International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 11, No.1 (Mar-2022)

https://dx.doi.org/10.12785/ijcds/1101101

Fault Tolerance Directed by Service Level Agreement in Cloud Computing Environments

Samir Setaouti[1], Djamel Amar Bensaber[2], Reda Adjoudj[1] and Mohammed Rebbah[3]

[1]*Evolutionary Engineering & Distributed Information Systems Laboratory (EEDIS), Computer Science Department, Djillali Liabes University of Sidi Bel Abbes, Sidi Bel Abbes, Algeria*
[2]*LabRI-SBA Laboratory, Ecole Superieure en Informatique, Sidi Bel Abbes, Algeria*
[3]*Computer Science Department, Mustapha Stambouli University of Mascara, Mascara, Algeria*

Received 5 Jul. 2021, Revised 4 Feb. 2022, Accepted 9 Mar. 2022, Published 31 Mar. 2022

Abstract: The Cloud Computing environments are susceptible to frequent failures as a result of their dynamic and unstable nature. Failures have a significant effect on cloud performance and on the benefits that customers and providers expect. Therefore, fault tolerance is necessary to mitigate the impact of failures and preventing revenue losses due to service level agreements (SLA) violation penalties. In this paper, we propose a fault tolerance directed by the SLA contracts established between cloud providers and customers. The proposed fault tolerance includes two phases, the first is based on the use of idles VMs according to selection strategies. The second phase is based on both advanced operation of degradation of quality of service and on VMs selection strategies. The advanced operation of degradation consists of beneficial combinations of VMs deallocation and allocation between customers leading to avoid SLA violation penalties. According to the experimental results, our proposed fault tolerance decreases by more than 10% the rate of considered SLA violations compared to existing approaches.

Keywords: Cloud Computing, Service Level Agreement, Fault Tolerance, SLA Violation, performance, Availability.

1. Introduction

According to the U.S. National Institute of Standards and Technology (NIST) definition [1]: "Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (for example servers, networks, storage, services, and applications) that can be quickly provisioned and released with least management effort or service provider interaction". The cloud resources are delivered to customers as a service in three main categories: infrastructure as a service (IaaS), platform as a service (PaaS) and software as a service (SaaS) [2]. The relations between Cloud providers and customers are managed by a clear Service Level Agreement (SLA). The SLA is defined as a contract that describes the negotiated service, the quality of service (QoS), the QoS metrics, the price and arrangements in cases of violations.

The cloud computing, as a fast evolving technology, is increasingly getting used to host many business or enterprise applications. However, these environments are known for their highly distributed and heterogeneous nature. Consequently, several types of faults may occur in the cloud environment leading to service unavailability and performance degradation [3] [4]. Availability and performance of cloud services are essential for maintaining customer's confidence and preventing revenue losses due to service level agreement violation penalties [5] [6]. To make cloud computing viable for both customers and providers, faults should be handled effectively by fault tolerance mechanisms.

Fault tolerance (FT) refers to a system's ability to continue performing its expected operation despite faults. In other words, FT is related to reliability, successful operation, and absence of breakdowns [7]. In the cloud computing system, fault tolerance awareness has been identified as one of the main issues to ensure reliability, robustness and availability of important services as well as running of applications [8].

In [9], we proposed a basic fault tolerance model that uses a random quality of service degradation. In this work, we go further by proposing a new fault tolerance directed by the SLA contracts established between cloud providers and customers. The proposed fault tolerance includes two phases, the first is based on the use of idles VMs according to selection strategies. The second phase is based on both advanced operation of degradation of quality of service and on VMs selection strategies. The advanced operation of degradation consists of beneficial combinations of VMs

E-mail address: samir.setaouti@univ-sba.dz, setao.samir@gmail.com, d.amarbensaber@esi-sba.dz, reda.adjoudj@univ-sba.dz, rebbahmed@univ-mascara.dz

deallocation and allocation between customers leading to avoid SLA violations penalties. The execution of the two FT phases depends on their efficiency in avoiding SLA violations. This efficiency is evaluated by a specific function.

This paper makes the following contributions:

The design of a generic SLA representation model that can be applied to different types of platforms. This model defines the type of resources requested, the margin of degradation accepted and the different regular and irregular states of customers that use the platform resources.

The proposal of a faults tolerance directed by the SLA contracts to avoid SLA violations penalties. The proposed FT provides three techniques: Faults Tolerance with the strategy Max Available (FT-MA), Faults tolerance with the strategy High Capacity (FT-HC) and Faults Tolerance with the strategy Low Capacity (FT-LC).

Implementation of experimentation to evaluate the effectiveness of our proposed model compared to other existing approaches.

This paper is divided into 5 sections. Section 2 presents related work, followed by the proposed model in section 3, in which we present a generic SLA representation model, the platform model and the proposed fault tolerance. Section 4 shows the experimental setup and the results. Section 5 concludes the paper with a conclusion and future research directions.

2. RELATED WORKS

For the various systems and platforms, the fault tolerance operation ensures a certain level of service in case of failure. This section is divided into two parts, the first part deals with the SLA and the second one discusses work realized on fault tolerance in cloud computing environments.

A. *Service Level Agreement*

A Service Level Agreement is defined by Wieder et al. [10] as a formal negotiated agreement between the service provider (SP) and the customer. Its goal is to establish a common understanding about QoS, priorities and responsibilities. SLAs can cover many aspects of the customer-SP relationship, including service performance, customer service, billing, and service provisioning.

Several works about SLA have emerged with the appearance of service oriented architectures (SOA) whose purpose is to ensure the QoS of Web services [10]. WSLA [11] is a framework composed of three sections: the parties, the service definitions and the obligations. WSLA offers the possibility of enriching the directory of performance criteria by creating new metrics. Slang [12] is a language for the SLA description, it is modeled in UML. In An SLA, Slang defines the responsibilities of customers and suppliers and the mutual responsibilities to determine the obligations of each party. WSOL [13] is a specification based on XML, which allows offering web services with different levels of services via functional constraints, constraints of QoS, price and access rights.

The rSLA [14] is a framework for managing Service Level Agreements in Cloud environments, it contains three components: the rSLA language to formally represent SLAs, the rSLA Service, which interprets the SLAs and implements the behavior specified in them, and a set of Xlets-lightweight, dynamically bound adapters to monitoring and controlling interfaces. In [15], Kouki et al propose a solution for managing SLA Cloud Service Contracts for Models (IaaS, PaaS, SaaS) where they introduces the Cloud Service Level Agreement (CSLA) as a new SLA language directly integrating some features dealing with QoS uncertainty and Cloud fluctuation.

CSLAM [16] is a Cloud SLA management framework based on WSLA which provides both SLA negotiation language and management framework. CSLAM also manages the SLA aware inter-cloud service provision. In [2] Labidi et al., propose CSLAOnto A Comprehensive Ontological SLA Model in Cloud Computing, whose objectives are : Improving the SLA representation and facilitating its Understanding, Providing an automated means of SLA monitoring for the client, Informing the supplier of the QoS degradation by notification.

B. *Faults Tolerance*

Fault tolerance in Cloud Computing platforms is a critical issue. Several works with different mechanisms and objectives have been proposed in this area. Uesheng Tan [17] describes a replication solution for VM FT, exclusively managed by the cloud provider. It proposes to improve efficiency by using passive VM replicas (with very few resources), which become active when a failure is detected. A mechanism is introduced to transfer/initialize the state of VM. In [18], a method of VM placement based on adaptive selection of fault tolerant strategy for cloud applications is proposed. The proposed method consists of two phases. The best evaluation function value of VM placement based on every fault-tolerant strategy is determined in the first phase. In the second phase, the VM placement plan is solved according to the solution of the first phase. Amoon.M [19] propose an adaptive framework for reliable cloud computing environments (AFRCE), it consists of two algorithms: the first for selecting VMs that meets customers needs and can carry out their requests. The second algorithm to select the fault tolerance technique, either replication or checkpoint-restart. In case of replication, AFRCE adjust the number of replicas by calculating the values of VMs. The value of a VM is a function of its failure probability and the profit of its use. If checkpoint-restart is selected, the checkpoint frequency is also adjusted with respect to the failure probability.

