Paper

FIM-SIM: Fault Injection Module for CloudSim Based on Statistical Distributions

Mihaela-Catalina Nita, Florin Pop, Mariana Mocanu, and Valentin Cristea

Faculty of Automatic Control and Computers, Computer Science Department, University Politechnica of Bucharest, Bucharest, Romania

Abstract—The evolution of ICT systems in the way data is accessed and used is very fast nowadays. Cloud computing is an innovative way of using and providing computing resources to businesses and individuals and it has gained a faster popularity in the last years. In this context, the user's expectations are increasing and cloud providers are facing huge challenges. One of these challenges is fault tolerance and both researchers and companies have focused on finding and developing strong fault tolerance models. To validate these models, cloud simulation tools are used as an easy, flexible and fast solution. This paper proposes a Fault Injector Module for CloudSim tool (FIM-SIM) for helping the cloud developers to test and validate their infrastructure. FIM-SIM follows the event-driven model and inserts faults in CloudSim based on statistical distributions. The authors have tested and validated it by conducting several experiments designed to highlight the statistical distribution influence on the failures generated and to observe the CloudSim behavior in its current state and implementation.

Keywords—cloud simulation, continuous distributions, discrete distributions, fault injector.

1. Introduction

Cloud computing is in this moment the most used computational technology with implementations from private in-house environments (private clouds) to public clouds offered commercially to the customers and all sharing the same characteristics providing reliable services, fault tolerant hardware, and scalable computational power [1]. Born from the idea of a system that can serve seamlessly and transparently the end user, the cloud system architecture needs to be able to act like a reliable infrastructure with a high availability and degree of resource integration within.

In this context, one of the top things that a cloud provider must have in mind is the fault tolerance assurance. The literature provides various fault tolerance techniques [2], [3] and both research institutes and companies are still digging for finding complex and better solutions. The question rising at this moment is "how to better validate these models?" One of the most popular methods is cloud simulation based on a dedicated tool. A simulation represents an environment in which a system that behaves similarly to another system, but is implemented in an entirely different way [4]. It provides the basic behavior of a system but it may not reproduce the exact output as the real one. It is important to distinguish between simulation and emulation, which presents a system, that behaves exactly like another system, and it is expected to have the same output as the real one. In other words, it represents a complete replication of another system, but operating in a different environment. The cloud simulation top benefits are: flexibility, easy to customize and low cost [5], [6]. Designing, developing, testing and afterwards redesigning and retesting on the cloud can be expensive.

For this work, the authors have chosen CloudSim, a widely used and easy to integrate simulation framework together with CloudReports a graphical extension for CloudSim. The aim of this work is to create a module that can automatically inject faults into CloudSim order to verify its behavior in case of a fault. The questions rising when designing such a module are: when to inject a fault? Where to place the fault? How much time does it take?

For answering the first question, there are three different kinds of simulation systems: continuous, discrete and discrete-event systems. A continuous system modifies its state continuously in time. On the other hand a discrete system is observed only at some fixed regular time points. A real life analogy would be the health exam that we are taking every six months. In a discrete-event system, its state is determined by random event times t_1, t_2 etc. A continuous system will determine the time until the first failure, but a discrete system will found out the period between two failures. In our tool we considered both discrete and continuous distribution based event generator. The fault injection module will help the end user in determining the system reliability and drawing conclusions like: the failure caused by a network bottleneck will respect a Weibull distribution with parameters $\beta = x$ and $\theta = y$ hours. By having these variables one can find out the system reliability.

Regarding the second question, the following type of failures is considered: host failures (memory and PEs failure), VM creation failures, and high level failures like cloudlets.

For the third question the following assumptions have been made: the affected resources will be down during the rest of the simulation period and the VM creation failure for a specific host will be activate only for the moment when the event is introduced into the system.

In this context, this paper proposes a Fault Injector Module for CloudSim tool (FIM-SIM) for helping the cloud developers to test and validate their infrastructure. FIM-SIM follows the event-driven model and inserts faults in CloudSim based on statistical distributions.
This paper is structured as follows. Section 2 covers the critical analysis of exiting work, focusing on fault tolerance and various cloud simulation tools, together with CloudSim, the chosen solution. Section 3 describes the model of proposed simulation model, the statistical distribution used, system architecture and interaction with the other modules. Section 4 describes the experimental setup and results obtained. The paper ends with conclusion and future work, presented in Section 5.

