# Autonomous Management of Virtual Machine Failures in IaaS Using Fault Tree Analysis

Alexandru Butoi, Alexandru Stan, and Gheorghe Cosmin Silaghi

Business Information Systems Department,
Babeş-Bolyai University, Cluj-Napoca, Romania
alex.butoi,alexandru.stan,gheorghe.silaghi@econ.ubbcluj.ro

**Abstract.** Cloud IaaS services bring the novelty of elastic delivery of computational resources in a virtualized form and resource management through easy replication of virtual nodes and live migration. In such dynamic and volatile environments where resources are virtualized, availability and reliability are mandatory for assuring an accepted quality of service for end users. In this context specific fault tolerance strategies are needed. Using concepts from fault tree analysis, we propose a distributed and autonomous approach where each virtualized node can assess and predict its own health state. In our setup each node can proactively take a decision about accepting future jobs, delegate jobs to own replicated instances or start a live migration process. We practically evaluate our model using real Xen log traces.

**Keywords:** VM Fault Tolerance, Fault Trees, Fault Agent, Xen Log Traces.

## 1 Introduction

Cloud computing is continuously changing the way we do computation in any domain: business or science. It comprises the idea of delivering computation as a service in a pay-per-use manner through virtualization technology. In the Infrastructure-as-a-Service (IaaS) model, resources are virtualized and delivered as "a service" using Service Level Agreements (SLA) which guarantee certain levels of quality of service (QoS) for end users. Within IaaS, service reliability and availability are of a great importance for a successful service delivery and these objectives can be achieved through complex distributed fault tolerant design. Virtualization brings elasticity in resource provisioning of the cloud, with the meaning that if end users request additional computing resources, they can be provisioned on-demand, transparent to the user.

In general, cloud providers pay a cost for supplying the above mentioned service characteristics: reliability, availability and elasticity. Thus, fault tolerant models that can mitigate provider costs with high levels of QoS are of great interest.

In this paper we develop and evaluate a fault tolerance model for achieving the above-mentioned characteristics within the IaaS cloud model. We use machine replication and live migration as basic tools and present a fault tolerant design to pro-actively decide when replication and live migration processes should be initiated. With the help of fault tree analysis, we are able to predict the health state of a virtual machine and based on this, to keep high levels of reliability and availability.

The paper is organized as follows. Section 2 describes the problem under study and presents some basic concepts from Fault Tree Analysis. Section 3 details our fault tolerance model, section 4 disseminates the experimental results of a primary implementation of the model. Section 5 reviews several important results in the field, and conclusion ends the paper.

## 2 Background

In this section we describe and formally present the problem under study and some insights about the modeling tool used in this respect.

### 2.1 Problem Specification

We approach a data center where computing power is delivered under the paradigm of "Infrastructure as a Service". IaaS is achieved with the help of a virtualization technique like Xen. Each deployed virtual machine (VM) requires a certain amount of resources, like CPU, RAM, bandwidth and storage, taken from the global pool of the data center. When a VM is instantiated, a service level agreement (SLA) is created for that VM, indicating the amount of resources allocated for the VM. Total available resources of the data center diminishes with every instantiated VM. For this paper, we will work only with the four types of resources enumerated above, without restricting the generality of our approach to other types of resources.

For a successful IaaS service delivery, two characteristics are mandatory to be accomplished: availability and reliability. Service availability is measured with the help of the accepted failure rate of the VM, denoted by $natural_Q$, expressed as the time the machine is not available for usage out of the total time the machine is allocated on the data center. Required service availability is included in the SLA created for a new VM.

A common practice for assuring availability and fault tolerance of the services is replication, where a VM is accompanied by a synchronized VM in a kept similar state. Replication in virtual environments is approached by a large number of authors using techniques like asynchronous checkpoint-recovery replication [5,4] or other more sophisticated techniques based on memory content similarity [7]. In our approach we will assume that each instantiated VM is accompanied by a replica, kept synchronous with the parent using one of the techniques enumerated above. The replica runs in parallel the same processes as the cloned VM. When the VM becomes unavailable, the replica should assure the continuity of the service. Of course that replication imposes extra resource usage generating extra costs for the data center owner.

When the replica fails too, there is a high probability to have a service outage, diminishing the availability figure of the IaaS service. In this case, we can use the elasticity characteristic of the cloud, migrating the machine to a healthy one, using the live migration process [17,1,12]. Live migration incurs allocating a new VM only on-demand basis, and transferring all processed from the unhealthy VM to this new VM; all the migration process being transparent to the cloud user. Live migration consumes the available resources in the data center and incurs additional costs for the IaaS provider.

Live migration concurs with other VM creation requests on the data center, and we will model this to obtain the failure probability of the live migration process.

We assume that each VM, during its lifetime, generates events that can be used for VM failure prediction.

