CloudSim Plus: A Modern Java 8 Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services

Table of Contents

Manoel C. Silva Filho[1,2]; Raysa L. Oliveira[2]; Claudio C. Monteiro[1]; Pedro R. M. Inácio[2]; Mário M. Freire[2]

[1]*Departamento de Informática - Instituto Federal de Educação do Tocantins (IFTO).* [2]*Instituto de Telecomunicações (IT) and Departamento de Informática, Universidade da Beira Interior (UBI).*

An always up-to-date white paper PDF is available here.

1. Introduction

The Cloud Computing paradigm has been driving innovation in different areas of knowledge and economic sectors by providing distributed, scalable, manageable and fault-tolerant computing resources over the Internet. The large adoption of Cloud computing services can be also explained by diverse reasons such as: its pay-per-use charging model, which enables cost reduction for customers; the rapid and automated allocation of resources, which enable applications to quickly respond to bursts; and full-featured web and console interfaces, which enable customers to configure hosted services easily.

The advances brought by Cloud Computing are backup up by academy and industry research and and continue to attract new researchers. Issues related with SLA fulfillment, optimal VM placement and migration, cost reduction, power efficiency, hotspot detection, load balance, fault tolerance, anomalies detection, security enforcement, traffic and latency reduction and so on, make of Cloud Computing an active and interesting research area.

Considering the complexity of a cloud infrastructure, computer-based simulation constitutes a rather attractive tool to carry out research in this field. Additional reasons to use computer-based simulation include: the need to model proposed solutions and evaluate them in a quick, cheap and repeatable way; the use of a controlled environment, which makes it easy to monitor and collect metrics; and the easiness to setup the required environment for experimentation.

In this article we present CloudSim Plus: a new, full-featured, re-designed, highly extensible and modern Java 8 framework for modeling and simulation of cloud computing infrastructure, services, underlying mechanisms and algorithms. CloudSim Plus enables researchers to model and simulate different cloud scenarios, by implementing them using Java. Such scenarios can be used to experiment existing and potentially new solutions for the issues mentioned above.

CloudSim Plus is an open source project available at http://cloudsimplus.org and Maven Central.

2. Overview

CloudSim Plus is based on CloudSim 3. It went through an extensive re-design and re-engineering process to provide an updated, modern, highly extensible and easier-to-use cloud simulation framework. These changes aim to enable sustainable project maintainability for long-term evolution. To achieve such goals, CloudSim Plus is founded on several software design and engineering metrics, principles and practices such as Coupling, Cohesion, Design Patterns, SOLID principles and other ones like Don't Repeat Yourself (DRY) and KISS.

3. Philosophy and Motivation

Software quality has become increasingly more important for the software industry, as can be seen by the number of current software design and development methodologies and processes such as Domain-Driven Design (DDD), Test-Driven Development (TDD), Behavior-Driven Development (BDD), Clean Code Programming and many more. Different tools have been used to collect and monitor software quality metrics to try detecting issues as soon as possible, such as Static Code Analysis and Integration Testing in an automated Continuous Integration environment.

Software quality helps reducing technical debt and avoids software erosion, delivering industry-standard quality products. We are strong adepts of the open source philosophy and we started by effectively contributing to CloudSim in 2015. However, since our team has different views of how the framework should evolve, we decided to start CloudSim Plus as an independent fork. We also believe that source code needs to be curated, without meaning it will restrict contributions, as it is successfully proved by important projects like the Linux Kernel.

Finally, enforcement of backward compatibility is a common concern in software development, mainly for public APIs such as those ones provided by a framework. However, such a concern slows down software evolution and we contrary understand that a cloud simulation framework is predominantly used for research purposes, even by industry. Cloud simulation scenarios are tightly coupled to a specific version of the simulation framework. After they produce the final results and the research goes on, new scenarios can be built using an updated version of the framework, without worrying about the older scenarios. Accordingly, we are not afraid of breaking compatibility so that we can foster project advance and innovation.

4. Architecture

CloudSim Plus is a Java Maven project that has a simple module and package structures. The entire project is compounded of 4 modules that were re-organized to directly inherit from the parent project, allowing a researcher to quickly have an overview of the structure. Redundant and out-of-date modules such as "distribution" and "documentation" were removed since building distribution artifacts and documentation is already automated using Maven.

Figure 1 presents the current project architecture and its modules are described as follows. The highlighted modules are new in CloudSim Plus.

