*Intelligent Automation & Soft Computing*
DOI: 10.32604/iasc.2023.033703
***Article***

Tech Science Press

# A Broker-Based Task-Scheduling Mechanism Using Replication Approach for Cloud Systems

**Abdulelah Alwabel***

Department of Computer Sciences, Prince Sattam Bin Abdulaziz University AlKharj, KSA
*Corresponding Author: Abdulelah Alwabel. Email: a.alwabel@psau.edu.sa
Received: 24 June 2022; Accepted: 29 September 2022

**Abstract:** The reliability and availability of cloud systems have become major concerns of service providers, brokers, and end-users. Therefore, studying fault-tolerance mechanisms in cloud computing attracts intense attention in industry and academia. The task-scheduling mechanisms can improve the fault-tolerance level of cloud systems. A task-scheduling mechanism distributes tasks to a group of instances to be executed. Much work has been undertaken in this direction to improve the overall outcome of cloud computing, such as improving service quality and reducing power consumption. However, little work on task scheduling has studied the problem of lost tasks from the broker's perspective. Task loss can happen due to virtual machine failures, server crashes, connection interruption, etc. The broker-based concept means that the backup task can be allocated by the broker on the same cloud service provider (CSP) or a different CSP to reduce costs, for example. This paper proposes a novel fault-tolerant mechanism that employs the primary backup (PB) model of task scheduling to address this issue. The proposed mechanism minimizes the impact of failure events by reducing the number of lost tasks. The mechanism is further improved to shorten the makespan time of submitted tasks in cloud systems. The experiments demonstrated that the proposed mechanism decreased the number of lost tasks by about 13%–15% compared with other mechanisms in the literature.

**Keywords:** Cloud computing; task scheduling; fault tolerance; replication; broker-based

## 1 Introduction

Cloud computing has become an efficient paradigm that offers computational abilities on a pay-per-usage basis. Parallel applications demonstrate a decline in the utilization of central processing unit (CPU) resources when parallelism increases [1]. Performance is reduced if tasks are not appropriately scheduled because cloud systems process vast amounts of data. Thus, scheduling mechanisms can play a crucial role in cloud computing. Task-scheduling algorithms plan and execute tasks with the most significant evaluated gains and benefits. The virtual machine (VM) layer of cloud systems handles the complexity of task scheduling, so scheduling can play a vital role in efficiently and effectively assigning resources to each task [2].

The main features of cloud systems are elasticity, resource pooling, and location independence [3]. Elasticity refers to users quickly acquiring more computing services or releasing them; resource pooling refers to users sharing a server in a cloud's data center without noticing others using virtual technology; location independency refers to users accessing cloud services anytime and anywhere. These features make cloud computing quite appealing to the public and private sectors. Still, they also make cloud systems prone to failures without prior knowledge of internal faults, such as hardware crashes, and external faults, such as network interruptions [4]. Therefore, due to the nature and features of cloud systems, one of the main challenges in cloud computing is to develop fault-tolerance mechanisms.

This phenomenon is even more noticeable in most cloud service providers, such as Amazon, because they are built on inexpensive commodity hardware with a much higher failure probability [5]. Amazon's EC2, for example, experienced a server failure in elastic block storage, bringing down thousands of hosted websites and applications for about 24–72 h [6]. Fault tolerance refers to a cloud system's ability to detect, identify, and handle a fault with minimum or no loss in the final output. Scheduling mechanisms approach to fault tolerance efficiently by allocating several copies of tasks on different computing instances [7]. Failures can affect the outcome of cloud systems. One outcome to be considered is the loss of tasks submitted for processing to a cloud system.

This paper presents a broker-based replication technique (BBRT). This novel task-scheduling mechanism allows the broker/client to assign cloud tasks so that if failure events happen on the node that is processing a task, another backup is processed on a different node. The broker-based concept means that the broker can allocate the backup task on the same or another cloud service provider (CSP) to reduce costs, for example. The main contributions of this paper are as follows: first, developing a novel task-scheduling mechanism that employs a primary backup (PB) technique to reduce lost tasks in cloud systems; second, enhancing that by adding a BBRTplus mechanism to decrease the total time required to execute a cloud task; third, showing the impact of failure events on the number of lost tasks using extensive simulation experiments based on real-world traces and comparing these mechanisms with the state-of-art mechanisms; and finally, studying the effectiveness of the proposed mechanism using simulation-based experiments for both private and public cloud systems.

The remainder of the paper is organized as follows: Section 2 provides background about scheduling mechanisms in the environment of cloud computing. Section 3 reviews related work in the literature. Section 4 explains the proposed mechanism, and Section 5 discusses their evaluation. Finally, Section 6 concludes with a summary and recommendations for future work.

## 2 Background

Resource management in cloud computing has several benefits, including better resource utilization, scalability, quality of service (QoS), and throughput. Therefore, since the emergence of cloud computing, the development of resource-management algorithms and mechanisms has attracted many researchers to improve the overall performance of cloud systems by maximizing their benefits. Two general approaches can be employed to achieve this goal: mapping VMs to physical machines (PMs) and assigning cloud tasks to VMs to be executed.

However, these two steps can be confused because they tackle similar issues in cloud systems. The literature also uses various terms and synonyms to describe the same mechanism: the mechanism that allocates or maps a VM to a PM can be called a VM-placement mechanism [8], a resource-allocation mechanism [9], a VM-management mechanism [10], a VM-allocation mechanism [11], or a VM-scheduling mechanism [12]. This paper uses the term VM-allocation mechanism to refer to the allocation of VMs to PMs. Submitting cloud tasks to be executed on VMs can be called a service-level agreement, a resource allocator [13], a service-scheduler mechanism [14], or a task-scheduling mechanism [15]. The term task-scheduling mechanism describes this process throughout this paper.

CSPs only implement a VM-allocation mechanism because it can be employed within the underlying infrastructure, such as data centers. However, this is not true for task-scheduling technique, as they can be implemented by CSPs, cloud brokers, and even cloud end-users. Fig. 1 illustrates the framework of a task-scheduling process employed by the proposed mechanism. In this scenario, an end-user submits an application or list of applications to a broker, and each application can be handled as a list of cloud tasks. The broker then presents a list of tasks to one CSP while submitting another list to a different CSP for various reasons. For example, the broker submits to one CSP that offers faster computation time while submitting to another that provides a cheaper storage service.

