SOFTWARE: PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2017; **47**:1243–1274
Published online 4 April 2017 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/spe.2491

Failover strategy for fault tolerance in cloud computing environment

Bashir Mohammed*,†, Mariam Kiran, Kabiru M. Maiyama, Mumtaz M. Kamala and Irfan-Ullah Awan

School of Electrical Engineering & Computer Science, University of Bradford, Bradford, UK

SUMMARY

Cloud fault tolerance is an important issue in cloud computing platforms and applications. In the event of an unexpected system failure or malfunction, a robust fault-tolerant design may allow the cloud to continue functioning correctly possibly at a reduced level instead of failing completely. To ensure high availability of critical cloud services, the application execution, and hardware performance, various fault-tolerant techniques exist for building self-autonomous cloud systems. In comparison with current approaches, this paper proposes a more robust and reliable architecture using optimal checkpointing strategy to ensure high system availability and reduced system task service finish time. Using pass rates and virtualized mechanisms, the proposed smart failover strategy (SFS) scheme uses components such as cloud fault manager, cloud controller, cloud load balancer, and a selection mechanism, providing fault tolerance via redundancy, optimized selection, and checkpointing. In our approach, the cloud fault manager repairs faults generated before the task time deadline is reached, blocking unrecoverable faulty nodes as well as their virtual nodes. This scheme is also able to remove temporary software faults from recoverable faulty nodes, thereby making them available for future request. We argue that the proposed SFS algorithm makes the system highly fault tolerant by considering forward and backward recovery using diverse software tools. Compared with existing approaches, preliminary experiment of the SFS algorithm indicates an increase in pass rates and a consequent decrease in failure rates, showing an overall good performance in task allocations. We present these results using experimental validation tools with comparison with other techniques, laying a foundation for a fully fault-tolerant infrastructure as a service cloud environment.

Received 1 February 2016; Revised 23 November 2016; Accepted 20 January 2017

KEY WORDS: cloud computing; fault tolerance; checkpointing; virtualization; load balancing; virtual machine

1. INTRODUCTION

Cloud computing is a popular paradigm and an attractive model for providing computing, information technology infrastructure, network, and storage to end users in both large and small business enterprises [1]. The surge in cloud popularity is mainly driven by its promise of on-demand flexibility and scalability, without committing any upfront investment in implementation, with reduction in operating costs of infrastructure and data centers [2]. Cloud ecosystems can be public, private, hybrid, or even community depending on the networking model used in delivering services [19, 22, 3]. Cloud computing relies on sharing resources to accomplish scale, sharing services, and infrastructure, as its delivery models. To maintain reliability and availability, fault tolerance (FT) in cloud becomes an important property, allowing the system to continue functioning properly in events of failure. Embedding a fault-tolerant design in cloud architecture allows the system to

*Correspondence to: Bashir Mohammed, School of Electrical Engineering & Computer Science, University of Bradford, BD7 1DP, UK.
†E-mail: b.mohammed1@bradford.ac.uk

continue its intended operation, even at a reduced level, preventing it from breaking down completely when unexpected failure events occur [5, 6].

In real-time high-performance large-scale systems with complex and dynamic cloud services, the system sometimes fails because of varying execution environments, removal and addition of system components, or intensive workload on the servers [1, 6]. Failures cost cloud vendors not only their businesses and clients but also their reputation. To prevent these, steps need to be taken to handle possible failures emerging within cloud infrastructures. These FT techniques are designed around the concepts of fault-finding principles, such as predicting and evaluating possible future failures, allowing the system to continue its functions at satisfactory levels [1]. Having a reliable fault-tolerant cloud infrastructure prevents the system from breaking down completely.

- A number of fault-tolerant strategies have been realized in research over the years based on fault-tree analysis, checkpointing, and prediction models. However, only a fraction of these have been applied to cloud computing systems bearing in mind that the risk of failure is increasing as the complexity of tasks increases in a typical cloud environment. This paper argues a new integrated virtualized optimal checkpointing FT approach for cloud data centers, using intelligent selection mechanisms based on pass rates (PRs) of computing virtual nodes with a fault manager. This smart failover strategy (SFS) FT approach results in an optimized infrastructure for infrastructure as a service (IaaS) cloud platforms, showing a considerable improvement in current research of cloud FT.

Along with analyzing current cloud FT approaches, the main contributions of this paper include the following:

- Providing high availability depending on cloud user requests to successful virtual machines (VMs) using the SFS algorithm.
- Develop an integrated virtualized failover strategy for cloud data centers, overall reducing the system service time.
- Prove the viability of the SFS approach through quantitative analysis and compare performance with existing methods. We validate the method using simulation to give details of successful performance in failure situations.

The paper has been organized as follows: Section 2 presents the problem definition and techniques currently being used to explore the problem of failure recovery in cloud environments. This is elaborated in Section 3 presenting the related background in FT for standard real-time cloud computing systems. Section 4 presents our FT architecture, use case scenarios, computation algorithm, and working model of the proposed approach using mathematical analysis and its implementation details with cloud infrastructures. These implementation details are expanded in Section 5 by presenting an experimental setup as well as performance comparison of results with existing approaches. Discussion of results is given in Section 6, and finally, Section 7 presents conclusions and future work of the approach.

2. PROBLEM DEFINITION

Cloud computing relies on sharing resources to accomplish scale of services and infrastructure. As cloud complexity grows, failures of virtual nodes providing the services increase and become difficult to predict, particularly with system components being constantly upgraded, intensive workload on cloud servers, and sometimes deploying faulty software. Achieving high availability of VMs always is a huge challenge because of the sheer number of virtual and physical machines involved, increasing the probability and risk of failure.

It is imperative to prevent failures emerging within the cloud infrastructures to prevent business and financial losses. In 2011, Microsoft cloud service outage lasted for 2.5 h [30], with Google Docs service outage lasting for 1 h. These were because of memory leaks due to a software update [42], costing both businesses millions of dollars. Similar reports were witnessed by Gmail services down for about 50 min and Amazon Web services for 6 h, while Facebook's photos and 'likes'

DOI: 10.1002/spe

services were down costing customer dissatisfaction. Multiple businesses hosting their websites, such as with GoDaddy, suffered 4 h downtime affecting five million websites [30].

A growing number of data center resources increase the global complexity of Information and Communications Technology (ICT) services, where cloud users use them to handle business-critical and high-computing processes [7]. As a result, it is of vital importance to ensure high reliability and availability to prevent resource failure. One of the most common challenges in cloud is failure to deliver its function, either by software or hardware failures [8, 9]. The service tasks executing over the cloud VMs have large time spans of a few days or months, running long tasks or jobs. Failure on these long or medium running jobs brings threats to fulfillment of service-level agreement contracts and delays in job completion times to perform computational processes [1, 10]. An example of such a failure occurred when a 20% revenue loss was reported by Google, as an experiment caused an additional delay of 500 ms in response time [11]. In another example, millions of customers were left without Internet access for 3 days when there was a core switch failure in BlackBerry's network. Another example is when one of UK's top cellular companies failed for 3 days, affecting seven million subscribed customers [11, 12].

There are three levels of essential services offered by cloud computing: IaaS, platform as a service (PaaS), and software as a service. Each level of service handles FT at different levels of complexity.

- IaaS is the most basic and important cloud service model under which VMs, load balancers, FT, firewalls, and networking services are provided. The client or cloud user is provided with capability to provision processing, storage, networks, and other fundamental computing resources, to deploy and run arbitrary software such as operating system and applications. Common examples of these services include Rackspace, GoGrid, EC2, and Amazon cloud [13].
- Under the PaaS model, a computing platform including APIs, operating system, and development environments is provided as well as programming language execution environment and web servers. The client maintains the applications, while the cloud provider maintains the service run times, databases, server software, integrated server-oriented architectures, and storage networks. Various types of PaaS vendors offerings can include complete application hosting, development, testing, and extensive integrated services that include scalability and maintenance. Some key players include Microsoft Windows Azure and Google Apps engine. The main benefits of these services include focusing on high-value software rather than infrastructure, leveraging economies of scale, and providing scalable go-to-market capability [2].
- Software as a service provides clients the capability to use provider application executing on a cloud infrastructure. An entire application is available remotely and accessible from multiple client devices through thin client interfaces such as web browsers. A cloud user does not manage or control the underlying cloud infrastructure [2], but providers install and operate the application software. Example providers for this service include Salesforce, Facebook, and Google Apps [2, 14, 15].

The main objective of a computational cloud platform is to execute user applications or jobs, where users submit their jobs to the service provider (SP) along with their quality of service (QoS) requirements. These requirements may include job deadlines, required resources for the job, and the needed platform. The SP submits a task to the cloud controller (CC), and the scheduler allocates each job with suitable resources.

Depending on the type of fault and FT policies, several FT techniques can be used [16, 20, 21] such as reactive FT policy, which reduces failure effects when they occur on application execution, and proactive FT policy, which avoids fault recovery by predicting and proactively replacing the suspected faulty components. In case of a fault-free scenario, results of successful jobs are returned to users after job completion. If there are failures during the job execution, then the cloud fault manager (CFM) is informed, and the job is rescheduled on another VM resource to re-execute the job from the last successful checkpoint. This results in more time consumed for the job than expected, risking the QoS not being satisfied.

Many FT strategies have been designed to reduce fault effects, but in this paper, we propose an optimized FT strategy in real-time cloud computing environment to increase system availability,

DOI: 10.1002/spe

reduce the service time, and enhance rapid and efficient recovery from faults. Our SFS approach is applied to the IaaS delivery layer, utilizing computing hardware resources and the virtualization hypervisor to manage VM instances running on physical servers.

To address the problem of job completion time, we integrated the PR-optimized selection technique with a job checkpointing mechanism in the SFS approach. Here, we restore the partially completed job from the last successful saved checkpoint rather than restarting the job. This greatly decreases the system re-computing time. However, we recognize a few drawbacks of checkpointing mechanism, such as performing identical processes regardless of stable resources, higher checkpoint overhead to store the entire system running states, and inappropriate checkpointing that causes delay in job execution. Commonly used checkpointing mechanisms are discussed in [17]; however, in real-time computational cloud environments, there are cases where resources satisfy QoS requirements (partial pass scenario) but are not selected because the load balancers assign tasks to VMs based on highest PR in job execution. To address these problems, our optimized selection technique selects only VMs with successful status check and successful task time limit checker (TTLC). Further components such as load balancers, FT engine, firewalls, and networking services are utilized by cloud data centers to help manage and regulate FT strategies in the cloud model [2, 18].

3. RELATED WORK

There are many approaches proposed to deal with FT in cloud, and recent studies have analyzed FT in cloud and grid computing [19–29] [9, 10, 30–37,20] and more broadly in the area of FT for standard real-time systems [6, 33, 38, 4, 31, 39–43,26, 27], [44], but very few works have addressed issues of optimized FT strategies in cloud environment in relation to high system availability. Various researchers have provided FT solutions specific to certain cloud delivery models by focusing on either high availability frameworks, using virtual nodes for fault prediction or using user-defined APIs to help optimize cloud performance in faulty situations.

Focusing on certain framework and delivery models, Tchana *et al*. [32] analyzed the implementation of FT by focusing on autonomic repair. They proved that in most current FT approaches, faults are exclusively handled by the provider or the customer who leads to partial or inefficient solutions, while collaborative solutions are much more promising. They demonstrated this with experiments on collaborative FT solutions by implementing an autonomic prototype cloud infrastructure. Maloney and Goscinski [24] reviewed issues relating to providing FT for long-running applications. They investigated several FT approaches where they found that rollback recovery provides a favorable approach for user applications in cluster systems. They further explained that two facilities can provide FT using rollback recovery: checkpointing and recovery. They concluded that the problems associated with providing recovery include providing transparent and autonomic recovery, by selecting appropriate recovery computers and maintaining consistent observable behavior when applications fail. Using the technique of record and playback, Kim *et al*. [43] proposed a two-node selection scheme, namely, playback node first and playback node first with prefetching, that can be used for a service migration-based fault-tolerant streaming service. Their proposed scheme demonstrated that the failure probability of a node currently being served is lower than that of a node not being served.

Addressing software bugs, Chen *et al*. [21] presented a lightweight-software fault-tolerance system in cloud, called SHelp, which can effectively recover programs from many types of software bugs in the cloud environment. They proposed a 'weighted' rescue point technique that effectively survives software failures through bypassing the faulty path. Their idea was that, in order to share error-handling information for multiple application instances running on different VM, a three-level storage hierarchy with several comprehensive cache-updating algorithms for rescue points management is adopted. Their experimental results showed that SHelp can recover server applications from these bugs in just a few seconds with modest performance overhead.

However, focusing on checkpointing, Qiang *et al*. [45] presented a multi-level fault-tolerant system for distributed applications in cloud named distributed application-oriented multi-level checkpoint/restart for cloud. The system backs up complete application states periodically as a snapshot-based-distributed checkpointing protocol, including file system state. The authors

DOI: 10.1002/spe

proposed a multi-level recovery strategy, which includes process-level recovery, VM recreation, and host rescheduling, enabling comprehensive and efficient FT for different components in cloud. Alshareef and Grigoras [25] introduced a checkpoint technique to capture session progress. The authors claimed the technique is an additional service to their cloud management of the Mobile ad hoc network (MANET). Their experimental results showed that the model is feasible and robust and saves time and energy if session breaks occur frequently. Additionally, Agbaria and Friedman [46] proposed a VM-based heterogeneous checkpointing mechanism, where they explored how to construct a mechanism at VM level rather than dumping the entire state of the application process. The mechanism dumps the state of application as maintained by a VM and during restart, and the saved state is loaded as a new copy of the VM, to continue running from here. The authors reported on main issues encountered in building such a mechanism and design choices made. They concluded by presenting a performance evaluation and ideas for extending the work to a native code O Caml and Java.

In other approaches, Kaur *et al*. [31] examined the implementation of FT in a complex cloud computing environment with a focus on first come first serve and shortest job first along with misses per instruction (MPI) on large method with FT property. Their proposed algorithm works for reactive FT among the servers and reallocates the faulty servers' task to the new server, which has a minimum load at the instant of the fault cloud infrastructure that we prototyped. It also includes algorithm comparison between MPI and MPI on large.