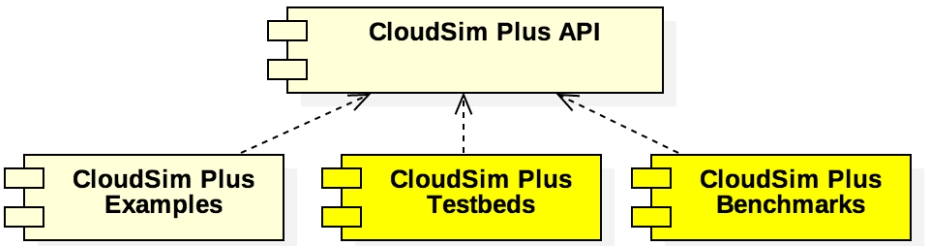

Figure 1. CloudSim Plus Modules

4.1. Modules

CloudSim Plus API is the main module that contains the framework API. It is the only independent module that is required to build simulation scenarios. Since such a module is available at Maven Central, there is no need to manually download the framework source code to build simulations. This module can just be added as a maven dependency to one's own project and he or she will be ready to start building simulation scenarios.

CloudSim Plus Examples provides the original CloudSim examples, with refactored, well organized and updated code to use the CloudSim Plus API. It also includes new examples for CloudSim Plus exclusive features.

CloudSim Plus Testbeds implements some simulation testbeds in a repeatable manner. It provides

base classes that allow a researcher to collect valid scientific results, such as average values and standard deviations, considering a specific confidence interval. They serve as examples on how to create broader testbed experiments.

CloudSim Plus Benchmarks is used just internally to measure the overhead of some CloudSim Plus features.

4.2. Package Structure

CloudSim Plus has a new package organization, which ensures better separation of concerns and make it easier to understand the project structure and to locate some class to use, extend or simply to analyse its source code and/or documentation. Packages with a strong color contain exclusive CloudSim Plus classes and interfaces, while the lightly colored ones were introduced to improve organization but usually just contain classes and interfaces from CloudSim. The white packages already existed in CloudSim.

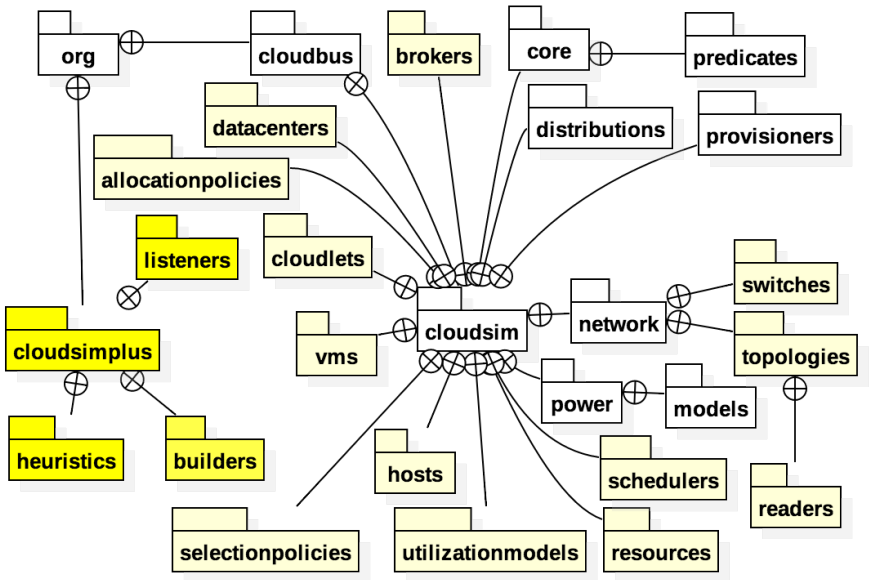

Figure 2. CloudSim Plus Simplified Package Diagram

CloudSim Plus also introduces documentation for every package, quickly explaining the goals of containing classes and interfaces, as well as the main class inside the package. Such a documentation is an excellent start point for researchers to get an overview of the framework.

4.3. How CloudSim Plus Works

Creating a cloud simulation using CloudSim Plus requires one to write a Java program that models the simulation scenario. The simplified diagram below presents the main interfaces involved in creating such scenarios. For every presented interface, there is one or more implementing classes that have to be actually instantiated (methods, attributes and several implementing classes were omitted for simplification).

