Real-Time Fault-Tolerant Scheduling Algorithm in Virtualized Clouds

Pengze Guo(✉) and Zhi Xue

Department of Electronic Engineering,
Shanghai Jiao Tong University, Shanghai 200240, China
{guopengze,zxue}@sjtu.edu.cn

Abstract. The past decade has witnessed the rapid development of cloud computing. Virtualization, which is the fundamental technique in providing Infrastructure as a Service (IaaS), has led to an explosive growth of the cloud computing industry. Fault-tolerance is a significant requirement of cloud computing due to the Service Level Agreements (SLA). In order to achieve high reliability and resilience of real-time systems in virtualized clouds, a Virtualization-based Fault-Tolerant Scheduling (VFTS) algorithm is proposed. In this paper, fault tolerance is implemented by using primary-backup approach. VFTS is designed for periodic and preemptive tasks in homogeneous environment. Simulation results demonstrate an impressing saving of processing resources compared with those needed by the dual-system hot backup approach, which proves the feasibility and effectiveness of the proposed VFTS algorithm.

Keywords: Fault tolerance · Real-time system · Scheduling · Virtualized cloud

1 Introduction

Cloud computing is a new paradigm for providing computing resources to users on-demand dynamically [1]. The feature of quick deployment relies on virtualization to a large extent. Virtualization is a technology that divides hardware resources to multiple logical computing units using software method [2]. With virtualization, dynamic resource allocation, flexible scheduling and cross-regional sharing can be realized. Virtualization makes it possible to elastically share cloud resources to multi-users at the same time.

A real-time system is described as one that processes data and returns result both correctly and timely [3]. In other words, correctness and timeliness are the main principles of real-time systems. Fault-tolerance plays a significant role in ensuring the functioning of cloud systems, especially for those with safety-critical property (e.g. nuclear power system, electronic cruise control system and medical electronics system). Fault-tolerant scheduling is a superior method which can combine fault-tolerant technique with many different scheduling methods.

Q. Chen et al. (Eds.): ChinaCom 2016, Part II, LNICST 210, pp. 443–452, 2018.
DOI: 10.1007/978-3-319-66628-0_42

Among all the fault-tolerant schemes, primary-backup (PB) approach is the most commonly used one. In the PB approach, each task is represented by two copies, i.e., the primary copy and the backup copy. The primary copy executes when the system functions normally, and the backup copy executes depending on its type. If the backup copy is an active one, it always runs just like primary copy. Passive copy executes only in case of system failure.

In this paper, a novel fault-tolerant scheduling algorithm VFTS that combines both virtualization and PB approach is proposed. VFTS assigns tasks to virtual machines (VMs) instead of to hosts directly like [4]. Meanwhile, VFTS provides fault-tolerance for cloud system by scheduling tasks among different VMs. Schedulability and effectiveness are verified by theorems and experiments. It is shown that VFTS can accomplish the purpose of fault-tolerance and saving more precise computing resources.

The remaining part of the paper is organized as follows. Related work in this area is reviewed in Sect. 2. Section 3 gives the notations, assumptions, and detailed descriptions of the scheduling model. Section 4 deals with the scheduling criteria and constrains. Based on the analysis, scheduling algorithm VFTS is then presented. In Sect. 5, simulation results evaluate the performance of VFTS algorithm compared with the simple duplication approach. Finally, Sect. 6 summarizes the major contribution of this paper and discusses future directions of this work.

2 Related Work

Since assigning real-time periodic tasks to processors has been proved to be NP-hard [5], several heuristic algorithms for allocating tasks have been researched. Rate-Monotonic (RM) algorithm for preemptively scheduling periodic tasks on a single processor was proposed by Liu and Layland [6]. In RM scheduling, tasks with smaller periods have higher priorities, and tasks with low priority will be preempted by tasks with high priority if their running time conflicts. Joseph and Pandya [7] proposed the sufficient and necessary condition for testing the schedulability of a bunch of priority driven tasks on a single processor, called the Completion Time Test (CTT). Rate-Monotonic First-Fit (RMFF), which extended RM to multiprocessor systems, was proposed by Dhall and Liu [8].