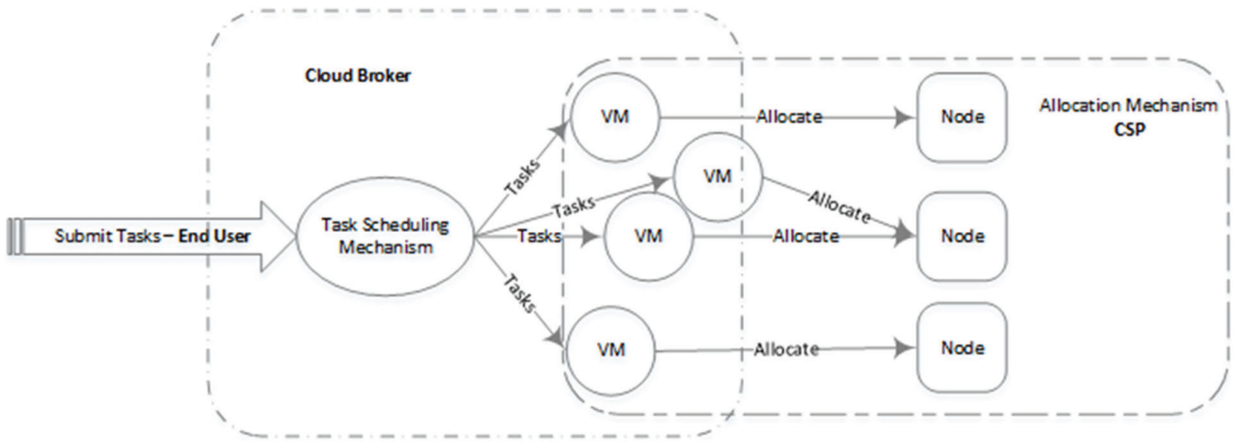

**Figure 1:** Task-scheduling framework

Failure events in cloud systems can affect their outcomes [16]. Fault types can vary based on computing resources and capabilities, including network, physical, process, and processor faults. Three general categories of techniques are employed to implement fault-tolerance mechanisms in cloud systems [17]: redundancy techniques, fault-aware policies, and load-balancing (LB) fault-tolerance techniques.

Redundancy techniques include the redundancy of hardware, software, and time. Hardware redundancy is the replication of hardware components to an identical number, each performing identical operations to mask faults. Software redundancy runs the same software modules on different hardware units and devices. A more specific version of this technique is running multiple versions of the same program with the same input but with varying algorithms of implementation [17]. Finally, for time redundancy, hardware and software components are identical to hide faults from users. Time-redundancy techniques tackle failures in cloud computing but are expensive and can yield poor resource utilization.

Fault-tolerance policies have two broad categories: proactive and reactive policies. Proactive policies try to predict and take proper action before faults occur to alleviate unwanted consequences. Reactive policies, in contrast, prepare no measures to predict or prevent failures because such measurements can waste resources and increase response time. LB fault-tolerance techniques distribute cloud systems to minimize both the probability of failure and the negative impacts of faults [18]. These techniques reduce the load on each working component, resulting in fewer faults and, thus, less output loss from a particular cloud system.

## 3 Related Work

The authors in [19] presented a heuristic mechanism called the deadline early tree. Their work minimized the cost of deadline-constrained applications in workflow scheduling to satisfy multiple QoS parameters. However, they did not consider the communication time between tasks. Likewise, [20] presented an efficient algorithm to minimize scheduling costs by selecting processors to satisfy budget constraints and reduce the schedule length of applications.

The authors in [21] proposed a task-scheduling algorithm based on an enhanced min-min technique. Their work determined the tasks with shorter execution times and the most appropriate resources for each task, resulting in the fastest times possible. However, their technique may lead to delays in cloud systems [22].

The authors in [23] proposed multi-object task scheduling using a particle-optimization algorithm based on a novel ranking strategy. The main contribution of this mechanism was scheduling tasks to VMs to minimize waiting time and maximize system throughput. The authors in [24] proposed novel task scheduling that combined particle-swarm and ant-colony algorithms by adjusting several learning factors to optimize scheduling tasks' fitness, cost, and operation cycles.

The authors in [25] presented a task-scheduling framework with a two-stage strategy. The first stage used a Bayes classifier to classify tasks based on historical data. Tasks were then dynamically matched in the second stage with the most suitable pre-created VMs to reduce waiting time and achieve LB among resources. However, this work assumed that VMs could be created beforehand, which is not always the case in cloud systems.

The authors in [26] developed an LB mechanism to update particle location, improving task scheduling and LB performance. The authors in [22] proposed a novel scheduling algorithm based on a prediction technique that estimated the computation times of tasks and communication time to improve the turnaround time of tasks and reduce computation and complexity. However, this proposal could not handle task loss.

The authors in [27] proposed multi-objective task scheduling for cloud systems. This mechanism considered execution time, bandwidth, CPU, and storage parameters to assign tasks to VMs using an optimization method. No previous studies have tackled the problem of failure events that can cause task loss in cloud systems.

The most acknowledged methods of managing failures to enhance the performance and reliability of distributed computing systems are the resubmission and replication of data to multiple hosts in different locations [28]. Resubmission attempts to re-execute the tasks on other standard computing resources after a failure [5]. The map-reduce approach handles failures in worker nodes, where the tasks on failed hosts are reset to their initial states and re-executed on other workers [29]. The authors in [30] proposed a novel fault-tolerance heuristic called resubmission impact for this purpose. However, resubmission approaches involve a substantial delay in re-executing tasks on other hosts after failures occur.

Replication is widely adopted to implement fault-tolerant task-scheduling mechanisms. Its primary concept in cloud computing is to execute multiple copies of the same task using different virtual resources to tolerate failure events that occur during runtime. Such techniques can enhance the reliability and flexibility of cloud systems [5]. However, replication can cause various problems in resource utilization and thus increase energy consumption. In addition, it can substantially increase the total execution time of tasks, which makes it unsuitable for real-time tasks due to their strict timing constraints.

The authors in [31] developed a fault-tolerant scheduling mechanism based on cloud systems' PB model to achieve fault tolerance and LB. However, they did not thoroughly examine the deadline restrictions of time-sensitive tasks or the improving methods of resource utilization.

