Fail Over 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 and 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 to 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 virtualised 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 to existing approaches, preliminary experiment of the SFS algorithm indicate 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 to other techniques, laying a foundation for a fully fault tolerant IaaS Cloud environment

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

1. INTRODUCTION

Cloud computing is a popular paradigm and an attractive model for providing computing, IT 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 continue its intended operation, even at a reduced level, preventing it from breaking down completely when unexpected failure events occur [5, 6].

---

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

In real-time high performance large-scale systems with complex and dynamic cloud services, the system sometimes fail due to 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 fault tolerance 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 virtualised optimal checkpointing fault tolerance approach for cloud data centers, using intelligent selection mechanisms based on pass rates of computing virtual nodes with a fault manager. This SFS fault tolerance approach results in an optimized infrastructure for IaaS cloud platforms, showing a considerable improvement in current research of cloud fault tolerance.

Along with analysing current cloud fault tolerance approaches, the main contributions of this paper include;

- Providing high availability depending on cloud user requests to successful virtual machines using the SFS algorithm.
- Develop an integrated virtualised failover strategy for cloud datacenters, 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 fault tolerance for standard real time cloud computing systems. Section 4 presents our fault tolerance 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 increases and becomes difficult to predict. Particularly with system components being constantly upgraded, intensive workload on cloud servers and sometimes deploying faulty software. Achieving high availability of virtual machines 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 hours [30], with Google Docs service outage lasting for an hour. These were because of memory leaks due to a software update [42], costing both business millions of dollars. Similar reports were witnessed by Gmail services down for about 50 minutes, Amazon Web services for 6 hours, while Facebook's photos and "likes" services were down costing customer satisfaction. Multiple business hosting their websites, such as with GoDaddy, suffered 4 hours downtime affecting 5 million websites [30].

Growing number of datacenter resources increases the global complexity of 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 virtual machines have large time spans of a few days or months, running long tasks or jobs. Failure on these long or medium running jobs bring threats to fulfilment of Service Level Agreement (SLA) 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 500ms in response time [11]. In another example, millions of customers were left without Internet access for three days when there was a core switch failure in BlackBerry's network. Another example is when one of UK's top cellular company failed for three days, affecting 7 million subscribed customers [11], [12].

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

- Infrastructure as a service (IaaS), is the most basic and important cloud service model under which virtual machines, load balancers, fault tolerance, 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 are 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 benefit of these services include focus on high value software rather than infrastructure, leverage economies of scale and provide scalable go-to-market capability [2].
- SaaS 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. Cloud user do 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 job and the needed platform. The SP submits a task to the cloud controller and the scheduler allocates each job with suitable resources.

Depending on the type of fault and FT policies, several fault tolerance technique can be used [16], 20, 21] such as Reactive Fault Tolerance policy, which reduces failure effects when they occur on application execution and Proactive Fault Tolerance 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 is informed and the job is rescheduled on another virtual machine 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 fault tolerance strategies have been designed to reduce fault effects, but in this paper we propose an optimized fault tolerance strategy in real time Cloud Computing Environment to increase system availability, reduce the service time and enhanced rapid and efficient recovery from faults. Our Smart Failover Strategy (SFS) approach is applied to the Infrastructure as a service (IaaS) delivery layer, utilizing computing hardware resources and the virtualisation hypervisor to manage virtual machine instances running on physical servers.

To address the problem of job completion time, the pass rate optimized selection technique is integrated with a job checkpointing mechanism in the SFS approach. Here we restore the partially completed job from the last checkpoint saved checkpoint rather than re-starting 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 entire system running states and inappropriate checkpointing that cause 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 virtual machines based on highest pass rate in job execution. To address these problems, our optimized selection technique selects only virtual machines with successful status check and successful task time limit checker. Further components such as load balancers, fault tolerance engine, firewalls and networking services are utilized by cloud datacenters to help manage and regulate fault tolerance strategies in the cloud model [2], [18].

3. RELATED WORK

There are many approaches proposed to deal with fault tolerance in cloud and recent studies have analyzed fault tolerance in cloud & grid computing [9, 10],[19]–[24], [25]–[29], [30]–[37] and more broadly in the area of fault tolerance for standard real time systems [4],[6], [26, 27], [31], [33], [38], [39]–[44], [44] but very few works have addressed issues of optimized fault tolerance 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 fault tolerance by focusing on autonomic repair. They proved that in most current fault tolerance approaches, faults are exclusively handled by the provider or the customer which leads to partial or inefficient solutions, while a collaborative solutions are much more promising. They demonstrated this with experiments on collaborative fault tolerance solutions by implementing an autonomic prototype cloud infrastructure. Maloney and Goscinski [24] reviewed issues relating to providing fault tolerance for long-running applications. They investigated several fault tolerance 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 fault tolerance 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 virtual machine, 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 (CDMCR). The system backs up complete application states periodically as a snapshot-based distributed checkpointing protocol, including file system state. The authors proposed a multi-level recovery strategy, that includes process-level recovery, virtual machine recreation, host rescheduling, enabling comprehensive and efficient fault tolerance 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 MANET. Their experimental results showed that the model is feasible, robust, and saved time and energy if session breaks occur frequently. Additionally, Agbaria and Friedman [46] proposed a virtual-machine-based heterogeneous checkpointing mechanism, where they explored how to construct a mechanism at virtual machine level rather than dumping the entire state of the application process. The mechanism dumps the state of application as maintained by a virtual machine and during restart, the saved state is loaded as a new copy of the virtual machine, 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 fault tolerance in a complex cloud computing environment with a focus on first come first serve (FCFS) and shortest-job-first (SJF) along with misses per instruction on large (MPIL) method with fault tolerance property. Their proposed algorithm works for reactive fault tolerance among the servers and reallocating the faulty servers task to the new server which has minimum load at the instant of the fault cloud infrastructure that we prototyped. It also includes algorithm comparison between misses per instruction (MPI) and MPIL.

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 check pointing 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 highly 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, 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 to traditional approaches. An alternate approach by Shen et al. [7] proposed a mechanism called Availability-on-Demand (AoD) which consisted of an API that allowed datacenter users to specify availability requirements and uses 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 AoD 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 check pointing and replication and evaluated it in a grid simulation environment called Dynamic Scheduling in Distributed Environments (DSiDE). 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 fault tolerance 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, fault tolerance was achieved by redundancy and fault handlers. Their technique was mainly designed to provide a reactive fault tolerance where the fault handler prevents the unrecoverable faulty nodes from having 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 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 datacenter 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 to currently used approaches, by showing 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 fault tolerance in real time cloud computing, based on computing the reliabilities of the virtual machines based on cloudlets, using million instructions per second (mips), RAM and bandwidth. In this approach, if there are two virtual machines, 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 a fault tolerance model for real time cloud computing, where the system would tolerate the faults and then makes the decision on the basis of reliability of the processing nodes or virtual machines. They presented a metric model for the reliability assessment where they assessed the number of times a decrease in reliability occurred compared to the number of times an increase happened. This proposed technique was based on the execution of design diverse variants on multiple virtual machines, 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 Fault Tolerance 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, showed that the wall clock execution time of the computation-intensive applications can be reduced by as much as 30% with the frequency of check pointing reduced to 50% compared to current FT approaches. Further work by the authors [54] presented a fault tolerance framework using a process level redundancy (PLR) 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 (MPI) and using check pointing for VMs. Using simulations show 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 fault tolerance approach in real time cloud applications for running virtual machines. Using the techniques based on parameters being optimized, we apply a selection rate process approach, where a virtual machine or node is selected for computation on the basis of its previous pass rate 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 virtualised fail-over strategy has been validated through quantitative and experimental results by simulations for testing performance for success in four scenarios – partial, full pass and partial, full fail situations for fault tolerance 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.

4. SMART FAULT TOLERANCE IN CLOUD – THE VISION

The overall vision of fault tolerance in cloud computing is to provide high availability to fulfill the client requests on service performance and completion time as defined by the SLA. A fault tolerance service is an essential part of the service level objective, therefore a fault tolerance function in a cloud environment is extremely crucial. This section presents a working model of the strategy and a mathematical relationship that represents the fault tolerance model for our cloud computing system using the FT checkpoint scheme. The FT checkpoint uses a Reward Renewal process (RRP) 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 fault tolerance system architecture consisting of four zones, the approach has been analysed with relation to some extreme use cases to analyse how the cloud controller would perform as presented in the next section.

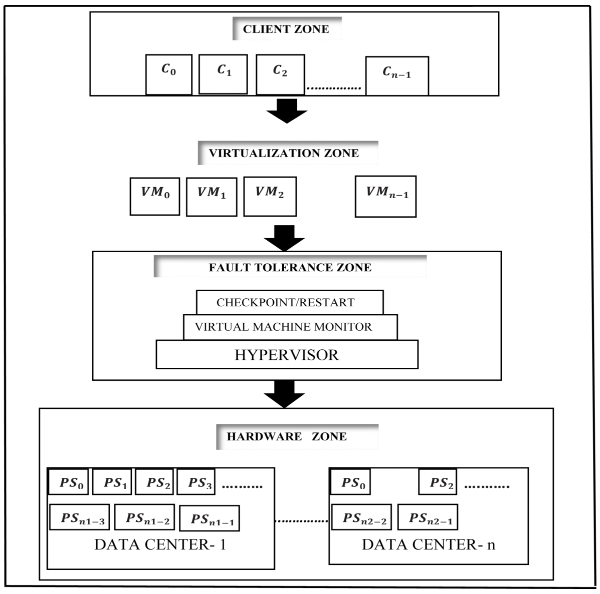

CLIENT ZONE
C_0 C_1 C_2 .... C_{n-1}
VIRTUALIZATION ZONE
VM_0 VM_1 VM_2 VM_{n-1}
FAULT TOLERANCE ZONE
CHECKPOINT/RESTART
VIRTUAL MACHINE MONITOR
HYPERVISOR
HARDWARE ZONE
PS_0 PS_1 PS_2 PS_3 ....
PS_{n1-3} PS_{n1-2} PS_{n1-1}
DATA CENTER- 1
PS_0 PS_2 ....
PS_{n2-2} PS_{n2-1}
DATA CENTER- n

Figure 1. Fault tolerance architecture

We define a mathematical representation of the fault tolerance model for a cloud computing system, presenting a working model of integrated virtualised fault tolerance approach. The fault tolerance system architecture shown in Figure 1, consist of four zones namely;

- The Client zone: One or more client can access service of cloud on-demand at any given time.
- The Virtualisation zone: One or more virtual machines instances can be started up, terminated and migrated within the data center. Also acting as link between client and fault tolerance cloud environment.
- The Fault tolerance 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 datacenter consist of numerous physical servers, providing hardware infrastructure for starting up virtual machines instances [7].