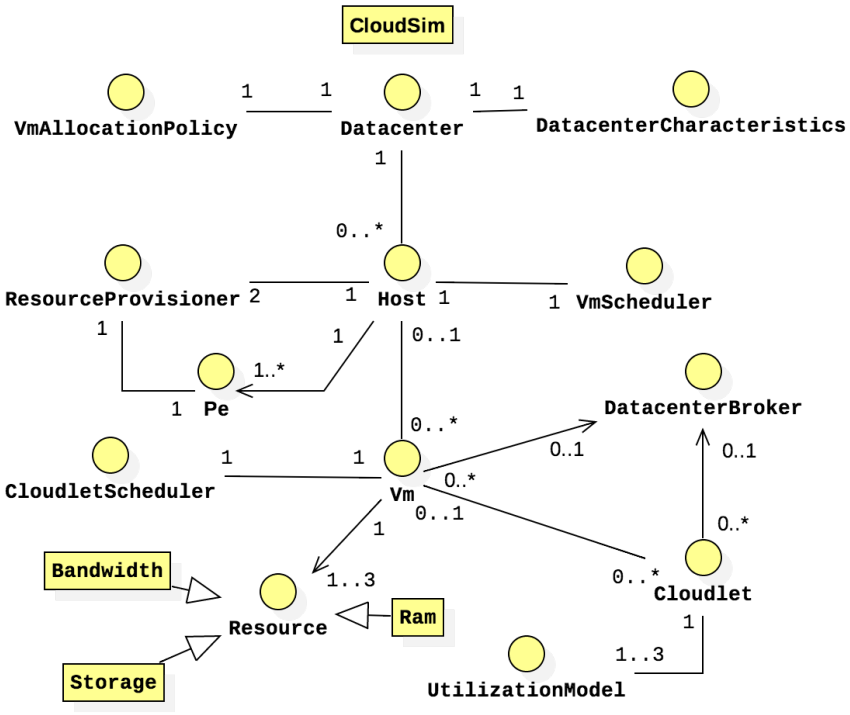

Figure 3. CloudSim Plus Simplified Class Diagram

The process of building a simulation scenario is described below. The goal of this section is to provide an overview of how to build cloud simulations using CloudSim Plus. It is not going to present the implementing classes for every presented interface. A complete simulation example is presented in the last sub-section of this section. More details on how to use CloudSim Plus are available at the official website.

`CloudSim` class is the first one that needs to be instantiated to start building a cloud simulation. CloudSim Plus just requires it to be instantiated using the default no-arguments constructor to initialize the simulation. The remaining objects that need to be instantiated are described as follows.

4.3.1. Datacenter

Instances of a `Datacenter` must be created to represent the different data centers that make up the cloud infrastructure. Each `Datacenter` has a set of characteristics (such as its timezone and costs for charging customers) defined by a `DatacenterCharacteristics` object. A `Datacenter` also needs a `VmAllocationPolicy`, which defines the policy used to select a `Host` to place or migrate Virtual Machines (`Vms`).

For each Datacenter, a set of `Hosts` must be created, which represents the physical machines with actual computing power. Each `Host` needs a `VmScheduler`, which defines the scheduling algorithm used by the `Host` to execute its `Vms` (there are 2 schedulers available). The `VmSchedulerTimeShared` enables executing `Vms` inside a `Host` in a preemptive, time-shared way. It allows different `Vms` to share CPU time when there are less CPUs than required by all `Vms`. The `VmSchedulerSpaceShared` is a non-preemptive scheduler that allocates physical CPUs to be exclusively used by each VM. If a `Vm` requires more CPUs than a `Host` possesses, it will not be placed inside that `Host`.

For each `Host`, `ResourceProvisioners` objects must be set. They define the policy used by a `Host` to

check whether a specific amount of resource is available for a requesting Vm. They then allocate that resource when such a Vm is created. A ResourceProvisioner must be defined to manage the allocation of RAM, bandwidth and CPU (Processor Element, Pe) from the Host to Vms.

4.3.2. DatacenterBroker

A Datacenter Broker is a software that receives requests from a cloud customer and is in charge to take required actions, on behalf of that one, to attend such requests. These requests usually are the creation and destruction of Vms and allocation of applications (called Cloudlets) inside some Vm. The broker is accountable for some decision making, such as: selecting a Datacenter and then a Host inside it to place each Vm; and mapping each Cloudlet to one of the available Vms. The instantiation of DatacenterBrokers is the next step, before creating and submitting Vms and Cloudlets to the cloud infrastructure.

The DatacenterBrokerSimple class always selects the first Datacenter to place submitted Vms, before trying other Datacenters when the allocation of a Vm fails in such a Datacenter. It provides a Round-robin policy to select a Vm to run each Cloudlet, without assessing Cloudlet requirements or provider and customer goals.

CloudSim Plus enables such DatacenterBroker behaviors to be changed in runtime, allowing a researcher to define the policies to place Vms and map Cloudlets to Vms, according to desired goals, without requiring the creation of a new DatacenterBroker. More details are discussed in the next section.