Further works by Singh *et al*. [47] presented an approach for providing high availability to the requests of cloud clients by proposing failover strategies for cloud computing using integrated checkpointing algorithms and implemented the strategies by developing a cloud simulation environment, which can provide high availability to clients in case of failure/recovery of service nodes. They conducted a comparison of developed simulator with existing methods and concluded that the purposed failover strategy will work on application layer and provide high availability for PaaS architectures. Kong *et al*. [39] analyzed the performance, fault tolerance, and scalability of virtual infrastructure management systems with three typical structures, including centralized, hierarchical, and peer-to-peer structures, giving a mathematical definition of the evaluation metrics using quantitative analysis for enhancing performance, fault tolerance, and scalability.

Addressing high availability, Jung *et al*. [48] provided an enhanced solution to this classical problem of ensuring high availability by maintaining performance and by regenerating software components to restore the redundancy of a system whenever failures occur. The authors achieved an improved availability by smartly controlling component placement and resource allocation using information about application control flow and performance predictions from queuing models, ensuring that the resulting performance degradation is minimized. The authors concluded that their proposed approach provided a better availability and significantly lower degradation of system response times compared with traditional approaches. An alternate approach by Shen *et al*. [7] proposed a mechanism called availability on demand, which consisted of an API that allowed data center users to specify availability requirements and use an availability-aware scheduler that can dynamically manage computing resources based on user-specified requirements. Their mechanism operates at a level of individual service instance, thus enabling fine-grained control of availability. While the authors argued that availability-on-demand mechanism can achieve high availability with low cost, the approach is extremely high in resource intensive. Another similar approach of dynamically adapting based on parameters, Chtepen *et al*. [49] introduced several information units on grid status, to provide high-job throughput in the presence of failure while reducing the system overhead. They presented a novel fault-tolerant algorithm combining checkpointing and replication and evaluated it in a grid simulation environment called dynamic scheduling in distributed environments. From their obtained experimental results, it was concluded that the adaptive approach can considerably improve system performance, while the solutions depend on system characteristics, such as load, job submission patterns, and failure frequency. Das *et al*. [50] proposed a virtualization and FT technique to reduce the service time and increase the system availability. The authors used a cloud manager module and a decision-maker in their scheme to manage virtualization and load balancing and also handle faults. By performing virtualization and load balancing, FT was achieved by redundancy and fault handlers. Their technique was mainly

DOI: 10.1002/spe

designed to provide a reactive FT where the fault handler prevents the unrecoverable faulty nodes from having an adverse effect.

Addressing optimization methodologies in fault situations, Elliott *et al*. [51] proposed a model and analyzed the benefit of C/R in coordination with redundancy at different degrees to minimize the total wall clock time and resources utilization of high-performance computing (HPC) applications. They carried out an experiment with an implementation of redundancy within the MPI layer on a cluster, and the results confirmed the benefit of dual and triple redundancy showing a close fit to the model. Yanagisawa *et al*. [52] proposed a mixed integer programming approach that considered the fluctuations of resource demands for optimal and dependable allocation of VMs by allocating VMs successfully in a cloud computing environment. Israel *et al*. [53] modelled an offline optimization problem and presented a bi-criteria approximation algorithm by presenting a much simpler and practical heuristic based on a greedy algorithm. They evaluated the performance of this heuristic over real data center parameters and showed that it performs well in terms of scale, hierarchical faults, and variant costs. Their results indicated that their scheme can reduce the overall recovery costs by 10–15%, compared with currently used approaches, by showing that the cost-aware VM placement may further help in reducing expected recovery costs, as it reduces the backup machine activation costs. Parveen *et al*. [33] proposed a model called high-adaptive FT in real-time cloud computing, based on computing the reliabilities of the VMs based on cloudlets, using million instructions per second (mips), RAM, and bandwidth. In this approach, if there are two VMs, both having the same reliabilities values, then the winning machine is chosen based on the priority assigned to them. Using parameters for optimizing behavior, Malik *et al*. [6] proposed an FT model for real-time cloud computing, where the system would tolerate the faults and then make the decision on the basis of reliability of the processing nodes or VMs. They presented a metric model for the reliability assessment where they assessed the number of times a decrease in reliability occurred compared with the number of times an increase happened. This proposed technique was based on the execution of design diverse variants on multiple VMs, and by assigning reliability to the results-produced variants. The main essence of their proposed technique is the adaptive behavior of the reliability weights assigned to each processing node by adding and removing nodes on the basis of reliability.

Further work by Egwutuoha *et al*. [38] presented a proactive FT approach to HPC systems in the cloud to reduce the wall clock execution time in the presence of faults. Their algorithm did not rely on a spare node for failure prediction and their experimental results, obtained from a real cloud execution environment, and showed that the wall clock execution time of the computation-intensive applications can be reduced by as much as 30% with the frequency of checkpointing reduced to 50% compared with current FT approaches. Further work by the authors [54] presented an FT framework using a process level redundancy technique to reduce the wall clock time of the execution of computational intensive applications. Other researchers such as Okorafor [40] also used HPC on the cloud by using message-passing interface and using checkpointing for VMs. Using simulations shows that the proposed approach compares favorably with the native cluster MPI implementations.

Following from various traditional approaches, this paper proposed a new model based on a smart FT approach in real-time cloud applications for running VMs. Using the techniques based on parameters being optimized, we apply a selection rate process approach, where a VM or node is selected for computation on the basis of its previous PR and overall task service time. If the VM does not show good performance, it can be deselected from the list of available VMs. This technique does not need to have a record and playback strategy because the guarantee of successful service completion is given by the initial decisions made at deployments of the service in VMs. This technique of using integrated virtualized failover strategy has been validated through quantitative and experimental results by simulations for testing performance for success in four scenarios – partial and full pass, and partial and full fail situations for FT in cloud environments. These results have been analyzed against the traditional approaches to see how well the cloud environment repairs and manages to fulfill the service completion tasks. The next section discusses the details of the approach.

DOI: 10.1002/spe

4. SMART FAULT TOLERANCE IN CLOUD – THE VISION

The overall vision of FT in cloud computing is to provide high availability to fulfill the client requests on service performance and completion time as defined by the service-level agreement. An FT service is an essential part of the service-level objective; therefore, an FT function in a cloud environment is extremely crucial. This section presents a working model of the strategy and a mathematical relationship that represents the FT model for our cloud computing system using the FT checkpoint scheme. The FT checkpoint uses a reward renewal process, which denotes that after each failure occurrence in the system, a backward recovery is performed and the VM is immediately restarted and recovered from the last successful checkpoint. Based on the FT system architecture consisting of four zones, the approach has been analyzed in relation to some extreme use cases to analyze how the CC would perform as presented in the next section.

We define a mathematical representation of the FT model for a cloud computing system, presenting a working model of integrated virtualized FT approach. The FT system architecture shown in Figure 1 consists of four zones, namely,

- The client zone: One or more clients can access service of cloud on demand at any given time.
- The virtualization zone: One or more VMs instances can be started up, terminated, and migrated within the data center, also acting as a link between client and FT cloud environment.
- The FT zone: Here, the hypervisor and virtual machine monitor (VMM) exist to support high-availability cloud service level and service-level objectives.
- The hardware zone: One or more distributed data centers in different locations with each data center consisting of numerous physical servers, providing hardware infrastructure for starting-up VMs instances [7].

Using the variables defined in Table I, let the FT model (FT_m) of a cloud computing system be represented by a finite-ordered list of elements or a sequence of five elements:

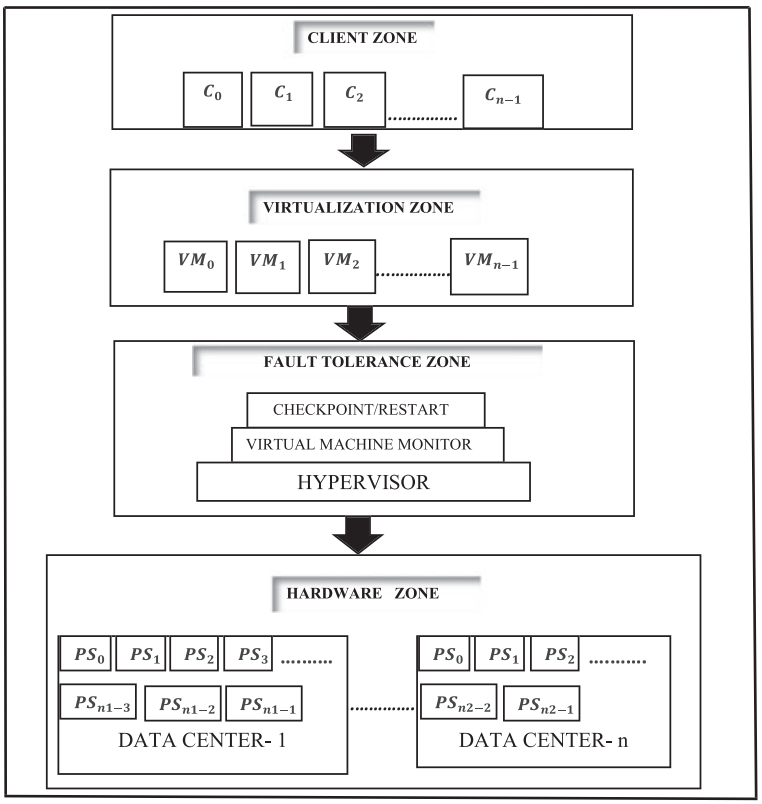


Figure 1. Fault tolerance architecture.

DOI: 10.1002/spe

Table I. Parameters of our architectural model

Parameters	Meaning
FT_m	FT model of a cloud computing system
C	A client set composed of n users
DC	Data center
FT_s	A set of defined FT service levels
OBJ_f	Objective function for optimizing an FT cloud
PR_A	Pass rate algorithm
FT_L	Fault tolerance level
Chk(Opt)	Checkpoint optimization strategy

FT, fault tolerance.

$$FT_m = \left(C, DC, FT_s, OBJ_f, PR_A\right), \tag{1}$$

$$\begin{cases} OBJ_f = \max(FT_L), \\ s.t \quad FT_L \in [0,\ 1], \\ \quad\quad Chk(Opt), \end{cases} \tag{2}$$

where PR_A is an algorithm that selects the optimal PR, $C = \{c_0, c_1, c_3, \ldots, c_{n-1}\}$ represents a set of n-clients that may request services separately, FT_L represents a set of FT service levels available by the cloud SP, OBJ_f is the cloud FT optimizing objective function as given in (1), and $DC = \{dc_0, dc_1, dc_3, \ldots, dc_{n-1}\}$ represents a data center set that is made up of dc_n data centers, where $dc_i = \left\{ps_0, ps_1, ps_3, \ldots, ps_{i_{dci-1}}\right\}$ and ps_{i_k} $(0 \leq k < dc_i)$ is the kth physical server of the ith data center dc_i.

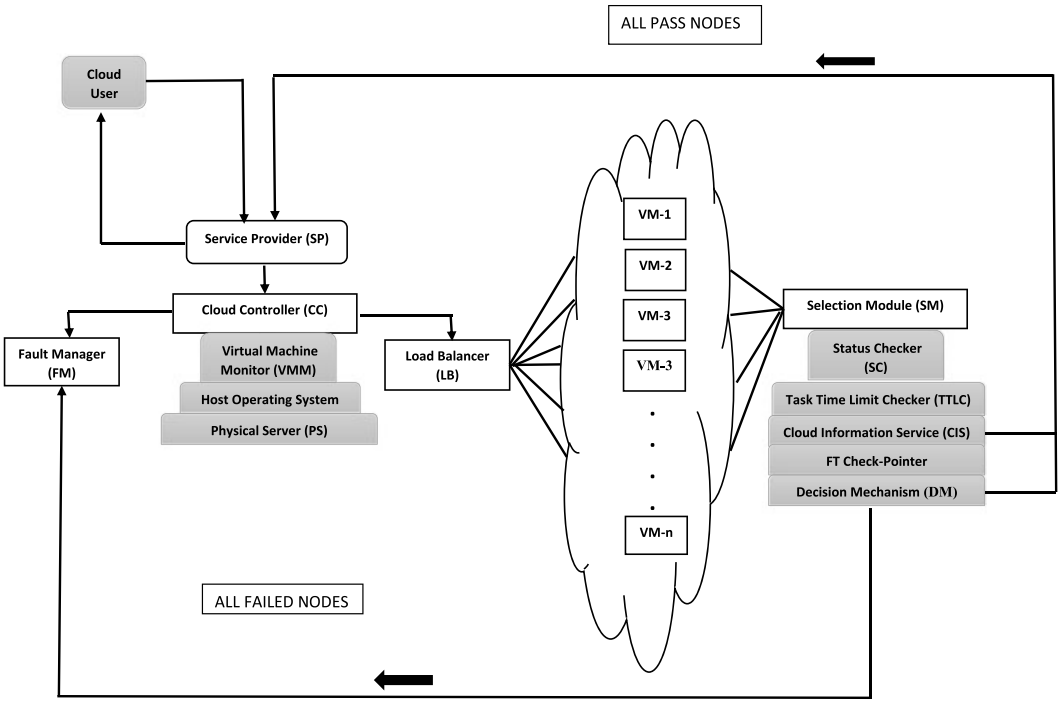
4.1. Working of the model

The technique aims at providing a high-availability system in presence of faults, achieved by using the selection rate process technique, where a VM or node is selected for computation on the basis of its previous PR. This node can be detached from the selection list if it does not operate well. According to the model, a set of nodes are created by requests from the resources of the host machine or the physical server. This is achieved by the VM monitor (or possibly the hypervisor) that is either software, hardware, or firmware that creates and runs VMs. The host machine is the server where the VMM or hypervisor runs guest VMs. The VMM presents the guest operating systems with virtual operating platforms and also manages the execution of these guest operating systems.