The authors in [20], focus on improving the reliability and availability in the IaaS model of cloud computing, they propose a new topology aware VM replica placement approach. The approach places the replicas of virtual machines on the optimal candidate servers according to a calculated weight. The weight is calculated for each PM depending on its distance from the VM and its current temperature. A low value of a PM's weight indicates its appropriateness for hosting a replica of a VM. In [21] Zhou et al. propose an approach to cloud service reliability enhancement via virtual machine placement optimization. The proposed approach is composed of three phases; each one is elaborated by an algorithm. Based on the network topology, the algorithm of the first phase selects a suitable set of VM hosting servers from a potentially large set of candidate host servers. The second algorithm determines an optimal strategy to place the primary and backup VMs on the selected host servers. The last phase concerns the recovery strategy decision; a heuristic is used to address the task-to-VM reassignment optimization problem, which is formulated as finding a maximum weight matching in bipartite graphs. Dailbag et al. [22] present an approach for providing high availability to the requests of cloud's customers based on a failover faults tolerance strategy for cloud computing, using integrated check-pointing algorithms.

Alain Tchana et al [23] propose a fault tolerance method in which both the Cloud customers and providers will collaboratively share their responsibilities in order to provide the required fault tolerance. According to Tchana, application faults can be detected and repaired at the customer level. But the Virtual Machine (VM) and hardware faults can be detected & repaired at the Cloud provider level. The recovery/restoration of the applications running on the refurbished VMs can be requested and performed at the customer level. Checkpointing technique is used to create restore points for the recovered VMs. VFT [24] is a virtualization and fault tolerance technique proposed to reduce the service time and to increase the system availability. It uses a cloud manager (CM) module and a decision maker (DM) to manage the virtualization, load balancing and to handle the faults. In [25], Bashir et al. propose an optimized fault approach in real-time cloud computing IaaS environment where a model is designed to tolerate faults based on the reliability of each compute node (VM) and can be replaced if it's performance is not optimal.

Attallah et al. [26] introduce a new proactive load balancing fault tolerance algorithm to maximize the cloud computing reliability and availability. The proposed algorithm handles the VMs CPU faults by monitoring the CPU utilization. A load balancing operation based on migration strategy is launched in case of high CPU utilization. In [27], an adaptive fault tolerance framework in the cloud computing is developed. The major objective is to minimize application failure rates and completion times. The developed framework combines the proactive and reactive approaches; VM migration and replication are used by the proactive approach to avoid the predicted application's faults, whereas the retry is adopted in the reactive approach to decrease the application execution failure in the cloud.

The works in [28], [29] and [30] deal with the different services level objective described in the SLA. In[28] , proposed spot instance scheme that users can decide a minimum cost according to SLA agreement between users and instances in Amazon's EC2. The scheme is based on a probabilistic model for the optimization of cost, performance and improved the reliability of services by changing dynamically conditions to satisfy user requirements. Yi et al [29] introduced the spot instances of the Amazon EC2 to offer fewer resource costs in exchange for reduced reliability. Based on the actual price history of EC2 spot instances, authors compared several adaptive checkpointing schemes in terms of monetary costs and the improvement of job completion time. In [30] to avoid delays in task completion, a price history based check-pointing scheme based on SLA is proposed. This is achieved by reducing the number of checkpoints and improving the performance of the tasks. Total cost and number of checkpoints are effectively reduced in this scheme. A coordinator component in this scheme supports and manages the SLA between users and instances. The checkpoints are taken in two places. One at the raising edge where the price exceeds the threshold and the other is taken at the failure time foreseen by the average failure time and failure possibility.

A comparison of the related works is presented in Table I .

3. PROPOSED MODEL

In this section, first, we present a generic SLA model for representing customer's requests. Second, we describe the platform model and finally, we present the proposed fault tolerance.

The Table II presents the description of the symbols used.

A. SLA Generic Model

In this paper, we model the customer request based on the SLA contract as the following form:
Req(#Res, Ti, d) such as :

$0 < \#Res <= Max\#Res$

$Ti \ 2 \ ST$,such ST is the set of the resources types provided by the platform.

$0 < d <= MaxD$

The requested resource can be material such as computing nodes in a computing grid, VMs of an IaaS cloud platform, software, web services...etc.

TABLE I. FAULT TOLERANCE APPROACHES COMPARISON

FT model	Policies	Parameters / Metrics	SLA parameters	SLA representation	Service Credit Minimization	Implemented Environment
Self Adaption FT Place [18]	Reactive	Reliability	No	No	No	Cloud simulator
Adaptive framework for reliable cloud computing environments [19]	Reactive	Reliability Availability	No	No	No	Cloud simulator
Topology-aware Virtual Machine Replication FT [20]	Reactive	Reliability Availability	No	No	No	Cloud simulator
Cloud Reliability Enhancement via Virtual Machine Placement [21]	Reactive	Reliability	No	No	No	Cloud simulator
Failover Strategies for Cloud using Integrated Checkpointing [22]	Reactive	Availability	No	No	No	Cloud simulator
A virtualization and FT for cloud computing (VFT) [24]	Reactive	Availability Service Time	No	No	No	Cloud simulator
Optimising FT in Real-time IaaS Environment [25]	Reactive	Reliability	No	No	No	Cloud simulator
Proactive load balancing FT [26]	Proactive	Reliability Availability	No	No	No	Physical Cloud
New Adaptive FT Framework in Cloud [27]	Proactive/ Reactive	Reliability	No	No	No	Cloud simulator
Efficient Checkpointing Scheme Using Price History of Spot Instances [30]	Reactive	Completion Time Monetory Cost	Yes	No	No	Cloud simulator
Proposed FT : FT Directed By SLA in Cloud	Reactive	Availability Performance	Yes	Yes	Yes	Cloud simulator

TABLE II. Symbols Used

Symbols	Description
#Res	The number of resources requested by the customer
Max#Res	The maximum number of available resources
Ti	Resource of type *i*
Ci	Resource of class *i*
#ResCi	The number of the resource from the class *Ci* assigned to the customer
MaxCi	The maximum accepted number of the resource of class *Ci*
MinCi	The minimum accepted number of the resource of class *Ci*
#VMs	The number of requested VMs by the customer.
d	The degradation margin accepted in the requested resources
MaxD	The maximum degradation in the resources permitted by the provider.
T.PV	Time spent by the customer in the level of potential violation.
T.RV	Time spent by the customer in the level of recorded violation.
RT	Potential violation resolution deadline

The resources are distinguished into different type due to their characteristics which can be the hardware's performances such as the frequency of CPU, the memory size, the network bandwidth,...etc, or it could be the type of the SaaS or PaaS service provided by the platforms.

Concerning the degradation, there are two categories. Firstly, the quantity degradation, it's defined by a reduction in the number of resources provided compared to that requested. The second category of degradation, concerns the performances such as the CPU, the memory, the response time of a service, etc. The degradation margin can be specified according to several levels or thresholds.

Based on both the types of resources and the degradation margin specified by the customer, we classify the resources as follows:
$C_1; C_2; C_3; :::C_{m+2}$

C_1: Resources that match the customer's choice. It contains the resources of type Ti.

$C_i(2 <= i <= m + 1)$: Resources with an accepted degradation margin(the degradation $<= d$). In other words, these classes contain the types of resources with an accepted degradation margin. Such that: m is the levels or thresholds number of the degradation margin.

