2013 IEEE International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing

Real-Time Fault-Tolerant Scheduling Based on Primary-Backup Approach in Virtualized Clouds

Ji Wang, Xiaomin Zhu, and Weidong Bao
Science and Technology on Information Systems Engineering Laboratory
National University of Defense Technology
Changsha, China 410073
Email: {wangji, xmzhu, wdbao}@nudt.edu.cn

***Abstract*—Cloud computing represents a revolutionary paradigm for the great provisioning of computing resources. However, the enormous scale of Clouds increases the probability of failures. So fault-tolerance becomes a significant requirement, especially for real-time applications due to their safety-critical nature. Fault-tolerant scheduling as an efficient fault-tolerant technique attracts extensive studies. Unfortunately, existing fault-tolerant scheduling algorithms, based on the primary-backup approach, do not take virtualization, one of the key features of Clouds into account. To address this issue, we propose a fault-tolerant scheduling algorithm for virtualized Clouds named FSVC using primary-backup approach to tolerate physical host failures. FSVC strives to enhance the performance by employing comprehensive overlapping techniques and Virtual Machine (VM) migration technique. The constraints of the two techniques are elaborately analyzed to realize fault-tolerance. Besides, a two-phase policy is incorporated in FSVC to further improve the performance. Extensive simulation experiments demonstrate that FSVC can improve the schedulability and resource utilization effectively in virtualized Clouds.**

I. Introduction

Cloud computing can be classified as a new paradigm for the dynamic provisioning of computing services supported by data centers that usually employ virtualization technologies for consolidation and environment isolation [1]. Noticeably, real-time applications have been deployed in Clouds, in which the correctness of system depends not only on the results of computation, but also on the time instants at which these results become available [2].

Due to the safety-critical nature of real-time systems, it is essential to preserve the ability to deliver expected services despite the occurrence of failures. So fault-tolerance in Clouds becomes a crucial issue. One efficient fault-tolerant technique is fault-tolerant scheduling. Many researches have been conducted on fault-tolerant scheduling algorithms for traditional distributed systems, among which the primary-backup (PB) approach plays an important role. In this approach, each task has two copies, namely, the primary and the backup, allocated to two different computational instances [3], [4]. Although the PB approach offers fault-tolerance, it is accompanied by larger consumption of computational resources. Hence, to improve the resource utilization, three variants of the primary-backup approach appear: 1) the active-backup scheme [5], [6]; 2) the passive-backup scheme [2], [7]; 3) the backup-backup, primary-backup overlapping techniques [8], [9].

In the active-backup scheme, the backup executes concurrently with the primary, which is easy to deploy [10], while it can cause a large resource consumption. The passive-backup is scheduled to execute only if the primary fails, and can be deallocated if the primary completes successfully. However, the passive-backup requires that the task has enough laxity to restart the backup, which may be unrealistic in practice. To take the advantages of the two schemes, an adaptive backup scheme that controls the status forms of backups based on tasks' laxity was investigated in [8], [10], [11]. Overlapping technique is an efficient method to improve the schedulability. Backup-backup overlapping allows backups whose corresponding primaries are assigned to different processors to overlap with each other [3]. Al-Omari *et al.* proposed a primary-backup overlapping technique where a primary can overlap with a backup of another task [8]. To achieve better quality of scheduling, most literatures integrate above variants into one approach [11], [12]. But there is little research on collectively considering all variants of the PB approach to further enhance the performance. Moreover, the literatures have not studied the conditions that must be met of using PB approach when the features of Clouds are considered.

Regarding the allocation of primaries and backups, fault-tolerant scheduling algorithms along with PB approach have been designed for traditional systems. In [13], a static scheduling algorithm that can tolerate one processor's failure was studied, yet it was designed only for homogeneous systems. Qin and Jiang developed a fault-tolerant scheduling algorithm in heterogeneous systems for dependent tasks [2]. Nonetheless the algorithm is static in nature not suitable for dynamic scheduling. Zheng *et al.* proposed two algorithms, called MRC-ECT and MCT-LRC, to schedule backups of independent tasks and dependent tasks for the purpose of minimum replication cost and minimum completion time with less replication cost, respectively [9]. However, only passive-backups are supported in these algorithms. Zhu *et al.* considered both active-backup and passive-backup schemes, and applied the ASAP strategy to scheduling primaries while applying the ALAP strategy to backups [11]. Although these algorithms exhibit good performance in traditional systems, they have two common drawbacks: 1) they only support part of the three primary-backup variants discussed above, and are not integrated with all of them seamlessly; 2) they are designed

978-0-7695-5088-6/13 $31.00 © 2013 IEEE
DOI 10.1109/HPCC.and.EUC.2013.159

for traditional systems without taking virtualization, a key enabling technology of Cloud computing, into account.

Virtualization technology [14] partition a single host into several parts, while the VM migration enables the fine-grained resource provisions in multiple-host environments. However, this technology makes the research of fault-tolerance in Clouds more challenging. Firstly, virtualization technology splits Clouds into three layers: hosts, VMs and tasks. Tasks are allocated to VMs rather than hosts directly. A host's failure leads to multiple computational instance failures, which raises the difficulty of the allocation of tasks. Secondly, to improve the resource utilization, the VM migration should be contained in our algorithms. But additional constraints for VM migration must be studied to satisfy fault-tolerance.

Our work aims at designing a fault-tolerant scheduling algorithm which endeavors to tackle the above problems.

II. System Model and Definitions

A. Fault Model

The assumptions of the fault model are similar to [2].

- At one time instant, at most one host can be failed. Tasks whose primaries are on the failed host can be finished by their backups before another host fails.
- Faults are transient or permanent and are independent.
- There exists a failure detection mechanism.