2. Related Work

In order to test a system's capabilities and availability, his response in exceptional situation is analyzed and monitored [7]–[9]. Fault injection [10] is the key operation for testing and creating these abnormal situations for a system, offering him as input faulty states. The motivation around this kind of testing where the system is intentionally exposed to unwanted scenarios is that real life events are hard to collect and preferable to be avoided. With these techniques we can understand failures and validate the system availability to extreme scenarios [11].
The most common failures are: crash, time out of response, incorrect response message, arbitrary fails (byzantine failure) [12]. Beside the above classification of commonly failures we still have the hardware fails encounters, which are most of the times bypassed by redundant components of server's key items. Here it's worth to mention disks failure, network connectivity issues (network overload or adapter failure) and not least the environment incidents (fire, floods, earthquakes, etc).
To define the server availability and viability, the industry uses most of the times the indicator Mean Time To Failure (MTTF). This parameter is defined as the up-time divided by the number of failures. As a short example in cloud storage, is a Google study in which the availability propriety of a storage system is 4.3 MTTF and the most failure events (approx. 10%) last longer than 15 minutes [13]. As a result, many failures are correlated with each other and can chain to a series of critical events that can take down the system.
The arbitrary fail is by far the most difficult failure to predict due to its apparently randomness. The fault tolerant technique, designed to prevent such type of failures, is inspired by the Byzantine Generals Problem [14]. In this case, the system's components will fail in arbitrary ways and the overall system may respond in an unpredictable way unless is designed to be fault tolerant. We can take a logical example of 3 functions in which the result of the first function it will serve as an input for the second function and so on. If the first output of the first function it will have even a small round-off error this will propagate and create a much larger error until the values produced are worthless. This is a typical case of a small deviation that can cause a very powerful impact over the whole system. In real life we had two such examples: Amazon S3 was down for several hours due to a single-bit hardware error propagated through the entire system and Google – due to a code type error (";" misplaced) system was propagating no availability through servers around the world. In this case, the fault injection will help the cloud provider by injecting into the system several events, following a mathematical distribution, with the main target of failing several components, for example, the create virtual machine module.

In [8] an Adaptive Fault Tolerance in real-time cloud computing is proposed. This scheme tolerates the faults on the reliability basis of each computing node. A virtual machine is selected for computation if it has a higher reliability level and can be removed, if does not perform well for real time applications. There are two main types of nodes: a set of virtual machines, running on cloud infrastructure, and an adjudication node. The virtual machine contains the real time application algorithm and an acceptance test for its logical validity. On the adjudicator, there is a time checker, reliability assessor and some decision mechanism modules. The location of adjudication node depends on the type of the real time applications and the scenario in which they are used. It can be a part of the cloud infrastructure or can be a part of the user infrastructure. Generally, it is placed near to the sensors, actuators, and submission node.

The proposed Fault Injector Module will also help the above proposed adaptive fault tolerance module, by offering the context of determining the reliability of a resource in a certain scenario [15]. A critical analysis of existing tools for implementing fault tolerance techniques is presented in Table 1.

Cloud computing, as the successor of the grid systems, has all the attributes of the parallel system gathering a collection of virtualized nodes, dynamically provisioned and presented as one unified computing resource. The resources are allocated through the rules of service level agreements and negotiated between the service provider and consumer. Analyzing and testing the performance of a distributed system such as a public cloud has become more of a challenge. Cloud computing environments are offering a dynamically large pool of resources, configurable and optionally rebalanced. A full test of a public cloud can result in a significant cost and time, with the possibility to go to thousands of processing core involved. The most feasible option to test the service discovery performance, scheduling, monitoring, etc., of these systems without a scalable environment is a simulation tool. This tool will need to be able to reproduce the relevant tests and behavior of a real system.

iCanCloud is a modeling and simulation platform for cloud computing systems. The main purpose of the platform is to provide to the user useful information about the cost of given applications ran on the cloud specific hard-

Table 1
Existing tools for implementing fault tolerance techniques

Fault tolerance techniques	Policies	System	Programming framework	Environment	Fault detected	Application type
Self-healing, job migration, replication	Reactive/ Proactive	HAProxy	Java	Virtual machine	Process/ node failures	Load balancing/ Fault tolerance
Check-pointing	Reactive	SHelp	SQL, Java	Virtual machine	Application failure	Fault tolerance
Check-pointing, retry, self-healing	Reactive/ Proactive	Assure	Java	Virtual machine	Host, Network failure	Fault tolerance
Job migration, replication, Sguard, Resc	Reactive/ Proactive	Hadoop	Java, HTML, CSS	Cloud environment	Application/ node failures	Data intensive
Replication, Sguard, task resubmission	Reactive/ Proactive	Amazon EC2	Amazon Machine Image, Amazon Map	Cloud environment	Application/ node failures	Load balancing/ Fault tolerance

