Noname manuscript No.
(will be inserted by the editor)

A methodology for validating cloud models using metamorphic testing

Alberto Núñez · Robert M. Hierons

Received: date / Accepted: date

Abstract Cloud computing is a paradigm that provides access to a flexible, elastic and on-demand computing infrastructure, allowing users to dynamically request virtual resources. However, researchers typically cannot experiment with critical parts of cloud systems such as the underlying cloud architecture, resource provisioning policies and the configuration of resource virtualisation. This problem can be partially addressed through using simulations of cloud systems. Unfortunately, the problem of testing cloud systems is still challenging due to the many parameters that such systems typically have and the difficulty in determining whether an observed behaviour is correct. In order to alleviate these issues, we propose a methodology to semi-automatically test and validate cloud models by integrating simulation techniques and metamorphic testing.

Keywords Metamorphic testing · Cloud computing · Simulation and Modelling

1 Introduction

Cloud computing is a paradigm that provides access to a flexible and on-demand computing infrastructure, by allowing the user to rent a number of virtual machines for a specific time slot. Currently, this paradigm is being adopted by several major enterprises in communications such as Google, IBM, Microsoft and Amazon. All of these organisations have invested billions of dollars in order to provide their own cloud computing solutions. In fact, the market is expected to rise from $40.7 billion in 2011 to more than $241 billion in 2020 [19].

The underlying complexity of cloud systems leads to testing being expensive and requiring much time and effort. Thus, it is desirable to design and develop formal testing methodologies for checking cloud systems [1]. Usually, formal testing approaches involve analysing the outputs generated by the system under test. If an analysed output is not the expected one, then the system under test has failed this test. The mechanism that reliably decides whether a test is passed or failed is called an oracle. Unfortunately, in some situations an oracle is not available or it is computationally too expensive to apply the oracle and alternative approaches must be used [21]. This problem arises in cloud systems, where there rarely is an oracle indicating whether the design of a cloud system is correct.

In general, we can say that a cloud system "works" if this system is able to perform several pre-defined tasks. For example, let us consider a data center with ten thousand physical machines, a communication network that interconnects them and a collection of services that provide virtualised resources for users, that is, a cloud system. If different users are able to rent virtual machines (in short, VMs) and execute applications using these VMs, we can state that this cloud system works. However, typically a system has to fulfil many additional conditions. Consider a cloud system where each machine contains a quad-core processor. If the cloud manager only uses two out of four CPU cores from each machine, then the cloud system is wasting half of its computing power. However, the users can still use the system by renting VMs and executing applications, because the number of physical cores managed by the

Alberto Núñez
Computer Science Faculty, University Complutense de Madrid, Spain
E-mail: alberto.nunez@pdi.ucm.es

Robert M. Hierons
School of Information Systems, Computing and Mathematics, Brunel University, UK
E-mail: rob.hierons@brunel.ac.uk

system is transparent to them. In this case, we can state that the system does not work correctly; there is inbuilt inefficiency.

In testing, validating or optimising a cloud system it is often necessary to customise management aspects of the system, like resource provisioning policies, hypervisors and configuration of virtualised resources. The process of adapting such aspects is typically expensive and time consuming and in some cases the researcher does not have access to a cloud system for which they have appropriate permissions. These factors have led to increasing interest in simulation. Usually, a cloud system modelled using simulation techniques consists of thousand of machines, networks, switches, users and other management modules. It may be necessary to assign values to thousands of parameters when configuring this model and so it usually is not feasible to test every possible input and check the outputs produced. Moreover, a single change in a module requires regression testing to be applied and this involves the tests being run again and the test output being checked, further increasing the cost of testing. These factors make testing expensive, with the checking of test output (the oracle) often being manual and so significantly contributing to the cost.

In this paper we propose a methodology that integrates a complete simulation platform for modelling cloud computing systems, with testing methods for checking the correctness of modelled cloud systems. The main goal of our research is to provide a mechanism that allows users to model both software and hardware parts of cloud systems, design new cloud system models and automatically test these models in a cost-effective manner. The main advantages of the proposed methodology can be summarised as follows:

- *Scalability*: The size of the simulated cloud can vary from several computers to thousand of machines.
- *Costless*: Using our proposed methodology does not require specific hardware to be executed. Also, a cloud is no longer required to perform experiments because the simulation platform can be executed in any computer.
- *Automatic testing*: Once users provide the required inputs for the testing process, the cloud model is automatically checked in order to analyse its correctness.

The rest of the paper is structured as follows. Section 2 describes the motivation for integrating a simulation platform and MT. Section 3 describes in detail our methodology for testing cloud systems using metamorphic testing. Section 4 presents the results of experiments. Section 5 presents some related work. Finally, Section 6 presents our conclusions and some directions for future work.

2 Motivation

There are several factors that make analysing or reasoning about the underlying architecture of a cloud system particularly challenging. First, cloud systems are usually very large and this fact hampers the analysis and study of these systems. Second, *virtualisation*, that is, the resources of the cloud provided to end-users are *virtual*, introduces additional complications because different VMs can be hosted in a single machine sharing the same resources among different users. Finally, we cannot oversee the vast number of users that are concurrently using a cloud system.

