CloudSim Plus: A Cloud Computing Simulation Framework Pursuing Software Engineering Principles for Improved Modularity, Extensibility and Correctness

Manoel C. Silva Filho*†, Raysa L. Oliveira†, Claudio C. Monteiro*, Pedro R. M. Inácio†, Mário M. Freire†
* Departamento de Informática - Instituto Federal de Educação, Ciência e Tecnologia do Tocantins (IFTO)
Palmas, Tocantins, Brazil 77021-090
{mcampos, ccm}@ifto.edu.br
† Instituto de Telecomunicações (IT) e Departamento de Informática, Universidade da Beira Interior (UBI)
Covilhã, Portugal 6201-001
{d1365, m6476, inacio, mario}@ubi.pt

***Abstract*—Cloud computing is an established technology to provide computing resources on demand that currently faces several challenges. Main challenges include management of shared resources, energy consumption, load balancing, resource provisioning and allocation, and fulfilment of service level agreements (SLAs). Due to its inherent complexity, cloud simulation is largely used to experiment new models and algorithms. This work presents CloudSim Plus, an open source simulation framework that pursues conformance to software engineering principles and object-oriented design in order to provide an extensible, modular and accurate tool. Based on the CloudSim framework, it aims to improve several engineering aspects, such as maintainability, reusability and extensibility. This work shows the benefits of CloudSim Plus, its particular features, how it ensures more accuracy, extension facility and usage simplicity.**

I. Introduction

Cloud computing is a model to provide computing resources as an utility [1]–[4] that has gained wide adoption, mainly leveraged by the reduction of costs, management and IT personnel [5], [6]. The current state-of-the-art cloud technology relies on efforts from both the academy and industry, requiring execution of experiments to assess, improve and validate potential solutions. Examples of such efforts include the development of algorithms for improving fault-tolerance, resilience, resource provisioning and allocation, load balancing and scalability. Despite the cloud provides an on-demand charging model, the implementation of large scale experiments in a real infrastructure is costly, time consuming, limits the experiments reproducibility and prejudices measurement due to an uncontrolled environment.

Alternatively, several studies are performed resorting to computer simulation, enabling researchers to conduct their experiments without having to afford the costs of cloud services. Computer simulations also provide a faster way to run experiments that would take hours or days to run in a real infrastructure, in just a few minutes or seconds. They also allow large scale experiments to be executed using just a fraction of time and computing resources that would be required using a real cloud infrastructure, additionally favoring reproducibility and sharing of resources.

Within the aforementioned context, several simulation tools for cloud computing have been developed by the academy. The most widely used is CloudSim, a generalized and extensible simulation framework for cloud computing [7].

The characteristics that contribute to the broad adoption of CloudSim, are that: (i) it is developed in Java, a widely used programming language; (ii) the project is open source, enabling contributions from other developers; (iii) it was the first open source specialized cloud simulation framework [8]; (iv) it provides great flexibility to create simulation scenarios, where each scenario has to be implemented using Java code instead of using a rigid graphical tool.

The shortcomings of the current version of the framework that motivated this work were the following: (i) limited documentation; (ii) amount of duplicated code that jeopardizes maintainability, extensibility and testing; (iii) absence of functional/integration tests, important to ensure simulator correctness and validity; (iv) absence of design patterns [9] to improve several metrics of software engineering and object oriented design; (v) lack of conformance to some software engineering practices and recommendations such as SOLID principles [10]; (vi) lack of a more organized package structure to allow better understanding and modularity of the project; (vii) lack of a better class structure to allow third-party developers to implement missing features into the framework, without needing to change core classes.

CloudSim Plus project was initiated as an independent fork of CloudSim, pursuing the application of the best software engineering patterns, practices and recommendations. Its main contributions are: (i) improved class hierarchy and code, which is easier to understand; (ii) increased application of reusability principles; (iii) overall review and improvement of code documentation; (iv) re-structuring of project modules and

978-3-901882-89-0 @2017 IFIP

packages in order to simplify usage and to improve separation of concerns (SoC) principle; (v) addition of integration tests to cover overall simulation scenarios; (vi) completely new set of features described in details at the official web site.