4.3.3. Vm

A Vm is a Virtual Machine that belongs to a specific cloud customer, which a DatacenterBroker acts on behalf of. It abstractly represents a Virtual Machine in terms of RAM, CPU, Storage and Bandwidth requirements. Each Vm needs a CloudletScheduler, which defines the scheduling policy to run applications inside the Vm. Beyond time- and space-shared schedulers, which work as the VmSchedulers used by Hosts, CloudSim Plus provides a implementation of the Completely Fair Scheduler used in recent Linux Kernels.

Vms must be created and submitted to the broker, that in turn will decide which Datacenter and Host each Vm will be placed into. When a Vm is submitted, the broker requests the selected Datacenter to create the Vm. If the Datacenter does not have a Host with enough capacity for such a Vm, the broker then forwards the request to other Datacenters, until the Vm is created or all Datacenters are requested.

4.3.4. Cloudlet

A Cloudlet abstractly represents an application running inside a Vm. Cloudlets and its related interfaces such as the UtilizationModel are used to define application models. A Cloudlet must define some resource requirements in advance (which currently are only CPU and storage). Such requirements can be used by a broker to decide how to map Cloudlets to Vms in order to attend such requirements and achieve provider and customer goals.

UtilizationModels define how different Vm resources will be used by a Cloudlet along the time, namely RAM, CPU and Bandwidth resources. Implementations such as the

`UtilizationModelStochastic` allows defining resource usage in a random way, using some pseudo random number generator (PRNG). A researcher can select the PRNG implementation to use, following a statistical distribution that meets his/her goals, such as Uniform, Normal (Gaussian), Exponential, Pareto distribution and so on. When summarizing data from multiple simulation runs, CloudSim Plus enables applying the Antithetic Variates Technique for uniform PRNG to reduce standard deviation.

`Cloudlet` resource utilization can also be defined using the `UtilizationModelPlanetLab` class, based on a trace file from Planet Lab Datacenters. On the other hand, the `WorkloadFileReader` class enables the creation of an entire set of `Cloudlets` based on different Datacenter traces file formats. Currently only the Standard Workload Format from The Hebrew University of Jerusalem is implemented.

4.3.5. A minimal and complete example

The source code below shows a minimal but complete simulation example. After presenting the objects required for a simulation, understanding the example is simple. Included comments provide a general view of what is being made and it is not the intention to go through the details. It just omits the package imports and the common structure of a Java class. The presented code is only the content of the `main` method, putting everything together into a single method, just for demonstration purposes. The complete example is available here.

```
Unresolved directive in README.adoc - include::../cloudsim-plus-
examples/src/main/java/org/cloudsimplus/examples/ReducedExample.java[tags="cloudsim-
plus-reduced-example",indent=0]
```

A more adequate and reusable example is available here, together with other examples. Examples of CloudSim Plus exclusive features can be found here.

5. Exclusive Characteristics and Features

CloudSim Plus is a full-featured simulation framework that has introduced exclusive features, presented below. The next sub-sections also discuss how such features can be used.

5.1. Dynamic Creation of Vms and Applications (Cloudlets)

CloudSim Plus allows on-demand creation of `Vms` and `Cloudlets`, without requiring creation of `DatacenterBrokers` at runtime. The `DatacenterBroker` class was refactored to enable submission of new `Vms` and `Cloudlets` during simulation execution, accordingly requesting the creation of such objects into the cloud infrastructure. It also enables delaying the creation of submitted `Cloudlets` and `Vms`, which may be used to control the time when the researcher wants these objects to be created. For instance, `Vm` creation can be delayed to avoid upfront allocation of resource that may not be required immediately.

5.2. Vm Scaling

Vm migration is a well-known mechanism to optimize allocation of physical resources, which can be applied for different goals, such as reduction of: costs, energy consumption and resource wastage by consolidating multiple `Vms` into the same `Host`; network traffic by placing inter-communication `Vms` as close as possible; SLA violations by ensuring that required resources will be available for hosted `Vms`; etc. It is a fundamental mechanism to provide the so called elasticity, which enables resources to be on-demand allocated or released.

However, Vm migration is an expensive operation that causes service downtime, introduces overhead and must be performed carefully. Sometimes Vm migrations can be avoided by simply scaling under or overloaded Vms. Appropriately, CloudSim Plus provides vertical and horizontal Vm scaling mechanisms.

