International Journal of Cloud Applications and Computing
Volume 13 • Issue 1

An Adaptable Approach to Fault Tolerance in Cloud Computing

Priti Kumari, Jaypee Institute of Information Technology, India
Parmeet Kaur, Jaypee Institute of Information Technology, India*

ABSTRACT

Existing fault tolerance approaches in the cloud are broadly based on replication and checkpointing. Each of these approaches has its advantages and limitations. This paper presents an adaptable fault tolerance method for determining which of the two approaches will be appropriate for the successful execution of a task in the given cloud conditions. The proposed method classifies the failure risk of host machines available for task execution based on their failure history. Subsequently, fuzzy logic is used to determine the appropriate fault tolerance approach by considering a host's failure risk, user-defined task's priority, and level of resource redundancy. Setting a task's priority provides a user with control to solicit a desired fault tolerance level while the availability of resources reflects a cloud provider's capability to offer fault tolerance. Simulation experiments have verified that the proactive selection of a fault-tolerance method increases the number of tasks that complete successfully.

KEYWORDS

Checkpointing, Classification, Cloud Computing, Fault Tolerance, Fuzzy Logic, Replication

1. INTRODUCTION

The extensive use of cloud computing services has been a consequence of the possibility of flexible and elastic provisioning of resources based on user requirements in a pay-per-use manner. Users are no longer required to buy, manage, or upgrade their computing resources (Zhou et al., 2017). Instead, they can get various information technology capabilities, such as computing, storage, memory, networks, software, etc., as services at low costs and without the hassles of owning or maintaining the resources. On the other side, cloud service providers benefit due to the scale of their service usage along with the idea of multi-tenancy. A multitenant architecture allows a cloud provider to share resources among multiple users; thereby, incurring a lower cost. It is also possible for a user to request dedicated resources though these will come at an increased cost (Kumari & Kaur, 2020).

Even though cloud computing services are being used for a huge number of applications, there are concerns related to failures and therefore, degradation of performance in the cloud. Failures affect the reliability as well as the availability of services and need to be handled effectively. Therefore, fault tolerance is one of the key elements of Service Level Objectives (SLOs) for cloud services' users (Buyya et al., 2009). FT is the capability of a device or system to continue its operations in

DOI: 10.4018/IJCAC.319032 *Corresponding Author

This article published as an Open Access article distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/4.0/) which permits unrestricted use, distribution, and production in any medium, provided the author of the original work and original publication source are properly credited.

the case where some of its components may have failed (Menychtas & Konstanteli, 2012) (Hasan & Goraya, 2018). Existing approaches to fault tolerance in the cloud are based on traditional distributed computing approaches that fall broadly into two categories: reactive and proactive. Reactive approaches include recovery protocols based on checkpointing and replication protocols and these approaches handle faults after they occur. In contrast, proactive approaches attempt to estimate the occurrence of failures before they occur. Examples include approaches to job migration and self-healing. Of the two categories, reactive approaches have found a wider implementation in cloud computing due to their simplicity and ease of implementation (Kumari & Kaur, 2018) (Saikia & Devi, 2014).

This paper explores the application of the reactive approaches of fault tolerance, i.e., checkpointing and replication, in the Infrastructure as a service (IaaS) cloud. Checkpointing is a well-known technique in which the state of an active virtual machine (VM) is saved periodically during normal execution. In case a failure occurs, the task needs to roll back to the last saved checkpoint instead of restarting; thus, preventing loss in execution. In cloud computing, checkpointing has been widely used to periodically save an active virtual machine's state for recovery on a physical machine other than the one on which it is hosted (Zhou et al., 2017). The VM can retrieve its last saved error-free state if the failure occurs before its completion. On the other hand, replication of a task or a resource implies maintaining a copy of the same. Here, the copy can be used to replace the primary task or resource in case of failure. Replication is based on the use of redundancy to handle failures and so has been used in cloud computing wherever resources are, in general, available in abundance.

Each of the two approaches, i.e., checkpointing and replication have its own benefits and limitations. Checkpointing uses much lower resources as compared to replication since only the state of a task needs to be saved periodically. However, checkpointing involves overhead at the time of saving checkpoints as well as during recovery (Zhou et al., 2016). Further, a checkpoint allows recovery from a single failure since simultaneous failures of the machines hosting the VM and storing its checkpoint cannot be handled. In comparison, k-replication, i.e., saving k replicas guarantees recovery from up to k-1 failures though it involves the use of k times the actually needed resources. Thus, it is evident that the selection of a fault-tolerance approach needs to be judicious for the best results with minimum resource consumption.

The present work proposes an adaptable approach to the selection of the appropriate FT approach in the cloud computing environment. The projected scheme is adaptable to the specific environment of implementation as it considers factors that reflect both a user's interests as well as the cloud provider's capabilities. Firstly, the historic data of physical machines in a cloud is used to classify the machines based on their risk of failure. This classification takes into account the recorded downtime of a machine, the nature of the fault, the physical machine's type, and its purpose. We have used the k Nearest Neighbour's (KNN) algorithm (Sharma, 2017) (Sun et al., 2018) to cluster the physical machines into high, medium, and low-risk categories of failures. Each machine has then been assigned a risk label depending on the cluster in which it lies. Subsequently, fuzzy logic has been used to frame rules to determine the appropriate fault tolerance approach for VMs hosted by a particular physical machine. The fuzzy rules are based on three input variables. The first is the VM's priority level as specified by the end-user. Secondly, we consider a physical machine's risk label, as assigned by the clustering algorithm, and the last input variable is taken as the level of storage availability. Here, the first input variable reflects the end user's interest while the other two variables are an indicator of the cloud provider's capabilities. The fuzzy rules aim to provide a result, i.e., checkpointing or replication, after a comprehensive consideration of both the stakeholders' interests. Simulation results have verified that the proposed approach outperforms checkpointing or replication approaches when performed alone.

The highlights of the paper are as follows:

- Proposes a novel classification and fuzzy rule-based fault tolerance approach.
- The use of a fuzzy inference system captures the real-world user requirements and a cloud provider's abilities.

- The proposed approach is adaptable and hence, can be used in a dynamic manner based on the actual system state.

The rest of the text is structured as follows: Section 2 deliberates upon the checkpointing and replication techniques. Related work on fault tolerance and fuzzy logic is also discussed in the section. The proposed approach is described in Section 3. Section 4 discusses the simulation results. Finally, this paper is winded up in Section 5.

2. BACKGROUND AND RELATED WORK

The dynamic nature of the cloud system rises the chance of failure. Therefore, fault tolerance (FT) methods are imperative to evade the consequence of failures in the cloud system. FT methods enable systems or devices to provide essential services even in the existence of faulty components. These assist in the detection and management of system failures caused by either hardware or software issues. According to the literature, FT methods for cloud systems are usually based on two approaches: reactive and proactive (Kumari & Kaur, 2018) (Amoon, El-Bahnasawy, Sadi, & Wagdi, 2019).

The reactive FT scheme is utilized to diminish the effect of breakdown in the cloud after the occurrence of the failures. The most repeatedly used reactive FT schemes are replication as well as checkpointing (Amin et al., 2015) (Ray et al., 2020). In comparison, the proactive FT methods are used to anticipate the failure in advance and replace the suspicious component with some running elements. The main purpose of the proactive FT approach is to avoid the existence of failures and evade the recovery process of the reactive FT approaches. Although the use of these methods aims to reduce the number of upcoming failures, these are less practically feasible in cloud systems (Liu et al., 2016) (Prakash & Vyas, 2022).

2.1 Background

To handle the failure in the cloud system reactively, the most commonly used methods are replication and checkpointing. The replication fault-tolerance method maintains multiple copies of VMs or tasks and places them in diverse locations within a data center (Zhou et al., 2016). In case of a task or VM's failure, execution can continue as long as a copy or replica is available. The main advantages of using replication are that it provides higher availability and reliability, causes less data movement over the network, and thus, has a lower communication overhead. However, resuming the replica VM or task consumes more resources in terms of time and network. In other words, implementing a replication method is expensive since it consumes more storage space and incurs an overhead of updating the replicas (Kumari & Kaur, 2020) (Agarwal & Kotakula, 2022).

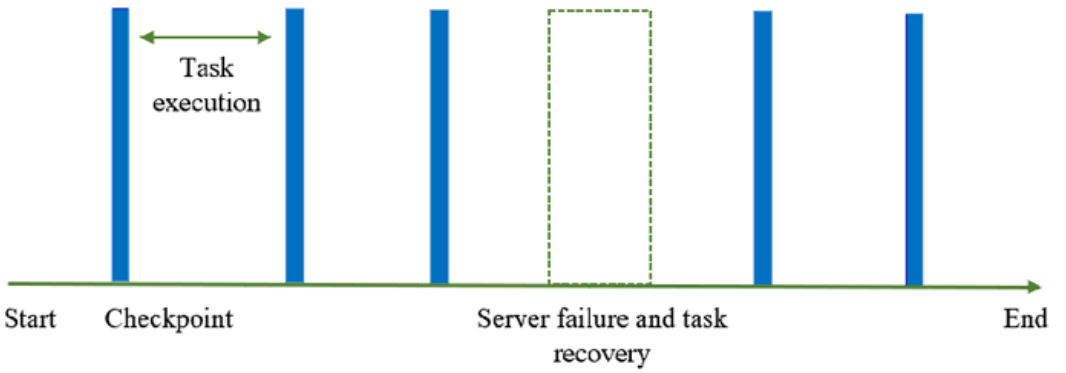
On the other side, the checkpointing method is used to save the error-free execution state of the system regularly to stable storage. In the event of a failure, the cloud system can go on with the request from the last preserved checkpoint state rather than restarting from the beginning. (Zhou et al., 2017). The checkpointing process is demonstrated in figure 1. The main advantage of implementing the checkpointing method is that it is less expensive than replication. However, saving checkpoints intermittently and resuming a failed service is time-consuming as well as incurs additional overheads during normal working (Zhao et al., 2016).