The authors in [5] proposed a fault-tolerant task-scheduling mechanism for cloud systems (FESTAL) that used the PB model to replicate tasks during run time. The authors considered node failures the leading causes of lost tasks. In addition, FESTAL tried to improve resource utilization by dynamically migrating VM backups from underutilized cloud nodes to more suitable nodes. However, the performance of FESTAL is questionable because the complexity of its scheduling mechanism is $\mathcal{O}(n^3)$, which makes its performance decline when the number of nodes and VMs grows. In addition, the FESTAL mechanism was designed to be managed by the CSP, leaving brokers with no control over tasks.

The authors in [28] proposed an energy-aware fault-tolerant dynamic scheduling scheme (EFDTS) that optimized resources by saving energy while maintaining an acceptable level of fault tolerance. This study employed the replication technique to schedule tasks based on historical scheduling records to classify arrival tasks using a Bayes classifier. The mechanism observed tasks and VM data to determine suitable VMs to host tasks.

Their scheduling algorithm tries to determine the types of available VMs before classifying tasks to assign suitable tasks to appropriate VMs. If no suitable VM exists for a specific task, the mechanism creates a new VM for the task's requirements. This work is carried out by the CSP and assumes the system is flexible regarding the number of running VMs. In other words, the system can create more VMs during runtime as needed, which may lead to increased user costs. Considering a failure-aware task-scheduling mechanism employed by brokers or end-users can be challenging. However, the literature shows that little research has been undertaken to develop fault-tolerant mechanisms from the broker's aspect.

## 4 The Proposed Mechanism

This section presents a novel task-scheduling mechanism that tackles the problem of failures in cloud systems from the broker's perspective. This section explains the failure model in cloud systems, which can affect any number of successfully executed tasks on a list of submitted tasks. It then presents the proposed mechanism and discusses how it is further extended to yield an improved outcome.

### 4.1 Failure Model

A cloud task submitted by a cloud broker for implementation by a CSP may be as follows:

$$c = \begin{cases} sucess, & if\ vm_c \notin VM_l \\ lost, & if\ vm_c \in VM_l \end{cases} \tag{1}$$

where $c$ denotes a cloud task, $vm_c$ is a VM currently processing $c$, and $VM_l$ is the set of all VMs hosted in one or a group of PMs that fail during runtime. A failure is defined in this paper as any kind of service interruption that leads to the loss of currently processed cloud tasks. The service interruption can be due to an intermittent connection, server crashes in data centers or VM operating-system crashes in VMs. Let $vm_u$ is the utilization level of a vm, and it is given as:

$$vm_u = 100 \times \frac{\sum_{i=1}^{m} Len(c_i)}{vm_{cpu}};\ \ c_i \in C \tag{2}$$

where $Len(c_i)$ is the length of a cloud task and $vm_{cpu}$ is the computing power of a VM. As a result of a failure event of a PM, all currently hosted VMs to this PM are lost. However, a failing PM can be up and running again to host new VMs. The failure period ($f_p$) is the period that lasts for a PM while it is down. The number of failure events plays a crucial role in deciding the number of lost tasks because the more failed PMs, the more lost VMs; therefore, more tasks will be lost. Let $PM_t$ be the total number of running PMs in a cloud system; total failure time ($FT$) is then as follows:

$$FT = \sum_{i=1}^{PM_t} pm_i \times f_p; \;\; \forall\, pm_i \in PM \tag{3}$$

where $PM$ denotes all running PMs in a cloud system and $pm_i$ is a PM in the set $PM$. Let $F_r$ refer to the ratio of failing PMs to the total number of running PMs. $F_r$ is then given as follows:

$$F_r = \frac{\sum pm}{PM_t} \times 100; \;\; \forall\, pm \in PM \tag{4}$$

The number of lost cloud tasks $n_{lc}$ is given as follows:

$$n_{lc} = \sum_{i=1}^{m} c_i; \;\; \forall\, c_i \in C \tag{5}$$

In this context, throughput can refer to the ratio of the number of successfully executed cloud tasks to the number of submitted tasks, denoted as $c_n$. Throughput ($P$) is given as follows:

$$P = \frac{c_n - n_{lc}}{c_n} \times 100 \tag{6}$$

The execution time is denoted as $t_e(c) =$. For a cloud task $c$, it is calculated as follows:

$$t_e(c) = \frac{c_l}{vm_{cpu}} \tag{7}$$

where $c_l$ denotes the length of the task $c$, which is measured in millions of instructions (MIs), and $vm_{cpu}$ denotes the CPU power of the hosted VM. The makespan time (MST) is the total time required for a task from its submission to execution [7]. It is:

$$MST(c) = t_{wc} + t_e(c) - t_{sc} \tag{8}$$

where $t_{wc}$ refers to the waiting time of task $c$ before it starts executing and $t_{sc}$ refers to the submission time of the task. Notably, the more tasks assigned to a single VM, the longer the waiting time because more tasks are stacked on the waiting list. Therefore, reducing the number of tasks on the waiting list is crucial to reduce the MST of each task assigned to a particular VM. During the run time of a cloud system, service level agreement violations (SLAVs) are defined if a cloud task cannot be finished and executed by a given deadline, denoted as $d_{SLA}$. SLAV is given as follows:

$$SLAV(c) = \begin{cases} 1, & if\ MST(c) > d_{SLA} \\ 0, & if\ MST(c) < d_{SLA} \end{cases} \tag{9}$$

### *4.2 BBRT Mechanism*

This section presents the BBRT, a novel task-scheduling technique that tackles the problem of failure events in a cloud computing environment from the broker/client perspective. This means that the task scheduler is implemented on the broker/client side so the scheduler can allocate backup tasks in different CSPs to reduce costs [1] since the primary copy is already running for each task. The BBRT mechanism is depicted in Algorithm 1, which starts by getting two lists: *taskList* and *vmList* lists. *TaskList* refers to the list of all submitted cloud tasks, while *vmList* refers to all VMs that host tasks. The mechanism tries to assign tasks to the VM found in a list of available VMs with the lowest utilization level; the VM with the fewest assigned tasks. The utilization level is given in Eq. (2). In the algorithm, the BBRT mechanism adopts a replication technique based on the PB model, which is popular for fault-tolerant scheduling.

Each task has two copies executing on two different computing instances [32]. The proposed technique employs the PB model to replicate each cloud task on primary and replica copies. A VM processes this primary task with the minimum level of utilization.

