International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

An Optimal and Distributed Checkpointing and Replication based Fault-tolerant Strategy for Reliable Cloud Computing

M. Damodhar[1], Dr. Ch. D. V. Subba Rao[2]

Submitted: 11/05/2024 **Revised**: 25/06/2024 **Accepted**: 03/07/2024

Abstract: In the area of information technology the emerging technology Cloud computing plays a major role. Cloud computing virtualization and its dependency on Internet leads to a variety of failures to happen and hence there is a need for reliability and availability becomes a major issue. To ensure proper reliability and availability of the cloud, an efficient fault tolerance strategy needs to be developed and implemented. Majority of the earlier fault tolerant approaches focused on using only one method for tolerating faults. This paper presents an efficient and effective fault-tolerant strategy to deal with the problem of fault tolerance in the environment of cloud computing. This fault-tolerant strategy depends on optimal and distributed checkpointing and replication scheme for obtaining a reliable cloud platform for carrying out customer requests. Further it determines the best fault tolerance strategy for every selected virtual machine (VM). Simulation experiments are carried out to evaluate the performance of the fault-tolerant strategy. The experiment results show that the proposed fault-tolerant strategy enhances the cloud performance in terms of overheads, throughput, availability and maintenance cost.

Keywords: *Cloud computing, fault-tolerant, optimal checkpointing, replication, virtual machines (VMs).*

1. Introduction

The current market of Information Technology (IT) has witnessed a considerable change due to the presence of cloud computing, which has become an integral part of most of the businesses [1]. Today, most of the businesses, from single to large enterprises, migrated to cloud computing in order to obtain a high level of productivity by entrusting their IT issues to an expert one. Cloud computing provides comprehensive IT services and solutions for both companies and individual users [2]. They can lease components of the cloud without expending time and money in constructing or buying these components. In cloud servers, computing is introduced as an abstract service on the Internet by hiding its implementation details [3].

The deployment models of cloud computing systems are public, private or hybrid. In public, services are pro- vided through the Internet in forms of cloud practical application. The main categories of these applications include Infrastructure-as-a-Service (IaaS), Software-as-a- Service (SaaS) and Platform-as-a-Service (PaaS). Most of IT businesses cannot invest in certain services such as supercomputer-class services. In IaaS, the cloud provides computing, storage and networking resources with any required configuration and capacity as paid services to the customers. Examples of practical applications of IaaS can include Amazon EC2 and Google Compute Engine. In most IT organizations, there are no enough experts to develop and run the required software applications. In SaaS, the cloud provides customers with access to professionally implemented software applications and thus they save the customers' money. Salesforce.com and Google Apps are examples of practical applications of SaaS. In PaaS, customers can run their custom applications on the general purpose software and hardware with the most recent configurations. Practical applications of PaaS include Google App Engine and Microsoft Azure [4].

Private clouds are implemented and maintained by various enterprises to provide internal services and further they have additional flexibility compared to public clouds however they are more expensive. In hybrid clouds, some portions of computing can be done in a public cloud while other portions can be done internally through the private one [4].

In spite of various services offered by cloud systems in cloud environment, they are not always perfectly reliable further they could suffer from outages of their services due to failures and sometimes disasters [5]. An outage is known to be a situation where the request of a customer is not completed in its required deadline. With the raise of cloud usage, the number of required cloud services increases and further there could be increased outage probability. The foremost reasons of these outages comprises software failures due to incorrect upgrade, extreme work load, hacking, etc. and hardware failures such as resource unavailability, network failure, power down times, etc.

[1] *Research Scholar, Dept. of Computer Science and Engg., SVUCE, Sri Venkateswara University, Tirupati, Andhra Pradesh - 517502, India*
Email: mdr.damodhar@gmail.com
[2] *Professor & Head, Dept. of Computer Science and Engg., SVUCE, Sri Venkateswara University, Tirupati, Andhra Pradesh - 517502, India*
Email: subbarao_chdv@hotmail.com

In general, outages are popular in public clouds in which an enormous number of services are provided to customers with required levels of service quality. In the last decade, many outages have occurred in most famous public cloud environments. In 2021, AWS has experienced a severe outage that disrupts services for several hours due to the issue of US-East-1 region. In 2022, Google Search, Drive, Maps and YouTube are down, returning HTTP 500 and HTTP 502 errors. After their services came back online, Google apologized and stated that there is a software update issue [6]. Further, other cloud service providers (CSP's) such as Apple iCloud and Microsoft Azure are among the technology vendors to experience major cloud outages. In 2023, the foremost cloud vendors like Amazon Web Services (AWS), Microsoft and Google experienced major service disruptions. For example, AWS us-east-1, an important critical region for Amazon Web Services, faced an outage with down-time of 24-hours that might cost up to $3.4 billion in their direct revenue.

Cloud outages or failures have a great impact on both the cloud vendors and the customers. For vendors, there will be no profit due to the cloud resources that are used to come back from the effects of outages happened. K. Bilal et al [7] have stated that each downtime hour in a data center costs around U$ 50,000. For customers, their requirements, such as deadline time, may not be achieved. So, there is a tremendous requirement for an always available and reliable cloud that consists of a dynamic method for fault tolerance. The method should transparently remove or reduce to some extent the effects of failures on both customers and profit needs.

Fault tolerance methods can be reactive or proactive. The foremost goal of the reactive methods is to reduce the effect of fault occurrence while the goal of the proactive methods is to avoid fault occurrence. Reactive methods mainly include replication and checkpointing. Most cloud computing systems depend on reactive methods, especially replication [9].

The replication method assumes that the likelihood of a single VM failure is extremely higher than the occurrence of simultaneous failures of multiple VMs. It permits multiple virtual machines to start concurrently by executing redundant copies of a single request in order to prevent re-computation of that task from the beginning in case of failure. Hence, the service can be efficiently offered to customers while providing their QoS requirements even in the case of failures in cloud servers. Whereas, in checkpointing, the cloud eventually saves the execution state of current request and its executing VM to a stable storage server in order to reduce the recovery time during the situation of failure. In case of failure, instead of starting the request from the beginning, it will be started from the point where the last recent checkpoint was saved [10], [11].

The key contribution of this paper is to provide an adaptive fault-tolerant strategy to handle the proactive and reactive faults in cloud environments. To cope with the proactive faults, the implemented fault-tolerant strategy needs customer requirements and the information about virtual machines during task scheduling. Also, this fault-tolerant strategy employs both optimal checkpointing and replication methods and it dynamically chooses the appropriate method based on the conditions of the cloud at that instant.

The rest of the paper is arranged as follows: Section 2 presents a brief illustration of the related work. Section 3 describes the problem. Section 4 provides the details of the proposed fault-tolerant strategy. In Section 5, the results obtained from simulation experiments are presented and finally the paper concludes in Section 6.