Table 1. 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 Centre
FT_s	Set of defined FT service levels
OBJ_f	Objective Function for optimizing a FT cloud
PR_A	Pass Rate Algorithm
FT_L	Fault Tolerance Level
$Chk(Opt)$	Checkpoint optimization strategy

Using the variables defined in Table 1, let the Fault Tolerance model (FT_m) of a cloud computing system be represented by a finite ordered list of elements or a sequence of five elements,

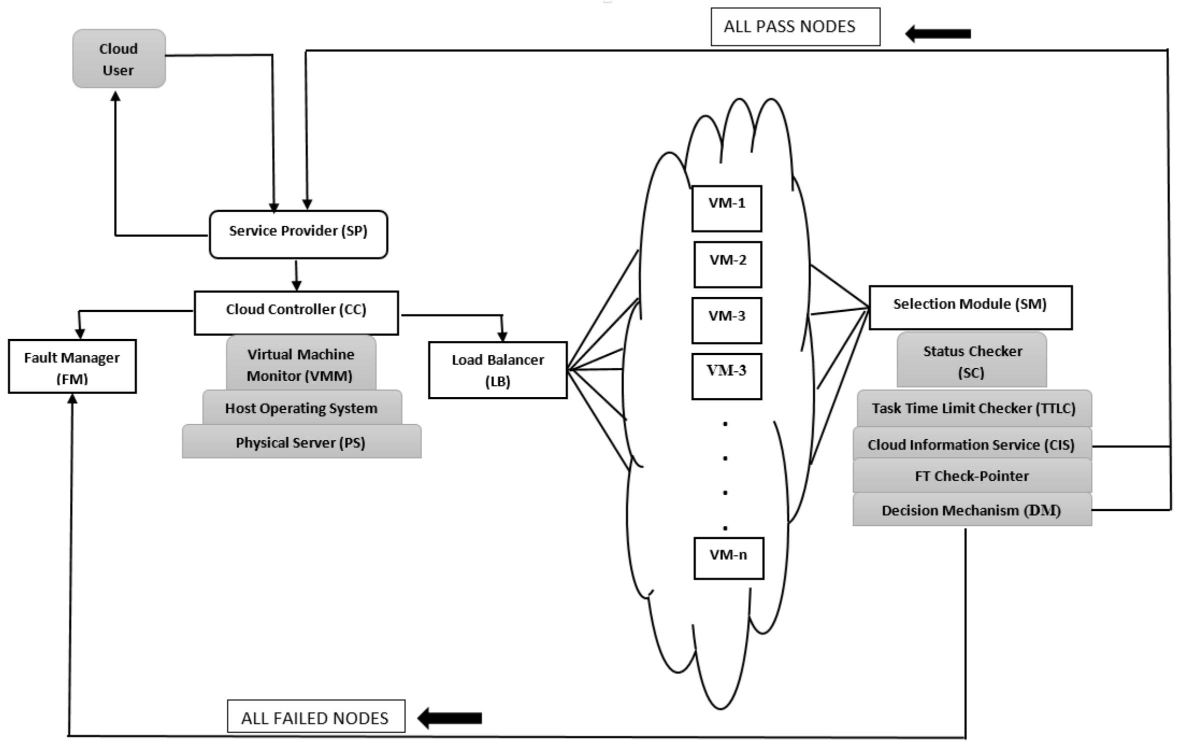
$$FT_m = (C, DC, FT_s\ , OBJ_f, PR_A) \tag{1}$$

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

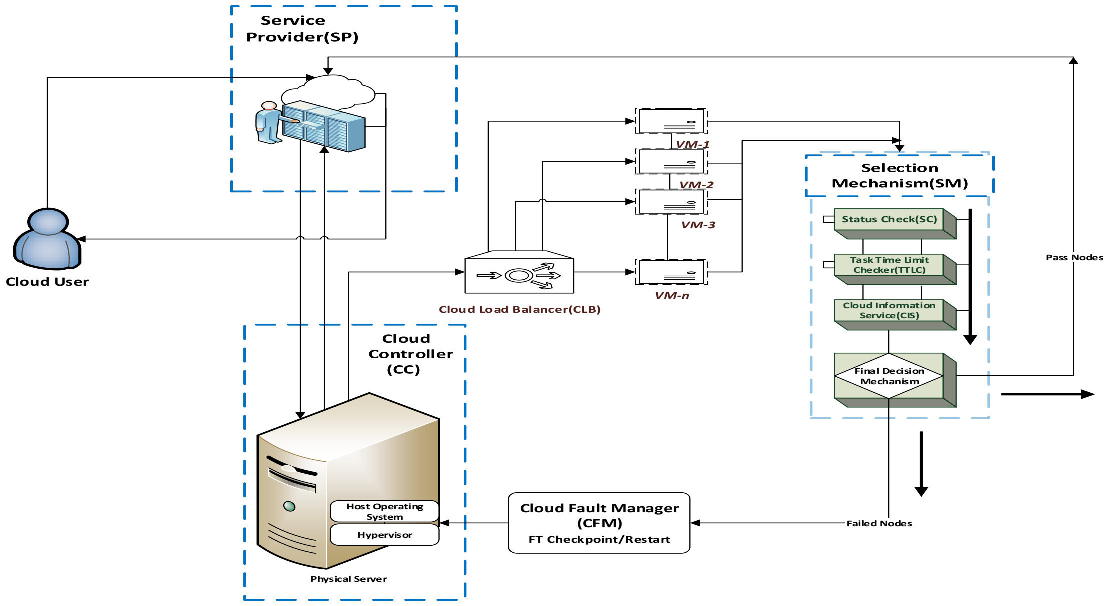
Where PR_A is an algorithm which selects the optimal pass rate, $C = \{c_0, c_1, c_3, \dots, c_{n-1}\}$ represents a set of n-clients that may request services separately, FT_L represents a set of fault tolerance service levels available by the cloud service provider, OBJ_f is the cloud fault tolerance optimizing objective function as given in (1), and $DC = \{dc_0, dc_1, dc_3, \dots, dc_{n-1}\}$ represents a data center set which is made up of dc_n data centers, where $dc_i = \{ps_0, ps_1, ps_3, \dots, ps_{i_{dci-1}}\}$ and ps_{i_k} $(0 \le k < dc_i)$ 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 virtual machine or node is selected for computation on the basis of its previous pass rate. 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 virtual machine monitor (or possibly the hypervisor) that is either software, hardware or firmware that creates and runs virtual machines. The host machine is the server where the VMM or hypervisor runs guest virtual machines. The VMM presents the guest operating systems with virtual operating platforms and also manages the execution of these guest operating systems.



(a) Block Representation



(b) Topology

Figure 2. Proposed System Model

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 pass rate. The model is made up of the following modules as shown (Figure 2)

- Service provider (SP) – Is responsible for forwarding the task submitted by clients to the Cloud Controller (CC). It also returns results obtained from the cloud controller to the cloud user.
- Cloud Fault Manager (CFM) – One of the most critical modules in the model as it keeps the system in operation and prevents break down. 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 cloud controller, here the CFM takes responsibility and updates the cloud information service record table. In other words during this time if there is no executing virtual node on the host server, the Cloud Fault manager (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 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.
- Cloud Controller (CC) – This is directly linked to the SP and is part of the cloud architecture. Virtualisation is done using the hypervisor, which provides system resources access to virtual machines 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 pass rate (PR) are contained in the Cloud Information Service (CIS), which helps to identify the virtual nodes and keeps record of tasks assigned to virtual nodes of a particular host or physical server.
- Cloud Load balancer (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 virtualisation. The CLB will assign task to virtual nodes whose corresponding physical servers have a high pass rate.
- Status Checker (SC) –This is the first sub module under the selection mechanism module. It checks the status of each virtual node either it is pass or fail.
- Cloud Information Service Record Table (CIS) – This is a performance record table that contains server IDs, virtual nodes IDs and pass rate values to identify corresponding virtual nodes.
- Task Time Limit Checker (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 assigned task is completed within an agreed time limit.

- Selection Module (SM) – This module provides the crucial process and is made 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 pass rate of corresponding node is increased and forwarded to the decision mechanism module for final selection process. But if both SC or TTLC fail, then corresponding virtual machine is not forwarded for final selection and instead the node is forwarded to the cloud fault manager for fault detection and recovery. In a scenario where SC is success but the task is not completed within the time limit, the pass rate in the CIS record table of that particular node is decreased and that node is not forwarded to the final decision mechanism sub module. Table 2 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 pass rate 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.

Table 2. Rules of the System

RULES	CONDITION	DECISION
1	If (SC status == pass) && (TTLC status ==pass)	Increase PR and forward to selection module for decision & selection.
2	If (SC status == pass) && (TTLC status ==fail)	Update database in CIS module, decrease PR, and corresponding VM not sent to selection module.
3	If (SC status == fail) && (TTLC status == pass)	Decrease PR and node sent to CFM for identification, detection & recovery.
4	If (SC status == fail) && (TTLC status ==fail)	Decrease PR and node also sent to CFM for detection & recovery.

4.2. Use Case Scenarios

In the above scheme, all virtual nodes run different algorithm resulting in different scenarios of pass and fail rates, representing diversification in software and timing constraints. Following are some scenarios that could occur.