To keep service availability at high rates and make the data center resource management operations transparent to the cloud user, including replication and live migration, in this paper we develop a fault tolerance model able to monitor the health state of the running machines and to predict the failure of a VM. Based on this model, we can construct a decision strategy about when to pro-actively transfer the control from a VM to its replica or when to apply the live migration. For our modeling, we will use the fault trees analysis, presented in the next subsection. Having in mind not to overload the virtualization hypervisor or to require other centralized mechanism for running the fault tolerance model, we apply fault tree analysis in decentralized way, such us that each machine to be able to autonomously run the fault tolerance mechanisms.

### 2.2 Fault Tree Analysis

Fault tree analysis was introduced by the U.S. Nuclear Regulatory Commission as the main instrument used in their reactor safety studies. Fault-tree models are part of an analytical technique studying specified undesired states of systems and their systems [9].

Basic fault tree analysis uses graphical tree representation of failure nodes of the system connected together by AND / OR gates. Each node is the equivalent of a subsystem and is characterized by the estimated probability of failure of the represented system or subsystem. Each node of the system can be further decomposed using a fault tree.

In our study we will use the analysis procedure of parallel subsystems - the AND gate, depicted in figure 1.

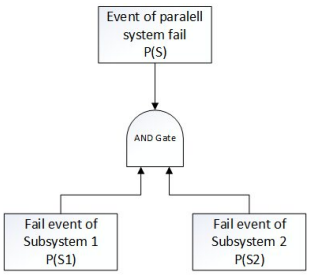



Fig. 1: Fault tree representations for parallel two-component systems[9]

In our case of a parallel system, the whole system is considered crashed when all subsystems that work in parallel are crashed. Having the fault probability for each subsystem $P(S_1)$ and $P(S_2)$, we can compute the failure probability of general fault of the parallel system: $P(S) = P(S_1 \cdot S_2) = P(S_1) \cdot P(S_2)$.

In terms of reliability ($Q$) and unreliability ($R$) at a time $t$ we have [9]: $Q(t) = \prod_{i=1}^{n} Q_i(t)$ and $R(t) = 1 - Q(t) \Rightarrow R(t) = 1 - \prod_{i=1}^{n} Q_i(t)$, where $Q_i$ represents the reliability of the $i$th subsystem.

In the context of virtualized environments, we will evaluate the health state of a virtual machine based on the events triggered in the environment using fault trees. Crash imminence will be assessed based on the health state of a VM and its replicas.

## 3 Approach

In this section we present our approach for fault tolerance analysis, based on the above-mentioned fault trees, using concept of fault agents, described below.

### 3.1 Fault Agents

As presented in section 2.1, a VM can have two types of relations with other VMs running in the datacenter, defined as follows: *replication* - between a virtual machine and its clone and *migration* - between a virtual machine and its clone obtained by live migration.

We equip each VM with a fault agent, capable of building the fault tree for the VM according to the relations between that VM and others (replicas or migrated ones). In order to have a bottom-up approach of the problem, the fault agent conceptually resides in the virtual machine and take as input events raised by the hypervisor in the relation with that VM.

The fault agent implements fault tree analysis principles to process the events delivered by the hypervisor and to predict the reliability of the associated virtual machine. The fault agent acts proactively on the behalf of the VM, taking the decision of transferring the control to its replica or to start the live migration process. Having a virtual machine $v_k$ and its replica $r_k$, the fault agent computes the fault tree described in Fig. 2.

The fault tree presented in Fig. 2 has two types of fault nodes:

(1) the computation nodes are fault nodes which correspond to a running or possible running (live migrated) virtual machines. $N_{v_k}$ represents the fault node of the virtual machine $v_k$, $N_{r_k}$ represents the fault node of its replica $r_k$, and $M_{r_k}$ is the fault node of the on-demand future migrated virtual machine.
(2) the aggregation nodes or the decision nodes do not have real corresponding virtual machines, but they aggregate the fault probabilities of the subsystems and are important for the fault agent to take decisions. $R_k$ represents the fault node of the subsystem resulted from the replication relation of $v_k$ with $r_k$ and it is computed from $N_{v_k}$ and $N_{r_k}$. $S$ represents the fail node of the whole system obtained out of the replication and live migration relations; it is computed from $R_k$ and $M_{r_k}$.

$R_k$ nodes are of great importance, as the fault agent residing on replicas will decide when a live migration has to start, assessing when the replication process is unreliable.

Each node in the fault tree stores two failure probabilities at time $t$: $tq_i$ is the theoretical fault probability of the node and $cq_i$ is the computed fault probability of a node at time instance $t_i$. For each VM, the theoretical fault probability $tq$ is initialized with the

