contain the data of task size in MI, we apply the method proposed in [35] to solving the problem.

$$ts_i = (time_{finish} - time_{schedule}) \times U_{avg} \times C_{CPU}, \quad (6)$$

where $time_{finish}$ and $time_{schedule}$ represent the timestamp of *FINISH* and *SCHEDULE* event, respectively; U_{avg} denotes the average CPU usage of this task. All the three values can be obtained in the tracelog. For C_{CPU}, it represents the processing capacity of the CPU in Google cloud. Since the data of machines' capacity is rescaled in the trace, we assume that it is similar to our experiment settings for hosts, $C_{CPU} = 1500$MIPS.

- The deadline of each task is designated through the ratio of response time over execution time. As discussed above, the average ratio is 2.89. So the deadline of each task is assumed to be β times larger than its maximum execution time, where β is uniformly distributed in the range of [2.6,3.2].

The experimental results shown in Table 2 specify that FESTAL can exhibit a good performance in practical. By virtue of the nearly infinite resources in the cloud, the *GR*s of the four algorithms are all relatively high. Nonetheless, there are still some tasks rejected due to the restriction of deadline in our experiment. The *HAT* of FESTAL is the smallest, while that of EFF is the largest. This result once again suggests that with the comprehensive employment of the techniques and the mechanisms proposed in our work, FESTAL can effectively reduce the active time of hosts. Meanwhile, without the help of I-ALAP when scheduling backups, EFF scatter the backups on all the VMs of active hosts, which makes few host be switched off when many backups are cancelled even with the employment of the VM migration technique, and thereby raises the *HAT*. It is worthwhile noting that NOFESTAL performs the worst in terms of *RTH*, even much worse than EFF. This observation indicates that the overlapping technique is a powerful method for improving the resource utilization in practical. From Fig. 9, it can be found that differing from the synthetic workloads, the real-world workload has many situations where a surge of tasks are submitted to the system almost at the same time. In these situations, the system is so over-burdened that adjusting the existing resources is almost impossible. Thereby, the overlapping technique is much more effective to make full use of the existing resources to accommodate more tasks, and avoids the large augment of resources, which obviously decreases the resource utilization. In addition, the *RTH* of NMFESTAL is a little higher than that of EFF, which means that thanks to the aid of I-AEAP and I-ALAP, NMFESTAL can exhibit better performance than EFF even without the VM migration technique. This result argues that I-AEAP and I-ALAP have the ability to reduce the need of VM migration for improving resource utilization in practice. They are able to abate the overhead brought by VM migration without the sacrifice of resource utilization. Based on the Google cloud tracelogs, in terms of *RTH*, FESTAL outperforms EFF, NMFESTAL and NOFESTAL 13.2%, 11.4% and 20.5%, respectively.

TABLE 2
Performance on Google cloud workloads

Metr. \ Alg.	FESTAL	EFF	NMFESTAL	NOFESTAL
GR	95.08%	95.03%	95.87%	93.04%
HAT($\times 10^6 s$)	5.30	6.58	6.26	6.44
RTH	3.70	3.27	3.32	3.07

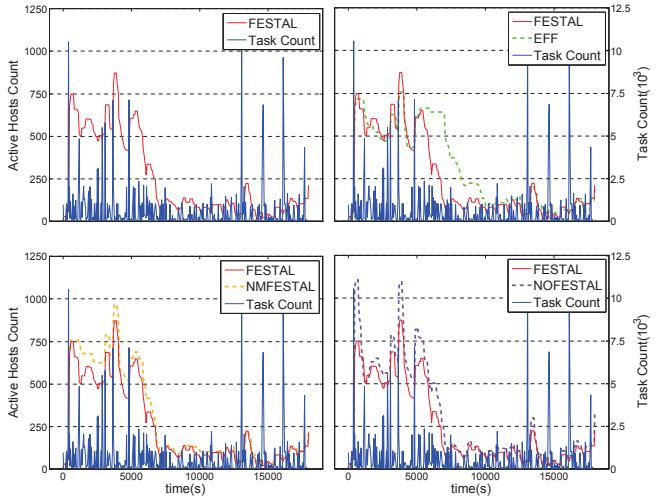


Fig. 11. The change of *AHC* and Task Count over time.

