View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by White Rose Research Online

This is a repository copy of *TEA-Cloud: A formal framework for testing cloud computing system*.

White Rose Research Online URL for this paper:
https://eprints.whiterose.ac.uk/163550/

Version: Accepted Version

Article:

Nunez, A., Canizares, P., Nunez, M. et al. (1 more author) (2020) TEA-Cloud: A formal framework for testing cloud computing system. IEEE Transactions on Reliability. ISSN 0018-9529

https://doi.org/10.1109/TR.2020.3011512

© 2020 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other users, including reprinting/ republishing this material for advertising or promotional purposes, creating new collective works for resale or redistribution to servers or lists, or reuse of any copyrighted components of this work in other works. Reproduced in accordance with the publisher's self-archiving policy.

Reuse

Items deposited in White Rose Research Online are protected by copyright, with all rights reserved unless indicated otherwise. They may be downloaded and/or printed for private study, or other acts as permitted by national copyright laws. The publisher or other rights holders may allow further reproduction and re-use of the full text version. This is indicated by the licence information on the White Rose Research Online record for the item.

Takedown

If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

White Rose university consortium
Universities of Leeds, Sheffield & York

eprints@whiterose.ac.uk
https://eprints.whiterose.ac.uk/

TEA-*Cloud*: A formal framework for testing cloud computing systems

Alberto Núñez, Pablo C. Cañizares, Manuel Núñez and Robert M. Hierons

Abstract—The validation of a cloud system can be complicated by the size of the system, the number of users that can concurrently request services, and the virtualisation used to give the illusion of using dedicated machines. Unfortunately, it is not feasible to use conventional testing methods with cloud systems. This paper proposes a framework, called TEA-*Cloud*, that integrates simulation with testing methods for validating cloud system designs. Testing is applied on both functional and non-functional aspects of the cloud, like performance and cost. The aim of the framework is to provide a complete methodology to help users to model both software and hardware parts of cloud systems and automatically test the validity of these clouds using a cost-effective approach. *Metamorphic testing* is used to overcome the lack of an *oracle* that checks whether the behaviour observed in testing is allowed. Metamorphic testing is based on *metamorphic relations* (MRs). We define three families of MRs, which target issues such as performance, resource provisioning and cost. TEA-*Cloud* was evaluated through an empirical study that used fault seeding (mutation) and ten MRs for testing different cloud configurations. The results were promising, with TEA-*Cloud* finding all seeded faults.

Index Terms—Cloud computing, Metamorphic testing, Simulation, Mutation testing

1 Introduction

Computing architectures based on cloud computing systems are currently the most cost-effective solution for many end-users: enterprises and scientists. However, a range of factors have to be managed when providing a system with features such as 24/7 availability, worldwide utilisation, and easy access for every user.

While a cloud system should provide the expected functionality, aspects like performance and resource provisioning are also important since a data center will contain a vast number of computers and communication networks. The importance of this issue is only likely to grow since cloud systems are designed to be scalable, with it being possible to add resources in order to increase the overall system capacity. In particular, the problem of resource provisioning in cloud computing environments is complicated by the need for end-users to not notice a performance loss.

One of the main problems that we have to overcome when developing cloud computing systems is to ensure that the behaviour of the system is consistent with expectations. Here, the behaviour of the system includes factors such as performance and user management. Currently, *testing* [1], [2] is the most widely used technique to validate the correctness of systems. If we start the development of a system from a formal model, then testing can be used to perform more rigorous analysis [3]. However, the development of formal testing methodologies for cloud systems is a significant challenge [4].

A cloud system will normally contain many elements that interact and are distributed, which makes it difficult to apply formal testing approaches. In testing it is normal to use an *oracle* that checks that the behaviour observed during testing, with a given test case, is allowed/acceptable. An oracle can be a realisation of a formal specification of the system or a set of properties that the system has to fulfill. However, in some situations an oracle is not available or is computationally too expensive to apply, and alternative approaches must be used [5], [6]. This is particularly problematic when testing complex systems, where many test cases must be generated and executed if we are going to completely check the behaviour of the system. These problems arise in cloud computing, where there is rarely an oracle and large test suites are required to check the critical parts of the system.

This paper presents a simulation-based framework, called TEA-*Cloud*, designed to alleviate these issues. TEA-*Cloud* integrates simulation – to represent the behaviour of cloud computing systems – with testing methods for checking the correctness of cloud systems. The main goal is to provide a methodology, supported by a tool, that allows users to model both the software and hardware parts of cloud systems, design new cloud systems and automatically test these systems using a cost-effective approach that considers both functional and non-functional aspects of the cloud.

The main advantages of TEA-*Cloud* can be summarised as follows:

- *A. Núñez, P. C. Cañizares and M. Núñez are with Design and Testing of Reliable Systems research group, Universidad Complutense de Madrid, Madrid, 28040, Spain.*
  E-mail: alberto.nunez@pdi.ucm.es, pablocc@ucm.es, mn@sip.ucm.es
- *R. Hierons is with Department of Computer Science, The University of Sheffield, Sheffield, SD1 4DP, UK.*
  E-mail: r.hierons@sheffield.ac.uk

This work has been supported by the Spanish MINECO-FEDER (grant number FAME, RTI2018-093608-B-C31) and the Region of Madrid (grant number FORTE-CM, S2018/TCS-4314).

- *Flexibility*: users can model and simulate a wide range of cloud computing systems, configuring both the hardware parts, like computers, network topology and racks management, and software parts, like hypervisors, virtual machines and user policies.
- *Scalability*: the size of the simulated cloud can vary from several computers to thousands of machines grouped in racks.
- *Costless*: using the proposed framework does not require specific hardware to be executed. Also, a cloud is not required for experiments.
- *Automatic testing*: the validity of each cloud environment can be automatically checked to increase the confidence in it functioning properly.

It is important to note that TEA-*Cloud* uses techniques for *testing cloud computing models*, which is different from *testing in the cloud*. We analyse complete cloud configurations and then check whether these designs are appropriate or not. A real cloud system is not needed because this process can be executed in any regular computer. In contrast, testing in the cloud uses a cloud system to execute tests, which may be related (or not) to checking the underlying cloud system where these tests are executed.

Virtualisation is one of the key aspects of cloud computing environments, since it is the component that allows the separation of physical and logical resources. Therefore, this part needs to be simulated accurately. In order to reach this goal, we provide several methods to design, with enough flexibility and accuracy, a cloud computing environment.

Metamorphic testing (MT) [7], [8], [9], [10] uses expected properties of the target functions to alleviate the oracle problem. These properties relate multiple test-inputs/observed-outputs obtained from the tested system using *metamorphic relations* (MRs). Consider, for example, the (much simpler) problem of testing an implementation $\sin$ of the trigonometric sine function. Any implementation of this function will be an approximation and the process of determining the expected value for a given input is complicated, expensive and error-prone. However, we do have expected properties such as $\sin(x) = -\sin(-x)$. If we were to use this property as an MR, then we would start with a test input such as 0.3 and call the function with it. We would then use the *follow-up* test input -0.3 and check whether the two resultant outputs were related as expected (one was the negation of the other). If the expected relation does not hold then we know that there has been a failure. Additionally, we use the constraints defined in the MRs to automatically generate test suites. This is especially important because the generated test cases check the specific features of the system that are reflected in the MRs.

We will provide an example to motivate the need for novel testing approaches, and why the combination of MT and simulation is a promising approach. Let us suppose that we are interested in validating the performance of a cloud system, which can be represented by, amongst other metrics, the average CPU utilisation, the number of deployed VMs per hour and the data transmitted through the network. Note that, in all these cases, the result is represented by a single value (e.g. the time needed to complete a certain task). It is important to emphasise that cloud systems consist of a wide variety of heterogeneous sub-systems. For example, they will normally contain computing, networking, storage and virtualisation facilities. Thus, in order to estimate, or predict, the result that should be provided by a test it is necessary to analyse a range of different aspects of the many components of the studied cloud system and how they interact. As a result, it is typically infeasible to determine the performance that a cloud system should have with a given test case; there is an oracle problem.

The cost of building a large cloud system means that testing should initially happen within a simulation because we do not want to build the system and then find that it has deficiencies that could be found through simulation. Coming back to our example, the tester will run the task in the chosen cloud (either in the *real* cloud or, as we propose, in a *simulation* of it) and obtain the time needed to complete the task. At this point the tester faces the following problem: how can they decide whether this is the expected time? They do not have an oracle (that is, an automatic procedure to decide whether this is the expected time) and it is infeasible for an expert, even after a careful study of the most relevant features of the cloud, to predict the performance of the cloud. In conclusion, it is not possible to decide whether the result of a single experiment is good or not.

MT does not solve the above specific problem, that is, it does not tell us whether a specific test shows a failure, but it does provide us with an elegant solution. Instead of analysing the correctness of a single output for a single input, MT tells us to put together a set of inputs, and their corresponding outputs, and jointly analyse them. In our running example, if we have two tasks and the first one is simpler to solve than the second one, then the time needed to complete the first task should be less than that needed to complete the second one. Moreover, MT also allows us to validate one model using another one. Let us assume that we have two different clouds with the same amount of resources (we can quantify this either in computational terms or in monetary ones), we run different tasks and we find that the performance of the first cloud is always better than the performance of the second one. In this case, using MT, we know that we can discard the second cloud.

In addition to the lack of oracle, in order to validate any complex system, it is desirable to run *many* tests so that we can explore a range of scenarios. Simulation allows us to launch them in parallel, thus reducing the total execution time. However, test automation is typically hampered by the absence of a test oracle or a separate specification that can act as an oracle. In the proposed framework, the above problem is addressed

by using MT to replace the oracle, allowing a large number of test executions to be run within a simulation. Importantly, as we will see, it is possible to use MT to check a number of different types of properties including some non-functional properties.

Initially, a set of default relations is provided by TEA-*Cloud*. However, due to the scalable design of TEA-*Cloud*, new relations can be added by end-users, increasing the functionality of the framework.

In addition to the previously described capabilities provided by TEA-*Cloud*, the main contributions of our research can be summarised as follows:

- In contrast to existing work, focusing only on simulating cloud systems, TEA-*Cloud* also provides techniques for automatically checking the correctness of the cloud. This includes the automatic generation of test cases that focus on critical parts of the system and test the appropriateness of both software and hardware components. This is achieved by integrating MT techniques in the proposed framework.
- Due to the scalable and flexible design of TEA-*Cloud*, each user may increase its functionality by adding and modifying MRs. The benefits of this feature are two-fold. First, the power of the framework can be increased through additional aspects of the cloud being tested. Second, the spectrum of cloud configurations to be tested by TEA-*Cloud* can be increased. The more MRs defined, the wider spectrum of configurations can be tested.
- We report on the results of a set of experiments in which we used *mutation testing* [11], [12], [13] techniques to inject faults in the studied systems.

This paper builds on top of preliminary work [14] that presented the basic guidelines of the approach. The main contributions of this paper with respect to our previous work are:

- We provide a language to fully model cloud systems. This allows users not only to represent cloud architectures but also to formally analyse them.
- We have extended and categorised the collection of MRs. We currently provide a structured approach to use different relations and understand the obtained results. It is important to note that, while previous work provided MRs describing basic aspects of the cloud, the work presented in this paper provides a more detailed collection of MRs dealing with monetary cost, user management and performance, amongst others. Moreover, different resource allocation and user management policies are analysed.
- We report on the results of experiments that investigated the proposed MRs. Within the experiments, we considered two aspects. First, we were interested in whether the MRs are valid, in the sense that they do not (incorrectly) suggest that a correct system is faulty. Second, we used fault injection to assess the fault detection effectiveness of the MRs.
- In previous work we used simple systems that contain manually inserted faults. In contrast, in this paper we model and analyse complex systems and use mutation testing to inject faults. The testing process is automatically carried out in the sense that i) test cases focusing on critical parts of the cloud are automatically generated using the provided MRs; and ii) we use the MRs to automatically check the outputs generated by the execution of the test cases.
- We report on an experimental study where up to 450,000 test cases were automatically generated and executed for testing 27 different cloud configurations. After a careful analysis of resultant data, we can conclude that TEA-*Cloud* can be used to analyse non-trivial properties over cloud systems. In particular, all the injected faults were detected by the MRs provided.

The rest of the paper is structured as follows. Section 2 presents related work. Section 3 describes the motivation for integrating a simulation platform and MT in TEA-*Cloud*. Section 4 describes the architecture of TEA-*Cloud* and Section 5 shows how MT is integrated into TEA-*Cloud*. Section 6 describes the experiments performed using TEA-*Cloud*. Section 7 discusses the potential threats to the validity of the experiments. Finally, Section 8 presents our conclusions and some directions for future work.

2 RELATED WORK

In this section we review some existing work related to this paper. First, we review previous work on simulation and testing of cloud systems. Next, we briefly review the main characteristics of MT.

2.1 Testing of cloud systems

Cloud systems are built out of tens of thousands of commodity machines and a simple failure in the system may produce catastrophic consequences. Therefore, it is important to ensure the good functioning of these systems. As an example, in 2011 Amazon EC2 suffered an unexpected crash during network reconfiguration [15]. This crash affected more than 70 organisations, including *FourSquare*, the *New York Times*, *Quora* and *Reddit*, in some cases causing sites to be off-line for many hours.

Testing [1], [2] is the most widely used technique for checking the validity of complex systems. Therefore, testing should play an important role when deploying and configuring cloud systems.

There are several proposals for testing different parts of cloud systems, like symbolic execution [16], fault injection in the target system [17] and random testing [18], [19]. These approaches can be categorised into two major groups: testing *the cloud* and testing *in the cloud* (also known as cloud-based testing or cloud testing). The former targets the validation of applications, environments and infrastructures that are available on a cloud environment. This ensures the correct operation

of each part of the cloud system against the expectations of the cloud computing business model. Cloud testing, or Testing as a Service (in short, TaaS), involves using cloud infrastructures in testing products and services.

There are not many proposals for using a formal approach in testing cloud architectures, and we have to look either at proposals for formal testing in the distributed architecture [20], [21] or of systems with asynchronous communications [22], [23], [24] and approaches that focus on Web applications [25], [26]. Although a Web application can be executed in a cloud environment, and a cloud system is inherently distributed and communications are usually asynchronous, in these lines of work the underlying architecture of the cloud is not the target of the testing process. One of the few exceptions dealing with (formal) testing of cloud systems presents a formalism where a computing cloud is modelled as a graph, computing resources, such as services and intellectual property access rights, are attributes of a graph node, and the use of a resource is modelled as a predicate on an edge of the graph [27]. There are other research lines related to testing, like reliability and fault tolerance, amongst others, where we can find work focusing on analysing faults in cloud systems [28], [29].

Another approach for testing cloud systems executes a real virtual machine instead of a model that mimics its underlying behaviour. D-Cloud [30] is a software testing environment that manages virtual machines and includes fault injection capabilities. Basically, D-Cloud sets up a test environment on cloud resources using a given system configuration file and automatically executes several tests according to a given scenario. D-Cloud has been built on top of Eucalyptus and uses QEMU [31] to build virtual machines that simulate faults in parts of the hardware including disk, network and memory. PreFail [17] is a programmable and efficient failure testing framework where testers can express a variety of failure exploration policies, skip redundant fault-injection tests, run failure testing in parallel, and reduce the time to debug failed test runs. Unlike D-Cloud, which provides simulated *actual* faults, PreFail inserts a *failure surface* between the target system and the OS library.

As previously mentioned, there is very little work on formal approaches to testing cloud systems. Automated test generation for cloud systems has clear potential benefits such as quality improvement, the possibility of executing more tests in less time, and easy reuse of test-ware. However, there are significant costs associated with developing test automation, especially in dynamic customised environments [32]. As a result, testing is normally a manual activity.

In recent years, simulation has become a widely adopted loosely formalised approach for testing cloud systems. There are several advantages that make simulation very suitable for testing complex systems like, among others, the possibility of analyzing a system without requiring access to its underlying hardware architecture, the reproducibility of the experiments and the overall performance for executing the experiments, which can be improved by using parallel execution techniques. The developer builds a simulation model that imitates the behaviour of the target system and then different measures, like performance and cost, are gathered by running simulations. Researchers have designed cloud models and then performed ad-hoc testing by manually simulating different scenarios and comparing obtained results. Among the available simulation tools that can be used to model and simulate cloud computing environments are CloudAnalyst [33], CloudSim [34], [35], CloudSim-Plus [36], DCSim [37], EMUSIM [38], GreenCloud [39], GroudSim [40], iCanCloud [41], [42] and SimGrid [43], [44]. Although TEA-*Cloud* has been developed focusing on simulation to validate designs of cloud systems, ad-hoc test cases are not manually created, but large test suites are automatically generated using MT techniques instead.