C_{m+2}: Resources with a capacity below the accepted margin. This class corresponds to all resource types which have a capacity below the accepted margin.

Depending on the resource classes provided to the customer according to the parameters specified in the SLA model, the customer can be in two states:

1) *RS*: is the regular state that defines the levels from L_1 to L_k , such as :

   a) L_1 : the highest level of quality, the customer receives all his resources from the requested type (from class C_1) : $\#ResC_1 = \#Res$:

   b) $[L_2; ::::L_k]$: the interval of degraded QoS accepted levels and L_k is the minimum level of accepted QoS : $\#ResC_1 + \#ResC_2 + : : : + \#ResC_{m+1} = \#Res$, such as : $MinCi <= ResCi <= MaxCi$:

2) *IS*: is the irregular state that contains the levels of Potential Violation (*PV*) and Recorded Violation (*RV*), such as:

   a) *PV*: this is the level of the Potential Violation (L_{k+1}). The QoS is below the accepted minimum level. The time spent by the customer in this state must not exceed the resolution time (*RT*).

   b) *RV*: if the potential violation is not resolved within the *RT* delay, the customer ends up in the level L_{k+2} that represents the status of the violation recorded (considered) involving penalties.

The cloud provider seeks to maximize his profit, firstly, by avoiding the recorded SLA violation involving penalties. Secondly, by minimizing the time spent by the customer in irregular states to reduce the rate of penalties. Therefore, for each customer, the objective function is formulated as follow:

Min(T) = $T.PV+T.RV$

Under the constraints:

$$\#Res = \sum_{i=1}^{m+1} ResC_i$$

The number of resources requested must be equal to the total number of resources allocated to the customer from the class C_1 and the classes with an accepted degradation margin.

$$8i\ 2\ [1; m + 1]; MinCi <= ResCi <= MaxCi$$

The minimum and maximum number of resources allowed from each class must be considered when allocating resources to customers.

$$T:PV <= RT$$

In case of potential violation, it must be resolved before the *RT* deadline to avoid the consideration of the violation and consequently revenue losses due to penalties.

B. Platform Model

As shown in the figure 1, the platform model adopted provides *M* different types of resources which are classified in *N* categories; each category is intended for a well-defined use. The customer can thus choose the type of resources that meet his needs.

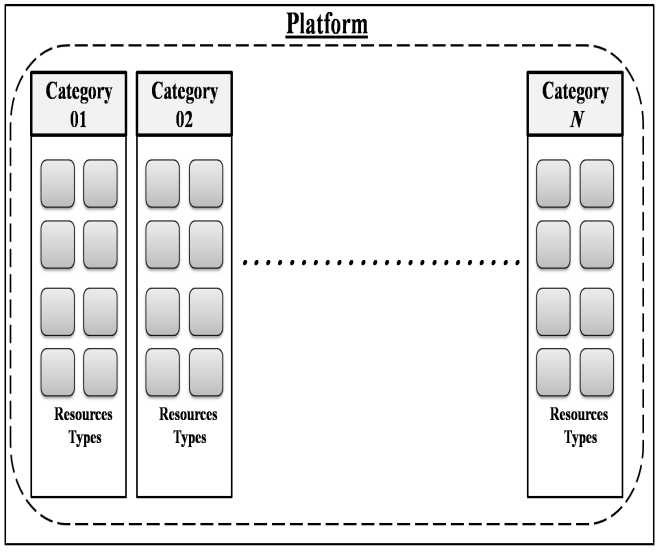


Figure 1. Platform Model

In our work, we focus on Cloud computing IaaS infrastructure that provides a set of VMs to customers. Known Cloud providers such as: Amazon EC2 [31], Micosoft Azure [32] use this model where they propose to their customers different types of instances optimized for different use case. Instance types are different combinations of CPU, memory, storage, and network capacity that give the flexibility to choose the appropriate resources.

C. Fault Tolerance Parameters

In this work, we focus on ensuring the availability and performance stability of the provided instances (VMs):

1) *The Availability*: High Availability (HA) in cloud computing services is some of the hot challenges. The availability of a system at time 't' is referred to as the probability that the system is up and functional correctly at that instance in time [33] [34]. In IaaS cloud computing infrastructure, unavailability can affect a limited number of VMs or the entire system where the service is completely unavailable for a period of time. Most of the cases, the unavailability is caused by physical faults that mainly occur in hardware resources, such as faults in CPUs, in memory, in storage, failure of power etc. [35].

2) *Performance*: the decrease in performance of VMs affects their capacities. Generally it's due to the resource consolidation [36]. For example, it occurs when a VM cannot get the requested amount of MIPS. This can occur when multiple VMs sharing the same host require higher CPU performance that cannot be provided due to VM consolidation [37]. Furthermore, the IaaS clouds platforms to their complex nature are prone to performance anomalies due to various reasons such as resource contentions, software bugs, or hardware failures [35] [38].

Cloud service providers are required to explain how the availability and performance are measured, as well as refunds in case of SLA violations. Amazon EC2 and Microsoft Azure define the downtime as the total of the accumulated minutes in which the service was in a state of unavailability. A service credit is reimbursed to customers according to the monthly uptime percentage [39] [40].

D. Proposed Faults Tolerance

When failures affect a customer's VMs and cause potential violations, fault tolerance operation is initiated to tolerate the occurred faults and resolve the potential violation as quickly as possible before being recorded. The flow chart presented in figure 2, explains the functioning of the fault tolerance. It is held in two phases:

1) *Phase 01 : FT using available free (idle) VMs*

When faults affect the cloud computing platform and cause SLA potential violations, a recovery operation is automatically initiated by restarting the failed VMs. Recovered VMs are added to the list of available VMs. This recovery operation is performed continuously. The available free VMs are VMs that are not assigned to customers and which are in the idle state.

Immediately after launching the VMs recovery,we evaluate for the customer in potential violation (level *PV*), the possibility of fault tolerance for its recovery in a regular state level by using the function ***Check_In_RS()*** (*Algorithm 1*).The function ***Chech_in_RS()*** evaluates the efficiency of this 1st FT phase by checking the possibility of fault tolerance to resolve the potential violation according to a level in the regular state, starting with the higher level L_1 until the last accepted level L_K. It compares the number of failed VMs that can be replaced by available free VMs of different classes C_i taking into consideration the maximum number of VMs allowed in each class (*MaxCi*). The function ***Check_in_RS()*** returns the resolution level index possible *L_RS*, if no resolution is possible for the potential violation by the fault tolerance, it returns the value 0.

If the evaluation of the possibility to recover the customer in a regular state level by the FT proves to be positive, the customer is put in the level *L_RS* returned by the function ***Check_in_RS()*** using the function ***Put_In_RS()*** (*Described in Algorithm 2*). The function ***Put_In_RS()*** replaces the customer's failed VMs by assigning new VMs based on the *L_RS* level. VMs are selected according to three different strategies (*MA : Max Available, HC : Higher Capacity and LC : Lower Capacity*) (*Described in the next part*) from the different classes C_i, such that i is including between 1 and *L_RS* and respecting the *MaxCi* in the allocation of VMs. The content of the customer's failed VMs is restored in the new assigned VMs based on saved image.

2) *Phase 2 : FT using released VMs*

If the violation is not resolved by the first phase of fault tolerance, the second phase is lunched. It uses the VMs released (deallocated) by the degradation of a number of selected customers from the regular state levels.