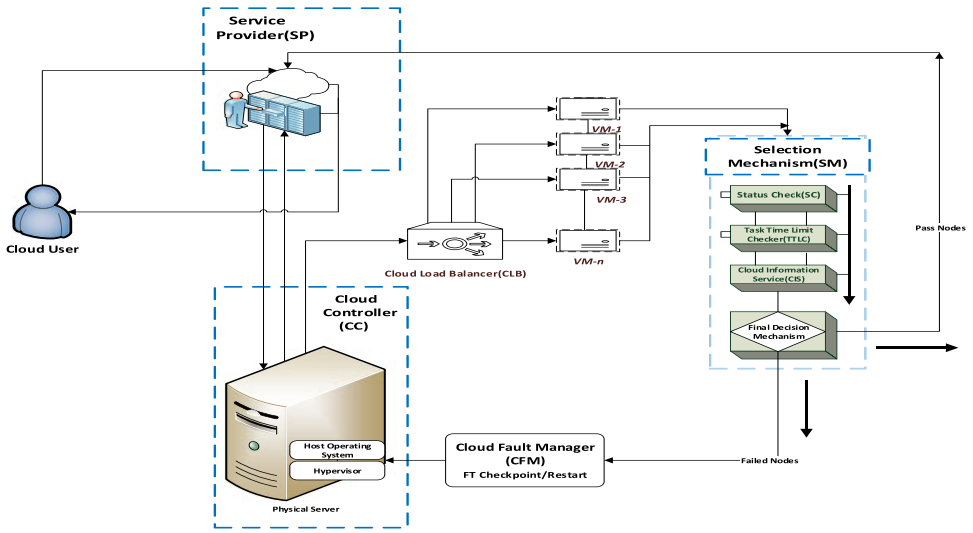
The VMM retains records of all virtual nodes created from the different host servers. In addition, it retains and manages records during the process when a load balancer mechanism assigns a job to a virtual node of a specific host server in order to evaluate the PR. The model is made up of the following modules as shown (Figure 2):

- SP – It is responsible for forwarding the task submitted by clients to the CC. It also returns results obtained from the CC to the cloud user.
- CFM – It is one of the most critical modules in the model as it keeps the system in operation and prevents breakdown. In a scenario where a virtual node develops a fault, as a result of some transient faults that occurred in remote host server of corresponding virtual node or due to some recoverable temporary software faults present in CC, here, the CFM takes responsibility and updates the cloud information service (CIS) record table. In other words, during this time, if there is no executing virtual node on the host server, the CFM module will remotely and automatically restart the server. At this particular point in time, the cloud load balancer (CLB) module is informed not to assign any further tasks to the virtual nodes of the concerned server. During this process, there might be a slight delay in the system restarting and jobs

DOI: 10.1002/spe



(a) Block Representation



(b) Topology

Figure 2. Proposed system model. [Colour figure can be viewed at wileyonlinelibrary.com]

waiting to be processed. However, the algorithm tries to recover quickly by minimizing the system service time and not losing any job during the process. It might also apply some fault detection strategy and successful recovery technique thereby making the virtual node of that host or physical server available for future request.

- CC – This is directly linked to the SP and is part of the cloud architecture. Virtualization is performed using the hypervisor, which provides system resources access to VMs and creates a virtual environment. In addition, it keeps record of virtual nodes and their corresponding physical nodes each time a virtual node is created. The virtual node IDs, server IDs, and PR are contained in the CIS, which helps to identify the virtual nodes and keeps record of tasks assigned to virtual nodes of a particular host or physical server.

DOI: 10.1002/spe

- CLB – The CIS is also available to the CLB and distributes the loads based on the information it gets from the record of physical systems used for virtualization. The CLB will assign a task to virtual nodes whose corresponding physical servers have a high PR.
- Status checker (SC) –This is the first submodule under the selection mechanism module. It checks the status of each virtual node either it passes or fails.
- CIS record table – This is a performance record table that contains server IDs, virtual nodes IDs, and PR values to identify corresponding virtual nodes.
- TTLC –The task time deadline of each task assigned is checked by the TTLC in the selection mechanism module. It also checks to see if the assigned task is completed within an agreed time limit.
- Selection module – This module provides the crucial process and is made up of the SC, TTLC, CIS record table, FT check pointer, and the final selection mechanism. Here, SC checks the status of each virtual node, and if the status is pass, then the task deadline time is checked by the TTLC. If both SC and TTLC are pass, then PR of corresponding node is increased and forwarded to the decision mechanism module for final selection process. But if both SC or TTLC fail, then the corresponding VM is not forwarded for final selection, and instead, the node is forwarded to the CFM for fault detection and recovery. In a scenario where SC is a success but the task is not completed within the time limit, the PR in the CIS record table of that particular node is decreased and that node is not forwarded to the final decision mechanism submodule. Table II present these rules in detail.

In addition, the final selection mechanism contains all virtual nodes that successfully passed the SC and TTLC module. After this point, the node with the highest PR value is selected, and checkpoint is made. But if all nodes failed, a backward recovery is carried out with help of the last successful checkpoint. Also, if there exist more than one node with same PR values, then a node will be selected at random.

4.2. Use case scenarios

In the aforementioned scheme, all virtual nodes run a different algorithm resulting in different scenarios of pass and fail rates, representing diversification in software and timing constraints. Table II below shows the rules of the system while Figure 3 presents the use case scenarios. The following are some scenarios that could occur:

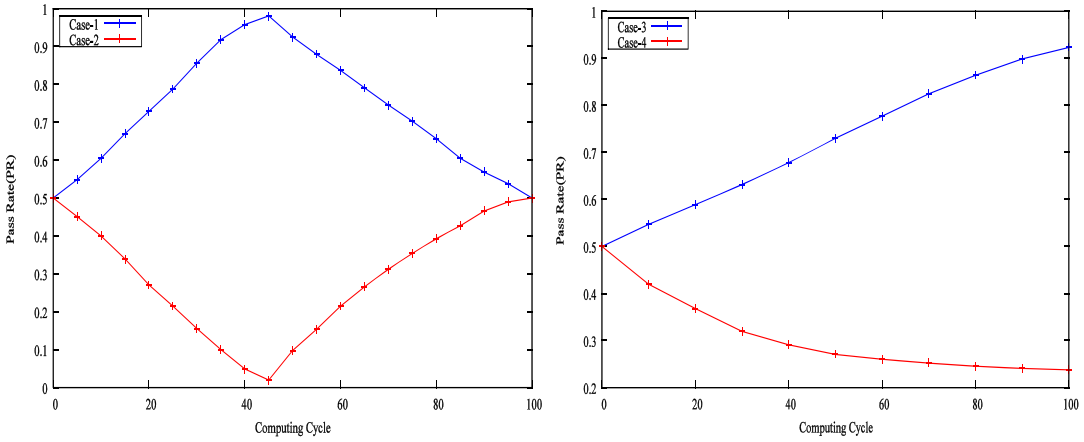
- Full pass scenario – The entire algorithm on each virtual node produces a successful outcome. Here, the SC and TTLC are also pass because the task is completed within the stipulated time limit. The PR of the corresponding node is then increased, and it goes to the selection mechanism for the final decision-making. The selection module contains all the virtual nodes that have successfully completed and passed the SC and TTLC module. The final selection module selects the node with the highest PR value and performs a checkpoint before sending back to the SP. However, in this case, no failure is recorded in any of the VMs.

Table II. Rules of the system

Rules	Condition	Decision
1	If (SC status == pass) && (TTLC status ==pass)	Increase PR and forward to selection module for decision and selection.
2	If (SC status == pass) && (TTLC status ==fail)	Update database in cloud information service module, decrease PR, and correspond virtual machine not sent to selection module.
3	If (SC status == fail) && (TTLC status == pass)	Decrease PR and node sent to CFM for identification, detection, and recovery.
4	If (SC status == fail) && (TTLC status ==fail)	Decrease PR and node also sent to CFM for detection and recovery.

PR, pass rate; CFM, cloud fault manager.

DOI: 10.1002/spe



(a)Use case 1 (Pass to Fail), Use case 2 (Fail to Pass)

(b)Use case 3 (Continuous Pass), Use case 4 (Continuous Fail)

Figure 3. Use case scenario for pass and fail. [Colour figure can be viewed at wileyonlinelibrary.com]

- Partial pass scenario – All VMs produce successful results where some of the results are generated within the time limit and some after the time. If the status check of a node is pass but the task is not completed within the agreed time limit, then the system is said to be in a partially pass state, and the node is not considered for a further decision by the final selection mechanism. The success rate for that particular node is also decreased, and an error signal is not generated for a failed VM. However, in this scenario, the system will continue to operate with forward recovery, and the selection mechanism will select the output from the nodes that have produced a good PR within the time limit.
- Full failure scenario – If the status check is failed, then automatically, the task limit check is also failed, and all the faulty nodes are sent to the CFM for fault detection and recovery. In this scenario, all nodes fail completely, and with the aid of the last successful checkpoint, a backward recovery is performed.
- Partial failure scenario – If either the SC or task limit time checker is failed, then the corresponding VM is not considered or forwarded to the final selection module. However, some VMs produce some pass results only when the SC is pass and the results are produced within the time limit, thereby sending the VM to the selection module and increasing the success rate of that node in the CIS record table. Here, error signals will be generated for failed VMs, and the corresponding node will not be sent to the final selection module. The system will continue to operate with forward recovery, and the last decision mechanism will only select the output from the nodes that have produced a pass result.

The definitions of PR and failure rate are as follows:

- PR is defined as the fraction or percentage of successful virtual nodes in the system after executing a complete computing cycle.
- Failure rate is defined as the level or rate at which the virtual node of the system fails. The failure rate of the system depends on time, status check, and TTLC. Details of this are presented mathematically in the next section.

4.3. Fault tolerance using checkpointing mechanism

Checkpointing strategies have drawn a significant attention over the last couple of years in the context of FT research in cloud computing [55].They have been explored for a large-scale cloud environment. Checkpointing mechanism is the process of saving a system state periodically to a stable storage during failure-free execution. Being the most common mechanism for FT in a cloud

DOI: 10.1002/spe

environment, we integrate it with the PR-optimized selection technique and have focused on it in this paper. Overall, it can be classified into two main types, namely, full checkpointing mechanism, which saves the entire system running state periodically to a storage platform, and the incremental checkpointing mechanism whose first checkpoint contains the running state of the complete system, while the subsequent checkpoint only saves pages that have been modified since the previous checkpoint [9].

In order to realize a high level of FT in cloud and to achieve an optimal level of cloud serviceability and cloud service-level agreement, we present a mathematical proof for FT strategy based on our model (Figure 4). Table III presents the key parameters of the model.

According to Figure 4, the time interval between consecutive checkpoints ΔJ is a critical factor to trade off checkpointing overhead T_{OV} and FT overhead, which relates to the checkpointing overhead during the longest failure-free time interval of the consecutive checkpoints, rollback time T_{Rol}, and the time interval between the failure point and the last successful checkpoint after system recovery T_{Rec}. So the failure density function (FDF) is given as $f(t)$, while the checkpoint density function is given as $\rho(t)$.

The scheme uses a checkpoint model that follows a reward renewal process, where after each failure occurring in the system, backward recovery is performed and the application is immediately restarted and recovered from the last successful checkpoint. In summary, the fault generated is repaired before the last task time deadline is reached, and after each node failure occurrence in

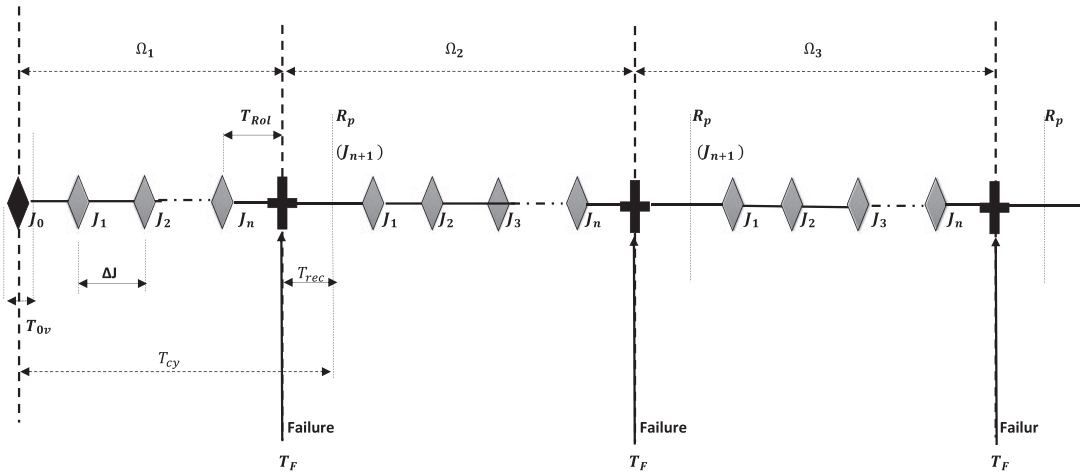


Figure 4. Full checkpointing strategy failure cycle.

Table III. Parameters of the fault tolerance model

$\mathbf{\Omega}_i$	The cycle between failure i and failure(i + 1)
$\mathbf{J}_n$	Time of the nth checkpoint
T_{Rol}	Rollback failure time between T_F and the last successful $\mathbf{J}_n$ checkpoint
R_p	Restart point
T_{Rec}	Recovery point/time or fault tolerance overhead
T_F	Failure occurrence point/time
T_{cy}	Total time interval of a complete failure cycle
T_{OV}	Checkpoint overhead
ΔJ	Time interval between consecutive checkpoint
CDF	Continuous density function
$F(t)$	Failure distribution function
$f(t)$	Failure density function
T_{OVFF}	Longest failure-free checkpoint overhead
$\rho(t)$	Checkpoint density function

DOI: 10.1002/spe

the system, the application will be restarted from the last successful checkpoint. The following assumptions were made.

Assumption 1

Let ($\boldsymbol{T_{ov}}$, $\boldsymbol{T_{rec}}$, $\boldsymbol{T_{Rol}}$) of each cycle be a sequence of independent identically random variable ($\boldsymbol{L}_1$, $\boldsymbol{L}_2, \boldsymbol{L}_3$), which is dependent on any point when time failure occurs in the system $\boldsymbol{\Omega}$, which stands for the k^{th} time between failures in each computing cycle.

$$E\ [T_{ov}\ ]\ <\ \infty. \tag{3}$$

To improve the checkpoint mechanism in our system, we looked at how to determine checkpoint intervals that decrease the time delay because a checkpoint should not be carried out too regularly to balance $\boldsymbol{T_{ov}}$ and rollback time of our application.