2. Literature Survey

The dynamic nature of the cloud raises the chances probability of failures. Therefore, to reduce or completely avoid the effects of such failures, the cloud should be applied with fault-tolerant strategy, that can be either reactive or proactive. Reactive fault tolerance methods are useful to minimize or eradicate the effect of failures on monetary and time costs. Generally, Checkpointing and Replication are the two methods mostly used for reactive fault tolerance.

The replication method is based on that the likelihood of failures will be reduced when multiple virtual machines are used to carry out the same customer's request. Recompilation of a request is avoided by performing multiple replicas of the request on different virtual machines at the same time. In case of virtual machine failure the cloud can still execute the request within the customer's needs and deadlines. The results of the virtual machine that finishes first are considered and results of other virtual machines are neglected [12].

Checkpointing is the other reactive method, where the request's execution status will be saved repeatedly to a stable and safe storage area during the task execution. During the situation of failure, the cloud would continue the execution of the requested task starting from the last saved check point where the status was properly recorded. This will avoid restarting the service of the request from its initial point of execution. Even though this can reduce the response time to carry a particular request, but could result in more wasted time. This wasted time owing to the recovery of a virtual machine from the failed state if it is the only one that can carry out the task. On the other hand, the cloud should make use of this method if there is only one virtual machine is available which can carry out the request of a customer's. The time gap between two

consecutive checkpoints is known as the checkpoint interval [13], [14].

In contrast, proactive methods are probabilistic and they can be employed to predict the virtual machines faults to some extent prior to their occurrence. The foremost goal of these methods is trying to avoid the occurrence of failures and then avoid recovery procedures of the reactive methods. While scheduling the requests, proactive methods take the advantage of scheduling decisions based on the information of prior failures of particular virtual machines. Consequently, the number of future failures can be reduced and the reliability of the cloud environment will be improved.

Fault tolerance is one of the most important issues in distributed computing systems such as grid and cloud computing systems. In grid computing, there is a lot of fault tolerance work have been done in the literature, whilst a little research has been devoted to the area of cloud computing.

In 2013, Hui et al. [15] proposed a fault tolerant method based on using coordinated checkpoints at the virtual machine level. Their method eliminates the unavailability with the usage of coordinated protocols for the recovery of checkpoints. In 2014, Limam and Belalem [15] together defined an adaptive checkpoint scheme with the goal of removing unnecessary checkpoints otherwise add additional checkpoints based on the existing status of a server in cloud region. Their method generally increases or decreases the checkpointing interval through a fixed rate. In 2015, J. Cao et al. [16] have defined a uniform fault tolerance strategy using checkpointing mechanism. Their method supports extensive jobs and priorities were allocated to jobs. In 2021, Purushottam S et al, [17] have defined a realized and best checkpointing control mechanism for computing systems. Their method based on aggregation of checkpointing overhead and the expected amount of rework after recovery for best checkpointing. Their defined method reduces number of check point and resulting in optimal checkpointing scheme considering real-time MTBF (mean time between failures) estimation. Further, in 2021, Yu Xiang et al, defined a Contention-Free and distributed VM checkpointing mechanism to provide reliability. Their method reduces checkpointing interference and thus improves the reliability of distributed network.

In 2013, Ganga and Karthik [9] have proposed a replication based fault tolerant method for fault tolerance while using scientific workflow systems. Das and Khilar [18] proposed a replication based method to decrease the service time and to increase the system availability. Their method depends on the usage of software variants on several virtual machines to tolerate faults. Furthermore, it reduces the likelihood of faults in future by stopping tasks scheduling to virtual machines of servers that has low success rates. Alhosban et al. [3] introduced a scheme that depends on the prediction and planning. A method of recovery is selected to be applied during the case of fault occurrance. The selection criterion depends on user requirements, failure history and requested service weight and its criticality. Methods that can be selected are replication and retry.

In 2015, Saranya et al. [19] presented and evaluated a method based on both replication and resubmission of tasks. Their defined method based on assigned task priority, task length, deadline of requested service and the out-degree of every task. In 2015, Liu and Wei [20] defined a replication scheme that considers the failures of hardware and software. In 2020, Jinwei Liu, et al [21] proposed a replication scheme for handling both correlated and non-correlated server failures, with high availability. Their replication scheme provides low cost to enterprises by reducing the replicas using correlated and non-correlated server failures. In 2021, Ahmed Awad, et al [22] presented a dynamic data replication scheme for cloud for selection and placement of data replicas. Their approach uses swarm and ant colony optimization algorithm for replica selection and placement and provides better data availability, low cost, and fewer bandwidth consumption.

In 2017, B Mohammed et al. [23] defined a smart failover framework that provides fault tolerance by considering redundancy, optimized selection of VMs, and checkpointing. Their defined scheme is similar to our scheme. Their proposed scheme uses various components in cloud such as fault manager, cloud controller, load balancer and further a selection mechanism. It is also able to eliminate temporary software faults from recoverable faulty nodes, thus making them available for further requests in the future. Further, M Amoon, also provided a similar framework that incorporates both replication and checkpointing schemes.[4]

The study of literature shows that most of the preceding work done are primarily based on using a single efficient fault tolerance method, either checkpointing or replication. There is a little work done that considers using both of the two methods together to tolerate faults in cloud computing systems. Also, most of the existing replication based work considers a static or fixed number of replicas and they do replication for all virtual machines in the cloud, which is not an economic approach. In the case of checkpointing, most of the proposed work assumes fixed or fixed change of the length of the checkpoint interval during the execution of the customer requests or jobs. There is a small effort done that considers the adaptive length of the checkpoint interval and also it is centralized mechanism. So, there is a need for a fault-tolerant strategy that considers both optimal checkpointing and replication

methods and selects the number of checkpoints or replica in an adaptive manner.

3. Proposed Model

Cloud services are provided either as storage services or computing services. Google, iCloud and Dropbox are best examples in offering storage services and Microsoft Azure and Amazon EC2 are examples cloud providers offering computing services. In order to be served, a customer submits his service request to the cloud provider along with the requirements needed for his request. The provider negotiates with the customer in order to determine both the quality of service and the price. In case of customer acceptance, the provider will formulate the cloud virtual machine that can perform the request and the service gets started.

Majority, cloud resources are not primarily designed to achieve the cloud's economic objective. These resources are composed into several virtual machines to undertake customer requests. So, it is expected that a number of failures will occur and then increase time expected to complete the customer requests and thus it will deplete the cloud resources. For customers, they will not get their services in the time expected. For the cloud, failures will lead to loss of cloud resources and then money. This will lead to a significant impact on the credibility, reliability, availability and reputation of the cloud [23]. Hence, it is more essential to implement and effective fault tolerance strategy in cloud computing regions to alleviate or omit the effect of failures on the cloud performance.