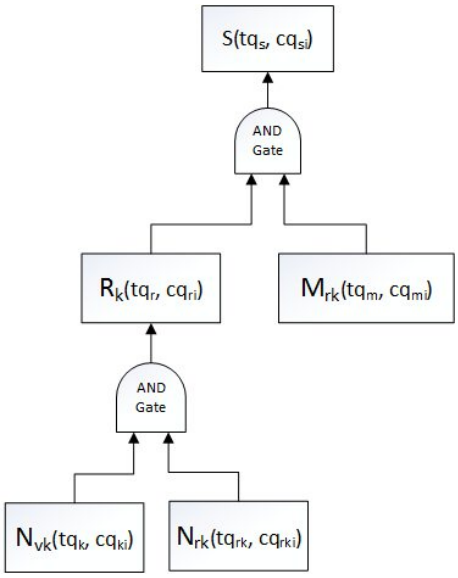



Fig. 2: Fault tree model of one replicated virtual machine

value of the maximum accepted fault rate, according to user-accepted QoS of that VM. Computed $cq$ indicators of the computation nodes are initialized with 0 and computed from the events raised by the hypervisor in relation with the VM. Composite $tq$ and $cq$ indicators of aggregation nodes are calculated in the fault tree according to AND gate rules. Below, we fully specify the life cycle of each fault agent.

The fault agent life cycle has two phases:

*1. The fault tree initialization phase* takes place once, when the VM is started and the fault agent is deployed. The fault tree is constructed using the model presented above. In this phase the $tq$ indicators are computed as follows:

1. $tq_k$ of $N_{v_k}$, $tq_{r_k}$ of $N_{r_k}$ and $tq_m$ of $M_{r_k}$ are initialized with the values of the natural fault probabilities of the virtual machines ($natural_Q$); these values can be found in SLA service availability specification.
2. $tq_r$ of $R_k$ is computed as following: $tq_r = tq_k \cdot tq_{r_k}$ (AND gate output)
3. $tq_s$ of $S$ is computed as following: $tq_s = tq_r \cdot tq_m$ (AND gate output)

*2. The reasoning phase*: the fault agent captures events raised by the VMs or by the hypervisor and updates the $cq$ indicators with the fault probability induced by each event. Each event indicates either the well-being of the system - describing the successful completion of a VM process or it is an undesired event and brings in a fault probability.

The computed probability of failure $cq$ of each node of the tree is updated and recalculated every time an event is raised by the VM. In this phase the $cq$ indicators are computed as presented in the next two subsections.

**Computation of *cq* for the VM nodes and replicas**

1. If at time $t_i$ an error event $E_i(p)$ is raised by $v_k$, inducing a probability $p$ for the machine to crash:
   (a) update the $cq$ indicator of the computation node $N_{v_k}$: $cq_{k_i} = cq_{k_{i-1}} + p$
   (b) recompute the $cq_{r_i}$ of the replication decision node $R_k$: $cq_{r_i} = cq_{k_i} \cdot cq_{r_{k_i}}$
   (c) recompute the $cq_s$ of the aggregation system node $S$: $cq_{s_i} = cq_{r_i} \cdot cq_{m_i}$
   (d) announce all the agents of other VMs which are in direct relation with $v_k$ to update their fault tree with the new values
2. If at time $t_i$ a non-error event $NE_i(p)$ is raised by $v_k$, where $p$ is the probability that this event not to take place - thus leaving space for an error event:
   (a) update $cq$ indicator of the corresponding fault node: $cq_{k_i} = max(0, cq_{k_{i-1}} - p)$
   (b) recompute $cq_{r_i}$ and $cq_s$ as explained at 1.
   (c) announce all the agents of other VMs which are in direct relation with $v_k$ to update their fault tree with the new values

Strictly positive $cq$ indicates the unhealthy state of a VM. When a non-error event is produced, the $cq$ is decreased, and after several non-error events, we consider that the virtual machine heals from an unreliable state to healthy one.

When a virtual machine raises an event and the corresponding fault node is updated, the agent announces other agents deployed in other machines which are in direct relation with the current one: for example if $v_k$ updates its corresponding node, the replica agent will be announced to update the corresponding node too in her internal fault tree.

**Computation of *cq* indicator for the migration fault node $M_{r_k}$.** While the $N_{v_k}$, $N_{r_k}$ and $R_k$ nodes are handled based on fault probabilities associated with each raised event originating in the corresponding VMs, the fault probability $cq_{m_i}$ of the live migration node $M_{r_k}$ is computed according to the chance of the VM to migrate at a time $t_i$, being aware that the available resources to be provisioned during the live migration process are limited and there can be other agents simultaneous claiming those resources at time $t_i$. Thus, $cq_{m_i}$ represents the probability of the VM not to be able to migrate at time $t_i$ and is computed as described bellow.

Virtual machine $v_k$ can migrate only if it has enough resources for all four types required. Given that $p_{storagefail}(t_i)$, $p_{cpufail}(t_i)$, $p_{ramfail}(t_i)$, $p_{bandfail}(t_i)$ represent the failure probabilities due to lack of a given type of resources, the probability of $v_k$ not being able to migrate due to insufficient resources (migration fault probability) at time $t_i$ is:

$$p_{v_k}(t_i) = p_{storagefail}(t_i) \cdot p_{cpufail}(t_i) \cdot p_{ramfail}(t_i) \cdot p_{bandfail}(t_i). \quad (1)$$

Now, we need to compute the values $p(r)_{fail}$ for each of the above mentioned resources: storage, cpu, ram, bandwidth, where $r$ is the requested stock of each resource needed in the live migration process.