In order to further investigate how our proposed techniques and mechanisms work, we depict the change of active hosts' count (*AHC* in short) over time in Fig. 11, and compare FESTAL with the other three algorithms respectively. The blue solid line represents the task count whose value is specified by the right side Y-axis. The red solid line and dashed lines represent the *AHC*s of FESTAL and the other algorithms whose values are specified by the left side Y-axis. Obviously, the *AHC* of FESTAL changes with the task count in the first figure, which indicates that the resource provisioning in the cloud system using FESTAL is elastic according to the resource request. When large amounts of tasks surge into the system, the *AHC* increases to accommodate these tasks. When the task count is at a low ebb, and the resource is over-provisioned, the *AHC* decreases to improve the resource uti-

lization. This result suggests that our proposed elastic resource provisioning mechanism works well in practical context. From the comparison of FESTAL and EFF, we can observe that the system using FESTAL turns on less active hosts than EFF especially when the workload changes from heavy to light. Take the time span from 5,000s to 10,000s for example, the tasks submitted to system are much fewer than those in the previous time, the *AHC* of FESTAL decreases sharply while that of NMEARH still keeps a relatively high value. The superiority of FESTAL in this situation can be attributed to the I-AEAP and I-ALAP, which enables idle VMs and active hosts to be switched off timely when many backups are cancelled. The comparison of FESTAL and NMFESTAL from 1,000s to 3,000s indicates that the introduction of the VM migration technique is also able to help the system eliminate idle resources timely by consolidating VMs to several active hosts. However, during the time span from 5,000s to 10,000s, the *AHC* of NMFESTAL shows the similar decreasing trend with that of FESTAL. This phenomenon once again specifies that the system using I-AEAP and I-ALAP can switch off the idle hosts timely when the resource request drops sharply even without the VM migration technique. Comparing the *AHC*s of FESTAL and NOFESTAL, we can find that the overlapping technique can reduce the demand of active hosts when large amounts of tasks surge into the system. This is because when the resource provisioning is inadequate, the adoption of the overlapping technique reduces the resource consumption of backups effectively, and leaves more available resources for other tasks. As a result, the system avoids turning on much more hosts to adapt to the quickly increased request.

7 Conclusions and Future Work

We presented in this paper an efficient fault-tolerant elastic scheduling algorithm FESTAL. FESTAL is based on a novel fault-tolerant mechanism that extends from the conventional PB model by accommodating the virtualization technology and the VM migration technology which are employed in most cloud date centers. FESTAL is the first of its kind reported in the literatures; it comprehensively addresses the issue of reliability, elasticity and schedulability of virtualized clouds. By the comprehensive employment of the elastic resource provisioning mechanism and the fault-tolerant scheduling algorithms, FESTAL is able to achieve both fault tolerance and high performance in terms of resource utilization. Extensive experiments based on the synthetic workloads and the real-world traces invalidate that FESTAL can enhance the performance of virtualized clouds effectively.

The following issue will be addressed in our future works: First, we will extend our fault-tolerant mechanism to tolerate multiple hosts' failure and propose corresponding scheduling algorithms. Second, we will take the communication time and other characteristics of hosts into consideration to enhance the accuracy of the model. Third, we plan to implement the FESTAL in a real cloud system to further test its performance.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and I. Stoica, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[2] B. Nicolae and F. Cappello, "Blobcr: virtual disk based checkpoint-restart for hpc applications on iaas clouds," *Journal of Parallel and Distributed Computing*, vol. 73, no. 5, pp. 698–711, 2013.

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

[4] S. Ghosh, R. Melhem, and D. Mossé, "Fault-tolerance through scheduling of aperiodic tasks in hard real-time multiprocessor systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 3, pp. 272–284, 1997.

[5] G. Manimaran and C. S. R. Murthy, "A fault-tolerant dynamic scheduling algorithm for multiprocessor real-time systems and its analysis," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 11, pp. 1137–1152, 1998.

[6] R. Al-Omari, A. K. Somani, and G. Manimaran, "Efficient overloading techniques for primary-backup scheduling in real-time systems," *Journal of Parallel and Distributed Computing*, vol. 64, no. 5, pp. 629–648, 2004.

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

[8] X. Zhu, X. Qin, and M. Qiu, "Qos-aware fault-tolerant scheduling for real-time tasks on heterogeneous clusters," *IEEE Transactions on Computers*, vol. 60, no. 6, pp. 800–812, 2011.