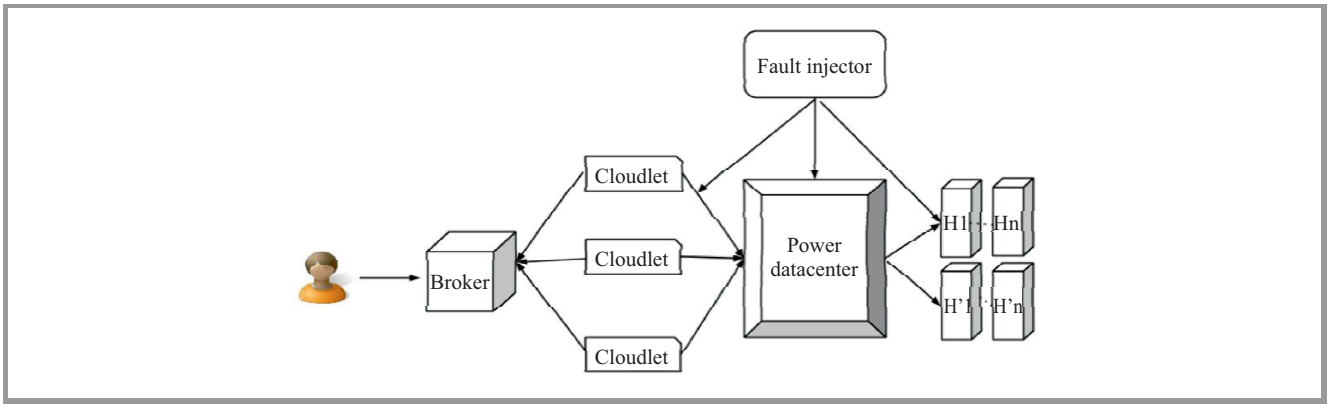

Fig. 1. FIM-SIM model system architecture.

ware and predict the trade-offs between cost and performance [16].

GreenCloud is packet-level simulator with the focus on the cloud communications, cloud computing data centers module with energy-aware modules. Also as a focus in data centers for energy saving, the tool is offering a detailed modeling for energy consumption by the IT equipment: computing nodes, network infrastructure and communication links [17].

The chosen simulation tool for proposed solution is CloudSim. The modularity of the tool made it the perfect choice. Each component is implemented as a Java class and can be extended very easy. CloudSim can provide an extensible simulation framework generalized by the main properties of the cloud concept [18].

3. FIM-SIM: Fault Injector Module for CloudSim

This section presents the proposed solution by providing further details on the implementation and the architecture.

3.1. FIM-SIM Model

The authors have developed a run-time, event driven fault injection module for cloud simulation. At random moments of time [19] it will generate an event and it will simulate a failure in the cloud system.

Its architecture is described in Fig. 1. We can notice that the Broker will send one or more cloudlets to the Datacenter and the Datacenter will schedule it, according with a Scheduling Policy, on a host. Each entity of CloudSim can send a certain event to another. In this case, the Fault Injector will send a message to the Datacenter and it will notify it about any failures that have occurred in the system. Sending the failure event is based on the following command:

```
sendNow(dataCenter.getId(),
        FaultEventTags.HOST_FAILURE,
        host);
```

One of the main characteristics of this fault injector module is the fact that it generates the events based on statistical distribution, both discrete and continuous. The fault injector is a thread that will be present for the whole simulation

period and it will try to insert faults based on a statistical method for generating random numbers.
For example at the moment t, the inject function will do the following:

```
mean = statisticalDistrbution.mean;
x = statisticalDistrbution.sample();

if (x > mean) {
    generateFault();
} else {
    Continue;
}
```

3.2. Statistical Distributions used by FIM-SIM

This section presents various statistical distributions: the ones that already exist in CloudSim plus another one – Poisson. It also presents the key concepts, implementations and further details for cloud simulation, fault tolerance and various simulation tools. We describe here only the Weibull, Poisson and Pareto distributions. The other ones used in our model are:

- Exponential distribution, used for analysis of the Poisson process,
- Uniform distribution (used very well in situation of risk analysis but also in algorithms for random generation of numbers due to its propriety of given equal probability over a known range for continuous distribution);
- Gamma distribution – model exponentially sums of random variables;
- LogNormal distribution – Galton distribution, used very often for reliability modeling of the application in order to achieve fault tolerance scenarios)
- Lomax distribution – Pareto 2 distribution, sed as an alternative to the exponential distribution with data heavily tailed;
- Zipf distribution, a discrete distribution with many applications in linguistics and modeling rare events.

Weibull distribution – for life data analysis, it is the most used statistical model [20]. As a continuous probability distribution, it is used in continuous simulations with application in economic forecasting, weather forecasting and all problems based on the solution of time dependent partial differential equations. The probability density function of a Weibull random variable is:

$$f(x;\lambda,k) = \begin{cases} \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k} & x \geq 0, \\ 0 & x < 0, \end{cases} \tag{1}$$

where λ is the scale parameter and k the shape parameter of the distribution function. The Weibull distribution, in particular cases, it interpolates between two known distribution: exponential distribution (where $k = 1$) and Rayleigh distribution (where $k = 2$). If we define the random Weibull variable x as time-to-failure then we will have a distribution where the rate of failure is proportional to a power of time. In this way, the Weibull distribution changes dramatically with the value of the shape parameter k. This parameter in the interval $(0,1)$ could be interpreted as follows:

- failure rate decreases in time for $k < 1$,
- failure rate increases with time for $k > 1$, an example here could be an aging process that is likely to fail as time goes by,
- constant failure rate in time for $k = 1$ (random external events are causing the failure).