For each resource type required by $v_k$, we compute the probability of not having enough resources at time $t_i$. We assume that at time $t_i$ there are $TC$ concurrent requests for the total $RS(t_i)$ stock of resource available at the data center at time $t_i$.

If $NCMALR$ represents the number of claims requesting at least the stock $r$ of the resource and $NCMLR$ represents the number of claims requesting less than the stock $r$ of resources, it gives that the probability of a successful migration requesting $r$ resources $p(r)_{func}$ is

$$p(r)_{func} = \frac{\text{NCMLR}}{\text{TC}} \tag{2}$$

and the probability of not being able to complete live migration because of the lack of resources $p(r)_{fail}$ is

$$p(r)_{fail} = 1 - p(r)_{func} = \frac{\text{NCMALR}}{\text{TC}} \tag{3}$$

A resource claim is a $j$-subset of $RS(t_i)$. Total possible concurrent requests $TC$ is equal with the number of $j$-subsets on $\mid RS(t_i) \mid$ elements and is therefore given by the binomial coefficient $\mathbf{C}^{j}_{|RS(t_i)|}$. Thus, the total number of possible resource claims is equal to:

$$\begin{aligned} TC &= \sum_{j=1}^{|RS(t_i)|} \mathbf{C}^{j}_{|RS(t_i)|} = \sum_{j=0}^{|RS(t_i)|} \mathbf{C}^{k}_{|RS(t_i)|} - \mathbf{C}^{0}_{|RS(t_i)|} \\ &= (1+1)^{|RS(t)|} - \mathbf{C}^{0}_{|RS(t_i)|} = 2^{|RS(t_i)|} - 1 \end{aligned} \tag{4}$$

The total number of claims $NCMLR$ requesting less than $r$ resources can be found using dynamic programming, solving the subset sum problem for a maximum $r$ resources, using the recurrence below:

$$\begin{aligned} &NCMLR(RS(t_i), \mid RS(t_i) \mid, r) = \\ &\qquad NCMLR(RS(t_i) \setminus \{x_{|RS(t_i)|}\}, \mid RS(t_i) \mid -1, r) + \\ &\qquad NCMLR(RS(t_i) \setminus \{x_{|RS(t_i)|}\}, \mid RS(t_i) \mid -1, r - x_{|RS(t_i)|}) \end{aligned} \tag{5}$$

With this approach, we can compute the four failure probabilities $p_{storagefail}(t_i)$, $p_{cpufail}(t_i)$, $p_{ramfail}(t_i)$, $p_{bandfail}(t_i)$.

The above heuristic computes only the probability of a VM not being able to migrate due to insufficient amount of required resources at time $t$. There can be situations when there are enough available resources to migrate but the live migration fails due to specific migration error or crashes. For the completeness of the approach we need to add another probability component $p_{residual}$ representing the probability of live migration process fault due to causes other than lack of resources. The probability of $v_k$ not being able to migrate is the probability of not being able to migrate because of not enough resources at time t and the live migration residual probability:

$$cq_{m_i}(t_i) = p_{storagefail}(t_i) \cdot p_{cpufail}(t_i) \cdot_{ramfail}(t_i) \cdot p_{bandfail}(t_i) \cdot p_{residual}(t_i) \tag{6}$$

### 3.2 Quantifying the Error Impact on Each Node

In our modeling, the pernicious effect of errors decays over time. Thus, a non-error event or events occurring after an error event indicates that the VM recovered somehow from the crash and the probability of failure decreases.

We considered three types of temporal error impact on the system: (1) punctual impact; (2) stochastic linear decay impact; (3) stochastic exponential decay impact. Below, we present the mathematical modeling for these sorts of decays.

**Punctual Impact** Punctual impact means that a non-error event following after an error event totally heals the machine. Mathematically, decay intervenes as a Heaviside step function $E(t_{err}) \cdot (1 - H[t_{err} + 1])$ and full decay instantaneously occurs at time $t_{err} + 1$. The differential equation of the process is $\frac{dE}{dt} = E(t_{err}) \cdot (1 - \delta(t_{err} + 1))$. Fig. 3 shows this sort of decay function.

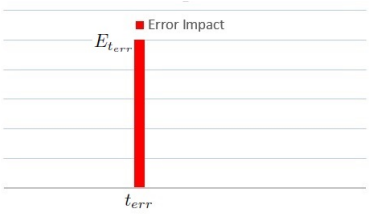



Fig. 3: Punctual error impact on the system

**Stochastic Linear Decay Impact** The linear model uses a constant rate of decay, and is the most simple decay function. It has the specific meaning of error impact decreasing in arithmetical progression. The differential equation of the process is $\frac{dE}{dt} = -\lambda_d$ paired with the initial condition $E(t_{err}) = E_{t_{err}}$, where $\lambda_d$ is the linear decay rate. Full decay intervenes after time $t_{err} + \frac{E_{t_{err}}}{\lambda_d}$. Fig. 4 shows this type of decaying impact.