The proposed simulator is an open source project available at http://cloudsimplus.org. The goal of this paper is to introduce CloudSim Plus, presenting its architecture, usage, advantages and new features. The paper is organized as follows: Section II presents the related work; Section III an overview of CloudSim Plus; Section IV the project architecture; Section V the improvements of CloudSim Plus over CloudSim; Section VI its main features; Section VII how to use the framework; and Section VIII presents the conclusion.

II. Related work

This section presents some cloud computing simulators found in the literature. CloudSim is the most widely used simulation framework for cloud computing [7]. It enables modelling of several characteristics and behaviours of a cloud provider, such as specification of infrastructure; implementation of management tasks such as resource allocation and VM management. Despite it is the most suitable framework for cloud simulation, it has some issues as introduced in the previous section.

WorkflowSim [11] is a tool for simulation of workflows in distributed environments. It implements mechanisms for workflow execution, introducing features such as failure models. The tool is focused on workflow simulations and it is not actively maintained anymore.

CloudNetSim++ [12] is a module for the OMNeT++ simulator [13] that allows modelling and simulation of physical network aspects of a cloud provider, making it difficult to simulate services of higher cloud computing layers.

III. CloudSim Plus overview

CloudSim Plus is a Java 8 simulation framework that allows modelling and simulation of different cloud computing services, ranging from infrastructure as a service (IaaS) to software as a service (SaaS) layers. It enables the implementation of simulation scenarios for experimentation, assessment and validation of algorithms for different goals. The framework allows developers to specify the characteristics of different entities of a cloud provider such as: (i) physical resources like datacenters, physical machines (hosts, servers or simply PMs) and network assets; (ii) logical resources like storage area networks (SANs), network topologies and applications; (iii) the virtualization layer that provides elements such as virtual machines (VMs) to enable virtualizing physical and logical resources; (iv) requirements and behaviour of applications and workloads.

It automates the management of all these resources that are provided as a service, working as virtual machine monitors (hypervisors or simply VMMs) that perform low level administration tasks such as: (i) VM lifecycle management (like creation, start, stop, destruction, placement and migration); (ii) management of active physical machines for energy saving; (iii) scheduling of VMs execution inside PMs and applications execution inside VMs; (iv) allocation of VMs for application and management of application lifecycle inside VMs.

CloudSim Plus allows simulation of entire cloud computing architectures and leaves developers concerned just in implementing specific new features they need to investigate, try and evaluate. By extending the basic mechanisms already provided by the framework, developers can focus in implementing algorithms to decide when and how these tasks are executed in order to achieve desired provider and/or customers goals. Such goals include load balancing, energy saving, fault-tolerance, scalability, elasticity and minimization of costs, SLA violations, network traffic, communication delay and so on.

IV. Architecture

CloudSim Plus is compound of different modules. CloudSim Plus API is the main module which represents the simulation framework API. It is the single module required to enable implementation of cloud simulation experiments. Figure 1 shows a simplified view of its package structure. Packages with a stronger color contain exclusive features of CloudSim Plus. Those lighter ones were introduced just to provide a better organization and separation of concerns (SoC) [14], but containing classes went through extensive refactorings and re-design.

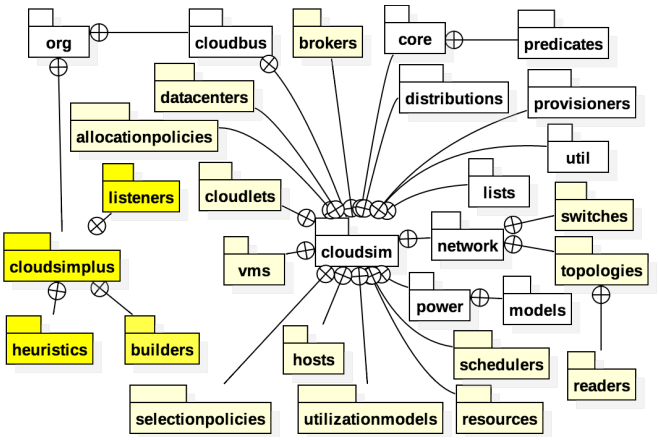

Fig. 1. CloudSim Plus API package structure.

The most relevant packages of CloudSim Plus and the classes they contain can be briefly described below:

- `distributions`: classes that provide generation of pseudo random numbers following several statistical distributions used by the simulation API. Additionally, they can be used by developers implementing their own simulations;
- `network`: classes for creation of datacenter network infrastructure allowing network simulations;
- `power`: classes to enable power-aware simulations, including power consumption models that can be extended by developers creating their simulations.