[9] J. Balasangameshwara and N. Raju, "Performance-driven load balancing with a primary-backup approach for computational grids with low communication cost and replication cost," *IEEE Transactions on Computers*, vol. 62, no. 5, pp. 990–1003, 2013.

[10] K. Plankensteiner, R. Prodan, T. Fahringer, A. Kertesz, and P. Kacsuk, "Fault-tolerant behavior in state-of-the-art grid worklow management systems," Inst. On Grid Information, Resource and Worklow Monitoring Services, CoreGRIDłNetwork of Excellence, Technical Report, 2007.

[11] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[12] K. Plankensteiner and R. Prodan, "Meeting soft deadlines in scientific workflows using resubmission impact," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 5, pp. 890–901, 2012.

[13] R. Al-Omari, A. K. Somani, and G. Manimaran, "An adaptive scheme for fault-tolerant scheduling of soft real-time tasks in multiprocessor systems," *Journal of Parallel and Distributed Computing*, vol. 65, no. 5, pp. 595–608, 2005.

[14] S. Antony, S. Antony, A. Beegom, and R. M S, "Task scheduling algorithm with fault tolerance for cloud," in *IEEE International Conference on Computing Sciences*. IEEE, 2012, pp. 180–182.

[15] P. Samimi, Y. Teimouri, and M. Mukhtar, "A combinatorial double auction resource allocation model in cloud computing," *Information Sciences*, p. http://dx.doi.org/10.1016/j.ins.2014.02.008, 2014.

[16] D. Warneke and O. Kao, "Exploiting dynamic resource allocation for efficient parallel data processing in the cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 6, pp. 985–997, 2011.

[17] C.-H. Hsu, K. D. Slagter, S.-C. Chen, and Y.-C. Chung, "Optimizing energy consumption with task consolidation in clouds," *Information Sciences*, vol. 60, no. 6, pp. 452–462, 2014.

[18] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh, "A cost-aware elasticity provisioning system for the cloud," in *IEEE International Conference on Distributed Computing Systems*. IEEE, 2011, pp. 559–570.

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

[20] N. Vasic, D. Novakovic, S. Miucin, D. Kostic, and R. Bianchini, "Dejavu: accelerating resource allocation in virtualized environments," *ACM SIGARCH Computer Architecture News*, vol. 40, no. 1, pp. 423–436, 2012.

[21] P. Graubner, M. Schmidt, and B. Freisleben, "Energy-efficient management of virtual machines in eucalyptus," in *IEEE International Conference on Cloud Computing*. IEEE, 2011, pp. 243–250.

[22] G. Le, K. Xu, and J. Song, "Dynamic resource provisioning and scheduling with deadline constraint in elastic cloud," in *IEEE International Conference on Service Science*. IEEE, 2013, pp. 113–117.

[23] Q. Zhang, M. F. Zhani, R. Boutaba, and J. L. Hellerstein, "Harmony: dynamic heteroheneity-aware resource provisioning in the cloud," in

IEEE International Conference on Distributed Computing Systems. IEEE, 2013, pp. 511–519.

[24] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[25] B. Xu, C. Zhao, E. Hu, and B. Hu, "Job scheduling algorithm based on berger model in cloud environment," *Advances in Engineering Software*, vol. 42, no. 7, pp. 419–425, 2011.

[26] L. Wu, S. Kumar Garg, and R. Buyya, "Sla-based admission control for a software-as-a-service provider in cloud computing environments," *Journal of Computer and System Sciences*, vol. 78, no. 5, pp. 1280–1299, 2012.

[27] S. Sadhasivam, N. Nagaveni, R. Jayarani, and R. V. Ram, "Design and implementation of an efficient two-level scheduler for cloud computing environment," in *IEEE International Conference on Advances in Recent Technologies in Communication and Computing*. IEEE, 2009, pp. 884–886.

[28] M.-Y. Tsai, P.-F. Chiang, Y.-J. Chang, and W.-J. Wang, *Heuristic scheduling strategies for linear-dependent and independent jobs on heterogeneous grids*. Springer, 2011.

[29] T. D. Braun, H. J. Siegel, N. Beck, L. L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, and D. Hensgen, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed computing*, vol. 61, no. 6, pp. 810–837, 2001.