2.2 Metamorphic testing

The complexity of a cloud system makes it very difficult to ensure that its behaviour is consistent with what the designers had in mind. Therefore, it is of the utmost importance to use sound engineering techniques to validate the behaviour of these systems. As already mentioned, *testing* is the most widely used method to increase the confidence in the correctness of systems. Testing has traditionally been a manual activity but there is increasing interest in the development of techniques to automate, as much as possible, the different testing activities. One important approach to test automation is to use *formal testing methods* [3]. These methods constitute a type of *Model-Based Testing* in which automation is based on either a complete model of the required behaviour of the system under test or on some specific aspect of this behaviour. In addition to supporting test automation, formal testing techniques were found to be significantly more cost effective than manual testing in an industrial study involving hundreds of testers [45].

Formal testing approaches usually assume the existence of an *oracle* to check whether the outputs returned by the system under test are the expected ones. However, often there is no oracle and it is then necessary to use alternative approaches to classical testing. This does not mean that we should not use formal methods at all, but that we need to combine formal approaches (in particular, use formal languages to design systems) with semi-formal ones for testing. In the frontier between formal and semi-formal approaches we find MT [8], [9]. MT is based on properties, called MRs [46], of related test inputs and the resultant test outputs. Given an initial test input x, typically a follow-up test case x' is produced such that some relationship between the outputs in response to x and x' should hold. For example, if we consider the trigonometric cosine function $\cos$ we have that $\cos(x) = \cos(-x)$; if we initially test with input 0.1

then we test with the follow-up test input -0.1 and check that the two outputs are the same. Experienced users are responsible for providing relevant MRs. MT is cost effective because the process of checking that the MRs hold can be automated.

MT has been used in very different application domains such as Web services [47], machine learning [48] and compilers [49], among others. Remarkably, MT was able to detect new faults [50], [51] in three out of seven programs in the Siemens suite [52], which has been studied in major software testing research projects for 20 years, and discover over one hundred faults [49] in two popular C compilers (GCC and LLVM). This versatility led us to use MT in our framework.

3 Motivation

In recent years cloud computing has gained significant attention due to its flexible and on-demand computing infrastructure. This interest has a significant impact in both the IT industry and the research community. On the one hand, leading companies such as IBM, Microsoft, Google and Amazon have spent valuable resources on cloud computing [53]. On the other hand, researchers are continuously developing tools and techniques to improve this emerging technology [54], [55].

Several factors make reasoning about the underlying architecture of a cloud system particularly challenging. First, cloud systems are very large and this hampers the analysis and study of these systems. Second, the resources of the cloud provided to end-users are *virtual* and this complicates analysis since different Virtual Machines (VMs) can be hosted in a single machine, sharing a resource among different users. Finally, we cannot oversee the vast number of users that are concurrently using a cloud system.

It can be difficult for researchers to access cloud structures. Researchers deal with two different types of clouds: public and private. Public clouds are characterised by their ease of use and ability to scale computing resources on demand. They provide the illusion of infinite resources and 24/7 availability. Hence, researchers are able to purchase virtual machines to perform experiments. Amazon EC2 is a good example of public cloud [56]. This platform allows users to pay only for the capacity that their applications actually need (pay-as-you-go model). However, this platform does not allow the low-level architecture to be configured for experiments. For instance, users do not know the network topology and researchers are not able to modify internal structures like hypervisors. Moreover, public clouds present significant variations in the overall performance depending on which machines are executed in the experiments [57]. In contrast, private clouds are operated solely within a single organisation. These clouds are typically much smaller than public clouds. Private clouds are often built in universities and research laboratories for research and teaching purposes. Therefore, researchers can fully customise and configure the underlying cloud architecture to perform experiments but even here there can be significant costs associated with running experiments. These factors motivate our interest in providing a tool for designing and testing cloud computing architectures that does not require access to a real cloud system.

When developing cloud systems it is important to have a tool that allows the developer to configure and test a wide range of scenarios. The proposed framework does not focus on a specific simulator, allowing any cloud simulation tool to be used to provide a model of the cloud under study and to simulate the required scenarios. A designed model is an abstraction of the underlying cloud system that contains the most relevant properties. This model can be fully parameterised, which provides enough flexibility to customise the configuration of the cloud to be simulated. Our interest in simulation was motivated by the following facts:

- Running simulations is cheaper than performing experiments in a real cloud. Renting machines from a public cloud requires a monetary investment, while executing a simulation requires a standard desktop computer.
- Simulation provides more flexibility. While in real clouds the users have to deal with the specific configuration of the system, simulation allows users to quickly set up a wide spectrum of configurations. These configurations may involve network topology, cost policies, hypervisors, etc.
- Scalability. In general, the number of VMs that a single user may rent in public clouds is limited. In contrast, in a simulation the number of machines can be fully customisable.
- Simulation models can be shared with other researchers in the community. Since the framework is Open Source, it can be shared with other researchers who can also modify it.

However, the use of simulation does entail some drawbacks. The main one is that simulation does not provide *real* data since we only simulate the performance of a cloud system during a given experiment, in contrast to executing the experiment in a cloud system. It is important to note that our work does not intend to replace experiments in real clouds. Instead, TEA-*Cloud* has been designed to help researchers to find, and then improve further, configurations that obtain better results and discard those that are invalid. However, the final stage of the research must consist of executing the corresponding experiments in a real cloud system. In addition, one of the main objectives of TEA-*Cloud* is to provide a methodology to systematically test the designed cloud systems. This is not trivial because the number of tests may range from a few hundred to millions, depending on the level of detail in the model of the cloud.

There are three main reasons for our interest to apply MT for checking the correctness of cloud systems. First, it has been shown that MT can overcome the

oracle problem in a range of scenarios and typically there will be no oracle when testing cloud systems. Second, the functionality of the proposed framework can be increased by adding new MRs. MRs can also be shared among different research groups. Third, it is possible to accelerate the execution of the testing process by concentrating the testing effort on a particular feature of the system by using specific MRs, targeting either a specific characteristic or a part of the system. This can be easily done because we will group MRs into sections, each section responsible for a specific feature of the cloud system and, therefore, we can use a reduced number of test cases for testing a specific part of the cloud.

In conclusion, MT is the most appropriate approach to validate cloud systems, especially since usually there is no oracle. Having an oracle is usually a prerequisite for model-based testing techniques [3], [4]. Let us illustrate this claim with a simple example. A typical cloud environment consists of thousands of physical machines, each of them containing a CPU, storage devices, memory and network interface(s), several configurations for virtual machines and a large number of users concurrently requesting resources to the cloud. Therefore, it is impossible to know beforehand the performance, for example in terms of time, of such a cloud when processing a specific workload (that is, we do not have an oracle). However, it is feasible to predict whether such a cloud system should be *better* or *worse* than another cloud system. For example, if we replace CPUs by faster/slower versions then we should obtain a better/worse performance in terms of time (even if we cannot predict the exact time values). Therefore, it is feasible to define MRs.

4 DESCRIPTION OF TEA-*Cloud*

This section describes the architecture of the TEA-*Cloud* framework. Basically, TEA-*Cloud* can be divided into three main parts: the catalogue of MRs, the testing engine, and simulation (see Figure 1).

Initially, an expert with deep knowledge in cloud computing systems designs a catalogue of MRs ①, which formally represents aspects of the behaviour of the cloud. Once the expert considers that the catalogue is complete, that is, the most relevant features of the cloud are accurately reflected in the properties of the MRs, each MR in the catalogue is coded by an expert programmer. The code representing each MR is included into TEA-*Cloud* using a hierarchical organisation. Let us emphasise that this architectural design makes it possible to easily and efficiently increase the functionality of TEA-*Cloud* by including new MRs into the system. It is important to remark that it is crucial to have an appropriate catalogue of MRs in order to carry out the testing process with precision and effectiveness. Also note that the cost required to provide accurate MRs does not only rely upon the effort associated with its development, but also requires an expert with deep knowledge on cloud systems.

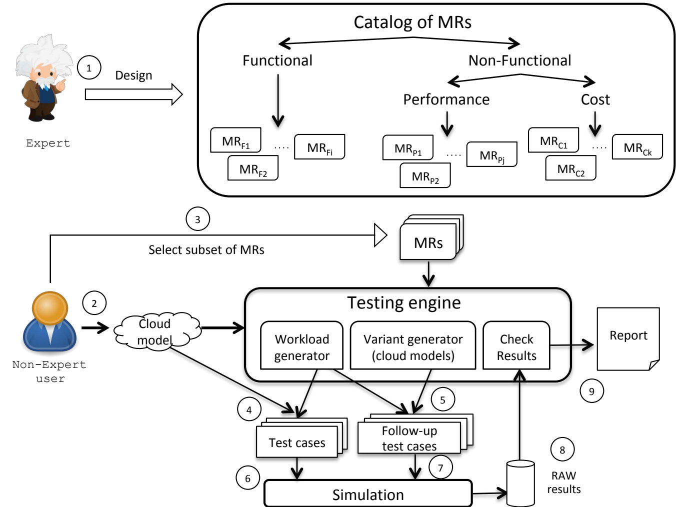


Fig. 1. Architecture of the TEA-Cloud framework

Users may create the cloud model to be tested and a workload ②. A cloud model is, essentially, the configuration of the components defining a cloud system. This configuration includes, among others, the network topology, aggregation of physical machines in racks, definition of physical machines, virtual machines and software pieces like managers and schedulers. A workload represents the operations that must be processed by the cloud. In essence, these operations consist of deploying the VMs requested by the users and executing applications over the VMs. Additionally, the user selects a subset of MRs, from the catalogue provided by TEA-*Cloud*, focusing on those parts of the system to be tested ③.

The test engine receives as input a cloud model, a workload and a set of MRs, previously selected by the user, in order to automatically test the cloud. Essentially, this module carries out the testing process. Hence, the following processes are automatically executed.

Initially, a collection of workloads is created. These workloads are automatically generated from the workload initially provided by the user. Once the workloads are created, the testing engine proceeds to generate the tests. First, a test suite is automatically generated by using both the cloud model generated by the user and the collection of workloads ④. A test case consists of a cloud model and a workload to be processed by the cloud. Second, a large collection of variants (clouds) are created by slightly modifying the cloud model provided by the user. Next, a large collection of follow-up test case is automatically created by combining the variants (clouds) and the workloads. Each test case, generated in ④, and each follow-up test case, generated in ⑤, must fulfill the input part of the MRs involved in the testing process. Third, all the test cases are executed over a simulation platform (see steps ⑥ and ⑦). TEA-*Cloud* does not require a specific simulator, making it possible to include any simulation platform, like CloudSim [34],

[35] or CloudSim-Plus [36], that simulates the behaviour of the cloud.

Next, the testing engine processes the provided results ⑧. This step is particularly relevant because the results corresponding to each test case are automatically checked without the intervention of a human. This is achieved by applying the chosen MRs to both the test cases and the results.

Finally, a report containing the result of the testing process is generated ⑨.

The concept of *variant* will be very important when defining our MRs since we will compare the results of different models, that is, the cloud model m provided by the user and a variant m'. In essence, a variant is generated by applying a slight modification over an original cloud model so that we create a *mutant*. In Section 4.1 we explain in detail how the variants are generated.

4.1 Formal modelling of cloud infrastructures

Each cloud system must provide a data center to allocate physical resources, like computing machines, storage servers, switches and communication networks. These hardware parts can be modelled using the simulation platform, which basically provides a repository that contains a collection of hardware components. Users can use and combine these to build customised models of physical machines, which rely on four main systems: CPU, memory, network and storage. In a typical cloud system, there are two different types of physical machines: computing nodes and storage servers. A computing node is a machine used to host one or several VMs. These VMs are provided to users that request cloud services, while storage servers are in charge of managing remote data access. First, we assume that there are partial orders over the sets of CPUs, memories, network components and storage systems. These partial orders indicate which entities provide better performance.

Definition 1. We represent a *cpu* as a pair $(speed, ncores)$, where $speed \in \mathbb{N}$ is the speed of *cpu*, measured in MIPS, and $ncores \in \mathbb{N}$ is the number of cores.

We denote by $\mathcal{CPU}$ the set of available processing systems (CPUs). We assume that there exists a partial order $\leq_{cpu} \subseteq \mathcal{CPU} \times \mathcal{CPU}$ such that $c_1 \leq_{cpu} c_2$ indicates that the CPU c_2 is preferable to c_1.

We represent a *memory* as a tuple $(size, rspeed, wspeed)$, where $size \in \mathbb{N}$ is the size of *memory*, measured in MBytes, and $rspeed, wspeed \in \mathbb{N}$ are the read and write speeds in Mbps, respectively

We denote by $\mathcal{MEM}$ the set of available memories. We assume that there exists a partial order $\leq_{mem} \subseteq \mathcal{MEM} \times \mathcal{MEM}$ such that $m_1 \leq_{mem} m_2$ indicates that the memory device m_2 is preferable to m_1.

We represent a *network* as a pair (bw, lat), where $bw \in \mathbb{N}$ is the bandwidth of *network*, measured in Mbps, and $lat \in \mathbb{N}$ is the latency measured in μs.

We denote by $\mathcal{NET}$ the set of available network interfaces. We assume that there exists a partial order $\leq_{net} \subseteq \mathcal{NET} \times \mathcal{NET}$ such that $n_1 \leq_{net} n_2$ indicates that the network interface n_2 is preferable to n_1.

We represent a *storage device* as a tuple $(size, rspeed, wspeed)$, where $size \in \mathbb{N}$ denotes the size of *storage device* in GBytes, and $rspeed, wspeed \in \mathbb{N}$ are the read and write speeds, measured in Mbps, respectively.

We denote by $\mathcal{STO}$ the set of available storage systems. We assume that there exists a partial order $\leq_{sto} \subseteq \mathcal{STO} \times \mathcal{STO}$ such that $s_1 \leq_{sto} s_2$ indicates that the storage device s_2 is preferable to s_1.

We represent a *switch* as a tuple $(cpu, bw, dela)$, where $cpu \in \mathcal{CPU}$ denotes the processing system of *switch*, $bw \in \mathbb{N}_+$ denotes the bandwidth of *switch*, measured in Mbps, and $dela \in \mathbb{N}_+$ is the average delay per packet of *switch* measured in microseconds.

We denote by $\mathcal{SWI}$ the set of available switches. We assume that there exists a partial order $\leq_{swi} \subseteq \mathcal{SWI} \times \mathcal{SWI}$ such that $s_1 \leq_{swi} s_2$ indicates that the switch s_2 is preferable to s_1. □

For instance, given two different storage devices, *disk1* $= (500, 110.2, 106.8)$ that models a Western Digital Black SATA3 HDD drive and *disk2* $= (500, 3500, 2500)$ that models a Samsung 970 Evo SDD drive, we can say that *disk2* is better than *disk1*, formally, *disk1* $\leq_{sto}$ *disk2*.

Next we introduce the formal definition of one of the basic components of a cloud system: the concept of physical machine.

Definition 2. A *physical machine* p is a tuple $(cpu, mem, net, sto, type)$, where $cpu \in \mathcal{CPU}$ denotes the CPU processor of p, $mem \in \mathcal{MEM}$ denotes the memory system of p, $net \in \mathcal{NET}$ denotes the network system of p, $sto \in \mathcal{STO}$ denotes the storage system of p, and $type \in \{stoSer, cmpNode\}$, where *stoSer* indicates that p is a storage server and $cmpNode$ indicates that p is a computing node. In order to access the different components of the tuple, we will use a subindex notation. For example, p_{cpu} denotes the first component of p, that is, its CPU. □

Next we present additional terminology regarding devices that connect physical machines.

Definition 3. A *hardware collection* is a set $H = P \cup S$, where P is a set of physical machines and S is a set of switches. We assume that P and S are disjoint.

Let $H = P \cup S$ be a hardware collection. A *data center* is a connected graph $\langle H, E \rangle$ where $E \subseteq ((S \times S) \cup (S \times P) \cup (P \times S))$ is a symmetric anti-reflexive relation denoting the set of network links between the elements of H. □

Since virtualisation facilitates the separation of physical and logical resources, it has became one of the major aspects in cloud computing environments. Hence, the behaviour of virtual resources needs to be appropriately simulated in order to obtain accurate results. A *virtual*

machine (VM) is a set of physical resources that users rent to execute their applications. VMs are intended to provide users with the functionality of a complete computing node.