As for fault-tolerant scheduling algorithms, active duplication approach is simple and commonly used. In order to reduce system overhead, backup overbooking and deallocation were proposed in [9] to tolerate fault in multiprocessor systems. But it is only for nonpreemptive and aperiodic tasks. Fault-Tolerant Rate-Monotonic First-Fit (FTRMFF) was proposed in [4] by extending the RMFF algorithm and combining backup overbooking and deallocation. Active Resource Reclaiming (ARR) was proposed in [10] to extend FTRMFF with the phasing delay technique [11], which reduces the overlapping between a primary and backup copy. Task Partition based Fault Tolerant Rate-Monotonic (TPFTRM) introduces a new type of backup – the overlapping backup, and abandons active backup to utilize the computing resources more efficiently.

However, none of the fault-tolerant scheduling algorithms mentioned above take virtualization, which is the key feature of cloud systems, into account. Wang et al. proposed a fault-tolerant mechanism FESTAL, which extends the primary-backup model to schedule real-time tasks in clouds [12]. Nonetheless, it is a dynamic algorithm for heterogeneous systems.

In this paper, we investigate a novel static scheduling algorithm that assigns tasks to multiple hosts, each of which contains several homogeneous virtual machines. Three objectives are satisfied: (1) tasks are finished before their deadlines, (2) fault-tolerance is guaranteed, and (3) virtualization characteristics are considered.

3 Scheduling Model

This section presents the characteristic descriptions and notations of the model.

3.1 Fault-Tolerance Model

For the fault-tolerance model, some assumptions are made for sake of convenience:

1. Hosts fail in a fail-stop manner, which means a host either functions well or breaks down.
2. Faults are independent. That is to say, a faulty host can not cause incorrect behaviours in a non-faulty host.
3. There exists a failure detection mechanism. The failure of a host is detected as soon as failure happens.
4. A second failure does not occur before the system recovers from the former failure.
5. All VMs in a faulty host would stop working.

3.2 Task Model

A periodic task t_i is characterized by a pair (C_i, T_i) parameter. Each task must complete before its deadline, which is equal to its period in this paper. Each task t_i has a primary copy t_i^P and a backup copy t_i^B.t_i^P and t_i^B execute on different hosts for purpose of fault-tolerance. Periodic tasks $t_1, t_2, ..., t_n$ are independent and preemptive. Backup copy is usually a simplified version of its primary copy. For the purpose of simplicity, it is assumed that primary and backup copy have the same parameter. The backup copy has two status: active and passive. Let W_i be the worst-case completion time (WCRT) of t_i^P. The recovery time of t_i^B is $B_i = T_i - W_i$. If $B_i < C_i$, then set $st(t_i^B)$ to active. Because if t_i^B is a passive backup copy, it would not start execute until failure occurs, and if failure happens just at the WCRT of its primary copy, it would not have enough time to recover from failure. If $B_i \geq C_i$, set $st(t_i^B)$ to passive.

A virtualized host is denoted as h_i. Each host h_i can hold multiple VMs, denoted as $vm_{i1}, vm_{i2}, ..., vm_{im}$. For sake of convenience in comparing between

different scheduling algorithms, it is assumed that each host accommodates the same number of identical VMs.

t_i^P and t_i^B are assigned to VMs instead of to 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. $host(t_i^P)$ and $host(t_i^B)$ are their corresponding hosts.

After assigning tasks to VMs, the task copies on the same VM are scheduled by the RM algorithm. Given n periodic tasks $t_1, t_2, ..., t_n$, the goal of the fault-tolerant scheduling algorithm is to minimize the number of VMs.

To facilitate the analysis, we summarize the notations of task model in Table 1.

Table 1. Task model parameters

Symbol	Meaning
t_i	Task i
t_i^P	Primary copy of t_i
t_i^B	Backup copy of t_i
C_i	Computation time of t_i
T_i	Period of t_i
W_i	Worst-case response time (WCRT) of t_i^P
B_i	Recover time of backup copy t_i^B
h_i	Host i
vm_{ij}	jth VM of h_i
$primary(vm_{ij})$	Primary copies on vm_{ij}
$active(vm_{ij})$	Active copies on vm_{ij}
$recover(vm_{ij}, h_f)$	Backup copies on vm_{ij} whose primary copies are on h_f
$st(t_i^B)$	Status of t_i^B
$pri(t_i)$	Priority of t_i
$N_{task}, N_{host}, N_{vm}$	Number of tasks, hosts and VMs per host

4 Virtualization-Based Fault-Tolerant Scheduling Algorithm VFTS

In this section, we leverage the virtualization techniques and scheduling schemes to develop a virtualization-based fault-tolerant scheduling algorithm VFTS. Scheduling strategies of primary, active and passive copies are discussed in detail. The pseudocode of VFTS algorithm is presented. It should be noted that all tasks should be re-indexed with the decreasing order of their priorities which are inversely proportional to their periods. The backup copy is scheduled following its primary. So the actual scheduling order is $t_1^P, t_1^B, t_2^P, t_2^B, t_3^P, t_3^B, \cdots$ with $T_1 \leq T_2 \leq T_3 \leq \cdots$.