It is important to emphasise that cloud systems are built out of tens of thousands of commodity machines, where a simple failure in the system may produce catastrophic consequences. Therefore, ensuring the good functioning of these systems is a priority. As an example, in 2011 Amazon EC2 suffered an unexpected crash during network reconfiguration [10]. This crash affected more than 70 organisations, including *FourSquare*, the *New York Times*, *Quora* and *Reddit*, in some cases causing sites to remain off-line for many hours.

In this work we use the iCanCloud simulation framework to represent the behaviour of cloud systems [18]. Moreover, an additional framework, called E-mc^2 [4], is used to model the energy consumption of each hardware device in the cloud. Both simulation frameworks are currently available under the GPL3 license at `http://www.iCanCloud.org`. The main reasons to use these simulation frameworks is three-fold: First, these simulators are open source. Thus, the source code is available and can be modified if required. Second, these simulators provide highly detailed models of both the hardware part of the cloud system and the virtualisation schema. Third, there is a GUI that allows the environments to be easily configured. Moreover, the specific motivations for using simulation in our work are:

- Simulation is cheaper than performing experiments directly in a real cloud.
- The level of flexibility obtained by using simulation is much higher than when using real cloud systems.
- Scalability. In particular, the number of VMs that a single user may rent in public clouds is limited. In contrast, in a simulated environment cloud systems can be modelled with a fully customisable number of machines.
- Researchers can share simulation models.

Also, this paper investigates the application of *Metamorphic Testing* (in short, MT) to the testing of cloud systems since such systems are typically complex and often have no oracle.

MT was developed in order to test systems where there is no oracle or it is expensive to compute the oracle. The essential idea is that instead of checking the output o_1 produced when testing with one input x_1, we test with a second (follow-up) input x_2, observing output o_2, and check that o_1 and o_2 are related as expected. Thus, in MT there are two relations: the relation between the original test input x_1 and the follow-up input x_2, and the expected relation between the two outputs.

Consider, for example, the problem of checking a program f that should be an implementation of the trigonometric sine function. While it may be difficult to check whether the application of f to an input is correct, we know a number of properties of the sine function and any correct implementation should have these properties. One such property is that $sin(-x) = -sin(x)$. In order to check this metamorphic relation, having tested f with an input we also test f with the negation of the original input and check the above property. If the property does not hold then f must be faulty and it must have failed on at least one of the two inputs (the original value and its negation). In this paper we adapt the idea of MT to cloud systems.

There are three main reasons for using MT in this work. First, it has been shown in different application scenarios that MT alleviates the oracle problem, providing a better approach for testing large and complex scenarios than conventional testing techniques. Second, the effectiveness of the proposed methodology can be increased by adding new metamorphic relations (in short, MRs). MRs can be shared among different research groups in order to increase the applicability to a wider range of cloud configurations. Third, it is possible to concentrate 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 if MRs are grouped in categories, each one being responsible for analysing a specific feature of the cloud system.

MT has been used in the past to test Web Services [7], focusing efforts on checking the correctness of these applications. The approach presented in this paper focuses not only on testing applications, but also the underlying architecture of the cloud system on which the application is being executed. In other words, our approach aims to model and test the complete system and not only an application.

Although this objective seems promising, there are some limitations that must be assumed by the user. In MT, experienced users/domain experts are responsible for providing useful MRs directly corresponding to the most relevant properties of the system under test. MT provides a simple method for deriving follow-up test cases and, most importantly, is cost effective because the process of checking the accuracy of the system can be performed automatically without either an oracle or human interaction.

Since the core of the proposed methodology lies in the definition of MRs, the results provided by the testing process depend of how appropriate these are. Thus, appropriate relations should provide useful information about the correctness of the cloud model, while poorly defined relations would provide very little value. It is the responsibility of the user to provide appropriate MRs.

Although simulation provides a lot of advantages that makes it a powerful tool for research, it also entails some drawbacks. The main, and obvious, 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 is not intended to replace experiments in real clouds. Instead, the proposed methodology has been designed to help researchers to find and improve those configurations that obtain better results and discard those that are not valid. However, the final stage of the research must consist in executing the corresponding experiments in a real cloud system.

3 The proposed methodology

This section describes in detail the proposed methodology for testing cloud systems. Fig. 1 shows the basic schema of this methodology that integrates simulation and testing.

In this work, a cloud model is defined by a data center, the virtualisation schema and a cloud manager. The data center defines the underlying architecture of a cloud system, that is, the number of physical machines

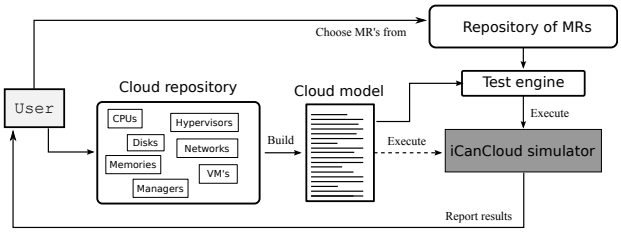

Fig. 1 Basic schema of the proposed methodology

allocated in the cloud system, the configuration of the basic subsystems of each physical machine and the network topology. The virtualisation schema consists of the configuration of each virtual machine, which is defined by setting the virtualised CPU, storage and memory. Finally, the cloud manager is an algorithm that allocates VMs in the available physical machines of the cloud.