B. Task Model

We use a set $T = \{t_1, t_2, ..., t_n\}$ of tasks that are assumed to be non-preemptive, and aperiodic. Each task t_i has three attributes: arrival time a_i, deadline d_i and computational size cs_i which is measured by Millions of Instructions (MI). Each task t_i has two copies denoted as t_i^P and t_i^B executed on two different hosts for fault-tolerance. Given a task $t_i \in T$, s_i^P and f_i^P represent the start time and finish time of t_i's primary, whereas s_i^B and f_i^B denote those of t_i's backup.

The virtualized Cloud is composed of 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), representing the infinite heterogeneous computational resource. It is worth noting that the computational resources in Clouds are infinite, but the count of active hosts is actually limited. A set $H_a = \{h_1, h_2, ..., h_{|H_a|}\}$ denotes the active hosts in H while the hosts in $H - H_a$ are turned-off hosts. If the hosts in H_a are not sufficient for workload, hosts in $H - H_a$ are turned on and added into H_a. Each host h_i can accommodate multiple VMs denoted by a set $V_i = \{v_{i1}, v_{i2}, ..., v_{i|V_i|}\}$ of VMs with different processing power P_{ij}. For the processing power of VMs on host h_i, it meets the constraint that $\sum_{j=1}^{|V_i|} P_{ij} \leq P_i$.

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. On the VM v_{kl}, the execution time $e_{kl}(t_i)$ of task t_i's copy is the ratio of its computational size over processing power of this VM.

In our work, the active-backup and passive-backup schemes are both adopted. We assign a task's primary before assigning its backup. The status of backup allocated to v_{kl} is denoted by $st(t_i^B)$. If $f_i^P + e_{kl}(t_i) \leq d_i$, $st(t_i^B)$ is passive, otherwise, $st(t_i^B)$ is active.

C. Definitions

Now we introduce four terminologies to facilitate the analysis in following sections.

Definition 1. *PB overlapping set* $\mathit{\Gamma}_{\{\cdot\}}$ is a set of tasks that are related due to primary-backup overlapping, where the subscript $\{\cdot\}$ represents the tasks in the set $\mathit{\Gamma}_{\{\cdot\}}$. A task t_j becomes an element of a set $\mathit{\Gamma}_{\{\cdot\}}$ on condition that t_j^P overlaps with t_i^B, or t_j^B overlaps with t_i^P, $\forall t_i \in \mathit{\Gamma}_{\{\cdot\}}$.

Definition 2. *Extended PB overlapping set* $\mathit{\Gamma}^e_{\{\cdot\}}$ is an extended set of PB overlapping set $\mathit{\Gamma}_{\{\cdot\}}$. For a PB overlapping set $\mathit{\Gamma}_{\{\cdot\}}$, it is extended to $\mathit{\Gamma}^e_{\{\cdot\}}$ by adding a tasks t_j when t_j^B overlaps with t_i^B, $\forall t_i \in \mathit{\Gamma}_{\{\cdot\}}$.

Definition 3. *PB host set* $HS(\mathit{\Gamma}_{\{\cdot\}})$ refers to a set of hosts where the primaries of tasks in $\mathit{\Gamma}_{\{\cdot\}}$ are allocated, $HS(\mathit{\Gamma}_{\{\cdot\}}) = \{host(t_i^P) \mid \forall t_i \in \mathit{\Gamma}_{\{\cdot\}}\}$.

Definition 4. *Extended PB host set* $eHS(\mathit{\Gamma}_{\{\cdot\}})$ denotes a set of hosts where the primaries and backups of tasks in $\mathit{\Gamma}_{\{\cdot\}}$ are allocated, $eHS(\mathit{\Gamma}_{\{\cdot\}}) = HS(\mathit{\Gamma}_{\{\cdot\}}) \cup \{host(t_i^B) \mid \forall t_i \in \mathit{\Gamma}_{\{\cdot\}}\}$.

III. Constraints of Scheduling in Clouds

A. Comprehensive Overlapping

In virtualized Clouds, especially when PB overlapping and active-backups are considered, the overlapping of copies are nontrivial and constraints need to be identified. Theorem 1 demonstrates the basic requirement of overlapping.

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

Proof. We prove by contradiction. Suppose that $host(t_i^P) = host(t_j^P) = h_1$, and t_i^B overlaps with t_j^B. When h_1 fails, all the VMs in h_1 fail, resulting in the failures of the copies allocated to these VMs, including t_i^P and t_j^P. Thus, t_i^B and t_j^B must execute. A timing conflict happens. Contradiction happens. Therefore, t_i^B cannot overlap with t_j^B.

Besides Theorem 1, other constraints are analyzed below. Without loss of generality, assume that two copies of task t_i have been scheduled, and t_j is a newly-arrived task.

1) How a primary overlaps with other backups

To begin with, we analyze how a primary overlaps with other backups. In terms of the arrival time of t_j, two cases should be discussed respectively.

a) $a_j \leq s_i^B$

This case is shown in Fig. 1 where the arrival time of t_j is earlier than or equal to the start time of t_i^B. In Fig. 1(a), t_i^B is a passive-backup while t_i^B is an active-backup in Fig. 1(b). However, in both two situations, the overlapping is infeasible.

Theorem 2. If the start time of t_j^P is earlier than or equal to s_i^B and $vm(t_j^P) = vm(t_i^B)$, then t_j^P cannot overlap with t_i^B, regardless of the status of t_i^B.

Proof. Prove by contradiction. Suppose that t_j^P overlaps with t_i^B. When $st(t_i^B) = passive$, if $host(t_i^P)$ fails, then t_i^B is due to execute. But non-preemptive t_j^P is already executing. t_i^B cannot execute, thus task t_i fails to complete. Contradiction

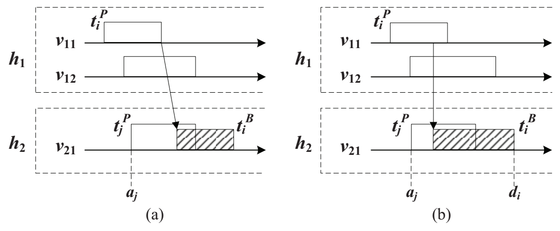


Fig. 1. Scenarios of case $a_j \leq s_i^B$.

happens. When $st(t_i^B) = active$, t_i^B is due to execute no matter $host(t_i^P)$ fails or not. The same contradiction happens. Therefore, t_j^P cannot overlap with t_i^B.

b) $a_j > s_i^B$

This case is shown in Fig. 2. Fig. 2(a) depicts that $st(t_i^B) = passive$. When s_j^P is later than s_i^B, the overlapping between t_j^P and t_i^B is feasible. Let $EST_{kl}(t_j^P)$ denote the earliest start time of t_j^P on v_{kl}. Combined with the analysis in Theorem 2, we can derive the following proposition.

