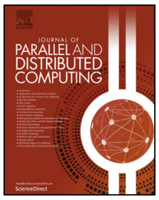
Journal of Parallel and Distributed Computing 158 (2021) 80–93

Contents lists available at ScienceDirect

Journal of Parallel and Distributed Computing

www.elsevier.com/locate/jpdc



Dynamic fault tolerant scheduling with response time minimization for multiple failures in cloud

Pushpanjali Gupta [a], Prasan Kumar Sahoo [a,b,*], Bharadwaj Veeravalli [c]

[a] Department of Computer Science and Information Engineering, Chang Gung University, Guishan, 33302, Taiwan
[b] Department of Neurology, Chang Gung Memorial Hospital, Linkou, 33305, Taiwan
[c] Department of Electrical and Computer Engineering, National University of Singapore, 119077, Singapore

ARTICLE INFO

Article history:
Received 11 February 2021
Received in revised form 22 May 2021
Accepted 26 July 2021
Available online 4 August 2021

Keywords:
Cloud computing
Scheduling
Fault tolerant
Resource utilization

ABSTRACT

With the increasing demand for large amount of computing resources, the cloud is widely used for executing large number of independent tasks. In order to successfully execute more tasks and maximize the revenues, the cloud service providers (CSPs) should provide reliable services, while maximizing the resource utilization. Providing better Quality of Service (QoS), while maximizing the resource utilization in the event of failures is a critical research issue which needs to be addressed. In this paper, an Elastic pull-based Dynamic Fault Tolerant (E-DFT) scheduling mechanism is designed for minimizing the response time while executing the backups during multiple failures of independent tasks. A basic core primary backup model is also used and integrated with the backup tasks overlapping (BTO) and backup tasks fusion (BTF) techniques to tolerate multiple simultaneous failures. Simulation results show that the proposed E-DFT scheduling can achieve better performance in terms of guarantee ratio and resource utilization over other existing scheduling algorithms.

1. Introduction

The emergence of cloud computing in the recent year has led to remarkable changes in the world of information technology with respect to the private and public cloud sectors [16]. With the proliferation of virtualization technology, the adoptions of cloud to process various data-intensive computations, CPU-intensive computations have become a trend in cloud computing. Running the jobs on virtual machines (VMs) has become an efficient solution for scalability, cost-efficiency and high resource utilization [33]. Moreover, the increasing demand for flexibility in obtaining and releasing the resources has resulted in the wider adoption of cloud. In order to meet such increasing demands of various applications, many CSPs have built large cloud data centers (CDCs). With the consequent increase in number of CSPs, the competition among different CSPs has increased [20]. With the ambition to maximize the revenue, the CSPs must provide better QoS to the users.

The cloud consisting of data centers is established by interconnecting large-scale physical machines (PMs) also referred to as hosts, which are accommodated with VMs rented to users for providing services using 'pay-per-usage' policy [12]. In addition, different methods for predicting the cost of VMs along with the workload and power consumption are also adopted by CSPs [22]. While the benefits are huge, there exist many probabilities of failures due to overloading of PMs, VMs, network congestions and hardware faults [24]. Moreover, these failures can be transient or permanent, thereby affecting the deadline-constrained applications scheduled on the VMs or PMs. As a result, there is increasing need to address these issues and provide users a reliable cloud to successfully execute the accepted tasks. In order to achieve fault tolerance, there are various techniques involved such as redundancy checking for error correction, tolerance policies involving replication and load balancing for avoiding failures [19]. For the real-time tasks, scheduling plays an important role in satisfying the users' requirement, while maximizing the resource utilization. Scheduling the tasks basically is meant to confirm the successful execution of tasks such that the deadlines of tasks are met even in the event of failures [25].

With the unique features such as the creation of multiple VMs by a single host, migration of VMs and dynamic resource scaling as per requirements, the cloud can provide benefit in two-fold; firstly, providing a reliable and secure cloud computing environment to the users and secondly, maximizing the revenues of the CSPs by increasing the number of accepted tasks and resource utilization [34]. Nevertheless, one of the most challenging requirements of

* Corresponding author at: Department of Computer Science and Information Engineering, Chang Gung University, Guishan, 33302, Taiwan.
E-mail addresses: d0521006@cgu.edu.tw (P. Gupta), pksahoo@mail.cgu.edu.tw (P.K. Sahoo), elebv@nus.edu.sg (B. Veeravalli).

https://doi.org/10.1016/j.jpdc.2021.07.019
0743-7315/© 2021 Elsevier Inc. All rights reserved.

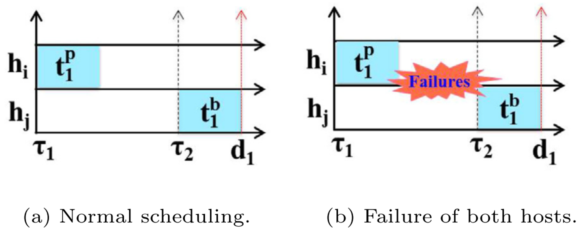


Fig. 1. A motivational example showing multiple failures.

users is that the accepted tasks should be successfully executed within the deadline. In order to satisfy this requirement, the CSPs should dynamically allocate the computing resources to the tasks in multiple PMs located in different CDCs lest the tasks should fail due to PM(s) failure. This is because the failure of a single task leads to the delay in execution of the task itself only. While on the other hand, the failure of a VM or PM results in the significant delay in the execution of all the allocated tasks.

Focusing on the probability of failure due overutilization and/or overrun of the resources, researchers have proposed different algorithms for preventing the failure occurrences, while considering the energy minimization [15], load balancing [11] and time-constraints. In addition to focusing on minimization of energy consumption and maximization of resource utilization, the authors in [10] have also proposed resource allocation framework to reduce the overheads caused due to migration. Similarly, authors in [18] have proposed the proactive fault tolerant approach to search for the optimal PM. The selected PM acts as the target for migrating the VMs from failing PMs. Irrespective of the type of failure, the scheduling performance and resource allocation is affected significantly. If the PMs are overloaded, the failure occurs due to overrun. On the contrary, if the PMs are underloaded, this allocation leads to the poor utilization of resources. Therefore, a fault-tolerant scheduling mechanism, which not only focuses on maximization of revenues for CSPs but also guarantees the successful execution of the accepted tasks should be developed.

1.1. Motivations

The main objective of the cloud computing is to provide platforms for different services, which can be easily used by different types of users with various applications. When a user's task is accepted by a CSP, it is the responsibility of the CSP to provide resources for successful execution of the task. However, due to various reasons such as overrun, overloading and network failures, the PMs and thereby the VMs present in the CDCs experience transient or permanent failure. The failures of PMs and VMs cause degradation in the QoS provided by the CSPs. Besides, sometimes such failure cannot be tackled well by the CSPs, leading to the failure of the tasks scheduled on the failed PMs and VMs. Although many research works have been carried out to handle the failure issues occurring in the cloud, very little attention has been paid to handling multiple failures that occur simultaneously. In addition, when there are multiple failures of VMs or PMs, multiple backups must be scheduled efficiently in different PMs with minimum resource consumption towards minimizing the overall response time of the tasks.

For example, as shown in Fig. 1, let us consider a scenario when a task t_1 arrives for execution. Usually, t_1^p, a primary copy of t_1 is scheduled in host h_i and the backup copy t_1^b is scheduled in host h_j. During execution of t_1^p, h_i may crash. Since, only one host is failed, backup copy of t_1 can be executed, leading to the execution of t_1^b at its scheduled place. This increases the waiting time of the task t_1. On the other hand, if both h_i and h_j crash simultaneously, both copies of t_1 are lost. This results in poor performance of cloud. One possible solution for providing better services is to create multiple copies of a task and schedule them in different hosts [14]. Though this can reduce the probability of loss of all copies, this can in turn increase the resource consumption. However, the resource consumption can be minimized as a whole, which gives benefit to both the users and CSPs when multiple copies of different tasks can be overlapped and fused together. In addition, when a failure of task is encountered, if one of the multiple backup copies of task can be executed at nearly next instance of failure time instead of waiting until the Latest Possible Start (LPS) of the failed task, there is two-fold benefit for the CSP. These are, firstly the impact of another failure within LPS time of the task can be avoided and secondly, the resources occupied by the task can be released when the task is already executed. Following this idea, in this paper, we attempt to design an Elastic pull-based Dynamic Fault Tolerant (E-DFT) scheduling mechanism.

1.2. Main contributions

Consequently, in order to improve the performance in terms of resource utilization and minimizing the response time of tasks, the following contributions are made in this paper:

1. An overlapping technique referred to as backup tasks overlapping (BTO) is designed and evaluated, which extends the traditional primary backup model in cloud to minimize the storage space required for the backup copies of tasks. This model helps in maximizing the resource utilization using fault tolerant scheduling mechanism.
2. A backup tasks fusion technique is designed and evaluated to integrate multiple tasks as a single task thus minimizing the switching time between tasks allocated to same host and also maximizing resource utilization.
3. A pull-based fault tolerant mechanism is developed and evaluated, which can pull the backup copies of task(s) for faster response time with a small delay in response. Using this pull-based technique, the response time of the backup execution could be minimized and the guarantee ratio could be maximized leading to maximization of the resource utilization and revenue.
4. The E-DFT scheduling can tolerate multiple number of failures that occur at a particular instance of time, considering energy and load balancing models to reduce the number of failures due to overloading and overrun
5. The proposed E-DFT achieves better guarantee ratio and resource utilization as compared to other protocols even in the event of multiple failures occurring simultaneously.

The remainder of this paper is organized as follows. Section 2 presents review of the related works. Section 3 presents the system model followed by the proposed protocol in Section 4. Simulation results and comparisons of the proposed protocol with other protocols are presented in Section 5. The concluding remarks are made in Section 6.

2. Related works

Scheduling of tasks in the form of jobs and requests has been extensively studied over the past few years [3]. In [5], the authors proposed the VM provisioning algorithm for time sensitive requests and workflow considering the deadline. The tasks are scheduled considering the priority and types of VM instances. In [32], authors have proposed scheduling mechanisms of delay bounded tasks in hybrid cloud. Focusing on cost minimization as main goal, the tasks are scheduled in FCFS, depending upon the workload the tasks are scheduled in private and public cloud. The authors in [31] proposed an algorithm to maximize the tasks' acceptance rate and the throughput of a private cloud. When exceeding capacity, the private cloud outsources some tasks to public cloud considering the deadline be met. Nevertheless, the time taken to migrate the tasks from private to public is considered negligible, a very sensitive assumption in case of delay bounded tasks. Similarly, the authors in [17] have proposed an approach to increase the resource utilization by accepting more jobs within a given time. The launching time of accepted jobs is postponed to the time until which the deadline can be satisfied. However, in cloud the traffic rate fluctuates, which might cause failure due to postponing the application. Thus, there is possibility that the job might miss the deadline. When considering the failure probability of resources, the above proposed mechanisms might miss the deadline of workflows and tasks leading to performance degradation.