Initially, users can create a cloud model by using the cloud repository. Each cloud model is represented by a text file readable by the simulator.

From now on, the term *user* denotes a person who uses our proposed methodology while *tenant* refers to a person who purchases services of the modelled cloud in a simulated environment. Basically, a tenant is defined by a set of purchased VMs, each purchased for a specific time-slot, and a set of applications that are executed in these VMs.

3.1 Modelling of a Cloud System

In order to illustrate the concepts described in this section, we present a running example. This example also presents a partial configuration of the model taken as input by the simulator (see Fig. 2). For the sake of clarity, only the most relevant parameters are showed.

Consider the data center of a modelled cloud system that consists of 128 physical machines (see lines [8-9]). Typically, each cloud has two types of machines, each one dedicated to a specific task: computing nodes and storage servers. A computing node is a machine used to host one or several VMs. Generally, these VMs are provided to tenants that request cloud services. In contrast, storage servers are in charge of managing remote data access. In this example, we use 112 computing nodes and 16 storage servers (see lines [16-17]). These nodes are connected through an Ethernet 10 Gbps network (see lines [3-5]).

Each physical machine must be configured by setting up its basic sub-systems: CPU, memory and storage. In this example, computing nodes use a quad-Core CPU, 16 GB of memory and 1 disk of 500 GB (see lines [26-31]), while storage nodes use a dual-Core CPU, 8 GB of memory and a 5-disk RAID system of 2 TB each (see lines [34-39]).

The virtualisation schema consists of the configuration of different VMs. Since virtualisation allows the separation of physical and logical resources, it is 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 VM is modelled as a portion of the resources

Fig. 2 Example of an input cloud model

```
network Cloud_128{

channel c extends DatarateChannel{
    delay = 125.0us;
    datarate = 10Gbps; }

parameters:
    int comNodes = 112;      // Computing nodes
    int stoNodes = 16;       // Storage nodes
    int nps = 64;            // Nodes per switch
    int numSwitches = 16;    // Number of switches

submodules:
    cloudManager: CloudManager {...}
    scheduler: CloudSchedulerFIFO {...}
    cn[comNodes]: Node {...}
    sn[stoNodes]: Node {...}
    s[numSwitches]:EtherSwitch {...}
    switchStorage: EtherSwitch {...}

connections:
 for i=0..numSwitches-1, for j=0..nps-1
   s[i].ethg++ <--> c <--> cn[(i*nps)+j].ethg++;
...
### Configuration of computing nodes
Cloud_128.cn[*].memory.size_GB = 16
Cloud_128.cn[*].memory.numDRAMChips = 8
Cloud_128.cn[*].cpuModule.CPUcore[*].speed = 51
Cloud_128.cn[*].numCores = 4
Cloud_128.cn[*].numStorageServers = 1
Cloud_128.cn[*].storageSystem.device[*].
    deviceType = "Disk_500GB_LI"
...
### Configuration of storage nodes
Cloud_128.sn[*].memory.size_GB = 8
Cloud_128.sn[*].memory.numDRAMChips = 8
Cloud_128.sn[*].cpuModule.CPUcore[*].speed = 43
Cloud_128.sn[*].numCores = 2
Cloud_128.cn[*].numStorageServers = 5
Cloud_128.sn[*].storageSystem.device[*].
    deviceType = "Disk_2000GB_LI"
...
```

of a given machine. Thus, a VM cannot exceed the resources of the physical machine where it is executed.

For example, a virtual machine (quad-Core 2.1 GHz, 0.5, 16 GB of RAM, 0.25, 500 GB disk, 0.1) indicates that we would like to use a machine providing a CPU at least as good as quad-Core and having at least 50% of its use, providing 4 GB of RAM and at least 10% of a storage system of 500 GB or better.

Finally, a cloud manager must be provided to complete the cloud system model. 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 algorithm. In this example we have used FIFO (see line 15 in Fig. 2).

The test engine module receives as input both a cloud model and a set of MRs, both previously selected by the user. Basically, this module automatically generates a set of follow-up test cases, which are built from the original model provided by the user. Each test is executed using iCanCloud and the results collected. The tool determines which instances of MRs were satisfied

and which were not and reports this to the user who is responsible for determining whether the results are acceptable.

3.2 Definition of Metamorphic Relations

We will consider MRs to formally compare the results obtained in the testing process. Next we formally define the pattern of our relations.

Definition 1 Let m be a cloud model and M' be a set of cloud models such that $m \notin M'$. Let T be a set of tenants. A *metamorphic relation* MR for m and T is the set of 5-tuples

$$MR = \left\{ \left(\begin{matrix} T, m, m', \\ T(m), T(m') \end{matrix} \right) \middle| \begin{matrix} m' \in M' \\ \wedge \\ p_1(m, m') \\ \Downarrow \\ p_2(m, m', T(m), T(m')) \end{matrix} \right\}$$

where p_1 is a relation over cloud models and p_2 is a relation over the cloud models and the execution of tenants on these models. □

The set of MRs is partitioned into three different categories, each representing a specific aspect of the system to be tested. Thus, users are able to choose a category in order to test a specific feature of the cloud, instead of testing the model completely.