The original CloudSim project comes with classes that provide basic functionalities to quickly implement simulation

scenarios, such as models for utilization of resources like RAM, CPU and bandwidth. It has a set of base classes and few interfaces that must be extended by its users in order to introduce specific features for their simulations. Aiming a better organization of classes and interfaces in CloudSim Plus, the lightly highlighted packages in Figure 1 were introduced, grouping classes and interfaces with specific goals into well defined packages. These new packages are described below:

(i) `hosts`, `datacenters`, `vms`, `cloudlets`: group classes that provide different implementations for `Datacenters`, `Hosts`, `Vms` and `Cloudlets` (applications), including implementations of power- and network-aware components;

(ii) `allocationpolicies`: classes that provide mechanisms for a `Datacenter` to select a `Host` to place or migrate a `Vm`. The framework provides a worst-fit policy called `VmAllocationPolicySimple` that selects the `Host` with less available processor cores to place a given `Vm`;

(iii) `brokers`: classes that act on behalf of a cloud customer, attending his/her requests for creation and destruction of `Cloudlets` and `Vms`, assigning such applications to specific `Vms`. These brokers can implement decision making algorithms to prioritize submission of applications to the cloud, define how a `Vm` is selected to run a given application, etc. It includes an exclusive `DatacenterBroker` that uses a Simulated Annealing heuristic to map `Cloudlets` to `Vms`;

(iv) `schedulers`: classes to schedule the execution of multiple `Cloudlets` inside a `Vm` and the execution of multiple `Vms` inside a `Host`. It includes time-shared and space-shared schedulers, besides an exclusive implementation of the Completely Fair Scheduler used in recent versions of the Linux Kernel;

(v) `resources`: classes that represent physical and logical cloud resources such as hard disks, processor cores (processing elements or simply `Pes`), RAM, bandwidth and user files;

(vi) `utilizationmodels`: classes that models utilization of resources such as CPU, RAM and bandwidth, defining how a given resource is used by an application along the time.

Exclusive CloudSim Plus packages are presented in a stronger color and are briefly describe below:

(i) `listeners`: classes used to get notifications during simulation execution and then monitor the simulation scenario. Using such monitoring features it is possible, for instance, to allocate `Vms` on demand, when a specific condition is met;

(ii) `heuristics`: classes that provide the base to implement heuristics that could be used for different purposes such as mapping of `Cloudlets` to `Vms` and `Vm` migration decisions;

(iii) `builders`: classes that implement the Builder design pattern [15], working as object factories that make it easier to create multiple simulation objects such as `Hosts`, `Vms` and *Cloudlets*. This new feature will be discussed in Section VI;

A. Main CloudSim Plus classes

CloudSim Plus relies on several classes to provide its functionalities. Figure 2 presents a reduced UML diagram which indicates those main ones that have to be used in order to model and execute a simulation. These classes provide basic functionalities that can be extended by a researcher. Elements with a stronger color were introduced in CloudSim Plus to provide a solid base for improving framework extensibility, and will be discussed in Section V. Due to introduction of some interfaces, several existing classes were refactored by extracting abstract classes to remove code duplication among their children. Other classes were renamed to ensure a more consistent name convention. Accordingly, such elements are presented in a lighter color.

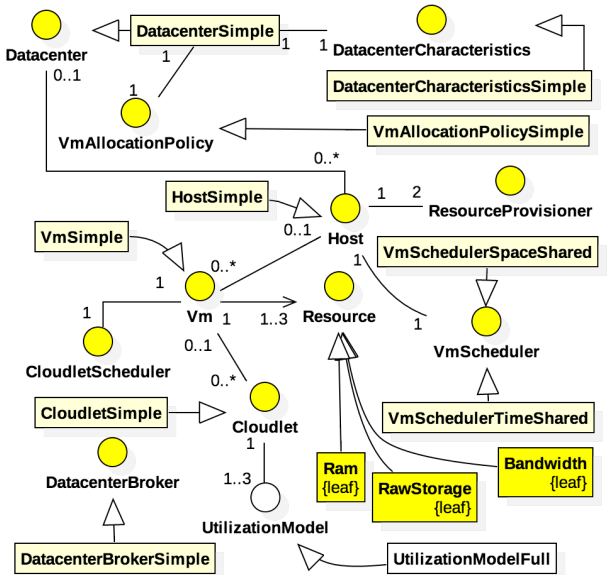

Fig. 2. Main classes involved in creating simulations using CloudSim Plus.