Therefore, the time delay can be expressed as follows:

$$L_t\ = \sum_{i=0}^{x_t} T_i, \tag{4}$$

where $x_t\ = \sup\{n : J_n \le t\} = \max\{n \in (1, 2, 3 \ldots\ .) | x_t \le t\}$.

And J_n refers to the k^{th} failure time of intervals $[J_n, J_{n+1}\ ]$, which is also called the renewal intervals, which is defined as follows:

$$J_n\ = \sum_{i=1}^{n} T_i, \tag{5}$$

Equation (4) is called the renewal reward process where L_t depends on ($\boldsymbol{T_{ov}}$, $\boldsymbol{T_{rec}}$, $\boldsymbol{T_{Rol}}$).

The renewal function is defined as the expected value of the number of failures observed up to a given time t:

$$f(x) = E\ [X_t\ ]. \tag{6}$$

So the renewal function satisfies

$$\lim_{t\to\infty}\ \frac{1}{t} f(x) = \frac{1}{\mathrm{E}\ [\boldsymbol{\Omega}_1]}. \tag{7}$$

Substituting (6) into (7) gives

$$\lim_{t\to\infty}\ \frac{X_t}{t} = \frac{1}{\mathrm{E}\ [\boldsymbol{\Omega}_1]}. \tag{8}$$

Proving the elementary renewal theorem, it is sufficient to show that for an elementary renewal theorem for renewal reward processes, the reward function is given as follows:

$$g(x) = \mathrm{E}\ [L_t\ ]. \tag{9}$$

The reward function thereby satisfies

$$\lim_{t\to\infty}\ \frac{1}{t} g(x) = \frac{\mathrm{E}\ [\mathbf{L}_1]}{\mathrm{E}\ [\boldsymbol{\Omega}_1]}. \tag{10}$$

DOI: 10.1002/spe

Substituting (9) into (10) gives

$$\lim_{t\to\infty} \frac{E\left[\mathbf{L}_t\right]}{t} = \frac{E\left[\mathbf{L}_1\right]}{E\left[\mathbf{\Omega}_1\right]}. \tag{11}$$

From Equation (4),

$$L_t = \sum_{i=0}^{x_t} T_i. \tag{12}$$

Then Equation (11) becomes

$$\lim_{t\to\infty} \frac{E\left[\sum_{i=0}^{x_t} T_i\right]}{t} = \frac{E\left[\mathbf{L}_1\right]}{E\left[\mathbf{\Omega}_1\right]}. \tag{13}$$

Therefore,

$$L_t = \frac{E\left[\mathbf{L}_1\right]}{E\left[\mathbf{\Omega}_1\right]}, \tag{14}$$

where L_t is called the renewal reward process as derived in [56, 57]. Conversely, there is an additional time to save the system application states, which is called the checkpoint overhead. In order to improve the checkpoint mechanism, checkpoints should not be performed too frequently in order to achieve balancing between the checkpoint overhead, recovery time, and application recomputing time as derived in [58, 59, 55], [60].

Assumption 2

In the proposed model, we assume that failures occur rarely and randomly to the system, rather than being an integral part of the system. The checkpointing mechanism is able to recognize and isolate faults when they occur to ensure the overall system performance is not affected.

Assumption 3

We assume that failure will be detected as soon as possible after the occurrence and the time between failures follows a similar probability density function in cloud systems. At the same time during system recovery period, failure will not happen.

Assumption 4

The system is failure free during system recovery period.

Assumption 5

The time between failures follows the same probability density function in a cloud environment.

Assumption 6

That ΔJ is constant, which implies that $T_{Rol} < \Delta J$ always.

DOI: 10.1002/spe

Assumption 7

T_{OV} is constant in a cloud environment.

Assumption 8

The failed system can always be recovered from the last successful checkpoint, which implies that $T_{rec} < \Delta J$ always.

From the figure previously, T_{ovFF} is defined as the checkpointing overhead during the longest failure-free time interval of consecutive checkpoint, and it is associated with the T_{ov}.

So

$$T_{ovFF} = \mathbf{\Omega}_1 - T_{Rol}. \tag{15}$$

We define a continuous density function (CDF) of a continuous checkpoint as a function that describes the probability for a checkpointing interval ΔJ to occur at a particular time t:

$$CDF = \rho(t) = \frac{1}{\Delta J}, \tag{16}$$

where $N_{j_x j_y}$ is the number of checkpoints to fall within a particular interval $[j_x, j_y]$, which is given by the integral of CDF over time interval $[j_x, j_y]$.

Therefore, integrating (16) becomes

$$\int_{j_x}^{j_y} \rho(t)\, d\tau = \int_{j_x}^{j_y} \frac{1}{\Delta J}\, d\tau = \quad N_{j_x j_y}. \tag{17}$$

Because $N_{j_x j_y}$ is the number of checkpoints to fall within an interval $[j_x, j_y]$, then $N_{j_{n-1} j_n}$ is the number of checkpoints to fall within an interval $[j_{n-1}, j_n]$.

From (17), we have

$$\int_{j_{n-1}}^{j_n} \rho(t)\, d\tau = \int_{j_{n-1}}^{j_n} \frac{1}{\Delta J}\, d\tau = \int_{j_{n-1}}^{j_n} \quad \frac{1}{j_n - j_{n-1}}\, d\tau = \quad N_{j_{n-1} j_n} = 1 \ . \tag{18}$$

So from the figure previously, we calculate the total checkpoint overhead during the longest failure-free time interval as follows:

$$T_{ovFF} = \quad T_{ov} * \ N_{j_0 j_n}, \tag{19}$$

$$T_{ovFF} = \quad T_{ov} * \int_{j_0}^{j_n} \rho(t)\, d\tau. \tag{20}$$

Because T_{Rol} is connected to T_F and in reality, T_F is unknown until failure occurs, therefore, we use failure expectation distribution value $E(T_{Rol})$ as the fault overhead.

If $f(t)$ represents the FDF of a continuous failure whose function describes the relative probability for the failure to occur at a particular time t, then we define the FDF as follows:

$$f(t) = F^{'}(t) = \frac{dF(t)}{dt}. \tag{21}$$

Such that $f(t) \geq 0$ and $F(t)$ represent the failure distribution function, which is connected to the FDF. We now have

$$\int_{-\infty}^{+\infty} f(\tau) d\tau = 1. \tag{22}$$

DOI: 10.1002/spe

If $F(t)$ of a continuous failure is defined as a function that describes random variable t with a given FDF $f(t)$, where $f(t) \leq t$, then

$$F(t) = P(-\infty \leq t) = \int_{-\infty}^{t} f(\tau) d\tau. \tag{23}$$

Simplifying Equation (23) gives

$$P(-\infty \leq t) = 1 - F(t) = \int_{t}^{-\infty} f(\tau) d\tau. \tag{24}$$

So

$$P\left(t_x < t \leq t_y \right) = F\left(t_y \right) - F\left(t_x \right). \tag{25}$$

Equation (25) now gives

$$= \int_{t_x}^{t_y} f(\tau) d\tau, \text{Where } \lim_{t \to \infty} F(t) = 0 \text{ and } \lim_{t \to +\infty} F(t) = 1. \tag{26}$$

Recall that because T_{Rol} is connected to T_F and in reality, T_F is unknown until failure occurs, therefore, we use failure expectation distribution value $E(T_{Rol})$ as the fault overhead. We then calculate the failure expectation value as follows:

$$E(t) = \int_{-\infty}^{+\infty} \tau . f(\tau) d\tau, \tag{27}$$

where $E(t)$ is defined as the failure distribution value of a continuous failure that describes the weighted average of all values of all possible failures that accept probability density function.

Because (FT_{OV}) is associated with T_F, and T_F falls within time interval $[J_n, J_{n,+1}]$, therefore, we can simply calculate T_F and J_n as well as the rollback failure time between T_F and the last successful $\boldsymbol{J}_n$ checkpoint, which is given as follows:

$$T_{Rol} = T_{Fi} - J_n. \tag{28}$$

From our checkpoint model, the checkpointing time interval in a cycle falls within an interval $[J_1, J_{n,+1}]$; breaking it down further gives $[J_1, J_2, J_3, \ldots, J_{n,+1}]$ and $[J_n, J_{n,+1}]$ where T_F falls between time interval $[J_n, J_{n,+1}]$.

So let $(J_n < T_F \leq J_{n+1} = X)$.

Therefore, from (27), the failure expectation distribution value gives

$$E(T_{Rol}|X) = \frac{1}{2 * \rho(t)}. \tag{29}$$

Substituting X into (29) gives

$$E(T_{Rol}|J_n < T_F \leq J_{n+1})) = \frac{1}{2 * \rho(t)}. \tag{30}$$

Because T_F falls within time interval $[J_n, J_{n,+1}]$, which implies that $J_n < T_{F,} \leq J_{n,+1}$, and t falls within $[J_n, T_F]$, the failure expectation value by substituting (28) gives us the following:

$$E(T_{Rol}|J_n < T_F \leq J_{n+1}) = E(T_{Fi} - J_n | J_n < T_F \leq J_{n+1}). \tag{31}$$

From (27), integrating gives

$$E(T_{Rol}|J_n < T_F \leq J_{n+1}) = \int_0^{J_{n+1} - J_n} P(\tau > T_{Fi} - J_n | J_n < T_{Fi} \leq J_{n+1}) d\tau. \tag{32}$$

DOI: 10.1002/spe

Simplifying (32) gives

$$E(T_{Fi} - J_n | J_n < T_F \leq J_{n+1}) = \int_0^{J_{n+1}-J_n} \frac{P(\tau > T_{Fi} - J_n, J_n < T_{Fi} \leq J_{n+1})}{P(J_n < T_{Fi} \leq J_{n+1})} d\tau, \tag{33}$$

$$E(T_{Fi} - J_n | J_n < T_F \leq J_{n+1}) = \int_0^{J_{n+1}-J_n} \frac{F(J_n + \tau) - F(J_n)}{F(J_n + 1) - F(J_n)} d\tau, \tag{34}$$

$$E(T_{Fi} - J_n | J_n < T_F \leq J_{n+1}) = \int_0^{J_{n+1}-J_n} \frac{\int_{J_n}^{J_{n+\tau}} f(x)dx}{\int_{J_n}^{J_{n+1}} f(x)dx} d\tau. \tag{35}$$

Failure rate is the frequency with which an engineered system or component fails, expressed in failures per unit of time. It can be defined with the aid of the reliability function, also called the survival function $R(t)$, the probability of no failure before time t.

$$\lambda(t) = \frac{f(t)}{R(t)}, \tag{36}$$

where $f(t)$ is the FDF of a continuous failure, which is related to the failure rate λ and not to time, because the system failure rate λ does not change in the time interval $[J_n, J_{n,+1}]$, and

$$R(t) = 1 - f(t). \tag{35}$$

Note that the $\lambda(t)$ function is a conditional probability of the FDF. The condition is that the failure has not occurred at time t.

Hence, Equation (35) becomes

$$E(T_{Fi} - J_n | J_n < T_F \leq J_{n+1}) = \int_0^{J_{n+1}-J_n} \frac{\int_{J_n}^{J_{n+\tau}} f(t)dx}{\int_{J_n}^{J_{n+1}} f(t)dx} d\tau. \tag{38}$$

Simplifying further gives

$$E(T_{Fi} - J_n | J_n < T_F \leq J_{n+1}) = \int_0^{J_{n+1}-J_n} \frac{(J_n + \tau - J_n) * f(t)}{(J_{n+1} - J_n) * f(t)} d\tau, \tag{39}$$

$$E(T_{Fi} - J_n | J_n < T_F \leq J_{n+1}) = \int_0^{J_{n+1}-J_n} \frac{\tau}{(J_{n+1} - J_n)} d\tau. \tag{40}$$

Factoring out and integrating *wrt* to τ give

$$E(T_{Fi} - J_n | J_n < T_F \leq J_{n+1}) = \int_0^{J_{n+1}-J_n} \frac{\tau}{(J_{n+1} - J_n)} d\tau, \tag{41}$$

$$E(T_{Fi} - J_n | J_n < T_F \leq J_{n+1}) = \frac{1}{(J_{n+1} - J_n)} \int_0^{J_{n+1}-J_n} \tau d\tau, \tag{42}$$

$$= \frac{1}{(J_{n+1} - J_n)} * \frac{(J_{n+1} - J_n)^2}{2} = \frac{(J_{n+1} - J_n)}{2}. \tag{43}$$

Therefore,

$$E(T_{Rol} | J_n < T_{Fi} \leq J_{n+1}) = \frac{(J_{n+1} - J_n)}{2}. \tag{44}$$

DOI: 10.1002/spe

Substituting $\Delta J=(J_{n+1}-J_n)$ into (44) gives

$$E(T_{Rol}|J_n < T_{Fi} \leq J_{n+1}) = \frac{(J_{n+1}-J_n)}{2} = \frac{\Delta J}{2}. \tag{45}$$

Substituting (16) into (45) gives

$$E(T_{Rol}|J_n < T_{Fi} \leq J_{n+1}) = \frac{1}{2*\rho(t)}. \tag{46}$$

From Figure 4 previously, the total overhead time interval of the complete failure cycle can be calculated as follows:

$$T_{cy} = T_{OVFF} + T_{Rol} + T_{rec}, \tag{47}$$

where

$$T_{rec} = T_{Rol} + T_{OV}. \tag{48}$$

Substituting (30) into (48) gives

$$T_{rec} = \frac{1 + 2*\rho(t)*T_{OV}}{2*\rho(t)} = \frac{1}{2*\rho(t)} + T_{OV}. \tag{49}$$

Substituting (20), (46), and (49) into (47) gives

$$T_{cy} = \ T_{ov}*\textstyle\int_{j_0}^{j_n} \rho(t)\, d\tau + \dfrac{1}{2*\rho(t)} + \dfrac{1}{2*\rho(t)} + T_{OV}. \tag{50}$$

Simplifying (50) and factoring out T_{OV} give

$$T_{cy} = \ T_{ov}*\left(1 + \textstyle\int_{j_0}^{j_n} \rho(\tau)\, d\tau\right) + \dfrac{1}{\rho(t)}. \tag{51}$$

The failure expectation distribution value $E(T_{cy})$ can be calculated because the time interval in our failure circle is $[J_0,J_{n,+1}]$, and $f(t)$ and $F(t)$ are the FDF and failure distribution function, respectively. So

$$E\left(T_{cy}\right) = \textstyle\int_0^{+\infty} T_{cy}*f(t)\, dt. \tag{52}$$

Integrating *wrt t* and substituting (51) into (52) give

$$E\left(T_{cy}\right) = \textstyle\int_0^{+\infty} \left(T_{cy}*\left(1 + \int_{j_0}^{j} \rho(\tau)\, d\tau\right) + \dfrac{1}{\rho(t)}\right)*f(t)\, dt, \tag{53}$$

where $E(T_{cy})$ is the failure expectation distribution value.