Thus, each of the two methods has its pros and cons and a judicious choice of the appropriate method is important.

2.2 Related Work

The key motive of applying FT techniques in the cloud system is to improve accessibility as well as reliability. In order to accomplish this goal, significant research has been undertaken in this field resulting in replication as well as checkpointing-based protocols.

Figure 1. Checkpointing process



Zhou et al. (2016) have projected an FT method based on replication to enhance the consistency of server-based cloud services. The authors have used a heuristic procedure to select an appropriate host server to keep the replica of VMs. Patra, et al. (2016) have projected an adaptive model to forecast and handle the failure that occurs in real-time cloud computing. The proposed adaptive model uses replication as well as the resubmission fault tolerance method. Another replication and resubmission method based on the reliability requirement has been presented by E. Abdelfattah et al. (2017). If the server with the maximum reliability fails, then the task or VM executing on this server is scheduled at the server having the second-best reliability. A two-stage FT technique is presented by Wu, Y. et al. (2019). In the first state, the authors have developed an algorithm to rank the services according to their significance. The replication strategies for services are changed according to their ranks. An A*-based heuristic alternative path finding procedure is implemented in the second stage to find suitable replacement systems to handle the failure of complex services.

A peer-to-peer checkpointing-based method for the cloud data center has been presented by J. Zhao, et al. (2018). Using this method, the checkpoint images of VMs are saved on the peer or neighboring server. A. Zhou, et al. (2017) have presented an EDCKP (edge switch failure-aware) fault tolerance method to handle the failure of the server as well as edge switch of fat-tree topology in a cloud data center. In this scheme, the checkpoint images of VMs are placed on the server that is in the same pod. This scheme improves the reliability of cloud services and also consumes fewer network resources. Amoon, et al. (2019) have presented a checkpointing method that employs a flexible checkpointing interval. The extent of the interval is defined based on the failure rate of the server where VMs are executing to finish the execution of the application. A scheduling method, presented by Chinnathambi, et al. (2019), is used to monitor the performance of a virtual node as well as an effective checkpointing scheme to bear the migration of tasks and jobs with slight overhead. Moreover, the proposed method efficiently identifies the VM's faults and immediately transfers the task to some working VM.

Further, it has been observed that research in this field has employed methods of soft computing for the development of effective FT methods. These methods are tolerant of uncertainty, imprecision, and approximation and provide traceability, robustness, and low-cost optimal solutions to difficult computational problems (Gupta et al., 2020) (Ejimogu & Başaran, 2017). The most commonly used soft computing approaches are evolutionary computing, artificial neural networks (ANN), and fuzzy logic (Srivastava & Srivastava, 2014). Of the soft computing methods, fuzzy logic deals with imprecise and vague information to make the best possible decision for the given input. Therefore, fuzzy logic finds application in all problems involving uncertain or imprecise data. Since cloud systems are essentially complex and dynamic systems, soft computing methods have found application in these systems.

M.A.H. Monil et al. (2016) presented an approach based on fuzzy logic for choosing as well as migrating VMs in the cloud. They have tested and evaluated the performance of the projected algorithm on the CloudSim toolkit. Their simulation outcomes demonstrated that this scheme is more energy-efficient compared to others. A proactive autoscaling prototype for a cloud system has been developed by D. Tran et al. (2017). This model consists of two components for prediction and scaling decisions. The prediction is done on the time-series data and for that, they have used the fuzzy method, neural network, and genetic algorithm. For the scaling decision, the author has used the predicted value as well as SLA violation approximations to accomplish final resource allocation. The testing of the proposed solution on real workload data has verified the feasibility of the proposed approach as well as its higher efficiency as compared to existing methods.

A fuzzy logic procedure and prediction scheme have been projected by D.M. Bui et al. in (2018) to proactively recognize the faults that may arise in the cloud system. According to the authors, the proposed method can offer improved performance in terms of reaction rate as well as accuracy, which enhances the reliability of the cloud system. M. N. Cheraghlou et al. (2019) have proposed a scheme called CFTFIS that evaluates the architecture of FT as well as determines the level of FT on the basis of fuzzy patterns. The proposed scheme consists of 3 parameters, namely policies, techniques for fault detection, and fault recovery methods. The architecture of FT and the improved percentage of projected ability are accomplished at five diverse fuzzy levels. In case the susceptibility of a cloud-based system to failures is recognized, a correct and appropriate choice of architecture is employed in the system. Rezaeipanah et al. (2022) put forth an adaptive method to identify the breakdowns in the cloud-based system in the initial stage. In order to monitor the system, a prediction technique has been used by the authors. Further, to identify the faults, they have used an algorithm based on fuzzy logic. In order to facilitate error-free task scheduling, Sathiyamoorthi et al. (2021) suggested an effective and adaptive fault-tolerant scheduling approach. To achieve optimal cloud computing infrastructure dependability and availability, Attallah et al. (2021) proposed a methodology that tolerates virtual machine CPU failures. By combining proactive and reactive techniques, Rawat et al. (2021) introduced a new Threshold-Based Adaptive Fault Tolerance (TBAFT) strategy in the cloud.

By analyzing the literature, it was observed that the existing works, in general, rely on anyone FT technique without considering the features of the cloud environment or the task being performed. However, without the use of an appropriate FT method, it is difficult to achieve reliability and availability. Hence, there is a requirement to develop a scheme that can adapt to the features of the system being considered. Therefore, this work has presented a fault tolerance method based on classification and fuzzy logic that selects an appropriate FT scheme for a given system.

3. PROPOSED APPROACH

The paper presents an adaptable approach to fault tolerance in IaaS clouds which provide computing resources as a service to end-users through virtualization technology. A physical machine (PM) present in the cloud is capable of hosting one or more virtual machines (VM) depending on its own and the hosted VM's configuration. Each VM is assumed to be executing a user's task. The failure of a VM will prevent or delay the completion of task execution resulting in reduced user satisfaction as well as loss to the cloud provider. A VM may fail due to multiple software or hardware-related causes. Of these, the software failures are a result of the task or application executing on the VM and are beyond a service provider's scope. However, a PM's failure is a risk that can be predicted based on a PM's historical data. Therefore, this work uses data on PMs' previous failures to label a PM according to its risk of failure. Subsequently, the risk labels of PMs are taken along with a task's priority level and the level of storage available in the cloud to determine the fault tolerance approach suitable for the task.

3.1 Clustering and Labeling PMs Based on Failure Data

As a first step towards fault tolerance, we consider the physical machines on which virtual machines are hosted. Data related to the past occurrences of PM failures are analyzed with the objective to determine the risk of a PM's failure in the future. The dataset from a cloud provider includes various features with regard to a PM's failure. The work has used the following features for risk labeling: the cause of failure, the downtime during a failure, and the physical machine's type and its purpose. The cause of fault indicates whether a fault is likely to recur or not. For instance, hardware faults may recur as they are related to the machine but a software fault may have been due to the hosted VM or the task executed on it and are not likely to happen again. Downtime is used as it is expected to reflect the fault severity.

KNN algorithm (Rong et al., 2016) has been used for unsupervised clustering of the failure data. It is a non-parametric algorithm and does not assume any specific data distribution and thus, suits the real-world failure data. Further, the data is unlabelled and KNN is used to firstly group the data into three clusters based on feature similarity (Java, n.d.). Then, the formed clusters were analyzed and given a risk label, i.e., *high, medium*, and *low*. Thus, PMs lying in the cluster with labels as (*high/medium/low*) are thus classified to be PMs with (high /medium/low) risk of failure. Clusters formed by the KNN algorithm on the NERSC (Mohammed et al., 2018) dataset and the predicted outcomes and its analysis in terms of precision, recall as well as F1-measure are carried out in table 1. The overall accuracy obtained is 99.98%. Subsequently, label tuning is performed on the clustered value to label the machine into high, moderate, and low risk.