Replication of both data and applications are used by majority of the existing cloud computing systems. It is even applied in Amazon S3 via storage of data objects on multiple storage server units. The iCloud can rent various infrastructure services from Microsoft's Azure or Amazon's EC2 to accomplish the replication. However, cloud outage reports suggest the fact that the reliability is still insufficient and necessary [22]. Applying fault tolerance methods in clouds faces the following challenges:

1. The cloud can have only a single copy of the virtual machine that can carry out the request of the customer. Also, the cloud can have multiple VMs that can perform the customer's request, but in case of only one server is available and the other servers are busy in executing other requests or else they are out of service. So, replication method cannot be applied.
2. The number of replicas cannot be static or fixed as it leads to a reduced influence on the cloud. This is due to the fact that additional virtual machines will be used to carry out the same service. Nevertheless, these virtual machines are useful to perform other customer services. Consequently, the cloud will lose profits.
3. It is not economical to implement replication for each service or virtual machine. Replication should only be applied for services that are allocated to the most valuable virtual machines that will have a great impact on the performance of the cloud if they fail. Determining the most valuable virtual machines is a great challenge.
4. In case of checkpointing method, predetermining the checkpointing interval's length is a crucial challenge. Checkpointing with fixed or static checkpoint interval could lead to redundant checkpoints that consume cloud resources and increases checkpointing latency.
5. In order to cope with the first challenge, optimal and distributed checkpointing method is involved in our fault-tolerant scheme beside replication. The proposed fault-tolerant strategy allows the cloud to choose either optimal and distributed checkpointing or replication in order to accomplish fault tolerance. Further, to address the subsequent challenge, a replication algorithm that adaptively regulates the number of replicas of an application is offered. For the third challenge, the percentage of profit gained by the cloud when using the virtual machine is involved in determining the number of replicas required for each virtual machine. For the fourth challenge, an optimal algorithm that adaptively determines the checkpointing interval's length is proposed. The algorithm assumes that the length of the checkpointing interval must not be fixed during the execution of the customer's task. The algorithm takes the failure probability of a virtual machine to estimate the subsequent checkpointing interval.

3.1. Cloud and Proposed Architectures

Cloud computing environments should have the ability to receive, perform, monitor and control customers' requests. The cloud should be reliable in order to provide its services within the limits of customer requirements. This section describes the proposed framework which enables the cloud to be reliable. As shown in Figure 1, the architecture of the proposed fault-tolerant strategy assumes that the cloud comprises of three major layers: physical, VM and application layers. One function of the application layer is to allow customers to interact with the cloud. Furthermore, it schedules the customers' requests or jobs to the virtual machines in the cloud. In addition, tolerating faults is the responsibility of the application layer. In order to perform these functions, the structure of the application layer comprises four modules:

1. *Service Verifier*: This module is liable towards ensuring the accomplishment of customer's QoS requirements. In this paper, the considered QoS requirements include time and monetary costs. A customer can submit his request to the cloud through this module along with the

QoS requirements. The module queries the Status Database module for appropriate VMs availability to execute the customer request and gets a response. If the response indicates the presence of appropriate VMs that can carry out the request within the boundaries of customer requirements, the Service Verifier will accept the request and it will deliver it to the Task/Job Scheduler module. Otherwise, the request will be discarded.

2. *Task/Job Scheduler*: The main function of the *Task/Job* Scheduler is to allocate every request to the appropriate virtual machine that can execute it within the boundaries of customer requirements. Also, the Job scheduler has the responsibility of determining the charge of serving the request. In addition, Job Scheduler has the responsibility of fault tolerance. In order to do its responsibilities, the *Task/Job* Scheduler module should contain the following components: VM Ranker, Price Estimation, Scheduling and Fault Tolerance Manager (SFTM), ODCP (Optimal and distributed checkpointing), replication modules and Dispatcher. Figure 2 illustrates the interactions between the main components of the Scheduler. The main role of the VM Ranker is to determine the most valuable VMs in the cloud. It receives customer's request with QoS requirements from the Service Verifier and contacts the Status Database module in order to get information about the virtual machines that can accomplish the request. Based on this information, it prepares a list of VMs that can fulfill the time and monetary requirements of the customer's request. Price Estimation component determines the charge associated to service that should be paid by the customer. SFTM component implements Algorithm 1 in order to select the appropriate fault tolerance method for the virtual machine assigned to each request. The algorithm selects either optimal and distributed checkpointing or replication based on information about virtual machines. Dispatcher delivers the requests of customers to the allocated VMs.

3. *Status Database*: It is known to be the central repository that contains all virtual machines information in the cloud such as storage capacity, computing capacity, failure history, usage history and cost.

4. *VM Monitor*: The foremost functionality of this module is to observe the performance of the virtual machines in the cloud. It notifies the Status Database to update the record of a VM in a case of the failure or the recovery of that VM. In addition, this module has the responsibility for forming or reforming virtual machines of the cloud. It has virtualization software that is helpful to produce unique and isolated virtual machines through cloud physical resources.

5.

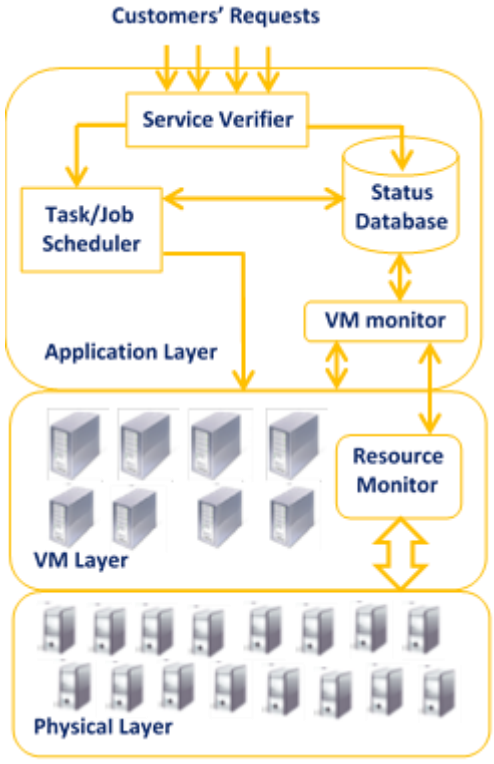

Fig 1: Cloud computing system: A layered architecture.