- Performance: This set contains those relations directly related to the performance of a given system. Depending on 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 functioning of a given system.
- Energy-aware: This set contains those relations that check the restrictions regarding the energy consumed by a given system, where this system can be a single device, like a CPU, or a complete cloud system.

In order to illustrate our approach we show the full results for one MR from each group. Next, the definition of each MR used is presented. We use two cloud models and one set of tenants, denoted by m, m' and T respectively, where m represents the original model provided by the user, m' represents a variant automatically generated by the testing engine and T represents the workload executed in each model.

MR_{PER}: The CPU system of m having better performance than the CPU system of m' (and all other aspects being the same), denoted by $\Delta(m_{cpu}) > \Delta(m'_{cpu})$, implies that the time required to execute T over m' is greater than or equal to the time required to execute T over m, denoted by $time(T_{(m')}) \geq time(T_{(m)})$. Formally:

$$MR_{PER} = \left\{ \left(\begin{matrix} T, m, m', \\ T(m), T(m') \end{matrix} \right) \middle| \begin{matrix} m' \in M \\ \wedge \\ \Delta(m_{cpu}) > \Delta(m'_{cpu}) \\ \Downarrow \\ time(T_{(m')}) \geq time(T_{(m)}) \end{matrix} \right\}$$

MR_{FUN}: Let m_P and m'_P be two sets of physical machines that represent the physical machines used to model m and m', respectively. If $|m_P| > |m'_P|$, that is, the model m contains more physical machines than the model m', both using the same hardware configuration, then if $T(m')$ is executed successfully, denoted as $\uparrow T(m')$, $T(m)$ must also result in a successful execution, denoted as $\uparrow T(m)$. Formally:

$$MR_{FUN} = \left\{ \left(\begin{matrix} T, m, m', \\ T(m), T(m') \end{matrix} \right) \middle| \begin{matrix} m' \in M \\ \wedge \\ |m_P| > |m'_P| \\ \Downarrow \\ \uparrow T(m') \rightarrow \uparrow T(m) \end{matrix} \right\}$$

We say that $T(m)$ is a successful execution of T over m, denoted by $\uparrow T(m)$, if every application of every tenant in T is executed completely, that is, none of these applications are aborted by the expiration of any time slot of any VM. We denote by $\downarrow T(m)$ an unsuccessful execution of T over m.

MR_{ENE}: If the energy required to execute $T(m)$, denoted by $\Omega(T_{(m)})$, is greater than the energy required to execute $T(m')$, denoted by $\Omega(T_{(m')})$, that is, $\Omega(T_{(m)}) = \alpha \cdot \Omega(T_{(m')})$ $(\alpha > 1)$, then the time required to execute $T(m)$ must be less than the time required to execute $T(m')$, denoted by $time(T_{(m)}) < time(T_{(m')})$. Formally:

$$MR_{ENE} = \left\{ \left(\begin{matrix} T, m, m', \\ T(m), T(m') \end{matrix} \right) \middle| \begin{matrix} m' \in M \\ \wedge \\ \frac{\Omega(T(m))}{\Omega(T(m'))} > 1 \\ \Downarrow \\ time(T_{(m')}) > time(T_{(m)}) \end{matrix} \right\}$$

3.3 Description of the testing process

Conventional testing methods check whether the output(s) returned by the system under test are the expected ones or not. Schematically, let p be a system. Let 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 program p we obtain a sequence of outputs $p(t_1), p(t_2), \ldots, p(t_n)$. Therefore, if we have a specification implemented by an oracle, called s, then we find an error if there exists $t_i \in \mathcal{T}$ such that $p(t_n) \neq s(t_n)$. But, in general, we will not have an oracle and, therefore, we look for *evidence* regarding whether an output is correct. Our proposed methodology includes a testing framework whose main purpose is to (semi-)automatically test the suitability of a cloud model by using an approach inspired by metamorphic testing. In this case, a single test case is represented by a set of tenants T executed over a single cloud model m, denoted by $T(m)$. Actually, this part can be performed by users even without using the testing part. The idea will be to consider *variants* of the original model, compute the application of the set of tenants to these variants, $T(m'_1), T(m'_2), \ldots, T(m'_k)$, and compare the obtained results. Let us note that our considered variants are not *mutants* in the sense of *mutation testing* [11]: our goal is not to *kill* the variants in order to decide the goodness of the considered test (in this case, the set of tenants T) but to compare the different obtained results to detect a wrong or suboptimal behaviour of the original model.

The test engine module is in charge of automatically generating test cases, by following a given strategy S. Initially, we use a basic strategy that consists in sorting all the components that are used to generate a cloud model. The sort criterion is based in the quality of these components. Then, for each variant to be generated from the original cloud model, the strategy selects the next preferable component of the list. Each variant is represented as a text file (see Fig. 2). Basically, the test engine generates a copy of the original cloud model, that is, a variant, where the corresponding modification is applied, e.g, using a better disk.