**Algorithm 1** BBRT Mechanism

```
input: $vmList$, $taskList$
foreach $c$ in $taskList$ do
    foreach $vm$ in $vmList$ do
        calculate $vm_u$
        if minimum($vm_u$) == $false$ then
            continue
        end if
        if $cap(c) \leq cap(vm)$ then
            $schedule(vm, c)$
            $vmList.update(vm)$
            break
        end if
    end for
    $c_r \leftarrow replicate(c)$
    foreach $vm$ in $vmList$ do
        if $isScheduled(vm, c) == true$ then
            continue
        end if
        if $cap(c_r) \leq cap(vm)$ then
            $schedule(vm, c_r)$
            $vmList.update(vm)$
            break
        end if
    end for
end for
```

In the BBRT mechanism, each task is replicated using the *replicate* function by copying the details of each task. Each repeated task is then assigned to a VM to be executed on the condition that the VM does not already host the primary copy of the task so that another copy of the task is running on a different VM. This condition is necessary in case a VM is lost to a failure event. Each VM has both CPU and RAM capacities to run cloud tasks. The *Cap* function ensures that the selected VM can accommodate the submitted task to be run on it, and the *schedule* method employed to assign $c_r$ to the selected VM. In this mechanism, submitted tasks can refer to any tasks submitted by a broker to a CSP. This mechanism can accept any number of tasks, but the number of unexecuted tasks can increase, increasing SLAV as shown in Eq. (9). Fig. 2 depicts the overall workflow of the system. It is worth mentioning that the broker can choose VMs from different CSPs when selecting a suitable VM.

However, the BBRT suffers from a performance issue regarding the timespan, as explained in Eq. (8). Replicating tasks allows each VM to be loaded with more tasks, and many tasks take more time to execute, which leads to a longer waiting time for other tasks. The replication technique in the BBRT mechanism thus increases the MST for most tasks. To mitigate this effect, we propose the BBRTplus to reduce the number of executing or to-be-executed tasks, as depicted in Algorithm 2. The BBRTplus can slightly increase the number of lost tasks compared to the BBRT to reduce the MST.

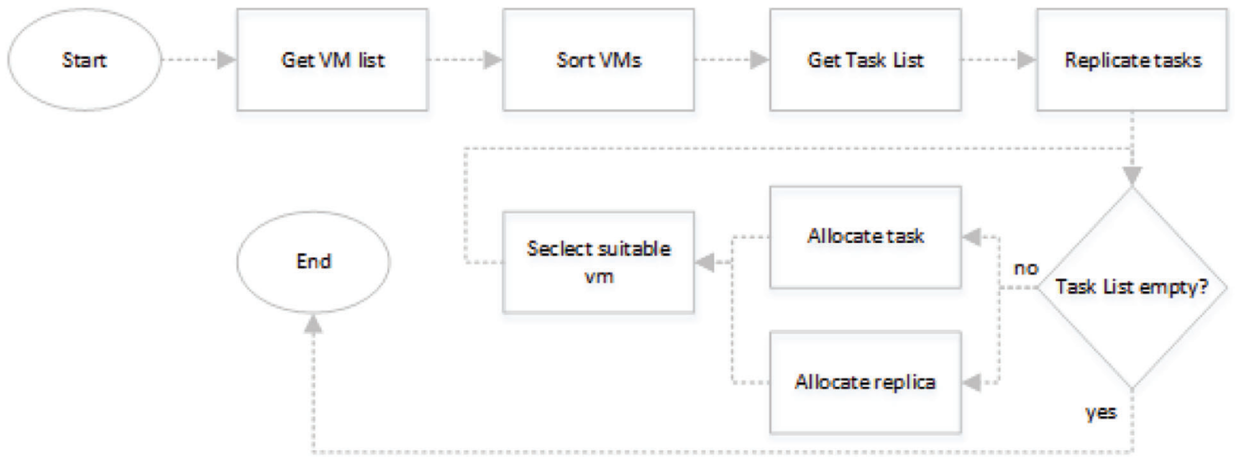

**Figure 2:** System workflow

**Algorithm 2** BBRTplus Mechanism

```
1: input: $c, taskWaitingList, vmList$
2: foreach $t$ in $taskWaitingList$ do
3:     if isReplica($t$, $c$) $==$ $true$ then
4:         $taskWaitingList.remove(t)$
5:         $vmList.update(taskWaitingList)$
6:         break
7:     end if
8: end for
```

The idea behind this extension is that when a cloud task is successfully executed, the mechanism seeks its replica to remove it, provided that it is still either being executed or will be executed (i.e., on the waiting list). This is because its replica is no longer needed if a cloud task is successfully executed. Notably, the behavior of the BBRTplus is the same regardless of whether the replica is executed first, as the primary task is removed since they both produce the same results. In short, if a copy is executed successfully, the other copy is removed regardless of whether it is a primary or a replica. This mechanism reduces the number of tasks on the waiting list, which can substantially reduce the MST of most tasks.

## 5 Evaluation

This section presents two experiments conducted to evaluate the BBRT and BBRTplus mechanisms. The first experiment demonstrates how the BBRT mechanism can reduce the effect of failure events on cloud systems. The second demonstrates how the BBRTplus mechanism can affect the timespan factor compared to the BBRT mechanism.

### *5.1 Experiment Configurations*

The experiments were conducted using DesktopCloudSim [33], a simulation tool based on CloudSim [34], a well-known simulation tool for cloud computing systems. DesktopCloudSim enables the simulation of failure events in cloud systems, a feature that CloudSim lacks. Fig. 3 depicts how the experiment works. The tool is fed with cloud tasks, VM specifications, failure events, and node specifications. Cloud tasks can have various parameters, such as the length of tasks measured in MIs. The length is assigned randomly. VM specifications include each VM's number of running VMs, RAM, and CPU. The specifications of a data center include details about nodes in a cloud system, such as CPU and RAM bandwidth. Table 1 lists the experiment configurations. Each experiment was conducted 180 times for the private cloud system and another 180 times for the public cloud systems. The number of replicas used in every rune is one replica. Each run simulated a 24-h period of run time.