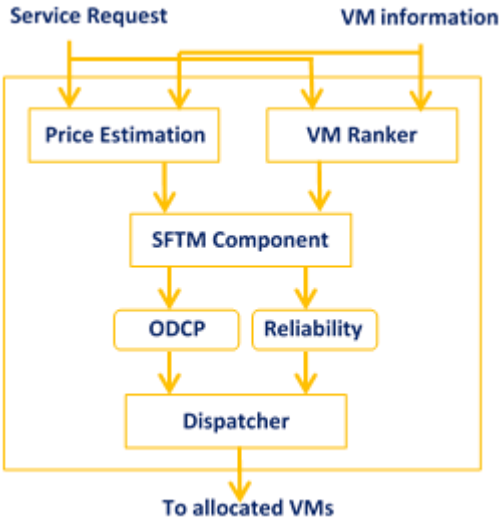

Fig 2: Task/Job scheduler components and their interactions.

VM layer is the second layer, which comprises virtual machines of the cloud where each virtual machine is formed with the usage of one or more physical resources. Also, each physical resource may be shared and used by multiple virtual machines. Furthermore, different VMs can be emulated on a single physical resource in order to satisfy the requirements needed by customer requests. The Resource Monitor in this layer is useful to perceive the performance of the physical resources of the cloud and further it notifies the VM monitor about the changes happened. Changes include resources leaving the cloud or new resources joining the cloud. Based on these changes, Resource Monitor is capable to reform the affected virtual

machines.

The physical layer is the bottom layer of the cloud which contains hardware and software resources. Resources are the real operators in the cloud environment.

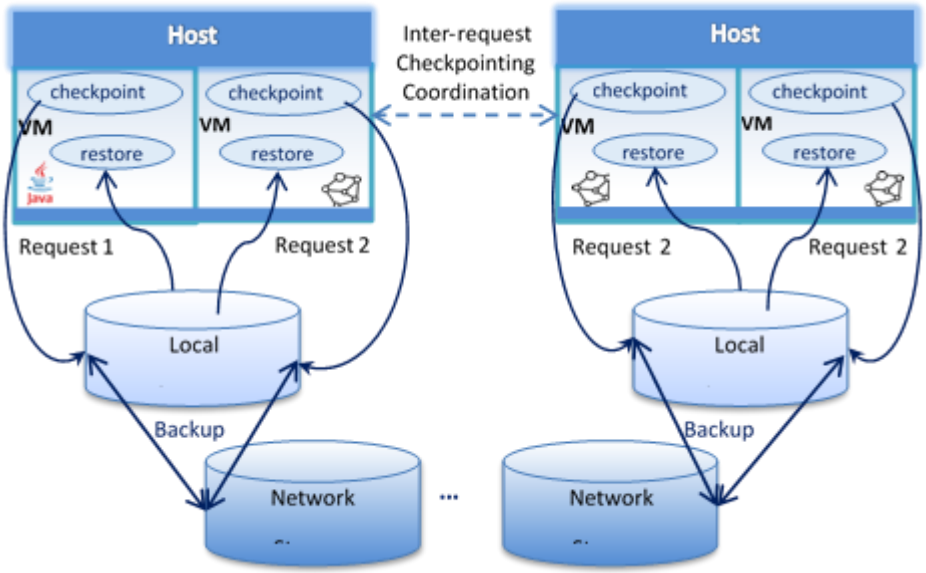

Fig 3: Architecture of ODCP scheduling consisting two requests allotted with 1 and 3 VMs respectively and hosted on 2 hosts.

Figure 3 shows an overview of the proposed system architecture. It illustrates 2 jobs consisting of 1 and 3 VMs respectively and placed on 2 hosts. Our checkpoints are organized at the job level - if a checkpoint of a job is triggered, all VMs that belong to the job first save their checkpoint images to the local storage (in order to minimize VM downtime) and then transfer them to the networked storage to avoid host failure.

In our design, each job achieves reliability optimization via self-management in two ways: first, each job autonomously determines its own checkpointing scheduling based on locally available information, e.g., the co-location of other jobs and occurrence of checkpoint contention. Second, each job autonomously updates its checkpoint rate based on locally available optimal solutions, which is done at runtime with no dependence on any centralized management decisions.

3.2. Implementation of Fault-tolerant Strategy

3.2.1. SFTM Algorithm

Algorithm 1 is called the Selecting Fault Tolerance Manager (SFTM) algorithm and it is proposed with the objective to select the appropriate method for tolerating faults in the cloud computing system. The algorithm is implemented in the SFTM component of the Scheduler module. In order to achieve its objective, the algorithm depends on using customer's requirements and the available information about virtual machines. First, the algorithm prepares a list of virtual machines that can carry out the customer's request and satisfies the customer's requirements. The customer's requirements considered by the algorithm include both time costs and monetary costs. Thereafter, the algorithm selects checkpointing method if there is only a single VM in the list. Otherwise, the algorithm selects replication method.

3.2.2. Replication Algorithm

Replication is applied when there are multiple and available virtual machines in the cloud that can carry out the customer's request. However, it is a central challenge to define the optimal number of replicas. Furthermore, it is not an economical method to carry out replication for every virtual machine [24]. So, we only need to replicate requests executed on the most valuable virtual machines that will have a great impact on the performance of the cloud if they fail.

Algorithm 1: *SFTM Algorithm*

Input: c_{iu} is the required cost by the customer u for request i,

τ_{iu} is the required deadline time by the customer u for request i,

c_{ij} is the estimated cost if the request i is executed by VM_j,

τ_{ij} is the expected time if the request i is executed by VM_j,

$i = 1$;

while (there are requests not served){

```
    for each request i do{
      Identify a list of VMs that can execute i;
      for each VM_j in the list do
        if (c_ij > c_iu || τ_ij > τ_iu)/*VM_j cannot serve request i*/
          remove VM_j from the list;
        if (list is not empty) /*The request can be served */{
          Sort the VMs list ascending based on c_ij × τ_ij;
          if (there is more than one VM in the list)
            Replication is selected;
          else
            Optimal and distributed checkpointing is selected;
        }
        else {
          Send ''Request cannot be served'' to the QoS Controller;
          End the algorithm for i;
        }
        i++; /* next request */
      }/* for end*/
    }/* while end*/
```

In order to find the top valuable VMs in the cloud, VMs ranking need to be performed according to their price and influence on the cloud. The ranking is based on failure probability of the virtual machine and the profit gained through using it. Failure history of a VM can determine its failure probability. For each virtual machine, failure history can be represented by the number of failures occurring, failure time, the time between failures and failure types. The need of a virtual machine to a fault tolerance method is determined by failure probability. As the value of the failure probability becomes high, the need for applying fault tolerance methods increases.