Let M be a set of all possible cloud models, the testing engine module takes as input a cloud model $m \in M$, a set of tenants T and a set of relations MR. If $T(m)$ is a successful test case, then the set of variants $M' = \{m'_1, m'_2, \ldots, m'_k\} \subseteq M$ from the original model m is generated. Next, M' is used to automatically generate the follow-up test cases $T(m'_1), T(m'_2), \ldots, T(m'_k)$. Finally, these test cases are sent to the simulator to be executed.

4 Performance experiments

This section describes experiments carried out with two models that acted as case studies. The main hardware features of these systems are shown in Table 1. In order to simplify the exposition both cloud systems are homogeneous systems, that is, all physical machines in a cloud system have the same hardware configuration. In these experiments, different algorithms implementing the cloud manager (First-Fit and Round-Robin) were used to perform this task.

Table 1 Modelling of two different cloud systems

Device	Cloud A	Cloud B
Computing nodes	80	96
Storage servers	16	32
CPU	4-Core 2.1GHz	4-Core 2.1GHz
Memory	8 GB	4GB
Network	Ethernet 1Gbps	Ethernet 1Gbps
Storage	500GB	500GB
Switches bandwidth	10 Gbps	10 Mbps
Cloud Manager	First Fit	Round-Robin

The virtualisation schema used in both cloud models is presented in Table 2.

Table 2 Modelling of different VM types

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 test a cloud system, it is required that several tenants execute applications in it. In this work, three applications were modelled using the iCanCloud simulation platform: a Web server, a CPU-intensive application and a High Performance Computing application (in short, HPC).

The first application models the behaviour of a Web server. The second application multiplies two large matrices. Initially, these matrices are stored in the disk of a storage server. This application reads these matrices, performs the calculation and finally the result is written to disk. Finally, the third application models an HPC application, called BIPS3D [17].

Table 3 shows the modelling of 4 types of tenant. We say that each type of tenant represents a group of users in the cloud that have a similar behaviour. The configuration of this behaviour can be set by using three parameters: the rented VMs, the applications to be executed, and how each application is assigned to a rented VM. The first column refers to the name that defines the type of a tenant. The second column represents the number of simulated tenants of a given type. The next three columns represent the number of applications requested by each type of tenant. These are followed by three columns that represent the VMs rented by each

S-1 A cloud model m and a set of tenants T must be provided by using the GUI jointly with the cloud repository. Each tenant $t \in T$ represents a set of applications that must be executed completely in the cloud model m.
S-2 Executing $T(m)$, the original model provided in [S-1], by using iCanCloud.
S-3 Depending on the features to be tested on m, a set of MRs, taken from the MR repository, must be selected. Basically, these features depend directly on the requirements of the user that provides the cloud model in step [S-1], like testing the overall system performance, the energy consumed by a specific subsystem or the main functionality of the cloud model. These selected MRs are denoted by the set R.
S-4 Following a strategy S_R, that will vary according to the chosen set of MRs, the test engine uses the model m to generate a set of variants $M' = \{m'_1, m'_2, \ldots, m'_n\}$.
S-5 For each available model $m' \in M$
  S-5.1 Executing $T(m')$ by using iCanCloud.
  S-5.2 For each $r \in R$, we check whether $(T, m, m', T(m), T(m')) \in r$.
    S-5.2.1 If we find that some of the relations do not contain the tuple, then m' is discarded, a report indicating those relations that are not passed by the model m is sent to the GUI and go to step [S-7].
    S-5.2.2 If all the relations contain the tuple, then go to step [S-6].
S-6 A report indicating that the testing process has been executed correctly with no failures found is sent to the GUI.
S-7 End of the testing process.

Fig. 3 Testing methodology

Table 3 Modelling of different types of tenants

Type	Instances	Matrix	Server	HPC	VM_{small}	VM_{med}	VM_{large}	Mapping
t_{comp}	30	5	0	0	3	0	0	Random
t_{HPC}	25	0	0	3	0	0	8	First-Fit
t_{server}	30	0	10	0	0	5	0	Best-Fit
t_{mix}	15	5	5	3	3	3	2	Random

type of tenant. The last column gives the algorithm used to map applications to VMs.

We now describe the results of experiments that were performed by executing 100 tenants (see Table 3) over Cloud A and Cloud B (see Table 1), using three MRs (MR_{PER}, MR_{FUN} and MR_{ENE}) by following the methodology described in Fig. 3. In each testing process, a total of 100 cloud models were generated.

The next table shows the number of tests that successfully fulfilled each MR. We can observe that using Cloud A, the percentage of successful tests is greater than 90%. In contrast, there is a noticeable drop in the percentage of tests that fulfil MRs when Cloud B is used. This difference is mainly due to Cloud B being poorly configured. In this case, the parameter that configures the bandwidth of the switches in Cloud B is set to Mbps, instead of Gbps. This "mistake" causes a bottleneck in the system. However, we obtain similar results in both cloud models when using the MR focusing on the functionality of the cloud. This is because the functional behaviour of both models is correct, even though the obtained performance is not that expected. While Cloud A obtains good results when analysing performance and energy consumption, Cloud B obtains around 60% of successful tests. The main reason for this result lies in the saturation of the communications network. This leads to relatively little parallelism being obtained when executing different VMs and so there being little difference in the total time of execution of the original model and the variants.