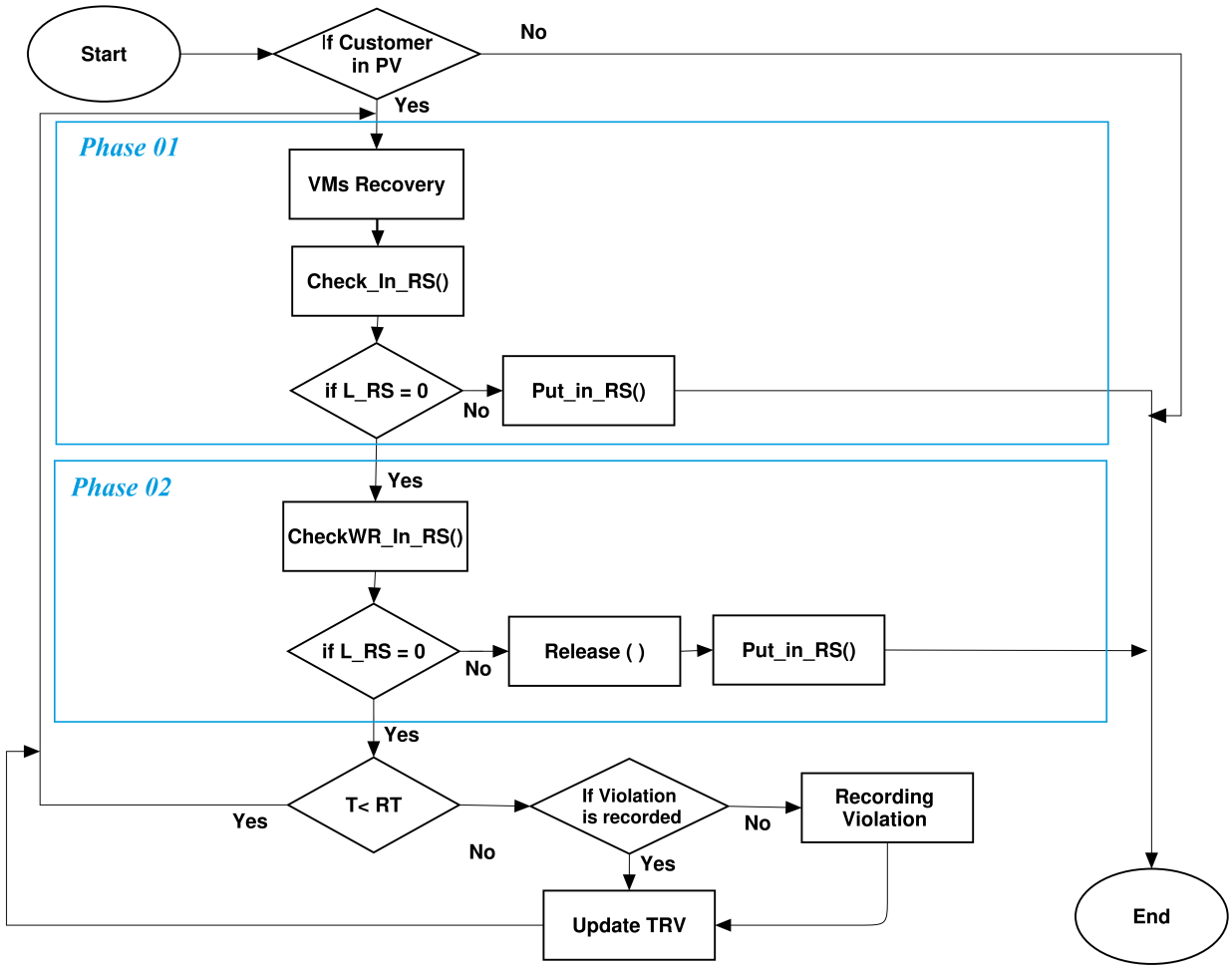


Figure 2. Flow Chart " Fault Tolerance "

Algorithm 1 : Check_In_RS ( )

Data: *Cust_Id*, *AvailableVmsC1List*, *AvailableVmsCkList*, *AssignedVMsList.*
Result: *L_RS*
ValidVMs *AssignedVMsList:nb*; *i* 1 , *L_RS* 0 , *Cont* *true*
while *(i <= K* ***AND*** *Cont = true)* **do**
  if *((ValidVMs + MaxcCi) < #VMs)* **then**
    if *(AvailableVmsCiList:nb <= MaxCi)* **then**
      ValidVMs *ValidVMs + AvailableVMsList:nb*
    else
      ValidVMs *ValidVMs + MaxCi*
    end
  else
    if *(AvailableVMsCiList:nb >= MaxCi* ***OR*** *(ValidVMs + AvailbaleVMsList:nb)>= #VMs)* **then**
      L_RS *i*; *Cont* *false*
      Break
    end
  end
  i *i + 1*
end
return L_RS

Algorithm 2 : Put_In_RS( )

Data: *Cust_Id*, *L_RS*, *AvailableVmsC1List*, *AvailableVmsCKList*, *AssignedVmsC1List*, *AssignedVmsCKList*
Result: *AssignedVmsList*

```
i  1
while (AssignedVmsList:nb < #Vms AND i <= L_RS ) do
    while (AvailableVmsCiList:nb ≠ 0 AND AssignedVmsCiList:nb < MaxCi) do
        SelectVm(AvailableVmsCiList)
        Allocate Vm to Customer Cust_Id
        AssignedVmsList:add(Vm)
        AssignedVmsCiList:add(Vm)
        AvailableVmsCiList:remove(Vm)
        if (AssignedVmsList:nb = #Vms) then
            Break
        end
    end
    i  i + 1
end
return AssignedVmsList
```

Algorithm 3 : CheckWR_In_RS ( )

Data: *Cust_Id*, *AvailableVmsC1List*, *AvailableVmsCKList*, *AssignedVmsC1List*, *AssignedVmsCKList*
Result: *L_RS*

```
L_RS  0 , i  2 , ValidVMs  AssignedVMsList:nb , Cont  true
 while ( i <= k AND Cont = true) do
    if (ValidVMs + MaxCi < #VMs ) then
        if ( (AvailableVmsCiList:nb + PotentialReleasedVMsList:nb) <= MaxCi ) then
            ValidVMs  ValidVMs + AvailableVmsCiList:nb + PotentialReleasedVMsList:nb
        else
            ValidVMs  ValidVMs + MaxCi
        end
    else
        if ((AvailableVmsCiList:nb+PotentialReleasedVMsList:nb)>= MaxCi OR ( ValidVMs+AvailableVmsCiList:nb+
          PotentialReleasedVMsList:nb )>= #VMs) then
            L_RS  i
            Cont  false
            Break
        end
    end
    i  i + 1
end
return L_RS
```

Algorithm 4 : Release( )

Data: *Cust_Id*, *AssignedVMsCiList*, *L_init* , *L_dg*
Result: *AvailableVMsList*

```
i  L_init
 if (Check_In_RS(Cust_Id) = true) then
    while (i >= L_dg) do
        while ( AssignedVMsCiList:nb > MinCi ) do
            AssignedVMsCiList:remove(VM)
            ReleasedVMsList:add(VM)
        end
        i  i  1
    end
    Put_In_RS(Cust_Id)
end
return AvailableVMsList
```

Algorithm 5 : SelectVm( )

Data: *AvailableVMsList*, *Strategy*
Result: *SelectedVm*
if *(Strategy = MA)* **then**
  Select *Vm* of Type *Tj* where *AvailableVmsCiTjList.nb* is Max
    Vm *SelectedVm*;
end
if *(Strategy = HC)* **then**
  Select *Vm* of Type *Tj* where *Tj* is the Higher capacity Available Type Belonging to the class *Ci*
    Vm *SelectedVm*;
end
if *(Strategy = LC)* **then**
  Select *Vm* of Type *Tj* where *Tj* is the Lower capacity Available Type Belonging to the class *Ci*
    Vm *SelectedVm*;
end
return *SelectedVm*

Firstly, we evaluate using the function ***CheckWR_In_RS()*** (*Described in Algorithm 3*) the efficiency to proceed the second phase of fault tolerance to resolve the potential violation. The function ***CheckWR_In_RS()*** checks for the possibility of fault tolerance that leads to a potential violation resolution, starting with the second level of regular state until the last accepted level. It search a possible VMs reallocation by comparing if the available VMs and those which can be released by degrading a set of targeted customers (*PotentialReleasedVMsList)*, makes it possible to replace the failed VMs. The function ***CheckWR_In_RS()*** returns the level of possible resolution *L_RS*.