Proposition 1. On the VM $vm(t_i^B) = v_{kl}$, if t_j^P overlaps with t_i^B and $st(t_i^B) = passive$, then

$$EST_{kl}(t_j^P) = \max\{a_j, s_i^B\}. \tag{1}$$

Then, the situation changes to $st(t_i^B) = active$ shown in Fig. 2(b). As the execution of t_i^B is terminated if t_i^P finishes successfully, t_j^P can overlap with t_i^B so long as it starts later than f_i^P. Hence, we can derive the proposition:

Proposition 2. On the VM $vm(t_i^B) = v_{kl}$, if t_j^P overlaps with t_i^B and $st(t_i^B) = active$, then

$$EST_{kl}(t_j^P) = \max\{a_j, f_i^P\}. \tag{2}$$

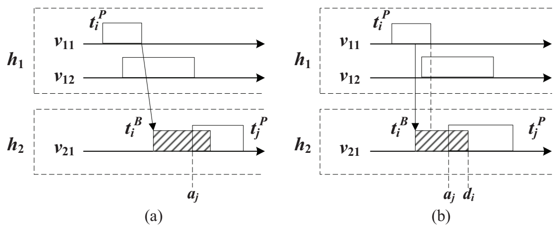


Fig. 2. Scenarios of case $a_j > s_i^B$.

2) How a backup overlaps with other copies

As mentioned above, we assign a task's primary before assigning its backup. It has a great impact on the backup's overlapping whether its corresponding primary overlaps with other copies. So we analyze the problem in two cases: a) t_j^P does not overlap with others; b) t_j^P overlaps with others.

a) t_j^P does not overlap with others

In this case, we first analyze the situation that t_j^B overlaps with a scheduled primary t_i^P. Let $LST_{kl}(t_j^B)$ denote the latest start time of t_j^B on v_{kl}. In Theorem 2, it is pointed out that an overlapped backup cannot start later than the primary that overlaps with it. So we can derive a proposition easily.

Proposition 3. On the VM $vm(t_i^P) = v_{kl}$, if t_j^B overlaps with t_i^P then

$$LST_{kl}(t_j^B) = \min\{d_j - e_{kl}(t_j), s_i^P\}. \tag{3}$$

The proof is similar to that of Theorem 2. If t_j^B starts later than t_i^P, it may result in a timing conflict.

Proposition 4. If $t_i \in \Gamma_{\{i,\cdots\}}$ and $host(t_j^P) \in eHS(\Gamma_{\{i,\cdots\}})$, then t_j^B cannot overlap with t_i^P.

Take an example shown in Fig. 3. Before t_j is scheduled, there is a PB overlapping set $\Gamma_{\{i,x\}} = \{t_i, t_x\}$, and $eHS(\Gamma_{\{i,x\}}) = \{h_1, h_2, h_3\}$. Suppose that $host(t_j^P) = h_3 \in eHS(\Gamma_{\{i,x\}})$, and t_j^B overlaps with t_i^P. If h_3 fails, t_j^B executes, thus t_i^P cannot start. t_x^B is due to execute. But t_x^B fails because of the failure of h_3. As a result, t_x cannot be completed. Therefore, t_j^B cannot overlap with t_i^P.

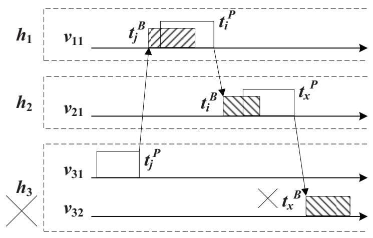


Fig. 3. An illustration example of Proposition 4.

Now, we study the situation that the backup t_j^B overlaps with a scheduled backup t_i^B.

Theorem 3. On the VM $vm(t_i^B) = v_{kl}$, $st(t_j^B) = active$ or $st(t_j^B) = passive$, if t_j^B overlaps with active-backup t_i^B, then

$$EST_{kl}(t_j^B) = f_i^P. \tag{4}$$

Proof. Prove it in two steps. Suppose that $s_j^B \leq s_i^B$. When $host(t_j^P)$ fails, t_j^B executes, but t_i^B is an active-backup, it will also execute, which leads to a timing conflict. Next it comes to $s_i^B < s_j^B < f_i^P$. As t_i^B is an active-backup and t_i^P has not finished, the execution of t_i^B cannot be terminated. Starting t_j^B during this time slot results in a timing conflict. Hence, $EST_{kl}(t_j^B) = f_i^P$.

Proposition 5. If $host(t_j^P) \in HS(\Gamma_{\{\cdot\}})$, then t_j^B cannot overlap with t_i^B, $\forall t_i \in \Gamma_{\{\cdot\}}$, regardless of the status of t_i^B.