4.1 Scheduling Criteria

In the classic paper [6], a critical instant of a task is defined to be a particular time when a task will get the latest finishing time.

Theorem 1. *A critical instant for any task occurs whenever the task is requested simultaneously with requests for all higher priority tasks.*

As a consequence, to check whether a task is schedulable, we just need to check its schedulability when it starts to execute with all higher priority tasks.

Joseph and Pandya proposed the sufficient and necessary condition for verifying the schedulability of a set of fixed-priority tasks on a single processor, called Completion Time Test (CTT) [7].

Theorem 2. *To check whether task t_i is schedulable in a VM, its worst-case response time (WCRT) is:*

$$W(t_i, \tau) = \sum_{\tau_k \in \tau} C_k \lceil W(t_i, \tau)/T_k \rceil \tag{1}$$

where τ is the task set assigned to the VM with priorities equal to or higher than t_i (including t_i). If $W(t_i, \tau) \leq T_i$, then t_i is schedulable in the VM.

To calculate the WCRT, an iterative method was proposed in [10]. The computation time on a processor occupied by tasks in VM during $[0, t]$ is:

$$W(t, \tau) = \sum_{\tau_k \in \tau} C_k \lceil t/T_k \rceil \tag{2}$$

$\lceil t/T_k \rceil$ is the number of periods that t_k experiences during interval $[0, t]$. Owing to t_k's higher priority than t_i, the length of the time interval occupied by t_k is $C_k \lceil t/T_k \rceil$. Let $S_0 = \sum_{\tau_k \in \tau} C_k$, and iterate $S_{l+1} = W(S_l, \tau)$ with $l = 0, 1, 2...$ until $S_n = S_{n+1}$. If $S_n \leq T_i$, then t_i is schedulable and its WCRT in the VM is S_n.

Theorem 3. *If $vm(t_i^P) \in h_j$, then $vm(t_i^B) \notin h_j$.*

Proof. Prove by contradiction. Suppose that t_i^B is assigned to $vm_{jk} \in h_j$, then $host(t_i^P) = host(t_i^B)$. When h_j fails, all the VMs in h_j fail. Thus, both primary and backup copies of t_i cannot execute. Therefore, t_i^P and t_i^B cannot be assigned to VMs in the same host. □

Theorem 4. *If t_i^* is a primary copy or active backup copy, then in case of fault free, the WCRT of t_i^* is*

$$W(t_i^*, \tau) = \sum_{t_k^P, t_k^B \in \tau} C_k \lceil W(t_i^*, \tau)/T_k \rceil \tag{3}$$

where τ is the primary copies or active backup copies on $vm(t_i^*)$ with priorities equal to or higher than t_i^* (including t_i^*). If $W(t_i^*, \tau) \leq T_i$, then t_i^* is schedulable in the VM.

Proof. The primary copies and active backup copies always execute in fault free case, and passive backup copies do not execute. The computation of WCRT is similar to Theorem 2. □

Theorem 5. *If $st(t_i^B)$=passive, then in case of fault free, t_i^B need not execute.*

Theorem 6. *For primary copy t_i^P, in the presence of h_f's failure ($h_f \neq host(t_i^P)$), the WCRT of t_i^P is*

$$W(t_i^P, \tau) = \sum_{t_k^P \in \tau} C_k \lceil W(t_i^P, \tau)/T_k \rceil + \sum_{t_k^B \in \tau} C_k \phi(t_k^B, W(t_i^P, \tau)) \tag{4}$$

where

$$\phi(t_k^B, t) = \begin{cases} \lceil t/T_k \rceil & if\ st(t_k^B) = active \\ 1 & if\ st(t_k^B) = passive\ and\ t \leq B_k \\ 1 + \lceil (t - B_k)/T_k \rceil & if\ st(t_k^B) = passive\ and\ t > B_k \end{cases}$$

$$\tau = \{\tau_k | \tau_k \in primary(vm(t_i^P)) \cup recovery(vm(t_i^P), h_f), pri(\tau_k) \geq pri(t_i^P)\}.$$

If $W(t_i^P, \tau) \leq T_i$, then t_i^P is schedulable in the VM.