Definition 4. A *virtual machine* v is a tuple $(cpu, p_c, mem, p_m, sto, p_s, c)$ where $cpu \in \mathcal{CPU}$ denotes the requested CPU, $mem \in \mathcal{MEM}$ denotes the requested memory device, $sto \in \mathcal{STO}$ denotes the requested storage system, $0 < p_c, p_m, p_s \leq 1$ denote, respectively, the percentage of the CPU, memory and storage system requested, and c denotes the cost to rent – for one hour – the virtual machine. □

For example, a virtual machine (*Intel i7 Quad-Core 3.4 Ghz*, 0.5, *(4 GB of RAM, 10667 Mbps)*, 0.3, *(500 GB disk, 960 Mbps read, 900 Mbps write)*, 0.1, 5) indicates that we would like to use a machine providing a CPU as least as good as an *Intel i7 Quad-Core* and having at least 50% of its use, providing *4 GB of RAM* or more with access rate of 10667 Mbps or better and having at least 30% of its use, and at least 10% of a storage system of *500 GB* or better (960 Mbps or better read, 900 Mbps or better write). Additionally, we are required to pay 5€ for each hour using this virtual machine. It is important to note that while the capacity of each virtualised device is exclusively used by each VM, the physical features of these devices, like bandwidth, are shared by all the VMs that use them.

From now on, the term *user* denotes a person who uses the TEA-*Cloud* framework while *tenant* refers to a person who purchases services of the modelled cloud in a simulated environment. Initially, each tenant has a budget. The amount and quality of each purchased VM directly depends on this budget. Basically, a tenant is defined by a set of purchased VMs, each of them purchased for a specific time-slot, and a set of applications that are executed in these VMs.

Definition 5. Let v be a VM. Given an application a, we denote by $a(v)$ the execution of the application a on the virtual machine v. We denote by $a_{\uparrow}(v)$ the successful execution of a on v, that is, a is completely executed without being aborted by the expiration of any time slot of v.

Let V be a set of VMs and A be a set of applications. A *tenant* t is a pair (V, A), where the first element of the pair represents the virtual machines purchased by t and the second element represents a set of applications that must be executed in V.

□

Finally, the cloud manager module is the master key of the simulation core. This module is directly linked to the resource manager module and to each VM in the cloud. The main objective of the cloud manager is to map the VMs requested by tenants to the available physical machines in the cloud. Thus, each cloud manager must implement its own mapping function. Also, this module can apply, in real-time, intelligent scheduling algorithms for optimising the trade-offs between cost and performance. An interesting feature of the proposed framework is that users of our framework are able to check models of single components, like hypervisors, CPUs and scheduling policies, to test both performance and functional aspects in different cloud environments.

Definition 6. Let V be a set of VMs and P be a set of physical machines. A *cloud manager* is a partial function $\Phi : V \rightarrow P$ that represents the assignment of a physical machine $p \in P$ to each virtual machine $v \in V$. This function must satisfy the following constraints:

- For all $v = (cpu, p_c, mem, p_m, sto, p_s, c) \in V$ we have that $cpu \leq_{cpu} \Phi(v)_{cpu}$, $mem \leq_{mem} \Phi(v)_{mem}$ and $sto \leq_{sto} \Phi(v)_{sto}$. That is, each virtual machine gets a physical machine having devices as least as good as the requested ones.
- For all $p \in P$ we have:
$$\sum_{\{v \in V \mid \Phi(v) = p\}} v_{p_c} \leq 1$$
- For all $p \in P$ we have:
$$\sum_{\{v \in V \mid \Phi(v) = p\}} v_{p_m} \leq 1$$
- For all $p \in P$ we have:
$$\sum_{\{v \in V \mid \Phi(v) = p\}} v_{p_s} \leq 1$$

Let D be a data center, V be a set of VMs and M be a cloud manager. A *cloud model* m is a tuple (D, V, M).

A *worklet* w is a pair (t, ts), where t is a tenant and ts is a timestamp representing the exact time when the tenant t arrives to the cloud. Thus, each worklet represents the execution of applications in the VMs requested by a tenant.

A workload ω is a sequence of worklets to be processed by the cloud system. We denote by $len_{ten}(\omega)$ the number of tenants in ω to be processed. We denote by $len_{VMs}(\omega)$ the total number of VMs requested by the tenants contained in ω.

The processing of a workload ω over a cloud model m is carried out by using simulation. Thus, we denote by $S(m, \omega)$ the simulation of an environment where the workload ω is executed over the cloud m. □

In Section 5 we introduce notation to access information contained in $S(m, \omega)$ such as time of execution, performance and cost.

4.2 Testing cloud computing systems

One of the most common problems for users when simulating programs is the difficulty of methodically testing the validity of their simulations. It is well-known that testing of software systems can take around 50% of the total budget of the project [1]. We are not aware of such figures, in terms of effort, for the case of simulation of cloud systems, but we expect similar numbers due

to the large number of components like disks, CPUs, networks and VMs, and the numerical computations of the simulator. It becomes impractical for end-users to manually predict the expected values and/or check the correctness of the reported outputs. Therefore, it would be desirable to use a methodology that reliably decides whether the testing process found an error. Unfortunately, testers usually adopt an *ad-hoc* strategy based on contrasting the results obtained from the simulator against real-world experiments. This strategy consists of providing a model of the system under test and then executing the same test in both environments, real world and simulated environment, to compare the obtained results. However, this strategy requires vast amounts of time and effort for even a very limited number of test cases.

When using conventional testing methods it is necessary to check whether the output(s) returned by the system under test are the expected ones or not. Schematically, let $\mathcal{S}$ be a system, I be the input domain and S be a test selection strategy. Let $\mathcal{T} = \{t_1, t_2, \ldots, t_n\} \subseteq I$ be the set of tests generated by using S. When these tests are sequentially applied to the system $\mathcal{S}$ we obtain a sequence of outputs $\mathcal{S}(t_1)$, $\mathcal{S}(t_2)$, …, $\mathcal{S}(t_n)$. Therefore, if we have a specification/oracle, called $\mathcal{O}$, then we find an error if there exists $t \in \mathcal{T}$ such that $\mathcal{S}(t) \neq \mathcal{O}(t)$. In general, we will not have an oracle and the most that we can do is to look for evidence that there has been a failure.

TEA-*Cloud* includes a testing engine whose main purpose is to automatically test a cloud model by using an approach inspired by MT. In TEA-*Cloud*, a single test case is a pair (m, ω) representing the processing of ω by the cloud m. Actually, this activity can be manually carried out by a user even without using the testing part of TEA-*Cloud*. The idea is to generate *follow-up test cases* from the original cloud under study, execute these new tests, $S(m'_1, \omega'_1), S(m'_2, \omega'_2), \ldots, S(m'_k, \omega'_k)$, and compare the obtained results. In essence, a follow-up test case consists of a variant model and a workload. Note that our variants are not *mutants* in the sense of *mutation testing*: our goal is not to *kill* the variants to decide the goodness of the considered test but to compare the different obtained results to detect a wrong or suboptimal behaviour of the original model. We will use MRs to compare these results. Next, we formally define the pattern of our relations.

Definition 7. Let m be a cloud model and m' be a variant of m. Let ω and ω' be workloads to be processed by m and m', respectively. A *metamorphic relation R* for m, m', ω and ω' is a set of 4-tuples

$$R = \left\{ \left(\begin{array}{c} (m,\omega), (m',\omega'), \\ S(m,\omega), S(m',\omega') \end{array} \right) \middle| \begin{array}{c} p_{in}((m,\omega),(m',\omega')) \\ \Downarrow \\ p_{out}(S(m,\omega), S(m',\omega')) \end{array} \right\}$$

where p_{in} is a relation over cloud models and workloads and p_{out} is a relation over the results provided by the simulation of the clouds for processing the workloads. □

If a tuple does not belong to a metamorphic relation, then we can say that we found an error. In other words, given a model m that we are validating and a workload ω, if we have that for some variant model m' and workload ω' the tuple $((m,\omega),(m',\omega'),S(m,\omega),S(m',\omega'))$ does not belong to *MR* then we know that there has been an unexpected behaviour. For example, it may happen that we expect the performance of m to always be better than the one corresponding to m'. However, for our chosen workload ω, the simulation to process ω over m and m' shows otherwise. In the next section we present some of the MRs that are included in our tool and how they are classified according to the type of property that they validate (performance, functional and cost). In addition, users of our framework can define new MRs (it will be their responsibility to ensure the soundness of those relations).

TEA-*Cloud* contains a catalogue of MRs that represents the underlying correct behaviour of a cloud. Thus, users can easily create, remove and edit current MRs from the repository. This is a powerful feature of TEA-*Cloud* because users can exchange MRs to easily and quickly improve their testing processes.

The testing methodology proposed in this paper validates whether a given cloud model works properly, when processing each tenant contained in a given workload, by requesting virtual resources and launching applications. The following schema shows the methodological steps used in TEA-*Cloud*:

- S-1 Initially, the expert designs and includes MRs into the catalogue (see step labelled by ① in Figure 1).
- S-2 A cloud model m must be provided by the user ②.
- S-3 Depending of the features to be tested on m, a subset $\mathcal{R}$ of the MR catalogue ③ must be selected.
- S-4 The testing engine ④ automatically generates:
  - S-4.1 A set of workloads $\mathcal{W}$.
  - S-4.2 A set of test cases $\mathcal{T} = \{(m,\omega) | \omega \in \mathcal{W}\}$.
- S-5 For each $R \in \mathcal{R}$, the testing engine ⑤ uses $\mathcal{T}$ and $\mathcal{W}$ to automatically generate a set of follow-up test cases $\mathcal{F} = \{(m',\omega') | \omega' \in \mathcal{W}\}$:
  - S-5.1 For each $t \in \mathcal{T}$:
    - S-5.1.1 For each $f \in \mathcal{F}$:
      - S-5.1.1.1 Execute $S(t)$ ⑥.
      - S-5.1.1.2 Execute $S(f)$ ⑦.
      - S-5.1.1.3 If $(t, f, S(t), S(f)) \notin R$ then a log indicating the relation that is not passed by the cloud model m is stored; otherwise, the log shows that the current experiment did not find a fault ⑧.
- S-6 The logs generated from [S-5] are processed by the testing engine, which generates a report showing the overall results ⑨.
- S-7 End of the testing process.

5 METAMORPHIC TESTING FOR CLOUD COMPUTING SYSTEMS

TEA-*Cloud* provides a catalogue of MRs that can be easily managed by users. These relations are the core engine of the testing process. Therefore, the user chooses some MRs on the basis of the features of the cloud system that are to be tested (step S-3 of the algorithm given in the previous section). For instance, if a user is interested in testing the performance of a given cloud system, then this user should select those MRs dealing with performance. In order to facilitate this process, the MRs incorporated in TEA-*Cloud* are grouped into three different sets, where each set represents an aspect of the system to be tested.

- *Performance*. This set contains those relations dealing with the performance of a given system. Depending on the part of the system to be tested, this performance is measured in Mbps, MIPS or execution time.
- *Functional*. This set contains those relations that check the underlying behaviour of a system.
- *Cost*. This set contains those relations that check the restrictions regarding the required monetary cost for renting virtual machines in the cloud.

Next, we introduce some notation to formally specify different aspects concerning performance and cost of a cloud system and its parts.

Definition 8. Let $m = (D, V, M)$ be a cloud model consisting of a data center D, a set of VMs V and a cloud manager M, where $D = \langle H, E \rangle$ and $H = P \cup S$ (see Definition 3).

Let $p = (cpu, mem, net, sto, type)$ be a physical machine. We denote by $\delta(p_x) \in \mathbb{R}_+$ the theoretical performance peak of the component x of p. Specifically, $\delta(p_{cpu})$ denotes CPU performance, measured in MIPS, $\delta(p_{mem})$ denotes memory performance, measured in Mbps, $\delta(p_{net})$ denotes network performance, measured in Mbps, and $\delta(p_{sto})$ denotes I/O performance, also measured in Mbps. Consequently, if given two devices x_1 and x_2 of the same type that satisfy the partial order described in Definition 1, for example x_2 is preferable to x_1, and two different machines p and p' that only differ in this component, then we must have $\delta(p_{x_1}) \leq \delta(p'_{x_2})$.

We denote by $nCores(m)$ the aggregated number of CPU cores provided by the physical machines of m.

We denote by $\Omega(m, \omega)$ the total monetary cost required to process all the tenants contained in ω over the cloud m.

□

The theoretical performance peak of a given device is usually provided by its manufacturer. Basically, this performance heavily relies on the physical characteristics of the hardware device. However, the maximum performance obtained when a hardware device is exploited in a real system rarely reaches the theoretical peak provided by its manufacturer.

The estimation of the real performance and cost in cloud systems is a difficult and complex task, which requires sophisticated techniques for representing the behaviour of each component of the system to be analysed. Due to the complex underlying architecture of cloud systems, there are many elements such as network bottlenecks, high number of users using shared resources concurrently and scheduling policies for managing available resources, just to name a few, that have a direct impact on the overall system performance. In order to obtain accurate estimates, all these elements must be taken into account. Therefore, we use simulation, where both the physical characteristics and the underlying behaviour of each component can be individually modelled with the objective of building fully customisable data centers.

In order to compute the previously described metrics in a cloud system, the simulator takes as input two different elements: a cloud model m that represents the infrastructure of a cloud and a workload ω that represents the tenants to be processed by m. Next, we introduce notation to represent different measures of executing the workload ω over the cloud model m using simulation:

- $sim_{\Omega}(m, \omega) \in \mathbb{R}_+$ denotes the total cost required to process the workload ω in the cloud m.
- $sim_{\Gamma}(m, \omega) \in \mathbb{R}_+$ denotes the time required to process the workload ω in the cloud m.
- $sim_{\uparrow Users}(m, \omega)$ denotes the number of *successfully processed* tenants of ω over m. For each worklet $(t, ts) \in \omega$, where $t = (V, A)$, we say that the tenant t has been successfully processed by the cloud m when every application $a \in A$ is completely executed over V.
- $sim_{\uparrow VMs}(m, \omega)$ denotes the number of *successfully deployed* VMs of ω over m, where $m = (D, V, \Phi)$. We let P be the set of physical machines contained in D. For each worklet $((V', A), ts) \in \omega$, we say that a virtual machine $v \in V'$ has been successfully deployed in m if there exists p such that $\Phi(v) = p \in P$, that is, v has been successfully assigned to an available physical machine of m.

It is important to observe that a proper design of the MRs is key to successfully applying MT. Let us illustrate this with a simple example, where we use the following MR to represent the behaviour of a cloud system: "Given two cloud models m and m', and a workload ω, if the number of physical machines of m is greater than the number of physical machines of m', then m requires less time than m' to process ω." In this case, we apply the workload ω over both clouds, m and m', and observe the performance provided. This is a clear example of a *bad* metamorphic relation. Specifically, if m' uses powerful CPUs with a large number of CPU cores, in particular, better than the CPUs used in m, then m' may outperform m. Therefore, a testing process using this MR might show inaccurate results.

Next, we enumerate our most relevant MRs. For the sake of clarity, we provide a brief description of each MR together with the corresponding formal definition. In order to simplify the description of each MR, we assume that the components of the model m that are not reflected in the MR, remain unmodified in the generated variant m'. We use two cloud models and two workloads, denoted by $m = (D, V, M)$, $m' = (D', V', M')$, ω and ω', respectively, where m represents the original model provided by the user, m' represents a variant model automatically generated by the testing engine and ω and ω' are two workloads to be processed by the cloud models. Initially, the workloads are arbitrary in the sense that no sorting criterion has been applied to them. Finally, we let $D = \langle H, E\rangle$, $D' = \langle H', E'\rangle$, $H = P \cup S$ and $H' = P' \cup S'$.