The new classes, highlighted with a darker color, represent bandwidth, storage and RAM resources of a `Host` or `Vm`. All these resources have certain attributes that need to be managed, such as capacity and current used amount. Accordingly, the interface `Resource` and implementing classes were introduced to remove all duplicated code used to manage these resources. The removal of duplicated code and inclusion of new test cases also ensure that functionalities work consistently and as expected. A description of the main classes used to create a simulation is presented below.

(i) `Datacenter`, `DatacenterCharacteristics` and `VmAllocationPolicy`: a `Datacenter` contains a set of physical machines (hosts, servers or simply PMs) that together provide the basic cloud infrastructure. Each `Datacenter` has attributes that define its characteristics, such as the costs associated with different physical resources from its `Hosts`. These attributes are defined by a `DatacenterCharacteristics` object. For each created `Datacenter`, a `VmAllocationPolicy` instance must be defined. This object decides which PM will host each `Vm`. The framework provides the `VmAllocationPolicySimple` implementation, a worst fit policy that allocates `Vms` into the `Host` with most available processor cores (`Pes`).

(ii) `Host, Pe` and `VmScheduler`: a `Host` represents a physical machine (PM) and for each PM, a list of processing elements (`Pes`) must be defined (the machine CPU cores). As the PM can host `Vms`, it is also required a scheduling algorithm that will be used to manage concurrent execution of multiple `Vms` in the host `Pes`. There are different `VmSchedulers`, such as time- and space-shared, that can be used.

(iii) `DatacenterBroker`: represents a software that acts on behalf of a cloud customer, receiving requests and performing requiring actions to attend them. These actions include submitting `Vms` to be allocated inside some `Host` of a `Datacenter`, submitting `Cloudlets` (applications) to be executed inside some of the created `Vms`, making decisions about what VM to select to place a given `Cloudlet`, etc.

(iv) `Vm` and `CloudletScheduler`: a `Vm` object represents a virtual machine that runs inside a `Host` and will execute applications (`Cloudlets`). A `CloudletScheduler` defines how concurrent execution of multiple applications is scheduled inside a `Vm`. It follows the same reasoning of `VmScheduler` and there are the same basic implementations available and a Completely Fair Scheduler, as described previously.

(v) `Cloudlet` and `UtilizationModel`: a `Cloudlet` represents an application that will run inside a `Vm`, abstractly defined in terms of its characteristics, such as the number of million instructions to execute, the number of required `Pes` and utilization models for CPU, RAM and bandwidth. Each `UtilizationModel` object defines how a given resource will be used by the `Cloudlet` along the time. Some basic `UtilizationModel` implementations are provided, such as the `UtilizationModelFull`, which indicates that a given available resource will be used 100% all the time.

V. CloudSim Plus Improvements over CloudSim

This section discusses some of the improvements of CloudSim Plus over CloudSim.

A. Extensibility Improvements

The colored interfaces in the class diagram of Figure 2, presented in Section IV, were introduced to improve framework extensibility. Accordingly, framework classes inherit from interfaces that provide a contract to be followed. They also allow conformance to the "program to an 'interface' not to an 'implementation'" recommendation [15] and also to the Liskov substitution principle (LSP) that states "objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program" [10]. Some classes were changed to interfaces, ensuring a consistent design, namely the new `DatacenterBroker, CloudScheduler, VmScheduler` and `VmAllocationPolicy` interfaces.

For each one of the introduced interfaces, the corresponding implementing classes were renamed so as to include the suffix `Simple` in their names, as already used in CloudSim for other classes. For instance, the `HostSimple` class replaces that `Host` (which is now an interface).

B. Reduced Code Duplication

Code duplication is a major problem for software maintenance and quality [16]–[22] and it is considered an anti-pattern [23]. It increases maintenance costs and inconsistent changes in the clones can introduce defects [24]. It also increases the amount of code to be changed and tested, sometimes leading developers to neglect tests. Code duplication also harms the design of application programming interfaces (APIs) such as those provided by this kind of project.