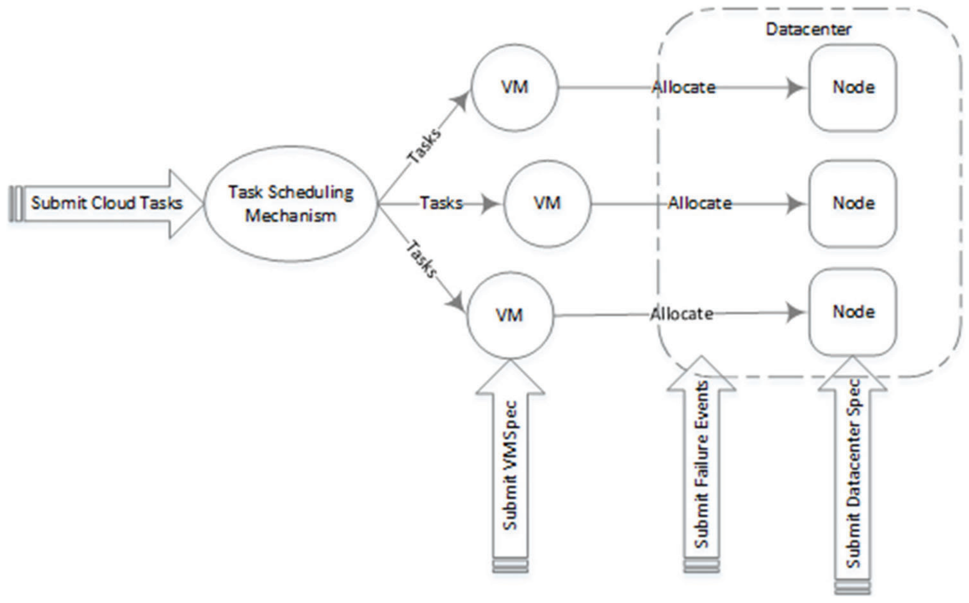

**Figure 3:** Experiment overview

**Table 1:** Experiment configurations

| Parameter | Value |
|---|---|
| Simulation duration | 24 h |
| Number of cloud tasks | 100 to 1000 |
| Length of cloud tasks | $1 \times 10^6$ to $21 \times 10^6$ MI |
| Number of VMs | 200 |
| VM CPU | 0.25, 0.5, 0.75 or 1 GHz |
| VM RAMM | 0.6, 0.8 or 1.7 GHz |
| Host CPU | 4 GHz |
| Host RAM | 8 GHz |
| Host storage | 1 TB |

Failure events can be submitted to nodes to determine which node fails, when it fails and when it restarts. Any node failure loses hosted VMs, and all cloud tasks running on a lost VM are forfeited as a result. Failure events in this study include any failure, from device crashes to network interruptions. These experiments use real-world datasets for failure events in cloud systems. The traces of failure events were retrieved from the failure trace archive (FTA) [35], which provides datasets of failure events that can be used to simulate failure events in a cloud system. Each experiment used two different traces: one simulating a private cloud system [36] and one from [37], which simulates a public cloud system. The main difference between them is that the number of failure events is lower in the former than in the latter.

### *5.2 Experiment I*

The BBRT mechanism was evaluated against several related mechanisms, which are EFDTS [28], LB [26], Best-Fit (BF) and First-Fit (FF) mechanisms. EFDTS was discussed in Section 3; it selects the most suitable VM for a submitted task based on the deadline. Then, it replicates tasks and allocates replicated tasks in a power consumption manner. LB, BF, and FF are typical scheduling mechanisms in traditional cloud middleware such as Eucalyptus, OpenNebula and OpenStack [38,39]. LB distributes tasks fairly to the available VMs to balance the load of tasks among VMs. BF searches for the most suitable VM that can host a particular task (i.e., it distributes tasks among VMs to increase the utilization of VMs). FF allocates tasks to the first available VM, provided that the VM can accommodate the task. However, LB, BF, and FF mechanisms do not employ fault-tolerance approaches. The BBRT outperformed the other mechanisms regarding the number of lost tasks, as depicted in Fig. 4, for both the private and public cloud datasets. The figure shows the average ratio of lost tasks for each number of submitted tasks. The BBRT mechanism achieved about 7% on average for private cloud systems and about 11% for public cloud systems. It achieved 20% for private and 26% for public cloud systems when the EFDTS mechanism was employed. Fig. 5 depicts how the mechanisms behaved in ten different runs; each run used tasks from 100 to 1000.