Minimizing the value of $E(T_{cy})$ from Equation (51), $\rho(t)$ can be obtained thereby optimizing ΔJ, which is the checkpoint interval.

So $\rho(t)$ is obtained as follows:

$$= \left(\frac{1}{T_{ov}} * \frac{f(t)}{(1-F(t))}\right)^{\frac{1}{2}}. \tag{54}$$

And ΔJ is obtained as follows:

$$= \left(\ T_{ov}* \frac{(1-F(t))}{f(t)}\right)^{\frac{1}{2}}\ . \tag{55}$$

DOI: 10.1002/spe

Therefore, minimizing ($\min E(T_{cy})$) is equivalent to

$$= \left(\left(T_{cy} * \left(1 + \int_{j_0}^{j} \rho(\tau)\, d\tau \right) + \frac{1}{\rho(t)} \right) * f(t)\, dt \right). \tag{56}$$

The sequence of interactions between components of the cloud using our proposed strategy as shown in Figure 5 is as follows:

1. At the start, users submit a task with user QoS requirements to the SP and dispatch tasks to the CC.
2. CC sends request to the CIS to get a list of available resources for each task.
3. CIS responds to this query by sending a list of registered resources that are suitable for executing the job and their information (CIS table).
4. After receiving the available list of resources, the CC performs the following:
   (a) Performs virtualization with the help of a hypervisor.
   (b) VMs (nodes) are created from the available resources of the physical server.
   (c) A CIS table containing the server IDs, virtual nodes, and PRs is made available to the load balancer. This table is maintained to identify the virtual nodes and to keep record of the number of times jobs are assigned and to also obtain the PR from those successful virtual nodes.
5. The load balancer distributes the task based on the information obtained from the CIS table, by assigning a task to those virtual nodes whose corresponding physical servers are having a good PR.
6. If the job is successfully completed then.
   (a) Both the SC and TTLC are a success.
   (b) The PR of corresponding node is increased and forwarded to the final decision mechanism module.
   (c) The decision mechanism delivers the result of a successful job to the SP, which is then dispatched or returned to the client.
7. If it fails to complete the job then,
   (a) Both SC and TTLC fail.
   (b) Either SC or TTLC fails, and the corresponding VM is not sent to the decision mechanism.

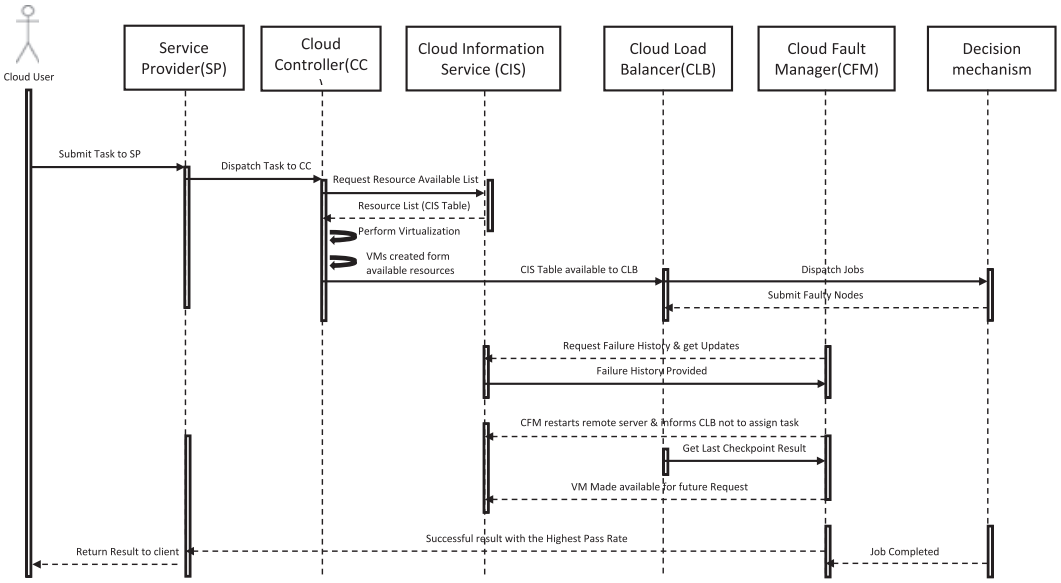


Figure 5. A sequence diagram for the interaction between the proposed system components.

DOI: 10.1002/spe

(c) The corresponding VM is sent to fault manager for fault detection and recovery. If all nodes fail, then the backward recovery is performed with the help of the last checkpoint.

Algorithm 1: Cloud Controller Computation Algorithm

Step 1: **Start**
Step 2: **Output** "Most Viable Node for operation with highest PR"
Step 3: **Input** "highestPassRate = 1"
Step 4: **Input** PR = 0.5 , $x_1 = 1$, $x_2 = 2$.
Step 5: **If** (nodeStatus = Pass), /SC and TTLC module for that node is Pass/
{$x_1 = x_1 + 1$, $x_2 = x_2 + 1$, PR = x_1/x_2
Update CIS record table} else,
Step 6: **If** (nodeStatus=Fail), /SC or TTLC module for that node or both is Fail/
{$x_2 = x_2 + 1$, PR = x_1/x_2Update CIS record table}
Step 7: **If** (PassRate > = highestPassRate)
{PassRate = highestPassRate}
Step 8: **If**(PassRate < = 0)
{Inform CLB not to assign task to the node, remove the node and CC will be informed to add a new node}
Step 9: **Stop**

Algorithm 2: Cloud Load Balancer Algorithm

Step 1: **Start**
Step 2: **Input** initial PR=0.5, x_1=1, x_2=2. ($0 < PR \leq 1$)
Step 3: **Input** "highestPassRate $= 1''$, PR $= \frac{x_1}{x_2}$
x_1 = Number of times the virtual node of particular physical server gives a succesfull result
x_2 = Number of times the CLB of the CC assigns a task to a particular servers virtual node
Step 4: **If** (if SC Status = = Pass) & & (TTLC status = = Pass)
Select the node
If (if SC Status = = Pass) & & (TTLC status = = Fail)
Select the node with the highest PR
If (if SC Status = = Fail) & & (TTLC status = = Pass)
Don't select the node if enough nodes are available
If (if SC Status = = Fail) & & (TTLC status = = Fail)
inform the CC and forward to CFM to perform recovery with the last successful checkpoint
Step 5: **Stop**

Algorithm 3: Final Selection Technique Algorithm

Step 1: **Start**
Step 2: **Output** "Select best Node with highest PR and minimum finish time"
Step 3: **Input** from TTLC: node PassRate, x_1= is the number of nodes with successful SC and TTLC results.
Step 4: **Input** PR = 0.5,
Step 5: **Input** highestPassRate
Step 6: **if** (x_1==0), then {Status = fail, Conduct a backward recovery with the help of the last successful checkpoint} else, {Status=Pass, bestPassRate=PassRate of the node with maximum PassRate and send outcome to the Service provide and perform checkpoint}
Step 7: **Stop**

DOI: 10.1002/spe

4.4. *Pass rate and fail rate assessment analysis*

We considered 200 computing cycles and present a metric analysis to evaluate the pass and fail scenarios of six virtual nodes, respectively. As part of our initial conditions, we assumed the following:

(i) Pass rate = 0.5, where x_1 represents the number of times a VM of host produces a pass outcome and x_2 represents the time the CC's load balancer designates a task to a virtual node.
(ii) Each VM belongs to a different host or physical server.

A comparison analysis performed for 200 computing cycles between the pass and failure scenarios is presented. We observe that scenario 1 continuously increased and passed successfully, while scenario 2 continuously decreased as shown in Figure 6. Scenario 3 passed and succeeded for the first 100 cycles and then decreased for the remaining cycles, while scenario 4 failed for the first 100 cycles and then succeeded for the remaining 100 cycles. The increase in PR after 200 computing cycles is 0.978 for scenario 1, whereas the decrease in PR for scenario 2 is 0.579, with the increase in PR for scenarios 3 and 4 being 0.38. This shows that the increase in PR is greater than the decrease, and the convergence towards decrement in reliability is much higher, displaying a good performance of algorithm. Further scenarios in a more complex environment contain a higher number of VMs and were tested for validation of result. This is discussed is Section 5.

5. EXPERIMENTAL SETUP AND RESULTS

The experiments were conducted using CloudSim [61–64] where 100 virtual nodes were created for performance comparison and scalability validation. We started by running the integrated virtualized optimal checkpointing algorithm using 10 computing cycles and created six virtual nodes with every individual node executing a series of tasks at a time. While these tasks are executed in one computing cycle, every virtual node runs a diverse algorithm. We then compared our results with existing approaches by creating three virtual nodes with 10 computing cycles as depicted in Section 5.1. To analyze the algorithm's performance in a larger and complex environment, we created 100 virtual nodes. Details of the simulation results are presented in the next section.

As earlier stated, the different pass and failure scenarios obtained from this experiment are a result of diversity in software and timing constraints. The selection or decision mechanism is responsible for receiving results obtained from the VMs before returning the result of the successful job to the client via the SP. At the SP level, the selection or decision mechanism is integrated with the CC module. In a situation where a failure occurs in one of the nodes, the system will automatically adapt a failover strategy and continue operating using the remaining nodes. The system will

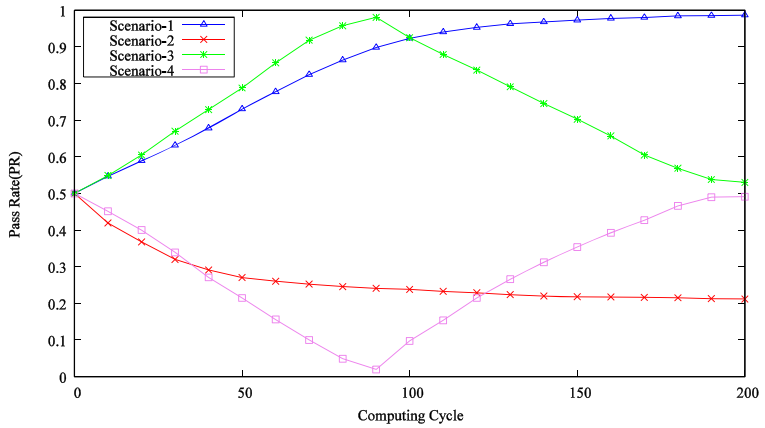


Figure 6. Pass/pass⇛fail and fail/fail⇛pass shifting plots. [Colour figure can be viewed at wileyonlinelibrary.com]

DOI: 10.1002/spe

maintain and continue its operation in a steady state until all nodes have failed. A node is then selected, and a checkpoint of the last successful saved point is made to keep the status of the system for future recovery. This is performed after a successful completion of one computing or instruction cycle. The approach assumes that the value of x_1, x_2 PR, virtual node ID, and corresponding server ID is available. The task deadlines are taken as input with initial values $x_1 = 1, x_2 = 2$, and PR = 0.5

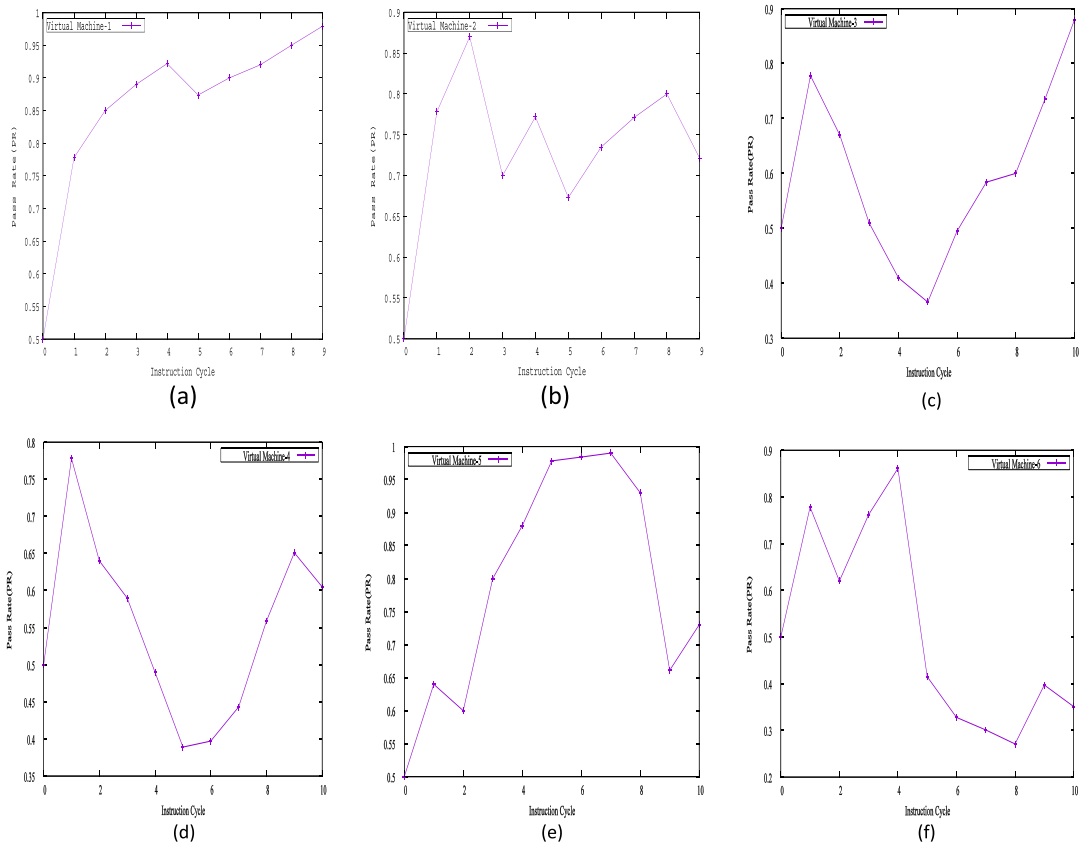


Figure 7. Experimental result from six virtual machines. [Colour figure can be viewed at wileyonlinelibrary.com]