The hazard rate of the distribution or failure rate is given by:

$$h(x;k,\lambda) = \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1}. \tag{2}$$

The **Poisson distribution** is a very useful and used distribution in experiment because many random events are following the pattern of this distribution [21]. The Poisson distribution is a discrete probability distribution that can be used to calculate the probability of certain event number to occur in a fixed interval of time and space. The events considered should be independent and with a known average occurrence rate. The probability function of the Poisson distribution for a given discrete random variable has the following definition:

$$f(k;\lambda) = \Pr(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}. \tag{3}$$

The notations used by the distribution are the following: $x = k$ means actual number of success resulted from the Poisson experiment, λ is the average number of successes that occurs in a certain known interval. In the Poisson experiment, the probability of a success to occur is proportional to the size on the interval/region and the smaller is the interval of time or region the probability will be close to zero.

Pareto Distribution – the distribution is named after the engineer Vilfredo Pareto and used to describe observable events in many fields of expertise. The statistical analysis [22] of the distribution can reveal the key events, which influence significantly the events chain part of the distribution. After rigorous analysis in quality control processes, charting the events based on the distribution, the Pareto rule was defined saying that 80% of the problems (events) are cause by 20% of key events/actions done wrong. The survival function is given by the probability of the Pareto random variable to be greater than some number x (x_m is the scale parameter and α is shape parameter for the Pareto distribution):

$$\overline{F}(x) = \Pr(X > x) = \begin{cases} \left(\frac{x_{\mathrm{m}}}{x}\right)^{\alpha} & x \geq x_{\mathrm{m}}, \\ 1 & x < x_{\mathrm{m}}. \end{cases} \tag{4}$$

The distribution can be used to describe many situations for equilibrium found in large/small items or events and make observations about the effectiveness of the process steps.

3.3. Integration in CloudSim

While describing CloudSim is important to mention the main entities/concepts its based on, in terms of terminology:

- **Processing element (PE)** or the unit responsible for computational execution. It can be seen as the smallest unit of the system responsible for the completion of a certain task;
- **datacenter** represents the resource provider. It is responsible for managing the available resources, i.e. hosts, PEs, VMs, memory;
- **broker** is responsible for mediating between the user and the datacenter. It represents the users needs. It sends the cloudlets for scheduling to the datacenter, monitors the cloudlets status and it informs the user about current state of his requirements;
- **cloudlet** represents the user requirement (a task for the cloud provider). It is characterized by length, PEs number (the number of PEs required for the cloudlet to be done);
- **host** is a physical resource characterized by a number of PE and RAM capacity (a computer);
- **Virtual machine (VM)** is a software-based emulation of a computer.

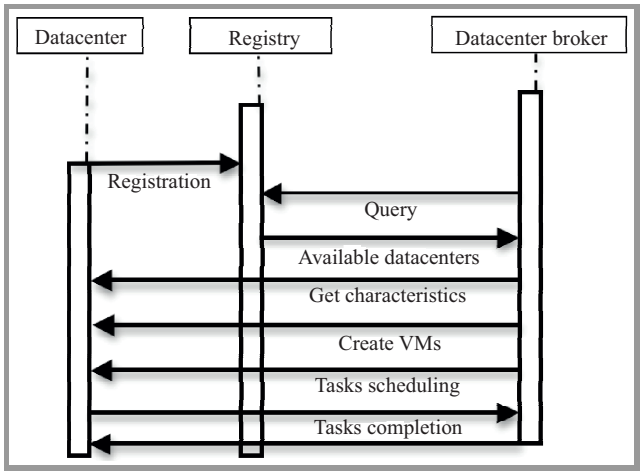

Fig. 2. CloudSim communication model.

The entities in CloudSim communicate through messages. Since host and VM are static entities, each change in their state should be realized by the datacenter. Figure 2 presents an example of the messages flow during the simulation between the broker and the datacenter. The broker, based on the simulation configuration (number of cloudlets and their specification) will request the VM creation, cloudlets scheduling and it will wait to be informed by the datacenter when the cloudlets completion is realized.
The Fault Tolerance Module is extending the CloudSim core functions with the following entities:

1. FaultInjector
   - extends the SimEntity class;
   - it will be started at simulation startup along with the other entities from the system;
   - it is responsible for inserting fault events at random moments of time;
   - the random generation of moments of time is based on a statistical distribution.
2. FaultEvent
   - extends the SimEvent class;
   - describes a fault event: source, destination, time and type;
   - tag type: HOST_FAILURE, CLOUDLET_FAILURE, CREATE_VM_FAILURE;
   - it is created in the Fault Injection Module.
3. FaultHandlerDatacenter
   - extends the datacenter class;
   - processes fault events sent by the FaultGenerator;
   - it updates the cloudlet execution/status according to the fault event type;
   - it handles VM migration;
   - since host and VM are static entities, all its state modification should be processed by the datacenter.