On the other hand, if failure occurs, the performance of the cloud can be maintained by using resubmission, particle swarm optimization based strategies [30] and replication [21] techniques. In resubmission, the task after failure in one resource is re-executed on other normal computing resources. Some systems use replication techniques to execute several copies of a task for supporting fault tolerance while guaranteeing tasks' execution before their deadlines. For instance in [27], authors have proposed the mixed integer programming based scheduling of direct acyclic graphs with optimal duplication strategy on homogeneous multi-processor system. However, such systems endure relatively large resource consumption, and might not be applicable for large cloud computing scenario Similarly, in [8] the authors have proposed proactive and reactive scheduling mechanism for the aperiodic tasks considering uncertain cloud environment while guaranteeing QoS by minimizing the energy consumption. On the contrary, in [1] the authors proposed fault tolerance aware scheduling for tasks based on dynamic clustering league championship algorithm. The proposed fault aware techniques helps the scheduler be aware of resources and failures in the environment, which results in significant reduction in failures.

In order to reduce the consumption of resources for backup copies, a technique for overlapping the backups is developed for fault tolerant scheduling in heterogeneous systems. In [35], focusing on the overlapping technique and considering dynamic scheduling, authors have developed two dynamic algorithms for the scheduling of backups of independent and dependent tasks in grids. Although the proposed mechanism considers the failure of nodes in grids consisting of clusters, when considering cloud, the strategy might not be applicable, due to geo-distributed locations, bandwidth and cost of replication to be considered during the selection of data centers for replication. Considering the maximization of resource utilization, the authors in [29] addressed the fault tolerant scheduling taking into account the elasticity of VM migration and virtualization technology of cloud. On the other hand, in [2], considering the checkpoint concept a reactive fault tolerance technique is proposed for distributed environment. Using the flexible checkpoints, the fault detector module maintains record of the allocated virtual machines and prompts the scheduler to reschedule the applications in another VM in case of failure of the originally assigned VMs. Although extensive studies have been carried out in both static and dynamic scheduling of tasks, requests, workflows in private, public and hybrid cloud, very few works focus on handling multiple failures occurring in the cloud [9]. In addition, when multiple failures occur in the cloud, all the tasks are to be executed through their backup copies. However, this might not be achieved when only one backup copy of the task is made, which also fails along with primary copy of task. Besides, the minimization of response time in case of multiple failures while considering maximization of resource utilization of the CSPs is not studied. Therefore, considering the issues such as minimization of the response time while handling multiple failures and maximization of resource utilization, E-DFT has been proposed in this study.

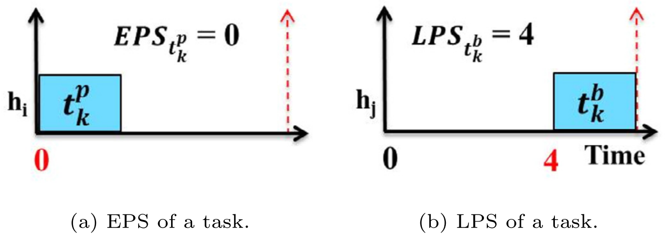


Fig. 2. Earliest possible start time and latest possible start time of a task.

3. System model

In this section the architecture of the proposed scheduling and fault tolerant structure is described, which consists of PMs, VMs distributed over the cloud. The PM or the host is referred to as h_i that consists of heterogeneous resources C_i, M_i and S_i to denote the total amount of CPUs, memory, and storage of the host h_i, respectively. Using these resources, the host h_i can host multiple VMs with same or different configurations of CPU, memory or storage resources, which is clearly elaborated in Section 3.2. These VMs are responsible for performing the execution of accepted tasks. When a task is submitted to cloud, it is expected that the task will be successfully executed within the deadline with minimum resources based on the size and deadline of task. The user's primary requirement would be successful execution of the submitted task satisfying the deadline constraint while the CSP's requirement is to maximize the revenue while satisfying user's requirement. In order to address this conflicting objective, an attempt is taken to design an Elastic pull-based Dynamic Fault Tolerant scheduling mechanism referred to as E-DFT scheduling. Consequently, the preliminaries about different terms used in the proposed protocol are described. Besides, different models such as cloud model, task allocation model, fault model, etc., are discussed. Furthermore, the types of tasks to be submitted with their attributes are also elucidated. Along with these, the notations and descriptions of frequently used symbols are summarized in a quick reference Table 1.

3.1. Preliminaries

1. Earliest Possible Start (EPS)
   The Earliest Possible Start (EPS) of a task is defined as the soonest possible time at which a task can start its execution with least delay as shown in Fig. 2a. In this case, the primary copy of a task t_k can be started when all of the tasks ahead of it in the queue q have been scheduled. It is mathematically defined in Eq. (1).

$$EPS_{t_k^p} = \begin{Bmatrix} 0 & |q| = 0 \\ \sum_{a=1}^{|q|} e_{t_a}^{ij} & |q| > 0, a \neq k \end{Bmatrix} \tag{1}$$

   where $\sum_{a=1}^{|q|} e_{t_a}^{ij}$ is the summation of execution times for all tasks ahead of t_k^p in the queue scheduled in VM v_j, which is hosted by $h_i \epsilon H_{\check{a}}$.
2. Latest Possible Start (LPS)
   The Latest Possible Start (LPS) of a task is the maximum time to which the start of a task can be delayed after its failure in order to complete its execution without violating the deadline

Table 1
Notations and descriptions.

Notations	Descriptions
t_k	The kth task
h_i	The ith host
v_{ij}	The jth VM hosted in ith host
a_k	The arrival time of kth task
s_k	The size of kth task
d_k	The deadline of kth task
τ_{fail}	The time of failure
δ	A constant representing delay in response after failure(s) has occurred
t_k^p	The pth primary copy of kth task
t_k^b	The bth backup copy of kth task
$H_{\check{a}}$	The set of hosts containing all h_i in active mode
$H_{\hat{a}}$	The set of hosts containing all h_i in inactive mode
$\mathbb{H}$	The set of hosts containing all h_i, where $\mathbb{H} = H_{\check{a}} \cup H_{\hat{a}}$
$e^{ij}_{t_k^p}$	The execution time required by t_k^p when executed in v_{ij}
t_k^o	The oth overlapped backup copy of t_k
t_k^f	The fth fused backup copy of t_k
$e^{ij}_{t_k^o/f}$	The execution time required by oth overlapped backup copy or fth fused backup copy of t_k when executed in v_{ij}
$EPS_{t_k^p}$	The earliest possible start time for the pth primary copy of kth task
$LPS_{t_k^b}$	The latest possible start time for the bth backup copy of kth task
$LPS_{t_k^o}$	The latest possible start time for the oth overlapped backup copy of t_k
$LPS_{t_k^f}$	The latest possible start time for the fth fused backup copy of t_k
$ST_{t_k^b}$	The start time of bth backup copy of kth task
f_{t_k}	The fault tolerant time for kth task
$c_j(\tau)$	The available amount of CPU resources in jth VM at time instance τ
$m_j(\tau)$	The available amount of memory in jth VM at time instance τ
$s_j(\tau)$	The available amount of storage in jth VM at time instance τ
C_i	The total amount of CPU resources in ith host h_i
M_i	The total amount of memory in ith host h_i
S_i	The total amount of storage in ith host h_i
$\ddot{l}_k^p(\tau)$	The status of pth primary copy of kth task in v_{ij}
$l_{ij}(\tau)$	The load of v_{ij} at time instance τ
$\acute{l}_i(\tau)$	The load of h_i at time instance τ
$E^*(h_i)$	The total energy consumed by host h_i while executing both primary and backup copies of tasks

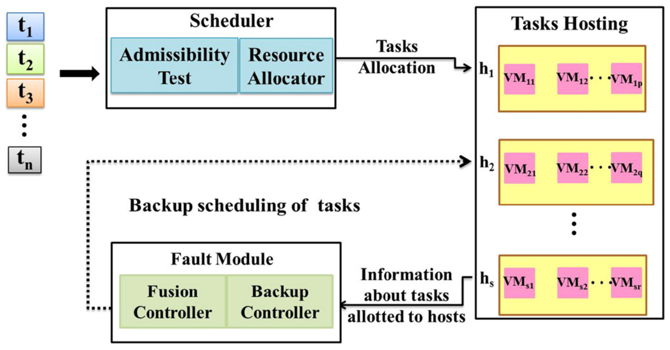


Fig. 3. System architecture of the proposed scheduling mechanism.

as shown in Fig. 2b. The LPS of a task takes into account the execution time of the backup copy t_k^b in the available VM v_{ij} as given in Eq. (2).

$$LPS_{t_k^b} = d_k - e^{ij}_{t_k^b} \tag{2}$$

3. Fault Tolerant Time (f_{t_k})
Fault tolerant time of a task t_k is defined as the time duration between the EPS and LPS of a task. Based on the value of f_{t_k}, the tasks are accepted by the scheduler for execution. This will be further discussed after the strategy used in this paper is introduced in later section.

3.2. Cloud system model

Let us consider that there exists a set of VMs represented as $V = \{v_1, v_2, ..., v_z\}$ consisting of z number of VMs. Each VM v_j is associated with three different tuples $c_j(\tau), m_j(\tau)$ and $s_j(\tau)$, representing the available resources CPUs, memory and storage of v_j at time τ, respectively. These VMs are hosted in the PMs with different configurations. Due to various factors such as the ongoing maintenance of the PMs or failures of the tasks at a particular instance of time, certain number of PMs may reside in the cloud which may be in active mode $H_{\check{a}}$ and certain number of PMs may be in inactive mode $H_{\hat{a}}$. Thus, the total number of PMs can be aggregated as a set $\mathbb{H} = H_{\check{a}} \cup H_{\hat{a}}$ including both active and inactive PMs. Since, the hosts $h_i \epsilon \mathbb{H}$ are heterogeneous, they can accommodate different number of VMs with varying configurations. Let n be number of tasks represented in the form of set $T = \{t_1, t_2, t_3, ..., t_n\}$. Each task t_k has certain attributes such as the arrival time a_k, size s_k and more importantly the deadline d_k. In order to maximize the utilization of the resources and to complete the execution within the deadline, these tasks are scheduled simultaneously in different VMs of same or different hosts.

3.3. Scheduler characteristics

In order to have an efficient scheduling mechanism, the scheduler performs an admissibility test and then only forwards the accepted tasks to the task allocation model, as shown in Fig. 3. At a particular time instance τ, the scheduler considers a set of tasks $T = \{t_1, t_2, t_3, ..., t_n\}$ with their respective attributes $\{a_k,\ s_k,$

$d_k\}$. Since, multiple tasks may arrive to the cloud at the same time, the tasks are stored in a queue based on the First Come First Serve (FCFS) principle. If two tasks arrive simultaneously, the tasks are further arranged on the basis of earliest deadline first. The size of a task represents the resource required by a tasks to complete the execution before deadline. For each task t_k, the time required ($\mathbb{T}_r$) to execute the task can be estimated as follows:

$$\mathbb{T}_r = s_k/p_j(\tau)$$

where, $p_j(\tau)$ is the processing speed of the fastest available VM at time τ. When $\mathbb{T}_r > d_{ks}$, the task t_{ks} cannot be accepted for completion within the deadline. Therefore, the scheduler performs the admissibility test based on the following Eq. (3) to avoid the acceptance of such type of tasks.

$$t_k = \begin{Bmatrix} Accept & if\ \ d_k > EPS_{t_k^p} + e_{t_k^p}^{ij} \\ Reject & otherwise \end{Bmatrix} \tag{3}$$

where, d_k is the deadline for task t_k, $EPS_{t_k^p}$ is the time taken for scheduling the primary copy of t_k, and $e_{t_k^p}^{ij}$ is the time taken to execute the t_k^p in v_{ij}. Using Eq. (3), the tasks are filtered out and the set containing the accepted tasks $\ddot{T}$ is forwarded to the task allocation model. It is to be noted that Eq. (3) is used for the scheduling of the primary copies of tasks. If the primary copies of the tasks are scheduled for the execution before the deadline, the scheduling of backup copies are certainly accommodated using either the passive backup or active backup procedure as illustrated in Fig. 4.