R_{P1}: Given two cloud models m and m' and two workloads ω and ω', if ω and ω' are equal and the theoretical aggregated CPU performance of m is greater than the theoretical aggregated CPU performance of m' (and all other aspects are the same), then the time required to execute ω over m must be less than or equal to the time required to execute ω' over m' or the number of tenants of ω that are successfully processed over m must be greater than the number of tenants in ω' that are successfully processed over m'.

$$R_{P1} = \left\{ \begin{pmatrix} (m,\omega), \\ (m',\omega'), \\ S(m,\omega), \\ S(m',\omega') \end{pmatrix} \middle| \begin{array}{c} \omega = \omega' \\ \wedge \\ \Delta(m_{cpu}) > \Delta(m'_{cpu}) \\ \Downarrow \\ sim_\Gamma((m,\omega)) \leq sim_\Gamma((m',\omega')) \\ \vee \\ sim_{\uparrow Users}(m,\omega) > \\ sim_{\uparrow Users}(m',\omega') \end{array} \right\}$$

R_{P2}: Given two cloud models m and m' and two workloads ω and ω', if ω and ω' are equal, and the number of switches used in D and D' is the same, and the switches used in D are preferable to the switches used in D', then the time required to execute ω over m must be less than the time required to execute ω' over m' or the number of tenants of ω that are successfully processed over m must be greater than the number of tenants in ω' that are successfully processed over m'.

$$R_{P2} = \left\{ \begin{pmatrix} (m,\omega), \\ (m',\omega'), \\ S(m,\omega), \\ S(m',\omega') \end{pmatrix} \middle| \begin{array}{c} \omega = \omega' \wedge |S| = |S'| \\ \wedge \\ \forall s \in S, s' \in S' : s' \leq_{swi} s \\ \Downarrow \\ sim_\Gamma((m,\omega)) < sim_\Gamma((m',\omega')) \\ \vee \\ sim_{\uparrow Users}(m,\omega) > \\ sim_{\uparrow Users}(m',\omega') \end{array} \right\}$$

R_{P3}: Given two cloud models m and m' and two workloads ω and ω', if ω and ω' are equal, and the number of CPU cores provided by m is greater than the number of CPU cores provided by m', then the time required to execute ω over m must be less than or equal to the time required to execute ω' over m' and the number of VMs that are successfully deployed when m executes ω must be greater than or equal to the number of VMs that are successfully deployed when m' executes ω', or the number of tenants of ω that are successfully processed over m must be greater than the number of tenants in ω' that are successfully processed over m'.

$$R_{P3} = \left\{ \begin{pmatrix} (m,\omega), \\ (m',\omega'), \\ S(m,\omega), \\ S(m',\omega') \end{pmatrix} \middle| \begin{array}{c} \omega = \omega' \\ \wedge \\ nCores(m) > nCores(m') \\ \Downarrow \\ \begin{pmatrix} sim_\Gamma((m,\omega)) \leq sim_\Gamma((m',\omega')) \\ \wedge \\ sim_{\uparrow VMs}(m,\omega) \geq \\ sim_{\uparrow VMs}(m',\omega') \end{pmatrix} \\ \vee \\ sim_{\uparrow Users}(m,\omega) > \\ sim_{\uparrow Users}(m',\omega') \end{array} \right\}$$

R_{P4}: Given two cloud models m and m' and two workloads ω and ω', if ω and ω' are equal, and the aggregated theoretical network performance of m is greater than the aggregated theoretical network performance of m', and the number of CPU cores of m is greater than or equal to the number of CPU cores of m', then the time required to execute ω over m must be less than the time required to execute ω' over m' or the number of tenants of ω that are successfully processed over m must be greater than the number of tenants in ω' that are successfully processed over m'.

$$R_{P4} = \left\{ \begin{pmatrix} (m,\omega), \\ (m',\omega'), \\ S(m,\omega), \\ S(m',\omega') \end{pmatrix} \middle| \begin{array}{c} \omega = \omega' \\ \wedge \\ \Delta(m_{net}) \geq \Delta(m'_{net}) \\ \wedge \\ nCores(m) \geq nCores(m') \\ \Downarrow \\ sim_\Gamma((m,\omega)) < sim_\Gamma((m',\omega')) \\ \vee \\ sim_{\uparrow Users}(m,\omega) > \\ sim_{\uparrow Users}(m',\omega') \end{array} \right\}$$

We now comment on why these relations were chosen and the types of faults that they might find. This first group of relations, namely R_{P1}, R_{P2}, R_{P3} and R_{P4}, focus on performance. These relation are designed to locate faults in cloud systems where we obtain the expected behaviour but in an inefficient way. In these relations, m represents the cloud under study while m' represents the variant cloud automatically generated for the follow-up test case. Hence, R_{P1} aims to detect faults in clouds that use powerful CPUs and provide a lower performance than expected. We detect an error when m', using slower CPUs than m, provides better results. In the case of R_{P2}, we are interested in checking the performance of the

networking subsystem and, therefore, we detect faults when m' is using a slow – or poorly configured – network and provides better results than m, which uses a fast and properly configured network. R_{P3} focuses on the number of CPU cores of the studied cloud. For this, we detect an error when m' contains fewer CPU cores than m and is able to allocate more VMs. Finally, R_{P4} combines the number of CPU cores and the cloud network. In this case, we study the total number of successfully processed users and the total execution time, which must be better in the cloud providing the better resources.

The second group of relations deals with functional properties.

R_{F1}: Given two cloud models m and m' and two workloads ω and ω', if ω' is a permutation of ω built after applying a sorting criterion to ω, denoted by $sort(\omega)$, that produces a chronologically sorted sequence $\omega' = ((t_1, ts_1), (t_2, ts_2), \ldots, (t_n, ts_n))$ such that for all $1 \leq i < n$ we have $ts_i \leq ts_{i+1}$, and m and m' are equal, then the number of tenants of ω' that are successfully processed over m must be equal to the number of tenants in ω.

$$R_{F1} = \left\{ \begin{pmatrix} (m, \omega), \\ (m', \omega'), \\ S(m, \omega), \\ S(m', \omega') \end{pmatrix} \middle| \begin{array}{c} sort(\omega) = \omega' \wedge m = m' \\ \Downarrow \\ sim_{\uparrow Users}(m, \omega') = \\ sim_{\uparrow Users}(m', \omega') = \\ len_{ten}(\omega) \end{array} \right\}$$

R_{F2}: Given two cloud models m and m' and two workloads ω and ω', if $\omega' = sort(\omega)$ and m and m' are equal, then the number of VMs that are successfully deployed when m executes ω' must be equal to the total number of VMs requested by the tenants contained in ω.

$$R_{F2} = \left\{ \begin{pmatrix} (m, \omega), \\ (m', \omega'), \\ S(m, \omega), \\ S(m', \omega') \end{pmatrix} \middle| \begin{array}{c} sort(\omega) = \omega' \wedge m = m' \\ \Downarrow \\ sim_{\uparrow VMs}(m, \omega') = \\ sim_{\uparrow VMs}(m', \omega') = \\ len_{\text{VMs}}(\omega) \end{array} \right\}$$

R_{F3}: Given two cloud models m and m' and two workloads ω and ω', if ω contains ω' and the number of CPU cores provided by m is less than or equal to the number of CPU cores provided by m', then the number of VMs that are successfully deployed when m executes ω must be less than or equal to the number of VMs that are successfully deployed when m' executes ω'.

$$R_{F3} = \left\{ \begin{pmatrix} (m, \omega), \\ (m', \omega'), \\ S(m, \omega), \\ S(m', \omega') \end{pmatrix} \middle| \begin{array}{c} \omega' \subseteq \omega \\ \wedge \\ nCores(m) \leq nCores(m') \\ \Downarrow \\ sim_{\uparrow VMs}(m, \omega) \leq \\ sim_{\uparrow VMs}(m', \omega') \end{array} \right\}$$

This second group of relations, namely R_{F1}, R_{F2} and R_{F3}, has been designed to check the functional properties of the cloud. In these relations, ω represents the workload processed by the cloud under study m and ω' is the workload processed by the variant cloud m'. Our first relation, R_{F1}, considers a single cloud system that processes two different workloads ω and ω', where in both cases the cloud processes the same users but in a different order. We detect an error in m if the number of processed users differs when processing ω and ω'. Similarly, R_{F2} focuses on the number of successfully deployed VMs. In this case, we detect an error when the number of VMs processed by the cloud is not the same. Finally, we are interested in checking the number of successfully deployed VMs by the cloud m processing ω, when compared with m' processing ω', when ω' is a subset of ω and also m' has more CPUs than m. In this case, we find an error when the number of VMs deployed by m' is greater than the number of VMs deployed by m.

Our last group of relations deals with properties related to costs.

R_{C1}: Given two cloud models m and m' and two workloads ω and ω', if m and m' are equal and ω' contains ω, then the total cost required for processing ω over m must be less than or equal to the total cost required for processing ω' over m' or the number of tenants of ω that are successfully processed over m must be greater than the number of tenants in ω' that are successfully processed over m'.

$$R_{C1} = \left\{ \begin{pmatrix} (m, \omega), \\ (m', \omega'), \\ S(m, \omega), \\ S(m', \omega') \end{pmatrix} \middle| \begin{array}{c} m = m' \wedge \omega \subseteq \omega' \\ \Downarrow \\ sim_{\Omega}(m, \omega) \leq \\ sim_{\Omega}(m', \omega') \\ \vee \\ sim_{\uparrow Users}(m, \omega) > \\ sim_{\uparrow Users}(m', \omega') \end{array} \right\}$$

R_{C2}: Given two cloud models m and m' and two workloads ω and ω', if m and m' are not the same, and ω and ω' are equal, then the total cost required for processing ω over m must be less than or equal to the total cost required for processing ω' over m' or the number of tenants of ω that are successfully processed over m must be greater than the number of tenants in ω' that are successfully processed over m'.

$$R_{C2} = \left\{ \begin{pmatrix} (m, \omega), \\ (m', \omega'), \\ S(m, \omega), \\ S(m', \omega') \end{pmatrix} \middle| \begin{array}{c} m \neq m' \wedge \omega = \omega' \\ \Downarrow \\ sim_{\Omega}(m, \omega) \leq \\ sim_{\Omega}(m', \omega') \\ \vee \\ sim_{\uparrow Users}(m, \omega) > \\ sim_{\uparrow Users}(m', \omega') \end{array} \right\}$$

R_{C3}: Given two cloud models m and m' and two workloads ω and ω', if m and m' are equal, and the

workload ω' is generated by concatenating the workload ω and the tenant t, then the total cost required for processing ω over m must be less than the total cost required for processing ω' over m'.

$$R_{C3} = \left\{ \left(\begin{array}{c} (m,\omega), \\ (m',\omega'), \\ S(m,\omega), \\ S(m',\omega') \end{array} \right) \middle| \begin{array}{c} m = m' \wedge \omega' = \omega \cdot t \\ \Downarrow \\ sim_{\Omega}(m,\omega) < \\ sim_{\Omega}(m',\omega') \end{array} \right\}$$

The last group of relations, namely R_{C1}, R_{C2} and R_{C3}, deals with monetary costs and focuses on the cost of renting the VMs requested by the users. The first relation, R_{C1}, locates an error in the cloud m when the cost associated with process ω is higher than the cost for processing ω' and ω is included in ω' (consequently, it should request fewer VMs). The idea is to establish a relation between the users in the workload and the cost related to process them. Similarly, R_{C2} finds an error in the cloud m while processing ω if there exists a cloud m' that processes more users than m and at lower cost. Finally R_{C3} establishes a relation between two workloads processed by a single cloud. In this case, ω' is built by including a new user into ω. Intuitively, the cost required to process ω should be lower than the cost to process ω'; otherwise, this relation locates an error in the cloud.

6 EMPIRICAL STUDY

In this section we describe an empirical study used to evaluate the applicability and usefulness of TEA-*Cloud*. In contrast to our previous work [14], where only one faulty cloud was analysed using three MRs and we had a very limited number of test cases, using our current framework we were able to analyse the effectiveness of a complete catalogue of MRs to check a wide range of cloud configurations. Also, since we used the cloudSimPlus simulator [36], it was possible to use large test suites during the testing process. For the sake of clarity, this study has been divided into different subsections. The first subsection gives the research questions addressed. Next, Section 6.2 describes the experimental settings in detail, explaining, among other aspects, the criteria used to select a cloud simulator for this study, the modelling of the source cloud models and the generation of the workloads. Section 6.3 presents the experiment performed and the results of this are given in Section 6.4. Finally, Section 6.5 provides a discussion regarding the obtained results and what they tell us about the research questions.

6.1 Research Questions

Ideally, we would like a testing technique to have the following two properties.

1) It is *valid*: testing is unlikely to reject/fail a correct system.
2) It is *effective*: testing is likely to reject/fail a faulty system.

The research questions correspond to these two properties.

Research Question 1. *Is the proposed technique* valid*, in the sense that it tends not to reject correct cloud systems?*

MT can be applied to test complex systems by using MRs that state properties of the system under test. In our case, the testing of cloud systems is especially challenging due to, among other aspects, the size of the system, the large number of inter-related parameters to model the cloud and the concurrent access by many users.

In our study, we were interested in investigating the suitability of applying MT for testing such cloud systems, when using the catalogue of MRs given. The first research questions was motivated by the fact that we want correct cloud systems to pass the tests.

Research Question 2. *Is the proposed technique* effective*, in the sense that it tends to detect faulty cloud systems?*

Currently, MT has been successfully applied to test a wide variety of systems such as Web services [47], machine learning [48] and compilers [49]. This last work is especially relevant because the authors discovered over one hundred faults in two popular C compilers (GCC and LLVM). In our study, we were interested in analysing how effective the MRs are in terms of detecting faults. In order to answer this question, we used mutation testing techniques to artificially inject faults into a cloud. We applied the tests to the *faulty* clouds and checked whether the MRs were satisfied.

6.2 Experimental settings

This section describes the experimental settings. First, we discuss the selection of the cloud simulator used to execute the experiments. Next, we describe the modelling of the source cloud models and how the workloads were generated.

6.2.1 Selection of the cloud simulator

During the last decade, simulation tools have been widely adopted by the research community. However, the large number of possible alternatives requires, in most cases, an additional effort on the part of the user to select the most appropriate simulator for carrying out the experiments. Generally, each simulator focuses on one - or several - features of the cloud like, among others, storage, virtualisation, SLAs and energy consumption. To the best of our knowledge, there is no simulator that fully simulates all the features of a cloud environment and, therefore, one or several simulation tools must be used to properly cover the requirements of the user for simulating the system under test.

In order to select the most appropriate simulator for this study, we first focused our efforts on analysing

existing surveys of cloud simulation tools [58], [59], [60]. These surveys provide a comprehensive study of major cloud simulators by highlighting their important features and analysing their strengths and weaknesses. After a careful analysis, we selected six well-known cloud simulators that are widely used by the research community: DCSim [37], GreenCloud [39], SimGrid [43], iCanCloud [41], [42], CloudSim [34] and CloudSim-Plus [36].

DCSim [37], also known as *The Data Centre Simulator*, is a Java extensible simulation framework for simulating a data center hosting. In essence, DCSim focuses on the IaaS layer for providing services to multiple tenants and supports VM management operations, such as VM live migration and replication. However, DCSim lacks a communication model. GreenCloud [39] is an open-source tool for simulating data centers focusing on data communication and energy cost in cloud computing systems. GreenCloud lacks flexibility for modelling other systems, like storage or user management. SimGrid [43] is a tool for simulating algorithms and distributed applications in distributed computing platforms. The resources are modelled by their latency and service rate, and the topology is configurable by the users. The main limitation of SimGrid is the difficulty of generating variants and follow-up test cases. iCanCloud [41] models and simulates cloud computing systems by providing different functionalities like resource provisioning and user management. Additionally, the framework *E-mc*^{2} [42] can be used for analysing energy consumption. Although iCanCloud provides models for the major part of the underlying cloud architecture, this simulator requires a high computational cost, which is reflected in long executions. CloudSim [34] is an extensible and open-source Java simulator, which enables modelling of cloud computing systems and application provisioning environments. CloudSim is considered the *de facto* standard cloud simulation platform due to its capabilities for simulating cloud systems, such as VM allocation and provisioning, energy consumption, federated clouds and the possibility to model different types of clouds like public, private, hybrid and multi-cloud environments. One of the key features of CloudSim is that it is possibile to include new functionality using extensions. CloudSim-Plus [36] is based on CloudSim and seeks to improve several engineering aspects, such as maintainability, reusability and extensibility.

After a careful study, we decided that cloudSim-Plus was the most appropriate simulation tool for this study. First, CloudSim-Plus is built upon CloudSim, which allows new functionalities to be added using *extensions*. Thus, if new MRs are required to represent additional features, these extensions can be used to deal with them. Second, CloudSim-Plus enhances CloudSim because it provides the flexibility to model a wider spectrum of configurations, which are adequate for creating follow-up test cases. Third, this simulator has an active community and it is currently maintained by its development team.

6.2.2 Source cloud models

In TEA-*Cloud*, a cloud model m is a tuple (D, V, M), where D is a data center containing the hardware resources, V is the set of virtual machines offered by the cloud and M is a cloud manager that manages the resource provisioning.

In this study we designed 9 different cloud models to be tested. These different cloud models used the same set of VMs V and three homogeneous data centers, namely D_m, D_s and D_l, which provided, respectively, 576, 320 and 1088 physical machines. The main parameters of D for modelling the data center are depicted as follows:

- Number of physical machines: 576 / 320 / 1088
  - # Computing nodes: 512 / 256 / 1024
  - # Storage nodes: 64
- Physical machine features:
  - RAM memory: 16 GB
  - Storage capacity: 500 GB
  - CPU: 4 cores @ 84000 MIPS
- Network: Ethernet 1 Gbps

The second piece to be modelled is the configuration of each VM offered by the cloud. In this case, we provided three different configurations of VMs (see Table 1).