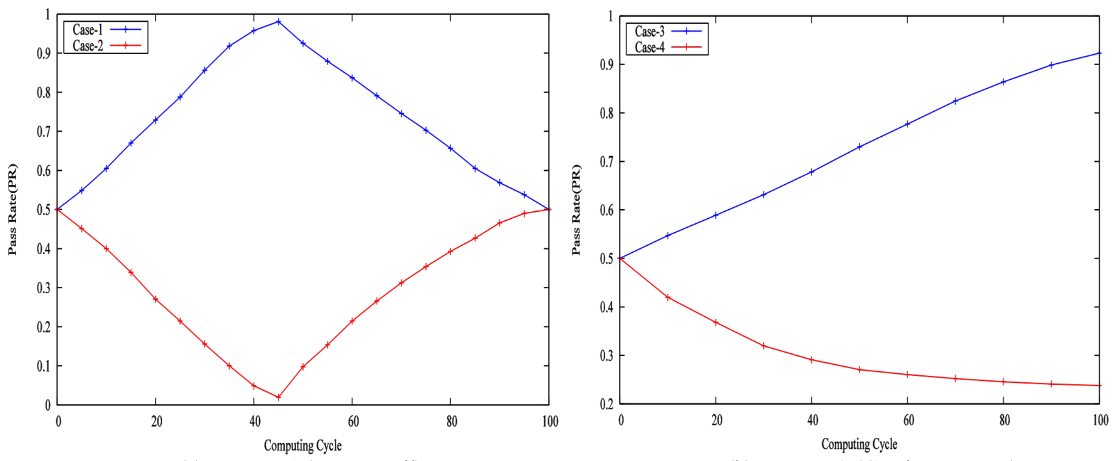
- Full Pass Scenario- The entire algorithm on each virtual node produces a successful outcome. Here the SC and TTLC is also pass because the task is completed within the stipulated time limit. The pass rate of the corresponding node is then increased and it goes to the selection mechanism for the final 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 pass rate value and performs a checkpoint before sending back to the service provider. However, in this case no failure is recorded in any of the virtual machines.

- Partial Pass Scenario- All virtual machines 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 virtual machine. 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 pass rate within the time limit.

- Full Failure Scenario – if the status check is fail, then automatically the task limit check is also fail and all the faulty nodes are sent to the Cloud Fault manager 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 fail then the corresponding virtual machine is not considered or forwarded to the final selection module. However, some virtual machines produce some pass results only when the SC is pass and the results are produced within the time limit, thereby sending the virtual machine to the selection module and increasing the Success rate of that node in the Cloud information service record table. Here error signals will be generated for failed virtual machines 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 pass rate and failure rate are as follows:
Pass Rate: 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 task time limit checker. Details of this is presented mathematically in the next section.



(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 Scenarios for Pass and Fail

4.3. Fault Tolerance using Checkpointing Mechanism

Checkpointing strategies has drawn significant attention over the last couple of years in the context of fault tolerance 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 fault tolerance in a cloud environment, we integrate it with the pass rate 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 fault tolerance in cloud and to achieve an optimal level of cloud

serviceability and cloud service level agreement, we present a mathematical proof for fault tolerance strategy based on our model. Table 3 presents key parameters of the model.

Table 3. Parameters of the Fault Tolerance Model

$\boldsymbol{\Omega}_i$	The cycle between failure i and failure(i+1)
$\boldsymbol{J}_n$	Time of the nth checkpoint
T_{Rol}	Roll Back failure Time between T_F and last successful $\boldsymbol{J}_n$ Checkpoint
R_p	Restart Point
T_{Rec}	Recovery Point/Time or FT 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 (FDF)
T_{OVFF}	Longest Failure Free Checkpoint Overhead
$\rho(t)$	Checkpoint density function

According to Figure 4 the time interval between consecutive checkpoints ΔJ is a critical factor to tradeoff checkpointing overhead T_{OV} and fault tolerance 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 is given as $f(t)$ while the checkpoint density function is given as $\rho(t)$.
The scheme uses a checkpoint model that follows a RRP, 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 the system, the application will be restarted from the last successful checkpoint. The following assumptions were made.

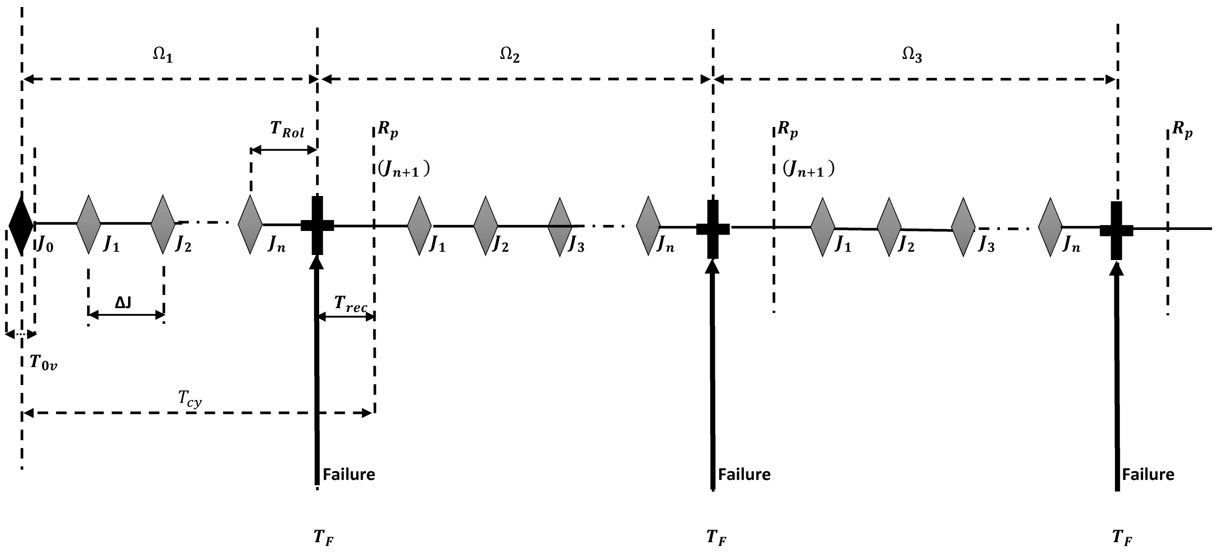


Figure 4. Full Checkpointing Strategy Failure Cycle

Assumption 1: Let $(\boldsymbol{T_{ov}}\ , \boldsymbol{T_{rec}}\ , \boldsymbol{T_{Rol}})$ of each cycle be a sequence of independent identically random variable $(\boldsymbol{L_1}\ , \boldsymbol{L_2}\ , \boldsymbol{L_3})\ \ldots\ldots,$ which is dependent on any point in time failure occurs in the system $\boldsymbol{\Omega}$ 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 decreases the time delay because checkpoint should not be carried out too regularly to balance the $\boldsymbol{T_{ov}}$ and Roll back time of our application.
Therefore the time delay can be expressed as

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

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

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

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 $$\lim_{t\to\infty} \frac{1}{t} f(x) = \frac{1}{E[\boldsymbol{\Omega_1}]} \tag{7}$$

Substituting (6) into (7) gives $$\lim_{t\to\infty} \frac{X_t}{t} = \frac{1}{E[\boldsymbol{\Omega_1}]} \tag{8}$$

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:

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

The reward function thereby satisfies $$\lim_{t\to\infty} \frac{1}{t} g(x) = \frac{E[\mathbf{L_1}]}{E[\boldsymbol{\Omega_1}]} \tag{10}$$

Substituting (9) into (10) gives $$\lim_{t\to\infty} \frac{E[\mathbf{L}_t]}{t} = \frac{E[\mathbf{L_1}]}{E[\boldsymbol{\Omega_1}]} \tag{11}$$

From equation (4), since $$L_t = \sum_{i=0}^{x_t} T_i \tag{12}$$

Then equation (11) becomes, $$\lim_{t\to\infty} \frac{E[\sum_{i=0}^{x_t} T_i]}{t} = \frac{E[\mathbf{L_1}]}{E[\boldsymbol{\Omega_1}]} \tag{13}$$

Therefore, $$L_t = \frac{E[\mathbf{L_1}]}{E[\boldsymbol{\Omega_1}]} \tag{14}$$

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 other to improve the checkpoint mechanism, checkpoints should not be performed too frequently in other to achieve balancing between the checkpoint overhead, recovery time and application re-computing 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 check pointing 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.

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 above, T_{ovFF} is defined as the checkpointing overhead during the longest failure-free time interval of consecutive checkpoint and it is associated to 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.

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

Where $N_{j_x \cdot j_y}$ 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

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

Since $N_{j_x \cdot j_y}$ is the number of checkpoints to fall within an interval $[j_x, j_y]$,

Then $N_{j_{n-1} \cdot j_n}$ is the number of checkpoints to fall within an interval $[j_{n-1}, j_n]$,

From (17) we have $$\int_{j_{n-1}}^{j_n} \rho(t)\, d\tau = \int_{j_{n-1}}^{j_n} \frac{1}{\Delta J}\, d\tau = \int_{j_{n-1}}^{j_n} \frac{1}{j_n - j_{n-1}}\, d\tau = N_{j_{n-1} \cdot j_n} = 1 \tag{18}$$

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

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

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

Since 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 Failure density function (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* $F(t)$ represents the failure distribution function which is connected to the failure density function. We now have

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

If $F(t)$ of a continuous failure is defined as a function that describes random variable t with a given failure density function$f(t)$, 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,

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

Equation (25) now gives

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

Recall that since 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 accepts probability density function.
Since(FT_{OV}) is associated to T_F, and T_F fall within time interval $[J_{n,} J_{n,+1}]$, therefore we can simply calculate T_F and J_n as well as the Roll Back failure Time between T_F and last successful $\boldsymbol{J_n}$ Checkpoint which is given as:

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

From our checkpoint model, the checkpointing time interval in a cycle fall 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 $T_{F,}$ fall 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

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

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

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

From (27) Integrating gives

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

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 failure density function FDF of a continuous failure which is related to the failure rate λ

and not to time, because the system failure rate λ doesn't change in the time interval $[J_{n,} J_{n,+1}]$, and

$$R(t) = 1 - f(t) \tag{37}$$

Note that the $\lambda(t)$ function is a conditional probability of the failure density function. 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 gives

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

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

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

Therefore,

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

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

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

Substituting (16) in (45) gives

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

From Figure 4 above, 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

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

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

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

The failure expectation distribution value $E(T_{cy})$ can be calculated since the time interval in our failure circle is $[J_{0,} J_{n,+1}]$, and $f(t)\ and\ F(t)$ are the failure density function and failure distribution function respectively. So,