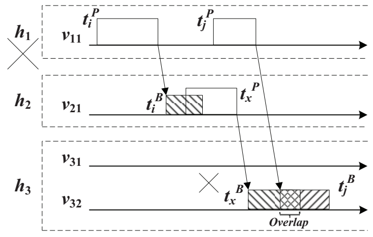


Fig. 4. An illustration example of Proposition 5

An example is shown in Fig. 4. There is a PB overlapping set $\Gamma_{\{i,x\}} = \{t_i, t_x\}$, and $HS(\Gamma_{\{i,x\}}) = \{h_1, h_2\}$. Suppose that $host(t_j^P) = h_1 \in HS(\Gamma_{\{i,x\}})$, and t_j^B overlaps with t_x^B. If h_1 fails, t_i^B executes, thus t_x^P cannot start, and so t_x^B executes. But t_j^B is also due to execute because of the failure of h_1. A timing conflict happens. Therefore, t_j^B cannot overlap with t_x^B.

b) t_j^P overlaps with others

The introduction of primary-backup overlapping brings additional constraints besides those analyzed above. When t_j^P overlaps with others, t_j will be an element of a PB overlapping set. Without loss of generality, we suppose $t_j \in \Gamma_{\{i,j,\cdots\}}$. How to schedule t_j^B is analyzed as follows.

Proposition 6. If $t_j \in \Gamma_{\{i,j,\cdots\}}$, then the backup of t_j cannot be scheduled on the host that belongs to $HS(\Gamma_{\{i,j,\cdots\}})$.

Take an example shown in Fig. 5. As t_j^P overlaps with t_i^B, thus $t_j \in \Gamma_{\{i,j\}}$, $HS(\Gamma_{\{i,j\}}) = \{h_1, h_2\}$. Suppose that $host(t_j^B) = h_1 \in HS(\Gamma_{\{i,j\}})$. When h_1 fails, t_i^B is due to execute, thus t_j^P fails to start, and t_j^B is invoked. But t_j^B also fails because of the failure of h_1. As a result, t_j cannot be completed. Therefore, $host(t_j^B)$ cannot belong to $HS(\Gamma_{\{i,j\}})$.

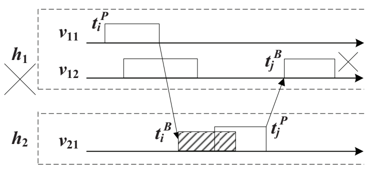


Fig. 5. An illustration example of Proposition 6.

Proposition 7. If $t_j \in \Gamma_{\{i,j,\cdots\}}$, $t_x \in \Gamma_{\{x,\cdots\}}$, and $HS(\Gamma_{\{i,j,\cdots\}}) \cap HS(\Gamma_{\{x,\cdots\}}) \neq \emptyset$, then the backup of t_j cannot overlap with t_x^B.

The illustration example is shown in Fig. 6. There are two PB overlapping sets $\Gamma_{\{i,j\}}$ and $\Gamma_{\{x,y\}}$. $HS(\Gamma_{\{i,j\}}) \cap HS(\Gamma_{\{x,y\}}) = h_1$; $t_j \in \Gamma_{\{i,j\}}$, $t_y \in \Gamma_{\{x,y\}}$. The overlapping between t_j^B and t_y^B is infeasible. When h_1 fails, both t_i^B and t_x^B are invoked, and then t_j^P and t_y^P fail to start, which leads to the execution of t_j^B and t_y^B. But a timing conflict happens. As a result, t_j^B cannot overlap with t_y^B.

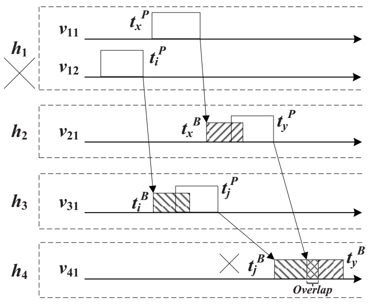


Fig. 6. An illustration example of Proposition 7

Till now, all the possible situations of scheduling primaries and backups to overlap with scheduled primaries, active-backups and passive-backups have been analyzed. Multiple copies' overlapping is feasible so long as the overlapping of each pair among the copies meets the requirements of the constraints discussed above.

B. VM Migration

The overlapping technique enhances the resource utilization at VM level, and further improvement can be achieved at host level. VM migration is just the one that can make further improvement at host level.

Proposition 8. Let fH_{kl} denote a set of hosts which v_{kl} cannot migrate to. $fH_{kl} = \{host(t_i^P) | \forall t_i \in T, vm(t_i^B) = v_{kl}\} \cup \{host(t_i^B) | \forall t_i \in T, vm(t_i^P) = v_{kl}\}$. v_{kl} cannot migrate to $h_j, \forall h_j \in fH_{kl}$.

This constraint attempts to ensure that a task's primary and backup are not scheduled in the VMs that are in the same host. Considering the constraints of backup-backup overlapping, we can get the following proposition.

Proposition 9. $\forall t_i \in T$, $vm(t_i^P) = v_{kl}$, if t_i^B overlaps with t_j^B, then v_{kl} cannot migrate to $host(t_j^P)$.

This constraint is derived from Theorem 1. The migration of v_{kl} violates Theorem 1, which causes two overlapped backups' primaries to be scheduled in the same host.

Then, it comes to a more complex situation in terms of primary-backup overlapping.

Proposition 10. $\forall t_i \in T$, $vm(t_i^P) = v_{kl}$ or $vm(t_i^B) = v_{kl}$, if $t_i \in \Gamma^e_{\{i,\cdots\}}$, then v_{kl} cannot migrate to h_i, $\forall h_i \in eHS(\Gamma^e_{\{i,\cdots\}})$.