TABLE 1
Modelling of different configurations of VM

Type	CPU cores	Memory	Storage
VM_{small}	1 core	1 GB	100 GB
VM_{medium}	2 cores	2 GB	250 GB
VM_{large}	4 cores	4 GB	500 GB

In order to conclude the modelling of each cloud model, a cloud manager must be provided. Basically, a cloud manager is a piece of software in charge of allocating the VMs requested by users to physical machines with available resources. In these experiments we used three different algorithms for allocating VMs.

- *Best-fit*. The cloud manager performs a search, using a criterion that minimises the fragmentation of CPU cores among the list of available physical machines in the cloud, that is, the physical machine with the fewest available CPU cores that are enough for deploying the requested VM is selected.
- *First-fit*. The cloud manager performs a linear search among the list of available physical machines in the cloud. The first physical machine having suitable resources to deploy the requested VM is selected.
- *Worst-fit*. The cloud manager performs a search among the list of available physical machines in the cloud using a criterion that maximises fragmentation, that is, the physical machine with most available CPU cores is selected.

Using these parameters, we constructed nine cloud models (see Table 2) by combining different cloud manager algorithms, that is, *Best-fit*, *First-fit* and *Worst-fit*, and three different data centers supporting the cloud. Later we describe three different policies, implemented by hypervisors, for mapping the execution of an application to a VM. There will therefore be 27 cloud configurations, each defined by a combination of a cloud model and a type of hypervisor.

TABLE 2
Cloud models

Cloud model	Data Center	Cloud Manager
m^m_{best}	D_m	Best-fit
m^m_{first}	D_m	First-fit
m^m_{worst}	D_m	Worst-fit
m^s_{best}	D_s	Best-fit
m^s_{first}	D_s	First-fit
m^s_{worst}	D_s	Worst-fit
m^l_{best}	D_l	Best-fit
m^l_{first}	D_l	First-fit
m^l_{worst}	D_l	Worst-fit

6.2.3 *Generation of the workload*

In order to analyse the behaviour of a cloud, a workload must be executed. We consider that a workload is, essentially, a sequence of tenants requesting VMs to the cloud for executing applications. Each tenant is given by a pair (V, A), where the first element represents the rented VMs and A represents the applications to be executed.

In order to provide accurate configurations for modelling VMs, we used models inspired by the VMs provided by Amazon EC2 [56]. These VMs execute different applications requested by the users. Each application was represented as an instance of a `Cloudlet` instance in the cloudSim-Plus simulator. Thus, we used a list of `Cloudlet` instances to represent the applications that are executed over each VM requested by the user. Specifically, each `Cloudlet` executes 1000 MIs, reads a file of 50 MB and writes to disk 75 MB of data. The interval between the execution of two consecutive applications was computed by using a `Uniform(10 ms, 33 s)` distribution.

Table 3 shows the modelling of four types of tenant, each one representing a group of users in the cloud that have a similar behaviour. The configuration of this behaviour can be set by using two parameters: the applications to be executed and the VMs deployed by the tenant. The first row of this table represents the number of application instances requested by each type of tenant. These are followed by three rows that represent the number of VMs rented by each type of tenant.

The idea was to represent the behaviour of different users, each one requesting different resources from the cloud, to generate a heterogeneous workload to be processed by the clouds under test.

TABLE 3
Modelling of different types of tenants

Configuration \ Tenant	t_A	t_B	t_C	t_D
# App instances	5	10	1	1
VM_{small}	5	0	0	0
VM_{med}	0	5	2	50
VM_{large}	0	5	0	0

We generated a workload using the configuration depicted in Table 3. This workload contains a total of 1024 tenants that are distributed as follows:

$$\omega = (512 \times t_A, 256 \times t_B, 128 \times t_C, 128 \times t_D)$$

6.3 Checking the validity of the MRs

This section describes the experiments used to assess the validity of the MRs provided and so address the first research question. Specifically, we expect a correct cloud system, or a simulation of a correct cloud system, to satisfy the MRs; if this is not the case then the failure of an MR might not indicate a fault.

In order to carry out this experiment, three different cloud models were used: m^m_{best}, m^m_{first}, and m^m_{worst}. Additionally, we used three different policies for mapping the execution of each application into the VMs. Typically, the hypervisor implements a policy for this task, which allocates one, or more, CPU cores from physical machines to deploy a VM, making it possible to share different CPU cores to execute multiple VMs. Next, we describe these policies:

- *Space-shared*: This policy allocates one, or several, CPU cores from a physical machine to a VM. In this case, sharing CPU cores is not allowed and, therefore, the allocated CPU cores will be used until the VM finishes running. In those cases where there are not enough available CPU cores as required by a VM, or where the available CPU cores do not have enough capacity, the allocation fails.
- *Time-shared*: This policy allocates VMs to physical CPUs using a fraction of the MIPS capacity of the physical CPU cores. Let us illustrate this with an example. Let us suppose that a VM requests a virtual CPU with computing power of 800 MIPS but, in this case, there is no physical machine with a CPU of such capacity. However, this policy is able to allocate these 800 MIPS among several physical CPUs, for instance, by allocating 500 MIPS to a CPU core and 300 MIPS to a different CPU core.
- *Time-shared oversubscription*: This policy allows the allocation, into a physical machine, of those VMs that require more CPU power than is currently available. If the physical machine has, at least, the number of requested CPU cores, then the VM is deployed in the physical machine. However, the VM only uses the real computing power provided by the available physical CPUs. This particular case

is known as *over-subscription*, which results in performance degradation because fewer MIPS may be allocated than required by a VM.

In order to check the validity of the MRs, we created three source test cases using the cloud models m^m_{best}, m^m_{first}, and m^m_{worst}, and the workload generated in Section 6.2.3. For each source test case, 10,000 different follow-up test cases were automatically generated. The idea was to execute each follow-up test case and then check if the obtained results satisfy the MRs. It is important to mention that the objective was to check whether the MRs properly represent the behaviour of the cloud system under test. Therefore, this experiment did not focus on locating faults.

We assume that an MR fully validates a feature of the cloud when all the follow-up test cases satisfy the MR. This can be identified by a value equal to 100 in Table 4, which shows the results of the experiments. As the table shows, R_{P1}, R_{P2}, R_{P3}, R_{P4}, R_{C1} and R_{C3} were satisfied by all the test cases. On the one hand, these results provide evidence that these MRs are valid; they appropriately reflect the expected (correct) behaviour of the cloud. On the other hand, the results also show that the CloudSim-Plus [36] simulator is indeed accurate because it was able to precisely represent the real behaviour of the cloud under study.

In the rest of the cases we identify two other possible scenarios: none of the follow-up test cases satisfy an MR or some of them, but not all of them, do.

The first scenario happened when R_{F1} and R_{F2} were used to check a cloud with a space-shared hypervisor, where none of the follow-up test cases satisfied these MRs. The catalogue of MRs was designed to represent the expected properties of a cloud in the sense of providing a general view of the system under test. In some cases, a particular configuration of the cloud may not be fully represented by a general relation. This happens, for example, with R_{F1} and R_{F2} that are designed to check the behaviour of a space-shared hypervisor, which is the most restrictive one. In this case, when at least one VM cannot be allocated in the physical resources of the cloud, the allocation fails and, consequently, the number of tenants that are successfully processed by the cloud leads to these MRs not being satisfied. However, it is important to note that these relations are useful for the other cloud configurations. Specifically, the remaining hypervisors, time-shared and time-shared oversubscription, share resources so that their functionality is less restrictive. For example, it may happen that even though the requested CPU resources were not available, the cloud is able to deploy VMs that share the available resources. Thus, in these cases we have MRs that are appropriate for some hypervisor policies but not for others.

The second scenario was observed in several cases. The first appears when R_{F3} was used to test a cloud with a space-shared hypervisor, where 95% of the follow-up test cases satisfied this MR. Although this percentage shows that the MR represents a general view of the cloud, there were some situations that were not considered by the restrictions reflected in R_{F3}. In particular, R_{F3} defines a constraint that compares the number of VMs that are successfully deployed in the cloud, that is, $sim_{\uparrow VMs}(m, \omega) \leq sim_{\uparrow VMs}(m', \omega')$. Due to the random nature of the follow-up test cases, the order in which the tenants appear in the generated workload ω' may affect the result for allocating the requested resources in the cloud. For instance, if the cloud is nearly reaching the saturation point, then it is usually easier to allocate two VM$_{small}$ instances, requiring 2 CPU cores in total, than one VM$_{large}$ requiring 4 CPU cores. In the latter case, the allocation fails, causing a reduced number of successfully deployed VMs and, consequently, not satisfying the MR. In order to alleviate this problem, we generated a large number of different and heterogeneous workloads so that our experiments faithfully reflect the real behaviour of the cloud.

The second case happened when R_{C2} was used to test the m^m_{best} cloud with a time-shared hypervisor. This situation can be categorised as a corner case, since it is mainly produced by the hypervisor, which increases the number of VMs that are sharing the same resources, leading to performance degradation due to VM migration. Moreover, since the best-fit policy aims to minimise the fragmentation of available physical CPUs by grouping, when possible, VMs in the same physical machine, it also leads to a scenario where the resources are saturated by being used for the same VMs. Consequently, these VMs require more time and cost to be deployed in the physical machines for executing the corresponding applications.

6.4 Checking fault detection effectiveness

We now describe an experiment used to analyse the effectiveness of the proposed methodology. In order to accomplish this analysis, we used mutation testing techniques to artificially seed faults into the clouds under test. The testing process was performed as follows: i) for each cloud configuration, 15 different mutants, representing faulty versions of the cloud, were generated; ii) for each mutant, approximately 1,100 different follow-up test cases were automatically created; iii) all the generated follow-up test cases were executed using the cloudSim-Plus simulator; iv) the catalogue of MRs was used to check whether the obtained results satisfy the constraints reflected in the MRs. In this experiment, up to 450,000 simulations were executed.

Table 5 shows a list of the generated mutants, which are categorised into four groups, namely General, Space-shared, Time-shared and Time-shared oversub. The second column of this table shows the mutant's ID and the last column presents a description of each mutant. For each cloud configuration, 15 different mutants were generated, that is, 12 general mutants and 3 specific mutants. General mutants focus on the global behaviour of the cloud, while specific mutants focus on the policy used

TABLE 4
Validity (in %) of each MR for testing cloud models using the cloudSim-Plus simulator

Hypervisor	Source test	Performance				Functional			Cost		
		R_{P1}	R_{P2}	R_{P3}	R_{P4}	R_{F1}	R_{F2}	R_{F3}	R_{C1}	R_{C2}	R_{C3}
Space-shared	(m^m_{best}, ω)	100	100	100	100	0	0	**95**	100	100	100
	(m^m_{first}, ω)	100	100	100	100	0	0	**95**	100	100	100
	(m^m_{worst}, ω)	100	100	100	100	0	0	**95**	100	100	100
Time-shared	(m^m_{best}, ω)	100	100	100	100	100	100	100	100	**55**	100
	(m^m_{first}, ω)	100	100	100	100	100	100	100	100	100	100
	(m^m_{worst}, ω)	100	100	100	100	100	100	100	100	100	100
Time-shared oversubscription	(m^m_{best}, ω)	100	100	100	100	100	100	100	100	100	100
	(m^m_{first}, ω)	100	100	100	100	100	100	100	100	100	100
	(m^m_{worst}, ω)	100	100	100	100	100	100	100	100	100	100

TABLE 5
Description of the generated mutants

Category	**Id**	**Description**
General	1	Error checking if a PM is suitable for a VM.
	2	Modification of the currently allocated MIPS from the physical PEs.
	3	Incorrect MIPS allocated for a VM reduced for the CPU migration overhead.
	4	Allocates the host with less PEs in use for a given VM.
	5	Error checking if resources required by the Vm already were provisioned.
	6	Error searching a valid physical machine to host a VM.
	7	Error reporting the resources allocation.
	8	Error reporting the creation of a VM.
	9	Wrong allocation of processing units.
	10	False error reported on a resource allocation.
	11	Wrong estimated time when a given cloudlet is supposed to finish executing.
	12	Modification of the minimum time between events when it is close to the finishing time.
Space-shared	13	Seeds a modification in the calculation of the PM using the scheduler. Specifically, if it has enough MIPS capacity to host a given VM.
	14	Modification on the list of processing elements of a PM.
	15	Modifies the calculation of the requested amount of MIPS, specifically if the PM has available MIPS to be allocated to a VM.
Time-shared	16	Allows to allocate a VM that requires exactly the same capacity of a physical PE.
	17	Seeds a modification that allows a minimum percentage of over-subscription.
	18	Error allocating the MIPS requested by a VM.
Time-shared oversub	19	Seeds a modification in the calculation of the PM using the scheduler.
	20	Error allocating the MIPS requested by a VM.
	21	Invalid VM allocation.

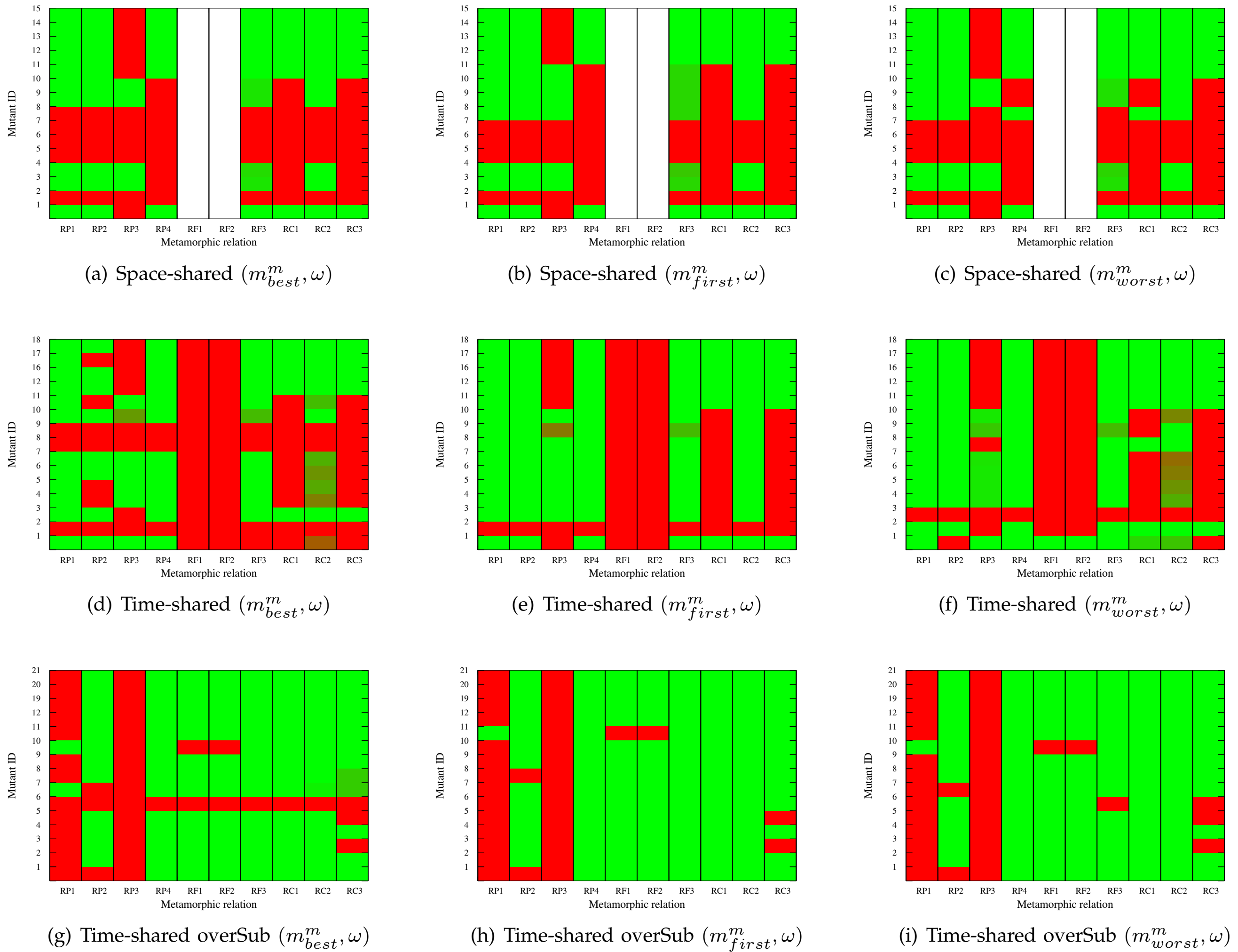
by the hypervisor. Thus, each group of specific mutants can only be applied to one hypervisor. These mutants were inspired by bugs located in real systems and papers found in the current literature. For example, mutant 1 represents a real bug located in the Git repository of the cloudSim-Plus simulator[1].