Proof. When failure occurs, passive backup copy t_k^B needs to finish C_k during its recovery time. After the recovery interval, t_k^B enters the periodic circulation, and finish C_k in every period. □

Theorem 7. *For backup copy t_i^B, in the presence of $host(t_i^P)$'s failure, the WCRT of t_i^B is*

$$W(t_i^B, \tau) = \sum_{t_k^P \in \tau} C_k \lceil W(t_i^B, \tau)/T_k \rceil + \sum_{t_k^B \in \tau} C_k \phi(t_k^B, W(t_i^B, \tau)) \tag{5}$$

where $\phi(t_k^B, t)$ is the same as that in Theorem 6, and

$$\tau = \{\tau_k | \tau_k \in primary(vm(t_i^B)) \cup recovery(vm(t_i^B), host(t_i^P)), pri(\tau_k) \geq pri(t_i^B)\}.$$

If $W(t_i^B, \tau) \leq T_i$ for active backup copy or $W(t_i^B, \tau) \leq B_i$ for passive backup copy, then t_i^B is schedulable in the VM.

Proof. For passive backup copy t_k^B, the worst case happens when the failure occurs in $host(t_k^P)$ just at the moment when t_k^P is about to finish at its WCRT, which means the time left for t_k^B to recovery is shortest, i.e., B_k. t_k^B must accomplish C_k in its recovery interval with the length of B_k. □

4.2 VFTS Algorithm

Combining the criteria and constraints stated above, the detailed description of the VFTS algorithm is given in Algorithm 1.

Algorithm 1. VFTS Algorithm

```
1  Sort tasks t_1, t_2, ..., t_n such that T_1 ≤ T_2 ≤ ··· ≤ T_n
2  foreach t_i in the task set do
3      foreach h_j in the host set do                                   // loop 1
4          foreach vm_k in the VMs of h_j do
5              (CheckNoFault, W_i) ←NoFaultCTT(t_i^P, vm_jk)
6              if CheckNoFault == True then
7                  for f = 1 → N_host do
8                      CheckFault ←FaultCTT(t_i^P, vm_jk, h_f)
9                      if CheckFault == False then
10                         break
11                 if CheckFault == False then
12                     continue
13                 Allocate t_i^P → vm_jk
14                 Break out of loop 1
15     if t_i^P fails to be scheduled in any VM then
16         Add a new host with N_host ← N_host + 1
17         Allocate t_i^P → vm_{N_host,1}
18     if T_i − W_i < C_i then
19         Status(t_i^B) ← active
20         foreach h_j in the host set do                               // loop 2
21             foreach vm_k in the VMs of h_j do
22                 CheckNoFault ←NoFaultCTT(t_i^B, vm_jk)
23                 CheckFault ←FaultCTT(t_i^B, vm_jk, h_{t_i^P})
24                 if CheckNoFault==True and CheckFault==True then
25                     Allocate t_i^B → vm_jk
26                     Break out of loop 2
27     else
28         Status(t_i^B) ← passive
29         foreach h_j in the host set do                               // loop 3
30             foreach vm_k in the VMs of h_j do
31                 CheckFault ←FaultCTT(t_i^B, vm_jk, h_{t_i^P})
32                 if CheckNoFault==True and CheckFault==True then
33                     Allocate t_i^B → vm_jk
34                     Break out of loop 3
35     if t_i^B fails to be scheduled in any VM then
36         Add a new host with N_host ← N_host + 1
37         Allocate t_i^B → vm_{N_host,1}
```

Function NoFaultCTT(t_i^*, vm_{jk}) is used to check schedulability of t_i^* on vm_{jk} in fault free case. For primary copy t_i^P, NoFaultCTT(t_i^P, vm_{jk}) tests the schedulability of $\tau = t_i^P \cup primary(vm_{jk}) \cup active(vm_{jk})$ on vm_{jk} using CTT. The

WCRT W_i of t_i^P is also calculated by this function. For active backup copy t_i^B, NoFaultCTT(t_i^B, vm_{jk}) tests the schedulability of $\tau = t_i^B \cup primary(vm_{jk}) \cup active(vm_{jk})$ on vm_{jk} using CTT. Passive backup copies do not need to perform this test because they do not execute in non-fault case.