Algorithm 2: *Replication Algorithm*

- $F_j(X)$: The failure probability of a VM_j
- P_j : The percentage of profit gained through the usage of VM_j
- *Rep* : The number of replicas
- $F_j\ (X)\ (k)$, $k = 0, 1, 2, \ldots, n$, are integers such that,

  $0 <= F_j\ (X)\ (k) <= 1.0$ and $F_j\ (X)\ (0) < F_j\ (X)\ (1) < \ldots < F_j(X\ )(n)$

- $P_j(y)$, $y = 0, 1, 2, \ldots, m$, are the percentage of cloud profit gained by virtual machine j such that,

  $0 <= P_j(y) <= 100$ and $P_j\ (0) < P_j\ (1) < \ldots < P_j\ (m)$

- *Rep (l) (w)* , $l = 0, 1, 2, \ldots, n$ *and* $w = 0, 1, 2, \ldots, m$, are integers

```
for (a = 0; a < n; a++){
  for (b = 0; b < m; b++){
    if (F_j (X) (a) ≤ F_j(X ) < F_j(X )(a + 1) and P_j (b) ≤ P_j < P_j (b + 1))
      Rep = Rep (a) (b);
  }
}
```

In general, the occurrence of random failures is a stochastic process [25] and Jump Linear Systems (JLSs) can be used to model it because they involve event driven and time evolving techniques. The process depends on the time period between two successive faults. In clouds, this time period is a random variable following general probability distributions and the process is often called semi-Markov process. The jump linear system of the semi-Markov process is known as semi-Markovian JLS with time-varying transition rates [26].

In this work, the failure probability of a virtual machine is assumed to follow Poisson distribution. This means that the number of failures in any two different or disjoint periods of time is independent over the time change. The failure probability distribution of VM_j at any given time interval can be expressed as follows:

$$F_j(X) = \frac{e^{-\mu}\mu^x}{x!}, 0 \le F_j(X) \le 1 and\ x = 0,1,2,\ldots,n, \quad (1)$$

where X (x_0, x_1, x_2, . . . , x_n) represents the number of failures occurred in a certain time period and μ is the average number of failures in the specified time period for a virtual machine j. The value of μ is calculated using:

$$\mu = \frac{f_j}{T_j/\tau_{ij}}, \quad (2)$$

where, f_j is the number of failures of a virtual machine j and T_j is the period of time in which f_j failures have occurred. τ_{ij} represents the estimated time when request or application i is executed on virtual machine j. Thus, the probability of one failure ($x = 1$) to take place during the execution of a request

is given by:

$$F_j(x_1) = \mu e^{-\mu}. \quad (3)$$

The virtual machine profit, denoted as P_j, represents the percentage of cloud profit gained through the usage of

virtual machine *j* in performing requests. The value of the virtual machine *j* to the cloud is determined by its profit. If the profit obtained by virtual machine is huge then it attains more value in the cloud.

The rank of a virtual machine is computed by the VM Ranker component of the Task/Job scheduler. The VM Ranker component acquires the failure probability and associated VM's profit from the Status Database. Thereafter, it calculates the rank of each virtual machine using the formula:

$$R_j = \mu e^{-\mu} \times P_j \,, \tag{4}$$

where R_j is the rank of VM_j, $\mu e^{-\mu}$ is the probability of a failure to occur and P_j is the profit of VM_j.

The fixed number of replicas is not an efficient choice in cloud computing environments because additional virtual machines will be used to carry out the same request. However, these virtual machines can be used to carry out requests of other customers. Thus, profit charges will be wasted. Also, it is not economically to implement replication for each request or for each VM.

Algorithm 2 is the replication algorithm proposed in this paper in order to adaptively determine the number of replicas of a request. The number of replicas will not be fixed for all requests or virtual machines. In order to adaptively determine the number of replicas, the operation of the algorithm depends on both the failure probability and the percentage of cloud profit gained by the virtual machine allocated to execute the customer's request. As either the failure probability or profit percentage of a virtual machine increases the need for more replicas increases. Consequently, virtual machines with higher values of profit or failure probability have higher fault-tolerance needs and then higher priority of replication than other VMs.

3.2.3. Checkpoint and ODCP Scheduling Algorithms

Distributed systems, such as grid computing systems, have widely used checkpointing as a reactive fault tolerance method to alleviate the impact of failures when occurred. Furthermore, most cloud computing systems implement replication techniques. However, from the perspective of the cloud service provider, replication results in profit loss due to allocating additional components to execute the replicas of a request, mostly these components may be useful for other requests. Also, from the perspective of customers, replication leads to time loss due to waiting for components that execute replicas to be free from executing other requests. So, the main advantage of using checkpointing over replication is to preserve the computing resources of the cloud to other customers' requests and to reduce the profit loss because of using replication.

Checkpointing interval and latency are the two parameters that strongly affect a checkpointing algorithm. The checkpointing interval represents the time between a checkpoint and the next checkpoint. Checkpointing latency is the time consumed in saving a checkpoint. In the case of smaller checkpoint interval, there will be a large number of checkpoints. This large number of checkpoints will heavily consume cloud resources when saving checkpoints and thus high checkpointing latency results in. Additionally, long checkpoint interval leads to a lesser number of checkpoints and then a considerable part of the request should be recomputed in the case of failure. This least number of checkpoints will slightly consume cloud resources while saving checkpoints and consequently low checkpointing latency results in.

So, determining the length of the checkpointing interval is the major challenge for a checkpointing technique. Fixed interval leads to redundant checkpoints that consume cloud resources and increase checkpointing latency. So, the main objective of our work is to develop an algorithm that adaptively determines the length of the checkpointing interval. Algorithm 3 assumes that the length of the checkpointing interval must not be fixed during the execution of the customer's task. The algorithm calculates the next checkpointing interval at the time of the current checkpoint. It is calculated based on the failure history of the VM on which the task is executed. In the case of a poor failure history, the algorithm will shorten the checkpoint interval. Additionally, the algorithm will prolong the checkpoint interval if there is a good failure history.

Algorithm 3: *Checkpoint Algorithm*

- τ_{ij} : The execution time of job i on VM_j
- Tr_{ij} : The remaining execution time of job *i* on VM_j
- $F_j\ (x_1)$: Failure probability of VM_j
- $F_j\ (x_0)$: Probability of no failure of VM_j
- *l*: Checkpoint interval
- *z*: Number of failures during the task execution

Calculate $F_j\ (x_1) = \mu e^{-\mu}$;

For each task *j* allocated to VM_j do{

$z = 0$; $l = \tau_{ij} \times Fj\ (x_z)$; //Initial checkpoint interval

$\tau r_{ij} = \tau_{ij}$;

Start execution of *i* on *j*;

do{

$\tau r_{ij} = \tau r_{ij} - l$;

if failure occurred then{

z++;

$\tau r_{ij} = \tau r_{ij} + l$;

$l = l(1 - F_j(x_z))$; // decrease checkpoint interval

Restore last checkpoint;

Restart execution from $\tau_{ij} - \tau r_{ij}$;

}

At time $\tau_{ij} - \tau r_{ij}$ create and schedule a checkpoint;