[30] Y. Ma, B. Gong, R. Sugihara, and R. Gupta, "Energy-efficient deadline scheduling for heterogeneous systems," *Journal of Parallel and Distributed Computing*, vol. 72, no. 12, pp. 1725–1740, 2012.

[31] W.-J. Wang, Y.-S. Chang, W.-T. Lo, and Y.-K. Lee, "Adaptive scheduling for parallel tasks with qos satisfaction for hybrid cloud environments," *The Journal of Supercomputing*, vol. 66, no. 2, pp. 783–811, 2013.

[32] W. Luo, X. Qin, X.-C. Tan, K. Qin, and A. Manzanares, "Exploiting redundancies to enhance schedulability in fault-tolerant and real-time distributed systems," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 39, no. 3, pp. 626–639, 2009.

[33] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall, "Entropy: a consolidation manager for clusters," in *ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*. ACM, 2009, Conference Proceedings, pp. 41–50.

[34] "Google cluster data v2," *https://code.google.com/p/googleclusterdata/wiki*.

[35] I. S. Moreno, P. Garraghan, P. Townend, and J. Xu, "An approach for characterizing workloads in google cloud to derive realistic resource utilization models," in *Service Oriented System Engineering*. IEEE, 2013, pp. 49–60.

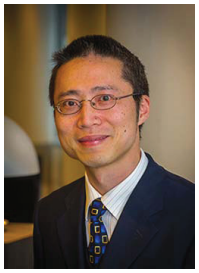
[36] C. Reiss, J. Wilkes, and J. Hellerstein, "Google cluster-usage traces: format + schema," Google Inc., White Paper, 2011.

Ji Wang received the B.S. degree in information systems from National University of Defense Technology, China, in 2008. Currently, he is a M.S. student in the School of Information System and Management at National University of Defense Technology. His research interests include real-time systems, fault-tolerance, and cloud computing.

Weidong Bao received the Ph.D. degree in information system from the National University of Defense Technology in 1999. He is currently a professor in the College of Information Systems and Management at National University of Defense Technology, Changsha, China. His recent research interests include cloud computing, information system, and complex network.

Xiaomin Zhu received the Ph.D. degree in computer science from Fudan University, Shanghai, China, in 2009. He is currently an associate professor in the College of Information Systems and Management at National University of Defense Technology, Changsha, China. His research interests include scheduling and resource management in green computing, cluster computing, and cloud computing. He has published more than 50 research articles in refereed journals and conference proceedings, such as IEEE TC, IEEE TPDS, and JPDC. He is a member of the IEEE, the IEEE Communication Society, and the ACM.

Laurence T. Yang research fields include networking, high performance computing, embedded systems, ubiquitous computing and intelligence. He has published around 300 papers in refereed journals, conference proceedings and book chapters in these areas. He has been involved in more than 100 conferences and workshops as a program/general/steering conference chair and more than 300 conference and workshops as a program committee member. Currently is the chair of IEEE Technical Committee of Scalable Computing (TCSC), the chair of IEEE Task force on Ubiquitous Computing and Intelligence, the co-chair of IEEE Task force on Autonomic and Trusted Computing. He is also in the executive committee of IEEE Technical Committee of Self-Organization and Cybernetics for Informatics, and of IFIP Working Group 10.2 on Embedded Systems.



Yang Xiang received his PhD in Computer Science from Deakin University, Australia. He is currently a full professor at School of Information Technology, Deakin University. He is the Director of the Network Security and Computing Lab (NSCLab). His research interests include network and system security, distributed systems, and networking. In particular, he is currently leading his team developing active defense systems against large-scale distributed network attacks. He has published more than 150 research papers in many international journals and conferences, such as IEEE TC, IEEE TPDS, and IEEE TISF. Two of his papers were selected as the featured articles in the April 2009 and the July 2013 issues of IEEE TPDS. He has served as the Program/General Chair for many international conferences such as ICA3PP 12/11, IEEE/IFIP EUC 11, IEEE TrustCom 13/11, IEEE HPCC 10/09, IEEE ICPADS 08, NSS 11/10/09/08/07. He has been the PC member for more than 60 international conferences in distributed systems, networking, and security. He serves as the Associate Editor of IEEE TC, IEEE TPDS, Security and Communication Networks (Wiley), and the Editor of Journal of Network and Computer Applications. He is the Coordinator, Asia for IEEE Computer Society Technical Committee on Distributed Processing (TCDP). He is a Senior Member of the IEEE.