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

Integrating $wrt\ t$ and substituting (51) into (52) gives,

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

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

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

And ΔJ is obtained as

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

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

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

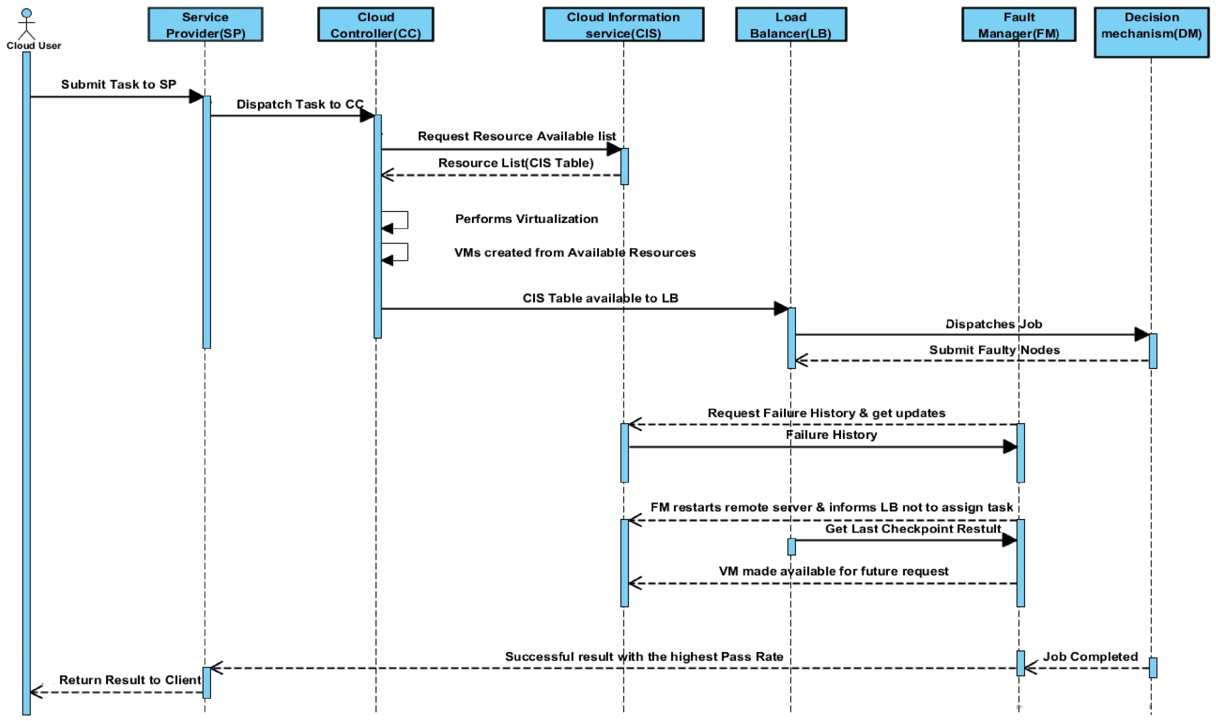

Cloud User
Service Provider(SP)
Cloud Controller(CC)
Cloud Information service(CIS)
Load Balancer(LB)
Fault Manager(FM)
Decision mechanism(DM)
Submit Task to SP
Dispatch Task to CC
Request Resource Available list
Resource List(CIS Table)
Performs Virtualization
VMs created from Available Resources
CIS Table available to LB
Dispatches Job
Submit Faulty Nodes
Request Failure History & get updates
Failure History
FM restarts remote server & informs LB not to assign task
Get Last Checkpoint Restult
VM made available for future request
Successful result with the highest Pass Rate
Job Completed
Return Result to Client