Fig. 4: Linearly decaying error impact on the system

We used a stochastic alternative of this equation of the following form: $\frac{dE}{dt} = -\lambda_d + \sigma \cdot dw(t)$, where $\sigma$ is the volatility around the linear decaying trend and $w(t)$ is a Wiener noise. The stochastic decaying process encounters an absorbing barrier at $E = 0$.

**Stochastic Exponential Decay Impact** The error impact is subject to exponential decay which decreases at a rate proportional to the initial impact. The process can be expressed by the differential equation $\frac{dE}{dt} = -\lambda_d E$, where $E(t)$ is the error impact at time $t$ and $\lambda_d$ is the exponential decay constant.

The solution to this equation is an exponential rate of change $E(t) = E_{t_{err}} e^{-\lambda_d t}$ $t \geq t_{err}$ where $E_{t_{err}} = E(t_{err})$ is the initial error impact on the system at time $t = t_{err}$.

In this case, the error occurrence in the system will induce an infinite impulse response with an impact decreasing exponentially, but never reaching zero and having half-live original impact at time $t_{1/2} = t_{err} + \frac{\ln(2)}{\lambda_d}$. Fig. 5 illustrates an example of this type of decaying impact.

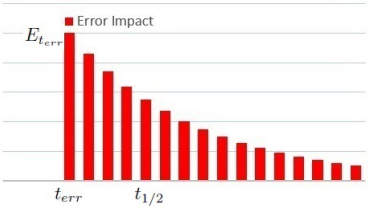



Fig. 5: Exponentially decaying error impact on the system

We used a stochastic derivation of the exponential decay of the following form: $\frac{dE}{dt} = -\lambda_d \cdot E(t) + \sigma \cdot dw(t)$. The stochastic decaying process encounters an absorbing barrier at $E = 0$.

We used one of the three types of decay functions presented above for computing the update healing probability for each error event occurring at one computation node followed by non-error events (see computation of $cq$ for VM and replica nodes in subsection 3.1).

### 3.3 Proactive Decision Making Algorithm

The agent takes a decision to relocate the running processes from the VM to the replica or to start live migration by comparing $cq$ with $tq$ using a threshold.

Given a threshold $0 < f < 1$, the decision process at time $t_i$ for transferring the control to replica or to migrate takes place as follows:

1. for node $N_{v_k}$ , if $\frac{cq_{k_i}}{tq_k} \geq f$ then the corresponding virtual machine $v_k$ is unreliable : transfer the control to its replica, $r_k$, only if the replica is reliable: at node $N_{r_k}$, $\frac{cq_{r_{k_i}}}{tq_{r_k}} < f$

2. for the replication composite node $R_k$, if $\frac{cq_{r_i}}{tq_r} >= f$, then both $v_k$ and $r_k$ became unreliable : try to migrate the healthiest machine only if the migration process is reliable: node $M_{r_k} : \frac{cq_{m_i}}{tq_m} < f$
3. for the root node $S$, if $\frac{cq_{k_i}}{tq_k} \geq f$ then the system is in an unreliable state

The possible decisions taken by the fault agent are:

1. REPLICATED - the control has to be transferred to replica- taken upon node $R_k$.
2. MIGRATED - the VM and its replica are unreliable and the migration is needed - taken upon node $R_k$.
3. OK-when a computation node is in a reliable state
4. REPLICATED to OK / MIGRATED to OK - when a VM is becoming reliable again and can be reversed from its previous state (replicated/migrated) to its normal state.
5. REPLICATION FAIL / MIGRATION FAIL - when the replication or migration process is unreliable too.
6. UNRELIABLE - a computation node is declared unreliable

## 4 Experimental Results

Our aim was to evaluate the strategy of migration as a secondary plan for QoS assurance, when the VM and it's replica are in a fault state. We study the evolution of computed fault probability of $R_k$ node which is used to take the decision of migration.

We simulated a virtual machine together with its replica requiring 2 CPUs, 2GB of RAM, 100MB/S of bandwidth and 10 GB of storage, and a maximum accepted failure rate of 0.1% with a threshold $f = \frac{cq_{ki}}{tq_k} = 0.8(80\%)$, varying the number of errors raised by the VM. Using traces generated from real Xen virtualization logs, we studied the behavior of the fault agent for the cases when the migration is needed in the above mentioned error impact scenarios: punctual error impact, linear decay impact and exponential decay impact. The XEN hypervisor was hosting similar VM configurations, having a medium load/hour. It produces three types of log records labeled as: ERROR, DEBUG and INFO. For our experimental setup, we used only the ERROR and INFO labeled log records while the DEBUG records targets the development process of XEN engine.