3.4. Fault Injector Integrated with CloudReports

The authors have chosen to integrate FIM-SIM in CloudReports, a GUI implementation for CloudSim. The module is an integrated part of CloudSim and can be further used in any other application that is based on CloudSim. CloudReports is an extension that can be used with CloudSim as a simulation tool. It's basically a graphical tool that helps to simulate distributed system environments, providing an easy interface to user and pluggable extension.
To meet all CloudSim proprieties, CloudReports can provide a number of datacenters, each been 100% customizable, and run them as a provider of services or in cloud terms as an Infrastructure as a Service (IaaS). The customers to this solution are also customizable with a full cost and resource allocation. They can modify and set the number of VMs needed as a user. The broker can allocate the resources and track down the consumptions. Virtual machines can be entirely customized from CPU processing

power to RAM and network bandwidth but can also run scheduling algorithms for tasks in place.

As this is a graphical tool, CloudReports can generate reports for each simulation from raw data to processed data. Those reports that can be displayed as HTML reports or exported for further analysis in third-party application tools. This application has been built on top of CloudSim as a framework for modeling and simulation of IaaS by Thiago T. Sa for final graduation project at Federal University of Ceara, Brazil. This software is licensed under GNU GPL 3.0.

This application have been chosen based on the fact that it provides a friendly GUI that permits easy creation, modification and removal of any cloud component. In addition, the fault injector was integrated in the graphical interface, allowing two options: enable/disable fault injector and choose the statistical method for generating the events.

As mentioned before one of the Cloud Reports benefits is its logging and reporting system, both in raw data and graphic interface. One can obtain all data types about the system performance, i.e., power consumption, storage usage, CPU performance needed and bandwidth used from provider and customer perspectives. In addition it offers information about the cloudlets successfully finished, the execution time evolution, the average start and finish time, etc.

4. FIM-SIM Evaluation

The several experiments have been conducted with the main goal of noticing CloudSim behavior in case of a failure occurrence, the overall system performance and the rate of faults generated based on the statistical distribution chosen. In performed tests the number of hosts was varied from 10 to 30 and the number of VMs from 4 to 10. The cloudlet generation is dynamic and continuous for the observation time specific to each simulation. It is realized by CloudReports broker: at every cloudlet finish it will generate another one with a random length. For this reason, the total number of cloudlets sent in a period of time t, may vary from simulation to simulation, even if all the describing parameters are the same.

The authors have inserted 2 types of failures: host failures and high level failures (cloudlet failures caused by any networking problem that CloudSim can not control). The second type of failures have been chosen for the tests where we followed only the faults generation rate or the relationship between the statistical parameters and total number of failures generated.

The experiments for VM_CREATION, VM_DESTROY and VM_SCHEDULING_ERROR faults are no relevant because the CloudReports simulations are not dynamic so it does not permit the dynamic creation of other VMs or further host integration. In this case, it was out of the target to focus on these types of failures since we have wanted to notice the system behavior during a simulation.

4.1. Poisson Failures Distribution

In the first experiment a 10 hosts, 4 VMs and an observation time of one hour have been chosen. There were failures generated for each of them and starting with $t > 50$ minutes the system failed to provide requests. When the number of hosts was modified to 30, the system continued to execute the cloudlets. It can be noticed that there is a tendency to generate more faults in the second half of the observed interval (see Fig. 3).

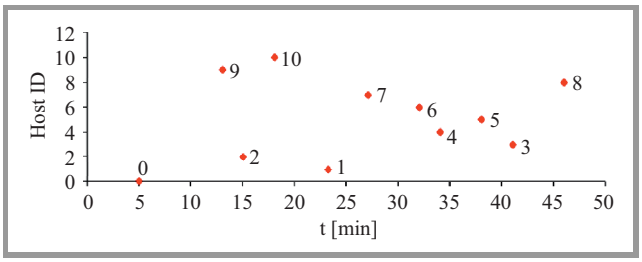

Fig. 3. Poisson failures distribution.

The host selection process was randomly for every moment that was considered a fault. It is important to mention that CloudSim has a sequential method of selecting the host that will be eligible for migration (based on a power consumption limit). In this case, if the selection of the next host to be failed is also sequential we will have a very big number of migrations generated. The overall performance of the cloud will be affected in terms of response time and resources consumption. The success rate it is not affected. Table 2 presents a short example of the number of migrations generated for several randomly host chosen simulations and for one sequential.

Table 2
Migrations generated for several simulations

Simulation ID	Number of migrations	Hosts implied
1	4	2
2	12	4
3	8	3
4	8	3
5	4	2
Sequential	44	11

As a brief conclusion, an average rate of 4/8 migrations can be expected, which will bring a better performance to the system then 44.

4.2. Weibull Failures Distribution

In this case the cloudlets failures for 2 different customers have been inserted, chosen in an alternative way, randomly each other. Here we can notice that the failures are spread

all over the interval with a frequency rate of at every 5 minutes (see Fig. 4).

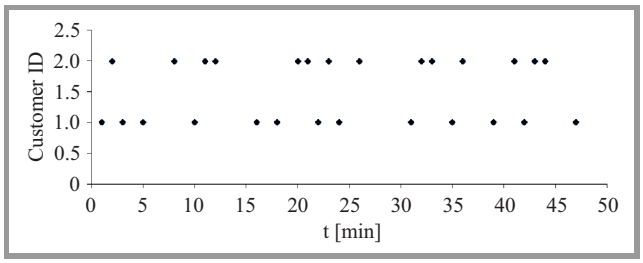

Fig. 4. Weibull failures distribution.