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

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 task with user quality of service requirements to the service provider and dispatches tasks to the Cloud Controller.
2. Cloud Controller sends request to the Cloud information Service 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 cloud controller performs the following:
   (a) Performs virtualisation with the help of a hypervisor.
   (b) Virtual machines (nodes) are created from the available resources of the physical server.
   (c) A CIS table containing the server IDs, virtual nodes and pass rates 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 pass rate from those successful virtual nodes.
5. The load balancer distributes the task based on the information gotten from the CIS table, by assigning 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 success
   (b) The PR of corresponding node is increased and forwarded to final decision mechanism module.
   (c) The decision mechanism delivers the result of 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 are fail
   (b) Either SC or TTLC is fail and the corresponding virtual machine is not sent to the decision mechanism.
   (c) The corresponding virtual machine 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=$x_1$/$x_2$ Update 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**
```

4.3. Pass Rate (PR) and Fail Rate (FR) 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 virtual machine of host produces a pass outcome and x_2 represents time the cloud controller'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 pass rate after 200 computing cycle is 0.978 for scenario-1, whereas decrease in pass rate for scenario-2 is 0.579, with increase in pass rate for scenario-3 and scenario-4 being 0.38. This shows that the increase in pass rate 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 containing higher number of virtual machines and were tested for validation of result. This is discussed is Section 5.

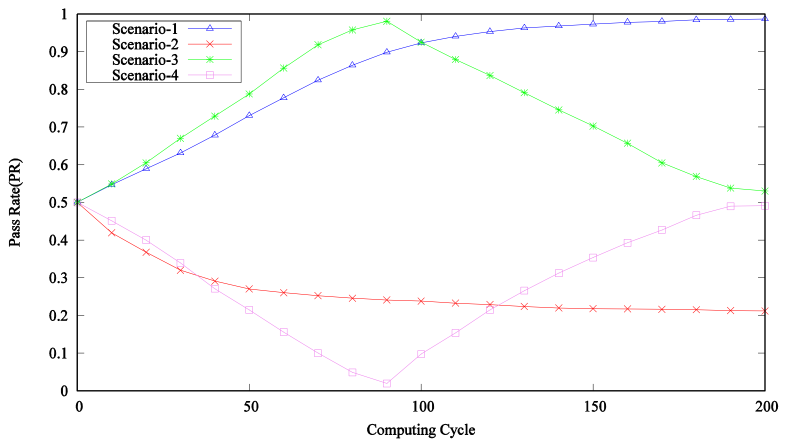


Figure 6. Pass/Pass⇨Fail & Fail/Fail⇨Pass Shifting plots

1. EXPERIMENTAL SETUP AND RESULTS

The experiments were conducted using CloudSim [61],[62]–[64] where 100 virtual nodes were created for performance comparison and scalability validation. We started by running the integrated virtualised optimal checkpointing algorithm using 10 computing cycles and created 6 virtual nodes with every individual node executing 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 3 virtual nodes with 10 computing cycle as depicted in section 5.1. To analyse 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 virtual machines before returning the result of the successful job to the client via the service provider. At the service provider level, the selection or decision mechanism is integrated with the cloud controller module. In a situation where a failure occurs in one of the nodes, the system will automatically adapt a fail-over strategy and continue operating using the remaining nodes. The system will 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 done 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 are available. The task deadlines are taken as input with initial values x_1 =1, $x_2 = 2$ and PR=0.5 considered for every node. Figure 7 shows some experimental results obtained from six virtual machines where pass and failure rate analysis.

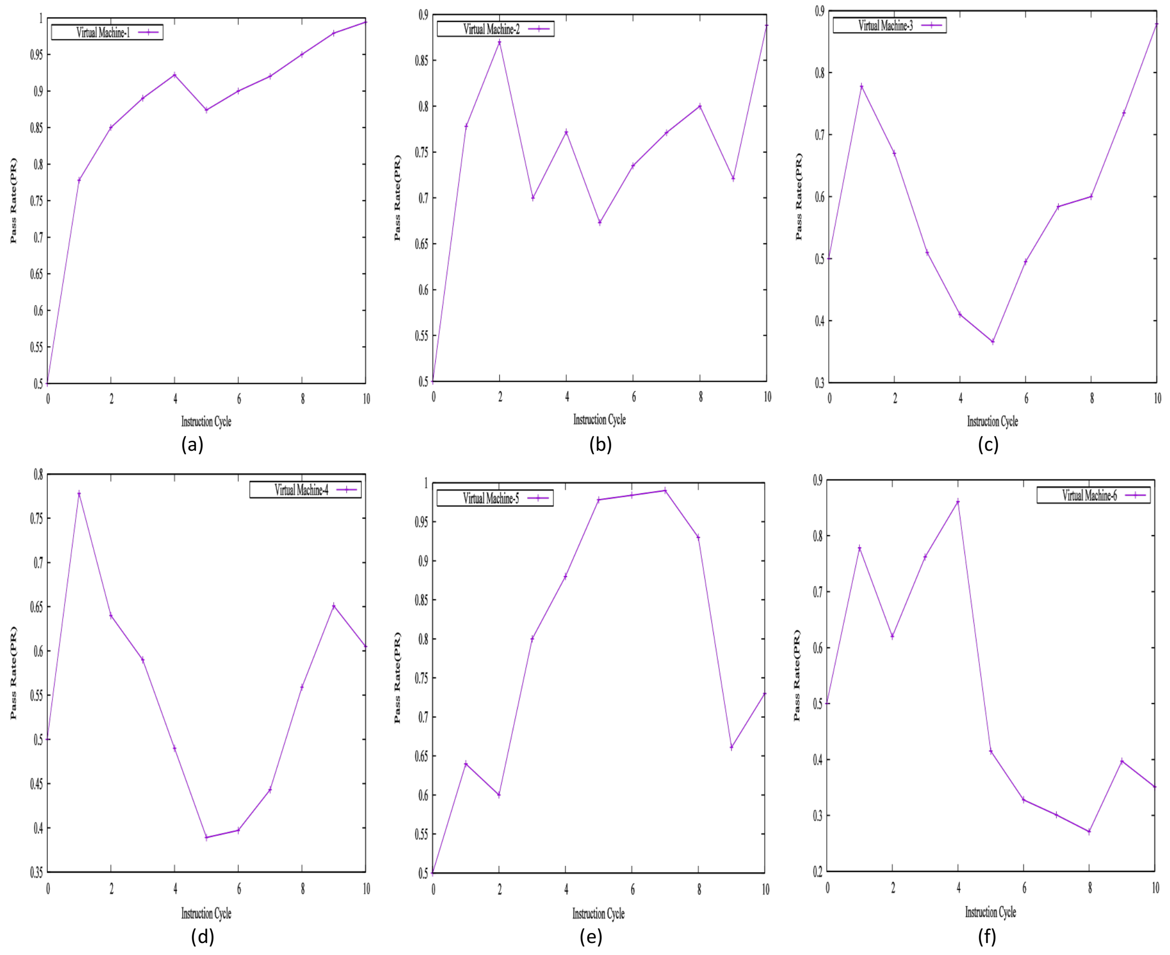


Figure 7. Experimental result from Six Virtual machines

5.1 Performance Comparison of Results

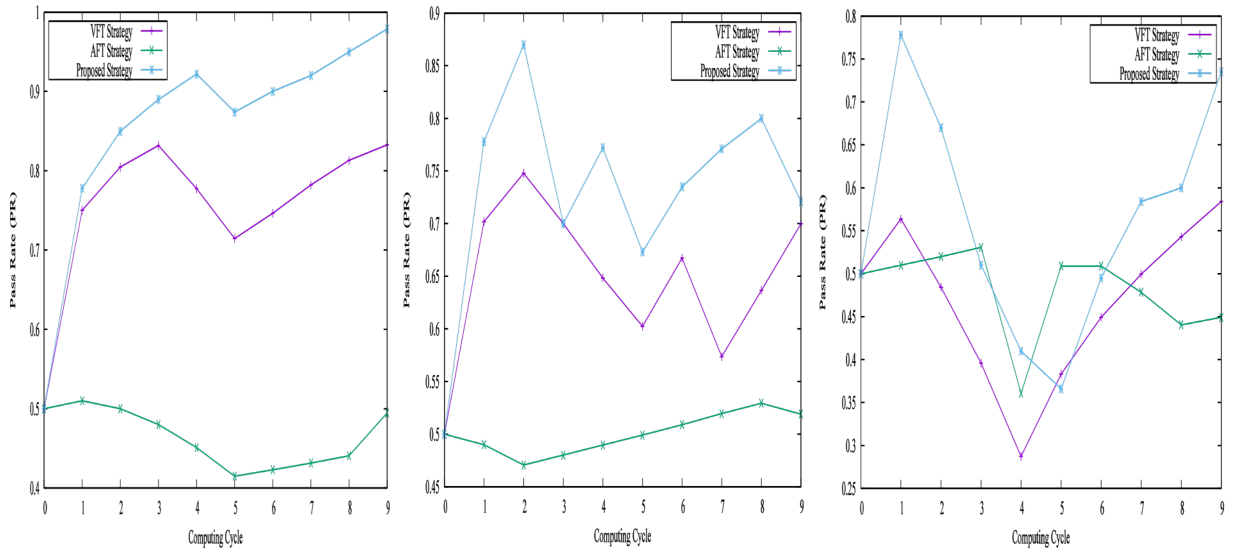
Figure 8 presents' results obtained from the performance comparison of our proposed strategy with other existing approaches where we observed that our proposed strategy has a better performance compared to the existing approaches, as increase in pass rate both for virtual node-1, 2 and 3 are higher than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

1. Virtualization and Fault Tolerance Approach (VFT) – Das et al. [50] proposed a virtualization and fault tolerance technique to reduce the service time and increase the system availability. In their proposed approach a Cloud Manager module and a Decision Maker we used to manage virtualization and load balancing which try to handle faults. By performing virtualization and load balancing, fault tolerance was achieved by redundancy and fault handlers. Their technique was mainly designed to provide a reactive fault tolerance where the fault handler prevents the unrecoverable faulty nodes from having adverse effect.
2. Adaptive Fault Tolerance Approach (AFT) –Malik et al. [6] proposed an adaptive fault tolerance 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 strategy [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}, 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.

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

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



(a) Virtual Node– 1 (b) Virtual Node – 2 (c) Virtual Node – 3

Figure 8. Performance Comparison of VFT, AFT and our Proposed Strategy

A performance evaluation of VFT and AFT strategy with our proposed model was carried out using the parameters in Table 4.

Table 4. Parameters

Parameter	Meaning
x_{re}	Relative error
x_i	Actual error
e_1	Average percentage relative error (APE)
e_2	Average absolute percentage relative error (AAPE)
e_3	Standard deviation (SD)
e_4	Average Actual error (AAA)
e_5	Average absolute actual error (AAAE)
e_6	Standard deviation about average actual error (SDAE)

Equation (59-64) gives the mathematical definition of these parameters and Table 5 presents the experimental results.