3.4. Task allocation model

In order to have successful execution of the tasks while tolerating multiple failures, multiple copies of the task t_k are made that comprise one primary and multiple backup copies. The primary copy t_k^p is scheduled on the suitable v_{ij} considering the status of p^{th} primary copy of k^{th} task in v_{ij} at time τ, which is denoted as $\ddot{l}_k^p(\tau)$ and is deduced in Eq. (4).

$$\ddot{l}_k^p(\tau) = \begin{Bmatrix} +1 & \text{Task is ongoing} \\ 0 & \text{No Task is executed} \\ -1 & \text{Task has successfully executed} \end{Bmatrix} \tag{4}$$

Based on the status of the task in v_{ij}, the load of each VM can be calculated as given in Eq. (5).

$$\breve{l}_{ij}(\tau) = \sum_{p=1}^{t_c} \ddot{l}_k^p(\tau) \tag{5}$$

where, $\breve{l}_{ij}(\tau)$ depicts the load of VM v_{ij} at time τ and is calculated as the summation of the loads of different currently scheduled tasks in the set T_c, $t_c \epsilon T_c$. Based on the above Eq. (5), the v_{ij} with least value of $\breve{l}_{ij}(\tau)$ is chosen first for the scheduling of t_k^p. The remaining multiple copies are referred to as backup copies of t_k. These copies of task are scheduled by the fusion controller and backup controller as discussed in Section 4.2.

3.5. Fault model

Considering the main goal of handling multiple failures with minimum response time, a fault tolerant model is described in this sub-section. After the primary copies of accepted tasks are scheduled onto the hosts, the information about the scheduled primary copies are sent from the task allocator to the backup controller. Now, the backup controller is responsible for scheduling the backup copies. The backup copies of t_k are to be scheduled onto the hosts other than the one hosting t_k^p. The host h_i chosen for storing the backups is the one that has either no primary scheduled or very few primaries scheduled and it can complete the execution of backup copy within the deadline with minimum cost. The failure can be transient or permanent leading to missing the deadline of the tasks and ultimately severe loss of revenue. In order to avoid this, it is assumed that the host h_i informs other hosts $h_g \epsilon \mathbb{H}$ before it is shutdown completely. Along with this, the host $h_i \epsilon \mathbb{H}$ and VMs fail due to various unpredictable factors such as overloading and higher temperature due to overrun. At a particular time instance τ_{fail}, there are multiple failures (say ω) of VMs and PMs. It is beyond the scope of this paper to design failures detection mechanisms. Hence, without loss of generality we assume that failures detection mechanisms exist to detect the failures of PMs. Therefore, there is no allocation of tasks among the failed PMs or VMs. When failures are detected [23], the information is sent to all healthy PMs containing the backups. This triggers the execution of backups of the corresponding failed tasks from that failure time τ_{fail}. On the other hand, when the primary copy of the task is successfully executed, the success message is broadcast to other PMs to free the resources occupied by the backup copies. As per our assumptions, the hosts that are considered for the scheduling of the backups would not schedule any primary copies. Therefore, it would not impact the execution of backups after failure at τ_{fail}.

Along with fault tolerant scheduling mechanism, this work aims to maximize the resource utilization. Therefore, the backup tasks overlapping (BTO) and backup tasks fusion (BTF) techniques are used here. The backups copies are scheduled based on the LPS time of task. In case of BTO technique, two tasks t_{k1} and t_{k2} can overlap their backups if $host(t_{k1}^b) = host(t_{k2}^b)$. However, in order to fuse the backup copies of t_{k1} and t_{k2}, the host of both t_{k1}^p and t_{k2}^p should be same and $ST_{t_{k2}^p} > ST_{t_{k1}^p}$. When two tasks are scheduled onto one host, they are combined together in terms of sharing resources and treated as one unit. We refer to the combination of tasks into a single unit as BTF technique. Thus, using BTF guarantees that the respective hosts for fused tasks are same and this property plays a vital role in our algorithm, which maximizes the resource utilization and minimizes the switching time between the tasks.

3.6. Energy model

It is to be noted that the probability of failure due to overrun is directly related to the energy consumed during the run time and increase in temperature of host h_i. If a particular host h_i is chosen as a suitable candidate for executing the tasks for a long time the host h_i may encounter a physical failure due to an increase in temperature. In order to avoid the failure due to overrun the *consumed energy* E^* must be calculated using Eq. (6). Consequently, the total energy consumed by a host h_i can be represented as the sum of energy consumed by the primary copies of tasks, backups of tasks if primary copies fail and active backup tasks as follows:

$$E^*(h_i) = \mathbb{P} * \left\{ \sum_{r=1}^{m} e_{t_k^p}^{ij} + \sum_{r=1}^{x} e_{t_k^o}^{ij} + \sum_{r=1}^{y} e_{t_k^f}^{ij} + \sum_{r=1}^{c} e_{per}^{ij}(t_k^b) \right\} \tag{6}$$

where, m represents total number of primary copies of m different tasks, x and y represent the total number of overlapped and fused copies of all backup tasks, respectively. Finally, c gives the total number of backup copies of tasks, which are executing concurrently with the primary copies of tasks. The variable $e_{t_k^p}^{ij}$ represents the execution time of the primary copy of the task, $e_{t_k^o}^{ij}$ and $e_{t_k^f}^{ij}$ show the execution time of the overlapped and fused backup copy of the task, respectively. The notation $e_{per}^{ij}(t_k^b)$ is the execution time of an active task executed in the host h_i. The execution time of primary copies, overlapped copies, and fused copies of different tasks are independent of each other. After multiplying the power consumed $\mathbb{P}$ with the total execution time of tasks, the energy consumed by the host is obtained. Furthermore, the total energy consumed by the system can be obtained by the summation of energy consumed by each host h_i that belongs to $\mathbb{H} = H_{\breve{a}} \cup H_{\hat{a}}$. In this work, it is always aimed to minimize the total energy consumed by the system by minimizing the active duration of the PMs. It is assumed that the total power consumption $\mathbb{P}$ can be found within the list of specification details of the physical host when it is used for the virtualization. Based on the calculated value of the total power, the backup controller will attempt to choose a host h_i with lower E^* for scheduling of the backup copies, since the higher E^* for h_i implies the higher probability of failure due to overrun.

3.7. Load balancing

In case of CDCs, the load of each host is usually different from the other. This is because each host h_i is heterogeneous in terms of different resources. Consequently, the number of VMs allocated to each host h_i also varies. In order to accommodate more number of tasks in VMs while avoiding the overloading, a load balancing scheme is applied to improve the performance of each host $h_i \epsilon \mathbb{H}$. Besides, v_j is assigned to host h_i taking into account the current load of the host h_i at time τ as presented in Eq. (7).

$$\acute{l}_i(\tau) = \sum_{j=1}^{|Q|} \breve{l}_j(\tau) \tag{7}$$

where, $\breve{l}_j$ refers to as the total load of v_j at time τ and Q represents the set containing all v_j associated with host h_i. When there are failures of primary copies of different tasks, backup copies of all failed tasks are activated in different hosts leading to uneven distribution of loads among the hosts $h_i \epsilon \mathbb{H}$. Therefore, there is the VM migration policy adapted to have load balancing, which is possible throughout the execution of $t_k^b \epsilon \ddot{T}$. This VM migration is carried out to allow parallel execution of backup copies of different tasks of the hosts, which cannot execute the backup of all failed tasks simultaneously due to resource constraint. This process is a tradeoff between the waiting time for the execution of the backup copy of a task t_k at $LPS_{t_k^{o/f}}$ in the current host h_i and the delay time α required for that task to migrate to h_g for executing the backups following the pull-based mechanism. However, care must be taken during the VM migration that the destination h_g is not overloaded. Therefore, a threshold value is set for the migration of VM from the host h_i to h_g. This is calculated by the mean of least loaded host ($\dot{l}_{h_{g1}}(\tau)$) and most loaded host ($x\dot{l}_{h_{g2}}(\tau)$) at a particular time τ as given in the following Eq. (8).

$$\Upsilon(\tau) = \forall_{\mathbb{H}}(x\dot{l}_{h_{g1}}(\tau) + \dot{l}_{h_{g2}}(\tau))/2 \tag{8}$$

After each migration of VM, the values of $\dot{l}_{h_{g1}}(\tau)$ and $x\dot{l}_{h_{g2}}(\tau)$ are updated. Besides, a host is considered suitable as long as the value of destination host satisfies the Eq. (8).

4. Elastic pull-based dynamic fault tolerant (E-DFT) scheduling

In order to maximize the resource utilization and minimize the response time for backups execution after ω number of simultaneous failures, an Elastic pull-based Dynamic Fault Tolerant scheduling mechanism referred to as E-DFT scheduling is proposed in this work.

Algorithm 1 Tasks Filtering.

```
Input: Set of arriving tasks $T$: $\{t_1,t_2,...,t_n\}$, and available VMs $V$: $\{v_1,v_2,...,v_z\}$.
Output: Set of accepted tasks $\ddot{T}$ : $\{\phi\}$.
1: while $t_k \epsilon T$ do
2:     if $\mathbb{T}_r > d_k$ then
3:         Task ought to be rejected;
4:     end if
5:     Perform the admissibility test of each $t_k \epsilon T$;
6:     $\ddot{T} = \{\ddot{T}\} \bigcup t_k$;
7: end while
```

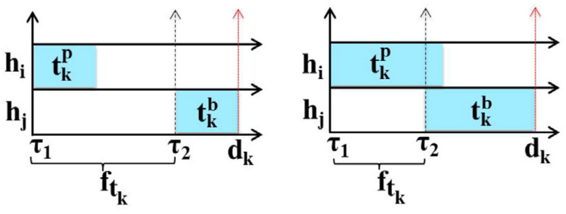
4.1. Task filtering and scheduling of primaries

Let us assume that multiple tasks arrive at different arrival times a_k, each with size s_k and deadline d_k. The arriving tasks are represented using a set T consisting of n number of tasks. Moreover, there exists some $v_j \epsilon V$ with processing speed $p_j(\tau)$, which may execute fewer or no tasks at time τ. For each task t_k, the total time required to execute the task is calculated as $\mathbb{T}_r$. If the $\mathbb{T}_r$ exceeds the given deadline of task, the task is rejected as described in the Algorithm 1 (Lines 1 - 4). Next, the admissibility test is performed using Eq. (3) to decide the scheduling of the primary considering the accepted tasks those are stored in a set $\ddot{T}$ (Lines 5 - 6).