$l = l(1 + F_j(x_z))$; // increase checkpoint interval

Resume execution;

}While ($\tau r_{ij} \neq 0$)

}

Algorithm 4: *ODCP Scheduling Algorithm*

- λ_i *:* Sensing rate of request *i*
- $\mathfrak{H}_i$*:Servicing hosts in cloud*
- B_i*: Back-off time*

Assign positive sensing rates $\lambda_i > 0 \; \forall i$

Each request independently performs:

Initialize backoff timer B_i

while request *i* is running

while $B_i > 0$

if any server in $\mathfrak{H}_i$ is busy

request *i* becomes silent

Generate new backoff: B_i = exponential with mean $1/\lambda_i$

end if

Update $B_i = B_i - 1$

end while

Checkpoint all VMs of request *i*

Generate new backoff: B_i = exponential with mean $1/\lambda_i$

end while

The proposed ODCP scheduling works as follows: Each request *i* makes the decision to create a remote checkpoint image based only on its local parameters and observation of contention. If request *i* senses there are ongoing checkpoints at any of its serving hosts (i.e., any host *s* such that $s \in \mathfrak{H}_i$), then it becomes silent. If none of its serving hosts are busy, then request *i* waits (or backs-off) for a random amount of time that is exponentially distributed with the mean $1/\lambda_i$ and later starts its checkpointing. During the back-off time, if some contending request starts taking checkpoints, then request *i* suspend its back-off until the contending checkpoint is complete. We note that waiting a random back-off time with different mean permits us to regulate different requests' checkpointing probabilities. It will not cause too much idle time because only the relative values of $1/\lambda_i$ matter and the mean waiting time can be set small enough in this CSMA (Carrier Sense Multiple Access) based model.

4. Results

There are many available cloud-simulator environments and CloudSim is one of the most of them [27], [28]. Among all classes and packages of the CloudSim, there is no one that supports the implementation of fault-tolerant clouds. So, the creation of an extra package is needed in order to support the implementation of fault-tolerant methods in the cloud computing systems. This created package provides services of fault tolerance through allowing some virtual machines of cloud data centers to be faulty. The classes of the package allow the development of fault tolerant based algorithms that can monitor virtual machines in order to detect failures and resolve them. The package can implement both checkpointing and replication techniques. The package provides the ability to measure throughput, availability, time overhead and monetary waste overhead.

The cloud used in our experiments is generated with 150 heterogeneous virtual machines that are connected with fast Ethernet technology (100Mb/s). The number of data centers used in each experiment ranges from 6 to 10. Each data center contains 3 to 4 hosts. The size of each host's memory is 16 GB and the storage is 2TB. The processing capacity of computational units in each host is assumed to be in the range from 1000 to 10000 MIPS. The number of customer requests ranges from 500 up to 3000 requests. Each virtual machine has a memory of 4 GB and one computational unit. The size of data required for each request processing is randomly selected from 10 MB and up to 1 GB. The cost associated with each cloud computing unit is assumed to be in the range from \$0.1 to \$12.

We evaluate the performance of our proposed fault-tolerance strategy by comparing it with the checkpointing based algorithm proposed in 2021 [12], named contention-free and distributed checkpointing scheduling (CDCS) scheme, which is based on using distributed checkpointing mechanism. Various simulation experiments have been carried out with a variable size of customers' requests. The performance metrics adopted in the evaluation consist of availability, throughput, checkpoints overheads and monetary cost.

Figure 4 demonstrates the results of the throughput comparison between the proposed fault-tolerant strategy and contention-free and distributed checkpointing scheme. The number of requests is presented in the x-axis and the

results of throughput, measured in requests per hour, are plotted as columns. Generally, the throughput of the proposed fault-tolerant strategy when compared with CDCS scheme increases with the increase in the submitted customers' requests. The figure clearly demonstrates that the proposed fault-tolerant strategy has a better throughput than the CDCS scheme. This is because the proposed fault-tolerant strategy has less turnaround time than the CDCS algorithm. This is attributed to the fact that the proposed algorithm considers the failure probability as a criterion when selecting virtual machines to carry out requests. Conversely, the CDCS algorithm considers only reducing checkpointing overhead. This makes our proposed fault-tolerant strategy is less prone to fail consequently more reliable than the CDCS algorithm.

Figure 5 demonstrates the comparison of checkpointing overheads of proposed fault-tolerant strategy with respect to CDCS algorithm. The figure indicates that the overheads of the proposed fault-tolerant strategy are less than that of the CDC algorithm. This is because our proposed fault-tolerant strategy adaptively determines the length of the next checkpointing interval and also distributed while the CDCS scheme changes it with constant rates. Thus, the proposed fault-tolerant strategy eliminates the redundant checkpoints and finally overheads are reduced with distributed nature.

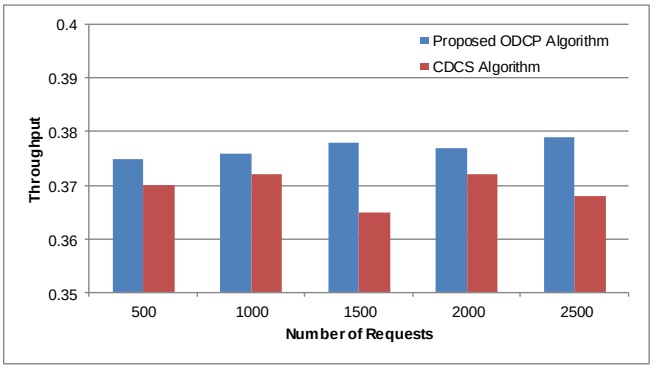

Fig 4: Comparison of Throughput.

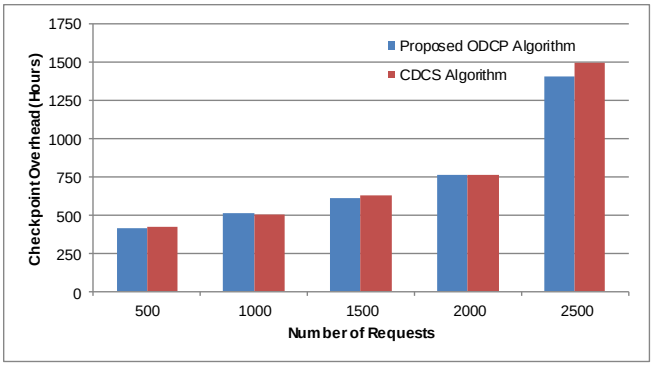

Fig 5: Comparison of Checkpointing Overhead.

Figure 6 shows the comparison of monetary cost between the proposed fault-tolerant strategy and CDCS mechanism. We can see that the proposed fault-tolerant strategy has a lesser monetary cost than the CDCS mechanism. This is due to that the proposed fault-tolerant strategy has a less failure rate and a fewer number of checkpoints than the CDCS algorithm. This will protect the cloud resources for further customer requests and consequently money is saved.