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

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

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

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

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

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

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 virtual machine-1, the pass rate 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 virtual machine-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 virtual machines displayed an improved performance. While in virtual machine-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. Overall this indicates that our proposed strategy has a better output and improved performance compared to the existing approaches.

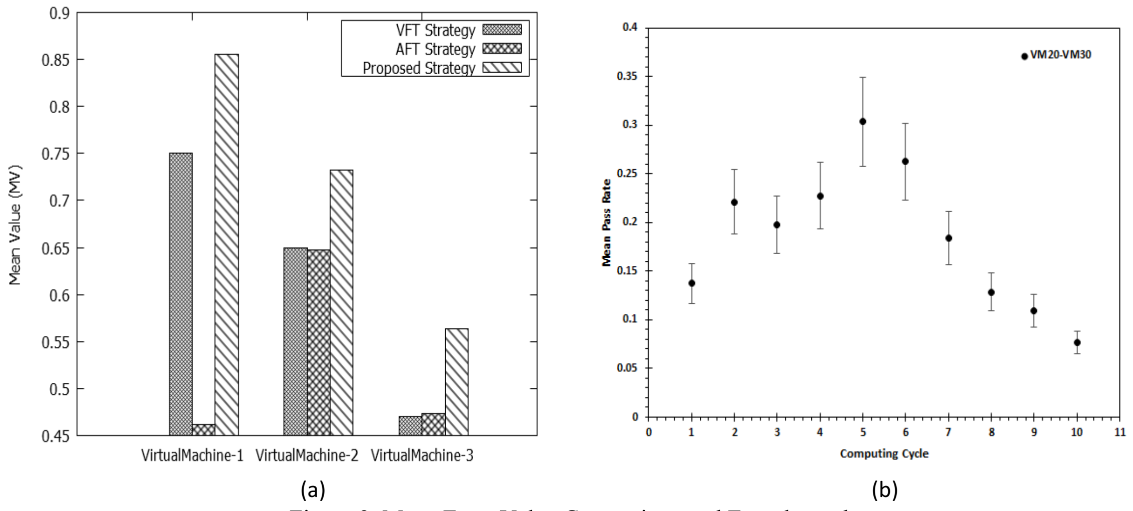


Figure 9. Mean Error Value Comparison and Error bars plot

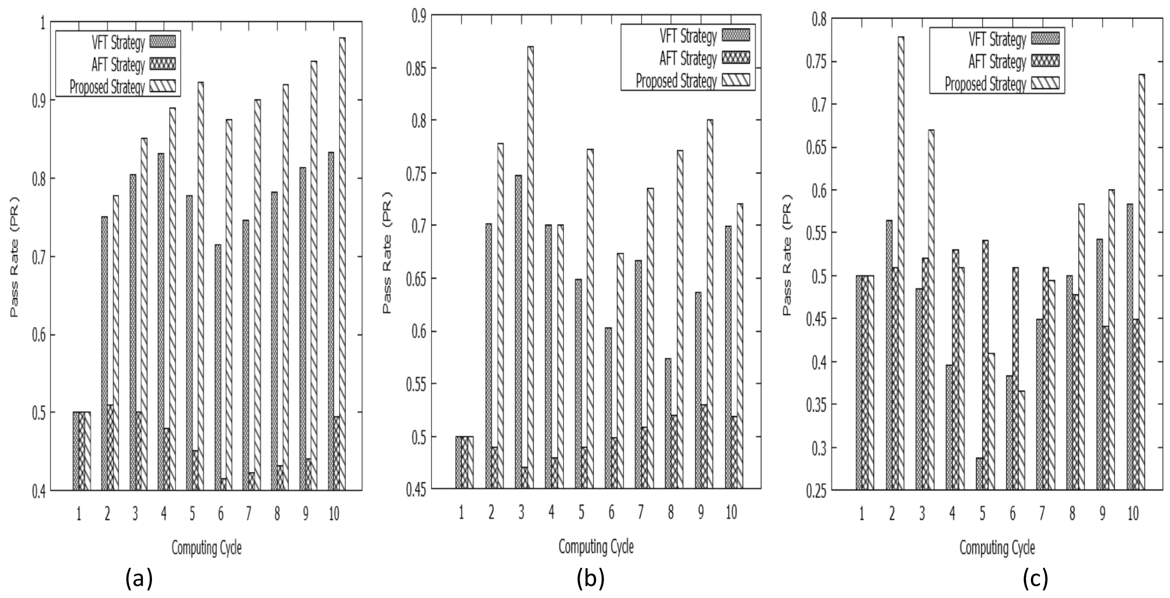


Figure 10. Success rate analysis of virtual machines

Based on the performance comparison analysis in Table 5, 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 pass rate is lesser compared to the measured pass rate of our proposed strategy. While for the AFT algorithm, the calculated pass rate is far much lesser than both the VFT algorithm and our proposed strategy. Details of this model can be found in [50] and [6] .

Table 5. Virtual Node-1 Performance Comparison of VFT, AFT with our Proposed Strategy

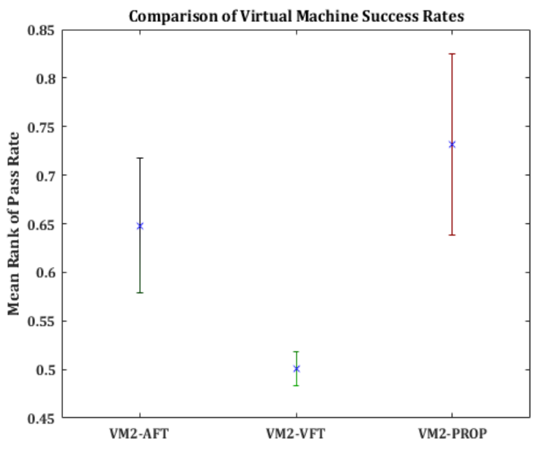
Virtual Node -1 Comparison Metrics						
Data Sources	ε_1 (%)	ε_2(%)	ε_3(%)	ε_4(%)	ε_5(%)	ε_6(%)
VFT	-11.0812	11.08117	25.83521	-0.10103	0.101032	0.308417
AFT	-43.4512	43.45116	60.19807	-0.39175	0.391750	0.297314

Table 6. Virtual Node-2 Performance Comparison of VFT, AFT with our Proposed Strategy

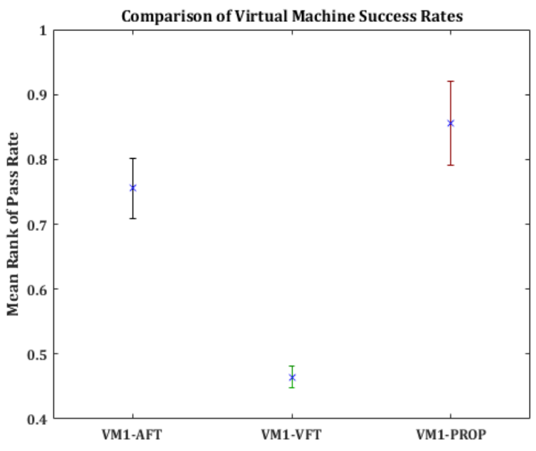
	Virtual Node -2 Comparison Metrics					
Data Sources	ε_1 (%)	ε_2(%)	ε_3(%)	ε_4(%)	ε_5(%)	ε_6(%)
VFT	-10.8510	10.85229	20.43801	-0.08423	0.08423	0.27140
AFT	-30.2022	30.20222	45.07832	-0.23141	0.23141	0.19480

Table 7. Virtual Node-3 Performance Comparison of VFT, AFT with our Proposed Strategy

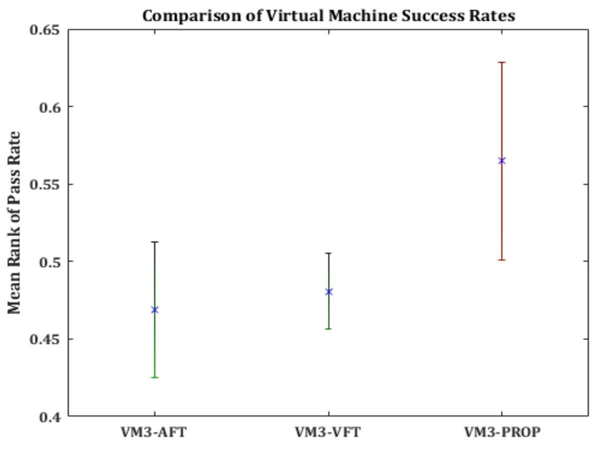
	Virtual Node -3 Comparison Metrics					
Data Sources	ε_1 (%)	ε_2(%)	ε_3(%)	ε_4(%)	ε_5(%)	ε_6(%)
VFT	-15.6587	16.5941	16.83521	-0.09576	0.09918	0.107865
AFT	-10.6525	19.8363	25.19807	-0.08409	0.11959	0.021855



(a)



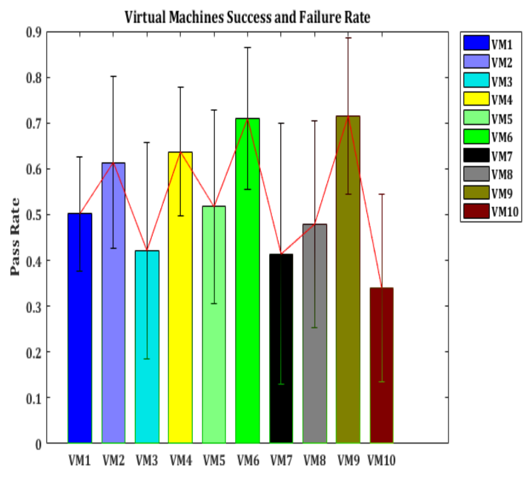
(b)



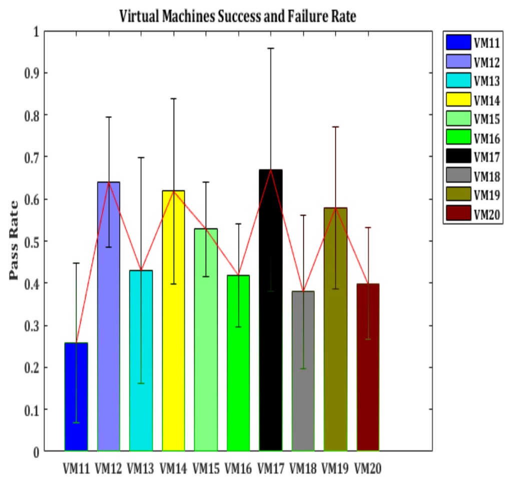
(c)

Figure 11. Mean Rank of Pass & Failure Rate with standard deviation (error bars)

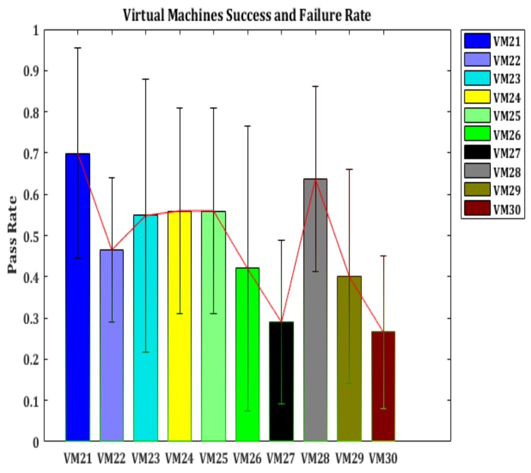
A Mean rank of success and failure rate of the VFT, AFT and our proposed strategy was conducted and the results are presented in Figure 11. We observed that both in Figure 11(a), 11(b) and 11(c) there is no overlap between our proposed strategy and VFT, which shows the significance of errors that exist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


Virtual Machines Success and Failure Rate
Pass Rate
VM1 VM2 VM3 VM4 VM5 VM6 VM7 VM8 VM9 VM10

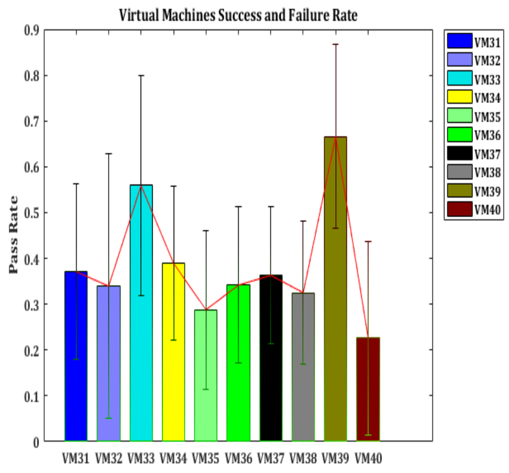
(a)


Virtual Machines Success and Failure Rate
Pass Rate
VM11 VM12 VM13 VM14 VM15 VM16 VM17 VM18 VM19 VM20

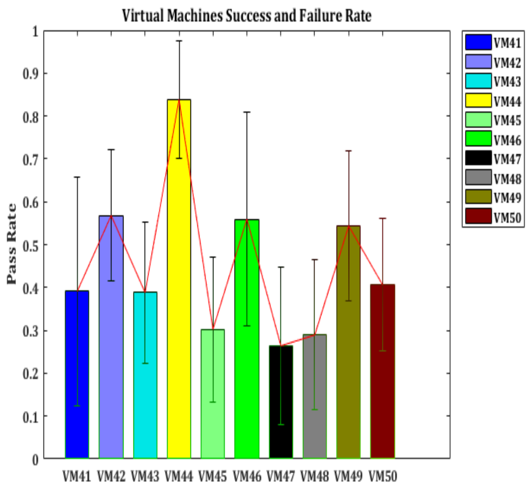
(b)


Virtual Machines Success and Failure Rate
Pass Rate
VM21 VM22 VM23 VM24 VM25 VM26 VM27 VM28 VM29 VM30

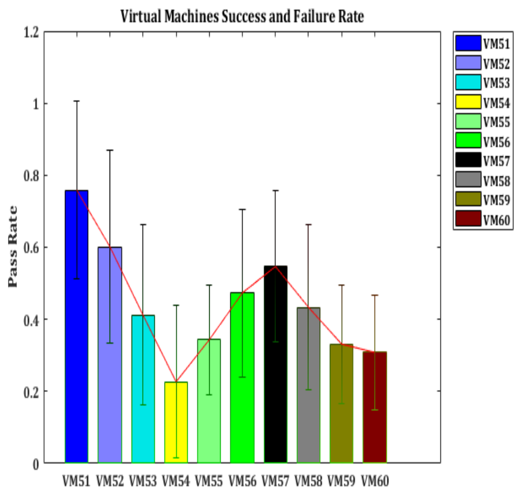
(c)


Virtual Machines Success and Failure Rate
Pass Rate
VM31 VM32 VM33 VM34 VM35 VM36 VM37 VM38 VM39 VM40

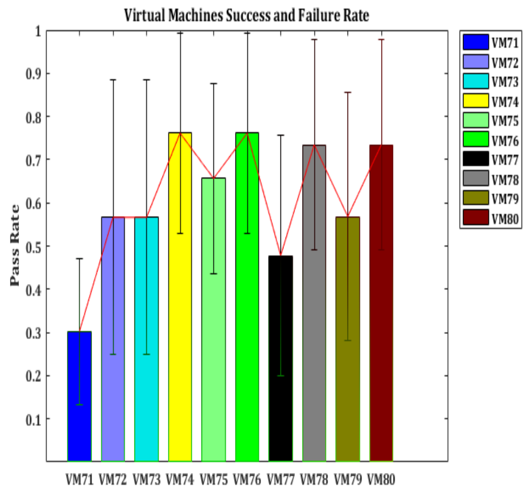
(d)


Virtual Machines Success and Failure Rate
Pass Rate
VM41 VM42 VM43 VM44 VM45 VM46 VM47 VM48 VM49 VM50

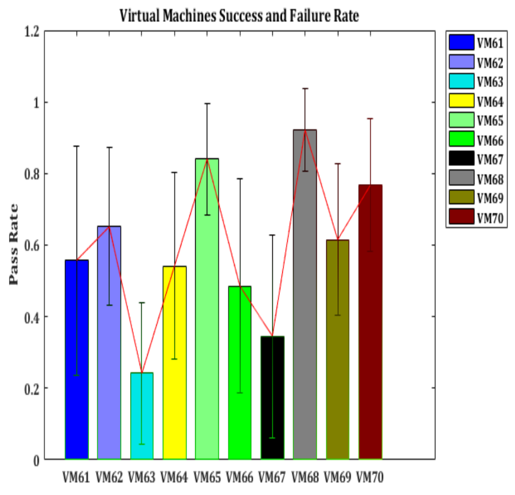
(e)


Virtual Machines Success and Failure Rate
Pass Rate
VM51 VM52 VM53 VM54 VM55 VM56 VM57 VM58 VM59 VM60

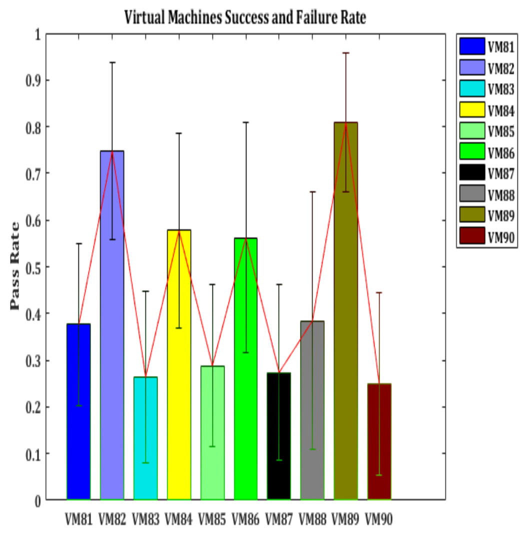
(f)



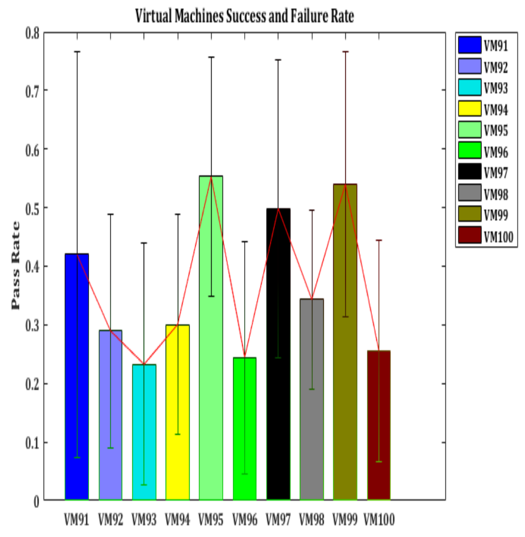
(g)



(h)



(i)



(j)

Figure 12. Simulation Plots for 100 Virtual Nodes (VM-1 to 100) with error bars

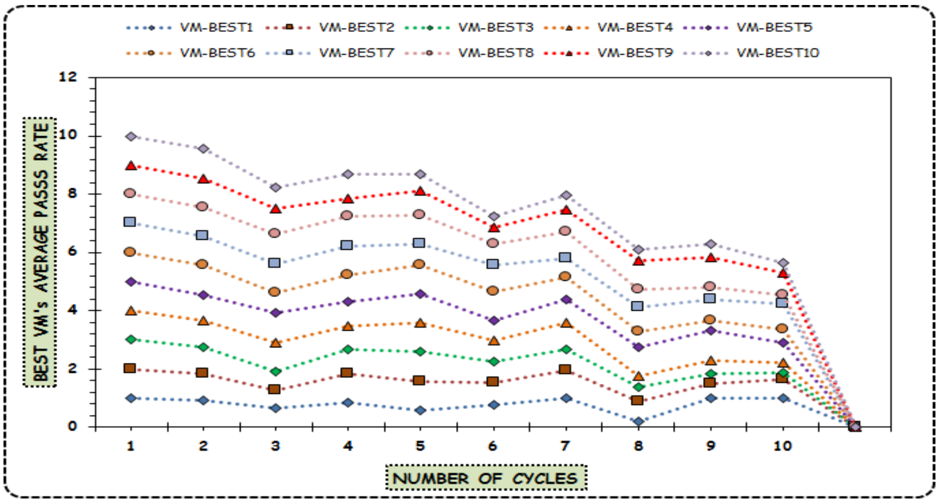


Figure 13. Average selected best Virtual machine from 1-100VMs

2. DISCUSSIONS

We conducted experiments using 3 virtual machines for performance comparison with other existing approaches (as shown in Figure 8), then we performed another experiment using six virtual machines after which we then performed the experiment using 100 virtual machines, which are both on a larger scale compared to the initial set up. In the first cycle, virtual machine-1, 2, 3, 4, 5 and 6 have the same pass rate but the result of Virtual Machine-1 (Node-1) has been selected because it has a lower task finish service time of 1700. In the same cycle Virtual Machine-5 did not pass the status check and the time task limit check also automatically failed. The Virtual Machine-2 output was selected by the decision mechanism in the second cycle because it has the highest pass rate of 0.87, while from cycle 3 to 4 the output of Virtual Machine-1was selected, as it has the highest pass rate among the competing virtual machines. (Details of some experimental details are presented in Table 7).

From cycle 5 to 7, the output of Virtual Machine-5 (Node-5) was selected because it has the highest pass rate among all other nodes. In cycle 5, Virtual Machine-4 and Virtual machine-6 does not pass the status check and the time task limit check also failed, with the same occurring in cycle 6 under Virtual Machine-6 and cycle 10 under Virtual Machine-4. Lastly the output of Virtual Machine-1was selected from cycle 8 to 10 because it has the highest pass rate.