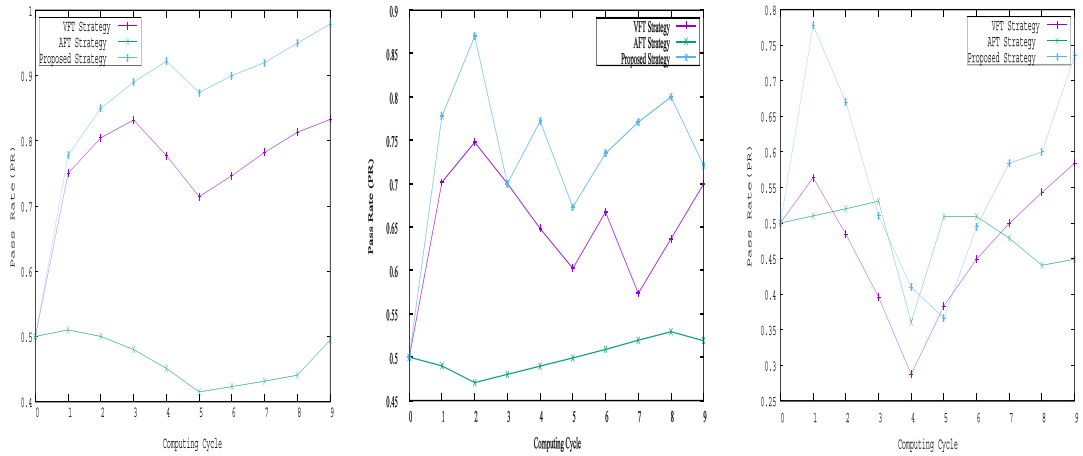


Figure 8. Performance comparison of virtualization and fault tolerance approach (VFT), adaptive fault tolerance approach (AFT), and our proposed strategy. [Colour figure can be viewed at wileyonlinelibrary.com]

DOI: 10.1002/spe

considered for every node. Figure 7 shows some experimental results obtained from six VMs for pass and failure rate analysis.

5.1. Performance comparison of results

Figure 8 presents results obtained from the performance comparison of our proposed strategy with other existing approaches where we observed that our proposed strategy has a better performance compared with the existing approaches, as the increase in PR for virtual nodes 1, 2, and 3 is higher than decrease in failure rate.

1. Virtualization and FT approach (VFT) – Das *et al*. [50] proposed a virtualization and FT technique to reduce the service time and increase the system availability. In their proposed approach, cloud manager and decision maker modules was introduced which they used to manage virtualization and load balancing as well as fault handling in the system. By performing virtualization and load balancing, FT was achieved by redundancy and fault handlers. Their technique was mainly designed to provide a reactive FT where the fault handler prevents the unrecoverable faulty nodes from having an adverse effect.
2. Adaptive FT approach (AFT) – Malik *et al*. [6] proposed an adaptive FT in time cloud computing where the main essence of their proposed technique was an adaptive behavior of the reliability weights assigned to each processing node and adding and removing of nodes on the basis of reliability.
3. Our proposed approach – For the purpose of evaluation, we compared our proposed strategy with the VFT [50] and AFT strategies [6] where we use results obtained from our proposed strategy as the measured parameter, while that of VFT and AFT are referred to as calculated parameters, respectively.

Comparing the three models, we first obtain the relative error x_{re}, and then we calculated the actual error x_i as the difference between the calculated and measured result. These are expressed as (57) and (58), respectively.

$$x_{re} = \left(\frac{q_{iCalculated} - q_{iMeasured}}{q_{iMeasured}} \right) \times 100, \tag{57}$$

$$x_i = q_{iCalculated} - q_{iMeasured}, \tag{58}$$

$$e_1 = \left[\frac{1}{N} \sum_{i=1}^{N} x_{re} \right]. \tag{59}$$

A performance evaluation of VFT and AFT strategies with our proposed model was carried out using the parameters in Table IV.

Table IV. Parameters

Parameter	Meaning
x_{re}	Relative error
x_i	Actual error
e_1	Average percentage relative error
e_2	Average absolute percentage relative error
e_3	Standard deviation
e_4	Average actual error
e_5	Average absolute actual error
e_6	Standard deviation about average actual error

DOI: 10.1002/spe

Equations (59)–(64) give the mathematical definition of these parameters, and Table 5 presents the experimental results.

$$e_2 = \left[\frac{1}{N} \sum_{i=1}^{N} |x_{re}| \right], \tag{60}$$

$$e_3 = \sqrt{\frac{\sum_{i=1}^{N} (x_{re} - e_1)^2}{N-1}}, \tag{61}$$

$$e_4 = \frac{1}{N} \sum_{i=1}^{N} x_i, \tag{62}$$

$$e_5 = \frac{1}{N} \sum_{i=1}^{N} |x_i|, \tag{63}$$

$$e_6 = \sqrt{\frac{\sum_{i=1}^{N} (x_i - e_4)^2}{N-1}}. \tag{64}$$

Figure 9(a) shows the mean error comparison, pass, and failure rate analysis between our proposed approach and existing approaches as well as the error bar plot for some of the virtual nodes. Under VM-1, the PR mean value of our proposed strategy was 0.85, while that of VFT and AFT are 0.75 and 0.46, respectively. It was observed that VFT performed better than AFT, which could be attributed to a better algorithm used by VFT. This also shows this limitation of the AFT algorithm. In VM-2, the difference between the mean value obtained for both VFT and AFT is less significant because VFT is slightly higher than the later. But our proposed strategy under this VMs displayed an improved performance. While in VM-3, AFT is slightly higher than VFT even though the result obtained is less significant, but most importantly, our proposed strategy shows a better result.

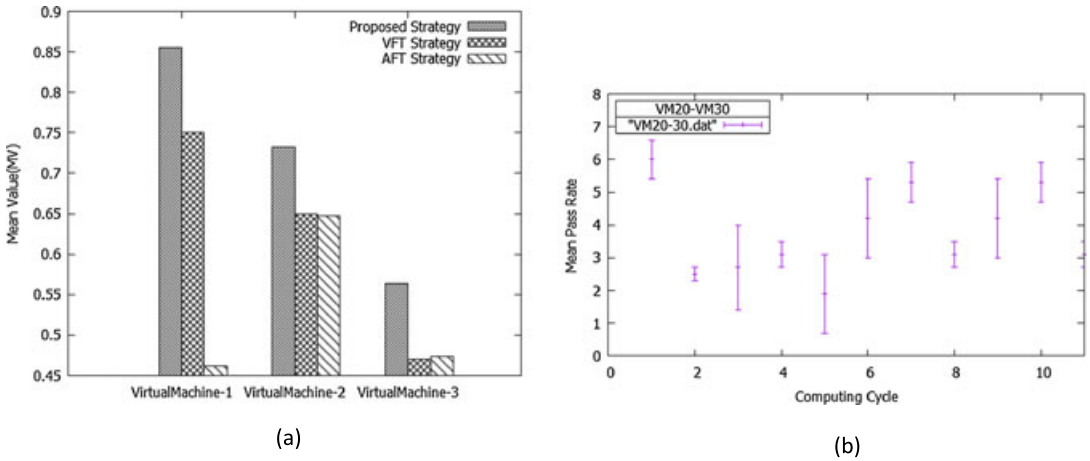


Figure 9. Mean error value comparison and error bars plot. VFT, virtualization and fault tolerance approach; AFT, adaptive fault tolerance approach. [Colour figure can be viewed at wileyonlinelibrary.com]

DOI: 10.1002/spe

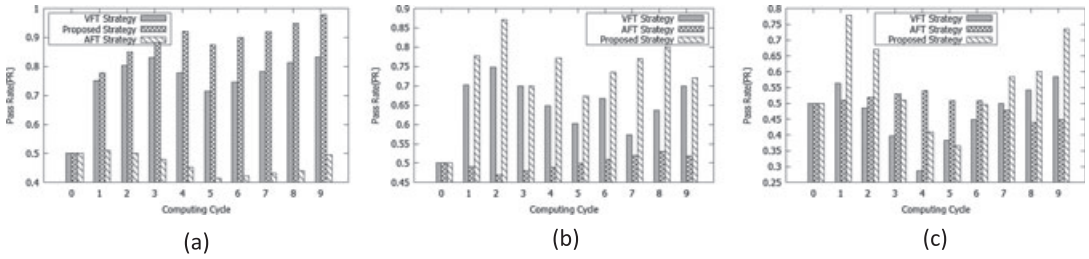


Figure 10. Success rate analysis of virtual machines. VFT, virtualization and fault tolerance approach; AFT, adaptive fault tolerance approach.

Table V. Virtual node 1 performance comparison of VFT and AFT with our proposed strategy

Virtual node 1 comparison metrics Data sources	$\varepsilon_1(\%)$	$\varepsilon_2(\%)$	$\varepsilon_3(\%)$	$\varepsilon_4(\%)$	$\varepsilon_5(\%)$	$\varepsilon_6(\%)$
VFT	−11.0812	11.08117	25.83521	−0.10103	0.101032	0.308417
AFT	−43.4512	43.45116	60.19807	−0.39175	0.391750	0.297314

VFT, virtualization and fault tolerance approach; AFT, adaptive fault tolerance approach.

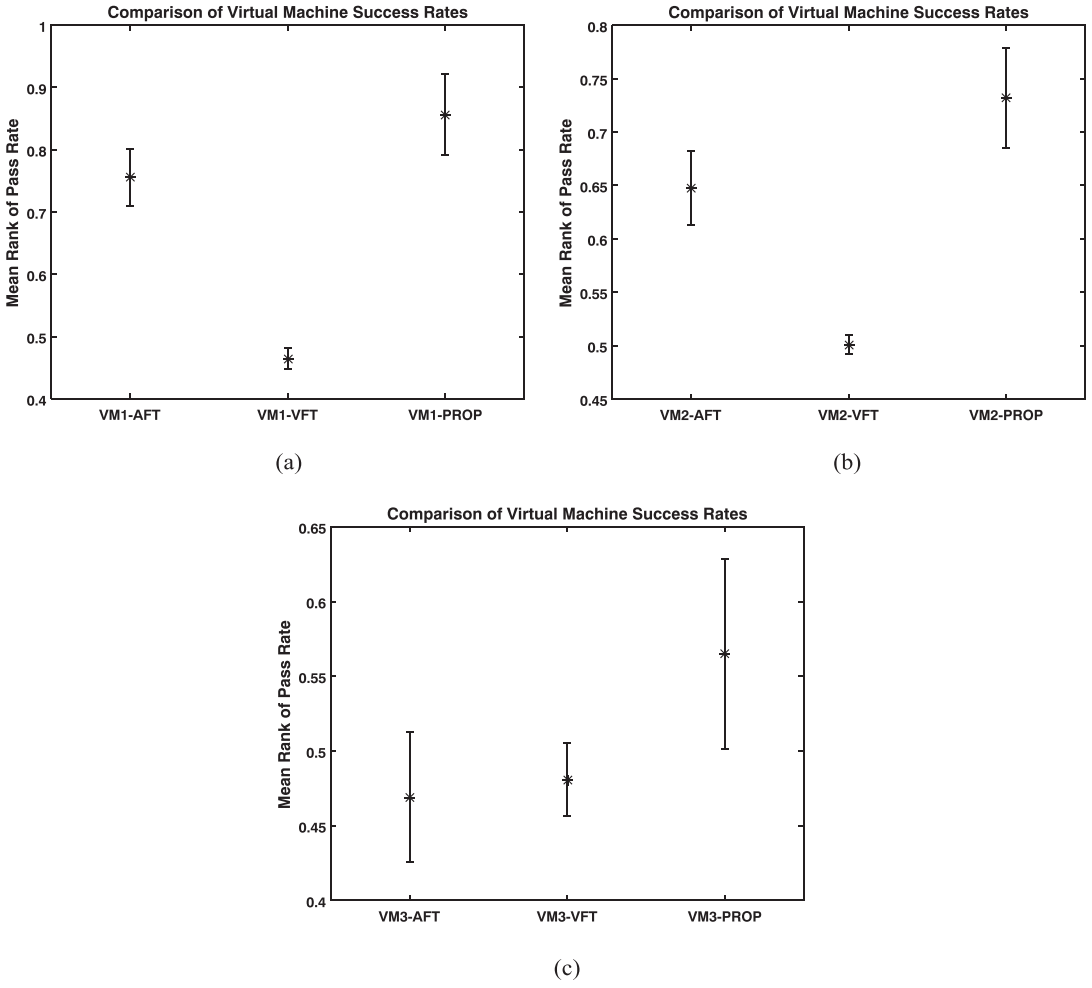


Figure 11. Mean rank of pass and failure rate with standard deviation (error bars). VFT, virtualization and fault tolerance approach; AFT, adaptive fault tolerance approach.

DOI: 10.1002/spe

Overall, this indicates that our proposed strategy has a better output and improved performance compared with the existing approaches (Figure 10).

Based on the performance comparison analysis in Table V, the standard deviation for virtual node 1 under the VFT and AFT strategies was estimated to be 25.835 and 60.19807, respectively, while that of VM-2 was estimated to be 20.438 and 45.078. A further deviation of strategies was also noticed in VM-3. This shows the degree of deviation of the calculated result from the measured result. For the VFT algorithm, our results have shown that the calculated PR is lesser compared with the measured PR of our proposed strategy. While for the AFT algorithm, the calculated PR is far much lesser than both the VFT algorithm and our proposed strategy. Details of this model can be found in [50, 6].

A mean rank of success and failure rate of the VFT, AFT, and our proposed strategy was conducted, and the results are presented in Figure 11. We observed that in Figure 11(a), (b), and (c), there is no overlap between our proposed strategy and VFT, which shows the significance of errors that exist between our proposed model and the current existing approach.