Firstly, we tested the clouds using D_m, that is, a data center containing 576 physical machines. Figure 2 shows the results of the testing process. Each chart shows a single cloud configuration, that is, the execution of the workload ω over a cloud using a specific cloud manager and one hypervisor. Thus, the charts placed in the same row represent clouds using the same hypervisor, while the charts placed in the same column represent clouds using the same cloud manager. The x-axis of each chart shows the MRs used in the testing process to detect faults and the y-axis refers to the mutant ID. In essence, these results represent the ability of each MR to detect the mutants, which represent a fault in the cloud. It is important to remark that approximately 15,000 different follow-up test cases were executed for each mutant. Since R_{F1} and R_{F2} cannot be used to validate the behaviour of the cloud when a space-shared hypervisor is used, we discarded them in figures 2.a, 2.b, and 2.c (the corresponding columns are blank).

If the results provided by the executions of all the follow-up test cases were not able to detect the mutant using an MR, then the mutant remained alive, which is represented in green. In contrast, when an MR was able to detect a mutant using the results provided by the execution of all the follow-up test cases, we say that this mutant was killed, which is shown in red. Additionally, we use a gradient to represent intermediate values, which ranges from red (100% of the follow-up test cases discover the mutant) to green (0% of the follow-up test cases discover the mutant). Let us note that, for the sake of clarity, Figure 2 provides a general view of the results. However, different tables providing a detailed version of these results are presented in the Appendix of this paper.

Although these results show that all the faults in the clouds were detected by at least one MR, it is important

1. https://github.com/manoelcampos/cloudsim-plus/commit/b69d1929cc12023b8134c734065f77de4f1cb2f2



(a) Space-shared (m^m_{best}, ω) (b) Space-shared (m^m_{first}, ω) (c) Space-shared (m^m_{worst}, ω)

(d) Time-shared (m^m_{best}, ω) (e) Time-shared (m^m_{first}, ω) (f) Time-shared (m^m_{worst}, ω)

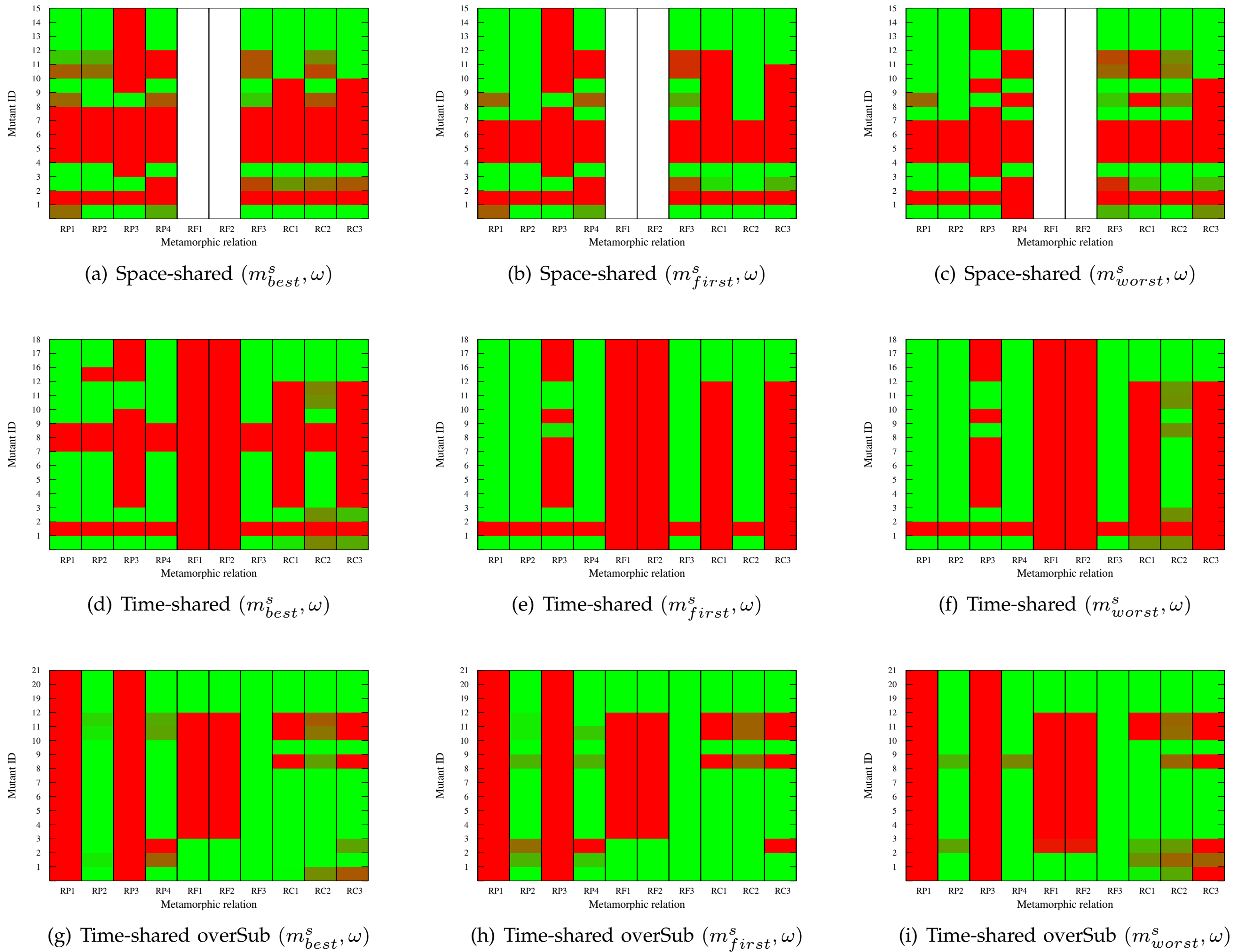
(g) Time-shared overSub (m^m_{best}, ω) (h) Time-shared overSub (m^m_{first}, ω) (i) Time-shared overSub (m^m_{worst}, ω)

Fig. 2. Testing process for locating faults in the clouds m^m_{best}, m^m_{first} and m^m_{worst} using the cloudSim-Plus simulator.

to mention that each cloud configuration provided different results concerning the detection of mutants. The best results were obtained when the cloud under test used a space-shared hypervisor, which is reflected in the highest number of detected mutants. We can see that in this situation more mutants are killed by more MRs. For example, mutants $4-7$ are detected by all the MRs used. We see that specific mutants, that is, mutants $13-15$, were more difficult to kill than generic mutants. In our case, we have that the previously mentioned specific mutants were killed by only one MR. Clouds using a time-shared hypervisor provided the lowest number of detected mutants. This was mainly caused by the restrictions established by the hypervisor, that is, while the space-shared policy does not allow users to access the cloud when the requested physical resources are not available, hypervisors based on time-shared policies allow access to the cloud in such situations.

It is worth mentioning that clouds using time-shared and space-shared hypervisors have a similar shape in the generated charts. For instance, specific mutants (see Table 5) were not detected by the MRs focusing on cost. In contrast, R_{F1} and R_{F2} were able to detect the majority of the mutants, with the exception of mutant 1 when the cloud uses a time-shared hypervisor and the worst-fit policy for allocating resources (see figures 2.d, 2.e and 2.f). Similar to the case when we considered the validity of the MRs, we observe that faulty versions of *flexible* hypervisors were more difficult to detect than restrictive ones, over-subscription representing the extremest situation. Similar to before, specific mutants were more difficult to kill. This is due to the fact that very specific faults require MRs targeting the concrete wrong behaviour and, therefore, most (generic) MRs were unable to detect the fault included in the mutant.

It is interesting to note that when a cloud used the time-shared oversubscription policy, the obtained results were completely different from the ones obtained from testing the previous cloud configurations. In this particular case, R_{P1} and R_{P3} were the most effective MRs. However, the rest of the relations provided low effectiveness. Although all the mutants were detected,



(a) Space-shared (m^s_{best}, ω) (b) Space-shared (m^s_{first}, ω) (c) Space-shared (m^s_{worst}, ω)

(d) Time-shared (m^s_{best}, ω) (e) Time-shared (m^s_{first}, ω) (f) Time-shared (m^s_{worst}, ω)

(g) Time-shared overSub (m^s_{best}, ω) (h) Time-shared overSub (m^s_{first}, ω) (i) Time-shared overSub (m^s_{worst}, ω)

Fig. 3. Testing process for locating faults in the clouds m^s_{best}, m^s_{first} and m^s_{worst} using the cloudSim-Plus simulator.

in some situations several MRs could not detect a single mutant (see R_{P4}, R_{C1} and R_{C2} in figures 2.h and 2.i).

The cloud manager used in the clouds has a small impact on the obtained results. Cloud managers based on the best-fit policy provided the best results, while the worst results were provided with the Worst-Fit policy. Observe that the results for the three different cloud managers were similar.

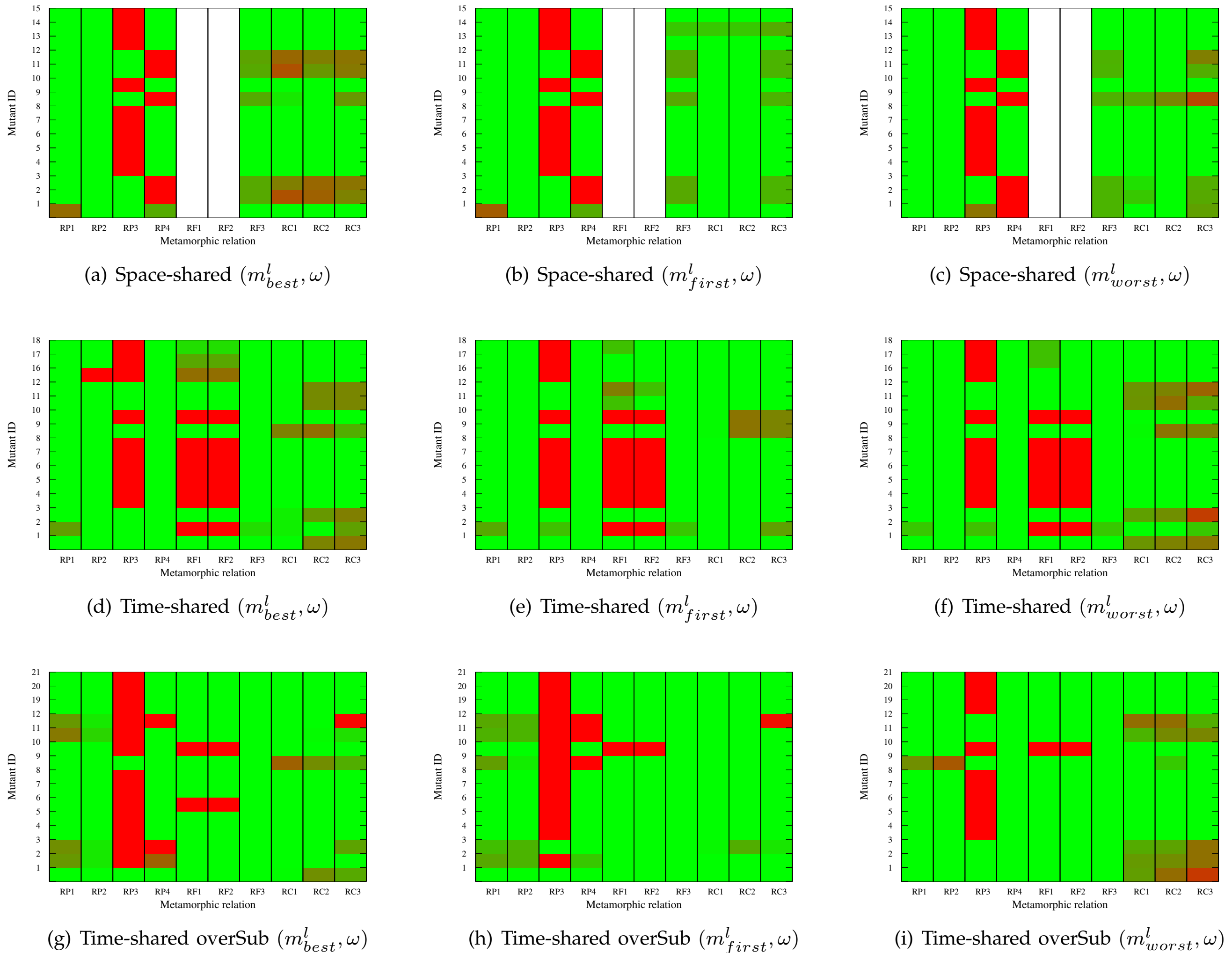
It is worth emphasising that we observe again the same pattern in the results. In this case, we obtained the worst results for the most *restrictive* hypervisor. This reinforces our previous observation that generic MRs were not well suited to detect faults when we use this type of hypervisors.

Next, in order to analyse the scalability of our approach, we tested several cloud configurations containing a different number of physical machines. The results of the testing process using a *small* cloud – containing 320 physical machines – are depicted in Figure 3, while the results obtained for testing a *large* cloud – containing 1088 physical machines – are shown in Figure 4.

Broadly speaking, the shape of these charts is similar to the ones provided from analysing the *medium* cloud, which contains 576 physical machines. Similar to the previous experiments, in these cases, R_{F1} and R_{F2} were not applied when the space-shared hypervisor was used (see figures 3.b, 3.c, 4.b, and 4.c).

When the *small* cloud was tested, a significant number of red areas can be seen in the charts, which means that the provided MRs were able to detect a large number of faults in the cloud. Since the workloads used in the experiments were the same, in this particular case the number of idle resources is very limited and, consequently, a wrong VM allocation algorithm is more easily detected. In addition, some faulty versions of the *time-shared over-subscription* algorithm were detected in this case, with R_{F1} and R_{F2} being the most efficient MRs in this configuration.

The results provided when testing the *large* cloud (see Figure 4) are rather different, that is, fewer errors were detected. This is mainly caused by the large amount of idle resources. Recall that the workload processed by all



(a) Space-shared (m^l_{best}, ω) (b) Space-shared (m^l_{first}, ω) (c) Space-shared (m^l_{worst}, ω)

(d) Time-shared (m^l_{best}, ω) (e) Time-shared (m^l_{first}, ω) (f) Time-shared (m^l_{worst}, ω)

(g) Time-shared overSub (m^l_{best}, ω) (h) Time-shared overSub (m^l_{first}, ω) (i) Time-shared overSub (m^l_{worst}, ω)

Fig. 4. Testing process for locating faults in the clouds m^l_{best}, m^l_{first} and m^l_{worst} using the cloudSim-Plus simulator.

the clouds is the same. Hence, a faulty algorithm for allocating VMs in the data center is more likely to find physical resources in this case and, consequently, the error is more difficult to detect. This behaviour is clearly shown when R_{F1} and R_{F2} are applied to test the cloud, using a *time-shared* hypervisor with a *best-fit* and a *first-fit* cloud manager. When the *small* and *medium* clouds were tested, these relations were able to detect all the mutants (see figures 2.d, 2.e, 3.d and 3.e). When we applied these MRs to test the same scenarios in the *large* cloud (see figures 4.d, 4.e) there was a noticeable number of undetected mutants. These differences notwithstanding, it is important to remark that all the mutants were detected in each analysed cloud configuration.

As a concluding remark, a careful analysis of the results allows us to conclude that relaxing the constraints in the hypervisor policy makes it more difficult to detect faulty clouds. Also, reducing the fragmentation of CPU resources among the physical machines, which occurs when the cloud manager implements the best-fit policy, slightly improves the obtained results.

6.5 Discussion of the results and answers to the research questions

We now discuss the results and what they tell us about the research questions presented in Section 6.1.

*6.5.1 **Research Question 1**: Are the MRs valid?*

In order to answer this question, we use the results described in Section 6.3. In this case, we executed up to 90,000 simulations to determine whether the MRs are valid when used with different cloud configurations. We measured the validity of the different MRs by using the percentage of test cases, for a correct cloud simulation, that satisfy each MR [61], [62], [63].

The main objective of this experiment was to determine whether these MRs are valid; whether correct cloud systems pass the MRs. It was found that most of the MRs were satisfied by all tests. However, there were some

exceptions, such as R_{F1} and R_{F2}, that were not valid with certain cloud scenarios.

Hence, the answer to **Research Question 1** is that *MT is largely suitable for testing cloud systems but, in certain situations, some MRs should not be used.*

6.5.2 ***Research Question 2**: How effective is MT at detecting faults in cloud systems?*

In order to answer this question, we use the results given in Section 6.4, where mutation was applied to generate a large number of faulty versions of the clouds under study. Let us mention that applying mutation testing techniques to check the effectiveness of MRs is a widely adopted approach in the MT community [8], [9], [64].