In a similar scenario cycle 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 Cloud controller module and to the TTLC. Here it is received before the time limit, but because no result was produced TTLC status is also fail. The fault manager then tries to repair the fault generated by performing checkpoint. As stated earlier, the result of our simulations are presented in Figure 7 where we saw that increase in PR is more than decrease, hence we can achieve a good performance of our algorithm.

Having assumed at the beginning that pass rate is 0.5, Figure 7(a) shows the pass rate analysis in Virtual machine-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 virtual machine-1 because increase in PR is more than decrease.

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

Additionally, we performed an experiment using 100 virtual machines and 10 computing cycle, where we studied the success and failure rate pattern 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 pass rate to enable us observe the overlap and access the level of significance among the VMs.

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

In a similar scenario presented in Figure 12(b), it was observed that VM12 to 20 had high success rate, which implies that the VMs across that range have a lesser tendency to fail, i.e. the pass rates are higher than the failure rates. Similar success and failure patterns were observed from Figure 12(c) to 12(j) which shows the reliability of across the nodes.

Based on the results obtained the following observations were made:

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

Figure 13 shows the overall average selected best virtual machine with a high pass rate and less failure rate. This was selected by computing the output obtained from the group of 10x10 virtual machines after each computing cycle across 100 nodes.

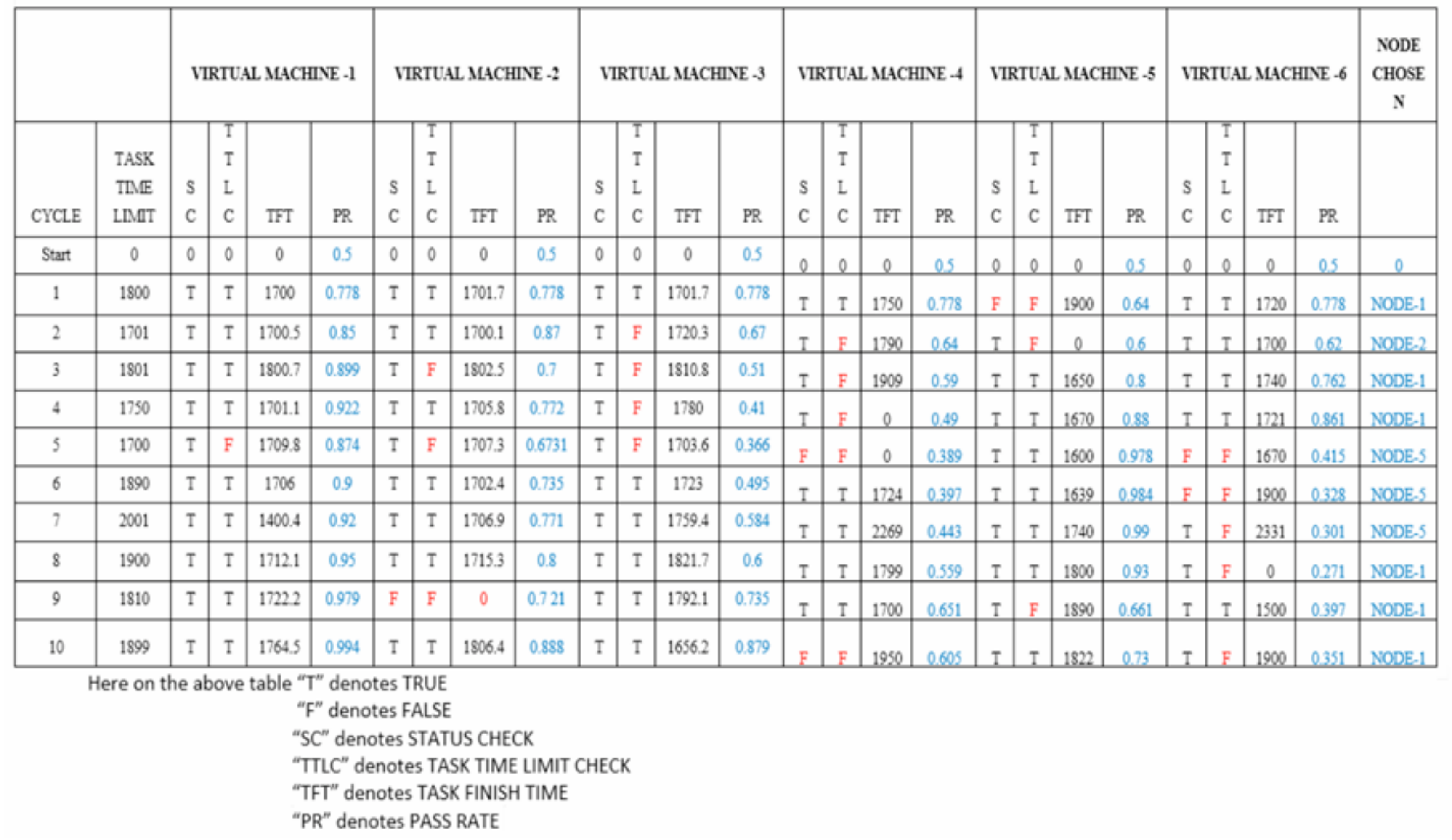
3. Conclusion

This paper presented an optimized fault tolerance approach for real time computing on the cloud infrastructure. The proposed strategy uses the pass rate 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 a fault tolerance 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 fault tolerance based on high pass rate of computing nodes and less service Task finish time, increasing the system availability.

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

In 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 analyse real time cloud failure data, including the root cause of failures and statistics by applying machine learning techniques such as clustering and anomaly finding. These will aid re-examining other current algorithms and techniques for fault tolerant cloud system, creating realistic benchmarks and test beds for fault tolerance testing.

Table 8. Simulation results for the Pass rate Assessment



		VIRTUAL MACHINE -1				VIRTUAL MACHINE -2				VIRTUAL MACHINE -3				VIRTUAL MACHINE -4				VIRTUAL MACHINE -5				VIRTUAL MACHINE -6				NODE CHOSEN
CYCLE	TASK TIME LIMIT	SC	TTLC	TFT	PR	SC	TTLC	TFT	PR	SC	TTLC	TFT	PR	SC	TTLC	TFT	PR	SC	TTLC	TFT	PR	SC	TTLC	TFT	PR	
Start	0	0	0	0	0.5	0	0	0	0.5	0	0	0	0.5	0	0	0	0.5	0	0	0	0.5	0	0	0	0.5	0
1	1800	T	T	1700	0.778	T	T	1701.7	0.778	T	T	1701.7	0.778	T	T	1750	0.778	F	F	1900	0.64	T	T	1720	0.778	NODE-1
2	1701	T	T	1700.5	0.85	T	T	1700.1	0.87	T	F	1720.3	0.67	T	F	1790	0.64	T	F	0	0.6	T	T	1700	0.62	NODE-2
3	1801	T	T	1800.7	0.899	T	F	1802.5	0.7	T	F	1810.8	0.51	T	F	1909	0.59	T	T	1650	0.8	T	T	1740	0.762	NODE-1
4	1750	T	T	1701.1	0.922	T	T	1705.8	0.772	T	F	1780	0.41	T	F	0	0.49	T	T	1670	0.88	T	T	1721	0.861	NODE-1
5	1700	T	F	1709.8	0.874	T	F	1707.3	0.6731	T	F	1703.6	0.366	F	F	0	0.389	T	T	1600	0.978	F	F	1670	0.415	NODE-5
6	1890	T	T	1706	0.9	T	T	1702.4	0.735	T	T	1723	0.495	T	T	1724	0.397	T	T	1639	0.984	F	F	1900	0.328	NODE-5
7	2001	T	T	1400.4	0.92	T	T	1706.9	0.771	T	T	1759.4	0.584	T	T	2269	0.443	T	T	1740	0.99	T	F	2331	0.301	NODE-5
8	1900	T	T	1712.1	0.95	T	T	1715.3	0.8	T	T	1821.7	0.6	T	T	1799	0.559	T	T	1800	0.93	T	F	0	0.271	NODE-1
9	1810	T	T	1722.2	0.979	F	F	0	0.7 21	T	T	1792.1	0.735	T	T	1700	0.651	T	F	1890	0.661	T	T	1500	0.397	NODE-1
10	1899	T	T	1764.5	0.994	T	T	1806.4	0.888	T	T	1656.2	0.879	F	F	1950	0.605	T	T	1822	0.73	T	F	1900	0.351	NODE-1

Here on the above table "T" denotes TRUE
"F" denotes FALSE
"SC" denotes STATUS CHECK
"TTLC" denotes TASK TIME LIMIT CHECK
"TFT" denotes TASK FINISH TIME
"PR" denotes PASS RATE

4. REFERENCES

[1] K. Bilal, O. Khalid, S. U. . Malik, M. U. S. Khan, S. . Khan, and A. . Zomaya, “Fault Tolerance in the Cloud,” *“Fault Tolerance in the Cloud” Encyclopedia on Cloud Computing*. John Wiley & Sons, Hoboken, NJ, USA, 2015, pp. 291–300, 2015.

[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] A. Bala and I. Chana, “Fault Tolerance- Challenges , Techniques and Implementation in Cloud Computing,” *Int. J. Comput. Sci.*, vol. 9, no. 1, pp. 288–293, 2012.

[6] S. Malik and F. Huet, “Adaptive Fault Tolerance in Real Time Cloud Computing,” *2011 IEEE World Congr. Serv.*, pp. 280–287, Jul. 2011.

[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] A. Greenberg, J. Hamilton, D. a Maltz, and P. Patel, “The cost of a cloud: research problems in data center networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 68–73, 2008.

[12] ITProPortal, “ITProPortal.com: 24/7 Tech Commentary & Analysis,” 2012. [Online]. Available: http://www.itproportal.com/. [Accessed: 24-Jun-2015].

[13] Z. Pantic and M. Babar, “Guidelines for Building a Private Cloud Infrastructure,” 2012.

[14] A. Sen and S. Madria, “Off-Line Risk Assessment of Cloud Service Provider,” *2014 IEEE World Congr. Serv.*, pp. 58–65, Jun. 2014.