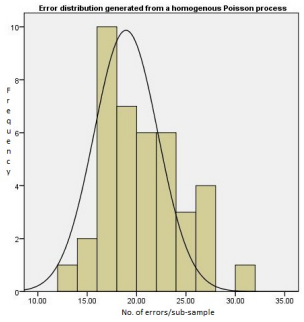
### 4.1 Error Samples Generated from Traces

The event samples for simulation runs were obtained by parsing each record of the XEN log file obtaining a specific event in the initial sample of VM events. If the log record was labeled as an ERROR, an event error was generated, while another was labeled as INFO a non-error event was generated. For each new created event a simple probability was computed:

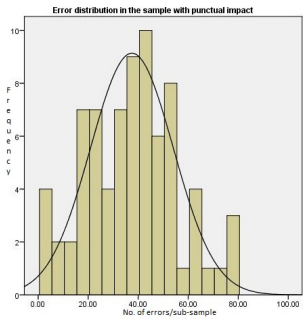
1. if the log trace was labeled as ERROR the probability is calculated as $\frac{1+\text{number of ERRORS raised before current error}}{\text{total number of events}}$;
2. if the log trace was labeled as INFO the associated probability is calculated as $\frac{1+\text{number of INFO raised before current error}}{\text{total number of events}}$;

After parsing all records we obtained a small sample of error and non-error events on which we applied three bootstrapping procedures: bootstrap with no decay effect - producing event queues in which each error has punctual impact, bootstrap with linear decay effect-producing an event queue in which every error has a linear decay impact and bootstrap with exponential decay effect - having as output an event queue in which error events have the effect of exponential decay.Every bootstrap procedure generates a number of event queues from the initial sample using the bootstrap procedures. In our simulations, a linear decay rate of 0.5 and an exponential decay rate of 0.8 where used, specifying the decreasing effect of the decay process.

For a more representative sample of error events, we performed bootstrapping on the event traces pool in order to evaluate our model near the limits and obtaining a larger event sample while preserving the structural characteristics of the initial sample.After each sample was generated using the bootstrap method we computed some descriptive indicators and histograms of the error events contained in the sample.




(a) Homogeneous Poisson process sample (Mean:19.88; Std.Dev:4.02; Skewness:0.37)

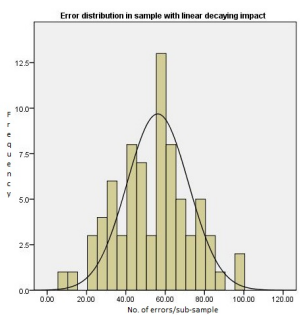



(b) Punctual error impact sample (Mean:37.60; StdDev:18.19; Skewness:0.79)

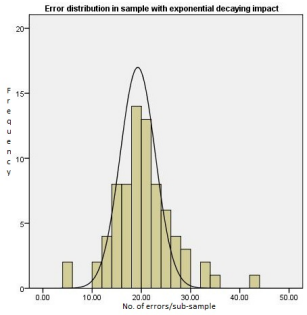
On the x-asis we plot the number of error events generated in each event queue by bootstrap routine. On y-axis we plot the frequencies of error occurrences in event queue obtaining the histograms of the error events in the samples. The error distributions are close to the normal distribution indicating that the bootstrap procedures generates robust and representative event samples.

### 4.2 The Different Error Impact Models in Assuring QoS

Previous simulations using randomly generated events from a homogeneous Poisson process showed that for an error rate below 13.41% per VM the replication cq indicator of fault node $R_k$(the replication node) is below the tq and migration is not needed.When the error rate is higher than 13.41% per Vm, migration is needed as we can see in Fig. 6, when $cq$ value peeks very close to $tq$ value. Next, in every case of error decays, we analyzed the evolution of $cq$ indicator of the replication node $R_k$ in relation with




(c) Linear decay error impact sample (Mean:53.89; Std-Dev:19.25; Skewness:0.002)




(d) Exponential decay error impact sample(Mean:19.95; Std-Dev:6.17; Skewness:0.69)

Fig. 5: Error distribution in the different scenario event samples

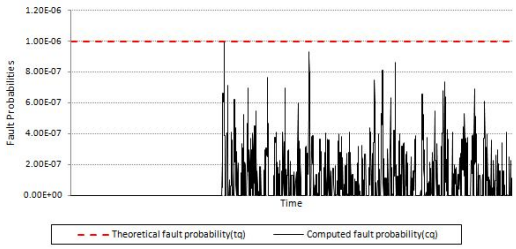



Fig. 6: Computed probability (cq) evolution on replication node for 13.41% errors from Poisson homogeneous process

the maximum accepted failure rate of 0.001 (99.9 % of availability of the service). Moreover these observations where made after 3000 VM events when migration was needed. Bellow this value no migration was needed.
Figure 7,8 and 9 present a comparative view of the evolution of cq indicator calculated for the replication node in each 3 event samples scenarios. The data used for these representations was considered from the moment when the fault agent is starting to take the decision for live migration due to VM and replica unreliability.

When the level of 0.001 $natural_Q$ is passed by the cq indicators live migration is required. Observing the frequency and amplitude of peeks correlated with the average percentage of live migration in each case, we can state that in the linear decay error impact scenario we have the smallest number of migrations, followed by the punctual decay error scenario and exponential decay error impact when live migration rate is the highest.Moreover the cq values have the natural tendency to return back to the imposed threshold of 0.001. This can be explained due to the compensation power of the non-error events.