Cloud Model	MR_PER	MR_FUN	MR_ENE
Cloud A	92/100	97/100	95/100
Cloud B	65/100	92/100	63/100

Fig. 4 shows the results of a subset of the experiments performed. For the sake of clarity, these charts show the generated models by modifying only two parameters of the cloud: CPU and number of storage servers. These charts show the tendency of the system when some changes are applied in the original model. In this case, the performance of each system is the target to be analysed, which is measured in seconds (less is better).

Using Cloud A, the MR_{PER} relation is fulfilled by 92/100 tests. This indicates that Cloud A is well configured for performance tests. In contrast, only 65 tests performed using Cloud B fulfilled the same relation. This means that some subsystem in the cloud is not working properly. The use of a better CPU does not lead to improved performance because the communication network is saturated.

When using MR_{FUN} with Cloud A, there are a few cases where the increase in the number of physical machines in the model leads to the execution requiring more time. This situation is mainly caused by the

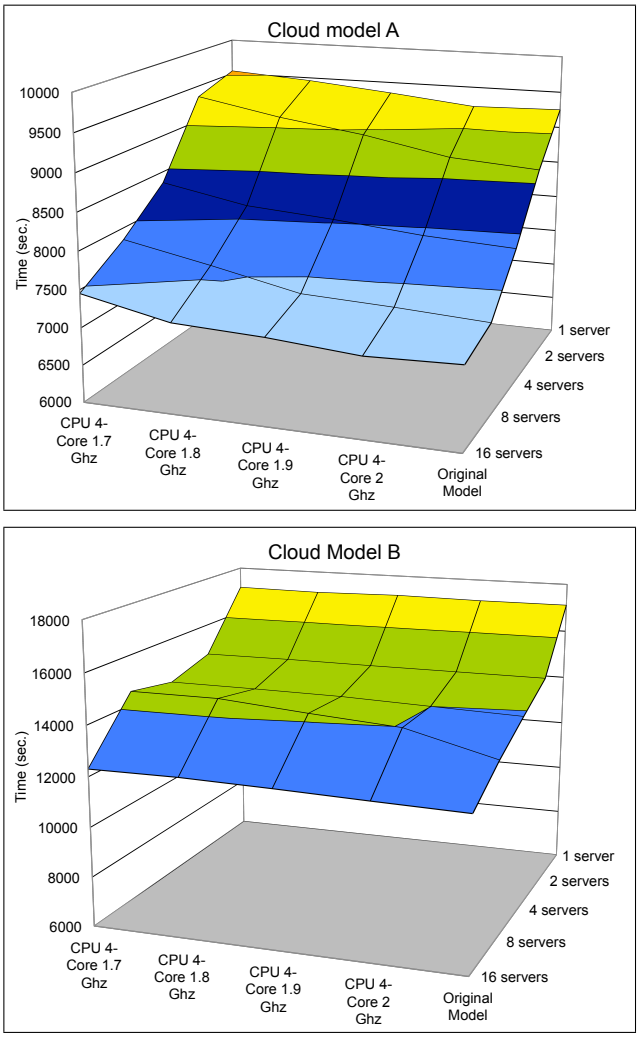

Fig. 4 Results of simulating cloud models A and B

BIPS3D application. When this application is executed in different physical machines that are not connected to the same switch, the time to perform network operations is higher. Similarly, 92/100 tests passed this relation when Cloud B is used. This means that this model works similarly when the size of the system is increased. That is, the bottleneck caused by the wrong configuration of the network does not affect this MR.

Finally, the relation MR_{ENE} is fulfilled by 97/100 test cases when Cloud A is used. The algorithm used for mapping applications to VMs is directly responsible for the 3 tests that do not fulfill this MR. In some cases, an application that requires significant resources is executed in a VM that contains limited resources, and vice versa. In these cases, powerful VMs are wasted and applications that require significant resources have long execution times, increasing also the energy consumption. In contrast, using Cloud B, only 63/100 tests fulfilled this MR. This is mainly caused by the wrong configuration of the network system. **Comment: I am not sure what the following sentence is meant to be saying so have not tried to rewrite it. Maybe we can discuss by email?** Since the switches are very slow, they cause an increment in the execution time of these applications, remaining these physical machines active, and consequently, consuming more energy.

The selection of the algorithm for allocating VMs to physical machines also affects the results. Cloud A uses a FIFO algorithm for allocating VMs to physical machines. Therefore, when the cloud manager receives a request to execute a VM, the first physical machine that contains sufficient available resources is selected. Consequently, even at the beginning of the execution a physical machine may have several VMs. In contrast, Round-Robin assigns the first available physical machine in the system, using a dedicated machine if possible. In this case, different VMs only share a physical machine when there are no idle machines in the system. While the FIFO algorithm leads to physical machines being shared by different VMs, where possible Round-Robin leads to a dedicated physical machine being assigned to each VM. Thus, Cloud A typically uses fewer physical machines in executing a workload, this being reflected in the energy consumption. This effect is more visible when the system is not well-configured, like Cloud B, causing a drop in the number of successful tests focusing on energy consumption.