CloudSim project has a high level of duplicated code that reduces software quality and extensibility. In order to assess these issues, IntelliJ IDEA Ultimate 2016.2 IDE and its code duplication analysis tool was used for CloudSim 3.0.3 (the version that CloudSim Plus is based on), CloudSim 4.0.0 (the latest version) and CloudSim Plus 1.0. Table I below summarizes the results.

TABLE I
CODE DUPLICATION FOR CLOUDSIM 3 AND 4 AND CLOUDSIM PLUS 1.0.

Project	Number of duplicated code blocks	Sum of duplicated code lines	% of duplicated lines from previous version
CloudSim 3.0.3	415	3646	N/A
CloudSim 4.0.0	798	10973	+300.96%
CloudSim Plus 1.0	439	2536	-30.44%

These results were generated using all default options of IntelliJ code duplication analysis tool, under the `src/main/java` directory of the main module of each project, excluding all test suits and examples. Table I shows that CloudSim 3.0.3 has 415 duplicated blocks of code, totalling 3646 lines of code. CloudSim 4.0.0 almost doubled the number of duplicated blocks to 798, totalling 10973 lines of code and 300% increase in duplicated lines since the previous version. On the other hand, CloudSim Plus reduced the number of duplicated lines of code from CloudSim 3.0.3 in 30%.

The aforementioned results constitute one of the main reasons why CloudSim Plus started as an independent project and why it was forked from the version 3 of CloudSim instead of 4. While the re-engineering work was being performed to reduce the code duplication in CloudSim 3, the project moved to the 4.0 release, that increased the code duplication.

C. Tests and Code Coverage

CloudSim implements several unit tests using the JUnit tool, trying to ensure the correctness of the simulation framework. However, code coverage mechanisms are not used to assess which parts of the code are being tested and which are not, while providing percentage of overall coverage of tests along the project source code. In order to assess the amount of code that is covered by the existing unit tests, code coverage reports using Java Code Coverage Library (JaCoCo) were included in both projects (in CloudSim it was just for experimentation).

Table II shows that CloudSim 3.0.3 has 18% of code covered by tests, while CloudSim 4.0.0 decreased to 10%, a regression of 44% from the previous version. This is explained by the considerable increase in code duplication from one

version to another, showing that new features are not being tested. Such a table also provides an example on how code duplication leads to neglecting tests. On the other hand, CloudSim Plus officially supports code coverage reports and has 35% of its code covered by tests, representing an evolution of 94% from CloudSim 3.0.3, while it is still work in progress.

TABLE II
CODE COVERAGE FOR CLOUDSIM 3 AND 4 AND CLOUDSIM PLUS 1.0.

Project	Code coverage %	% of code coverage from previous version
CloudSim 3.0.3	18%	N/A
CloudSim 4.0.0	10%	-44.44%
CloudSim Plus 1.0	35%	+94.44%

VI. CLOUDSIM PLUS FEATURES AND ADVANTAGES

CloudSim Plus is a fork of CloudSim 3 that was re-engineered primarily to avoid code duplication for improved code reusability and to ensure better compliance to software engineering and object-oriented design principles and recommendations. It aims to provide a more extensible, modular, well documented, accurate and easy-to-use framework. It focuses on usage of software engineering standards and recommendations such as Design Patterns [15], SOLID [10] and general responsibility assignment software patterns (GRASP) [25] principles to achieve these goals. Following sub sections present the main features and advantages of CloudSim Plus over CloudSim.

A. Dynamic Arrival of `Cloudlets` and `Vms` and `Cloudlets` priorization

CloudSim Plus allows `DatacenterBrokers` to delay the submission of `Cloudlets` to the cloud infrastructure, simulating resources allocation waiting time. Te `DatacenterBrokers` are also able to submit dynamically created `Vms` and `Cloudlets`, allowing on demand allocation of resources. These features are missing in CloudSim and developers have to change core framework classes in order to enable such behaviors.

The priority attribute of `Cloudlets` is now in fact used by the new Completely Fair Scheduler to allow defining processes with higher priorities than other ones. These features thus enable the implementation of more realistic simulations.

B. Very Extensible and new `DatacenterBrokers`

`DatacenterBroker` is a fundamental entity for the simulation. It has to implement some policies for decision making in response to customer requests. CloudSim Plus provides a completely re-engineered package for `DatacenterBrokers`. The new `DatacenterBroker` interface publishes methods that allow researchers to simply implement such methods in order to define policies for selection of: VMs to execute `Cloudlets`; a `Datacenter` to place VMs inside its hosts; an alternative `Datacenter` in case of VM allocation failure, working as a fallback mechanism. All these features are just possible in CloudSim by modifying core classes of the framework.