Algorithm 2 Scheduling of Primaries.

```
Input: Set of accepted tasks $\ddot{T}$ as input from Algorithm 1
    $\mathbb{A} = \{(a_1, s_1, d_1), (a_2, s_2, d_2)...(a_k, s_k, d_k) \mid k \epsilon \ddot{T}\}$, set of arrival time,
    size and deadline of each task, available VMs $V$: $\{v_1,v_2,...,v_z\}$. $\mathbb{V}^* = \{\phi\}$, $\mathbb{H}^* =$
    $\{\phi\}$.
Output: Information of scheduled primaries.
1: for $v_j \epsilon V$ do
2:     Calculate the status of tasks in the $v_j$;
3:     Calculate the load of $v_j$;
4:     $\mathbb{V}^* \leftarrow$ Sort $V$ based on increasing load of $v_j$;
5: end for
6: for $h_i \epsilon H_{\breve{a}}$ do
7:     Calculate the Load of $h_i$;
8:     $\mathbb{H}^* \leftarrow$ Sort $H_{\breve{a}}$ based on increasing load of $h_i$;
9: end for
10: Map the top $v_j$ to top $h_i$ of $\mathbb{V}^*$, $\mathbb{H}^*$, respectively;
11: while $t_k \epsilon \ddot{T}$ do
12:     if $a_{k1} < a_{k2}$ then
13:         Schedule $t_{k1}$ at $EPS_{t_{k1}^p} < EPS_{t_{k2}^p}$;
14:     else if $a_{k1} = a_{k2}$ then
15:         if $d_{k1} < d_{k2}$ then
16:             Schedule $t_{k1}$ at $EPS_{t_{k1}^p} < EPS_{t_{k2}^p}$;
17:         end if
18:     end if
19:     Update the load of $v_j$ and $h_i$ and sort the queue $\mathbb{V}^*$ and $\mathbb{H}^*$, respectively;
20: end while
```

In the scheduling of primaries stated in Algorithm 2, the objective is to schedule the primary copies of different tasks $t_k \epsilon \ddot{T}$. As given in Algorithm 1, the filtered tasks form a new set $\ddot{T}$, which is forwarded to the task allocator for the scheduling of primaries. For the scheduling of tasks in different VMs, the status of different tasks present in $v_j \epsilon V$ is determined using Eq. (4). Consequently, the current load of the $v_j \epsilon V$ is calculated using Eq. (5) (Lines 1 - 5). The VMs are sorted in ascending order of loads to form a new set $\mathbb{V}^*$ after determining the loads of different VMs. Similarly, the loads of hosts, $h_i \epsilon H_{\breve{a}}$ at time τ are estimated using the Eq. (7) and sorted in ascending order, $\mathbb{H}^*$ (Lines 6 - 9). The sorting of PMs and VMs is followed by mapping the top $v_j \epsilon \mathbb{V}^*$ to top $h_i \epsilon \mathbb{H}^*$ (Line 10). This mapping allows the scheduling of $t_k \epsilon \ddot{T}$ based on their arrival time a_k. If two tasks t_{k1} and t_{k2} have same arrival time, i.e., $a_{k1} = a_{k2}$, the task with earlier deadline is scheduled first (Lines 12 - 18). After each iteration of scheduling, the status of task is updated in $v_j \epsilon \mathbb{V}^*$ and also the loads of v_j and h_i are updated. Finally, $\mathbb{V}^*$ and $\mathbb{H}^*$ are also sorted in ascending order (Line 19) and the process is repeated until $\ddot{T}$ becomes empty.



(a) Passive backup. (b) Active backup.

Fig. 4. Fault tolerant time.

Theorem 1. *In case of Fault Tolerant Scheduling, when $ST_{t_k^b} \le f_{t_k} \le LPS_{t_k^b}$ is satisfied, the failures can be tolerated.*

Proof. Let t_k be a task with arrival time $a_k = \tau_1$, size $= s_k$ and deadline $= d_k$. Supposing the waiting time $= 0$, t_k is scheduled at $EPS_{t_k^p} = \tau_1$ calculated using Eq. (1). Considering the processing speed of available $v_j \epsilon V$ the $LPS_{t_k^b} = \tau_2$ (say) is calculated as given in Eq. (2). The time duration between τ_1 and τ_2 is the f_{t_k}, as shown in Fig. 4. If any task fails after τ_2, the start time for execution of backup copy of task t_k, $ST_{t_k^b}$ becomes $\tau_2 + \delta$, which is $\tau_2 + \delta > LPS_{t_k^b}$. Eventually, the starting of execution of backup copy of task t_k at $ST_{t_k^b} = \tau_2 + \delta$ followed by the execution time $e_{t_k^b}^{ij}$ exceeds d_k. Therefore, the failures that occur until τ_2 time duration can only be tolerated by the system and exceed τ_2, which leads to miss the deadline. □

4.2. Scheduling of backups

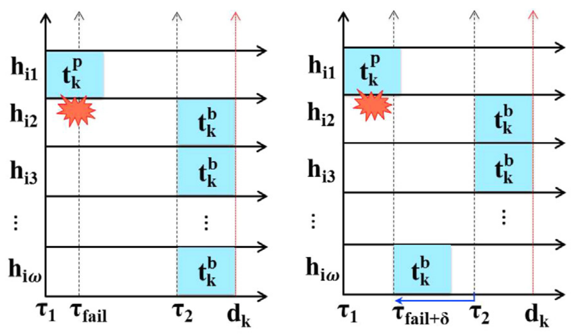
When a task is accepted for execution, the information is provided to the backup and fusion controller using the Algorithm 2 after the primary copy is scheduled in a host. The backups of a task must be scheduled on hosts $h_i \epsilon \mathbb{H}$, which do not contain any primary or very few primaries as described in Algorithm 3. In order to choose the suitable hosts for the scheduling of backup copies of tasks, the backup controller checks the energy consumed and load of the $h_i \epsilon \mathbb{H}$, which are determined as given in Lines 1-9. After sorting the hosts in ascending order of the energy and load of each PM, ω number of backup copies are created and scheduled at $LPS_{t_k^{o/f}}$ in the hosts for every task $t_k \epsilon \ddot{T}$ those have less loads. The energy, load and load threshold of h_i are updated using Eqs. (6), (7), and (8), respectively (Lines 2-6). Furthermore, the fusion controller checks if two tasks belong to the same hosts. Upon satisfying the required condition (Line 8), the fusion controller overlaps the backup copies of different tasks t_{k1}^o and t_{k2}^o to minimize the resource consumption. It is assumed that two tasks t_{k1} and t_{k2} can be merged and compressed to reduce the storage space required by them if they are in the same host as adopted in [4]. Besides, if two overlapped tasks have primary copy scheduled in same host: $h_i(t_{k1}^p) = h_i(t_{k2}^p)$, the overlapped copies of those tasks are fused together into a single task. This allows the minimization of switching time between the tasks belonging to the same host (Lines 10-13).

Now, when there are multiple failures at time τ_{fail}, the backup copies of tasks are pulled to $\tau_{fail+\delta}$ in a $v_j \epsilon h_i$. Since, there are both overlapped copies and fused copies of the tasks, the fused copies of the tasks are pulled if $h_i(t_{k1}^p) = h_i(t_{k2}^p)$. On the other hand, if $h_i(t_{k1}^p) \ne h_i(t_{k2}^p)$, just the required overlapped copies of tasks are pulled to $\tau_{fail+\delta}$ and are executed. The algorithm terminates after successful execution of all tasks $t_k \epsilon \ddot{T}$ (Lines 14-20).

Algorithm 3 Fault Tolerant Scheduling of Backups.

Input: Information of scheduled primary tasks as input from Algorithm 2.
Output: Applied fault tolerant mechanisms.

```
 1: Calculate the Energy consumed by $h_i$;
 2: while $t_k \epsilon \ddot{T}$ do
 3:     if $E^*(h_{i1}) < E^T$ & $\acute{l}_i(\tau) < \Upsilon(\tau)$ then
 4:         Create $\omega$ copies of $t_k^b$ and schedule at $LPS_{t_k^{o/f}}$ in $h_{i1} \epsilon \mathbb{H}^*$;
 5:         Update Energy, Load, Load Threshold of $h_{i1}$;
 6:     end if
 7: end while
 8: if $LPS_{t_{k2}^p} > LPS_{t_{k1}^p}$ then
 9:     Overlap copies of $t_{k1}^o, t_{k2}^o$ in $h_{i1} \epsilon \mathbb{H}^*$;
10:     if $h_i(t_{k1}^p) = h_i(t_{k2}^p)$ then
11:         Merge $t_{k1}^o, t_{k2}^o$ and form fused copies $t_{k1}^f, t_{k2}^f$ to be scheduled
           at $LPS_{t_{k1}^f}$ in $h_{i2} \epsilon mathbbH^*$;
12:     end if
13: end if
14: while (There are failures of tasks $t_{k1}^p$, $t_{k2}^p$ at time $\tau_{fail}$) do
15:     if $h_{i1}(t_{k1}^p) = h_{i2}(t_{k2}^p)$ then
16:         Pull the fused copy to $\tau_{fail+\delta}$ and execute it;
17:     else if $h_{i1}(t_{k1}^p) \ne h_{i2}(t_{k2}^p)$ then
18:         Pull the overlapped copy of $t_{k1}^o$ to $\tau_{fail+\delta}$ and execute it in $v_{j2} \epsilon V$;
19:         Pull the overlapped copy of $t_{k1}^o$ to $\tau_{fail+\delta}$ and execute it in $v_{j1} \epsilon V$;
20:     end if
21: end while
```



(a) Failure of h_1, h_2, h_3. (b) Recovering backups.

Fig. 5. Handling multiple failures, $1 \le \omega < |\mathbb{H}|$.

Theorem 2. *For any task t_k accepted for scheduling, the number of failures tolerated is $1 \le \omega < |\mathbb{H}|$, assuming all failures occur within f_{t_k}.*

Proof. Let t_k be a task with arrival time $a_k = \tau_1$, size $= s_k$ and deadline $= d_k$. Assuming the waiting time = 0, t_k is scheduled at its $EPS_{t_k^p} = \tau_1$. Considering the processing speed of the available $v_j \epsilon V$, the $LPS_{t_k^{o/f}} = \tau_2$, as shown in Fig. 5.

For achieving fault tolerant scheduling, multiple backup copies of t_k are placed in multiple hosts $\{h_1, h_2, \ldots, h_\omega\}$, $h_i \epsilon \mathbb{H}$ and (say $i = 0$). Let us consider there are multiple failures of hosts h_1, h_2,..., h_f, $f < \omega$ at time τ_{fail} such that $\tau_1 < \tau_{fail} \le \tau_2$. As seen from Fig. 5a, τ_{fail} lies within f_{t_k}. Therefore, a backup copy of t_k is pulled from τ_2 to $\tau_{fail+\delta} < \tau_2$ in host h_ω for achieving the fault tolerance, Fig. 5b. Hence, the system tolerates multiple failures ω, $\forall i \le \omega < |\mathbb{H}|$. □

Lemma 1. *Using E-DFT response time is minimized when failure occurs during f_{t_k}.*

Proof. Let t_k be a task with arrival time $a_k = \tau_1$, size $= s_k$ and deadline $= d_k$. Assuming the waiting time = 0, t_k is scheduled at its $EPS_{t_k^p} = \tau_1$. Considering the processing speed of available $v_j \epsilon V$, the $LPS_{t_k^{o/f}} = \tau_2$, as shown in Fig. 6.