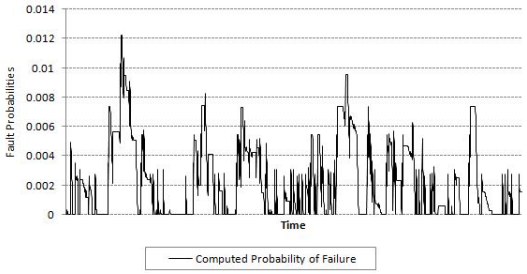



Fig. 7: Computed probability (cq) evolution on replication node in punctual error impact scenario [Average: MIGRATED-7%; MIGRATED-OK-0.2%]

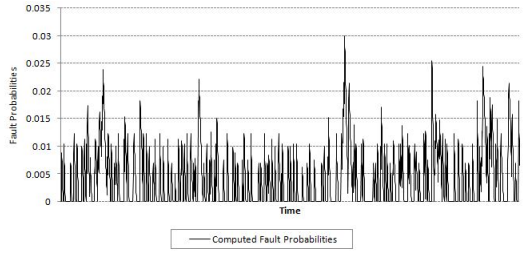



Fig. 8: Computed probability (cq) evolution on replication node in linear decay error impact [Average: MIGRATED-6.36%); MIGRATED-OK-0.23%]

## 5 Related work

Reliable cloud computing infrastructures require specially selected and optimized conditions for which the hardware equipment are not primarily conceived [6,20]. The complexity and the heterogeneity of hardware infrastructures can make cloud-based systems prone to significant amounts of failures and functioning incidents [11]. In failure-prone environments, fault tolerance is playing an essential part when assuring high levels of reliability and availability of the services [15]. To tolerate failures, these environments are characterized by systematic resource replication. By replicating individual virtual machines, fault tolerance and resource costs of applications can be improved [19].

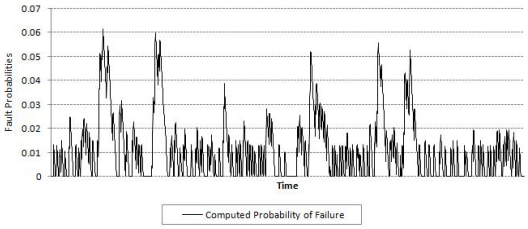



Fig. 9: Computed probability (cq) evolution on replication node in exponential decay error impact [Average: MIGRATED-10.3%); MIGRATED-OK-0.13%]

There are many symbolic representations used to describe the ability of a distributed system to cope with failures. In this paper, we describe the failure behaviour of a fault tolerant system through fault tree models [10,2], which are well fitted to present the different levels of errors specific to cloud computing infrastructures [8,18].

System virtualization technology is at the foundation of the cloud computing. Through virtualization, extra hosts can be easily added or removed at any time after the computing infrastructure has been set up. Thus, cloud infrastructure can start at initial modest scales, and then scale up progressively, rendering system services in a flexible and scalable way [16]. Virtualization [13] is highly beneficial since virtual machine instances level up computing resources stemming from potentially hundreds of thousands of commodity servers with heterogeneous CPU, memory and storage characteristics. When the replicas are located at physical hosts with different geographical location, the failures incidence on different replicas can become increasingly independent and the system much more reliable [3,21].

Application deployment on virtual machines instances within clouds bring new risks as failures in data centres are normally outside the clouds clients' control. Traditional approaches in assuring fault tolerance require the users to have very good knowledge of the base mechanisms. Cloud computing, on the other hand, through the abstraction layers, make transparent the architectural details for their clients. This changes the paradigm in approaching fault tolerance, as the cloud computing platforms need to address clients' reliability and availability concerns. Our approach is employing Xen [14] as a virtualization system but it is not constrained to it.

## 6 Conclusions

The paper presents a novel approach to autonomous management of virtual machine faults in Infrastructure as a Service cloud model, by deploying fault agents which conceptually reside in each virtual machine. Having the objective of avoiding QoS breaches, using fault tree analysis, fault agents can decide whether the virtual machine is reliable. To preserve the availability rate of the VMs, we use a strategy which combines active machine replication and live migration. When replication strategy fails, the migration process of virtual machine is used in order to assure continuity of the service.

We evaluated our fault tolerance model using a practical approach by generating virtual machine events traces from production Xen engine logs. We tested our model for robustness in a stress evaluation setup by using bootstrapping. Considering three sorts of error impact behaviors: punctual impact, linear decay and exponential decay, we showed that after an error event is raised, several consecutive non-error events can have the power to establish the reliable status of a virtual machine. Further work will focus on autonomous establishing when to apply each impact behavior and which one suits best for the IaaS setups.