If the evaluation realized by the function ***CheckDW_In_RS()*** is positive, we proceed to releasing the VMs of the classes searched by the degradation of targeted customers from the regular state levels. The degradation operation is elaborated using the function ***Release()*** (*Described in Algorithm 4*). The function ***Release()*** performs degradations of the targeted customers specified in *CustTargetedList*, one by one from their current levels *L_init* to the new levels of the specified regular state *L_new*. The VMs are deallocated from the customers respecting the minimum assigned number from each class. The customers concerned by the release of VMs, pass temporarily to the potential violation level, waiting the allocation of their new VMs using the function ***Put_In_RS()*** to regain a regular state level. The content of the VMs deallocated from the customers concerned by the degradation are migrated to their new assigned VMs. The released VMs are added to the list of available VMs *AvailableVmsList*.

After the release of the VMs by the function ***Release()***, the customer is put in the level returned by the function ***CheckWR_In_RS()*** using the function ***Put_In_RS()***, which replaces the customer failed VMs by VMs from the list *AvailableVMsList*, which contains the available free VMs, as well as those recently released. Finally, the content of the customer's failed VMs is restored in the new assigned VMs based on saved image.

Throughout this process, the VMs recovery operation is performed in the background and the recovered VMs are added to the available VM list.

In case of the potential SLA violation is not resolved using the second phase and the time *RT* is not expired, the process is resumed from the beginning to avoid recording the violation. If the resolution time has expired, the fault tolerance process is restarted to minimize the time spent by the customer in the state of recorded SLA violation.

To select the type of available VMs that will be assigned to the customer among the types belonging to a class *Ci*, we use the function ***SelectVm( )*** (*Described in Algorithm 5*). This function defines three different strategies for the VMs selection:

Max Available (MA): The VMs selected first are those of the most available type compared to other types in the same class

Higher Capacity (HC): VMs with the higher capacities compared to the others VMs belonging to the same class are selected first.

Lower Capacity (LC): VMs with the lower capacities compared to the others VMs belonging to the same class are selected in first.

4. PERFORMANCE EVALUATION

To verify the effectiveness of our proposed model, we have conducted the following experiments. First, we have evaluated and compared the results of the proposed fault-tolerance by adopting three VMs selection strategies (*MA*, *HC*, and *LC*):

FT-MA: Faults Tolerance technique with the strategy *Max Available VMs*.

FT-HC: Faults Tolerance technique with the strategy *Higher Capacity VMs*.

FT-LC: Faults Tolerance technique with the strategy *Lower Capacity VMs*.

Second, we have compared our model to the following techniques:

No FT: no fault tolerance techniques are used.

AFRCE [19]: The framework is adaptive. It consists of two algorithms. The first for selecting VMs that meet customers' needs and can carry out their requests and the second algorithm for selecting the replication or checkpoint-restart fault tolerance technique. In the case of replication, AFRCE adjusts the number of replicas by calculating the values of VMs. The value of a VM is a function of both its probability of failure and the benefit of its use. If checkpoint-restart is selected, the checkpoint frequency is also adjusted with respect to the failure probability [19].

To compare the techniques proposed with the AFRCE technique, we consider all the VMs with an accepted degradation margin, that they satisfy the customer's needs. The price cost used is the same as that of Amazon EC2 [41].

A. Performance metrics

As an endeavor to evaluate the efficiency of the proposed techniques, we have used these metrics:

The Rate of recorded Violations (RV): this metric is the ratio between the numbers of the recorded SLA violations compared to the full number of the customer's SLA contracts.

$$\%RV = \frac{\#RecordedSLAviolations}{\#SLAcontracts} \quad (1)$$

Our main objective is to maximize the rate of potential violations resolved and therefore, minimizing the number of violations considered involving penalties (*#Recorded Violations*).

Quality of Resolution: this metric concerns the rate of customers in the highest regular state level $\%Cust\_L1$:

$$\%Cust\_L1 = \frac{\#CustomersInL1}{\#AllCustomers} \quad (2)$$

Violation Resolution Time (VRT):It represents the time needed to tolerate the faults to avoid having recorded violations (*RV*). Optimizing this metric minimizes the time spent by the customers in the state of potential violation (*PV*).

$$VRT = \sum_{i=1}^{X} VRTC_i \quad (3)$$

Such as:
$VRTC_i$: violation resolution time for the customer i.
n: number of potential violations resolved.

B. Experiment setup

The platform targeted in our work is a generic Cloud Computing environment (IaaS infrastructure). To approve the performance of our proposed model and study its efficiency, we have resorted to simulation. We have chosen the *CloudSim toolkit* [42] as a simulation platform. It provides a generalized and extensible simulation framework that enables seamless modeling, simulation, and experimentation of emerging Cloud computing infrastructures and application services. An extra package was created to support the fault-tolerance techniques. The created package provides a set of classes that permit simulating the VMs failure, the fault tolerance techniques as well as the detection of the potential violations and recorded violations.

In our experiments, we have generated a cloud of 10000 virtual machines that are connected with fast Ethernet technology (100Mb/s). The Virtual machines are provided by 500 Hosts contained in a set of data centers. The size of each host's RAM range from 16 to 64 GB and the storage is 1TB. The frequency of the host's processors is given in MIPS and is assumed to range from 3000 to 10000 MIPS. The virtual machines are classified into three categories: general use, optimized calculation, and optimized storage. For each category, there are five models: nano, micro, small, medium, and large, which provide in all 15 types of VMs. The number of customer's requests is fixed at 300. Each request expresses the type and the number of VMs required by the customer, as well as the degradation margin accepted. All the experiments are conducted using a random workload.

As an example, *Req(10, T5 :General_Use_Large, 10%)* expresses the request of a customer for 10 VMs of type *T5 (General_Use_Large)* with an accepted degradation of 10%. This also indicates that in case of a failure, the customer's failed VMs can be replaced with VMs of the following types:

T4 : General_Use_Medium.

T14 : Optimized_Storage_Medium.

T15 : Optimized_Storage_Large.

To perform experiments, we have adopted a checkpointing based on the probability of VM failures. This probability is obtained from the VMs failures history. The checkpointing interval is shortened in case of the high probability of VM failures. Otherwise, it is prolonged in case of a low probability. The interval is calculated as

follows:
$INT_i = T(t_j; VM_i)$ (1 $PF(VM_i)$) such as:

INT_i: The checkpointing interval for the VM i.

$T(t_j; VM_i)$:The execution time of the task j on the VM i.

$PF(VM_i)$: Failure probability of the VM i.

The check-pointing files are stored in the available free VMs candidate to replace the failed VMs. This significantly reduces the restoring time. The candidate VMs are selected according to the VMs types accepted by the customer.

C. Experiment Results

1) Impact of degradation margin

To study the impact of the degradation margin *%d*, experiments are carried out with a degradation margin of 10% and then 20%. The rate of available free VMs is set at 10% and that of recovered VMs at 05%. The rate of failed VMs varies from 10% to 50%.