FaultCTT(t_i^*, vm_{jk}, h_f) is used to check schedulability of t_i^* on vm_{jk} in case h_f ($f \neq k$) encounters a failure. The analysis is in concert with Theorem 4.

Theorem 8. *The time complexity of VFTS is* $O(N_{task}N_{host}^2N_{vm})$.

Proof. To assign a primary copy, at most $N_{host}N_{vm}$ VMs are tried. Each trial requires in turn one execution of NoFaultCTT and $N_{host} - 1$ executions of FaultCTT, in the worst case. Thus, the time complexity of VFTS is $O(N_{task})O(N_{host}N_{vm})O(N_{host}) = O(N_{task}N_{host}^2N_{vm})$. □

5 Simulation Experiments

In order to calculate the number of VMs needed by the VFTS algorithm to provide fault-tolerance for virtualized cloud systems, simulation experiments are performed. Simple duplication with RMFF (DRMFF) is used to compare with VFTS. DRMFF schedules tasks with the method of RMFF and provide fault-tolerance by duplicating the hosts. We denote with N the number of VMs required by VFTS algorithm, and with M the number of VMs required by DRMFF algorithm. Since the optimal assignment of tasks to VMs is difficult to figure out, we use the total load $U = U_1 + U_2 + \cdots + U_n$ as the minimum and optimal number of VMs. For simplicity of comparison, each host is assumed to contain 8 identical VMs. Once a new host is added, 8 VMs are created simultaneously. The VFTS and DRMFF algorithms were developed in Matlab and run on a PC with Intel Core i7-2600K CPU and 8 GB RAM.

Task sets with length of $100 \leq n \leq 1000$ are generated. The maximum load $\alpha = max\{U_1, U_2, \cdots, U_n\}$ is chosen to be 0.2, 0.5 and 0.8. Each task period T_i is randomly distributed in the interval [1, 500], and each computation time C_i is uniformly distributed in the interval [0, αT_i]. For every chosen n and α, the experiment is repeated for 30 times, and the average result is calculated.

Figure 1 shows the ratios between the number of VMs required by VFTS or DRMFF and the total load. Generally the ratio increases with the increasing of α. The increasing trend of DRMFF is less obvious than VFTS because DRMFF does not take into account the complicated relations and constraints between primary and backup copies, and always tries to fully fill every VM regardless of α. For smaller number of tasks, the ratios are relatively higher because more VMs are not fully utilized due to limited number of tasks.

Figure 2 shows a remarkable saving of VMs compared with those needed by DRMFF. The percentage of VMs saved by VFTS is about 33% for $\alpha = 0.2$, 25% for $\alpha = 0.5$ and 4% for $\alpha = 0.8$ when the number of tasks is large enough.

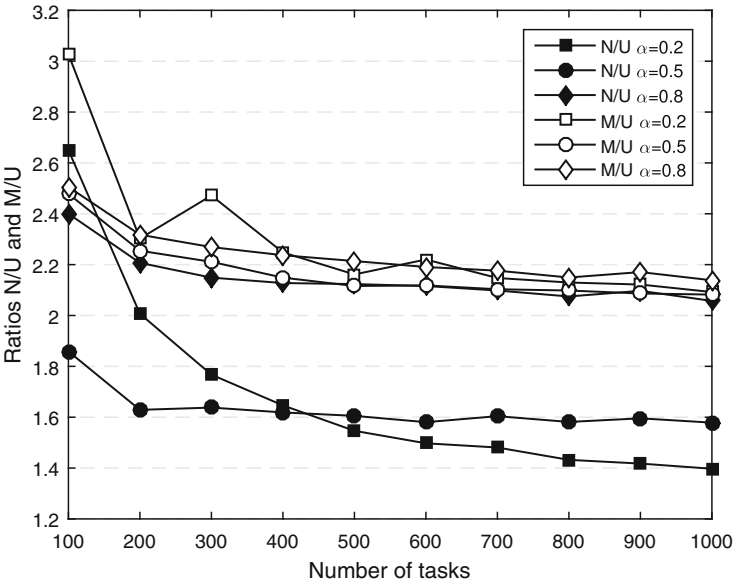

Fig. 1. Ratios between the number of VMs required by VFTS or DRMFF and the total load.