These two different kinds of Vm scaling, additionally with Vm migration algorithms, can be used selectively by an Hypervisor to provide a very efficient Vm allocation policy mechanism to achieve intended goals. This mechanism can decide the time to perform Vm migration, vertical or horizontal Vm scale. Depending on specific conditions, one action can be favored over other ones or even different actions can be performed at a given time. CloudSim Plus Vm Scaling mechanisms are discussed below.

5.2.1. Vertical Vm Scaling

Vertical Vm Scaling performs on-demand down or up allocation of Vm resources such as RAM, Bandwidth and CPUs, according to under or overloaded Vm condition, respectively. Since actual hypervisors such as KVM and VMware ESX allows dynamically changing allocation of RAM and CPU for a Vm, that can be used to avoid VM migration in specific conditions.

Bandwidth scaling is simpler to be performed since it is a more abstract resource, different from virtualized RAM and CPU that are linked to the physical corresponding resource from a physical machine. This way, such scaling can be easily performed by routers and CloudSim Plus enables simulating this mechanism.

By using Vertical Vm Scaling feature it is possible to accommodate rising demand of applications running inside a Vm, without migrating the Vm or creating another ones. This CloudSim Plus feature enables performing simulations to assess, for instance, the workload limit that a Vm supports, for a given application model. Therefore, the creation of new Vms to balance the load can be postponed up to when they are really required, to avoid allocation of resources that stay idle for long time periods. This is discussed in the next sub-section.

5.2.2. Horizontal Vm Scaling

Horizontal Vm Scaling allows dynamic destruction or creation of Vms, according to an under or overload condition, respectively. Such conditions are defined by a predicate that can check different Vm resources usage such as CPU, RAM or Bandwidth, to define if a Vm is under or overloaded.

Depending on the model of an application running inside a Vm, by just performing a vertical up scaling when the Vm is overloaded may not be enough to support the demand. Consider a web

application relying on a Database Management Systems (DBMS) and a Web/Application Server. Often, it is required to create new Vms running separate instances of such a Web/Application Server to distribute user requests among them. It may be not enough just assigning more CPUs or RAM to a Vm, in an attempt to enable more threads to process user requests. Usually a single process may struggle to handle so many threads.

Alternatively, Horizontal Vm Scaling comes in handy by cloning a Vm to balance the load. This feature allows a researcher to implement and evaluate load balancing algorithms for dynamic workloads and burst conditions, by enabling the creation of new Vms to attend the demand. Some cloud platforms such as Amazon Web Services provide an Auto Scaling feature, that can be alike simulated in CloudSim Plus.

5.3. Parallel Execution of Simulations

Production of scientifically valid simulation results depends on several factors, which include the accuracy of the simulator, experiments reproducibility and collection of statistic metrics over multiple simulation runs. Although CloudSim Plus provides a lightweight, fast and easy way to model and run cloud simulation experiments, depending on the experiment scale, it may take several minutes to run. This is specially true when the workload is created from real and large datacenter traces.

CloudSim Plus was re-designed to enable running multiple experiments in parallel, in a multi-core machine, to reduce simulation time. Such a feature was enabled by changing every simulation attribute that was being managed in a static way, inside the `CloudSim` class, to be managed by instances of such a class. The benefits of this approach are two-fold: to initialize the simulation, one has just to instantiate a `CloudSim` object, instead of calling a static method with redundant parameters; each `CloudSim` instance owns all the objects and state that belong to a specific simulation, allowing each simulation to run in an independent and isolated manner.

The real time reduction that can be achieved by running simulations in parallel is tightly dependent of the simulation scenario and its scale. If the simulation is CPU-bound and is comprised of several runs, then the parallelization might provide large time reduction. On the other hand, small scale simulations or I/O-bound ones are not expected to take advantage of this feature.

An example available here shows how it is simple to parallelize simulation experiments in CloudSim Plus, using the Java 8 Stream API. Consider there is: a class called `ParallelSimulationsExample`, which represents a simulation scenario and contains a method `run()` to build and start a simulation; and then a list of instances of such a scenario with different configurations. Running each scenario instance is as simple as calling the single line of code below:

```
Unresolved directive in README.adoc - include::../cloudsim-plus-
examples/src/main/java/org/cloudsimplus/examples/ParallelSimulationsExample.java[tags=
"parallelExecution",indent=0]
```

Since `CloudSim` class was re-designed, it enables using the Parallel Streams feature of Java 8, which makes it straightforward to execute simulations in parallel, as presented above. The syntax `ParallelSimulationsExample::run` may look unfamiliar, but it is the new Java 8 Method Reference feature for Functional Programming. It enables passing functions as parameter to another function,

instead of just values. A reference to the `run()` method from the `ParallelSimulationsExample` class is being passed to the `forEach()` method. Thus, that line is not calling the `run()` method. Rather, it will be called inside the `forEach()` method when required.