6. DISCUSSIONS

We conducted experiments using three VMs for performance comparison with other existing approaches (as shown in Figure 8), Table V–VII also presents a virtual node comparison using 3 VMs then we performed another experiment using six VMs after which we then performed the experiment using 100 VMs, which are both on a larger scale compared with the initial setup. In the first cycle, VM-1, VM-2, VM-3, VM-4, VM-5, and VM-6 have the same PR, but the result of VM-1 (node 1) has been selected because it has a lower task finish service time of 1700. In the same cycle, VM-5 did not pass the status check, and the time task limit check also automatically failed. The VM-2 output was selected by the decision mechanism in the second cycle because it has the highest PR of 0.87, while from cycles 3 to 4, the output of VM-1 was selected, as it has the highest PR among the competing VMs. (Details of some experimental details are presented in Table VIII.)

From cycles 5 to 7, the output of VM-5 (node 5) was selected because it has the highest PR among all other nodes. In cycle 5, VM-4 and VM-6 do not pass the status check, and the time task limit check also failed, with the same occurring in cycle 6 under VM-6 and cycle 10 under VM-4. Lastly, the output of VM-1 was selected from cycles 8 to 10 because it has the highest PR (Table VIII).

In a similar scenario, cycles 5 and 6, where SC and TTLC failed or only SC failed, an error signal is generated and sent to the fault manager of the CC module and to the TTLC. Here, it is received before the time limit, but because no result was produced, TTLC status is also failed. The fault manager then tries to repair the fault generated by performing checkpoint. As stated earlier, the

Table VI. Virtual node 2 performance comparison of VFT and AFT with our proposed strategy

Virtual node 2 comparison metrics Data sources	$\varepsilon_1(\%)$	$\varepsilon_2(\%)$	$\varepsilon_3(\%)$	$\varepsilon_4(\%)$	$\varepsilon_5(\%)$	$\varepsilon_6(\%)$
VFT	−10.8510	10.85229	20.43801	−0.08423	0.08423	0.27140
AFT	−30.2022	30.20222	45.07832	−0.23141	0.23141	0.19480

VFT, virtualization and fault tolerance approach; AFT, adaptive fault tolerance approach.

Table VII. Virtual node 3 performance comparison of VFT and AFT with our proposed strategy

Virtual node 3 comparison metrics Data sources	$\varepsilon_1(\%)$	$\varepsilon_2(\%)$	$\varepsilon_3(\%)$	$\varepsilon_4(\%)$	$\varepsilon_5(\%)$	$\varepsilon_6(\%)$
VFT	−15.6587	16.5941	16.83521	−0.09576	0.09918	0.107865
AFT	−10.6525	19.8363	25.19807	−0.08409	0.11959	0.021855

VFT, virtualization and fault tolerance approach; AFT, adaptive fault tolerance approach.

DOI: 10.1002/spe

Table VIII. Simulation results for the pass rate assessment

		Virtual machine-1				Virtual machine-2				Virtual machine-3			
Cycle	Task time limit	Status check	TTLC	Task finish time	Pass rate	Status check	TTLC	Task finish time	Pass rate	Status check	TTLC	Task finish time	Pass rate
Start	0	0	0	0	0.500	0	0	0	0.500	0	0	0	0.500
1	1800	True	True	1700	0.778	True	True	1701.7	0.778	True	True	1701.7	0.778
2	1701	True	True	1700.5	0.850	True	True	1700.1	0.870	True	False	1720.3	0.670
3	1801	True	True	1800.7	0.899	True	False	1802.5	0.700	True	False	1810.8	0.510
4	1750	True	True	1701.1	0.922	True	True	1705.8	0.772	True	False	1780	0.410
5	1700	True	False	1709.8	0.874	True	False	1707.3	0.6731	True	False	1703.6	0.366
6	1890	True	True	1706	0.900	True	True	1702.4	0.735	True	True	1723	0.495
7	2001	True	True	1400.4	0.920	True	True	1706.9	0.771	True	True	1759.4	0.584
8	1900	True	True	1712.1	0.950	True	True	1715.3	0.800	True	True	1821.7	0.600
9	1810	True	True	1722.2	0.979	False	False	0	0.7 21	True	True	1792.1	0.735
10	1899	True	True	1764.5	0.994	True	True	1806.4	0.888	True	True	1656.2	0.879

		Virtual machine-4				Virtual machine-5				Virtual machine-6				
Cycle	Task time limit	Status check	TTLC	Task finish time	Pass rate	Status check	TTLC	Task finish time	Pass rate	Status check	TTLC	Task finish time	Pass rate	Virtual machine or node selected
Start	0	0	0	0	0.500	0	0	0	0.500	0	0	0	0.500	0
1	1800	True	True	1750	0.778	False	False	1900	0.640	True	True	1720	0.778	Node 1
2	1701	True	False	1790	0.640	True	False	0	0.600	True	True	1700	0.620	Node 2
3	1801	True	False	1909	0.590	True	True	1650	0.800	True	True	1740	0.762	Node 1
4	1750	True	False	0	0.490	True	True	1670	0.880	True	True	1721	0.861	Node 1
5	1700	False	False	0	0.389	True	True	1600	0.978	False	False	1670	0.415	Node 5
6	1890	True	True	1724	0.397	True	True	1639	0.984	False	False	1900	0.328	Node 5
7	2001	True	True	2269	0.443	True	True	1740	0.990	True	False	2331	0.301	Node 5
8	1900	True	True	1799	0.559	True	True	1800	0.930	True	False	0	0.271	Node 1
9	1810	True	True	1700	0.651	True	False	1890	0.661	True	True	1500	0.397	Node 1
10	1899	False	False	1950	0.605	True	True	1822	0.730	True	False	1900	0.351	Node 1

TTLC, task time limit check.

DOI: 10.1002/spe

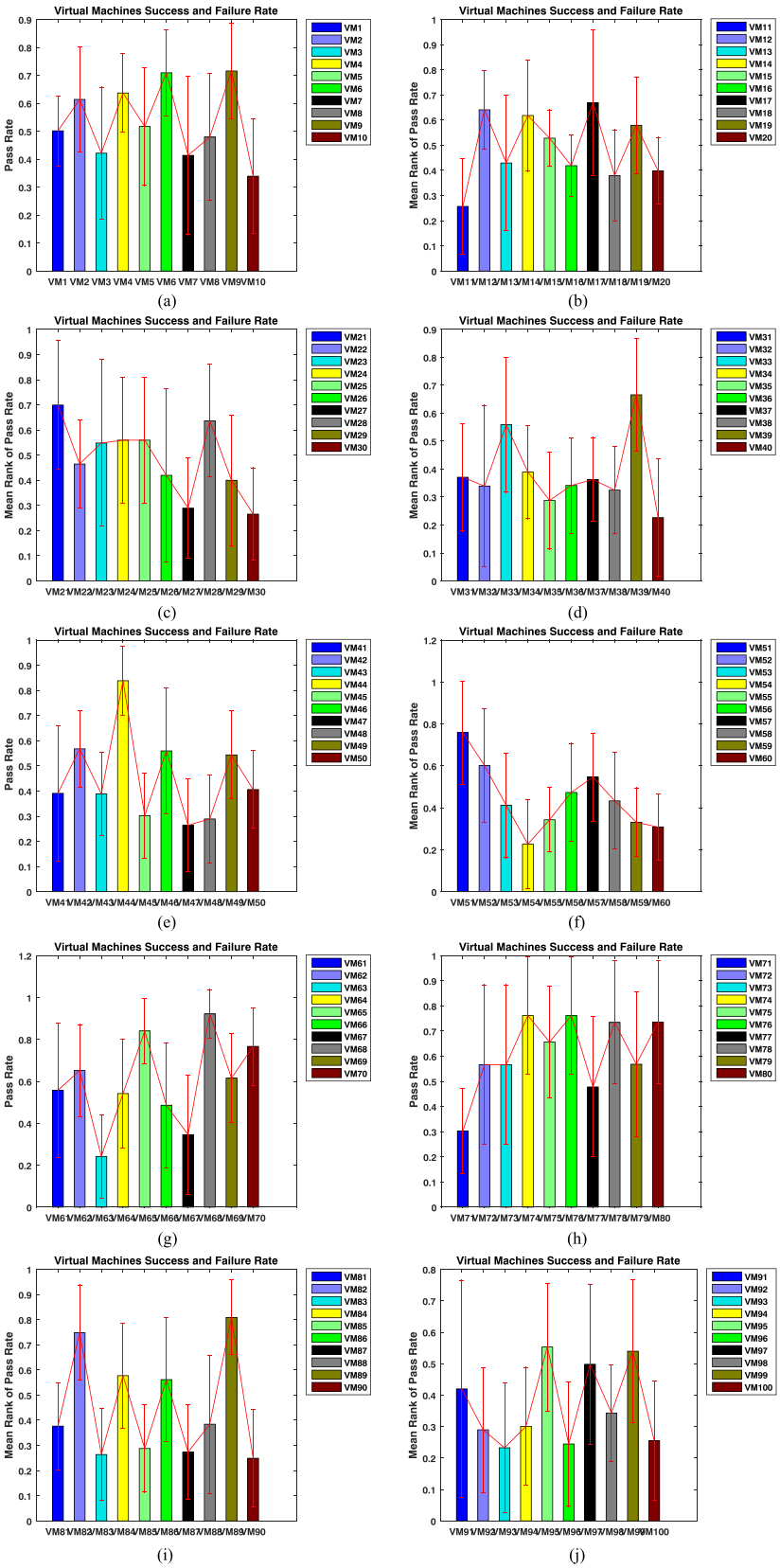


Figure 12. Simulation plots for 100 virtual nodes (VM-1 to VM-100) with error bars. [Colour figure can be viewed at wileyonlinelibrary.com]

DOI: 10.1002/spe

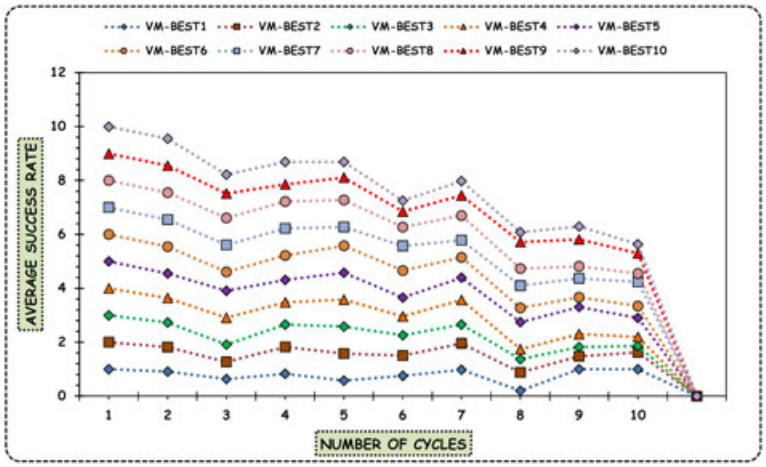


Figure 13. Average selected best virtual machine from 1 to 100 virtual machines (VMs). [Colour figure can be viewed at wileyonlinelibrary.com]

result of our simulations is presented in Figure 7 where we saw that the increase in PR is more than the decrease; hence, we can achieve a good performance of our algorithm.

Having assumed at the beginning that PR is 0.5, Figure 7(a) shows the PR analysis in VM-1, where a steady increase was observed from 0.5 to 0.778 and a further increase to 0.922 in cycle 4. A slight decrease was noticed from 0.922 to 0.874 and then a continuous increase from 0.874 to 0.994, which shows we can achieve a good performance overall on VM-1 because the increase in PR is more than the decrease.

In Figure 7(b), a steady increase from 0.5 to 0.87 was noticed, and from cycles 2 to 9, there was a decrease and increase in PR and a final increase from 0.721 to 0.888, which also shows that the increase in PR is greater than the decrease; hence, a good performance can be achieved from this VM. Similar scenarios occur from VM-3 to VM-6 where there was a combination of both decrease and increase in PR, but overall, we can see that the increase in PR is greater than the decrease in PR in all the VMs, which indicates a favorable performance of our integrated approach, because decrease in failure rate is less than the increase in PR.

Additionally, we performed an experiment using 100 VMs and 10 computing cycles, where we studied the success and failure rate patterns across the 100 nodes, as well as a performance comparison across the nodes. We plotted the error bars for VM-1 to 100 against the PR to enable us to observe the overlap and access the level of significance among the VMs.

From Figure 12(a), VM-9, VM-6, VM-4, and VM-2 have an excellent high PR followed by VM-1, VM-3, VM-5, and VM-8, which have high PR and are good compared with less PR. This implies that only VM-10 has a low PR, which indicates a good performance for the first 10 nodes.

In a similar scenario presented in Figure 12(b), it was observed that VM-12 to VM-20 had high success rate, which implies that the VMs across that range have a lesser tendency to fail; that is, the PRs are higher than the failure rates. Similar success and failure patterns were observed from Figure 12(c) to (j), which shows the reliability across the nodes.

Based on the results obtained, the following observations were made:

1. Overall, from VM1-100, the nodes have high PRs compared with the failure rates, which implies that the failure tendency across the entire 100 virtual node infrastructures is lesser. (Details of the results are presented in Figure 12.)
2. The error bars plotted across the 100 nodes show an overlap across each other, which indicates that the difference in error bars is not significant across the entire infrastructure.

Figure 13 shows the overall average selected best VM with high PR and less failure rate. This was selected by computing the output obtained from the group of 10 × 10 VMs after each computing cycle across 100 nodes.

DOI: 10.1002/spe

7. CONCLUSION

This paper presented an optimized FT approach for real-time computing on the cloud infrastructure. The proposed strategy uses the PR of computing virtual nodes with a fault manager using the checkpoint/replay technique, applying the reward renewal process theorem. It repairs faults generated before the deadline as an FT mechanism. Our results have shown that the scheme is highly fault tolerant because it brings all advantages of forward and backward recovery, which the system takes advantage of diverse software tools. Our algorithm integrates concepts of FT based on high PR of computing nodes and less service task finish time, increasing the system availability.

Six VMs were used in parallel with integrated virtualized checkpointing FT based on high PR of computing nodes and less task finish time. Additional experiments used 100 VMs in parallel and analyzed the pass and failure rates across them. Analyzing the pass and fail rate with performance of existing approaches, we found that our proposed FT scheme gives an improved performance. This is represented in Figures 7–10 as shown. Compared with adaptive and virtualized FT methods, our nodes showed higher PR and lesser failure rate. The average mean plots with standard deviation are able to verify these results statically. The error bars show an overlap across each other, indicating that the difference in error bars is not significant across the entire infrastructure.