The results were promising, with the catalogue of MRs being able to detect all faulty clouds. We found that the effectiveness of the catalogue of MRs (to detect faults) relates to the level of restrictions imposed by the hypervisor, that is, it is easier to find faults with highly restrictive hypervisors than with *relaxed* hypervisors. In contrast, the policy implemented by the cloud manager had very little impact on the results. In these cases, we obtained the best results when the CPU fragmentation was minimised, which was achieved by the best-fit policy.

We also noticed that some MRs were more effective in certain scenarios. For example, when the cloud to be tested used a time-shared hypervisor, R_{F1} and R_{F2} were able to detect most of the faulty clouds, with the exception of mutant 1 (see Figure 2.f). R_{P1} and R_{P3} provided promising results for detecting faults in those clouds using hypervisors based on time-shared over subscription policies.

To answer **Research Question 2** we can conclude that *MT is effective in detecting faulty clouds system. The effectiveness of MRs depends on the configuration used, with it being more difficult to detect faults in clouds using hypervisors based on non-restrictive policies*.

7 THREATS TO VALIDITY

This section presents the threats to the validity of the results of our empirical study.

7.1 Internal threats

Internal validity focuses on determining if our findings, which are based on the data produced by the experiments, truly represent a cause-and-effect relationship. Hence, the internal validity of this study relates to the implementation of the experiments.

The MRs presented in Section 5 were designed by two experts. The ability of MT to detect faults in the system is directly correlated with the selection of MRs and, consequently, the results may be different if other MRs were used. However, using domain-specific properties should make the approach more effective. In order to mitigate this issue, in Section 6.3 we report on analysis that checks the adequacy of each MR involved in the testing process. The MRs and the algorithm used to generate follow-up test cases were implemented in Java. In order to increase the confidence in the correctness of the implementations, three researchers inspected the source code and different tests were manually executed. We evaluated the MRs using the source test cases that were manually generated by the user. The follow-up test cases were automatically generated using the constraints reflected in the MRs. Additionally, we used a well-know simulator, CloudSim-Plus, that has been widely adopted by the research community to analyse a wide spectrum of cloud configurations.

7.2 External threats

External validity concerns the extent to which the results of a study can be generalised.

We used 27 cloud configurations in our empirical study, with these being generated by combining nine cloud models with three different hypervisors. Moreover, up to 450,000 different follow-up test cases were automatically generated using, as basis, the source test cases and the MRs. We believe that the cloud models are representative. However, there is no guarantee that the obtained will be the same for other scenarios.

7.3 Constructs threats

Construct validity concerns whether the used measures are representative or not.

We checked the effectiveness of MRs using the number of test cases that satisfy each MR, which is a widely used measure in the research community. Unknown defects in the CloudSim-Plus simulator, in our proposed MRs or in our algorithm for generating follow-up test cases could be a threat to construct validity. However, we controlled this threat by testing the implementation using a wide range of test cases.

8 CONCLUSIONS AND FUTURE WORK

This paper presented a framework, called TEA-*Cloud*, for testing cloud computing systems. TEA-*Cloud* integrates simulation and MT techniques, with this making it possible to automate the generation and execution of large test suites over the clouds under test. The data provided by the execution of the test cases are automatically checked by a catalogue of MRs that represents expected properties of the cloud.

In order to check the validity and effectiveness of TEA-*Cloud* we performed an experimental study, where 27 different cloud configurations were analysed using mutation testing techniques to artificially inject 15 different faults into each. Interestingly, the percentage of test cases that satisfy MRs significantly dropped when the cloud under test used a hypervisor based on non-restrictive policies, like time-shared oversubscription. In contrast, better results were obtained when a restrictive

hypervisor was used. All the generated mutants were detected by at least one MR, hence obtaining promising results.

The main limitation of TEA-*Cloud* is that the methodology requires appropriate MRs. However, there is evidence that if a domain expert provides appropriate MRs then the testing process provides very useful information. We have mitigated this issue (the need for appropriate MRs) by providing a set of MRs and analysing the adequacy of these MRs for testing cloud configurations. In our study, we discarded two MRs when considering cloud configurations that use a hypervisor based on a space-shared policy.

There are several possible lines of future work. First, we plan to extend the set of MRs. This will allow us to increase the functionality of our methodology. As part of this, we expect to analyse the suitability of combining different MRs for testing cloud configurations. Also, we plan to build a repository of MRs, where the results of analysing different cloud systems, using a wide range of cloud simulators, can be stored.

ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers for the careful reading of the paper and the many constructive comments, which have helped us to further strengthen the paper.

REFERENCES

[1] G. J. Myers, C. Sandler, and T. Badgett, *The Art of Software Testing*, 3rd ed. John Wiley & Sons, 2011.
[2] P. Ammann and J. Offutt, *Introduction to Software Testing*, 2nd ed. Cambridge University Press, 2017.
[3] R. M. Hierons, K. Bogdanov, J. Bowen, R. Cleaveland, J. Derrick, J. Dick, M. Gheorghe, M. Harman, K. Kapoor, P. Krause, G. Luettgen, A. Simons, S. Vilkomir, M. Woodward, and H. Zedan, "Using formal specifications to support testing," *ACM Computing Surveys*, vol. 41, no. 2, pp. 9:1–9:76, 2009.
[4] A. R. Cavalli, T. Higashino, and M. Núñez, "A survey on formal active and passive testing with applications to the cloud," *Annales of Telecommunications*, vol. 70, no. 3-4, pp. 85–93, 2015.
[5] E. J. Weyuker, "On testing non-testable programs," *The Computer Journal*, vol. 25, no. 4, pp. 465–470, 1982.
[6] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE Transactions on Software Engineering*, vol. 41, no. 5, pp. 507–525, 2015.
[7] T. Y. Chen, S. C. Cheung, and S. M. Yiu, "Metamorphic testing: a new approach for generating next test cases," Department of Computer Science, Hong Kong University of Science and Technology, Tech. Rep. HKUST-CS98-01, 1998.
[8] S. Segura, G. Fraser, A. B. Sánchez, and A. Ruiz-Cortés, "A survey on metamorphic testing," *IEEE Transactions on Software Engineering*, vol. 42, no. 9, pp. 805–824, 2016.
[9] T. Y. Chen, F.-C. Kuo, H. Liu, P.-L. Poon, D. Towey, T. H. Tse, and Z. Q. Zhou, "Metamorphic testing: A review of challenges and opportunities," *ACM Computing Surveys*, vol. 51, no. 1, pp. 4:1–4:27, 2018.
[10] M. Olsen and M. Raunak, "Increasing validity of simulation models through metamorphic testing," *IEEE Transactions on Reliability*, vol. 68, no. 1, pp. 91–108, 2019.
[11] R. M. Hierons, M. G. Merayo, and M. Núñez, "Mutation testing," in *Encyclopedia of Software Engineering*, P. A. Laplante, Ed. Taylor & Francis, 2010, pp. 594–602.
[12] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE Transactions on Software Engineering*, vol. 37, no. 5, pp. 649–678, 2011.
[13] M. Papadakis, M. Kintis, J. Zhang, Y. Jia, Y. L. Traon, and M. Harman, "Mutation testing advances: An analysis and survey," ser. Advances in Computers, A. M. Memon, Ed. Elsevier, 2019, vol. 112, pp. 275 – 378.
[14] A. Núñez and R. M. Hierons, "A methodology for validating cloud models using metamorphic testing," *Annales of Telecommunications*, vol. 70, no. 3-4, pp. 127–135, 2015.
[15] L. Garber, "News briefs," *IEEE Computer*, vol. 44, no. 6, pp. 18–20, 2011.
[16] L. Ciortea, C. Zamfir, S. Bucur, V. Chipounov, and G. Candea, "Cloud9: a software testing service," *ACM SIGOPS Operating Systems Review*, vol. 43, no. 4, pp. 5–10, 2010.
[17] P. Joshi, H. Gunawi, and K. Sen, "PREFAIL: a programmable tool for multiple-failure injection," *ACM SIGPLAN Notices*, vol. 46, no. 10, pp. 171–188, 2011.
[18] M. Oriol and F. Ullah, "YETI on the Cloud," in *3rd Int. Conf. on Software Testing, Verification, and Validation Workshops*. IEEE Computer Society, 2010, pp. 434–437.
[19] J. Morán, A. Bertolino, C. de la Riva, and J. Tuya, "Automatic testing of design faults in mapreduce applications," *IEEE Transactions on Reliability*, vol. 67, no. 3, pp. 717–732, 2018.
[20] R. M. Hierons, M. G. Merayo, and M. Núñez, "Implementation relations and test generation for systems with distributed interfaces," *Distributed Computing*, vol. 25, no. 1, pp. 35–62, 2012.
[21] ——, "Bounded reordering in the distributed test architecture," *IEEE Transactions on Reliability*, vol. 67, no. 2, pp. 522–537, 2018.
[22] ——, "An extended framework for passive asynchronous testing," *Journal of Logical and Algebraic Methods in Programming*, vol. 86, no. 1, pp. 408–424, 2017.
[23] M. G. Merayo, R. M. Hierons, and M. Núñez, "Passive testing with asynchronous communications and timestamps," *Distributed Computing*, vol. 31, no. 5, pp. 327–342, 2018.
[24] ——, "A tool supported methodology to passively test asynchronous systems with multiple users," *Information & Software Technology*, vol. 104, pp. 162–178, 2018.
[25] H. Lu, W. K. Chan, and T. H. Tse, "Testing pervasive software in the presence of context inconsistency resolution services," in *30th Int. Conf. on Software Engineering, ICSE'08*. ACM Press, 2008, pp. 61–70.
[26] B. Marin, T. Vos, G. Giachetti, A. Baars, and P. Tonella, "Towards testing future web applications," in *5th Int. Conf. on Research Challenges in Information Science, RCIS'11*. IEEE Computer Society, 2011, pp. 1–12.
[27] W. Chan, L. Mei, and Z. Zhang, "Modeling and testing of cloud applications," in *4th IEEE Asia-Pacific Services Computing Conference, APSCC'09*. IEEE Computer Society, 2009, pp. 111–118.
[28] L. Luo, S. Meng, X. Qiu, and Y. Dai, "Improving failure tolerance in large-scale cloud computing systems," *IEEE Transactions on Reliability (in press)*, pp. 1–13, 2019.
[29] J. Liu, S. Wang, A. Zhou, S. A. P. Kumar, F. Yang, and R. Buyya, "Using proactive fault-tolerance approach to enhance cloud service reliability," *IEEE Transactions on Cloud Computing*, vol. 6, no. 4, pp. 1191–1202, 2018.
[30] T. Banzai, H. Koizumi, R. Kanbayashi, T. Imada, T. Hanawa, and M. Sato, "D-cloud: Design of a software testing environment for reliable distributed systems using cloud computing technology," in *10th IEEE/ACM Int. Conf. on Cluster, Cloud and Grid Computing, CCGrid'10*, 2010, pp. 631–636.
[31] F. Bellard, "QEMU, a fast and portable dynamic translator," in *Annual conference on USENIX Annual Technical Conference, ATEC '05*. USENIX Association, 2005, pp. 41–41.
[32] A. Bertolino, "Software testing research: Achievements, challenges, dreams," in *Workshop on the Future of Software Engineering: FOSE'07*. IEEE Computer Society, 2007, pp. 85–103.
[33] B. Wickremasinghe, R. N. Calheiros, and R. Buyya, "CloudAnalyst: A CloudSim-Based Visual Modeller for Analysing Cloud Computing Environments and Applications," in *24th IEEE Int. Conf. on Advanced Information Networking and Applications, AINA'10*. IEEE Computer Society, 2010, pp. 446–452.
[34] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. D. Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software - Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.
[35] S. F. Piraghaj, A. V. Dastjerdi, R. N. Calheiros, and R. Buyya, "ContainerCloudSim: An environment for modeling and simulation of

containers in cloud data centers," *Software: Practice and Experience*, vol. 47, no. 4, pp. 505–521, 2017.

[36] M. C. S. Filho, R. L. Oliveira, C. C. Monteiro, P. R. M. Inácio, and M. M. Freire, "CloudSim Plus: A cloud computing simulation framework pursuing software engineering principles for improved modularity, extensibility and correctness," in *15th IFIP/IEEE Symposium on Integrated Network and Service Management, IM'17*. IEEE Computer Society, 2017, pp. 400–406.

[37] M. Tighe, G. Keller, M. Bauer, and H. Lutfiyya, "DCSim: A data centre simulation tool for evaluating dynamic virtualized resource management," in *8th Int. Conf. on network and service management, CNSM'12*. IEEE Computer Society, 2012, pp. 385–392.

[38] R. N. Calheiros, M. A. S. Netto, C. A. F. D. Rose, and R. Buyya, "EMUSIM: an integrated emulation and simulation environment for modeling, evaluation, and validation of performance of Cloud computing applications," *Software: Practice and Experience*, vol. 43, no. 5, pp. 595–612, 2013.

[39] D. Kliazovich and P. B. amd S. U. Khan, "GreenCloud: A packet-level simulator of energy-aware cloud computing data centers," *The Journal of Supercomputing*, vol. 62, no. 3, pp. 1263–1283, 2012.

[40] S. Ostermann, K. Plankensteiner, R. Prodan, and T. Fahringer, "GroudSim: An Event-Based Simulation Framework for Computational Grids and Clouds," in *Euro-Par 2010 Parallel Processing Workshops, LNCS 6586*. Springer, 2011, pp. 305–313.

[41] A. Núñez, J. L. Vázquez-Poletti, A. C. Caminero, G. G. Castañé, J. Carretero, and I. M. Llorente, "iCanCloud: A flexible and scalable cloud infrastructure simulator," *Journal of Grid Computing*, vol. 10, no. 1, pp. 185–209, 2012.

[42] G. Castañé, A. Núñez, P. Llopis, and J. Carretero, "E-mc^2: A formal framework for energy modelling in cloud computing," *Simulation Modelling Practice and Theory*, vol. 39, pp. 56–75, 2013.

[43] H. Casanova, A. Legrand, and M. Quinson, "SimGrid: A generic framework for large-scale distributed experiments," in *10th Int. Conf. on Computer Modeling and Simulation, UKSIM'08*, 2008, pp. 126–131.

[44] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Versatile, scalable, and accurate simulation of distributed applications and platforms," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2899–2917, 2014.

[45] W. Grieskamp, N. Kicillof, K. Stobie, and V. Braberman, "Model-based quality assurance of protocol documentation: tools and methodology," *Software Testing, Verification and Reliability*, vol. 21, no. 1, pp. 55–71, 2011.

[46] H. Liu, X. Liu, and T. Y. Chen, "A new method for constructing metamorphic relations," in *12th Int. Conf. on Quality Software, QSIC'12*. IEEE Computer Society, 2012, pp. 59–68.

[47] S. Segura, J. A. Parejo, J. Troya, and A. Ruiz-Cortés, "Metamorphic testing of RESTful web APIs," *IEEE Transactions on Software Engineering*, vol. 44, no. 11, pp. 1083–1099, 2018.

[48] X. Xie, J. W. K. Ho, C. Murphy, G. E. Kaiser, B. Xu, and T. Y. Chen, "Testing and validating machine learning classifiers by metamorphic testing," *Journal of Systems and Software*, vol. 84, no. 4, pp. 544–558, 2011.

[49] V. Le, M. Afshari, and Z. Su, "Compiler validation via equivalence modulo inputs," in *35th ACM SIGPLAN Conf. on Programming Language Design and Implementation, PLDI'14*. ACM, 2014, pp. 216–226.

[50] P. Rao, Z. Zheng, T. Y. Chen, N. Wang, and K. Cai, "Impacts of test suite's class imbalance on spectrum-based fault localization techniques," in *13th Int. Conf. on Quality Software, QSIC'13*. IEEE Computer Society, 2013, pp. 260–267.

[51] X. Xie, W. E. Wong, T. Y. Chen, and B. Xu, "Metamorphic slice: An application in spectrum-based fault localization," *Information and Software Technology*, vol. 55, no. 5, pp. 866–879, 2013.

[52] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand, "Experiments of the Effectiveness of Dataflow and Controlflow-based Test Adequacy Criteria," in *16th Int. Conference on Software Engineering, ICSE'94*. ACM Press, 1994, pp. 191–200.

[53] A. Weiss, "Computing in the clouds," *netWorker*, vol. 11, no. 4, pp. 16–25, 2007.

[54] Y. Mansouri, A. N. Toosi, and R. Buyya, "Data storage management in cloud environments: Taxonomy, survey, and future directions," *ACM Computing Surveys*, vol. 50, no. 6, pp. 91:1–91:51, 2017.