5.4. Event Listeners

One of the features a cloud infrastructure must provide is the ability to monitor running services. Monitoring capabilities can be used in different ways by involved parties. The cloud provider can, for instance: collect resource utilization to charge customers in a pay-per-use basis; assess fulfillment of customer SLA; or optimize resource allocation to avoid under and over resource provisioning. Customers can, for instance, assess if the kind of resources he/she has contracted is appropriated to his/her demand and then take the required actions if they are not.

Despite cloud resources usage is charged in a pay-per-use basis, it is up to the customer to correctly configure the services he/she is using, to ensure an expected quality of service for his/her final users. Scaling mechanisms that provide the so called elasticity are normally enabled and configured by the customer. Otherwise, resources are not automatically scaled, for instance, to attend bursts. CloudSim Plus thus provides Listeners as a mechanism to monitor simulation in runtime, allowing collection of metrics, resource allocation decision making (such as Vm scaling) and granular simulation execution feedback. Since the final goal of a simulation is the collection of data to be processed, assessed and validated, Listeners enable researchers to collect such data at any time interval they need, and writing it in any desired portable format such as CSV, JSON, XML, YML or any other.

Despite CloudSim Plus is far easier to use and provides several new useful features, understanding and correctly implementing large scale simulation scenarios may be challenging. Listeners can be used by researchers to follow up simulation execution and to collect data for debug purposes. Listeners are a variation of the Observer Design Pattern and were implemented in CloudSim Plus using the Java 8 Functional Interfaces. These interfaces enable a researcher to use Lambda Expressions to define the code that will capture an event, without requiring an anonymous class for each event to be handled.

Listeners were designed to allow defining different handlers for the same event, also enabling the framework itself to use them internally, as a type-safe message-passing mechanism. More information about existing issues with the current mechanism can be found here. Such Listeners are currently being used to implement Integration Tests to assess the accuracy of the simulation framework. Examples using some Listeners can be found here. Some available listeners are presented below, grouped by the class they belong to.

5.4.1. Host Listeners

Such Listeners enable getting notifications about events happening inside a `Host`. Currently, only the `onUpdateProcessingListener` is defined, which enables receiving notifications when the execution progress of `Vms` inside a `Host` is updated. This `Vm` processing update is as if the `Host` operating system was allowing a `Vm` to execute some CPU instructions, that may impact usage of resources such as RAM, Bandwidth and obviously CPU too.

Using such a Listener, a researcher can monitor, for instance, if a specific Vm placement

configuration is enabling a balanced resource usage, i.e., whether resources are begin used in similar proportions along execution of Vms. An unbalanced resource usage situation sets a resource wastage scenario. That prevents new Vms to be placed at the Host, due to the unavailability of some resources while other ones are plenty.

5.4.2. Vm Listeners

Such Listeners enable getting notifications about events happening inside a Vm and are presented below:

- onHostAllocationListener: get notifications when a Host is allocated to a Vm, that is, when a Vm is placed inside some Host. Such a Listener can be used to log the different Hosts where a Vm may be placed along its life cycle, since the Vm may be migrated to a different Host when the previous one becomes under or overloaded. It also enables to check, in runtime, if a specific Vm allocation policy is placing Vms at the expected Hosts;
- onHostDeallocationListener: this is the opposite of the previous listener, being notified every time when a Vm is destroyed or moved from a Host. Vms must be created before applications (Cloudlets) are submitted to a broker. Sometimes applications finish running inside a Vm, then such a Vm becomes idle for long time periods. That may potentially detain other Vms to be placed on a Host, because resources that could be used by new Vms are provisioned to an idle one. Using this Listener, it is possible to determine the difference between the time when the Vm was destroyed and when it became idle. By this way, the broker policy that defines when Vms should be freed can be assessed to optimize resource allocation;
- onUpdateProcessingListener: get notifications when the execution progress of a Vm is updated. This event is fired by the event of same name from the Host where the Vm is placed. In the same way, it can be used, for instance: to log Vm resource usage along its execution, enabling detection of under and overload conditions; to determine if such resources are balanced; to detect and log SLA violations. Such a Listener is also used internally by the Horizontal and Vertical Vm Scaling mechanisms, presented above, to verify if a Vm is under or overloaded, aiming to perform down or up scaling, respectively;
- onCreationFailureListener: get notifications when a Vm fails to be placed into a selected Datacenter, due to lack of a Host with enough resources. This notification is fired by a DatacenterBroker which usually will try other available Datacenters to place the Vm. If too many Vm creation failures are happening, there may be issues on the Vm placement policy implemented by the DatacenterBroker or it may be a signal that the cloud infrastructure is not being able to attend the demand. In any case, this Listener enables logging such failures for further assessment. It may be used, for instance: to assess the suitability of a specific Vm placement policy to reduce the time to attend a Vm placement request; or to study current cloud provider restrictions and expansion needs.