A new `DatacenterBroker` that uses a Simulated Annealing heuristic to find a sub-optimal mapping between `Cloudlets` and `Vms` was introduced to provide an alternative for the Round-robin mapping that is provided by the single broker existing in CloudSim. The Round-robin mapping does not consider the fitness of a `Cloudlet` to a `Vm`. The introduced broker uses an heuristic to find a solution that reduces the mapping cost, defined as the number of idle `Vm` CPU cores.

C. Re-engineered network module and new set of interfaces

The module for network simulations was re-engineered in order to fix the issues caused by code duplication. Code duplication also leaded to neglecting tests, once there is no test case for those classes. The module used a lot of hard-coded values inside core classes for creating objects such as `Cloudlets` and `Vms`, which would be defined externally by the developer creating his/her own simulations. These values were clearly used just to perform the experiments presented in [26]. Further, they do not make the module directly reusable and even violate the open/closed principle (OCP).

A complete set of Java interfaces was introduced to start providing a more structured class hierarchy and to reduce code duplication. The classes `Datacenter`, `DatacenterCharacteristics`, `DatacenterBroker`, `Host`, `PowerHost`, `Pe`, `Vm`, `VmAllocationPolicy`, `VmScheduler`, `Cloudlet` and `CloudletScheduler` are now defined as interfaces.

These interfaces enable conformance to the Liskov substitution principle (LSP) [27], allowing instances of a given class to be substituted by different implementations at runtime. For instance, declaring an object using the `Host` interface allows different implementations of that interface to be used, such as a regular `HostSimple`, a `NetworkHost` or a power-aware `PowerHost` (currently inside the `hosts` package).

D. Event Listeners

The package `org.cloudsimplus.listeners` includes classes that implements the new `EventListener` interface to provide event notification for simulations. These notifications are related to changes in the state of simulation entities. The listeners enable notifying when (i) a `Host` updates the processing of its `Vms`, is allocated to a `Vm` or is deallocated to a `Vm`; (ii) a `Vm` has its processing updated or fails to be placed at a `Host` due to lack of resources; (iii) a *Cloudlet* has its processing updated, finishes its execution inside a `Vm`; or (iv) a simulation processes any kind of event, providing information about it.

These listeners were implemented using Java 8 functional interfaces, enabling the use of Lambda Expressions that allow a function reference to be passed as parameter to another function. Such a reference will be used to automatically call the function every time the listened event is fired. Researchers

developing in Java 7 can also use these listeners in the old-way by passing an anonymous class to them.

Listeners allow developers to perform specific tasks when different events happen and can be largely used for monitoring purposes and metrics collection.

E. Builder Classes

The package `org.cloudsimplus.builders` includes classes and interfaces that implement the Builder design pattern [15]. They were introduced as an alternative way to help creating simulation objects, such as `Hosts`, `Datacenters`, `DatacenterBrokers`, `Vms` and `Cloudlets`.

Creation of these objects requires many parameters and some of them have to be created before others (e.g.: `Hosts` have to be created before a `Datacenter`). Furthermore, it is a repetitive task that can lead to code redundancy. The builder classes help setting default values to be used when creating objects. Once these values are set, a single method suffices to create as many instances of the object as desired.

F. Integration tests

Integration tests allow an entire system to be tested in a more holistic manner. They allow to check how different software components work together and whether results are as expected. Sometimes, only the unit tests are not enough to assert the correctness of a software. Methods can give expected results when tested isolatedly, but may work in an unexpected way when interacting with other components. Accordingly, integration tests were introduced in CloudSim Plus to provide a more accurate simulation framework.

G. Software Design Quality Metrics

CloudSim Plus relies on a continuous integration service [28] that provides automatic tests and builds execution. This service ensures more reliability for CloudSim Plus users, given that any test or build failure will be immediately reported. Additionally, as project design and code quality directly impact software quality, CloudSim Plus uses a code analysis service [29] that automates code review and provides public code quality and coverage reports.

VII. How to use CloudSim Plus

A Java code snippet of a minimal simulation example is shown in Listing 1. The code that creates secondary objects such as schedulers and allocation policies is omitted due to space restrictions.