3.2 Fault Tolerance Method Selection Fuzzy Inference System (FTMSFIS)

Fuzzy logic is a method of reasoning and decision-making that is often used in systems involving uncertain, imprecise, or vague information. It is based on the concept of relative graded membership motivated by human perception and cognition process (Bui & Lee, 2018). Fuzzy logic is applicable in problems involving human expertise, little data, or where input parameters are linguistic in nature or do not possess sharp or crisp boundaries (Cheraghlou et al., 2019). Since it provides an effective way for the resolution of multi-criteria decision problems, it has been used in the development of intelligent systems for optimization, pattern recognition, control, project risk assessment, trust evaluation systems for cloud service providers, and many more. The present work employs fuzzy logic to derive an effective method of FT in the cloud. The choice of fuzzy logic is depending on the fuzzy nature of input parameters of storage availability, the machine's risk of failure, and a user's priority level. It is postulated that a fuzzy logic-based system resembling human cognition will yield effective results for the current problem. The proposed fuzzy rule-based system, referred to as the Fault Tolerance Method Selection Fuzzy Inference System (FTMSFIS) selects an appropriate FT method to handle the failures in the cloud. FTMSFIS is a Mamdani fuzzy inference system (FIS) (Rezaeipanah et al., 2022) to adapt the input parameters to two output classes of replication and checkpointing.

Figure 2 demonstrates the FTMSFIS model. This fuzzy system consists of three attributes as input variables namely: Machine Risk (MR), Storage Availability (SA), and Task Priority (TP). The single output variable is the Fault Tolerance Method (FTM). A detailed description of values for input and output variables is summarized in Table 2.

Table 1. Statistical analysis

	High	Medium	Low
Precision	100	83	100
Recall	100	100	91
F1 -Measure	100	91	95

Figure 2. Proposed model of FTMSFIS

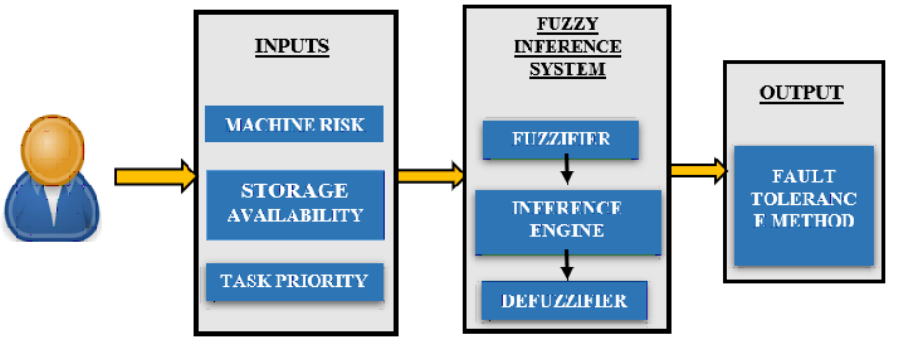


Table 2. System variables, linguistic terms, and Interval endpoints

Variables	Variable type	Linguistic Terms	Interval endpoints	
			Scale	Points
Machine Risk	Input	Low	(0 -2)	(0, 0, 1)
		Moderate		(0, 1, 2)
		High		(1, 2, 2)
Storage Availability	Input	Less	(0 -100)	(0, 0, 10, 30)
		Medium		(20, 50, 80)
		High		(40, 60, 100, 100)
Task Priority	Input	Low	(0 -10)	(0, 0, 3)
		Medium		(2, 5, 8)
		High		(6, 8, 10, 10)
Fault Tolerance Method	Output	Checkpointing	(0 -80)	(0, 30, 50)
		Replication		(30, 45, 80)

- **Fuzzy Inference System (FIS):** A FIS is composed of a fuzzifier, inference engine, and defuzzifier. The term fuzzifier is used to translate input parameters into linguistic terms. The defuzzifier transforms the inference engine's outcome into a crisp value. The inference engine is responsible for acquiring output using well-defined rules and kept in the database.
- **Input Variables:** See below:
  - **Machine Risk (MR):** This variable denotes the types of risk associated with the physical machine. This variable is represented over a scale of 0-2. The variable is signified using the 3 fuzzy sets such that low, moderate, and high.
  - **Storage Availability (SA):** This variable signifies the amount of storage available at the particular physical machine. The SA is described in terms of percentage, i.e., over a scale of 0-100. This variable is represented using the three fuzzy sets such that less, medium, and high.

- **Task Priority (TP):** This attribute signifies the priority associated with the tasks i.e., being executed on the virtual machine. This variable is represented over a scale of 0-10 using the 3 fuzzy sets, low, medium, and high.

- **Output Variable:** This fuzzy system has only one output variable i.e., *Fault Tolerance Method (FTM).* The FTM denotes which type of fault tolerance technique is selected to deal with failure proactively. This variable is represented using the two fuzzy sets of checkpointing and replication and denoted over a scale of 0-80.
- **Membership Function for Input and Output Variables:** In this paper, to decrease the complications, the trapezoid and triangle membership function is used for the input as well as output variables. The fuzzy set membership of the variable machine risk, storage availability, task priority, and the fault tolerance method is described as follows:
  - **Machine Risk (MR):** This variable is defined using three fuzzy sets: Low, Moderate, and High, as depicted in figure 3. The horizontal and the vertical axis denote the input value and degree of membership of the attribute MR respectively.
  - **Storage Availability (SA):** This input variable is defined using three fuzzy sets, Less, Medium, and High, as presented in figure 4.
  - **Task Priority (TP):** This variable is defined using 3 fuzzy sets as Low, Medium, as well as High, as illustrated in figure 5.
  - **Fault Tolerance Method (FTM):** This variable is defined using two fuzzy sets of Checkpointing and Replication, as presented in figure 6.

The membership function μ of the MR attribute is defined as follows:

$$\mu_{low\_MR}(x) = \begin{cases} 0, & if\ x > 1 \\ (1-x), & if\ \ 0 \le x \le 1 \end{cases} \tag{1}$$