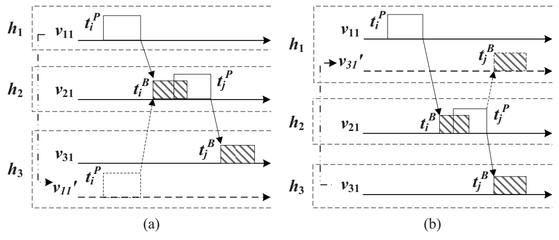


Fig. 7. Examples of $t_i \in \Gamma_{\{i,j\}}$ in Proposition 10

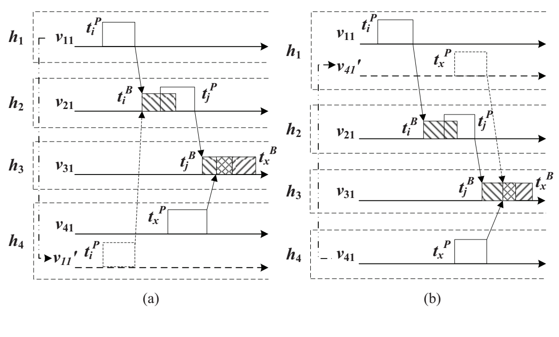


Fig. 8. Examples of $t_x \in \Gamma^e_{\{i,j\}}$ in Proposition 10

We illustrate the proposition by two steps. First, $\forall t_i \in T$, $vm(t_i^P) = v_{kl}$ or $vm(t_i^B) = v_{kl}$, if $t_i \in \Gamma_{\{i,\cdots\}}$, then v_{kl} must not migrate to h_i, $\forall h_i \in eHS(\Gamma_{\{i,\cdots\}})$. This constraint ensures that the VM migration does not result in a violation of Proposition 6. Fig. 7 it two examples of this situation. In Fig. 7(a), there is a PB overlapping set $\Gamma_{\{i,j\}}$, and $eHS(\Gamma_{\{i,j\}}) = \{h_1, h_2, h_3\}$. Before the migration, $vm(t_i^P) = v_{11}$, $h_3 \in eHS(\Gamma_{\{i,j\}})$, and $t_i \in \Gamma_{\{i,j\}}$. v_{11} migrates from h_1 to h_3, which causes t_i^P and t_j^B to be scheduled in the same host, leading to the possible failure of t_j when h_3 fails. In Fig. 7(b), $vm(t_j^B) = v_{31}$, $h_1 \in eHS(\Gamma_{\{i,j\}})$, and $t_j \in \Gamma_{\{i,j\}}$. v_{31} migrates from h_3 to h_1, which also causes the possible failure of t_j when h_1 fails.

Next, we extend $\Gamma_{\{i,\cdots\}}$ to $\Gamma^e_{\{i,\cdots\}}$. According to Definition 2 in Section II, $\Gamma^e_{\{i,\cdots\}}$ is extended from $\Gamma_{\{i,\cdots\}}$ by adding a tasks t_x when t_x^B overlaps with t_i^B, $\forall t_i \in \Gamma_{\{i,\cdots\}}$. So on the basis of the first step, we only need to analyze the situation in terms of t_x. Two examples are shown in Fig. 8. In Fig. 8(a), as t_x^B overlaps with t_j^B, $t_j \in \Gamma_{\{i,j\}}$, $\Gamma_{\{i,j\}}$ can be extended to $\Gamma^e_{\{i,j\}} = \{t_i, t_j, t_x\}$, $eHS(\Gamma^e_{\{i,j\}}) = \{h_1, h_2, h_3, h_4\}$. Before the migration, $vm(t_i^P) = v_{11}$, and $t_i \in \Gamma^e_{\{i,j\}}$. v_{11} migrates to $h_4 \in eHS(\Gamma^e_{\{i,j\}})$. In Fig. 8(b), $vm(t_x^B) = v_{41}$, and $t_x \in \Gamma^e_{\{i,j\}}$. v_{41} migrates to $h_1 \in eHS(\Gamma^e_{\{i,j\}})$. Both the migrations in Fig. 8(a) and Fig. 8(b) violate Proposition 7, which may cause a timing conflict between t_j^B and t_x^B when h_4 fails in Fig. 8(a), and when h_1 fails in Fig. 8(b), respectively.

With these constraints, VM migration can be applied without dissatisfaction of fault-tolerant requirements.

IV. Fault-Tolerant Scheduling Algorithm

In this section, we leverage the comprehensive overlapping and VM migration techniques to develop a fault-tolerant scheduling algorithm FSVC. A task is rejected when either the primary or the backup is unable to be scheduled.

A. Algorithm for Scheduling Primaries

To reserve sufficient time laxity for the backups, the primaries should finish as early as possible. In addition, according to Theorem 1, it is the basic requirement of backups overlapping that their primaries are not scheduled on the same host. Thus the primaries should not centralize on several hosts, otherwise it will make overlapping infeasible. Then, two scheduling objectives are listed as follows.

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

To reach the above two objectives, the status of hosts should also be taken into account. Therefore, a two-phase policy is proposed.

- First, determine candidate hosts. Some hosts with fewer primaries scheduled on are chosen as the candidate hosts. In our algorithm, the top $\alpha\%$ hosts with fewer primaries are chosen as candidate hosts.
- Then, choose a VM where the primary are scheduled. Among the VMs that are allocated in the candidate hosts, we choose a VM in which the finish time of the primary is earliest.

With this policy, the newly-arrived primary is scheduled on the host with fewer primaries preferentially. The smaller α is,

Algorithm 1: Primaries Scheduling in FSVC

```
1  Sort $H_a$ in an increasing order by count of scheduled primaries;
2  $H_{candidate} \leftarrow$ top $\alpha\%$ hosts in $H_a$;
3  $find \leftarrow FALSE$; $EFT \leftarrow +\infty$; $v \leftarrow NULL$;
4  while !all hosts in $H_a$ has 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 $EFT_{kl}(t_i^P) \le d_i$ then
9                  $find \leftarrow TRUE$;
10                 if $EFT_{kl}(t_i^P) \le EFT$ then
11                     $EFT \leftarrow EFT_{kl}(t_i^P)$; $v \leftarrow v_{kl}$;
12     if $find == FALSE$ then
13         $H_{candidate} \leftarrow$next top $\alpha\%$ hosts in $H_a$;
14     else
15         break;
16 if $find == FALSE$ then
17     Creat $newVm$ with processing power $P_{new}$;
18     foreach $h_k$ in $H_a$ do
19         if $h_k$ can accommodate $newVm$ then
20             Allocate $newVm$ to $h_k$;
21             $v \leftarrow newVm$; $find \leftarrow TRUE$;
22             break;
23     if $find == FALSE$ then
24         $sourceHost \leftarrow$vmMigration($P_{new}$);
25         if $sourceHost \ne NULL$ then
26             Allocate $newVm$ to $sourceHost$;
27             $v \leftarrow newVm$; $find \leftarrow TRUE$;
28 if $find == FALSE$ then
29     Turn on a host $h_{new}$ in $H - H_a$;
30     if $h_{new}$ can accommodate $v_{new}$ then
31         Allocate $newVm$ to $h_{new}$;
32         $v \leftarrow newVm$; $find \leftarrow TRUE$;
33 if $find == TRUE$ then Allocate $t_i^P$ to $v$;
34 else Reject $t_i$;
```

the narrower of the scope of the candidate hosts is. This can make primaries be scheduled more evenly.