Listing 1. A snippet of a CloudSim Plus simulation example in Java

```
1  simulation = new CloudSim();
2  datacenter = createDatacenter(simulation);
3  broker0 = new DatacenterBrokerSimple(simulation);
4
5  Vm vm0 = createVm(broker0);
6  vmList.add(vm0);
7  broker0.submitVmList(vmList);
8
9  for(int i = 0; i < 2; i++){
10   Cloudlet cloudlet = createCloudlet(broker0, vm0);
11   cloudletList.add(cloudlet);
12 }
13 broker0.submitCloudletList(cloudletList);
14
15 simulation.start(); //blocks until finished
16 new CloudletsTableBuilderHelper(broker0.
       getCloudletsFinishedList()).build();
```

The depicted code initializes CloudSim Plus (line 1). After that, it instantiates a new `Datacenter` (line 2) and then a `DatacenterBroker` (line 3) that will act on behalf of a customer to submit his/her `Vms` and `Cloudlets` to the cloud. A `Vm` is then instantiated and submitted to the cloud by the broker (lines 5 to 7). After that, two `Cloudlets` are instantiated and submitted to the broker (lines 9 to 13). The simulation is started and the framework waits until it finishes, stopping the simulation automatically (line 15). Finally, executed `Cloudlets` are obtained from the broker and results are printed in the console in a tabular way (line 16).

Figure 3 shows the results for such a simulation scenario containing: 1 `Vm` with just 1 CPU core (`Pe`) capable of executing 1000 million instructions per second (MIPS); and 2 `Cloudlets` to be executed inside that `Vm`, needing to execute 10000 million instructions (MI) each one. The figure indicates that the `Cloudlets` were executed, both in the `Datacenter` with id 2 and inside the Vm with id 0. As there are 2 `Cloudlets` but just 1 `Pe`, and since this example uses a space shared scheduler for execution of `Cloudlets` inside the `Vm`, the first `Cloudlet` finished in 10 seconds, while the other waited the first one to finish.

```
Cloudlet|Status |DC|Host|VM|CloudletLen|CloudletPEs|StartTime|FinishTime|ExecTime
      ID|       |ID|  ID|ID|         MI|  CPU cores|  Seconds|   Seconds| Seconds
------------------------------------------------------------------------------------
       0|SUCCESS| 2|   0| 0|      10000|          1|        0|        10|      10
       1|SUCCESS| 2|   0| 0|      10000|          1|       10|        20|      10
```

Fig. 3. Results of a cloud computing simulation using CloudSim Plus.

VIII. Conclusion and Future Work

Cloud computing research relies on simulation experiments to develop, test, evaluate and tune-up potential solutions. It is difficult to accurately replicate a real system in a simulation experiment, mainly concerned in modelling the arrival of stochastic events such as workload bursts. Therefore, to contribute for valid results, a simulation framework has to (i) be well-designed (ii) get away from code duplication to avoid code degeneration (iii) be extensively tested and (iv) provide classes following software engineering principles. CloudSim Plus is accordingly aligned with these requirements. It was totally refactored to apply several of these principles to provide a state-of-the-art cloud simulation framework. As future work, there are several features planned to be included, such as SLA management and automatic vertical scaling of `Vms`.

Acknowledgements

This work is supported by the Portuguese Fundação para a Ciência e a Tecnologia (FCT) under the UID/EEA/50008/2013 Project and by the Brazilian CAPES foundation through the Proc. no 13585/13-4. We would like to acknowledge the work of the team behind CloudSim for providing such a framework.

REFERENCES

[1] S. S. Manvi and G. Krishna Shyam, "Resource management for Infrastructure as a Service (IaaS) in cloud computing: A survey," *Journal of Network and Computer Applications*, vol. 41, pp. 1–17, 2013.

[2] P. Mell and T. Grance, "The NIST Definition of Cloud Computing: Recommendations of the National Institute of Standards and Technology," National Institute of Standards and Technology, Gaithersburg, MD, Tech. Rep., 2011. [Online]. Available: http://www.nist.gov

[3] M. Masdari, S. S. Nabavi, and V. Ahmadi, "An overview of virtual machine placement schemes in cloud computing," *Journal of Network and Computer Applications*, vol. 66, pp. 106–127, may 2016.

[4] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, apr 2010.