5.4.3. Cloudlet Listeners

Such Listeners enable getting notifications about events happening inside a Cloudlet:

- onUpdateProcessingListener: get notifications when the execution progress of a Cloudlet is updated. This event is fired by the event of same name from the Vm where the Cloudlet is running. It enables monitoring applications resource usage along execution time, for instance,

to evaluate if such Vm applications are not contending the same resources. As a concrete example, if a Vm has multiple CPU-bound Cloudlets, their execution performance may be affected, increasing response and task completion time. Such a condition may lead to SLA violations and reveals a flaw at the DatacenterBroker policy used to select a Vm to run each Cloudlet;

- onFinishListener: get notifications when a Cloudlet finishes executing. Such a Listener can be used to collect different metrics such as wait, actual execution and completion time for each application. These metrics and application requirements may be used, for instance, in a back propagation process to enable a DatacenterBroker to improve its Cloudlet to Vm mapping policy. This listener enables collecting metrics immediately after Cloudlets finish, without having to wait the simulation to end. By this way, using techniques such as machine learning, the learning phase can be adjusted at simulation runtime.

5.4.4. CloudSim Listeners

Listeners inside the CloudSim class are the most general ones available in CloudSim Plus, which may have a broader applicability. The current implemented Listeners are described below:

- onClockTickListener: get notifications when the simulation clock advances, enabling to perform any desired action at a given time. Since CloudSim Plus allows dynamic creation of Cloudlets and Vms, as presented before, such a Listener can be used to submit new instances of those objects to a DatacenterBroker, simulating the dynamic arrival of customers requests and workload generated by final users;
- onSimulationPausedListener: get notifications when the simulation is paused. Such a Listener can be used, for instance, to collect partial simulation data after the simulation is intentionally paused at a given time, ensuring that the simulation state will not change during data collection;
- onEventProcessingListener: get notifications when any event is processed by CloudSim Plus. It is a more general Listener that provides the original data related to the happened event, that is totally dependent of the kind of event. Since CloudSim Plus is a discrete event simulation framework, any event that happens will be caught by such Listeners. As it is the most generic Listener available, it may have a wide applicability.

5.5. Strongly Object-oriented Framework

CloudSim Plus was comprehensively re-engineered to create relationships among classes, enabling chained calls such as cloudlet.getVm().getHost().getDatacenter(). This way, it stores references to actual objects, instead of just integer IDs to represent these relationships, which does not conform to an object-oriented design. These relationships can be seen at the Class Diagram already presented.

The line of code shown above provides a direct way to know what Vm a Cloudlet is running or will run, what Host such a Vm is or was placed into, and finally what Datacenter such a Host is settled down. The Null Object Design Pattern was also implemented to avoid the so propagated NullPointerException when making such a chained call.

5.6. Classes and Interfaces Allowing Implementation of Heuristics

Considering the large scale of cloud infrastructures, finding an optimal solution for issues such as Vm Placement is impracticable, since this is a NP-hard problem. Alternatively, heuristic techniques can be used to find a sub-optimal and satisfactory solution in a reasonable time. Some well-know heuristic methods include Tabu Search, Simulated Annealing and Ant Colony Systems. These methods usually start with an initial random solution for a defined problem and iterative and randomly look for other solutions. A fitness value to be maximized for each solution is computed by an utility function, then the solution finding stops when a desired fitness or number of iterations is reached.

CloudSim Plus provides a set of classes and interfaces to enable a researcher to build such heuristics for solving problems like Vm placement and migration. The interfaces provide a contract, by defining method signatures to: implement a solution generation and solution cost function (the fitness function is just the inverse of the cost); implement a function to update the solution search state; specify the number of maximum iterations, the probability for accepting each random solution and the predicate that defines when the solution finding must stop. The package org.cloudsimplus.heuristics contains such classes and interfaces and also includes a Simulated annealing heuristic to perform the map between `Cloudlets` and `Vms`.