[55] A. Bernal, M. E. Cambronero, V. Valero, A. Núñez, and P. C. Cañizares, "A framework for modeling cloud infrastructures and user interactions," *IEEE Access*, vol. 7, pp. 43 269–43 285, 2019.

[56] Amazon Elastic Compute Cloud, http://aws.amazon.com/ec2/. Date of last access: 10th June, 2019.

[57] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz, "Runtime measurements in the cloud: observing, analyzing, and reducing variance," *Proceedings of the VLDB Endowment*, vol. 3, pp. 460–471, 2010.

[58] A. Ahmed and A. S. Sabyasachi, "Cloud computing simulators: A detailed survey and future direction," in *4th IEEE International Advance Computing Conference, IACC'14*. IEEE Computer Society, 2014, pp. 866–872.

[59] J. Byrne, S. Svorobej, K. Giannoutakis, D. Tzovaras, P. J. Byrne, P. O. Östberg, A. Gourinovitch, and T. Lynn, "A review of cloud computing simulation platforms and related environments," in *7th Int. Conf. on Cloud Computing and Services Science, CLOSER'17*, 2017, pp. 651–663.

[60] F. Fakhfakh, H. H. Kacem, and A. H. Kacem, "Simulation tools for cloud computing: A survey and comparative study," in *16th IEEE/ACIS Int. Conf. on Computer and Information Science, ICIS'17*. ACM Press, 2017, pp. 221–226.

[61] F. Kuo, S. Liu, and T. Y. Chen, "Testing a Binary Space Partitioning Algorithm with Metamorphic Testing," in *26th ACM Symposium on Applied Computing, SAC'11*. ACM Press, 2011, pp. 1482–1489.

[62] M. Jiang, T. Y. Chen, F. Kuo, and Z. Ding, "Testing central processing unit scheduling algorithms using metamorphic testing," in *4th IEEE Int. Conf. on Software Engineering and Service Science, ICSESS'13*, 2013, pp. 530–536.

[63] P. C. Cañizares, A. Núñez, and J. d. Lara, "An expert system for checking the correctness of memory systems using simulation and metamorphic testing," *Expert Systems with Applications*, vol. 132, pp. 44–62, 2019.

[64] M. Asrafi, H. Liu, and F. Kuo, "On Testing Effectiveness of Metamorphic Relations: A Case Study," in *5th Int. Conf. on Secure Software Integration and Reliability Improvement, SSIRI'11*. IEEE Computer Society, 2011, pp. 147–156.

APPENDIX

The following tables show – in detail – the results of the testing process. For each mutant, approximately 1100 different follow-up test cases have been generated and executed. The generated data is checked using the MRs. These results are measured in % of follow-up test cases that are not able to detect a mutant using an MR. Thus, a value of 100% means that none the follow-up test cases detect the mutant (the mutant is alive), while 0% means the opposite, that is, all the follow-up test cases detect the mutant. The last row shows the average of follow-up test cases that cannot detect the mutants using an MR.

TABLE 6
Results of the testing process for locating faults in m_{best} using the time-shared hypervisor (lower is better)

Id	$\mathbf{MR_1}$	$\mathbf{MR_2}$	$\mathbf{MR_3}$	$\mathbf{MR_4}$	$\mathbf{MR_5}$	$\mathbf{MR_6}$	$\mathbf{MR_7}$	$\mathbf{MR_8}$	$\mathbf{MR_9}$	$\mathbf{MR_{10}}$
0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	55.81	100.0
1	100.0	100.0	100.0	100.0	0.0	0.0	0.0	0.0	41.62	0.0
2	0.00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	100.00	100.0	0.0	100.0	0.0	0.0	100.0	100.0	100.0	100.0
4	100.00	0.0	100.0	100.0	0.0	0.0	100.0	0.0	54.34	0.0
5	100.00	0.0	100.0	100.0	0.0	0.0	100.0	0.0	69.043	0.0
6	100.00	100.0	100.0	100.0	0.0	0.0	100.0	0.0	61.31	0.0
7	100.00	100.0	100.0	100.0	0.0	0.0	100.0	0.0	71.32	0.0
8	0.00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	100.00	100.0	64.0	100.0	0.0	0.0	75.0	0.0	100.0	0.0
11	100.00	0.0	100.0	100.0	0.0	0.0	100.0	0.0	76.0	0.0
12	100.00	100.0	0.0	100.0	0.0	0.0	100.0	100.0	100.0	100.0
16	100.00	100.0	0.0	100.0	0.0	0.0	100.0	100.0	100.0	100.0
17	100.00	0.0	0.0	100.0	0.0	0.0	100.0	100.0	100.0	100.0
18	100.00	100.0	0.0	100.0	0.0	0.0	100.0	100.0	100.0	100.0
Avg.	80.0	53.33	44.26	80.00	0.00	0.00	71.66	33.33	64.90	33.33

TABLE 7
Results of the testing process for locating faults in m_{first} using the time-shared hypervisor (lower is better)

Id	$\mathbf{MR_1}$	$\mathbf{MR_2}$	$\mathbf{MR_3}$	$\mathbf{MR_4}$	$\mathbf{MR_5}$	$\mathbf{MR_6}$	$\mathbf{MR_7}$	$\mathbf{MR_8}$	$\mathbf{MR_9}$	$\mathbf{MR_{10}}$
0	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
1	100.00	100.00	100.00	100.00	28.00	28.00	99.48	100.00	100.00	10.29
2	100.00	100.00	0.00	100.00	0.00	0.00	100.00	100.00	100.00	100.00
3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	100.00	100.00	100.00	100.00	0.00	0.00	100.00	0.00	100.00	0.00
5	100.00	100.00	100.00	100.00	0.00	0.00	100.00	0.00	100.00	0.00
6	100.00	100.00	100.00	100.00	0.00	0.00	100.00	0.00	100.00	0.00
7	100.00	100.00	100.00	100.00	0.00	0.00	100.00	0.00	100.00	0.00
8	100.00	100.00	100.00	100.00	0.00	0.00	100.00	0.00	100.00	0.00
9	100.00	100.00	100.00	100.00	0.00	0.00	100.00	0.00	100.00	0.00
10	100.00	100.00	52.00	100.00	0.00	0.00	75.00	0.00	100.00	0.00
11	100.00	100.00	100.00	100.00	0.00	0.00	100.00	0.00	100.00	0.00
12	100.00	100.00	0.00	100.00	0.00	0.00	100.00	100.00	100.00	100.00
16	100.00	100.00	0.00	100.00	0.00	0.00	100.00	100.00	100.00	100.00
17	100.00	100.00	0.00	100.00	0.00	0.00	100.00	100.00	100.00	100.00
18	100.00	100.00	0.00	100.00	0.00	0.00	100.00	100.00	100.00	100.00
Avg.	93.33	93.33	56.80	93.33	1.86	1.86	91.63	40.00	93.33	34.02

TABLE 8
Results of the testing process for locating faults in m_{worst} using the time-shared hypervisor (lower is better)

Id	$\mathbf{MR_1}$	$\mathbf{MR_2}$	$\mathbf{MR_3}$	$\mathbf{MR_4}$	$\mathbf{MR_5}$	$\mathbf{MR_6}$	$\mathbf{MR_7}$	$\mathbf{MR_8}$	$\mathbf{MR_9}$	$\mathbf{MR_{10}}$
0	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
1	100.00	0.00	100.00	100.00	100.00	100.00	100.00	85.71	78.40	1.43
2	100.00	100.00	0.00	100.00	0.00	0.00	100.00	100.00	100.00	100.00
3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	100.00	100.00	91.84	100.00	0.00	0.00	100.00	0.00	70.79	0.00
5	100.00	100.00	91.84	100.00	0.00	0.00	100.00	0.00	61.03	0.00
6	100.00	100.00	91.84	100.00	0.00	0.00	100.00	0.00	52.90	0.00
7	100.00	100.00	90.45	100.00	0.00	0.00	100.00	0.00	46.93	0.00
8	100.00	100.00	0.00	100.00	0.00	0.00	100.00	100.00	100.00	0.00
9	100.00	100.00	80.00	100.00	0.00	0.00	75.00	0.00	100.00	0.00
10	100.00	100.00	91.84	100.00	0.00	0.00	100.00	0.00	56.96	0.00
11	100.00	100.00	0.00	100.00	0.00	0.00	100.00	100.00	100.00	100.00
12	100.00	100.00	0.00	100.00	0.00	0.00	100.00	100.00	100.00	100.00
16	100.00	100.00	0.00	100.00	0.00	0.00	100.00	100.00	100.00	100.00
17	100.00	100.00	0.00	100.00	0.00	0.00	100.00	100.00	100.00	100.00
18	100.00	100.00	0.00	100.00	0.00	0.00	100.00	100.00	100.00	100.00
Avg.	93.33	86.66	42.52	93.33	6.66	6.66	91.66	52.38	77.80	40.09

TABLE 9
Results of the testing process for locating faults in m_{best} using the space-shared hypervisor (lower is better)

Id	MR_1	MR_2	MR_3	MR_4	MR_5	MR_6	MR_7	MR_8	MR_9	MR_{10}
0	100.00	100.00	100.00	0.00	0.00	0.00	95.24	100.00	100.00	100.00
1	100.00	0.00	100.00	0.00	0.00	0.00	49.40	0.00	32.65	71.43
2	100.00	100.00	0.00	100.00	0.00	0.00	100.00	100.00	100.00	100.00
3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	100.00	100.00	100.00	0.00	0.00	0.00	90.48	0.00	100.00	0.00
5	100.00	100.00	100.00	0.00	0.00	0.00	87.30	0.00	100.00	0.00
6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
10	100.00	100.00	100.00	0.00	0.00	0.00	90.48	0.00	100.00	0.00
11	100.00	100.00	100.00	0.00	0.00	0.00	90.48	0.00	100.00	0.00
12	100.00	100.00	0.00	100.00	0.00	0.00	100.00	100.00	100.00	100.00
13	100.00	100.00	0.00	100.00	0.00	0.00	100.00	100.00	100.00	100.00
14	100.00	100.00	0.00	100.00	0.00	0.00	100.00	100.00	100.00	100.00
15	100.00	100.00	0.00	100.00	0.00	0.00	100.00	100.00	100.00	100.00
Avg.	66.67	60.00	33.33	33.33	0.00	0.00	60.54	33.33	62.17	38.09

TABLE 10
Results of the testing process for locating faults in m_{first} using the space-shared hypervisor (lower is better)

Id	MR_1	MR_2	MR_3	MR_4	MR_5	MR_6	MR_7	MR_8	MR_9	MR_{10}
0	100.00	100.00	100.00	100.00	0.00	0.00	95.54	100.00	100.00	100.00
1	100.00	0.00	100.00	0.00	0.00	0.00	38.64	0.00	100.00	71.43
2	100.00	100.00	0.00	100.00	0.00	0.00	100.00	100.00	100.00	100.00
3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	100.00	100.00	100.00	0.00	0.00	0.00	85.23	0.00	100.00	0.00
5	100.00	100.00	100.00	0.00	0.00	0.00	80.30	0.00	100.00	0.00
6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
9	100.00	100.00	100.00	0.00	0.00	0.00	85.23	0.00	100.00	0.00
10	100.00	100.00	100.00	0.00	0.00	0.00	85.23	0.00	100.00	0.00
11	100.00	100.00	100.00	0.00	0.00	0.00	85.23	0.00	100.00	0.00
12	100.00	100.00	100.00	0.00	0.00	0.00	85.23	0.00	100.00	0.00
13	100.00	100.00	0.00	100.00	0.00	0.00	100.00	100.00	100.00	100.00
14	100.00	100.00	0.00	100.00	0.00	0.00	100.00	100.00	100.00	100.00
15	100.00	100.00	0.00	100.00	0.00	0.00	100.00	100.00	100.00	100.00
Avg.	73.33	66.66	46.66	26.66	0.00	0.00	63.03	26.66	73.33	31.42

TABLE 11
Results of the testing process for locating faults in m_{worst} using the space-shared hypervisor (lower is better)

Id	MR_1	MR_2	MR_3	MR_4	MR_5	MR_6	MR_7	MR_8	MR_9	MR_{10}
0	100.00	100.00	100.00	100.00	100.00	0.00	94.32	100.00	100.00	100.00
1	100.00	0.00	100.00	0.00	0.00	0.00	39.77	0.00	52.96	71.43
2	100.00	100.00	0.00	100.00	0.00	0.00	100.00	100.00	100.00	100.00
3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	100.00	100.00	100.00	0.00	0.00	0.00	88.64	0.00	100.00	0.00
5	100.00	100.00	100.00	0.00	0.00	0.00	84.85	0.00	100.00	0.00
6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
9	100.00	100.00	0.00	100.00	0.00	0.00	0.00	100.00	100.00	0.00
10	100.00	100.00	100.00	0.00	0.00	0.00	88.64	0.00	100.00	0.00
11	100.00	100.00	100.00	0.00	0.00	0.00	88.64	0.00	100.00	0.00
12	100.00	100.00	0.00	100.00	0.00	0.00	100.00	100.00	100.00	100.00
13	100.00	100.00	0.00	100.00	0.00	0.00	100.00	100.00	100.00	100.00
14	100.00	100.00	0.00	100.00	0.00	0.00	100.00	100.00	100.00	100.00
15	100.00	100.00	0.00	100.00	0.00	0.00	100.00	100.00	100.00	100.00
Avg.	73.33	66.66	33.33	40.00	0.00	0.00	59.36	40.0	70.19	38.09

TABLE 12
Results of the testing process for locating faults in m_{best} using the space-shared oversubscription hypervisor (lower is better)

Id	$\mathbf{MR_1}$	$\mathbf{MR_2}$	$\mathbf{MR_3}$	$\mathbf{MR_4}$	$\mathbf{MR_5}$	$\mathbf{MR_6}$	$\mathbf{MR_7}$	$\mathbf{MR_8}$	$\mathbf{MR_9}$	$\mathbf{MR_{10}}$
0	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
1	0.00	0.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
2	0.00	100.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
3	0.00	100.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	2.63
4	0.00	100.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
5	0.00	100.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	0.00
6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
7	100.00	0.00	0.00	100.00	100.00	100.00	100.00	100.00	94.70	81.58
8	0.00	100.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	81.58
9	0.00	100.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
10	100.00	100.00	0.00	100.00	0.00	0.00	100.00	100.00	100.00	100.00
11	0.00	100.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
12	0.00	100.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
19	0.00	100.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
20	0.00	100.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
21	0.00	100.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
Avg.	13.33	80.00	0.00	93.33	86.66	86.67	93.33	93.33	92.98	77.71

TABLE 13
Results of the testing process for locating faults in m_{first} using the space-shared oversubscription hypervisor (lower is better)

Id	$\mathbf{MR_1}$	$\mathbf{MR_2}$	$\mathbf{MR_3}$	$\mathbf{MR_4}$	$\mathbf{MR_5}$	$\mathbf{MR_6}$	$\mathbf{MR_7}$	$\mathbf{MR_8}$	$\mathbf{MR_9}$	$\mathbf{MR_{10}}$
0	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
1	0.00	0.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
2	0.00	100.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
3	0.00	100.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	2.78
4	0.00	100.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
5	0.00	100.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	0.00
6	0.00	100.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
7	0.00	100.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
8	0.00	0.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
9	0.00	100.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
10	0.00	100.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
11	100.00	100.00	0.00	100.00	0.00	0.00	100.00	100.00	100.00	100.00
12	0.00	100.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
19	0.00	100.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
20	0.00	100.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
21	0.00	100.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
Avg.	6.67	86.67	0.00	100.00	93.33	93.33	100.00	100.0	100.00	86.85

TABLE 14
Results of the testing process for locating faults in m_{worst} using the space-shared oversubscription hypervisor (lower is better)

Id	$\mathbf{MR_1}$	$\mathbf{MR_2}$	$\mathbf{MR_3}$	$\mathbf{MR_4}$	$\mathbf{MR_5}$	$\mathbf{MR_6}$	$\mathbf{MR_7}$	$\mathbf{MR_8}$	$\mathbf{MR_9}$	$\mathbf{MR_{10}}$
0	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
1	0.00	0.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
2	0.00	100.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
3	0.00	100.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	2.70
4	0.00	100.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
5	0.00	100.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	0.00
6	0.00	100.00	0.00	100.00	100.00	100.00	0.00	100.00	100.00	0.00
7	0.00	0.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
8	0.00	100.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
9	0.00	100.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
10	100.00	100.00	0.00	100.00	0.00	0.00	100.00	100.00	100.00	100.00
11	0.00	100.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
12	0.00	100.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
19	0.00	100.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
20	0.00	100.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
21	0.00	100.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
Avg.	6.66	86.66	0.00	100.00	93.33	93.33	93.33	100.00	100.00	80.18