Figure 3 shows the comparison between the three proposed fault-tolerance techniques (*FT-MA, FT-LC, and FT-HC*). The recorded violation rate *%RV* gradually increases with that of failed VMs. The results show that *FT-MA* minimizes the *%RV* with %5 compared to *FT-LC* and *FT-HC*. This is explained by the variation made by *FT-MA* in the selection of the available free VMs. Each time, *FT-MA* selects the most available types of VMs unlike *FT-LC* and *FT-HC*, which select the VMs of lowest (*FT-LC*) or highest (*FT-HC*) capacities. This makes VMs of these types unavailable for direct replacement of failed VMs or by the combination of deallocation and reallocation of VMs between customers to resolve potential violations detected.

In figure 3 (a), we observe that the increase in degradation margin *%d* from 10% to 20% reduces the *%RV* by an average value of 5% for *FT-MA* and 03% for *FT-LC* and *FT-HC*. The increase in the degradation margin *%d* extends the list of VM types accepted by the customer that improves the probability of replacing failed VMs, either directly from the list of available free VMs or by the combinations of deallocation and reallocation so that we can avoid recorded violations.

Concerning the rate of customers in level L_1 *%Cust_L1* presented in figure 3 (b), *FT-MA* provides the best rates comparing with *FT-LC* and *FT-HC*. These rates are improved with a value between 2% and 3% by increasing the degradation margin *%d* to 20%. The improvement is noted for the rates of failed VMs that range from 10% to 30% due to the increase in the deallocation of VMs from the same customers. Therefore, we minimize the number of customers degraded from L_1 to resolve the potential violations.

2) Impact of Available Free VMs

This experiment is carried out with a rate of free VMs of 10% and then 20%. The degradation margin is set at 10% and that of recovered VMs at 05%.

The results presented in figure 4 (a), demonstrate that *FT-MA* provides better *%RV* compared to *FT-LC* and *FT-HC*. The increase in the rate of free VMs allows an average reduction of 07% in *%RV* for *FT-MA* and 05% for *FT-LC* and *FT-HC*. The reduction in *%RV* is explained by the increase in the availability of VMs for direct replacement of failed VMs, as well as by the improvement in the possible number of customers' degradations by the combination of deallocation and reallocation of VMs.

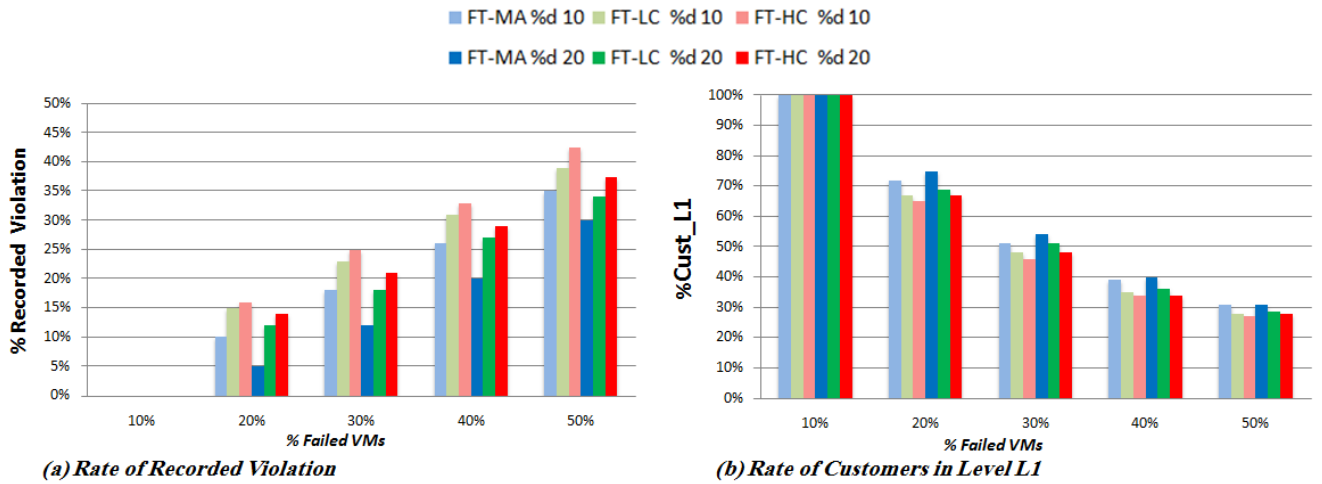
In figure 4 (b), with the increase in the rate of failed VMs, we notice at the beginning of the experiment (rate of failed VMs from 10% to 30%) an improvement in *%Cust_L1* that ranges from 03% to 05%. This improvement is caused by the availability of more free VMs to replace failed VMs without the need to degrade customers from level *L1*. The improvement in *%Cust_L1* decreases as the rate of failed VMs increases in which more customers are degraded from L_1 to allow replacing a large number of failed VMs.

3) Comparison with existing techniques

In the previous experiments, we demonstrate that *FT-MA* provides the best results compared to *FT-LC* and *FT-HC*. In this part, we compare *FT-MA* with existing techniques.

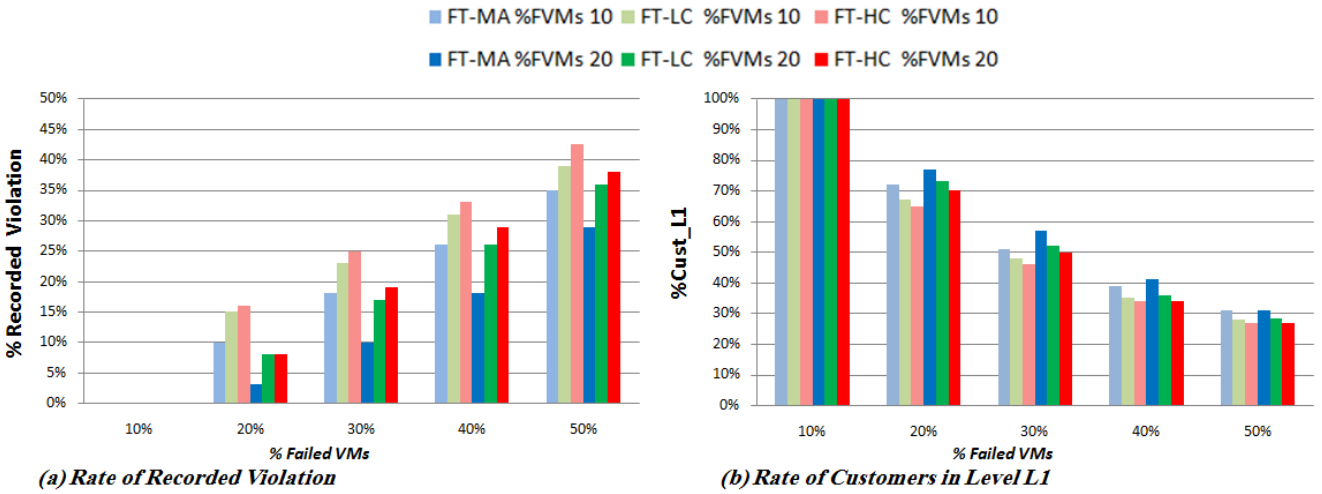
Figure 5 presents the effect of the degradation margin factor in the comparison between the proposed technique *FT-MA* with *AFRCE* and *No-FT*. In this experiment, the rate of available free VMs is set at 10% and that of recovered VMs at 05%. Figure 5 (a) illustrates that *FT-MA* produces the best results that avoid more than half of violations recorded by *No-FT* when no FT mechanism is in place, as well as it reduces by more than 10% the *%RV* obtained compared to *AFRCE*. This is explained by the fact that *AFRCE* uses the available free VMs that satisfies the customers' needs to elaborate replication or check-pointing to the valuables VMs. However, with the increase in the rate of failed VMs, it becomes extremely difficult to find VMs according to the customer's needs. On the other hand, our proposed technique uses the customer's degradation by developing combinations of deallocation and reallocation of VMs between the customers so as to replace the failed VMs with other VMs according to the customer's needs.