5.7. Implementation of the Linux Completely Fair Scheduler

Implementations of the `CloudletScheduler` interface, as presented in the Section How CloudSim Plus Works, define the algorithm used by a `Vm` to schedule the execution of its `Cloudlets`. One of the criticisms against simulation experiments is differences between some behaviors of the actual system being simulated and the simulation itself, which may reduce the simulation accuracy. Process scheduling is one of the behaviors that was neglected in cloud computing simulations up to now. The scheduling algorithm impacts some application metrics such as wait time and task completion time. A bad scheduling may lead to processes waiting for long time periods to use the CPU or, when a process is assigned to a CPU, it is not given enough CPU time. That situation is called starvation and may cause SLA violations.

The Completely Fair Scheduler used in recent version of the Linux Kernel provides a very efficient policy to avoid the mentioned issues. As an actual scheduler, it considers assigned tasks priorities to define the time slice that each process is allowed to use the CPU. It also tries to be fair when allocating these time slices to avoid starvation of low priority processes. CloudSim Plus introduces a implementation of the Completely Fair Scheduler to increase the accuracy of processes execution in simulation environments.

The already existing `CloudletSchedulerTimeShared` class provides a simplistic implementation that does not take processes priority into account and, in fact, it does not perform a preemption process when there are more processes to execute than the number of available CPUs. As an example, consider there are 3 CPU cores and 12 processes to be executed. Such a scheduler will make all these processes to be executed simultaneously, each one using 25% of a CPU core capacity (3 cores / 12 processes), enabling 4 processes to run in each core.

Even in recent actual processors, the presented situation is not possible, since technologies such as Intel Hyper-Threading (HT) just enables up to 2 processes running at the same time on each CPU core. In a real scheduler, if there are, for instance, 2 Hyper-Threading CPU cores allocated to 4 processes that are not using the entire cores capacity, a fourth process cannot use this remaining capacity. That capacity is in fact wasted, forcing the process to wait one of the others to be preempted, to open room for it to use the CPU. Therefore, the simplistic `CloudletSchedulerTimeShared` may achieve better but inaccurate results. However, how CPU capacity is wasted by actual process schedulers can be assessed in CloudSim Plus.

5.8. Additional Characteristics

Besides all the exclusive features that have been presented, CloudSim Plus has additional characteristics that make it a promising cloud simulation framework. Some of them include:

- Completely re-designed and reusable network module. Totally refactored network examples to make them clear and easy to change.
- Throughout documentation update, improvement and extension.
- Improved class hierarchy, modules and package structure that is easier to understand, following the Separation of Concerns principle (SoC). For instance, power-aware `Host` classes and interfaces are included into the intuitive org.cloudbus.cloudsim.hosts.power package, as well as network-enabled ones are included into the org.cloudbus.cloudsim.hosts.network package. And if one needs to find a power or network-enabled `Vm`, he/she will intuitively know where to find it.
- As it is usual to extend framework classes to provide some specific behaviors, a researcher will find a totally refactored code that follows clean code programming, SOLID, Design Patterns and several other software engineering principles and practices. This way it is far easier to understand the code and implement a required feature.
- Integration Tests to increase framework accuracy by testing entire simulation scenarios.
- Updated to Java 8, making extensive use of Lambda Expressions and Streams API to improve efficiency and provide a cleaner and easier-to-maintain code.

6. Conclusion

CloudSim Plus is an updated cloud simulation framework that relies on the most recent advances of the Java language. It provides a more extensible, cleaner and easy-to-understand code that encourage developers to contribute. It uses industry-standard tools to:

- measure code quality;
- automate builds and the execution of unit and integration tests into a continuous integration environment;
- host its meaningful and extended documentation in a searchable way, enabling documentation versioning.

The redesign and refactoring performed in CloudSim Plus enabled reducing code duplication, making it easier to extend. The tools presented above provide an ecosystem to properly support

contributions by tracking code quality and software regression. In this process, several issues were detected and fixed, improving the framework correctness.

Finally, all the new CloudSim Plus features allow researchers to implement more realistic, complex and accurate simulations. Even the scale of simulation experiments may be enlarged by running experiments in parallel. All these characteristics and features make CloudSim Plus a promising cloud simulation framework.

7. Acknowledgements

CloudSim Plus is developed through a partnership among the Systems, Security and Image Communication Lab of Instituto de Telecomunicações (IT, Portugal), the Universidade da Beira Interior (UBI, Portugal) and the Instituto Federal de Educação Ciência e Tecnologia do Tocantins (IFTO, Brazil). It is supported by the Portuguese Fundação para a Ciência e a Tecnologia (FCT) (under the UID/EEA/50008/2013 Project) and by the Brazilian foundation Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) (Proc. no 13585/13-4).

We would like to thank these institutions for all the provided support and the EU-Brazil Cloud Forum for this opportunity.