If a primary t_i^P cannot fit in any existing VM, a new VM will be created. The processing power P_{new} of the new VM satisfies the following expression:

$$a_i + cs_i/P_{new} + delay = d_i, \tag{5}$$

where $delay$ denotes the delay that comes from the creation time of a new VM and the boot time of a new host. The pseudocode for the algorithm is presented in Algorithm 1.

The VM migration technique is incorporated in the scheduling of FSVC to improve the resource utilization. The pseudocode is presented in Algorithm 2.

B. Algorithm for Scheduling Backups

Before proposing the algorithm for scheduling backups, we introduce a concept proposed in [9], replication cost, which is defined as

$$C^R_{kl}(t_i^B) = \frac{e_{kl}(t_i^B) - t^O_{kl}(t_i^B)}{e_{kl}(t_i^B)}, \tag{6}$$

Algorithm 2: Function vmMigration(P_{new})

```
Input: $P_{new}$
Output: $sourceHost$
1  Sort $H_a$ in a decreasing order by count of scheduled primaries;
2  $sourceHoset \leftarrow NULL$;
3  foreach $h_i$ in $H_a$ do
4      Sort $h_i.vmList$ in increasing order by count of primaries;
5      $nowP \leftarrow 0$; $migrationList \leftarrow \emptyset$;
6      foreach $v_j$ in $h_i.vmList$ do
7          foreach $h_k$ in inverted $H_a$ do
8              if $v_j$'s migration to $h_k$ meets the VM migration
               constraints && $h_k$ can accommodate $v_j$ then
9                  $nowP \leftarrow nowP + v_j.power$;
10                 $migrationList$.add$\{v_j, h_k\}$;
11                 break;
12         if $nowP \geq P_{new}$ then break;
13     if $nowP \geq P_{new}$ then
14         $sourceHost \leftarrow h_i$;
15         Migrate VMs in $migrationList$;
16         break;
```

where $t_{kl}^{O}(t_i^B)$ denotes the amount of time that t_i^B can overlap with others on v_{kl}. A small $C_{kl}^{R}(t_i^B)$ means that the time actually needed by the backup is small. To improve the resource utilization, two scheduling objectives are listed as follows.

- Schedule backups passive form when possible;
- Schedule backups with small replication cost.

For the two policies, ALAS strategy is applied. The pseudocode for the algorithm is presented in Algorithm 3.

Algorithm 3: Backups Scheduling in FSVC

```
1  Sort $H_a$ in the increasing order by the remaining MIPS;
2  $find \leftarrow FALSE$;
3  foreach $h_k$ in $H_a$ except host($t_i^P$) do
4      foreach $v_{kl}$ in $h_k.vmList$ do
5          Calculate the latest start time $LST_{kl}(t_i^B)$;
6          if $LST_{kl}(t_i^B) \leq d_i$ then
7              $find \leftarrow TRUE$;
8              Calculate the replication cost $C_{kl}^{R}(t_i^B)$;
9              if $LST_{kl}(t_i^B) \geq d_i$ || $LST_{kl}(t_i^B) = LST$ &&
               $C_{kl}^{R}(t_i^B) < C$ then
10                 $LST \leftarrow LST_{kl}(t_i^B)$; $C \leftarrow C_{kl}^{R}(t_i^B)$;
11                 $v \leftarrow v_{kl}$;
12 The following part is similar to line 16∼34 in Algorithm 1.
```

V. Performance Study

In this section, we compare the performance of the proposed algorithm with single-phase-FSVC (SFSVC) and non-migration-FSVC (NMFSVC), which both are derived from FSVC. In SFSVC, the VM where a primary is to be scheduled is chosen directly without consideration of the count of primaries in hosts. The difference between NMFSVC and FSVC lies in NMFSVC does not employ the VM migration technique.

TABLE I
Parameters of tasks

Parameter	Value(Fixed)−(Min,Max,Step)
Computational size($\times 10^5$MI)	([1,2])
Interval time $1/\lambda$	(2) − (0,18,2)
baseDeadline($\times 10^2$s)	(4) − (1.5,8,0.5)

Three performance metrics are considered in our experiments. The first one is *guarantee ratio* (GR), defined to be the percentage of real-time tasks that have been scheduled successfully among all submitted tasks. The second one is *active hosts count* (AHC), which describes the hosts that are turned on in a virtualized Cloud. The third one, *hosts per task* (HPT), is defined to be the ratio between the count of active hosts and that of scheduled tasks.

A. Simulation Method and Parameters

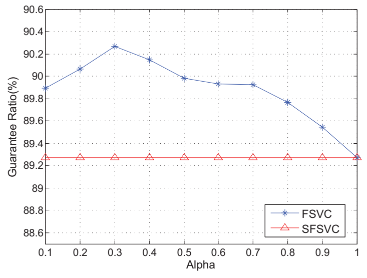
We use CloudSim to simulate a Cloud data center. Each host is modeled to have one CPU core with performance equivalent to 1000, 1500 or 2000 MIPS. Each VM requires one CPU core with 200, 300 or 400 MIPS. We set the time required for turning on a host and creating a VM is 90 and 15 seconds.