[5] A. Lin and N.-c. Chen, "Cloud computing as an innovation : Percepetion , attitude , and adoption," *International Journal of Information Management*, vol. 32, no. 2012, pp. 533–540, 2012.

[6] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud Computing and Grid Computing 360-degree compared," in *Grid Computing Environments Workshop, GCE 2008*. Austin, TX: IEEE, 2008, pp. 1–10.

[7] R. N. R. Calheiros, R. Ranjan, A. Beloglazov, and A. F. D. Rose, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. Issue 1, pp. 23–50, 2011.

[8] R. N. Calheiros, R. Ranjan, C. A. F. De Rose, and R. Buyya, "CloudSim: A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services," p. 9, 2009. [Online]. Available: http://arxiv.org/abs/0903.2525

[9] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

[10] R. C. Martin, *Agile software development principles, patterns, and practices*, 1st ed. Pearson, 2002.

[11] W. Chen, M. Rey, and M. Rey, "WorkflowSim : A Toolkit for Simulating Scientific Workflows in Distributed Environments," in *IEEE 8th International Conference on E-Science (e-Science)*. IEEE, 2012, p. 8.

[12] A. W. Malik, K. Bilal, K. Aziz, D. Kliazovich, N. Ghani, S. U. Khan, and R. Buyya, "CloudNetSim++ : A Toolkit for Data Center Simulations in OMNET++," in *11th Annual High Capacity Optical Networks and Emerging/Enabling Technologies (Photonics for Energy)*. IEEE, 2014, pp. 104–108.

[13] A. Varga, "The OMNeT++ discrete event simulation system," in *European Simulation Multiconference (ESM)*. Prague, CZ: CiteSeer, 2001. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.331.1728

[14] R. Pressman, *Software Engineering: A Practitioner's Approach*, 7th ed. McGraw-Hill Education, 2009.

[15] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1st ed. Addison-Wesley Professional, 2007.

[16] R. C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*, 1st ed. Prentice Hall, 2008.

[17] M. Fowler, *Refactoring: Improving the Design of Existing Code*, 1st ed. Addison-Wesley Professional, 1999.

[18] I. Sommerville, *Software Engineering*, 9th ed. Pearson, 2010.

[19] N. T. Krishnan, D. Mazinanian, and G. P., "Assessing the Refactorability of Software Clones," *IEEE Transactions on Software Engineering*, vol. 41, no. 11, pp. 1055–1090, 2015.

[20] Z. Li, S. Lu, S. Myagmar, and Y. Zhou, "CP-Miner: finding copy-paste and related bugs in large-scale software code," *IEEE Transactions on Software Engineering*, vol. 32, no. 3, pp. 176–192, 2006.

[21] S. Demeyer, S. Ducasse, O. Nierstrasz, S. Demeyer, S. Ducasse, and O. Nierstrasz, "Detecting Duplicated Code," in *Object-Oriented Reengineering Patterns*, 2003, ch. 8, pp. 173–185.

[22] S. McConnell, "Why you should use routines...routinely," *IEEE Software*, vol. 15, no. 4, pp. 94–95,96, jul 1998.

[23] F. Palomba, A. De Lucia, G. Bavota, and R. Oliveto, "Anti-Pattern Detection: Methods, Challenges, and Open Issues," in *Advances in Computers*, 2014, vol. 95, ch. 4, pp. 201–238.

[24] E. Juergens, F. Deissenboeck, B. Hummel, and S. Wagner, "Do Code Clones Matter?" in *Proceedings of the 31st International Conference on Software Engineering*, ser. ICSE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 485–495.

[25] C. Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*, 3rd ed. Prentice Hall, 2004.

[26] S. K. Garg and R. Buyya, "NetworkCloudSim: Modelling parallel applications in cloud simulations," in *Proceedings of 4th IEEE International Conference on Utility and Cloud Computing, UCC 2011*. Victoria, NSW: IEEE, 2011, pp. 105–113.

[27] B. H. Liskov and J. M. Wing, "A Behavioral Notion of Subtyping," *ACM Trans. Program. Lang. Syst.*, vol. 16, no. 6, pp. 1811–1841, 1994.

[28] T. CI, "Travis CI - Test and Deploy with Confidence." [Online]. Available: http://travis-ci.org

[29] Codacy, "Codacy: Review less, merge faster." [Online]. Available: http://codacy.com