Figure 3. The Machine Risk (MR) fuzzy set

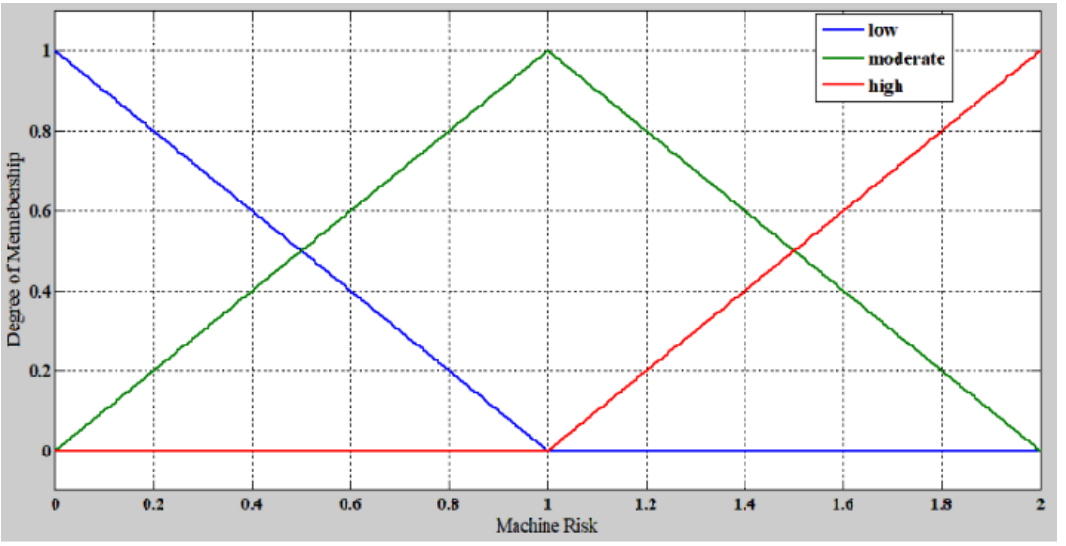


Figure 4. The SA fuzzy set

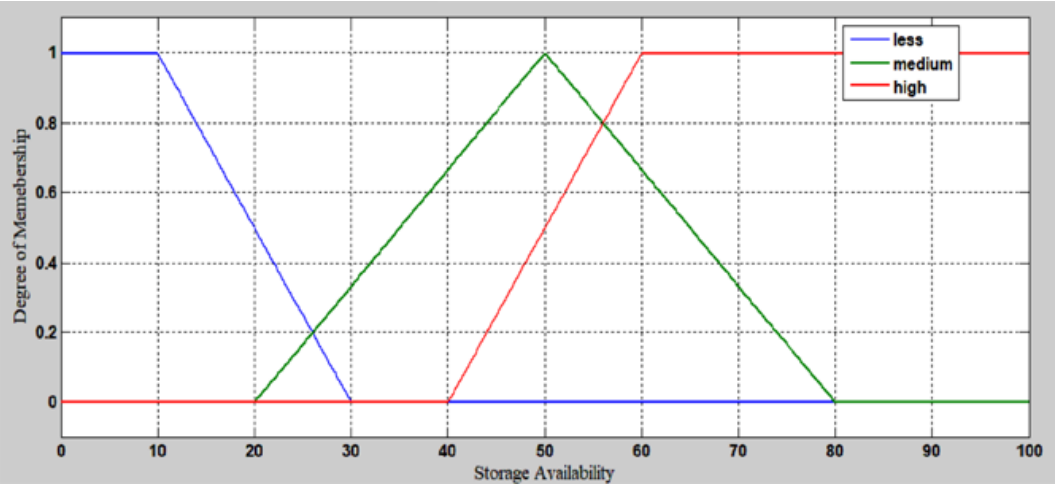
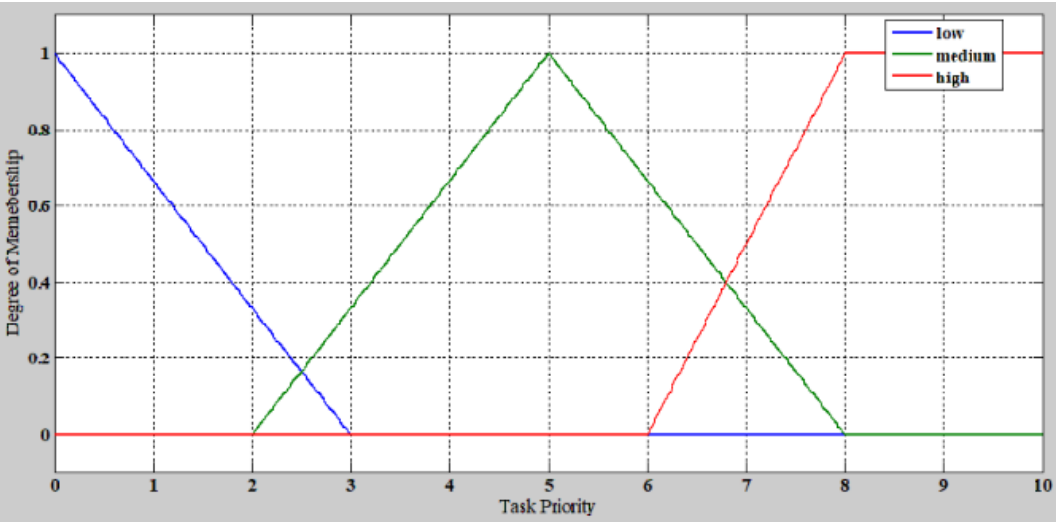


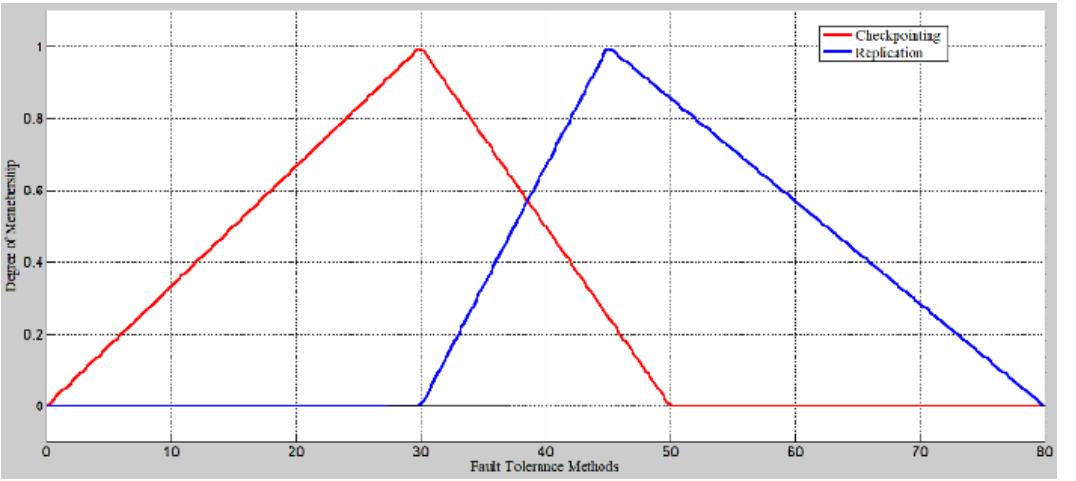
Figure 5. The TP fuzzy set



$$\mu_{\mathrm{mod}\_MR}(x)=\begin{cases}0, & if\ x<0\\ (x), & if\ \ 0\le x\le 1\\ (2-x), & if\ \ 1\le x\le 2\\ 0, & if\ x>0\end{cases} \tag{2}$$

$$\mu_{high\_MR}(x)=\begin{cases}0, & if\ \ x<1\\ (x-1), & if\ \ \ 1\le x\le 2\end{cases} \tag{3}$$

Figure 6. The FTM fuzzy set



The horizontal and the vertical axis denote the input value and degree of membership of the attribute SA respectively. The membership function μ of the SA attribute is defined as follows:

$$\mu_{less\_SA}(x) = \begin{cases} 1, & if \ x < 10 \\ \dfrac{(30-x)}{20}, & if \ 10 \le x \le 30 \\ 0, & if \ x > 30 \end{cases} \tag{4}$$

$$\mu_{med\_SA}(x) = \begin{cases} 0, & if \ x < 20 \\ \dfrac{(x-20)}{30}, & if \ 20 \le x \le 50 \\ \dfrac{(80-x)}{30}, & if \ 50 \le x \le 80 \\ 0, & if \ x > 80 \end{cases} \tag{5}$$

$$\mu_{high\_SA}(x) = \begin{cases} 0, & if \ x < 40 \\ \dfrac{(x-40)}{20}, & if \ 40 \le x \le 60 \\ 1, & if \ x > 60 \end{cases} \tag{6}$$

The horizontal and the vertical axis denote the input value and degree of membership of the attribute TP respectively. The membership function, μ of the TP attribute, is defined as follows:

$$\mu_{low\_TP}(x) = \begin{cases} 0, & if\ x > 3 \\ \dfrac{(3-x)}{3}, & if\ 0 \le x \le 3 \end{cases} \tag{7}$$

$$\mu_{med\_TP}(x) = \begin{cases} 0, & if\ x < 2 \\ \dfrac{(x-2)}{3}, & if\ 2 \le x \le 5 \\ \dfrac{(8-x)}{3}, & if\ 5 \le x \le 8 \\ 0, & if\ x > 8 \end{cases} \tag{8}$$

$$\mu_{high\_TP}(x) = \begin{cases} 0, & if\ x < 6 \\ \dfrac{(x-6)}{2}, & if\ 6 \le x \le 8 \\ 1, & if\ x > 8 \end{cases} \tag{9}$$

The horizontal and the vertical axis denote the input value and degree of membership of the attribute FTM respectively. The membership function, μ of the FTM attribute, is defined as follows:

$$\mu_{CP}(x) = \begin{cases} 0, & if\ x < 0 \\ \dfrac{x}{30}, & if\ 0 \le x \le 30 \\ \dfrac{(50-x)}{20}, & if\ 30 \le x \le 50 \\ 0, & if\ x > 50 \end{cases} \tag{10}$$

$$\mu_{\mathrm{Re}\,p}(x) = \begin{cases} 0, & if\ x < 30 \\ \dfrac{(x-30)}{15}, & if\ 30 \le x \le 45 \\ \dfrac{(80-x)}{35}, & if\ 45 \le x \le 80 \\ 0, & if\ x > 80 \end{cases} \tag{11}$$

3.3 Fuzzy Rules

An inclusive set of 27 fuzzy rules is framed for determining the value of the *FTM* variable (output variable) on the basis of the values of 3 input variables. To form the fuzzy rules, we followed certain conditions:

Case 1: If the storage availability on a physical machine is high and the priority of the task submitted by the user is high then irrespective of the value of machine risk, the replication fault-tolerance method is selected to handle the failure.

Case 2: If the storage availability on a physical machine is high or medium and the priority of the task submitted by the user is low then there can be two sub-conditions:

- If physical machine risk is high, then the replication fault-tolerance method is selected to deal with the failure.
- If physical machine risk is low, then the checkpointing fault-tolerance method is selected to handle the failure.

Case 3: In the case of low storage availability on a physical machine, irrespective of the value of task priority and machine risk, the checkpointing fault-tolerance method is selected to deal with the failure.

The 27 fuzzy inference rules consider all the possible combinations of the above-defined conditions (Case 1 to Case 3) for the input variables. Considering Case 1, Case 2, and Case 3, three rules are designed for each. Hence, the total number of framed rules is 3*3*3 = 27. The 27 rules for the proposed fuzzy system are illustrated in Table 3.