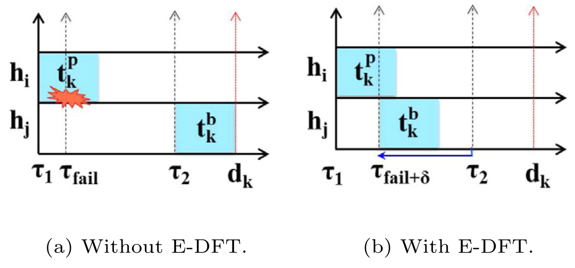


Fig. 6. Pull-based mechanism.

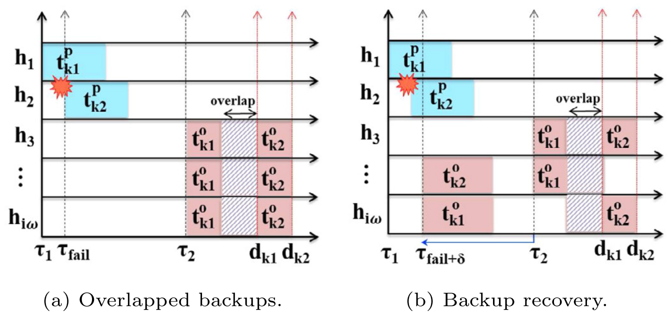


Fig. 7. Storage area minimization using BTO.

Let us consider t_k^p fails at time τ_{fail}. This failure information is sent to all healthy hosts containing the backups t_k^b. In general, the execution of backup copy starts at τ_2 such that the task is successfully completed at d_k. However, using the proposed E-DFT, the backup copy, t_k^b is pulled to $\tau_{fail+\delta}$, assuming δ as the response time. Hence, the execution of t_k^b is successful at time $\tau_{fail+\delta}+s_k/p_j < d_k$. Therefore, the proposed mechanism minimizes the response time efficiently. □

Lemma 2. *The storage resource for two tasks t_{k1} and t_{k2} with similar arrival times $a_{k1} \simeq a_{k2}$ can be minimized by overlapping them.*

Proof. Let t_{k1} and t_{k2} be two independent tasks with arrival times a_{k1}, a_{k2} and deadline d_{k1}, d_{k2}, respectively. Based on Theorem 2, the backups of t_{k1} and t_{k2} can be overlapped as shown in Fig. 7a with space requirement $\omega * (s_{k1} + s_{k1} - overlap(s_{k1k2}))$ for $h_{|\omega|} \epsilon |\mathbb{H}|$. Let us consider there are failures of h_1 and h_2 at τ_{fail}. In order to have faster response, the overlapping backup copies t_{k1}^o and t_{k2}^o are pulled to $\tau_{fail+\delta}$ as demonstrated in Fig. 7b, minimizing the response time for execution of backups. □

Lemma 3. *If the overlapping copies of the tasks t_{k1} and t_{k2} are in the same host, t_{k1}^o and t_{k2}^o can be fused as t_{k1}^f and t_{k2}^f, thus minimizing the switching time during execution of the backups of a single host h_i.*

Proof. Let t_{k1} and t_{k2} be two independent tasks with arrival time a_{k1}, a_{k2} and deadline d_{k1}, d_{k2}, respectively. The tasks belong to the same host. Therefore, both can have their backups fused in the form of t_{k1}^f and t_{k2}^f. If failure occurs at τ_{fail} as seen in Fig. 8a, some time is required by the system to recover. To overcome such transient failure, the fused copies of the tasks of same host is pulled to $\tau_{fail+\delta}$ and are executed, as shown in Fig. 8b, saving the switching time between the tasks of the same host. □

Theorem 3. *The time complexity for scheduling of primaries in E-DFT algorithm is $\mathcal{O}(m \log m)$, where m is total number of accepted tasks.*

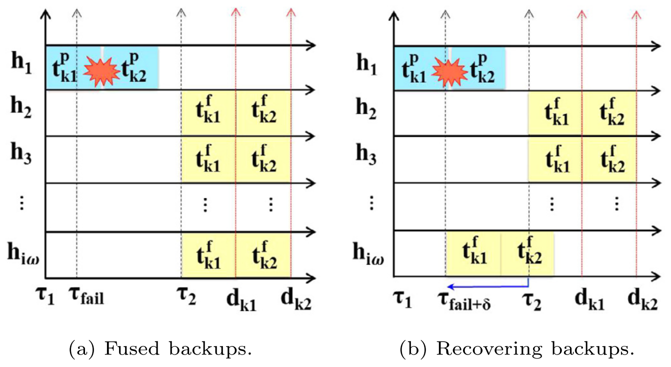


Fig. 8. Saving switching time using BTF.

Proof. In the Algorithm 1, the time taken for admissibility test of n tasks is $\mathcal{O}(n)$ and sorting m tasks using *Heapsort* is $\mathcal{O}(m \log m)$ time. The worst case time complexity is $\mathcal{O}(m \log m)$, when all tasks are accepted ($n = m$). The time taken by *Heapsort* for sorting VMs and PMs in Algorithm 2 (Lines 1-5) and PMs (Lines 6-9) is $\mathcal{O}(z \log z)$ and $\mathcal{O}(|H_{\breve{a}}| \log |H_{\breve{a}}|)$, respectively. The scheduling of primary copies of accepted tasks requires $\mathcal{O}(m)$ time. In worst case, one task is assigned to one VM ($m = z$) and number of VMs is always greater than number of PMs. Hence, the time complexity for scheduling of primaries in EDF algorithms is $\mathcal{O}(m \log m)$. □

Theorem 4. *The time complexity for scheduling and executing the backups in E-DFT algorithm is $\mathcal{O}(m)$, where m is total number of accepted tasks.*

Proof. The time complexity for scheduling the backups in Algorithm 3 is $\mathcal{O}(m)$ (Lines 2-6) and for executing the backups is $\mathcal{O}(\omega)$ (Lines 8-20), where ω is the number of failures. As a result, the worst case time complexity for scheduling and executing the backups in E-DFT algorithm is $\mathcal{O}(m)$, when all accepted tasks have failed. □

4.3. Example of E-DFT

In order to have a comprehensive idea about the working of the proposed E-DFT mechanism, an illustration is provided in this section. For the working, let us consider there are five tasks in a set $T = \{t_1, t_2, t_3, t_4, t_5\}$ with three different integers in the brackets, as shown in Fig. 9 representing a_k, s_k and d_k, respectively for each task. The d_k is the maximum time that can be allotted to complete the task t_k. Before accepting the tasks, $\mathbb{T}_r$ is used to check whether deadline for the task is suitable for acceptance. Considering $p_j(0) = 500$ of available v_j, it is found that $\mathbb{T}_r$ for t_1, t_2, t_3, t_4 is less than their deadlines 16, 20, 1.5, 7, respectively. However, the task t_5 has $d_5 = 20$ for $s_5 = 40000$, which causes the rejection of t_5, since $\mathbb{T}_r = 80$ is greater than deadline. Assuming the waiting time is equal to zero, the $EPS_{t_k^p}$ is also considered as zero, making the scheduling possible as soon as the task is accepted. Finally, there is a list $\ddot{T} = \{t_1, t_2, t_3, t_4\}$ to be scheduled using E-DFT for successful execution in case of multiple failures. In phase 1 of the E-DFT scheduling as shown in Fig. 9, the tasks are scheduled on the different hosts h_i considering the earliest possible start time of the tasks. Along with this, loads of the hosts are also checked to schedule the tasks. It is assumed that loads of hosts $\acute{l}_1(0) < \acute{l}_2(0) < \acute{l}_3(0)$ in the initial phase. Therefore, the primary copy t_1^p is scheduled in h_1 based on early possible start time. Both t_2 and t_3 arrive at time $a_2 = a_3 = 1$. However, $d_3 < d_2$, which allows t_3^p to be scheduled in h_2 followed by t_2^p in h_3 and so on. It can be observed that updated loads at $\tau = 5$ are $\acute{l}_2(5) < \acute{l}_1(5) < \acute{l}_3(5)$. On the other hand, at time $\tau = 22$, the energy

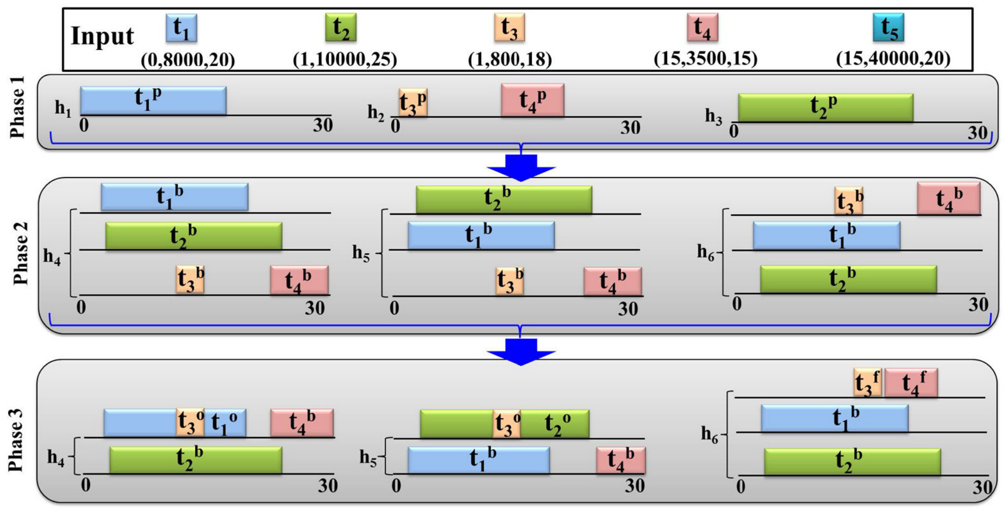


Fig. 9. Illustration of E-DFT.

consumed is $\acute{l}_2(22) > \acute{l}_3(22) > \acute{l}_1(22)$, since h_2 needs to execute t_4^p leading to overrun and chances of failure. In phase 2 of E-DFT, multiple backup copies of the tasks are scheduled onto different hosts based on the $LPS_{t_k^{o/f}}$ in order to tolerate multiple failures due to failures of some active hosts. The phase 3 of the E-DFT focuses on overlapping the backup copies of the tasks to minimize the resource requirements, such as t_1^o and t_3^o, t_2^o and t_3^o are overlapped in h_4 and h_5, respectively. Further, if a host fails, it will take certain time to recover. During that instance, all the backup copies of the tasks in that host are supposed to be executed. Therefore, the tasks with primary copies in the same host are fused together to minimize the switching time between the tasks of the same host. Based on the E-DFT scheduling in phase 1 and 3, if hosts h_1, h_2 and h_4 fail simultaneously, the backup copy of t_1 can be executed in the host h_5. Since, t_3 and t_4 are of the same host h_2, the fused copy of both tasks t_3^f and t_4^f can be used to have successful execution in host h_6 with faster response time. Thus, the proposed scheduling mechanism E-DFT handles multiple failures with faster response time.