In our experiments, 10,000 tasks are made to arrive at the Cloud, which is assumed to follow Poisson distribution with the average interval time $1/\lambda$ that is uniformly distributed in the range $[1/\lambda, 1/\lambda + 2]$. The computational size of tasks are uniformly distributed in the range $[1 \times 10^5, 2 \times 10^5]$. The deadline is chosen as $d_i = a_i + deadlineTime$, and $deadlineTime$ subjects to uniform distribution, $U(baseDeadline, 4baseDeadline)$. Table I gives the parameters and their values. Each experiment runs 50 times.

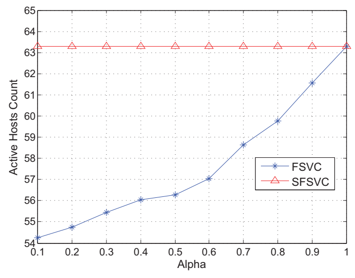
B. Impact of Parameter Alpha on Performance

One of the significant modifications is made to the existing algorithms by the introduction of our two-phase policy. The value of α determines the scope of candidate hosts with fewer primaries in order to make the primaries evenly scheduled. We vary the value of α from 0.1 to 1, while keeping other parameters in table I in a fixed value. When $\alpha = 1$, all the active hosts are chosen as the candidate hosts, so FSVC changes into SFSVC. Fig. 9 shows the results.

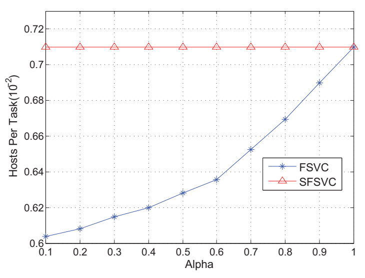
From Fig. 9(a), it can be found that the guarantee ratio of FSVC turns out to be higher than that of SFSVC. This result can be attributed to the fact that two-phase policy makes the primaries evenly scheduled, which enables more backups to overlap with each other, and thus improves the schedulability of the system. Further, we find that the guarantee ratio increases when α increases from 0.1 to 0.3, and reaches the peak around 0.3. Then, it deteriorates into SFSVC with α's approaching 1. The reason is that a small α causes that the scope of candidates is too small, which results in the delay of primaries' execution, and affects the scheduling of backups.



(a)

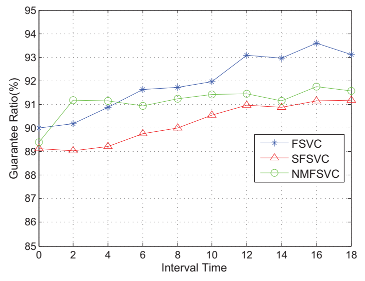


(b)

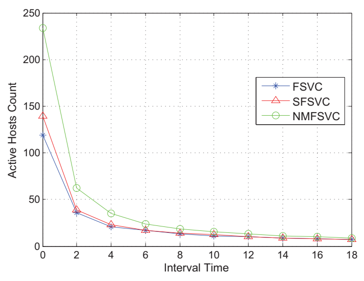


(c)

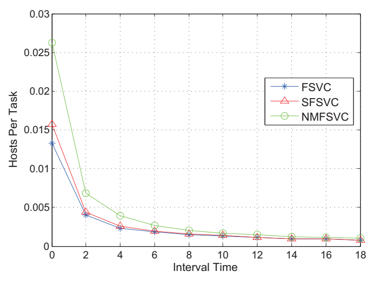
Fig. 9. Performance impact of Alpha.



(a)



(b)



(c)

Fig. 10. Performance impact of Interval Time.

When α approaches 1, the candidates becomes too more, making two-phase policy deteriorate into a single-phase policy.

Fig. 9(b) reveals that the *AHC* of FSVC increases with the increase of α. Nonetheless, FSVC shows a great advantage compared with SFSVC. The reason is similar to that of Fig. 9(a). Thanks to the even distribution of primaries that comes from a small α, more backups can overlap with each other. When α increases, backups occupy more resource.

In Fig. 9(c), we can see that the variation of *HPT* is similar to that of *AHC*. It means FSVC can utilize less computing resources than SFSVC with little change of guarantee ratio. FSVC outperforms SFSVC 20.3% in terms of *HPT*.

The above group of experimental results show that the value of α has a great impact on the performance of FSVC. To get a good tradeoff between the guarantee ratio and resource utilization, we set $\alpha = 0.2$ in the following experiments.

C. Impact of Arrival Rate on Performance

Fig. 10(a) demonstrates that when interval time $1/\lambda$ varies, FSVC, SFSVC and NMFSVC all have a relatively high guarantee ratio, which owes to the infinite resource in the Cloud. However, the arrival of large amounts of tasks almost at the same time causes the system's overload, which results in the rejection of tasks because of the boot time of new hosts and the time restriction of deadline. It is found that NMFSVC outperforms FSVC a bit in the interval from 0 to 5. This can be explained that NMFSVC tends to turn on more hosts, which can relieve the overload for the future arrival tasks.

Fig. 10(b) depicts that FSVC requires fewer active hosts than SFSVC and NMFSVC. This is due to the introduction of two-phase policy and VM migration, which eases the demand of new active hosts. Besides, Fig. 10(b) shows that when the system is overloaded, NMFSVC requires double amount of hosts than FSVC, while the *AHC* of SFSVC is only 30% larger than that of FSVC. It indicates that VM migration is more effective than two-phase policy in terms of saving resources. This is because the two-phase policy just tries to reduces the usage of VMs, when the VM migration technique functions at host level, and reduces the active hosts count directly.

The *HPT*s of the three algorithms are shown in Fig. 10(c). In the interval from 0 to 4, FSVC outperforms SFSVC and NMFSVC 12.1% and 70.8%, respectively. This result further proves that the VM migration technique exhibits huge advantages with respect to improving resource utilization.