Table 3. Proposed rules for FTMSFIS

Rule No.	Machine Risk	Storage Availability	Task Priority	Fault Tolerance Method
1	Low	Less	Low	Checkpointing
2	Low	Less	Medium	Checkpointing
3	Low	Less	High	Checkpointing
4	Low	Medium	Low	Checkpointing
5	Low	Medium	Medium	Checkpointing
6	Low	Medium	High	Replication
7	Low	High	Low	Checkpointing
8	Low	High	Medium	Checkpointing
9	Low	High	High	Replication
10	Moderate	Less	Low	Checkpointing
11	Moderate	Less	Medium	Checkpointing
12	Moderate	Less	High	Checkpointing
13	Moderate	Medium	Low	Checkpointing
14	Moderate	Medium	Medium	Checkpointing
15	Moderate	Medium	High	Replication
16	Moderate	High	Low	Checkpointing
17	Moderate	High	Medium	Checkpointing
18	Moderate	High	High	Replication
19	High	Less	Low	Checkpointing
20	High	Less	Medium	Checkpointing
21	High	Less	High	Checkpointing
22	High	Medium	Low	Replication
23	High	Medium	Medium	Replication
24	High	Medium	High	Replication
25	High	High	Low	Replication
26	High	High	Medium	Replication
27	High	High	High	Replication

Subsequently, on evaluation of the rules in Table 3 according to the given conditions, we found that some rules could be combined to get a minimized set of rules in order to further optimize the proposed fuzzy system. For instance, rules 1-3 indicate that if machine risk is low and storage availability is less, then irrespective of Task Priority, the FT method to be used is Checkpointing. Hence, Rules 1-3 in Table 3 are merged to form the new rule 1 in Table 4. Similarly, rules 10 - 12 in Table 3 are combined to form rule 8 in Table 4, and 19 - 21 are merged to form rule 15 in Table 4. Following the same approach, rules from 22 to 24, and 25 to 27 are combined to form rules 16 and 17 respectively in table 4. Thus, after merging a few rules, the total number of fuzzy rules for the proposed fuzzy system is 17. The reduced set of fuzzy rules is illustrated in Table 4.

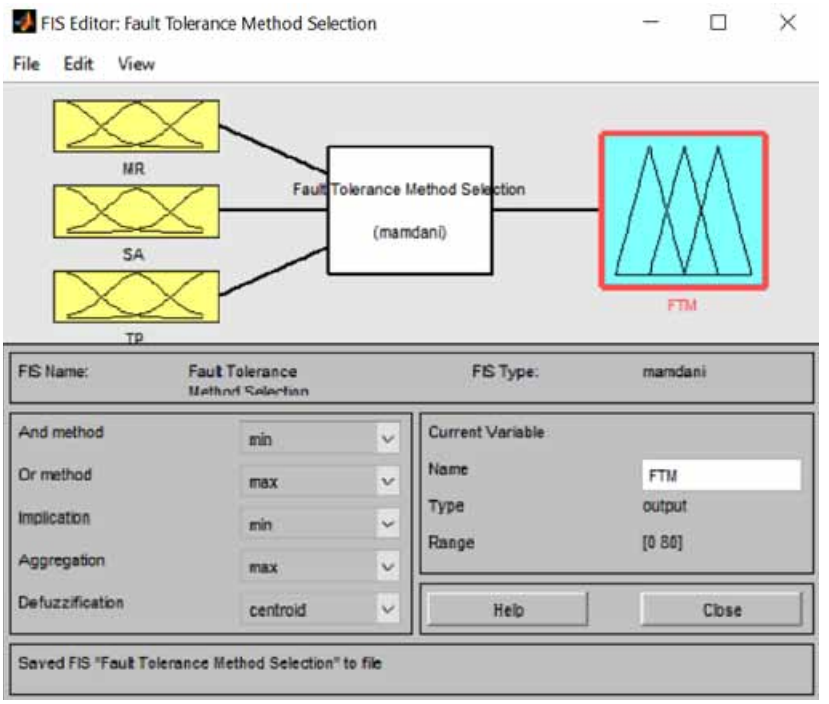
Further, Figure 7 illustrates the design of the proposed fuzzy system in the MATLAB environment. We have designed the Mamdani FIS with three inputs, one output, and 17 rules. The fuzzifier transformed input attributes in a linguistic term set with the help of membership functions. The fuzzifier's result is forwarded to the inference engine, which uses the database's well-defined rules to provide the best possible output.

To represent the final system output, the produced fuzzy result is sent to the defuzzifier, which translates the outcome into a crisp value. Defuzzification is done using the centroid method. The FTM of a system is evaluated in either checkpointing or replication states based on the defuzzifier result. The FTMSFIS inference engine database rules, which were executed in MATLAB, are shown in Figure 8a. Figure 8b depicts a Surface Viewer interpretation of the predicted fuzzy system.

Table 4. Minimized rules for FTMSFIS

Rule No.	Machine Risk	Storage Availability	Task Priority	Fault Tolerance Method
1	Low	Less	Low/Medium/High	Checkpointing
2	Low	Medium	Low	Checkpointing
3	Low	Medium	Medium	Checkpointing
4	Low	Medium	High	Replication
5	Low	High	Low	Checkpointing
6	Low	High	Medium	Checkpointing
7	Low	High	High	Replication
8	Moderate	Less	Low/Medium/High	Checkpointing
9	Moderate	Medium	Low	Checkpointing
10	Moderate	Medium	Medium	Checkpointing
11	Moderate	Medium	High	Replication
12	Moderate	High	Low	Checkpointing
13	Moderate	High	Medium	Checkpointing
14	Moderate	High	High	Replication
15	High	Less	Low/Medium/High	Checkpointing
16	High	Medium	Low/Medium/High	Replication
17	High	High	Low/Medium/High	Replication

Figure 7. FTMSFIS implementation in MATLAB



4. SIMULATION RESULTS

In this segment, we measure the effectiveness of the projected scheme with simulation-based experiments. The suggested scheme is compared with the replication and checkpointing method in terms of the total amount of successful recoveries as well as the maximum storage consumption. The simulations are run on a PC with an Intel® Core (TM) CPU i5-7200 processor running at 2.7 GHz and 8 GB of RAM. We assume that each PM can have 0 to 4 VMs for executing the submitted tasks.

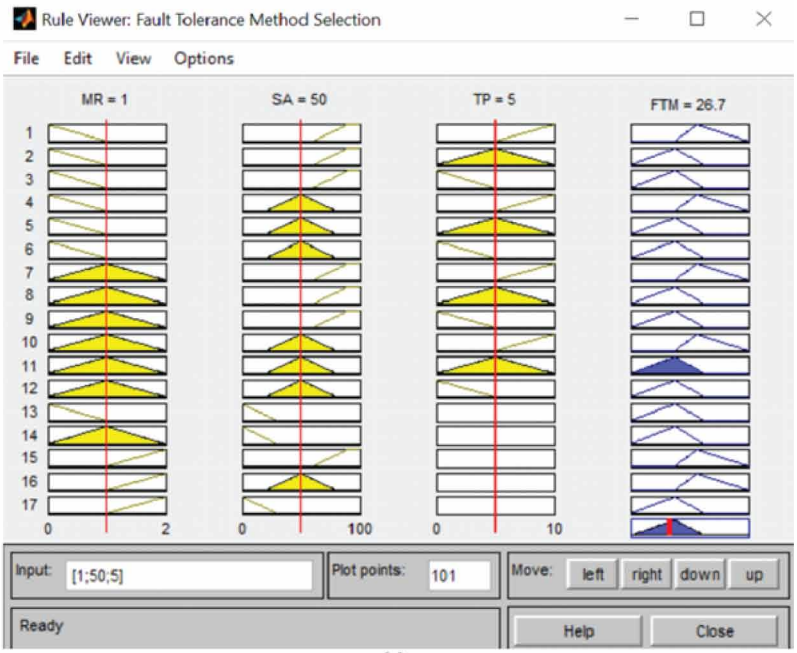
4.1 Metrics

1. **Recoverability:** This metric is used to calculate the percentage of successful VM recoveries and is defined in equation 12:

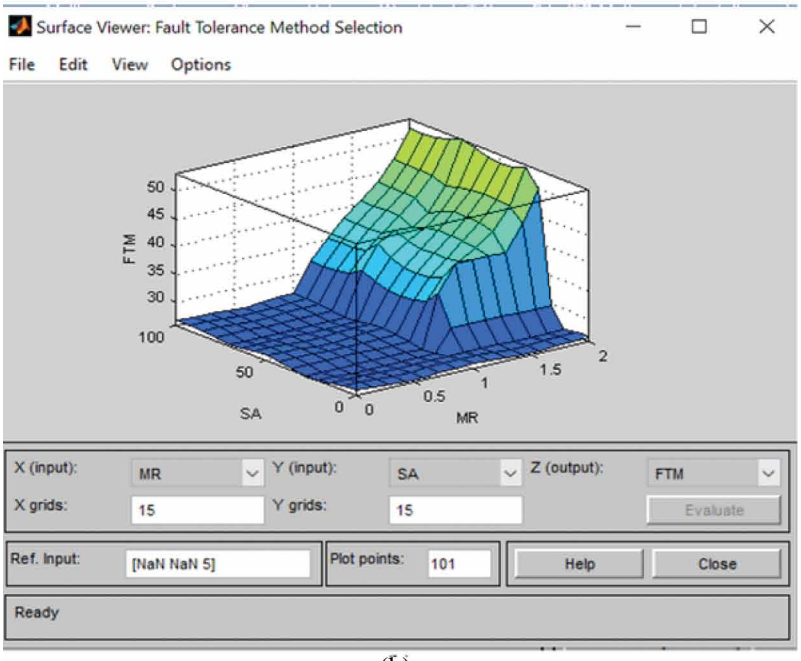
$$Recoverability = \frac{Total\ \ Number\ \ of\ \ Successful\ \ Recovery}{Total\ \ Number\ \ of\ \ Failure} \times 100 \tag{12}$$

2. **Maximum Storage Consumption (MSC):** This metric is used to calculate the maximum amount of storage consumed on PMs while executing checkpointing and replication methods as well as the proposed method. The MSC is defined in equation 13:

Figure 8. (a) an illustration of inference engine rules intended in MATLAB; (b) an illustration of Surface Viewer of FTMSFIS rules



(a)



(b)

$$Replication\ \ Method = \sum_{i}^{n} pm_i * v * s$$
$$Checkpointing\ \ Method = \sum_{i}^{n} pm_i * v * cpi \quad (13)$$
$$Proposed\ \ Method = \sum_{i}^{h} pm_i * v * s + \sum_{j}^{l} pm_j * v * cpi$$

where:

- n represents the total number of PMs.
- v signifies the number of VMs on each PM.
- s denotes the size of each virtual machine.
- cpi denotes the checkpoint image size of each virtual machine.
- h denotes the total number of high-risk machines.
- l denotes the total number of low-risk machines.