5. Performance evaluation

In order to demonstrate the performance of the proposed protocol E-DFT, certain comparisons are made with three standard algorithms- Earliest Deadline First (EDF) [13], Minimum Completion Time (MCT) [26] and Deadline Monotonic scheduling Algorithm (A) [28]. In case of EDF algorithm, all the arriving tasks are arranged in a queue following ascending order of the deadline. EDF is a dynamic priority scheduling algorithm for independent tasks, whose main goal is to execute the tasks based on the earliest deadline. However, when system is overloaded, the deadline can be missed to much greater extent.

Secondly, MCT algorithm maps a new task on a VM that can complete the task at the earliest time while maintaining the finish time upper bound before the task's deadline. In addition, all the tasks are allocated to VMs upon their arrivals. Therefore, MCT is selected as the representative of classic greedy scheduling algorithms to demonstrate the performance improvements gained by the protocol when compared with classic greedy scheduling algorithms. The final comparison is made with DMA, a fixed priority scheduling algorithm, which assigns high priority to the tasks with earlier deadline. The goal of comparing the proposed protocol with EDF, MCT and DMA is to demonstrate that the proposed protocol can produce better scheduling performance than these algorithms even in the event of multiple failures, while minimizing the response time and maximizing the resource utilization.

5.1. Simulation setup

A widely recognized simulator in both industry and academia, CloudSim [7] is chosen as the building block of the considered simulating environment. Different settings of parameters are used in CloudSim such as the Million Instructions Per Second (MIPS) of each host is modeled with performance similar to 1000, 1500, 2000, 3000 and 5000 MIPS and the processing configuration of VMs are considered equivalent to 250, 500, 700, and 1000 MIPS.

With regards to the impact of different parameters of cloudlets (tasks) on the performance of clouds, the simulation is conducted based on cloudlets. The tasks are assumed to arrive at the cloud with size less than 1×10^5 MI. In addition, the deadlines of the tasks are distributed with range $[1 \times 10^2$ to $1 \times 10^3]$ seconds. The impact of each of the parameters is determined by repeating the experiments 30 times and average value is considered with standard error as discussed in [6]. All algorithms are compared using the following practically useful metrics. The first metric, *Guarantee Ratio(GR)* is defined as the ratio of number of tasks accepted by the scheduler to the tasks those are successfully executed within deadline. This metric depends on the availability of resources at the time of arrival of task along with the deadline of the task and response time due to failure of task. When the deadline for the task is very short, both the primary and backup copies of the task cannot be scheduled, resulting in the task being rejected.

The second metric, *Average Execution Time* represents the average time taken for the execution of a task. This is estimated by keeping some parameters constant such as keeping the deadline fixed or number of tasks fixed or generating fix number of failures. If there occurs no failure, the execution time of a task is the average duration from arrival to completion time of the primary copy of task, t_k^p. On the other hand, if failure occurs, the execution time is the average duration from arrival to completion time of the backup copy of the task. The third metric, *Active Time of PMs* shows the total active time of all the hosts in the cloud, thus implying at least one task is running in the system. The fourth metric, *Resource Utilization*, is defined as the ratio of the utilized resources to the total available resources of all the allocated hosts in the cloud. This represents the resource utilization of the system considering the case of both with and without failures.

The fifth metric, *Response Time* determines how quickly the task or the backup of task is re-executed in case of failures. The response time is the minimum time taken to start the execution of the backup copy of a task in case of failures of PMs and VMs. Another metric *Number of failures* N_f that is tolerated is defined as the percentage of tasks which are successfully re-executed when multiple failures of host and VMs occur. The N_f shows the effectiveness of a protocol in tolerating multiple failures and also serves as a Quality of Service parameter for the CSPs.

5.2. Simulation results

The CloudSim that supports the seamless modeling, simulation and experimentation of large-scale cloud infrastructures is chosen to simulate our cloud environment, which consists of hosts with abundant resources. It is assumed that initially no task is scheduled on any host though all are active. Upon arrival of the tasks in the form of cloudlets, the scheduler allocates resources to the tasks. Based on this assumption of resource scalability, the proposed E-DFT algorithm is evaluated by comparing it with EDF, MCT and DMA. In order to analyze the impact of one metric over the others, the value of one metric is changed keeping the rests constant. Therefore, the number of tasks is increased from $1,000$ to $50,000$, keeping the deadline constant as 1000 seconds and the performance of the EDF, MCT, DMA, and E-DFT are recorded with respect to different metrics for analyzing the performance impact of number of tasks.

The impact of number of arriving tasks over Guarantee Ratio is shown in Fig. 10a, where all algorithms maintain high Guarantee Ratio when number of tasks is less, which is due to highly scalable resource in cloud. Due to virtualization nature of cloud, multiple tasks can be executed concurrently and independently without missing deadlines. However, when the number of tasks increases, the guarantee ratio decreases. This is due to the start up time of PMs and VMs, which adds to delay in execution of tasks that ultimately leads to miss the deadline. With less number of arriving tasks, all four algorithms maintain high guarantee ratios; which are attributed to the availability of abundant resources in the cloud. The resource in cloud can be scaled-up or scaled-down as per the resource requirements. However, the guarantee ratios decrease gradually due to additional time required to create VMs as per the requirements as the number of tasks increases. The EDF and DMA always focus on prioritizing tasks based on deadline, and MCT focuses on selecting VMs with highest processing speed. These comparisons among tasks lead to miss the deadline as the number of tasks increases. On the other hand, the guarantee ratio of E-DFT is better than EDF, MCT, and DMA, since E-DFT only focuses on the execution time and deadline of the task.

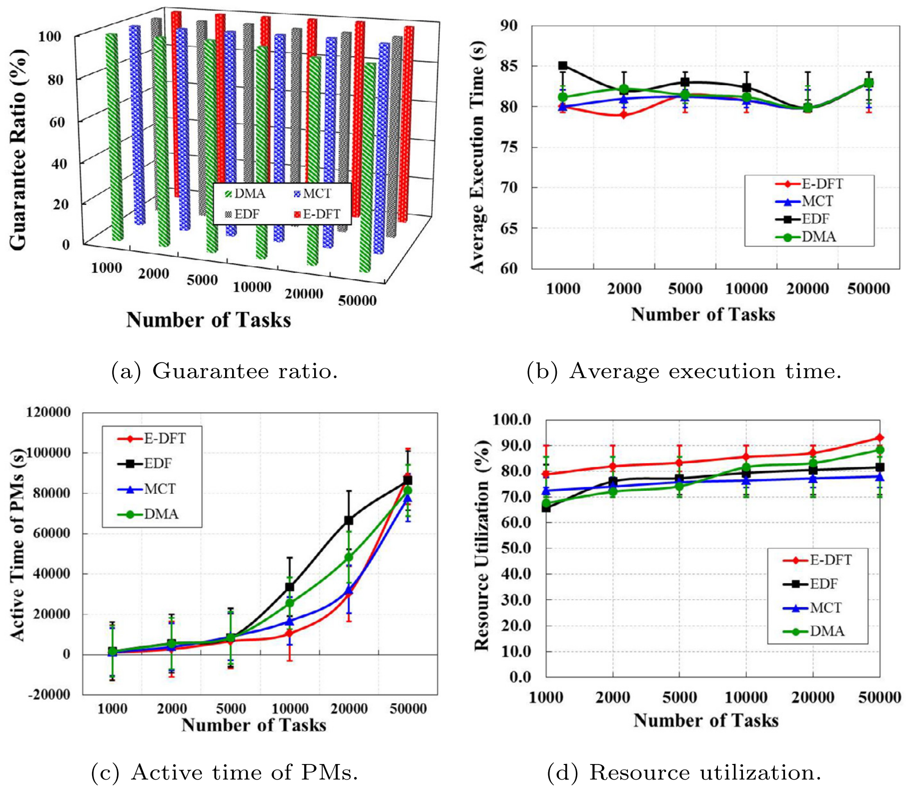
Similarly, Fig. 10b shows the average execution time of each task when number of tasks ranges from $1,000$ to $50,000$. As shown in the Fig. 10b, the average execution time of the tasks remains within the deadline, forming slight variation in the execution that is due to variable task sizes. Moreover, since the task size varies within 10^2 to $10^5 MI$, when multiple tasks are scheduled on VMs of different configurations, the average execution time reduces. When number of tasks increases, requirement for more number of best available resources also increases. In order to successfully execute the tasks within deadline, the PMs remain active for longer duration. This can be easily deduced from Fig. 10c, which shows the impact of number of tasks over active time of PMs. It can be clearly seen from the Fig. 10c that with the addition of tasks the active time of PMs increases, but E-DFT has better performance than EDF, MCT, and DMA. This is due to load distribution and energy model, which assigns the tasks to different PMs evenly based on its scheduling strategy. Finally, the percent of resources utilized by the different algorithms are determined in Fig. 10d. In case of cloud, huge amount of resources are available. In favor of achieving better guarantee ratio more resources are reserved for concurrent execution of tasks. As a result, this causes the PMs to be idle leading to underutilization of resources. On the contrary, when the resources are already reserved, based on the objectives of E-DFT, to resolve the multiple failures of PMs and VMs, scheduling of multiple backups of each task are performed on these underutilized PMs, which is clearly visible in the Fig. 10d. Since every CSP wants to increase the QoS, in case of E-DFT, the idle resources are used to create backups of the tasks leading to maximization of resource utilization, enhancing it further to BTO and BTF for maximizing revenue.

Along with the number of tasks, the deadline has also much impact on the performance of various algorithms, which is justified by performing the following simulation. The base deadline is varied from 1×10^2 to 1×10^3 seconds. Here, the size of the tasks vary from 10^2 to $10^5 MI$ and number of tasks remain constant as 20000. As represented in Fig. 11a, the Guarantee Ratios of all the algorithms increase gradually with extended deadline. This is because when the deadline is extended, more number of resources can be added and more number of tasks can be accepted for execution. Furthermore, when the deadline becomes more flexible, the Guarantee Ratio reaches almost 100% due to scalable resources of Cloud. For all the algorithms, variability in the performance of average execution time of the tasks is observed as shown in Fig. 11b, when considering various deadlines. This is primarily due to the fact that tasks those are accepted for execution are variable in size with variable deadline. Moreover, when deadline is relaxed, large number of VMs are launched, which can be used for executing large number of tasks of variable sizes and priorities. However, the execution time of MCT increases with increase in deadline, this is because it accepts tasks with longer completion time, which in turn increases the average execution time.

In Fig. 11c, the performance impact of deadline on the active time of PMs is shown. When deadline is extended, more number of tasks are accepted and so the active time of PMs increases. However, E-DFT outperforms other algorithms by applying load balancing and energy consumption model, while distributing the tasks and the backups among all the hosts instead of scaling up the resources. Finally, the impact of deadline over resource utilization is shown in Fig. 11d. When the deadline is short, less number of tasks are accepted but those accepted must be executed successfully within the deadline. This creates start up of more number of PMs for concurrent execution, leading to lower utilization of resources. On the other hand, when the deadline is extended, huge number of tasks are accepted satisfying the admissibility test and executed without increasing number of PMs, thus leading to maximization of resource utilization.