The increase of the degradation margin *%d* to 20% minimizes the *%RV* for *FT-MA* by 5% and for *AFRCE* by an average value of 2%. For *FT-MA*, the minimization is due to the expansion of the list of VMs accepted by the customers with other classes of VMs, as well as by the improvement of the possibility of customer's degradations seeking to replace failed VMs. On the other hand, for *AFRCE*, it is only due to the expansion of the list of VMs that satisfied the customer's needs.



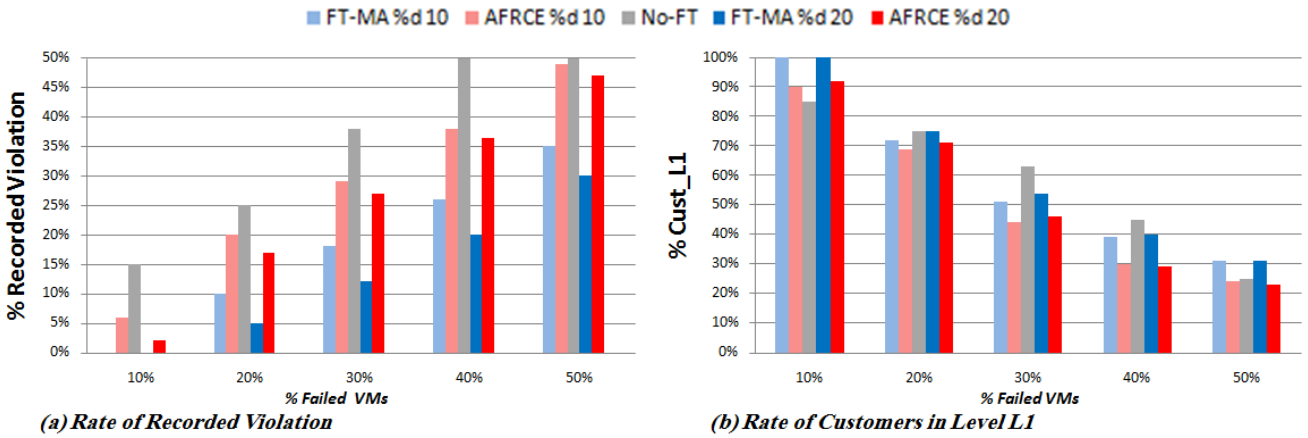
(a) Rate of Recorded Violation *(b) Rate of Customers in Level L1*

Figure 3. Impact of degradation margin



(a) Rate of Recorded Violation *(b) Rate of Customers in Level L1*

Figure 4. Impact of Available free VMs



(a) Rate of Recorded Violation *(b) Rate of Customers in Level L1*

Figure 5. Comparison Results 01

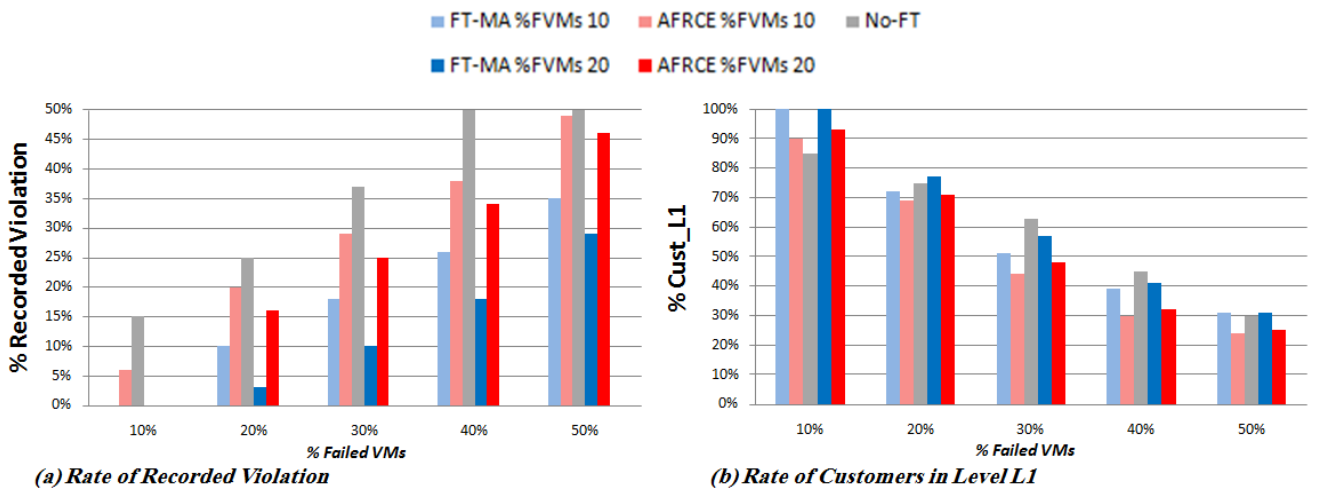


Figure 6. Comparison Results 02

Figure 5 (b) presents the rate of customers in level *L1*. The proposed *FT-MA* technique provides better results compared to *AFRCE*. This is due to the functioning of the *FT-MA* technique which seeks in the first phase of FT to recover the customer in the level *L1* by replacing the failed VMs from class *C1* before moving to other classes of VMs accepted. For *No-FT*, we cannot discuss the results of customers in the level *L1* since the customers are either in the *L1* level or in the recorded violation level.

In figure 6 (a), increasing the rate of available free VMs to 20% enables *AFRCE* to find more VMs that satisfy customer requests. This diminished the *%RV* by 04% compared to the same experimentation with a rate of free VMs of 10%. Our proposed *FT-MA* technique produces minimized *%RV* of 15% on average compared to *AFRCE*.

The rates of customers in the level *L1* presented in Figure 6 (b) demonstrate an improvement for *FT-MA* by a value between 03% and 05% as explained in the section impact of available free VMs. For *AFRCE*, the improvement is between 02% to 03%. It is due to the availability of more VMs of class *C1* in the list of VMs that satisfy the customer's needs.

Figure 7 illustrates the time required to perform the *FT-MA* and *AFRCE* techniques. At the beginning of the experiment, *AFRCE* necessitates more time compared to *FT-MA*. This is explained by the number of replication performed when more than one VMs are available in the list of VMs that meets the customer's needs. On the other hand, *FT-MA*, benefits from the available free VMs to develop the FT by the 1st phase which corresponds to the direct replacing of the failed VMs.

With the increase in the rate of failed VMs, the time required by *FT-MA* exceeds that of *AFRCE*. This is due to the number of potential violations resolved by *FT-MA* which is higher than that of *AFRCE*.

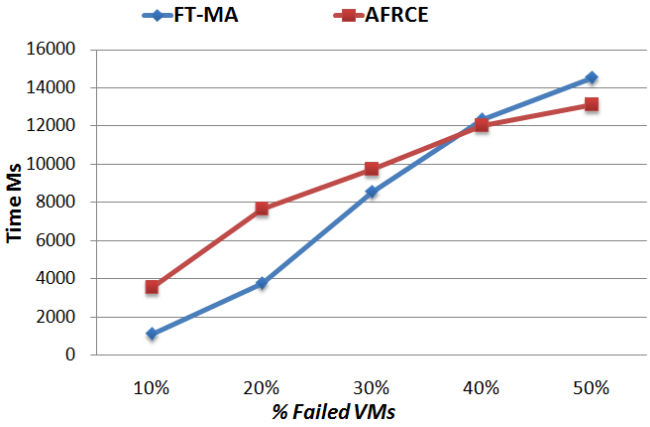


Figure 7. Execution Time

5. CONCLUSION

In this paper, we propose a fault tolerance directed by the SLA contracts in the cloud computing environments. The proposed model provides three fault tolerance techniques: *FT-MA*, *FT-HC* and *FT-LC*. These techniques take into consideration the parameters described in a generic SLA model to ensure availability and performance that lead to avoid SLA violation penalties. To verify the effectiveness of our proposed fault tolerance, experiments were conducted using *CloudSim*. The three FT techniques were evaluated using three metrics: rate of considered SLA violations, rate of customer in the highest level and time. The *FT-MA* technique provides performance superiority compared to *FT-HC*, *FT-LC* and to related work in terms of considered SLA violations and rate of customers in the highest level.