## Acknowledgement

This work was co-financed from the European Social Fund through Sectoral Operational Programme Human Resources Development 2007-2013, project number POS-

DRU/159/1.5/S/134197 "Performance and excellence in doctoral and postdoctoral research in Romanian economics science domain". G.C. Silaghi acknowledges support from UEFISCDI under project JustASR - PN-II-PT-PCCA-2013-4-1644.

## References

1. Atif, M., Strazdins, P.: Adaptive parallel application resource remapping through the live migration of virtual machines. Future Generation Computer Systems 37, 148–161 (2014)
2. Bobbio, A., Portinale, L., Minichino, M., Ciancamerla, E.: Improving the analysis of dependable systems by mapping fault trees into bayesian networks. Reliability Engineering & System Safety 71(3), 249 – 260 (2001)
3. Clark, C., Fraser, K., Hand, S., Hansen, J.G., Jul, E., Limpach, C., Pratt, I., Warfield, A.: Live migration of virtual machines. In: Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2. pp. 273–286. NSDI'05, USENIX Association (2005)
4. Colesa, A., Mihai, B.: An adaptive virtual machine replication algorithm for highly-available services. In: Computer Science and Information Systems (FedCSIS), 2011 Federated Conference on. pp. 941–948. IEEE (2011)
5. Cully, B., Lefebvre, G., Meyer, D., Feeley, M., Hutchinson, N., Warfield, A.: Remus: High availability via asynchronous virtual machine replication. In: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation. pp. 161–174. San Francisco (2008)
6. Feller, E., Rilling, L., Morin, C.: Snooze: A scalable and autonomic virtual machine management framework for private clouds. In: 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid). pp. 482–489. IEEE Press (May 2012)
7. Gerofi, B., Vass, Z., Ishikawa, Y.: Utilizing memory content similarity for improving the performance of replicated virtual machines. In: Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on. pp. 73–80. IEEE (2011)
8. Guerraoui, R., Yabandeh, M.: Independent faults in the cloud. In: Proceedings of the 4th International Workshop on Large Scale Distributed Systems and Middleware. pp. 12–17. LADIS '10, ACM Press (2010)
9. Haimes, Y.: Risk Modeling, Assessment, and Management. Wiley (2005)
10. Jhawar, R., Piuri, V.: Fault tolerance management in iaas clouds. In: IEEE First AESS European Conference on Satellite Telecommunications (ESTEL). pp. 1–6. IEEE Press (2012)
11. Jhawar, R., Piuri, V.: Fault tolerance and resilience in cloud computing environments. In: Vacca, J.R. (ed.) Cyber Security and IT Infrastructure Protection, pp. 1 – 28. Syngress (2014)
12. Jin, H., Deng, L., Wu, S., Shi, X., Chen, H., Pan, X.: Mecom: Live migration of virtual machines by adaptively compressing memory pages. Future Generation Computer Systems 38, 23–35 (2014)
13. Kim, D.S., Machida, F., Trivedi, K.S.: Availability modeling and analysis of a virtualized system. In: Proceedings of the 2009 15th IEEE Pacific Rim International Symposium on Dependable Computing. pp. 365–371. PRDC '09, IEEE Computer Society (2009)
14. Nagarajan, A.B., Mueller, F., Engelmann, C., Scott, S.L.: Proactive fault tolerance for hpc with xen virtualization. In: Proceedings of the 21st Annual International Conference on Supercomputing. pp. 23–32. ICS '07, ACM Press (2007)
15. Nicolae, B., Cappello, F.: BlobCR: Virtual disk based checkpoint-restart for HPC applications on IaaS clouds . Journal of Parallel and Distributed Computing 73(5), 698 – 711 (2013)
16. Sampaio, A.M., Barbosa, J.G.: Towards high-available and energy-efficient virtual computing environments in the cloud. Future Generation Computer Systems 40, 30–43 (2014)

17. Travostino, F., Daspit, P., Gommans, L., Jog, C., De Laat, C., Mambretti, J., Monga, I., Van Oudenaarde, B., Raghunath, S., Yonghui Wang, P.: Seamless live migration of virtual machines over the man/wan. Future Generation Computer Systems 22(8), 901–907 (2006)
18. Undheim, A., Chilwan, A., Heegaard, P.: Differentiated availability in cloud computing slas. In: Proceedings of the 2011 IEEE/ACM 12th International Conference on Grid Computing. pp. 129–136. GRID '11, IEEE Computer Society (2011)
19. Vallee, G., Engelmann, C., Tikotekar, A., Naughton, T., Charoenpornwattana, K., Leangsuksun, C., Scott, S.: A framework for proactive fault tolerance. In: Third International Conference on Availability, Reliability and Security, (ARES). pp. 659–664. IEEE Press (2008)
20. Vishwanath, K.V., Nagappan, N.: Characterizing cloud computing hardware reliability. In: Proceedings of the 1st ACM symposium on Cloud computing. pp. 193–204. ACM Press (2010)
21. Wang, S.S., Wang, S.C.: The consensus problem with dual failure nodes in a cloud computing environment. Information Sciences 279, 213 – 228 (2014)