E-DFT mainly focuses on the minimization of response time while focusing on the resource utilization, when multiple failures occur. In order to show the efficiency of the proposed protocol, the failure handling experiment is performed using random failure generators for checking the impact of failure on different metrics. In E-DFT protocol, when multiple tasks t_{k1} and t_{k2} are accepted for execution along with one primary copy of each task t^p_{k1} and t^p_{k2}, multiple backup copies are scheduled as t^b_{k1} and t^b_{k2}, which can be further overlapped with other tasks based on the condition mentioned in Section 4.2 and Algorithm 3. It is implemented by using the custom code for reducing the size of two backup tasks by 0.15% for achieving the backup tasks overlapping (BTO), if backup tasks are scheduled in the same host. Besides, two overlapped tasks are merged by adding their sizes to form the fused backup tasks for achieving the backup task fusion (BTF), if primaries of both overlapped tasks are scheduled in the same host.

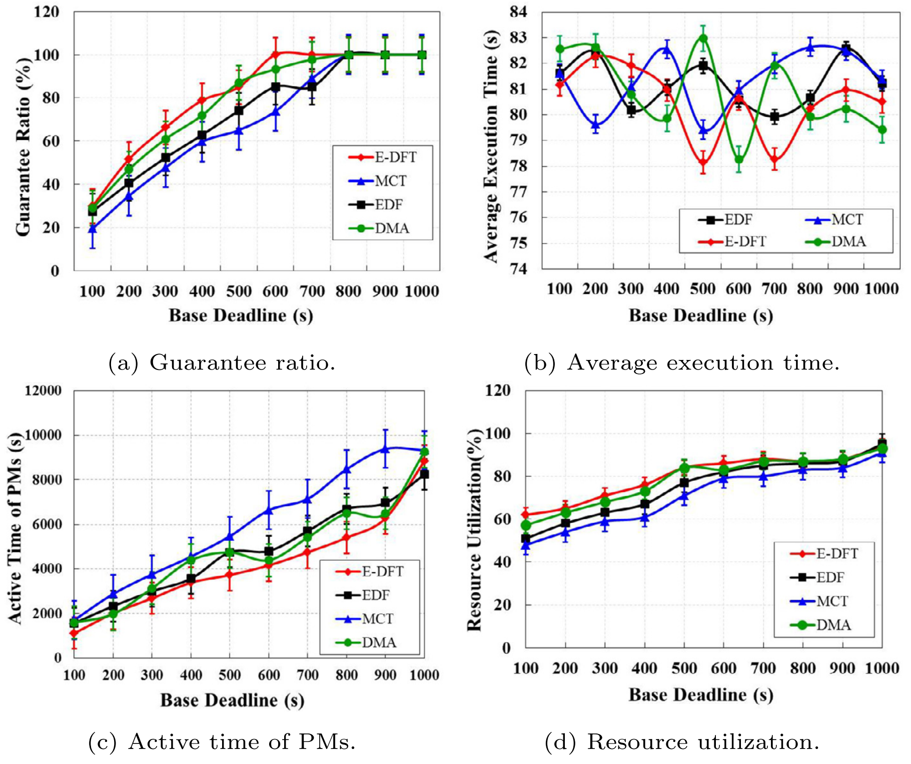
Keeping number of tasks constant at 10000, the number of failures N_f in the range of 100 *to* 500 are generated, which could lead to the simultaneous failures of VMs and PMs. As shown in Fig. 12a, Guarantee Ratio is affected by the number of failures.



(a) Guarantee ratio. (b) Average execution time.

(c) Active time of PMs. (d) Resource utilization.

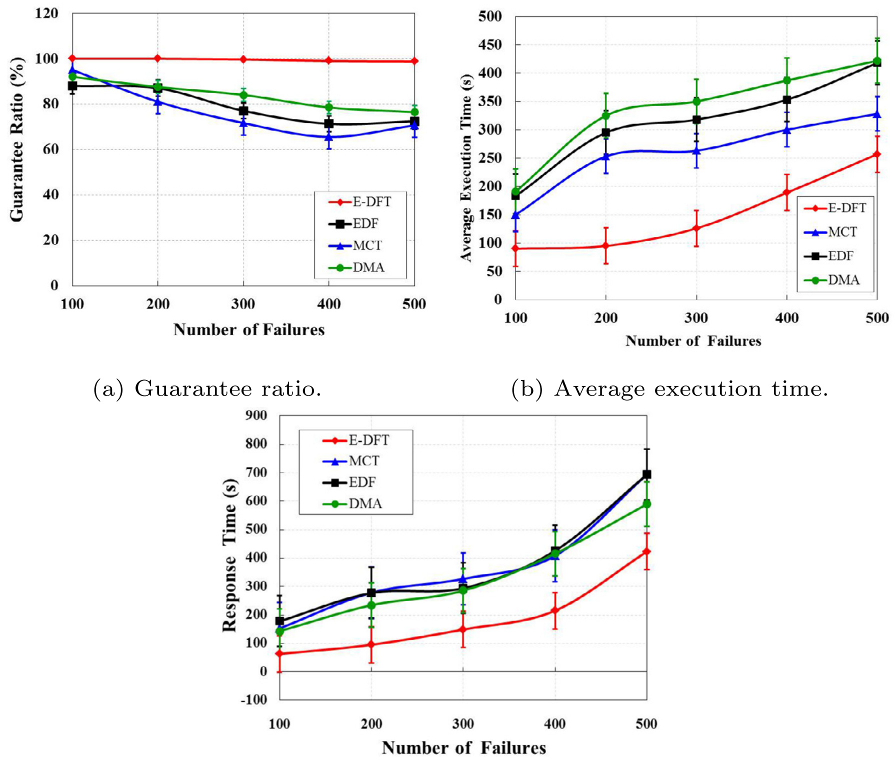
Fig. 10. Number of tasks versus different metrics.



(a) Guarantee ratio. (b) Average execution time.

(c) Active time of PMs. (d) Resource utilization.

Fig. 11. Base deadline versus different metrics.



(a) Guarantee ratio.

(b) Average execution time.

(c) Response time.

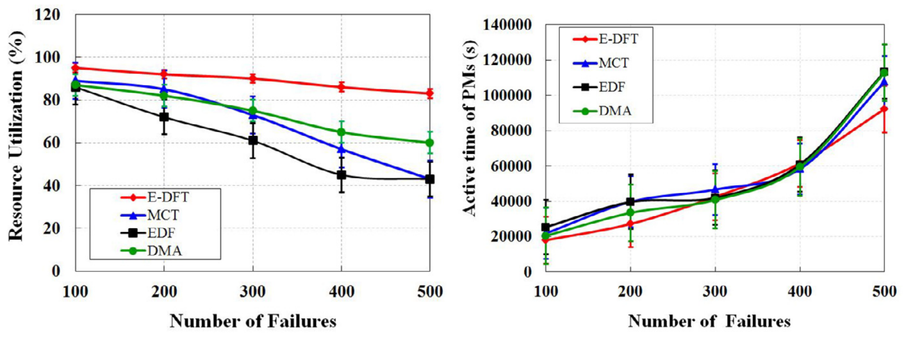
Fig. 12. Number of failures versus different metrics.

With increase in the value of N_f, the Guarantee Ratio decreases for EDF, DMA and MCT. This is due to failures of PMs and VMs that lead to the missing of the deadline of the tasks those are scheduled in them. Moreover, in some cases, it was also observed that some low priority tasks scheduled a healthy PM is successfully executed. On the contrary, due to poor response times of EDF and DMA, the deadlines for some high priority tasks are missed when the tasks are re-executed due to failures of the PMs. Despite the failures, E-DFT outperforms over EDF, DMA and MCT due to faster response time and multiple backups in different hosts and thereby the Guarantee Ratio is almost maintained. When there are multiple failures, the executions of ongoing tasks are interrupted. However, the healthy PMs are already notified by the failed ones according to our E-DFT protocol. The healthy PMs with backups start the pull-based backup scheduling of the overlapped and fused tasks with response time of δ. Although the tasks are executed successfully due to multiple backup copies stored in different PMs, the average execution time of the tasks is increased as the new execution time incurs delay in response. Therefore, as shown in Fig. 12b, it leads to delay in response time as with increase in the number of failures, the numbers of tasks to be re-executed increase. However, E-DFT performs better over EDF, MCT and DMA, since it adopts backup scheduling. The response time for executing already-scheduled backup copies is less than finding a suitable PM for re-executing the failed tasks as adopted by other algorithms.

Fig. 12c shows the performance impact of N_f on response time of the four protocols. It can be clearly seen from the Fig. 12c that E-DFT has lower response time as compared to EDF, DMA and MCT as multiple copies of the backups are already scheduled in different PMs. If the PM with t_k^p fails, the nearest backup with minimum response time is used. This minimizes delay in E-DFT as compared to EDF, DMA and MCT, where the re-execution of the failed tasks must be carried out by suitably scheduling the tasks on the available PMs. This increases the response time and therefore the tasks miss their deadlines.

The overhead of this proposed protocol is discussed in terms of resource utilization and energy consumption as shown in Fig. 13. E-DFT creates multiple backup copies of tasks and schedule them in different hosts. In the event of multiple failures, some of the active hosts that have executed the primaries may fail, resulting in reduction of resource utilization. However, the inactive hosts may be active to execute the backup tasks, which aid to improve the resource utilization. As a result, as shown in Fig. 13a, E-DFT has better resource utilization in comparison to EDF, MCT and DMA, since it uses the resources for scheduling of both primary and backup copies. In case of failures, the resource utilization that is reduced due to failures of primaries is compensated to some extent by improving the resource utilization with execution of the backups. In addition, with the use of BTO and BTF mechanism, the overlapped and fused backup copies of the tasks are pulled towards τ_{fail} with small delay δ, which is less than $LPS_{t_b^k}$. This pull-based fault tolerant mechanism allows faster completion of tasks and release of VMs and PMs resources, which in turn reduces the active time of PMs.

In case of failures, the response time for execution of the backup tasks is increased. As a result, the PMs need to remain active for longer duration. Consequently, the increase in active time of PMs results in increase in the energy consumption of the system. However, as shown in Fig. 13b, E-DFT has lower active time of PMs in comparison to EDF, MCT and DMA, which also lowers the energy consumption of the system. As a result, the proposed protocol can be used for providing better QoS as required by the users and can be used for maximizing the revenue by achieving better Guarantee Ratio as desired by the CSPs even in the event of failures.



(a) Number of failures versus Resource utilization.

(b) Number of failures versus Active time of PMs.

Fig. 13. Effectiveness of E-DFT in terms of resource utilization and active time of PMs.

6. Conclusion

This paper investigates the elastic pull-based dynamic fault tolerant scheduling of different independent tasks in cloud computing. The scheduling aims to minimize the response time while responding to simultaneous failures of PMs and VMs considering the resource utilization and guarantee ratio. The resource utilization of the proposed protocol is maximized by using BTO and BTF techniques in various VMs and multiple hosts. In case of failures, the response time is minimized by using our scheduling mechanism through increasing the guarantee ratio. In order to evaluate the performance of the proposed protocol, simulation was performed using CloudSim as building block. The proposed protocol, E-DFT is efficient in terms of response time, resource utilization, guarantee ratio, and energy consumption compared to EDF, DMA, and MCT especially during the failures. This work mainly focuses on the successful execution of the accepted tasks, while maximizing the resource utilization and minimizing energy consumption of the CSPs. How the CSP could provide better QoS even in the event of multiple resource failures is investigated here. Different pricing policies adopted for renting the VMs from users' point of view is not considered here. This could be considered as our future work, where a user can decide where he wants to submit the tasks based on the pricing policy of different CSPs. Failures in case of workflows and designing failure prediction model with implementation in real cloud computing platform will also be considered as our future work to improve the efficiency of the proposed protocol.