[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] H. S. Paul, A. Gupta, and R. Badrinath, “Performance comparison of checkpoint and recovery protocols,” *Concurr. Comput. Pract. Exp.*, vol. 15, no. 15, pp. 1363–1386, 2003.

[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] K. Singh, S. Smallen, S. Tilak, and L. Saul, “Failure analysis and prediction for the CIPRES science gateway Kritika,” *Concurr. Comput. Pract. Exp.*, vol. 22, no. 6, pp. 685–701, 2016.

[20] M. Fu, L. Zhu, D. Sun, A. Liu, L. Bass, and Q. Lu, “Runtime recovery actions selection for sporadic operations on public cloud,” *Softw. - Pract. Exp.*, vol. 39, no. 7, pp. 701–736, 2016.

[21] G. Chen, H. Jin, DeqingZou, B. B. Zhou, and W. Qiang, “A lightweight software fault-tolerance system in the cloud enviroment,” *Concurr. Comput. Pract. Exp.*, vol. 22, no. 6, pp. 685–701, 2015.

[22] X. Pei, X. M. YijieWang, and F. Xu, “Repairing multiple failures adaptively with erasure codes in distributed storage systems Xiaoqiang,” *Concurr. Comput. Pract. Exp.*, vol. 22, no. 6, pp. 685–701, 2015.

[23] C. Bertolli and M. Vanneschi, “Fault tolerance for data parallel programs C.,” *Concurr. Comput. Pract. Exp.*, vol. 22, no. 6, pp. 685–701, 2011.

[24] A. Maloney and A. Goscinski, “A survey and review of the current state of rollback-recovery for cluster systems,” *Concurr. Comput.*

Pract. Exp., vol. 22, no. 6, pp. 685–701, 2009.

[25] H. N. Alshareef and D. Grigoras, “Robust cloud management of MANET checkpoint sessions,” *Concurr. Comput. Pract. Exp.*, vol. 22, no. 6, pp. 685–701, 2016.

[26] D. W. Bin Hong, Fuyang Peng, Bo Deng, Yazhou Hu, “DAC-Hmm: detecting anomaly in cloud systems with hidden Markov models,” *Concurr. Comput. Pract. Exp.*, vol. 22, no. 6, pp. 685–701, 2015.

[27] P. Chen, Y. Xia, S. Pang, and J. Li, “A probabilistic model for performance analysis of cloud infrastructures,” *Concurr. Comput. Pract. Exp.*, vol. 22, no. 6, pp. 685–701, 2015.

[28] M. A. Salehi, B. Javadi, and R. Buyya, “Resource provisioning based on preempting virtual machines in distributed systems Mohsen,” *Concurr. Comput. Pract. Exp.*, vol. 22, no. 6, pp. 685–701, 2013.

[29] M. Nazari Cheraghlou, A. Khadem-Zadeh, and M. Haghparast, “A Survey of Fault Tolerance Architecture in Cloud Computing,” *J. Netw. Comput. Appl.*, vol. 61, pp. 81–92, 2015.

[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] J. Kaur and S. Kinger, “Efficient Algorithm for Fault Tolerance in Cloud Computing,” *2014 IJCSIT Int. J. Comput. Sci. Inf. Technol.*, vol. 5, pp. 6278–6281, 2014.

[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] M. Amoon, “A job checkpointing system for computational grids,” *Open Comput. Sci.*, vol. 3, no. 1, pp. 17–26, 2013.

[36] S. Siva Sathya and K. Syam Babu, “Survey of fault tolerant techniques for grid,” *Comput. Sci. Rev.*, vol. 4, no. 2, pp. 101–120, 2010.

[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] E. Okorafor, “A fault-tolerant high performance cloud strategy for scientific computing,” *IEEE Int. Symp. Parallel Distrib. Process. Work. Phd Forum*, pp. 1525–1532, 2011.

[41] R. Nogueira, F. Araujo, and R. Barbosa, “CloudBFT: Elastic Byzantine Fault Tolerance,” *2014 IEEE 20th Pacific Rim International Symposium on Dependable Computing.*

[42] N. Yadav and S. K. Pandey, “FAULT TOLERANCE IN DCDIDP USING HAProxy,” pp. 231–237.

[43] H. K. H. Kim, S. K. S. Kang, and H. Y. Yeom, “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)*, vol. 2, no. 1, pp. 6–12, 2003.

[44] D. Sheng and C. Franck, “GloudSim: Google trace based cloud simulator with virtual machines,” *Softw. - Pract. Exp.*, vol. 39, no. 7, pp. 701–736, 2015.

[45] W. Qiang, C. Jiang, L. Ran, D. Zou, and H. J. Services, “CDMCR: multi-level fault-tolerant system for distributed applications in cloud,” *Int. J. Appl. Eng. Res.*, vol. 9, no. 22, pp. 5968–5974, 2015.

[46] A. Agbaria and R. Friedman, “Virtual-machine-based heterogeneous checkpointing,” *Softw. - Pract. Exp.*, vol. 32, no. 12, pp. 1175–1192, 2002.

[47] D. Singh, J. Singh, and A. Chhabra, “High availability of clouds: Failover strategies for cloud computing using integrated checkpointing algorithms,” *Proc. - Int. Conf. Commun. Syst. Netw. Technol. CSNT 2012*, pp. 698–703, 2012.

[48] G. Jung, K. R. Joshi, M. A. Hiltunen, R. D. Schlichting, and C. Pu, “Performance and availability aware regeneration for cloud based multitier applications,” *Proc. Int. Conf. Dependable Syst. Networks*, pp. 497–506, 2010.

[49] M. Chtepen, F. H. a Claeys, B. Dhoedt, F. De Turck, P. Demeester, and P. a. Vanrolleghem, “Adaptive Task Checkpointing and Replication: Towards Efficient Fault-Tolerant Grids,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 2, pp. 180–190, 2008.

[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] J. Elliott, K. Kharbas, D. Fiala, F. Mueller, K. Ferreira, and C. Engelmann, “Combining partial redundancy and checkpointing for HPC,” *Proc. - Int. Conf. Distrib. Comput. Syst.*, pp. 615–626, 2012.

[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. (Box) 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] E. Bin, O. Biran, O. Boni, E. Hadad, E. K. Kolodner, Y. Moatti, and D. H. Lorenz, “Guaranteeing high availability goals for virtual machine placement,” *Proc. - Int. Conf. Distrib. Comput. Syst.*, pp. 700–709, 2011.

[60] D. Sun, G. Chang, C. Miao, and X. Wang, “Analyzing, modeling and evaluating dynamic adaptive fault tolerance strategies in cloud computing environments,” *J. Supercomput.*, vol. 66, no. 1, pp. 193–228, 2013.

[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.