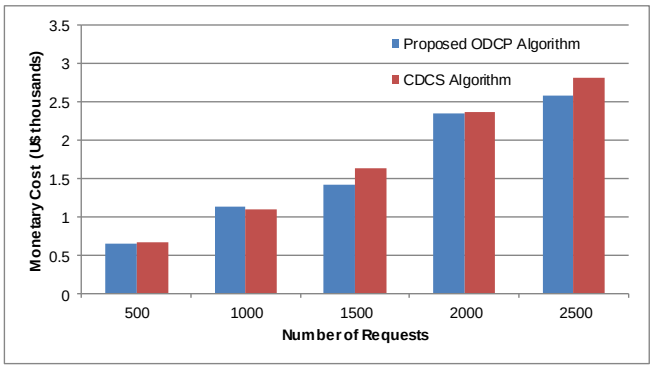

Fig 6: Comparison of Monetary Cost.

The service availability of a cloud depends on percentage of the service time and failure rate. The service time is known as cloud operational time at certain instant of time. Figure 7 shows the comparison of availability between the proposed fault-tolerant strategy and CDCS algorithm. The figure illustrates that the proposed fault-tolerant strategy provides better availability than the CDCS algorithm. This is owing to the nature of adaptive checkpoint interval that our proposed fault-tolerant strategy provides which helps to reduce the failure rate.

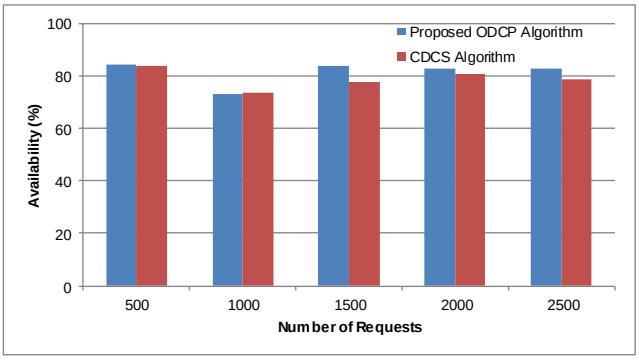

Fig 7: Comparison of Availabilty.

Also, we evaluate the performance of our proposed fault-tolerant strategy by comparing it with the replication based algorithm proposed in 2023 [12]. As this algorithm considers a dynamic with variable number of replicas but it is done initially, we will denote it the dynamic resource estimation as static algorithm only. Figure 8 illustrates overheads' comparison between the proposed fault-tolerant strategy and static resource estimation algorithm. The term overhead denotes the number of replicas required for a task in the cloud. The figure demonstrates that the overheads of the proposed fault-tolerant strategy are better than static

resource estimation algorithm. This is because the proposed scheme chooses an adaptive number of replicas that can dynamically change for every request based on the existing conditions of the virtual machine allocated to execute the request. Conversely, the static resource estimation algorithm selects a fixed number of replicas without considering the current situation of the virtual machines allocated.

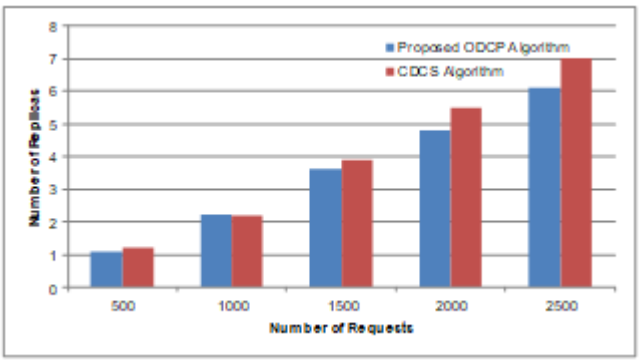

Fig 8: Comparison of Replica Overhead.

Figure 9 shows the monetary waste comparison between the proposed fault-tolerance strategy and static algorithm. We can see that the proposed fault-tolerance strategy has a lower monetary waste than the static algorithm. This is because the proposed fault-tolerance strategy only replicates the top-most valuable virtual machines compared to the static algorithm that replicates all VMs. This will decrease the number of virtual machines consumed in the replication. Thus, profit of the cloud will not be lost.

From the above results of the experiments, it is shown that the proposed fault-tolerance strategy improves the performance of the cloud in terms of availability, throughput, overheads and monetary cost. The adaptive nature of the proposed strategy gives it supremacy over the other similar ones. This adaptive nature appears when determining the number of replicas for virtual machines or when calculating the checkpoint intervals with distributed nature for storage. Improving throughput will improve the number of services the cloud can serve in the same time and then the profit of the cloud increases. Improving the overheads leads to saving resources of the cloud for other customers. This will decrease the waiting time of a customer request in the cloud. Improving the amount of monetary waste will permit the cloud provider to improve the services of the cloud via continuous maintenance and new resources are added efficiently. Increasing availability of the cloud will strengthen the customers trust.

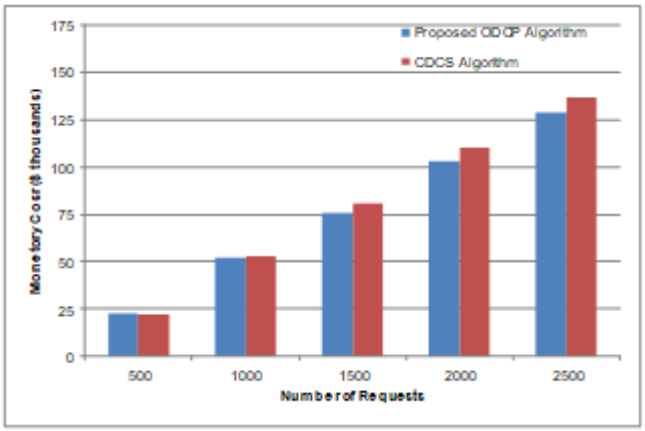

Fig 9: Comparison of Monetary cost.

Table 1: *Performance Improvement*

Metrics	Percentage of Improvement
Throughput	2%
Availability	3%
Overheads	6.6%
Monetary cost	5.3%

5. Conclusion

Failures are unavoidable in cloud computing environments. To treat this issue, an adaptive fault-tolerant strategy for tolerating faults in cloud computing environments has been proposed in this paper. The proposed strategy has one algorithm for selecting virtual machines to carry out customers' requests and another algorithm for selecting the suitable fault tolerance method. Both replication and checkpointing methods are included in the proposed strategy. The performance of the scheme is evaluated with a replication-based algorithm and also with a checkpointing based algorithm in terms of throughput, monetary cost, cloud overheads and availability. Experimental results indicate that the proposed scheme improves the cloud's performance as shown in Table 1.