CRediT authorship contribution statement

Pushpanjali Gupta, Prasan Kumar Sahoo and **Bharadwaj Veeravalli** conceived the idea. Pushpanjali Gupta and Prasan Kumar Sahoo developed the algorithms and theoretical models. Pushpanjali Gupta designed the simulation outline, performed the simulation and prepared the manuscript. Prasan Kumar Sahoo supervised the work and supported the infrastructure and funding. Prasan Kumar Sahoo and Bharadwaj Veeravalli revised the manuscript.

Declaration of competing interest

The authors declare no conflicts of interest.

Acknowledgment

This work was supported in part by the Ministry of Science and Technology (MOST), Taiwan, under Grant number 109-2221-E-182-014.

References

[1] S.M. Abdulhamid, M.S. Abd Latiff, S.H.H. Madni, M. Abdullahi, Fault tolerance aware scheduling technique for cloud computing environment using dynamic clustering algorithm, Neural Comput. Appl. 29 (1) (2018) 279–293.
[2] M. Amoon, N. El-Bahnasawy, S. Sadi, M. Wagdi, On the design of reactive approach with flexible checkpoint interval to tolerate faults in cloud computing systems, J. Ambient Intell. Humaniz. Comput. 10 (11) (2019) 4567–4577.
[3] A. Arunarani, D. Manjula, V. Sugumaran, Task scheduling techniques in cloud computing: a literature survey, Future Gener. Comput. Syst. 91 (2019) 407–415.
[4] B. Balasubramanian, V.K. Garg, Fault tolerance in distributed systems using fused data structures, IEEE Trans. Parallel Distrib. Syst. 24 (4) (2012) 701–715.
[5] R. Begam, W. Wang, D. Zhu, Timer-cloud: time-sensitive vm provisioning in resource-constrained clouds, IEEE Trans. Cloud Comput. (2017) 1.
[6] S. Belia, F. Fidler, J. Williams, G. Cumming, Researchers misunderstand confidence intervals and standard error bars, Psychol. Methods 10 (4) (2005) 389.
[7] R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A.F. De Rose, R. Buyya, Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Softw. Pract. Exp. 41 (1) (2011) 23–50.
[8] H. Chen, X. Zhu, H. Guo, J. Zhu, X. Qin, J. Wu, Towards energy-efficient scheduling for real-time tasks under uncertain cloud computing environment, J. Syst. Softw. 99 (2015) 20–35.
[9] M.N. Cheraghlou, A. Khadem-Zadeh, M. Haghparast, A survey of fault tolerance architecture in cloud computing, J. Netw. Comput. Appl. 61 (2016) 81–92.
[10] M. Dabbagh, B. Hamdaoui, M. Guizani, A. Rayes, An energy-efficient vm prediction and migration framework for overcommitted clouds, IEEE Trans. Cloud Comput. (2017) 1.
[11] E.J. Ghomi, A.M. Rahmani, N.N. Qader, Load-balancing algorithms in cloud computing: a survey, J. Netw. Comput. Appl. 88 (2017) 50–71.
[12] Y. Guo, A. Stolyar, A. Walid, Online vm auto-scaling algorithms for application hosting in a cloud, IEEE Trans. Cloud Comput. (2018) 1.
[13] Z.A. Hammadeh, S. Quinton, R. Ernst, Weakly-hard real-time guarantees for earliest deadline first scheduling of independent tasks, ACM Trans. Embed. Comput. Syst. 18 (6) (2019) 1–25.
[14] G. Joshi, E. Soljanin, G. Wornell, Efficient replication of queued tasks for latency reduction in cloud systems, in: 2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton), IEEE, 2015, pp. 107–114.
[15] C. Kathpal, R. Garg, Survey on fault-tolerance-aware scheduling in cloud computing, in: Information and Communication Technology for Competitive Strategies, Springer, 2019, pp. 275–283.
[16] Y. Laalaoui, J. Al-Omari, A planning approach for reassigning virtual machines in iaas clouds, IEEE Trans. Cloud Comput. (2018) 1.
[17] D. Li, C. Chen, J. Guan, Y. Zhang, J. Zhu, R. Yu, Dcloud: deadline-aware resource allocation for cloud computing jobs, IEEE Trans. Parallel Distrib. Syst. 27 (8) (2016) 2248–2260.
[18] J. Liu, S. Wang, A. Zhou, S. Kumar, F. Yang, R. Buyya, Using proactive fault-tolerance approach to enhance cloud service reliability, IEEE Trans. Cloud Comput. (2017) 1.
[19] L. Luo, S. Meng, X. Qiu, Y. Dai, Improving failure tolerance in large-scale cloud computing systems, IEEE Trans. Reliab. 68 (2) (2019) 620–632.
[20] A. Marahatta, Y. Wang, F. Zhang, A.K. Sangaiah, S.K.S. Tyagi, Z. Liu, Energy-aware fault-tolerant dynamic task scheduling scheme for virtualized cloud data centers, Mob. Netw. Appl. 24 (3) (2019) 1063–1077.
[21] C.D. Martino, S. Sarkar, R. Ganesan, Z.T. Kalbarczyk, R.K. Iyer, Analysis and diagnosis of sla violations in a production saas cloud, IEEE Trans. Reliab. 66 (1) (2017) 54–75.
[22] Y.S. Patel, A. Page, M. Nagdev, A. Choubey, R. Misra, S.K. Das, On demand clock synchronization for live vm migration in distributed cloud data centers, J. Parallel Distrib. Comput. 138 (2020) 15–31.

[23] C. Pham, L. Wang, B.C. Tak, S. Baset, C. Tang, Z. Kalbarczyk, R.K. Iyer, Failure diagnosis for distributed systems using targeted fault injection, IEEE Trans. Parallel Distrib. Syst. 28 (2) (2017) 503–516.
[24] D. Puthal, R. Ranjan, A. Nanda, P. Nanda, P.P. Jayaraman, A.Y. Zomaya, Secure authentication and load balancing of distributed edge datacenters, J. Parallel Distrib. Comput. 124 (2019) 60–69.
[25] M. Soualhia, F. Khomh, S. Tahar, A dynamic and failure-aware task scheduling framework for hadoop, IEEE Trans. Cloud Comput. (2018) 1.
[26] G.L. Stavrinides, H.D. Karatza, Scheduling real-time bag-of-tasks applications with approximate computations in saas clouds, Concurr. Comput., Pract. Exp. 32 (1) (2020) e4208.
[27] Q. Tang, L.-H. Zhu, L. Zhou, J. Xiong, J.-B. Wei, Scheduling directed acyclic graphs with optimal duplication strategy on homogeneous multiprocessor systems, J. Parallel Distrib. Comput. 138 (2020), https://doi.org/10.1016/j.jpdc.2019.12.012.
[28] J. Teraiya, A. Shah, Analysis of dynamic and static scheduling algorithms in soft real-time system with its implementation, in: Soft Computing: Theories and Applications, Springer, 2020, pp. 757–768.
[29] J. Wang, W. Bao, X. Zhu, L.T. Yang, Y. Xiang Festal, Fault-tolerant elastic scheduling algorithm for real-time tasks in virtualized clouds, IEEE Trans. Comput. 64 (9) (2015) 2545–2558.
[30] L. Wang, K.S. Trivedi, Architecture-based reliability-sensitive criticality measure for fault-tolerance cloud applications, IEEE Trans. Parallel Distrib. Syst. 30 (11) (2019) 2408–2421.
[31] H. Yuan, J. Bi, W. Tan, B.H. Li, Temporal task scheduling with constrained service delay for profit maximization in hybrid clouds, IEEE Trans. Autom. Sci. Eng. 14 (1) (2017) 337–348.
[32] H. Yuan, J. Bi, W. Tan, M. Zhou, B.H. Li, J. Li, Ttsa: an effective scheduling approach for delay bounded tasks in hybrid clouds, IEEE Trans. Cybern. 47 (11) (2017) 3658–3668.
[33] M. Zakarya, L. Gillam, Managing energy, performance and cost in large scale heterogeneous datacenters using migrations, Future Gener. Comput. Syst. 93 (2019) 529–547.
[34] P. Zhang, M. Zhou, Dynamic cloud task scheduling based on a two-stage strategy, IEEE Trans. Autom. Sci. Eng. 15 (2) (2017) 772–783.
[35] Q. Zheng, B. Veeravalli, C.K. Tham, On the design of fault-tolerant scheduling strategies using primary-backup approach for computational grids with low replication costs, IEEE Trans. Comput. 58 (3) (2009) 380–393.

Pushpanjali Gupta received the BSc degree in Computer Science with Honors from Fakir Mohan University, India in June 2011 and the MCA degree from Orissa University of Agriculture and Technology, India in June 2014. Currently she is working toward the PhD degree in the Department of Computer Science and Information Engineering, Chang Gung University, Taiwan. Her research interests include fault tolerance, real-time scheduling, and resource management problems of cloud computing.

Prasan Kumar Sahoo received the B.Sc. degree in physics (with Honors), the M.Sc. degree in mathematics both from Utkal University, Bhubaneswar, India, in 1987 and 1994, respectively. He received the M.Tech. degree in computer science from the Indian Institute of Technology (IIT), Kharagpur, India, in 2000, the first Ph.D. degree in mathematics from Utkal University, in 2002, and the second Ph.D. degree in computer science and information engineering from the National Central University, Taiwan, in 2009. He is currently a Professor in the Department of Computer Science and Information Engineering, Chang Gung University, Taiwan. He is an Adjunct Researcher in the Department of Neurology, Chang Gung Memorial Hospital, Linkou, Taiwan. He was a Visiting Associate Professor in the Department of Computer Science, Universite Claude Bernard Lyon 1, Villeurbanne, France. His current research interests include artificial intelligence, edge computing, and IoT. He is currently Editorial Board Member of Elsevier's Journal of Network and Computer Applications and Inderscience's International Journal of Vehicle Information and Communication Systems. He is also Topic Editor of Electronics Journal, MDPI. He has worked as the Program Committee Member of several IEEE and ACM conferences and is a senior member, IEEE.

Bharadwaj Veeravalli received his BSc degree in Physics, from Madurai-Kamaraj University, India, in 1987, the Master's degree in Electrical Communication Engineering from the Indian Institute of Science, Bangalore, India in 1991, and the PhD degree from the Department of Aerospace Engineering, Indian Institute of Science, Bangalore, India, in 1994. He received gold medals for his bachelor degree overall performance and for an outstanding PhD thesis (IISc, Bangalore India) in the years 1987 and 1994, respectively. He is currently with the Department of Electrical and Computer Engineering, Communications and Information Engineering (CIE) division, at The National University of Singapore, Singapore, as a tenured Associate Professor. His main stream research interests include cloud/grid/cluster computing (big data processing, analytics and resource allocation), scheduling in parallel and distributed systems, Cybersecurity, and multimedia computing. He is one of the earliest researchers in the field of Divisible Load Theory (DLT). He is currently serving the editorial board of IEEE Transactions on Parallel and Distributed Systems as an associate editor. He is a senior member of the IEEE and the IEEE-CS.