5. RESULTS

5.1 Case 1: High Storage Availability and High Task Priority

5.1.1 Measuring Successful Recoverability

The objective of this experiment is to measure the number of VMs that recovered successfully in a given time period using the proposed method. In this scenario, the replication method is selected by the proposed approach. Hence, we compare the outcomes with the number of VMs successfully recovered achieved using the checkpointing method. From the outcomes in Fig. 9, it can be seen that the proposed strategy achieves higher recoverability than the checkpointing scheme by about 4% on average.

The next set of experiments is done to evaluate the number of recoveries under varying failure rates (FR) of VM. FR means how often a PM or VM breaks down in a quantified period of time.

Figure 9. Successful recovery of virtual machines

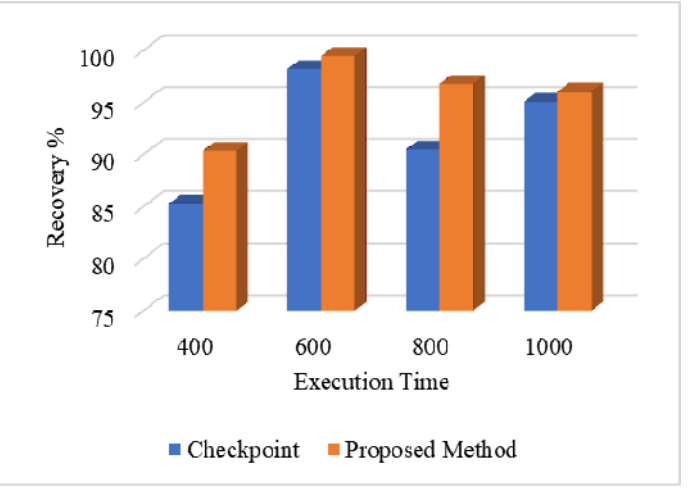


Fig. 10 demonstrates the recoverability of the system for FR as 0.01, 0.05, 0.1, and 0.5 for varying execution times.

In both the above cases, the proposed approach is providing a higher recoverability than the checkpointing scheme.

5.2 Case 2: High Storage Availability and Low Task Priority

5.2.1 Measuring Recoverability

The results of the suggested methodology are compared to the number of successful VM recoveries accomplished using replication and the checkpointing strategy in this experiment. From the outcomes in Fig. 11, it can be observed that the proposed method accomplishes higher recoverability, of nearly 56% and 48% on average than the replication and checkpointing schemes respectively.

The subsequent set of experiments evaluates the number of recoveries under different failure rates (FR) of VM. Fig. 12 demonstrates the recoverability of the system for FR as 0.01, 0.05, 0.1, and 0.5 for varying execution times.

In both the above cases, the proposed approach is providing a higher recoverability than the replication and checkpointing scheme. The same can be attributed to the adaptability of the proposed scheme depending on the task priority and the given cloud environment.

5.2.2 Measuring Maximum Storage Consumption

In this experiment, the outcomes of the proposed method are compared with the replication and the checkpointing method in terms of maximum storage consumption. From the outcomes in Fig. 13, it can be observed that the proposed method consumes less storage space than the replication method but is higher than the checkpointing method.

Figure 10. Successful recovery vs failure rate

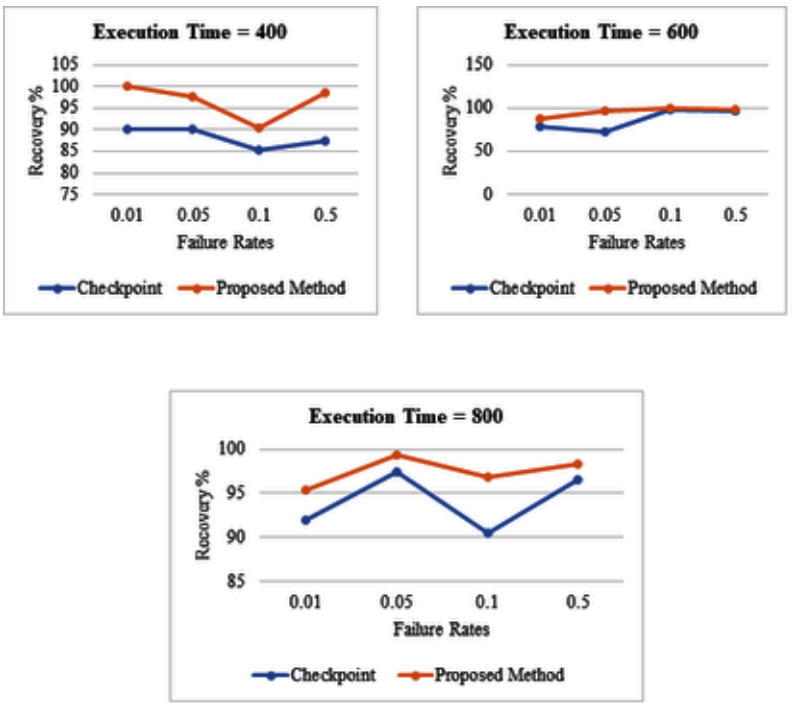
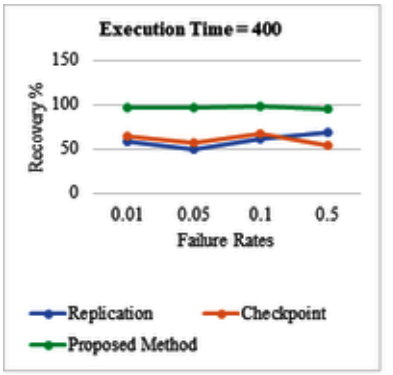
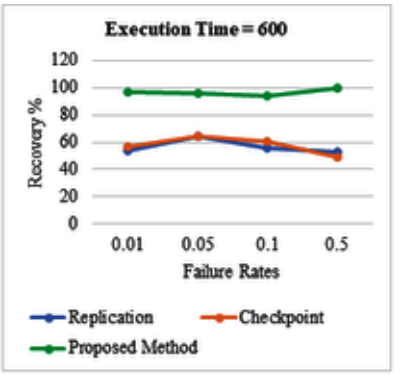
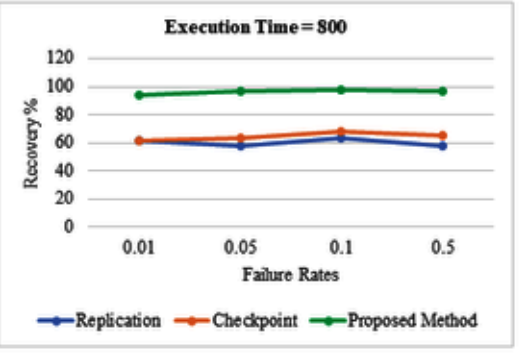


Figure 11. Successful recovery of virtual machines

Figure 12. Successful recovery vs failure rate

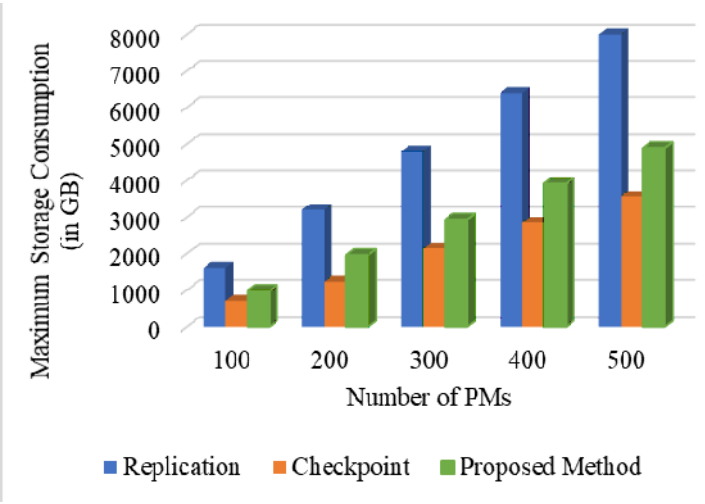






In the above-defined instances, the proposed approach is providing a higher recoverability than the replication and checkpointing scheme. However, the proposed approach consumes slightly higher storage space than checkpointing but less than the replication scheme. The results are according to expected outcomes since the proposed approach flexibly selects either checkpointing, which takes