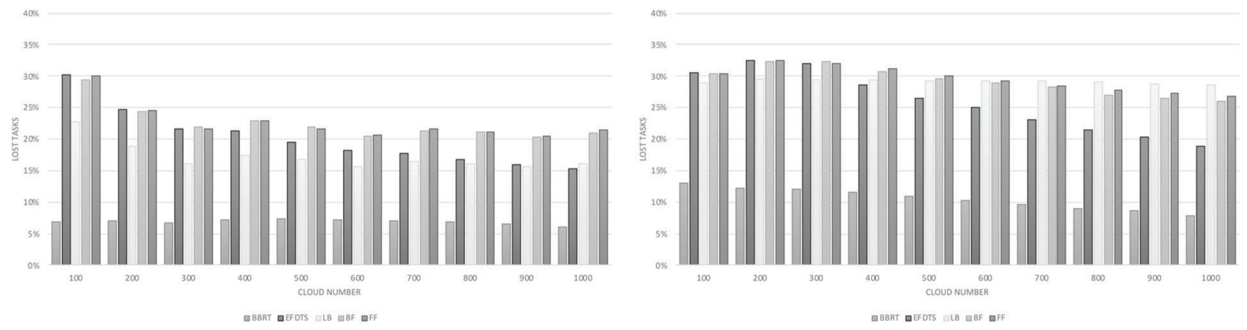

**Figure 4:** Lost tasks–private cloud (left) and public cloud (right) workloads

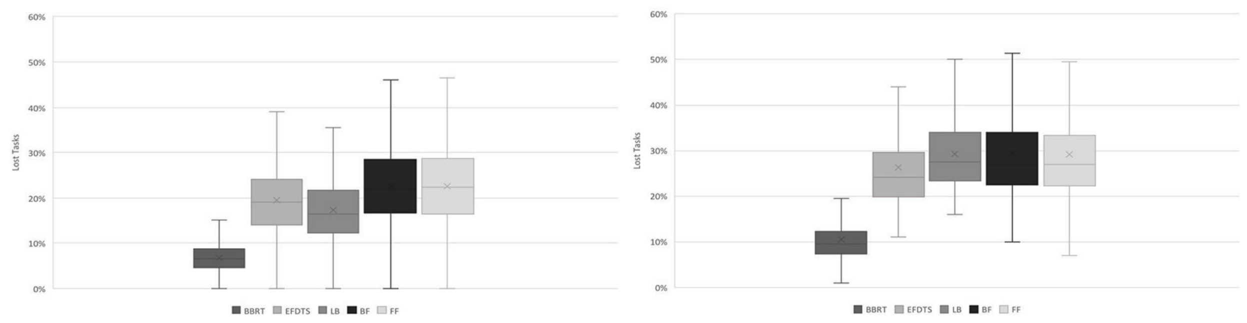

**Figure 5:** Range of lost tasks–private cloud (left) and public cloud (right) workloads

In terms of the number of successfully executed tasks, the BBRT outperformed its counterparts, as illustrated in Fig. 6. The average number of executed tasks for the BBRT is 438 in private cloud systems and 412 in public cloud systems. However, when the number of tasks reached 900 and 1,000, the LB managed to serve more than the BBRT. This is because the BBRT duplicates the number of tasks. However, the BBRT mechanism increased the MST of tasks compared to the other mechanisms, as shown in Fig. 7, because of its replication technique, as explained in Section 4.2. The average MST for the BBRT was about 85,000 s for both private and public cloud systems, whereas it was about 82,000 s when the EFDTS mechanism was employed. The BBRT increases the MST by about 4% compared with the EFDTS, which is an acceptable level in return for the reduced impact of lost tasks. In addition, the SLAV increased to 13% and 16% in private and public cloud systems, respectively, when the BBRT mechanism was employed. Tables 2 and 3 show the summary of results for private cloud and public cloud systems.

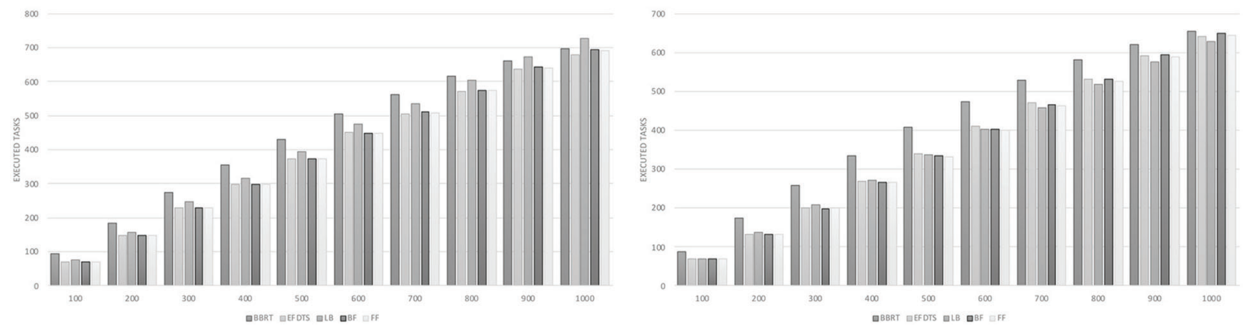

**Figure 6:** Number of executed tasks–private cloud (left) and public cloud (right) workloads

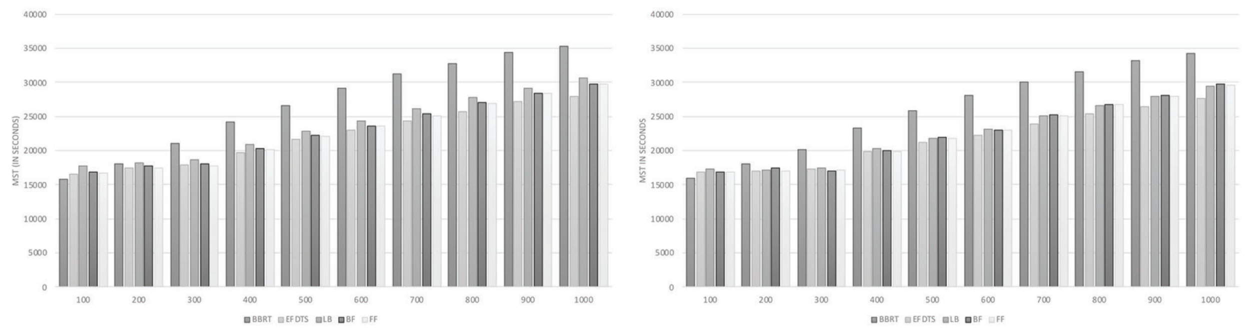

**Figure 7:** Makespan time–private cloud (left) and public cloud (right) workloads

### 5.3 *Experiment II*

The previous section demonstrated that the BBRT outperformed other mechanisms regarding the number of lost tasks but only at the expense of MST. This section evaluates the BBRTplus against the BBRT to study the impact of the BBRTplus mechanism on reducing the MST. The BBRTplus was applied to mitigate this penalty. The experiment was conducted under the same settings and datasets as Section 5.1.

**Table 2:** Private cloud results summary

| Parameter | BBRT | EFDTS | LB | BF | FF |
|---|---|---|---|---|---|
| Minimum task lost (%) | 0% | 0% | 0% | 0% | 0% |
| Maximum task lost (%) | 28% | 68% | 49% | 63% | 65% |
| Average task lost (%) | 7% | 20% | 17% | 22% | 23% |
| Minimum MST (seconds) | 720 | 720 | 720 | 720 | 720 |
| Maximum MST (seconds) | 85,440 | 82,800 | 86,397 | 86,370 | 86,383 |
| Average MST (seconds) | 26,822 | 22,140 | 23,633 | 22,914 | 22,804 |
| Minimum executed tasks (#) | 72 | 32 | 49 | 36 | 35 |
| Maximum executed tasks (#) | 813 | 854 | 836 | 852 | 842 |
| Average executed tasks (#) | 438 | 396 | 420 | 399 | 398 |
| Minimum SLAV (%) | 0% | 0% | 0% | 0% | 0% |
| Maximum SLAV (%) | 41% | 26% | 17% | 20% | 24% |
| Average SLAV (%) | 13% | 7% | 5% | 5% | 5% |
| Minimum failure event (%) | 0% | 0% | 0% | 0% | 0% |
| Maximum failure event (%) | 13% | 13% | 13% | 13% | 13% |
| Average failure event (%) | 5% | 5% | 5% | 5% | 5% |