There do not appear to be configuration that are perfect for all the applications, which means that in some cases one application demands a different configuration to obtain better performance. This explains why in all cases some of the MRs were not satisfied. When many tests do not fulfill an MR, the user has to check those tests that fail and explore potential solutions.

5 Related work

Testing [16] is probably the most widely used technique for checking the correctness of complex industrial systems. Therefore, testing should play an important role when deploying and configuring cloud systems. There are several proposals in the literature to test cloud systems. This body of work covers a wide range of techniques for testing different parts of cloud systems, like symbolic execution [8] and fault injection in the target system [13]. These approaches can be categorised into two major groups: testing *the cloud* and testing *in the cloud* (a.k.a. cloud-based testing or cloud testing). It is important to note that our proposal uses techniques for *testing cloud computing systems*, which is different from

testing in the cloud. In the first case, cloud systems are analysed in order to determine whether their underlying designs are appropriate or not. For this process, a real cloud system is not needed if we can simulate the cloud system on a regular computer. In contrast, testing in the cloud needs a cloud system to execute tests, which may be related (or not) to checking the underlying cloud system where these tests are executed.

Unfortunately, there are not many proposals focusing on testing cloud architectures using a formal approach, and we have to look either at proposals for formal testing in the distributed architecture [12] or specifically focus on web applications [15,14]. Although a web application can be executed in a cloud environment and a cloud system is inherently distributed, in these cases 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 or 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 [5].

In recent years, simulation has become a widely adopted loosely formalised approach for testing cloud systems. Basically, simulators build a model of the system to be simulated, such that it imitates the behaviour of the target system and then different measures, like performance and power consumption, are gathered by observing how the model works. Researchers have designed cloud models and then performed ad-hoc testing by manually simulating different scenarios. Among the available simulation tools that can be used to model and simulate cloud computing environments are CloudSim [2], GreenCloud [9], SimGrid [3] and iCanCloud [18].

Formal testing approaches usually assume the existence of an *oracle* to check whether the outputs returned by the system under test are those expected. However, in real systems we rarely have an oracle. Therefore, it is necessary to use alternative approaches to test the developed systems. 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 to test the validity of a system. In the frontier between formal and semi-formal approaches we find *metamorphic testing* [6]. Metamorphic testing was designed to alleviate the *oracle problem*. In fact, it is an automated testing method that employs expected properties of the target functions to test programs without human involvement. These properties relate inputs provided by the tests and outputs obtained from the tested system and are called *metamorphic relations*. Typically, after a test input has been used and the output observed, a second follow-up test input is generated and applied, a second output observed, and a process checks that the two outputs are related as expected.

Metamorphic testing has been used in very different application domains such as web applications [7], middleware [20] and machine learning [22]. This versatility suggests that there is scope to apply metamorphic testing to the testing of cloud systems.

6 Conclusions and Future work

This paper presents a methodology that integrates simulation techniques with testing methods for checking the correctness of cloud systems. In particular, the iCanCloud simulation platform has been used to model and simulate cloud systems, while techniques inspired by metamorphic testing were used to validate these models.

The main goal of this work is to facilitate the process of modelling and checking complete cloud systems semi-automatically. Thus, when the testing process is applied to a cloud model provided by the user, the generated results should provide useful information about how this cloud model is working.

In order to show the usefulness and applicability of our approach, different cloud systems have been modelled and tested using the proposed methodology. While the first cloud model fits well with the typical architecture of cloud systems, the second cloud model provides a slow network that does not fit well with the typical configuration of cloud systems. Moreover, we gave different MRs, each one focused on checking a specific aspect of the cloud.

The main objective of these experiments was to check whether the proposed method can reveal that a cloud model is not well designed. We expect that almost all tests will fulfill each MR when a cloud model is well designed. In contrast, when we applied the technique to a cloud model defined using inappropriate values for parameters, a significant number of tests did not satisfy the MRs.

Given an initial cloud model and different MRs, it was possible to automatically generate follow-up tests and check whether the MRs were satisfied. The results of the experiments showed the potential to determine that a cloud system model was poorly designed. In this case, the cloud that uses a slow network clearly shows a drop in the number of tests that fulfil MR focused on performance. This is because a slow network acts as a system bottleneck.

It is important to remark that, although one of the the cloud models was poorly configured, the same work-

load could be executed in both cloud models. The main difference lies in the time required to execute it. Since both models provided the require functionality, almost all tests satisfied the MR focused on functionality. Although the process of identifying mistakes in the design of a cloud system is not entirely automated, the user can focus on those MRs that contains a high percentage of failures and this should help them find mistakes in a cloud model.

Future work will present a formal definition of a cloud system in order to provide more consistent notation for including new metamorphic relations. Further analysis of a wide-range of cloud models will be performed. Finally, an extended collection of metamorphic relations will be defined.