In the future, we will extend the SFS algorithm to a more complex and large-scale high-performance environment, as well as in a real-life scenario by simulating them over an OpenStack IaaS. Designing a highly dependable and failure-free system requires a good understanding of failure characteristics. Here, we will study and analyze real-time cloud failure data, including the root cause of failures and statistics by applying machine-learning techniques such as clustering and anomaly finding. This will aid in re-examining other current algorithms and techniques for fault-tolerant cloud system, creating realistic benchmarks and test beds for FT testing.

ACKNOWLEDGEMENTS

One of the authors Bashir Mohammed is a Petroleum Technology Development Fund (PTDF) scholar. We would like to express our sincere gratitude to PTDF for its funding support under the OSS scheme with grant number (PTDF/E/OSS/PHD/MB/651/14).

REFERENCES

1. Bilal K, Khalid O, Malik SU, Khan MUS, Khan S, Zomaya A. Fault tolerance in the cloud. In *"Fault Tolerance in the Cloud" Encyclopedia on Cloud Computing*, vol. **2015**. John Wiley & Sons: Hoboken, NJ, USA, 2015: 291–300.
2. O. Sefraoui, M. Aissaoui, and M. Eleuldj, "Cloud computing migration and IT resources rationalization," *2014 Int. Conf. Multimed. Comput. Syst.*, pp. 1164–1168, Apr. 2014.
3. Y. Jararweh, Z. Alshara, M. Jarrah, M. Kharbutli, and M. N. Alsaleh, "TeachCloud: a cloud computing educational toolkit," no. 2012, pp. 1–16.
4. R. Jhawar, V. Piuri, and I. Universit, "Fault tolerance management in IaaS clouds," *2012 IEEE First AESS Eur. Conf. Satell. Telecommun.*, pp. 1–6, 2012.
5. Bala A, Chana I. Fault tolerance-challenges, techniques and implementation in cloud computing. *International Journal of Computer Science* 2012; **9**(1): 288–293.
6. Malik S, Huet F. Adaptive fault tolerance in real time cloud computing. *2011 IEEE World Congr. Serv.* 2011; (Jul. 2011): 280–287.
7. S. Shen, A. Iosup, A. Israel, W. Cirne, D. Raz, and D. Epema, "An availability-on-demand mechanism for datacenters," *2015 15th IEEE/ACM Int. Symp. Clust. Cloud Grid Comput.*, pp. 495–504, 2015.
8. B. Mohammed and M. Kiran, "Analysis of cloud test beds using opensource solutions," *2015 3rd Int. Conf. Futur. Internet Things Cloud*, pp. 195–203, 2015.
9. D. Sun, G. Chang, C. Miao, and X. Wang, "Analyzing, modeling and evaluating dynamic adaptive fault tolerance strategies in cloud computing environments," *Journal of Supercomputing*, vol. 66, no. 1. J Suercomputer (), pp. 193–228, 2013.
10. M. Pradesh, "A survey on various fault tolerant approaches for cloud environment during load balancing," vol. 4, no. 6, pp. 25–34, 2014.
11. Greenberg A, Hamilton J, Maltz DA, Patel P. The cost of a cloud: research problems in data center networks. *SIGCOMM Comput. Commun. Rev.* 2008; **39**(1): 68–73.
12. ITProPortal, "ITProPortal.com: 24/7 Tech Commentary & Analysis," 2012. [Online]. Available: http://www.itproportal.com/. [Accessed: 24-Jun-2015].

DOI: 10.1002/spe

13. Z. Pantic and M. Babar, "*Guidelines for Building a Private Cloud Infrastructure*," 2012.
14. Sen A, Madria S. Off-line risk assessment of cloud service provider. *2014 IEEE World Congr. Serv* Jun. 2014: 58–65.
15. S. Yadav, "Comparative study on open source software for cloud computing platform: Eucalyptus, Openstack and Opennebula," vol. **3**, no. 10, pp. 51–54, 2013.
16. A. D. Meshram, "Fault tolerance model for reliable cloud computing general terms:," no. July, 2013.
17. Paul HS, Gupta A, Badrinath R. Performance comparison of checkpoint and recovery protocols. *Concurr. Comput. Pract. Exp.* 2003; **15**(15): 1363–1386.
18. C.-T. Yang, Y.-T. Liu, J.-C. Liu, C.-L. Chuang, and F.-C. Jiang, "Implementation of a cloud IaaS with dynamic resource allocation method using OpenStack," *2013 Int. Conf. Parallel Distrib. Comput. Appl. Technol.*, pp. 71–78, Dec. 2013.
19. Singh K, Smallen S, Tilak S, Saul L. Failure analysis and prediction for the CIPRES science gateway Kritika. *Concurr. Comput. Pract. Exp.* 2016; **22**(6): 685–701.
20. Fu M, Zhu L, Sun D, Liu A, Bass L, Lu Q. Runtime recovery actions selection for sporadic operations on public cloud. *Softw. - Pract. Exp.* 2016; **39**(7): 701–736.
21. Chen G, Jin H, DeqingZou B, Zhou B, Qiang W. A lightweight software fault-tolerance system in the cloud enviroment. *Concurr. Comput. Pract. Exp.* 2015; **22**(6): 685–701.
22. Pei X, YijieWang XM, Xu F. Repairing multiple failures adaptively with erasure codes in distributed storage systems Xiaoqiang. *Concurr. Comput. Pract. Exp.* 2015; **22**(6): 685–701.
23. Bertolli C, Vanneschi M. Fault tolerance for data parallel programs C. *Concurr. Comput. Pract. Exp.* 2011; **22**(6): 685–701.
24. Maloney A, Goscinski A. A survey and review of the current state of rollback-recovery for cluster systems. *Concurr. Comput. Pract. Exp.* 2009; **22**(6): 685–701.
25. Alshareef HN, Grigoras D. Robust cloud management of MANET checkpoint sessions. *Concurr. Comput. Pract. Exp.* 2016; **22**(6): 685–701.
26. Bin Hong DW, Peng F, Deng B, Hu Y. DAC-Hmm: detecting anomaly in cloud systems with hidden Markov models. *Concurr. Comput. Pract. Exp.* 2015; **22**(6): 685–701.
27. Chen P, Xia Y, Pang S, Li J. A probabilistic model for performance analysis of cloud infrastructures. *Concurr. Comput. Pract. Exp.* 2015; **22**(6): 685–701.
28. Salehi MA, Javadi B, Buyya R. Resource provisioning based on preempting virtual machines in distributed systems Mohsen. *Concurr. Comput. Pract. Exp.* 2013; **22**(6): 685–701.
29. Nazari Cheraghlou M, Khadem-Zadeh A, Haghparast M. A survey of fault tolerance architecture in cloud computing. *Journal of Network and Computer Applications* 2015; **61**: 81–92.
30. A. Ganesh, M. Sandhya, and S. Shankar, "A study on fault tolerance methods in cloud computing," *2014 IEEE Int. Adv. Comput. Conf.*, pp. 844–849, 2014.
31. Kaur J, Kinger S. Efficient algorithm for fault tolerance in cloud computing. *2014 IJCSIT Int. J. Comput. Sci. Inf. Technol.* 2014; **5**: 6278–6281.
32. A. Tchana, L. Broto, and D. Hagimont, "Approaches to cloud computing fault tolerance," *IEEE CITS 2012–2012 Int. Conf. Comput. Inf. Telecommun. Syst.*, 2012.
33. K. Parveen, G. Raj, and K. R. Anjandeep, "A novel high adaptive fault tolerance model in real time cloud computing," pp. 138–143, 2014.
34. K. J. Naik and N. Satyanarayana, "A novel fault-tolerant task scheduling algorithm for computational grids," *2013 15th Int. Conf. Adv. Comput. Technol.*, pp. 1–6, 2013.
35. Amoon M. A job checkpointing system for computational grids. *Open Comput. Sci.* 2013; **3**(1): 17–26.
36. Siva Sathya S, Syam Babu K. Survey of fault tolerant techniques for grid. *Comput. Sci. Rev.* 2010; **4**(2): 101–120.
37. "Issue information," *Concurr. Comput. Pract. Exp.*, vol. **27**, no. 14, pp. i–ii, Sep. 2015.
38. I. P. Egwutuoha, S. Chen, D. Levy, B. Selic, and R. Calvo, "A proactive fault tolerance approach to high performance computing (HPC) in the cloud," *Proc. - 2nd Int. Conf. Cloud Green Comput. 2nd Int. Conf. Soc. Comput. Its Appl. CGC/SCA 2012*, pp. 268–273, 2012.
39. X. Kong, J. Huang, C. Lin, and P. D. Ungsunan, "Performance, fault-tolerance and scalability analysis of virtual infrastructure management system," *2009 IEEE Int. Symp. Parallel Distrib. Process. with Appl.*, pp. 282–289, 2009.
40. Okorafor E. A fault-tolerant high performance cloud strategy for scientific computing. *IEEE Int. Symp. Parallel Distrib. Process. Work. Phd Forum* 2011: 1525–1532.
41. R. Nogueira, F. Araujo, and R. Barbosa, "CloudBFT: elastic byzantine fault tolerance," *2014 IEEE 20th Pacific Rim International Symposium on Dependable Computing.*
42. Yadav N, Pandey SK. Fault tolerance in DCDIDP using HAProxy: 231–237.
43. Kim HKH, Kang SKS, Yeom HY. Node selection for a fault-tolerant streaming service on a peer-to-peer network. *2003 Int. Conf. Multimed. Expo. ICME '03. Proc. (Cat. No.03TH8698)* 2003; **2**(1): 6–12.
44. Sheng D, Franck C. GloudSim: Google trace based cloud simulator with virtual machines. *Softw. - Pract. Exp.* 2015; **39**(7): 701–736.
45. Qiang W, Jiang C, Ran L, Zou D, Services HJ. CDMCR: multi-level fault-tolerant system for distributed applications in cloud. *International Journal of Applied Engineering Research* 2015; **9**(22): 5968–5974.
46. Agbaria A, Friedman R. Virtual-machine-based heterogeneous checkpointing. *Softw. - Pract. Exp.* 2002; **32**(12): 1175–1192.
47. Singh D, Singh J, Chhabra A. High availability of clouds: failover strategies for cloud computing using integrated checkpointing algorithms. *Proc. - Int. Conf. Commun. Syst. Netw. Technol. CSNT* 2012; **2012**: 698–703.

DOI: 10.1002/spe

48. Jung G, Joshi KR, Hiltunen MA, Schlichting RD, Pu C. Performance and availability aware regeneration for cloud based multitier applications. *Proc. Int. Conf. Dependable Syst. Networks* 2010: 497–506.
49. Chtepen M, Claeys FHA, Dhoedt B, De Turck F, Demeester P, Vanrolleghem PA. Adaptive task checkpointing and replication: towards efficient fault-tolerant grids. *IEEE Transactions on Parallel and Distributed Systems* 2008; **20** (2): 180–190.
50. P. Das and P. M. Khilar, "VFT: a virtualization and fault tolerance approach for cloud computing," *2013 IEEE Conf. Inf. Commun. Technol. ICT 2013*, no. Ict, pp. 473–478, 2013.
51. Elliott J, Kharbas K, Fiala D, Mueller F, Ferreira K, Engelmann C. Combining partial redundancy and checkpointing for HPC. *Proc. - Int. Conf. Distrib. Comput. Syst.* 2012: 615–626.
52. H. Yanagisawa, T. Osogami, and R. Raymond, "Dependable virtual machine allocation," *2013 Proc. IEEE INFOCOM*, pp. 629–637, 2013.
53. A. Israel and D. Raz, "Cost aware fault recovery in clouds," 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM2013), pp. 9–17, 2013.
54. I. P. Egwutuoha, S. Chen, D. Levy, and B. Selic, "A fault tolerance framework for high performance computing in cloud," Proc. - *12th IEEE/ACM Int. Symp. Clust. Cloud Grid Comput. CCGrid 2012*, pp. 709–710, 2012.
55. Y. Liu, R. Nassar, C. Leangsuksun, N. Naksinehaboon, M. Paun, and S. L. Scott, "An optimal checkpoint/restart model for a large scale high performance computing system," *2008 IEEE Int. Symp. Parallel Distrib. Process.*, pp. 1–9, 2008.
56. G. F. Lawler, "Introduction to stochastic processes." p. 248, 2006.
57. R. Gallager, "Discrete stochastic processes," no. 0, pp. 92–138, 1996.
58. R. Nassar, B. Leangsuksun, and S. Scott, "High performance computing systems with various checkpointing schemes 2 full checkpoint/restart model," vol. IV, no. 4, pp. 386–400, 2009.
59. Bin E, Biran O, Boni O, Hadad E, Kolodner EK, Moatti Y, Lorenz DH. Guaranteeing high availability goals for virtual machine placement. *Proc. - Int. Conf. Distrib. Comput. Syst.* 2011: 700–709.
60. Sun D, Chang G, Miao C, Wang X. Analyzing, modeling and evaluating dynamic adaptive fault tolerance strategies in cloud computing environments. *The Journal of Supercomputing* 2013; **66**(1): 193–228.
61. M. Nita, F. Pop, M. Mocanu, and V. Cristea, "FIM-SIM: fault injection module for CloudSim based on statistical distributions."
62. B. Wickremasinghe, R. N. Calheiros, and R. Buyya, "CloudAnalyst: a CloudSim-based visual modeller for analysing cloud computing environments and applications," *2010 24th IEEE Int. Conf. Adv. Inf. Netw. Appl.*, pp. 446–452, 2010.
63. R. Buyya, R. Ranjan, and R. N. Calheiros, "Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: challenges and opportunities," *Proc. 2009 Int. Conf. High Perform. Comput. Simulation, HPCS 2009*, pp. 1–11, 2009.
64. A. Zhou, S. Wang, Q. Sun, H. Zou, and F. Yang, "FTCloudSim: a simulation tool for cloud service reliability enhancement mechanisms," *Proc. Demo Poster Track ACM/IFIP/USENIX Int. Middlew. Conf.*, pp. 2:1–2:2, 2013.

DOI: 10.1002/spe