**Table 3:** Public cloud results summary

| Parameter | BBRT | EFDTS | LB | BF | FF |
|---|---|---|---|---|---|
| Minimum task lost (%) | 1% | 11% | 16% | 10% | 7% |
| Maximum task lost (%) | 98% | 99% | 98% | 99% | 99% |
| Average task lost (%) | 11% | 26% | 29% | 29% | 29% |
| Minimum MST (seconds) | 720 | 720 | 720 | 720 | 720 |
| Maximum MST (seconds) | 84,496 | 82,800 | 86,381 | 86,395 | 86,395 |
| Average MST (seconds) | 26,822 | 22,140 | 23,633 | 22,914 | 22,804 |
| Minimum executed tasks (#) | 2 | 1 | 0 | 0 | 0 |
| Maximum executed tasks (#) | 769 | 805 | 862 | 834 | 837 |
| Average executed tasks (#) | 412 | 365 | 360 | 364 | 362 |
| Minimum SLAV (%) | 0% | 0% | 0% | 0% | 0% |
| Maximum SLAV (%) | 97% | 50% | 17% | 17% | 16% |
| Average SLAV (%) | 16% | 16% | 16% | 16% | 16% |
| Minimum failure event (%) | 16% | 16% | 16% | 16% | 16% |
| Maximum failure event (%) | 44% | 44% | 44% | 44% | 44% |
| Average failure event (%) | 23% | 23% | 23% | 23% | 23% |

Table 4 illustrates the results of both mechanisms with and without the BBRTplus feature. Regarding the cloud tasks lost, the BBRTplus mechanism yielded similar results compared with the BBRT mechanism. However, the BBRTplus outperformed the BBRT in terms of MST in both private and public cloud systems, reducing the total time on average by about 3%, from about 27,000 to about 26,000 s. The public cloud also reduced the MST by about 700 s; on average, the MST was about 26,000 s for the BBRT and 25,500 s for the BBRTplus, a reduction of about 2%. The BBRTplus mechanism yielded an increase in terms of SLAV by about 6% for private clouds and about 7% for public clouds compared to the BBRT mechanism.

**Table 4:** BBRTplus results summary

| Parameter | BBRT | | BBRTplus | |
|---|---|---|---|---|
| | Private cloud | Public cloud | Private cloud | Public cloud |
| Minimum task lost (%) | 0% | 0% | 0% | 1% |
| Maximum task lost (%) | 28% | 33% | 98% | 99% |
| Average task lost (%) | 6% | 11% | 7% | 11% |
| Minimum MST (seconds) | 720 | 720 | 720 | 720 |
| Maximum MST (seconds) | 84,960 | 85,440 | 86,381 | 85,680 |
| Average MST (seconds) | 26,822 | 26,162 | 26,043 | 25,720 |
| Minimum executed tasks (#) | 0 | 2 | 0 | 1 |
| Maximum executed tasks (#) | 813 | 769 | 688 | 658 |
| Average executed tasks (#) | 438 | 412 | 399 | 373 |
| Minimum SLAV (%) | 0% | 0% | 0% | 0% |
| Maximum SLAV (%) | 41% | 79% | 47% | 87% |
| Average SLAV (%) | 13% | 16% | 20% | 23% |
| Minimum failure event (%) | 0% | 16% | 0% | 16% |
| Maximum failure event (%) | 13% | 44% | 13% | 44% |
| Average failure event (%) | 5% | 23% | 5% | 23% |

## 6 Conclusion and Future Work

This paper presents BBRT, a novel task-scheduling mechanism that tackles the problem of failure events in cloud systems, which can significantly decrease the number of successfully executed cloud tasks. The proposed technique adopts a replication technique to replicate cloud tasks before submitting them to the CSP, and it employs the BBRTplus to reduce the MST of each task. Our experiments showed that the BBRT mechanism could successfully reduce the impact of failure events by decreasing the number of lost tasks while reasonably increasing the MST of cloud tasks. The effectiveness of this mechanism is evaluated against several mechanisms from the literature. The BBRT is thus a broker/client mechanism that enables distributing cloud tasks to various CSPs simultaneously with high fault tolerance.

In the future, this work can be improved to be cost-aware because the proposed mechanism can handle various CSPs, which enables it to choose an optimum level of cost-reliability among available CSPs. Such an extension would introduce cost as a factor to consider before replicating tasks. MST can be effectively reduced by adding more VMs, but this raises the cost. However, many clients are willing to pay extra for their tasks, so it may be wise to develop a model that can use cheap CPS to create VMs dedicated to hosting replicated tasks to improve overall reliability.

**Funding Statement:** This project was supported by the Deanship of Scientific Research at Prince Sattam Bin Abdulaziz University under research Project No. 2018/01/9371.

**Conflicts of Interest:** The author declares that he has no conflicts of interest to report regarding the present study.

## References

[1] K. Bilal, S. U. R. Malik, O. Khalid, A. Hameed, E. Alvarez *et al.,* "A taxonomy and survey on green data center networks," *Future Generation Computer Systems*, vol. 36, pp. 189–208, 2014.

[2] A. Beloglazov and R. Buyya, "Energy efficient resource management in virtualized cloud data centers," in *10th IEEE/ACM Int. Conf. on Cluster, Cloud and Grid Computing*, Washington, DC, USA, IEEE, pp. 826–831, 2010.

[3] P. Mell and T. Grance, "The NIST definition of cloud computing," in *National Institute of Standards and Technology*, Gaithersburg, MD, NIST Spec. Publ., pp. 800, 2011.

[4] 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, May 2013.

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

[6] R. Jhawar, V. Piuri and M. Santambrogio, "A comprehensive conceptual system-level approach to fault tolerance in cloud computing," in *2012 IEEE Int. Systems Conf.*, Vancouver, BC, Canada, IEEE, pp. 1–5, Mar. 2012.

[7] M. A. Rodriguez and R. Buyya, "A taxonomy and survey on scheduling algorithms for scientific workflows in IaaS cloud computing environments," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 8, pp. e4041, 2017.

[8] K. Mills, J. Filliben and C. Dabrowski, "Comparing vm-placement algorithms for on-demand clouds," in *2011 IEEE Third Int. Conf. on Cloud Computing Technology Sci.*, Athens, Greece, IEEE, pp. 91–98, Nov. 2011.

[9] 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, May 2012.

[10] S. K. Garg, A. N. Toosi, S. K. Gopalaiyengar and R. Buyya, "SLA-based virtual machine management for heterogeneous workloads in a cloud datacenter," *Journal of Network and Computer Applications*, vol. 45, pp. 108–120, Oct. 2014.

[11] A. Alwabel, R. Walters and G. Wills, "Evaluation metrics for vm allocation mechanisms in desktop clouds," in *Proc. of the 2nd Int. Workshop on Emerging Software as a Service and Analytics*, Lisbon, Portugal, pp. 63–68, 2015.