4.3. Exponential Failures Distribution: λ Variation

For the exponential distribution the λ is increased and growing failures number is observed (see Fig. 5).

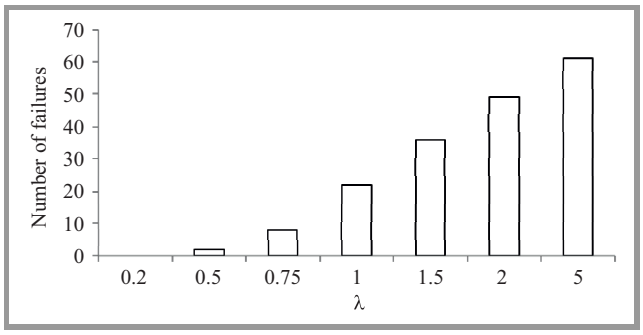

Fig. 5. Exponential distribution failure rate (λ variation).

4.4. Pareto Failures Distribution

In the case of Pareto distribution mostly the same number of failure is obtained generated without any observable relationship between the parameters and faults injected. However the maximum value of failures inserted was 22 (see Table 3).

Table 3
Pareto Distribution Results

α	x_m	Mean	Number of faults
1.5	0.5	0.15	9
2	0.5	1	14
3	0.5	0.75	20
4	0.5	0.66	21
5	0.5	0.625	22
10	0.5	0.555	22
20	0.5	0.52	22
50	0.5	0.51	22
100	0.5	0.5005	22
2	1	2	14
2	2	4	14
2	3	6	14
2	10	20	14
2	100	200	14

4.5. System Performance without VM Migration

For the last experiment the VM migration was deactivated and a 10 different simulations without any change in the initial state of the system were run. The results are presented in Fig. 6 The best success rate obtained, in case of a host failure, depending on the random sequence of failures was 52%. The events were generated based on Poisson distribution.

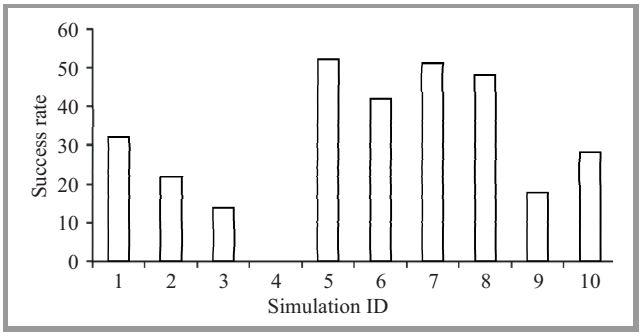

Fig. 6. System performance without VM migration.

Other observations realized during the performed experiments:

- CloudSim can assure host failures through migration;
- If a failure is sent during the execution of some cloudlets those cloudlets will fail even though a VM migration was started. A success rate of 90–100% was obtained;
- It does not resubmit the failed cloudlets;
- CloudReports is based on power aware module: it will generate migration if the power consumption has level up at certain threshold.

5. Conclusion

The work presented in this paper can be summarized as follows. A Fault Injector Module, named FIM-SIM, have been designed and implemented, on cloud simulation with the main goal to provide a helpful tool for validation and testing of various fault tolerance models or any new policy that can be faulty. In authors vision, the main characteristic of this fault injector is its intention to reproduce faults in a more natural and realistic way. We all have faced some moment in times when something went wrong and we have described it as “randomly happen”. Maybe, in some cases, randomly is not so randomly as it seems to be. Probably there are failures that tend to happen with a certain frequency or tend to respect a certain pattern in time. For these situations a Fault Injector Module have been built that tends to produce faults event according to a certain distribution.
The authors have built FIM-SIM module as a integrated part of CloudSim and it was extended with CloudReports,

a GUI solution for CloudSim. This module was designed as an event generator with various ways of "randomly" generating the events in time following distributions as: a Poisson, Weibull, Gamma, Pareto, and Exponential.

In conducted experimental results the authors have noticed some tendencies in failures distribution under Poisson and Weibull distributions. For the Poisson distribution, the failures tend to happen more in the second half of the analyzed interval but in the Weibull distribution it tends to respect a certain frequency: 3 every 5 minutes for example.

CloudSim can respect requests even if the resources are failing by activating the VM migration. There is a strong relationship between the sequence of failures and the VM migration. If the VM migration is realized based on the location criteria and there are failures that happens in a certain area, then the number of migration will increase and the overall performance will be lower. Another important observation is that CloudSim does not resubmit the failed cloudlets.

The authors think that FIM-SIM module can have a great impact in the research area for cloud providers. Sometimes is very expensive to validate a model in real life, to realize that it has many drawbacks and you have to redesign it an implement it again. As a further work, the authors intend to propose some fault tolerance techniques and try to extend the type of failures that can appear in a cloud environment.