Figure 13. Maximum storage consumption on PMs (in GB)



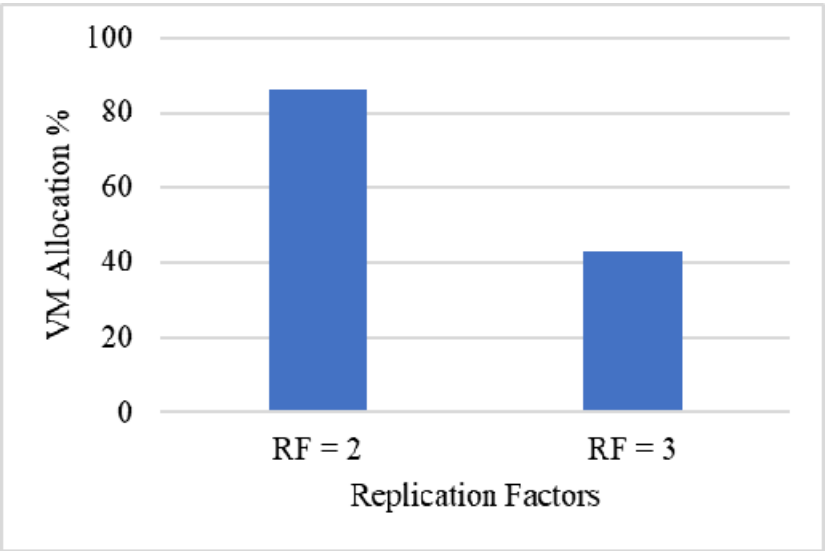
up less space, or replication, which consumes higher space. Thus, the proposed approach improves the reliability of the cloud systems while taking up less storage than replication.

5.3 Case 3: Low Storage Availability and High or Low Task Priority

5.3.1 Measuring Replication Factor

This experiment is done to measure the percentage of VM allocation (successful placement of VMs) on the varying replication factor. From the outcomes in fig. 14, it can be observed that the number of VM allocations using replication factor 2 is better than the replication factor 3. However, in case of failure, the system with replication factor 3 provides better reliability.

Figure 14. VM allocation vs Replication Factor



5.3.2 Measuring Recoverability

The aim of this experiment is to determine the number of VMs that successfully recovered using the suggested method in a particular time frame. The results are compared to the number of VMs that have been successfully recovered using the replication approach. The results in Fig. 15 show that the proposed strategy achieves higher recoverability than the replication scheme, of nearly 6% on average.

The next set of experiments is done to evaluate the number of recoveries under varying failure rates (FR) of VM. Fig. 16 demonstrates the recoverability of the system for FR as 0.01, 0.05, 0.1, and 0.5 for varying execution times.

In both the above cases, the proposed approach is providing a higher recoverability than the replication scheme.

5.3.3 Measuring Maximum Storage Consumption

This experiment is done on a distinct set of PMs to evaluate the amount of storage consumed using the proposed method. The outcomes of the proposed method are compared with the replication scheme in terms of maximum storage consumption. From the result in Fig. 17, it can be observed that the proposed method consumes less storage space than the replication method.

In the above-defined instances, the proposed approach is providing a higher recoverability than the replication scheme. However, the proposed approach also consumes less storage space than the replication scheme.

6. CONCLUSION

The dynamic nature of the cloud computing environment makes it vulnerable to varied types of faults. This results in the significance of FT methods in the implementation of reliable and robust cloud-based systems. The paper presents a classification and fuzzy logic-based adaptable approach to FT in the cloud. It considers parameters related to a user's preferences, a physical machine's risk of failure, and a cloud system's capabilities to determine the appropriate fault tolerance method to be used for a task submitted by the user. Here, two types of FT methods are considered: checkpointing and replication. The performance of the proposed method is evaluated through simulation experiments. It has been seen that the suggested adaptable approach outperforms the methods of checkpointing

Figure 15. Successful recovery of virtual machines

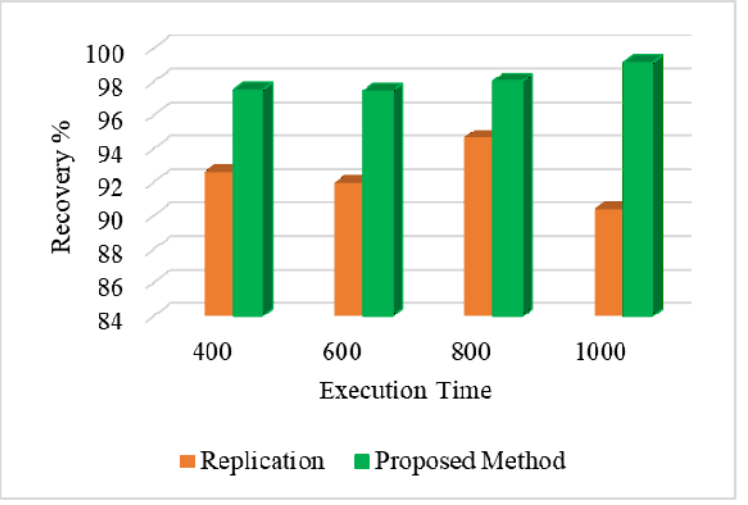


Figure 16. Successful recovery vs failure rate

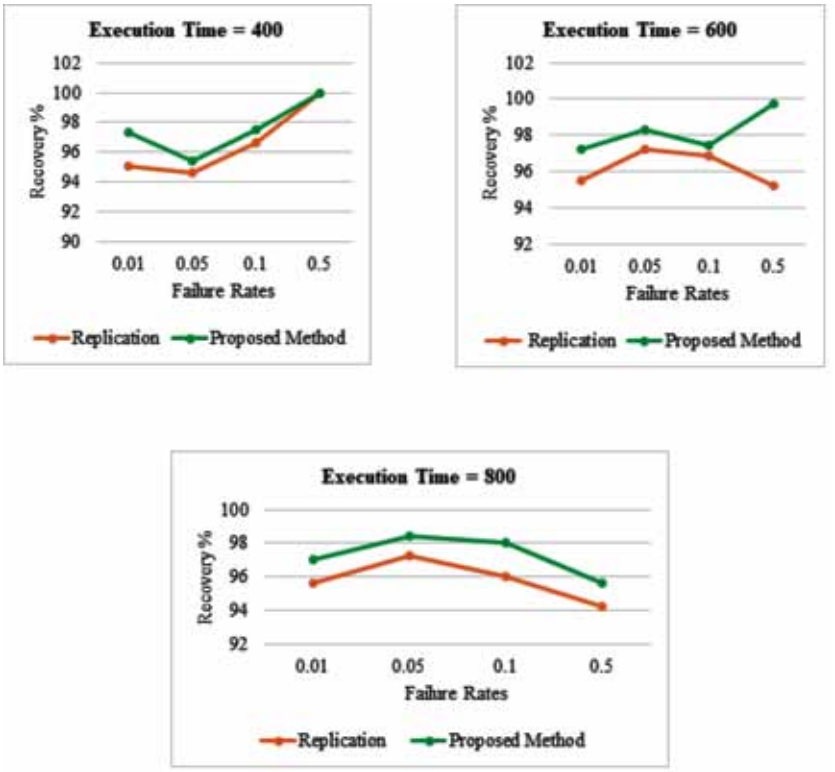
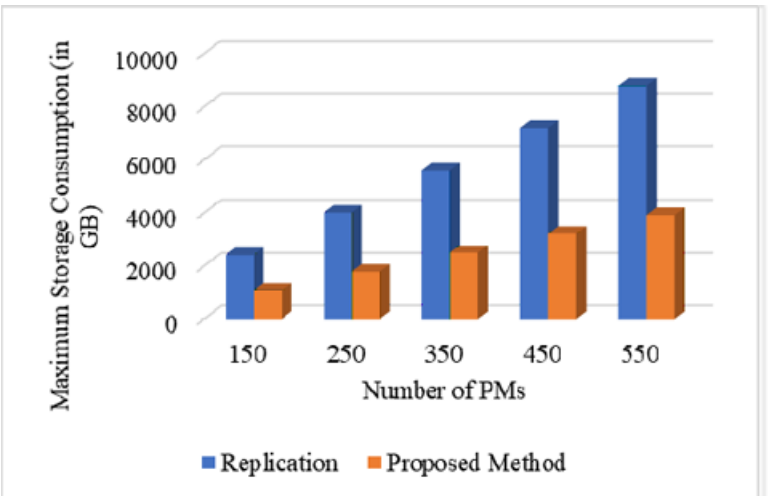


Figure 17. Maximum storage consumption on PMs (in GB)



and replication when used alone. The implementation of the adaptable method results in a higher number of successful recoveries than the other methods; thus, leading to efficient use of resources, user satisfaction, and economical profits for users as well as cloud providers.

DATA AVAILABILITY STATEMENT

The dataset analyzed during the current study is NERSC data available CFDR repository at https://www.usenix.org/cfdr-data.