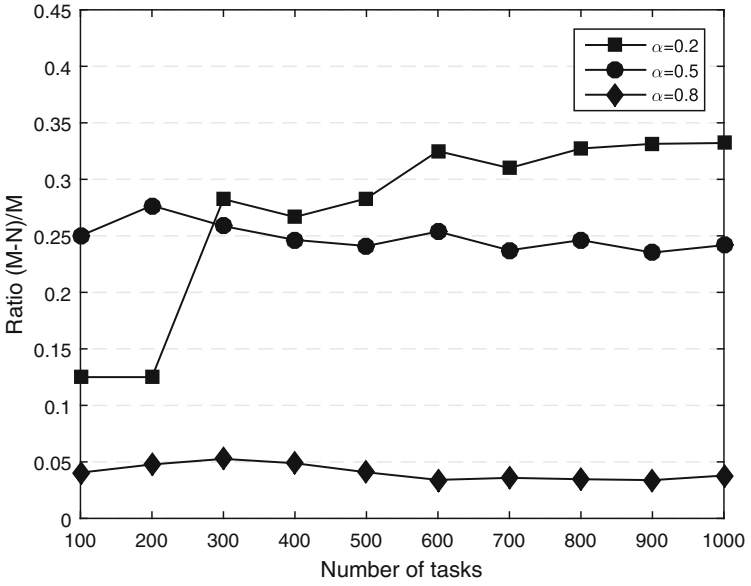

Fig. 2. The percentage of the VMs that VFTS saved compared with DRMFF.

6 Conclusions

This paper considers the problem of providing fault-tolerance for virtualized clouds with scheduling algorithms. VFTS combines the characteristics of cloud and traditional scheduling schemes to provide simple, efficient and low-cost fault-tolerance. VFTS assigns tasks in virtualized clouds instead of isolated computing nodes. Compared with simple duplication fault-tolerant method, VFTS makes use of the computing resources more efficiently. The analysis and simulation results have verified the effectiveness of the VFTS algorithm.

Finally, future research could deal with the strategies of assignment, which can further leverage the idle time of passive backup copy and reduce redundancy of active backup copies.

Acknowledgment. This work was supported in part by the National Natural Science Foundation of China under Grant No. 61332010.

References

1. Mell, P.M., Grance, T.: The NIST definition of cloud computing. Technical report, National Institute of Standards and Technology, Gaithersburg (2011)
2. Nanda, S., Chiueh, T.: A survey on virtualization technologies. Technical report, Department of Computer Science, SUNY at Stony Brook (2005)
3. Stankovic, J.: Misconceptions about real-time computing: a serious problem for next-generation systems. Computer **21**(10), 10–19 (1988)
4. Bertossi, A., Mancini, L., Rossini, F.: Fault-tolerant rate-monotonic first-fit scheduling in hard-real-time systems. IEEE Trans. Parallel Distrib. Syst. **10**(9), 934–945 (1999)
5. Leung, J.Y.T., Whitehead, J.: On the complexity of fixed-priority scheduling of periodic. Real-Time Tasks. Perform. Eval. **2**(4), 237–250 (1982)
6. Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard-real-time environment. J. ACM **20**(1), 46–61 (1973)
7. Joseph, M., Pandya, P.: Finding response times in a real-time system. Comput. J. **29**(5), 390–395 (1986)
8. Dhall, S.K., Liu, C.L.: On a real-time scheduling problem. Oper. Res. **26**(1), 127–140 (1978)
9. Ghosh, S., Melhem, R., Mosse, D.: Fault-tolerance through scheduling of aperiodic tasks in hard real-time multiprocessor systems. IEEE Trans. Parallel Distrib. Syst. **8**(3), 272–284 (1997)
10. Bertossi, A., Mancini, L., Menapace, A.: Scheduling hard-real-time tasks with backup phasing delay. In: 10th IEEE International Symposium on Distributed Simulation Real-Time Applications, pp. 107–118. IEEE (2006)
11. Tindell, K.: Adding Time-Offsets to Schedulability Analysis, pp. 1–28. University of York, Deparment of Compututer Science (1994)
12. Wang, J., Bao, W., Zhu, X., Yang, L.T., Xiang, Y.: FESTAL: fault-tolerant elastic scheduling algorithm for real-time tasks in virtualized clouds. IEEE Trans. Comput. **64**(9), 2545–2558 (2015)