Acknowledgements

The research presented in this paper is supported by following projects: "*SideSTEP - Scheduling Methods for Dynamic Distributed Systems: a self-* approach*", ID: PN-II-CT-RO-FR-2012-1-0084; *CyberWater* grant of the Romanian National Authority for Scientific Research, CNDI-UEFISCDI, project number 47/2012; *MobiWay*: Mobility Beyond Individualism: an Integrated Platform for Intelligent Transportation Systems of Tomorrow - PN-II-PT-PCCA-2013-4-0321; *clueFarm*: Information system based on cloud services accessible through mobile devices, to increase product quality and business development farms – PN-II-PT-PCCA-2013-4-0870.

References

[1] A. Hameed *et al.*, "A survey and taxonomy on energy efficient resource allocation techniques for cloud computing systems", *Computing*, 2014 (in Press).

[2] R. Jhawar and V. Piuri, "Fault tolerance management in Iaas clouds", in *Proc. IEEE 1st AESS Eur. Conf. Satell. Telecommun. ESTEL 2012*, Rome, Italy, 2012, pp. 1–6.

[3] M. Nastase, C. Dobre, F. Pop, and V. Cristea, "Fault tolerance using a front-end service for large scale distributed systems" in *Proc. 11th Int. Symp. Symb. Numeric Algorithms for Scient. Comput. SYNASC 2009*, Timisoara, Romania, 2009, pp. 229–236.

[4] Z. Zheng, T. C. Zhou, M. R. Lyu, and I. King, "Component ranking for fault-tolerant cloud applications", *IEEE Trans. Serv. Comput.*, vol. 5, no. 4, pp. 540–550, 2012.

[5] J. Kolodziej, H. Gonzalez-Velez, and L. Wang, "Advances in data-intensive modeling and simulation", *Future Gener. Comp. Syst.*, 2014 (in Press).

[6] C. Dobre, F. Pop, and V. Cristea, "A fault-tolerant approach to storing objects in distributed systems", in *Proc. Int. Conf. on P2P, Parall., Grid, Cloud and Internet Comput. 3PGCIC 2010*, Fukuoka, Japan, 2010, pp. 1–8.

[7] P. Das and P. M. Khilar, "VFT: A virtualization and fault tolerance approach for cloud computing", in *Proc. IEEE Conf. Inform. Commun. Technol. ICT 2013*, Jeju Island, South Korea, 2013, pp. 473–478.

[8] S. Malik and F. Huet, "Adaptive fault tolerance in real time cloud computing", in *Proc. IEEE World Congr. Serv. SERVICES 2011*, Washington, DC, USA, 2011, pp. 280–287.

[9] Y. He *et al.*, "A simulation cloud monitoring framework and its evaluation model", *Simul. Modell. Pract. and Theory*, vol. 38, pp. 20–37, 2013.

[10] S. Vilkomir, "Cloud testing: A state-of-the-art review", *Inform. & Secur.: An Int. J.*, vol. 28, no. 2, pp. 213–222, 2012.

[11] A. Boteanu, C. Dobre, F. Pop, and V. Cristea, "Simulator for fault tolerance in large scale distributed systems", in *Proc. IEEE 6th Int. Conf. Intell. Comp. Commun. and Process. ICCP 2010*, Cluj-Napoca, Romania, 2010, pp. 443–450.

[12] A. Costan, C. Dobre, F. Pop, C. Leordeanu, and V. Cristea, "A fault tolerance approach for distributed systems using monitoring based replication", in *Proc. IEEE 6th Int. Conf. Intell. Comp. Commun. and Process. ICCP 2010*, Cluj-Napoca, Romania, 2010, pp. 451–458.

[13] D. Ford *et al*., "Availability in globally distributed storage systems", in *Proc. 9th USENIX Conf. Operat. Sys. Design and Implemen. OSDI'10*, Berkeley, CA, USA, 2010, pp. 1–7. USENIX Association.

[14] Y. Zhang, Z. Zheng, and M. R. Lyu, "BFTCloud: A byzantine fault tolerance framework for voluntary-resource cloud computing", in *Proc. IEEE 4th Int. Conf. Cloud Comput. CLOUD '11*, Washington, DC, USA, 2011, pp. 444–451.

[15] F. Cappello, "Fault tolerance in petascale/ exascale systems: current knowledge, challenges and research opportunities", *Int. J. High Perform. Comput. Appl.*, vol. 23, no. 3, pp. 212–226, 2009.

[16] A. Núñez *et al.*, "A flexible and scalable cloud infrastructure simulator", *J. Grid Comput.*, vol. 10, no. 1, pp. 185–209, 2012.

[17] L. Liu *et al.*, "Greencloud: A new architecture for green data center", in *Proc. 6th Int. Conf. Industry Session on Autonomic Comput. and Communi. Industry Session ICAC-INDST '09*, Barcelona, Spain, 2009, pp. 29–38.

[18] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms", *Softw. Pract. Exper.*, vol. 41, no. 1, pp. 23–50, 2011.

[19] K. J. Lang, "Practical algorithms for generating a random ordering of the elements of a weighted set", *Theor. Comp. Sys.*, vol. 54, no. 4, pp. 659–688, 2014.