In the future work, heuristic algorithms can be incorporated to improve the VMs assignment, and we will endeavor to consider more SLA parameters.

References

[1] P. M. Mell and T. Grance, "Sp 800-145. the nist definition of cloud computing recommendations of the national institute of standards and technology." Gaithersburg, MD, USA, Tech. Rep., 2011.

[2] T. Labidi, A. Mtibaa, and H. Brabra, "CSLAOnto: A comprehensive ontological SLA model in cloud computing," vol. 5, no. 3, pp. 179–193. [Online]. Available: https://doi.org/10.1007/s13740-016-0070-7

[3] M. Nazari Cheraghlou, A. Khadem-Zadeh, and M. Haghparast, "A survey of fault tolerance architecture in cloud computing," *J. Netw. Comput. Appl.*, vol. 61, no. C, p. 81–92, feb 2016. [Online]. Available: https://doi.org/10.1016/j.jnca.2015.10.004

[4] Z. Amin, H. Singh, and N. Sethi, "Review on fault tolerance techniques in cloud computing," *International Journal of Computer Applications*, vol. 116, no. 18, pp. 11–17, April 2015.

[5] J. A. Jamilson Dantas, Rubens Matos and P. Maciel, "Eucalyptus-based private clouds: availability modeling and comparison to the cost of a public cloud," *Computing*, vol. 97, p. 1121–1140, 2015.

[6] S. Son, G. Jung, and S. C. Jun, "An SLA-based cloud computing that facilitates resource allocation in the distributed data centers of a cloud provider," vol. 64, no. 2, pp. 606–637. [Online]. Available: https://doi.org/10.1007/s11227-012-0861-z

[7] E. Dubrova *et al.*, "Fault tolerant design: An introduction," *Department of Microelectronics and Information Technology, Royal Institute of Technology, Stockholm, Sweden*, pp. 22–3, 2008.

[8] S. M. Abdulhamid, M. S. Abd Latiff, S. H. H. Madni, and M. Abdullahi, "Fault tolerance aware scheduling technique for cloud computing environment using dynamic clustering algorithm," vol. 29, no. 1, pp. 279–293. [Online]. Available: https://doi.org/10.1007/s00521-016-2448-8

[9] S. Setaouti, D. A. Bensaber, R. Adjoudj, and M. Rebbah, "Fault tolerance model based on service delivery quality levels in cloud computing," in *2017 International Conference on Mathematics and Information Technology (ICMIT)*, 2017, pp. 84–91.

[10] T. W. Y. R. Wieder P, Butler Joe M, "Service level agreements for cloud computing." *Springer, New York. Library of Congress Control Number:2011939783, ISBN 978-1-4614-1613-5, e-ISBN 978-1-4614-1614-2.*, 2011. [Online]. Available: https://doi.org/10.1007/978-1-4614-1614-2

[11] A. Keller and H. Ludwig, "The WSLA framework: Specifying and monitoring service level agreements for web services," vol. 11, no. 1, pp. 57–81. [Online]. Available: https://doi.org/10.1023/A:1022445108617

[12] J. S. D.D. Lamanna and W. Emmerich., "Slang a language for defining service level agreements." *The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems, pages 100–106, 20O3.*, 2003.

[13] V. Tosic, B. Pagurek, K. Patel, B. Esfandiari, and W. Ma, "Management applications of the web service offerings language (wsol)," *Inf. Syst.*, vol. 30, no. 7, p. 564–586, nov 2005.

[14] M. Mohamed, O. Anya, S. Tata, N. Mandagere, N. Baracaldo, and H. Ludwig, "rsla: An approach for managing service level agreements in cloud environments," *International Journal of Cooperative Information Systems*, vol. 26, no. 02, p. 1742003, 2017. [Online]. Available: https://doi.org/10.1142/S0218843017420035

[15] Y. Kouki. and T. Ledoux., "Csla: A language for improving cloud sla management," in *Proceedings of the 2nd International Conference on Cloud Computing and Services Science - CLOSER,*, INSTICC. SciTePress, 2012, pp. 586–591.

[16] M. Torkashvan and H. Haghighi, "Cslam: A framework for cloud service level agreement management based on wsla," in *6th International Symposium on Telecommunications (IST)*, 2012, pp. 577–585.

[17] D. L. Uesheng Tan and J. Wang., "Cc-vit: Virtualization intrusion tolerance based on cloud computing," *in 2nd International Conference on Information Engineering and Computer Science (ICIECS)*, pp. 1–6, Wuhan, China, 2010.

[18] X. Chen and J.-H. Jiang, "A method of virtual machine placement for fault-tolerant cloud applications," *Intelligent Automation & Soft Computing*, vol. 22, no. 4, pp. 587–597, 2016. [Online]. Available: https://doi.org/10.1080/10798587.2016.1152775

[19] M. Amoon, "Adaptive framework for reliable cloud computing environment," *IEEE Access*, vol. 4, pp. 9469–9478, 2016. [Online]. Available: https://doi.org/10.1109/ACCESS.2016.2623633

[20] P. Kumari and P. Kaur, "Topology-aware virtual machine replication for fault tolerance in cloud computing systems," p. 193 – 206, 1 Jan. 2020.

[21] A. Zhou, S. Wang, B. Cheng, Z. Zheng, F. Yang, R. N. Chang, M. R. Lyu, and R. Buyya, "Cloud service reliability enhancement via virtual machine placement optimization," *IEEE Transactions on Services Computing*, vol. 10, no. 6, pp. 902–913, 2017. [Online]. Available: https://doi.org/10.1109/TSC.2016.2519898

[22] D. Singh, J. Singh, and A. Chhabra, "High availability of clouds: Failover strategies for cloud computing using integrated checkpointing algorithms," in *2012 International Conference on Communication Systems and Network Technologies*, 2012, pp. 698–703.

[23] A. Tchana, L. Broto, and D. Hagimont, "Approaches to cloud computing fault tolerance," *2012 International Conference on Computer, Information and Telecommunication Systems (CITS)*, pp. 1–6, 2012.

[24] P. Das and P. M. Khilar, "Vft: A virtualization and fault tolerance approach for cloud computing," in *2013 IEEE Conference on Information Communication Technologies*, 2013, pp. 473–478.

[25] B. Mohammed, M. Kiran, I.-U. Awan, and K. M. Maiyama, "Optimising fault tolerance in real-time cloud computing iaas environment," in *2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud)*, 2016, pp. 363–370.

[26] S. M. Attallah, M. B. Fayek, S. M. Nassar, and E. E. Hemayed, "Proactive load balancing fault tolerance algorithm in cloud computing," *Concurrency and Computation: Practice and Experience*, vol. 33, no. 10, p. e6172, 2021, e6172 CPE-19-1199. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.6172

[27] A. Rawat, R. Sushil, A. Agarwal, A. Sikander, and R. S. Bhadoria, "A new adaptive fault tolerant framework in the cloud," *IETE Journal of Research*, pp. 1–13, 2021. [Online]. Available: https://doi.org/10.1080/03772063.2021.1907231