[12] X. Li, X. Jiang, P. Garraghan and Z. Wu, "Holistic energy and failure aware workload scheduling in cloud datacenters," *Future Generation Computer Systems*, vol. 78, pp. 887–900, 2018.

[13] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 17, Jun. 2009.

[14] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 13, pp. 1397–1420, Sep. 2012.

[15] L. M. Zhang, K. Li and Y. -Q. Zhang, "Green task scheduling algorithms with speeds optimization on heterogeneous cloud servers," in *Proc.-2010 IEEE/ACM Int. Conf. on Green Computing and Communications & Int. Conf. on Cyber, Physical and Social Computing*, Hangzhou, China, IEEE, pp. 76–80, 2010.

[16] C. Colman-Meixner, C. Develder, M. Tornatore and B. Mukherjee, "A survey on resiliency techniques in cloud computing infrastructures and applications," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 2244–2281, 2016.

[17] M. Nazari Cheraghlou, A. Khadem-Zadeh and M. Haghparast, “A survey of fault tolerance architecture in cloud computing,” *Journal of Network and Computer Applications*, vol. 61, pp. 81–92, 2016.

[18] S. S. Manvi and G. Krishna Shyam, “Resource management for infrastructure as a service (iaas) in cloud computing: A survey,” *Journal of Network and Computer Applications*, vol. 41, no. 1, pp. 424–440, May 2014.

[19] Y. Yuan, X. Li, Q. Wang and X. Zhu, “Deadline division-based heuristic for cost optimization in workflow scheduling,” *Information Sciences*, vol. 179, no. 15, pp. 2562–2575, 2009.

[20] W. Chen, G. Xie, R. Li, Y. Bai, C. Fan *et al.,* “Efficient task scheduling for budget constrained parallel applications on heterogeneous cloud computing systems,” *Future Generation Computer Systems*, vol. 74, pp. 1–11, 2017.

[21] G. Wang and H. Yu, “Task scheduling algorithm based on improved min–min algorithm in cloud computing environment,” *Applied Mechanics and Materials*, vol. 303–306, pp. 2429–2432, 2013.

[22] B. A. Al-Maytami, P. Fan, A. Hussain, T. Baker and P. Liatsist, “A task scheduling algorithm with improved makespan based on prediction of tasks computation time algorithm for cloud computing,” *IEEE Access*, vol. 7, pp. 160916–160926, 2019.

[23] E. S. Alkayal, N. R. Jennings and M. F. Abulkhair, “Efficient task scheduling multi-objective particle swarm optimization in cloud computing,” in *2016 IEEE 41st Conf. on Local Computer Networks Workshops (LCN Workshops)*, Dubai, United Arab Emirates, IEEE, pp. 17–24, 2016.

[24] X. Chen, Y. Zhou, B. He and L. Lv, “Energy-efficiency fog computing resource allocation in cyber physical internet of things systems,” *IET Communications*, vol. 13, no. 13, pp. 2003–2011, 2019.

[25] P. Zhang and M. Zhou, “Dynamic cloud task scheduling based on a two-stage strategy,” *IEEE Transactions on Automation Science and Engineering*, vol. 15, no. 2, pp. 772–783, 2018.

[26] J. P. B. Mapetu, Z. Chen and L. Kong, “Low-time complexity and low-cost binary particle swarm optimization algorithm for task scheduling and load balancing in cloud computing,” *Applied Intelligence*, vol. 49, no. 9, pp. 3308–3330, 2019.

[27] S. Geng, D. Wu, P. Wang and X. Cai, “Many-objective cloud task scheduling,” *IEEE Access*, vol. 8, pp. 79079–79088, 2020.

[28] A. Marahatta, Y. Wang, F. Zhang, A. K. Sangaiah, S. K. S. Tyagi *et al.,* “Energy-aware fault-tolerant dynamic task scheduling scheme for virtualized cloud data centers,” *Mobile Networks and Applications*, vol. 24, no. 3, pp. 1063–1077, 2019.

[29] J. Dean and S. Ghemawat, “MapReduce: Simplified data processing on large clusters,” *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[30] 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, May 2012.

[31] S. Antony, S. Antony, A. S. A. Beegom and M. S. Rajasree, “Task scheduling algorithm with fault tolerance for cloud,” in *2012 Int. Conf. on Computing Sciences*, Phagwara, India, IEEE, pp. 180–182, 2012.

[32] S. Ghosh, R. Melhem and D. Mosse, “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, Mar. 1997.

[33] A. Alwabel, R. Walters and G. B. Wills, “Desktopcloudsim: Simulation of node failures in the cloud,” in *The Sixth Int. Conf. on Cloud Computing, GRIDs, and Virtualization CLOUD COMPUTING 2015*, Nice, France, 2015.

[34] R. Buyya, R. Ranjan and R. N. Calheiros, “Modeling and simulation of scalable cloud computing environments and the CloudSim toolkit: Challenges and opportunities,” in *2009 Int. Conf. on High Performance Computing & Simulation*, Leipzig, Germany, IEEE, pp. 1–11, Jun. 2009.

[35] B. Javadi, D. Kondo, A. Iosup and D. Epema, “The failure trace archive: Enabling the comparison of failure measurements and models of distributed systems,” *Journal of Parallel and Distributed Computing*, vol. 73, no. 8, pp. 1208–1223, Aug. 2013.

[36] D. Kondo and B. Javadi, “The failure trace archive: Enabling comparative analysis of failures in diverse distributed systems,” in *CCGRID: 2010 10th IEEE/ACM Int. Conf. on Cluster, Cloud and Grid Computing*, Melbourne, VIC, Australia, IEEE, pp. 398–407, 2010.

[37] P. Anderson, J. Cobb, E. Korpela, M. Lebofsky and D. Werthimer, "SETI@home an experiment in public-resource computing," *Communications of the ACM*, vol. 45, no. 11, pp. 56–61, 2002.

[38] R. W. Ahmad, A. Gani, S. H. Hamid, M. Shiraz, A. Yousafzai *et al.,* "A survey on virtual machine migration and server consolidation techniques for cloud data centers," *Journal of Network and Computer Applications*, vol. 52, pp. 11–25, 2015.

[39] M. Randles, D. Lamb and A. Taleb-Bendiab, "A comparative study into distributed load balancing algorithms for cloud computing," in *2010 IEEE 24th Int. Conf. on Advanced Information Networking and Applications Workshops*, Perth, WA, Australia, pp. 551–556, 2010.