REFERENCES

AbdElfattah, , EElkawkagy, , MEl-Sisi, , A. (2017), A reactive fault tolerance approach for cloud computing. *13th International Computer Engineering Conference (ICENCO)*. IEEE.

Agarwal, K. K., & Kotakula, H. (2022). Replication Based Fault Tolerance Approach for Cloud. In *International Conference on Distributed Computing and Internet Technology*, (pp. 163-169). Springer. doi:10.1007/978-3-030-94876-4_11

Amin, Z., Singh, H., & Sethi, N. (2015). Review on fault tolerance techniques in cloud computing. *International Journal of Computers and Applications*, *116*(18).

Amoon, M., El-Bahnasawy, N., Sadi, S., & Wagdi, M. (2019). On the design of reactive approach with flexible checkpoint interval to tolerate faults in cloud computing systems. *Journal of Ambient Intelligence and Humanized Computing*, *10*(11), 4567–4577. doi:10.1007/s12652-018-1139-y

Amoon, M., El-Bahnasawy, N., Sadi, S., & Wagdi, M. (2019). On the design of reactive approach with flexible checkpoint interval to tolerate faults in cloud computing systems. *Journal of Ambient Intelligence and Humanized Computing*, *10*(11), 4567–4577. doi:10.1007/s12652-018-1139-y

Attallah, S. M., Fayek, M. B., Nassar, S. M., & Hemayed, E. E. (2021). Proactive load balancing fault tolerance algorithm in cloud computing. *Concurrency and Computation*, *33*(10), e6172. doi:10.1002/cpe.6172

Bui, D. M., & Lee, S. (2018). Early fault detection in IaaS cloud computing based on fuzzy logic and prediction technique. *The Journal of Supercomputing*, *74*(11), 5730–5745. doi:10.1007/s11227-017-2053-3

Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., & Brandic, I. (2009). Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, *25*(6), 599–616. doi:10.1016/j.future.2008.12.001

Cheraghlou, M. N., Khademzadeh, A., & Haghparast, M. (2019). New fuzzy-based fault tolerance evaluation framework for cloud computing. *Journal of Network and Systems Management*, *27*(4), 930–948. doi:10.1007/s10922-019-09491-2

Chinnathambi, S., Santhanam, A., Rajarathinam, J., & Senthilkumar, M. (2019). Scheduling and checkpointing optimization algorithm for Byzantine fault tolerance in cloud clusters. *Cluster Computing*, *22*(6), 14637–14650. doi:10.1007/s10586-018-2375-9

Ejimogu, O. H., & Başaran, S. (2017). A systematic mapping study on soft computing techniques to the cloud environment. *Procedia Computer Science*, *120*, 31–38. doi:10.1016/j.procs.2017.11.207

Gupta, B. B., Agrawal, D. P., Yamaguchi, S., & Sheng, M. (2020). *Soft computing techniques for big data and cloud computing.* Springer.

Hasan, M., & Goraya, M. S. (2018). Fault tolerance in cloud computing environment: A systematic survey. *Computers in Industry*, *99*, 156–172. doi:10.1016/j.compind.2018.03.027

Java. (n.d.). *Java Point.* Java. https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning

Kumari, P., & Kaur, P. (2018). A survey of fault tolerance in cloud computing. *Journal of King Saud University-Computer and Information Sciences*.

Kumari, P., & Kaur, P. (2020). Topology-aware virtual machine replication for fault tolerance in cloud computing systems. *Multiagent and Grid Systems*, *16*(2), 193–206. doi:10.3233/MGS-200328

Liu, J., Wang, S., Zhou, A., Kumar, S. A., Yang, F., & Buyya, R. (2016). Using a proactive fault-tolerance approach to enhance cloud service reliability. *IEEE Transactions on Cloud Computing*, *6*(4), 1191–1202. doi:10.1109/TCC.2016.2567392

Menychtas, A., & Konstanteli, K. G. (2012). Fault detection and recovery mechanisms and techniques for service-oriented infrastructures. In *Achieving real-time in distributed computing: from grids to clouds* (pp. 259–274). IGI Global. doi:10.4018/978-1-60960-827-9.ch014

Mohammed, B., Modu, B., Maiyama, K. M., Ugail, H., Awan, I., & Kiran, M. (2018). Failure analysis modeling in an infrastructure as a service (IaaS) environment. *Electronic Notes in Theoretical Computer Science*, *340*, 41–54. doi:10.1016/j.entcs.2018.09.004

Monil, M. A. H., & Rahman, R. M. (2016). VM consolidation approach is based on heuristics, fuzzy logic, and migration control. *Journal of Cloud Computing*, *5*(1), 8. doi:10.1186/s13677-016-0059-7

Patra, P. K., Singh, H., Singh, R., Das, S., Dey, N., & Victoria, A. D. C. (2016). Replication and resubmission based adaptive decision for fault tolerance in real-time cloud computing: A new approach. *International Journal of Service Science, Management, Engineering, and Technology*, *7*(2), 46–60. doi:10.4018/IJSSMET.2016040104

Prakash, S., & Vyas, V. (2022). Analysis of Fault Tolerance Techniques in Virtual Machine Environment. In ICT Analysis and Applications, (pp. 121-131). Springer. doi:10.1007/978-981-16-5655-2_12

Rawat, A., Sushil, R., Agarwal, A., Sikander, A., & Bhadoria, R. S. (2021). A new adaptive fault tolerant framework in the cloud. *Journal of the Institution of Electronics and Telecommunication Engineers*, 1–3. doi: 10.1080/03772063.2021.1907231

Ray, B., Saha, A., Khatua, S., & Roy, S. (2020). *Proactive Fault-Tolerance Technique to Enhance Reliability of Cloud Service in Cloud Federation Environment*. IEEE Transactions on Cloud Computing. IEEE.

Rezaeipanah, A., Mojarad, M., & Fakhari, A. (2022). Providing a new approach to increase fault tolerance in cloud computing using fuzzy logic. *International Journal of Computers and Applications*, *44*(2), 1–9. doi:10.1080/1206212X.2019.1709288

Rong, H., Wang, H. M., Liu, J., & Xian, M. (2016). Privacy-preserving k-nearest neighbor computation in multiple cloud environments. *IEEE Access: Practical Innovations, Open Solutions*, *4*, 9589–9603. doi:10.1109/ACCESS.2016.2633544

Saikia, L. P., & Devi, Y. L. (2014). Fault tolerance techniques and algorithms in cloud computing. *International Journal of Computer Science & Communication Networks, 4*(1), 01-08.

Sathiyamoorthi, V., Keerthika, P., Suresh, P., Zhang, Z. J., Rao, A. P., & Logeswaran, K. (2021). Adaptive fault tolerant resource allocation scheme for cloud computing environments. *Journal of Organizational and End User Computing*, *33*(5), 135–152. doi:10.4018/JOEUC.20210901.oa7

Sharma, S. (2017). Enhance data security in cloud computing using machine learning and hybrid cryptography techniques. *International journal of advanced research in computer science, 8*(9).

Srivastava, N. P., & Srivastava, R. K. (2014). Soft Computing Approaches To Fault-Tolerant Systems. *International Journal of Advanced Networking and Applications*, *5*(6), 2096.

Sun, J., Du, W., & Shi, N. (2018). *A Survey of kNN Algorithm*. Information Engineering and Applied Computing. doi:10.18063/ieac.v1i1.770

Tran, D., Tran, N., Nguyen, G., & Nguyen, B. M. (2017). A proactive cloud scaling model based on fuzzy time series and SLA awareness. *Procedia Computer Science*, *108*, 365–374. doi:10.1016/j.procs.2017.05.121

Wu, Y., Peng, G., Wang, H., & Zhang, H. (2019). A two-stage fault tolerance method for large-scale manufacturing network. *IEEE Access: Practical Innovations, Open Solutions*, *7*, 81574–81592. doi:10.1109/ACCESS.2019.2923660

Zhao, J., Xiang, Y., Lan, T., Huang, H. H., & Subramaniam, S. (2016). Elastic reliability optimization through peer-to-peer checkpointing in cloud computing. *IEEE Transactions on Parallel and Distributed Systems*, *28*(2), 491–502. doi:10.1109/TPDS.2016.2571281

Zhou, A., Sun, Q., & Li, J. (2017). Enhancing reliability via checkpointing in cloud computing systems. *China Communications*, *14*(7), 1–10. doi:10.1109/CC.2017.8010962

Zhou, A., Wang, S., Cheng, B., Zheng, Z., Yang, F., Chang, R. N., & Buyya, R. (2016). Cloud service reliability enhancement via virtual machine placement optimization. *IEEE Transactions on Services Computing*, *10*(6), 902–913. doi:10.1109/TSC.2016.2519898

Priti Kumari received the MCA degree from the Banasthali University of Rajasthan, India in 2014. Got the M.Tech degree in Computer Science & Engineering from the Banasthali University of Rajasthan, India in 2016. She is currently a Ph.D. student in Jaypee Institute of Information Technology, Noida, India, Computer Science branch. Her research interests include Cloud Computing, Fault Tolerance, and Distributed Computing.