D. Impact of Deadline on Performance

In Fig. 11(a), it can be observed that the guarantee ratios of the three algorithms increase obviously when the *baseDeadline* is prolonged. Compared with Fig. 10(a), the impact of deadline on performance is much greater. This is explainable in that the tight deadline makes turning on new hosts meaningless because of the additional boot time. When the deadline is loose enough, almost all tasks can be scheduled in the three algorithms in virtue of the infinite resources in the virtualized Cloud. Besides, Fig. 11(a) shows that the guarantee ratio of SFSVC is the smallest, while that of NMFSVC is even larger than that of FSVC in the interval from 300 to 500. This reason is twofold. First, single-phase policy makes

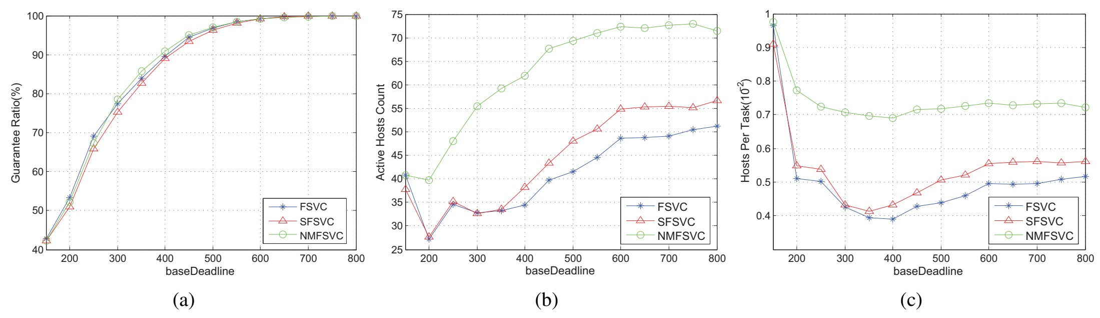


Fig. 11. Performance impact of Deadline.

overlapping difficult for more copies, so more tasks are unable to be scheduled. Second, when the *baseDeadline* varies in the interval from 300 to 500, the deadline is still tight for backups to be scheduled while it is sufficient to turn on new hosts, thereby it is better to turn on hosts than migrating VMs in this condition to improve the guarantee ratio.

In Fig. 11(b), when *baseDeadline* is 150, the deadlines are so tight that all the three algorithms strive to turn on hosts to accommodate these tasks. When the deadline becomes a bit looser, the *AHC*s of FSVC and SFSVC decrease sharply because the time laxity is relatively enough for VM migration. When the deadline becomes further loose, the *AHC* increases as more tasks can be scheduled by turning on new hosts. In addition, we observe that the *AHC*s of FSVC and SFSVC is almost the same in the interval from 150 to 350. This is because two-phase policy narrows the scope of candidate hosts, which postpones the start time of primaries and makes overlapping more difficult, and consequently uses more resources. Therefore, FSVC shows a similar performance to SFSVC in this *baseDeadline* interval in terms of *AHC*.

It is demonstrated in Fig. 11(c) that FSVC always has a smaller *HPT* than SFSVC and NMFSVC. In addition, it can be observed that there is a slow increase of *HPT*s of FSVC and SFSVC when the *baseDeadline* is larger than 400. This is because when *baseDeadline* varies from 200 to 400, VM migration is able to exhibit its advantage of improving resource utilization. However, when the deadline becomes looser, more tasks are accepted while VM migration is not sufficient to support these newly accepted tasks. More hosts need to be turned on for these tasks, which reduces the resource utilization. Thereby the *HPT*s increase.

VI. Conclusions

In this paper, we presented an efficient fault-tolerant scheduling algorithm FSVC, in which independent real-time tasks can tolerate one host failure. The fault-tolerant capability is obtained by using primary-backup approach. FSVC employs all variants of primary-backup approach and VM migration technique to improve the schedulability and enhance the resource utilization. Considering the features of virtualized Clouds, the constraints of overlapping and VM migration are elaborately analyzed. In addition, FSVC adopts a two-phase policy to further enhance the performance of the virtualized Cloud. Extensive simulation studies show that the introduction of VM migration and two-phase policy can improve the quality of scheduling effectively.

References

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, “Xen and the art of virtualization,” *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 164-177, 2003.

[2] 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.

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

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

[5] C. H. Yang, G. Deconinck, and W.-H. Gui, “Fault-tolerant scheduling for real-time embedded control systems,” *Journal of Computer Science and Technology*, vol. 19, no. 2, pp. 191-202, 2004.

[6] H. Liu and S.-M. Fei, “A fault-tolerant scheduling algorithm based on EDF for distributed control systems,” *Journal of Software*, vol. 14, no. 8, pp. 1371-1378, 2003.

[7] F. Yang, W. Luo, and L. Pang, “An efficient real-time fault-tolerant scheduling algorithm based on multiprocessor systems,” *Wuhan University Journal of Natural Sciences*, vol. 12, no. 1, pp.113-116, 2007.

[8] 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.

[9] 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,” *Computers, IEEE Transactions on*, vol. 58, no. 3, pp. 380-393, 2009.

[10] 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,” *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol. 39, no. 3, pp. 626-639, 2009.

[11] X. Zhu, X. Qin, and M. Qiu, “QoS-aware fault-tolerant scheduling for real-time tasks on heterogeneous clusters,” Computers, IEEE Transactions on, vol. 60, no. 6, pp. 800-812, 2011.

[12] W. Sun, Y. Zhang, C. Yu, X. Defago, and Y. Inoguchi, “Hybrid Overloading and Stochastic Analysis for Redundant Real-time Multiprocessor Systems,” in *Reliable Distributed Systems, 2007. SRDS 2007. 26th IEEE International Symposium on*, pp. 265-274, 2007.

[13] Y. Oh and S. H. Son, “Scheduling real-time tasks for dependability,” *Journal of the Operational Research Society*, vol. 48, no. 6, pp. 629-639, 1997.

[14] S. Nanda, T. Chiueh and S. Brook, “A survey on virtualization technologies,” *RPE Report*, pp. 1-42, 2005.