In the future work, we will provide more considerations towards replication and the task migration between data centers.

References

[1] R. Buyya, C. Yeo, S. Venugopal, J. Broberg, and I. Brandic, ''Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility,'' *Future Generat. Comput. Syst.*, vol. 25, no. 6, pp. 599–616, Jun. 2009.

[2] M. Chen, Y. Ma, J. Song, C. -F. Lai, and B. Hu, ''Smart Clothing: Connecting human with clouds and big data for sustainable health monitoring,'' *Mobile Netw. Appl.*, vol. 21, no. 5, pp. 825–845, Oct. 2016.

[3] A. Alhosban, K. Hashmi, Z. Malik, and B. Medjahed, ‘‘Self-healing framework for cloud-based services,’’ in *Proc. Int. Conf. Comput. Syst. Appl.*, May 2013, pp. 1–7.

[4] Mohammad Amoon, “Adaptive Framework for Reliable Cloud Computing Environment”, *IEEE Access*, vol. 4, pp. 9469-9478, Nov. 2016.

[5] M. Armbrust *et al.*, ‘‘Above the clouds: A Berkeley view of cloud computing,’’ Univ. California at Berkeley, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2009-28. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf.

[6] Ng, Abigail, "Google is back online after users around the world reported a brief outage". CNBC. [Online] Available: https://www.cnbc.com/2022/08/09/google-down-outage-reported-by-thousands-users-around-the-world.html. Retrieved 9 Aug. 2022.

[7] K. Bilal *et al.*, ‘‘Trends and challenges in cloud data centers,’’ *IEEE Cloud Comput. Mag.*, vol. 1, no. 1, pp. 10–20, 2014.

[8] Zulfiqar Ahmad, Ali Imran Jehangiri, Nader Mohamed, et al, “Fault Tolerant and Data Oriented Scientific Workflows Management and Scheduling System in Cloud Computing”, in *IEEE Access*, vol. 10, pp. 77614-77632, 2022.

[9] K. Ganga and S. Karthik, ‘‘A fault tolerent approach in scientific workflow systems based on cloud computing,’’ in *Proc. Int. Conf. Pattern Recognit., Informat. Mobile Eng. (PRIME)*, Feb. 2013, pp. 378–390.

[10] A. U. Rehman , Rui L. Aguiar, et al, “Fault-Tolerance in the Scope of Cloud Computing”, in *IEEE Access*, vol. 10, pp. 63422-63441, June 2022.

[11] Vahid Mohammadian, Nima Jafari Navimipour, et al, “Fault-Tolerant Load Balancing in Cloud Computing: A Systematic Literature Review”, in *IEEE Access*, vol. 10, pp. 12714-12731, 2022.

[12] Deepika Saxena and Ashutosh Kumar Singh, “A High Availability Management Model based on VM Significance Ranking and Resource Estimation for Cloud Applications”, *IEEE Transactions On Services Computing*, vol. 16, Issue 3, pp. 1604-1615, 2023.

[13] H. Hui et al., ‘‘An efficient checkpointing scheme in cloud computing environment,’’ in Proc. 2nd Int. Conf. Comput. Appl., Harbin, China, 2013, pp. 251–254.

[14] Yu Xiang, Hang Liu, Tian Lan, et al, “Optimizing Job Reliability Through Contention-Free, Distributed Checkpoint Scheduling”, Vol. 18, Issue: 2, pp. 2077-2088, 2021.

[15] S. Limam and G. Belalem, ‘‘A migration approach for fault tolerance in cloud computing,’’ Int. J. Grid High Perform. Comput., vol. 6, no. 2, pp. 24–37, Apr./Jun. 2014.

[16] J. Cao, M. Simonin, G. Cooperman, and C. Morin, ‘‘Checkpointing as a service in heterogeneous cloud environments,’’ in Proc. 15th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput., Shenzhen, China, May 2015, pp. 61–70.

[17] Purushottam Sigdel, Xu Yuan, et al, “Realizing Best Checkpointing Control in Computing Systems”, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 32, Issue: 2, pp.315-329, 2021.

[18] P. Das and P. M. Khilar, ‘‘VFT: A virtualization and fault tolerance approach for cloud computing,’’ in *Proc. IEEE Conf. Inf. Commun. Tech- nol. (ICT)*, Apr. 2013, pp. 473–478.

[19] S. M. Saranya, T. Srimathi, C. Ramanathan, and T. Venkadesan, ‘‘Enhanced fault tolerance and cost reduction using task replication using spot instances in cloud,’’ *Int. J. Innov. Res. Sci., Eng. Technol.*, vol. 4, no. 6, pp. 12–16, May 2015.

[20] Y. Liu and W. Wei, ‘‘A replication-based mechanism for fault tolerance in mapreduce framework,’’ *Math. Problems Eng.*, vol. 2015, 2015, Art. no. 408921.

[21] Jinwei Liu; Haiying Shen, et al., “A Low-Cost Multi-Failure Resilient Replication Scheme for High-Data Availability in Cloud Storage”, in *IEEE/ACM Transactions on Networking*, Vol. 29, Issue: 4, pp. 1436-1451, Aug. 2021.

[22] Ahmed Awad, Rashed Salem, “A Novel Intelligent Approach for Dynamic Data Replication in Cloud Environment”, in *IEEE Access*, vol. 9, pp. 40241-40254, 2021.

[23] Bashir Mohammed, Mariam Kiran, et al., “Failover strategy for fault tolerance in cloud computing environment”, in Wiley Online Library, DOI: 10.1002/spe.2491, 2017. [Online].

[24] E. Bauer and R. Adams. *Reliability and Availability of Cloud Computing*. Hoboken, NJ, USA: Wiley, 2012.

[25] Y. Wei, J. Qiu, H. Lam, and L. Wu, ‘‘Approaches to T-S fuzzy- affine-model-based reliable output feedback control for nonlinear Ito stochastic systems,’’ IEEE Trans. Fuzzy Syst., to be published, doi: 10.1109/TFUZZ.2016.2566810.

[26] Y. Wei, X. Peng, and J. Qiu, ‘‘Robust and non-fragile static output feedback control for continuous-time

semi-Markovian jump systems,'' Trans. Inst. Meas. Control, vol. 38, no. 9, pp. 1136–1150, 2016.

[27] *CloudSim: A Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services*. (Apr. 2016). [Online]. Available: http://www.cloudbus.org/cloudsim.

[28] Amit Sundas, Surya Narayan Panda, "An Introduction of CloudSim Simulation tool for Modelling and Scheduling", *2020 International Conference on Emerging Smart Computing and Informatics (ESCI),* [Online]. *DOI: 10.1109/ESCI48226.2020.9167549.*