[20] A. M. Razali and A. A. Al-Wakeel, "Mixture weibull distributions for fitting failure times data", *Appl. Math. Comput.*, vol. 219, no. 24, pp. 11358–11364, 2013.

[21] G. Shmueli, T. P. Minka, J. B. Kadane, S. Borle, and P. Boatwright, "A useful distribution for fitting discrete data: revival of the Conway-Maxwell-Poisson distribution", *J. Royal Statist. Soc.: Series C (Applied Statistics)*, vol. 54, no. 1, pp. 127–142, 2005.

[22] M. E. J. Newman, "Power laws, pareto distributions and Zipf's law", *Contemporary Phys.*, vol. 46, no. 5, pp. 323–351, 2005.

Mihaela-Catalina Nita finished his M.Sc. in 2014 on Parallel and Distributed Computing Systems program at University Politehnica of Bucharest, Faculty of Automatic Control and Computers, Computer Science Department. She is an active member of Distributed Systems Laboratory. Her research interests are in cloud system and resource management, especially in fault-tolerance systems, SLA assurance, multi-criteria optimization, cloud middleware tools, complex applications design and implementation.

E-mail: catalina.nita@hpc.pub.ro
Faculty of Automatic Control and Computers
Computer Science Department
University Politechnica of Bucharest
313, Splaiul Independentei, Sector 6
060042 Bucharest, Romania

Florin Pop received his Ph.D. in Computer Science at the University Politehnica of Bucharest in 2008. He received his M.Sc. in Computer Science in 2004 and the Engineering degree in Computer Science in 2003, at the same University. He is Associate Professor within the Computer Science Department and also an active member of Distributed System Laboratory. His research interests are in scheduling and resource management, multi-criteria optimization methods, grid middleware tools and applications development, prediction methods, self-organizing systems, contextualized services in distributed systems. He is the author or co-author of more than 150 publications. He served as guest-editor for International Journal of Web and Grid Services and he is managing editor for International Journal of Grid and Utility Computing. He was awarded with "Magna cum laude" distinction from University Politehnica of Bucharest for his Ph.D. results, one IBM Faculty Award in 2012 or the project CloudWay Improving resource utilization for a smart cloud infrastructure, two Prizes for Excellence from IBM and Oracle (2008 and 2009), Best young researcher in software services Award, FP7 SPRERS Project, Strengthening the Participation of Romania at European R&D in Software Services in 2011 and two Best Paper Awards. He worked in several international and national research projects in the distributed systems field as coordinator and member as well. He is a senior member of the IEEE, a member of the ACM and a member of the euroCRIS.

E-mail: florin.pop@cs.pub.ro
Faculty of Automatic Control and Computers
Computer Science Department
University Politechnica of Bucharest
313, Splaiul Independentei, Sector 6
060042 Bucharest, Romania

Mariana Mocanu is a Professor of the Computer Science and Engineering Department of the University Politehnica of Bucharest, and has a long experience in developing information systems for industrial and economic processes, and in project management. She performs teaching for both undergraduate and master's degree in software engineering, systems integration, software services and logic design. At the University of Regensburg, as visiting professor, she thought Process Computers. She worked for ten years in a multidisciplinary research team for vehicles, being co-author of two patents. She participated in numerous research projects, implementing information systems for control/optimization of processes in various areas (transport, environment, medicine, natural resources management). Her results of research and teaching are reflected in articles published in journals, in papers presented at national and international conferences, and books. She is a member of the University Senate, at the faculty she is responsible for quality assurance and is a board member of the department.

E-mail: mariana.mocanu@cs.pub.ro
Faculty of Automatic Control and Computers
Computer Science Department
University Politechnica of Bucharest
313, Splaiul Independentei, Sector 6
060042 Bucharest, Romania

Valentin Cristea is a Professor of the Computer Science and Engineering Department of the University Politehnica of Bucharest, and Ph.D. supervisor in the domain of Distributed Systems. His main fields of expertise are large scale distributed systems, cloud computing and e-services. He is co-founder and director of the National Center for Information Technology of UPB, and has a long history of experience in the development, management and coordination of international and national research projects. He led the UPB team in COOPER (FP6), datagrid@work (INRIA "Associate Teams" project), CoLaborator project for building a Center and collabora-

tive environment for high-performance computing in Romania, distributed dependable systems project DEPSYS, and others. He co-supervised the UPB Team in European projects SEE-GRID-SCI (FP7) and EGEE (FP7). The research results have been published in more than 230 specialist papers in international journals or peer-reviewed proceedings, and more than 30 books and book chapters. Prof. Cristea organized several scientific workshops and conferences. In 2003 and 2011 he received the IBM faculty award for research contributions in e-Service and Smart City domains. He is a member of the Romanian Academy of Technical Sciences.

E-mail: valentin.cristea@cs.pub.ro
Faculty of Automatic Control and Computers
Computer Science Department
University Politechnica of Bucharest
313, Splaiul Independentei, Sector 6
060042 Bucharest, Romania