Acknowledgements This research was partially supported by the Spanish MEC projects TESIS (TIN2009-14312-C02-01) and ESTuDIo (TIN2012-36812-C02-01).

References

1. Bertolino, A., Grieskamp, W., Hierons, R.M., Le Traon, Y., Legeard, B., Muccini, H., Paradkar, A., Rosenblum, D., Tretmans, J.: Model-based testing for the cloud. In: Practical Software Testing : Tool Automation and Human Factors, no. 10111 in Dagstuhl Seminar Proceedings, pp. 1–11. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany (2010)
2. Buyya, R., Ranjan, R., Calheiros, R.N.: Modeling and simulation of scalable cloud computing environments and the CloudSim toolkit: Challenges and opportunities. In: 7th High Performance Computing and Simulation Conference, HPCS'09, pp. 1–11. IEEE Computer Society (2009)
3. Casanova, H., Legrand, A., Quinson, M.: SimGrid: A generic framework for large-scale distributed experiments. In: 10th Int. Conf. on Computer Modeling and Simulation, UKSIM' 08, pp. 126–131 (2008)
4. Castañé, G., Núñez, A., Llopis, P., Carretero, J.: E-mc^2: A formal framework for energy modelling in cloud computing. Simulation Modelling Practice and Theory **39**, 56–75 (2013)
5. Chan, W., Mei, L., Zhang, Z.: Modeling and testing of cloud applications. In: 4th IEEE Asia-Pacific Services Computing Conference, APSCC'09, pp. 111–118. IEEE Computer Society (2009)
6. Chen, T.Y., Cheung, S.C., Yiu, S.M.: Metamorphic testing: a new approach for generating next test cases. Tech. Rep. HKUST-CS98-01, Department of Computer Science, Hong Kong University of Science and Technology (1998)
7. Chen, T.Y., Sun, C., Wang, G., Mu, B., Liu, H., Wang, Z.S.: A metamorphic relation-based approach to testing web services without oracles. International Journal of Web Services Research **9**(1), 51–73 (2012)
8. Ciortea, L., Zamfir, C., Bucur, S., Chipounov, V., Candea, G.: Cloud9: a software testing service. ACM SIGOPS Operating Systems Review **43**(4), 5–10 (2010)
9. Dzmitry Kliazovich Pascal Bouvry, S.U.K.: GreenCloud: A packet-level simulator of energy-aware cloud computing data centers. The Journal of Supercomputing **62**(3), 1263–1283 (2012)
10. Garber, L.: News briefs. IEEE Computer **44**(6), 18–20 (2011)
11. Hierons, R.M., Merayo, M.G., Núñez, M.: Mutation testing. In: P.A. Laplante (ed.) Encyclopedia of Software Engineering. Taylor & Francis (2010)
12. Hierons, R.M., Merayo, M.G., Núñez, M.: Implementation relations and test generation for systems with distributed interfaces. Distributed Computing **25**(1), 35–62 (2012)
13. Joshi, P., Gunawi, H., Sen, K.: PREFAIL: a programmable tool for multiple-failure injection. ACM SIGPLAN Notices **46**(10), 171–188 (2011)
14. Marin, B., Vos, T., Giachetti, G., Baars, A., Tonella, P.: Towards testing future web applications. In: 5th Int. Conf. on Research Challenges in Information Science, RCIS'11, pp. 1–12. IEEE Computer Society (2011)
15. Mei, L., Chan, W.K., Tse, T.H.: Data flow testing of service choreography. In: 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering, ESEC/FSE'09, pp. 151–160. ACM Press (2009)
16. Myers, G.: The Art of Software Testing, 2nd edn. John Wiley and Sons (2004)
17. Núñez, A., Fernández, J., Filgueira, R., García, F., Carretero, J.: SIMCAN: A flexible, scalable and expandable simulation platform for modelling and simulating distributed architectures and applications. Simulation Modelling Practice and Theory **20**(1), 12–32 (2012)
18. Núñez, A., Vázquez-Poletti, J.L., Caminero, A.C., Castañé, G.G., Carretero, J., Llorente, I.M.: iCanCloud: A flexible and scalable cloud infrastructure simulator. Journal of Grid Computing **10**(1), 185–209 (2012)
19. Ried, S., Kisker, H., Matzke, P., Bartels, A., Lisserman, M.: Sizing the cloud - a BT futures report. Understanding and quantifying the future of cloud computing. Forrester Research Report (2011)
20. Tse, T.H., Yau, S.S., Chan, W.K., Lu, H., Chen, T.Y.: Testing context-sensitive middleware-based software applications. In: 28th Annual Int. Computer Software and Applications Conference, COMPSAC'04, pp. 458–465. IEEE Computer Society (2004)
21. Weyuker, E.J.: On testing non-testable programs. The Computer Journal **25**(4), 465–470 (1982)
22. Xie, X., Ho, J.W.K., Murphy, C., Kaiser, G.E., Xu, B., Chen, T.Y.: Testing and validating machine learning classifiers by metamorphic testing. Journal of Systems and Software **84**(4), 544–558 (2011)