
CloudSim Plus Documentation

Release 1.0.0

Manoel C. Silva Filho, Raysa L. Oliveira, Claudio C. Monteiro, Pedro

Nov 26, 2019

Contents:

1	Frequent Asked Questions (FAQ)	1
2	Publications	7
3	Syncing your fork or clone with the latest version from CloudSim Plus repository	9
4	Indices and tables	11

Frequent Asked Questions (FAQ)

Here, you find answer to the most recurrent questions in the CloudSim Plus mailing list. This FAQ was adapted from CloudSim, under the terms of the GPL license. Copyright (c) 2009-2012, The University of Melbourne, Australia.

1.1 Getting started

1.1.1 1. What is CloudSim Plus? What does it do and what doesn't it do?

CloudSim Plus is a toolkit (library) for simulation of Cloud computing scenarios. It provides basic classes for describing data centers, virtual machines, applications, users, computational resources, and policies for management of diverse parts of the system (e.g., scheduling and provisioning).

These components can be put together for users to evaluate new strategies in utilization of Clouds (policies, scheduling algorithms, mapping and load balancing policies, etc). It can also be used to evaluate efficiency of strategies from different perspectives, from cost/profit to speed up of application execution time. It also supports evaluation of Green IT policies.

The above are some common scenarios we envisioned and that users have been explored. Nevertheless, there is no limit on the utilization you can make from it: classes can be extended or replaced, new policies can be added and new scenarios for utilization can be coded. Think of it as the building blocks for your own simulated Cloud environment.

Therefore, CloudSim Plus is not a ready-to-use solution were you set parameters and collect results for use in your project. Being a library, CloudSim Plus requires that you write a Java program using its components to compose the desired scenario. Nevertheless, CloudSim Plus can be used to build such a ready-to-use solution.

1.1.2 2. Is CloudSim Plus the right tool for my project?

CloudSim Plus is a simulator, so it doesn't run any actual software technology. Simulation can be defined as "running a model of a software in a model of hardware". As it's all about models, specific technology details are abstracted. More information about differences between simulation, emulation, and other experiments methodologies can be found in:

Gustedt, J.; Jeannot, E.; and Quinson, M. Experimental methodologies for large-scale systems: a survey, *Parallel Processing Letters*, World Scientific, 2009, 19(3), 399-418.

1.1.3 3. What do I need to use CloudSim Plus ? How can I install it?

The only previous knowledge you need to use CloudSim Plus is basic Java programming (as CloudSim Plus is written in Java) and some knowledge about Cloud computing. Knowledge on programming IDEs like [Eclipse](#) or [NetBeans](#) is also handy, as they simplifies a lot application development tasks. Alternatively, [Maven](#) can be used to build your applications.

CloudSim Plus does not have to be installed: you just clone the GitHub repository at your machine, download the source code from GitHub and unpack on any directory, or use it as a maven dependency inside your own project and it is ready to be used. See [How to Use section](#) for more details.

The current version requires JDK 8.

1.1.4 4. How can I learn more about CloudSim Plus ?

You can start by the examples available in the `cloudsim-plus-examples` module that is available when you download the source codes. They start with simple scenarios and progress to more complex ones. After that, you will have a good idea on how you can put components together to build your own scenarios.

As the basic components are not sufficient for your project, you may start to study the API, so you will be able to use advanced features, and to extend or replace components.

Other source of information is the [CloudSim Plus Group](#).

1.1.5 5. Can CloudSim Plus run my application X?

No, CloudSim Plus is a simulator – it does not run real applications. It is intended to be used for simulating and experimenting with various scheduling and VM allocation algorithms.

1.1.6 6. I don't know anything about Java 8. How can I use CloudSim Plus?

You aren't required to know Java 8 to use CloudSim Plus. Despite there are some examples using Java 8 features such as [Lambda Expressions](#) and [Streams API](#), you can perfectly write Java 7 code to build your simulations. It's just needed to use Java 8 to build and run the examples.

Anyway, if you don't know Java 8 yet, it's a good time to start learning. There are lots of free and awesome content online. You can start by checking the links above or subscribing for the [JDK 8 Massive Open and Online Course: Lambdas and Streams Introduction](#) at the Oracle Learning Library. If new classes aren't scheduled yet, the videos are available at [YouTube](#). But if you can attend the course, you'll have more resources such as a exercises and a discussion forum.

1.2 CloudSim Plus components, communication, and events

1.2.1 1. What are the default behavior of components provided in CloudSim Plus package? How can I change them?

Datacenter behaves like an IaaS provider: it receives requests for VMs from brokers and create the VMs in hosts.

The most basic Broker (DatacenterBrokerSimple) provided in CloudSim Plus only submits a list of VMs to be created and schedules Cloudlets sequentially on them. **Usually you have to create your own Broker that implements the desired scheduling policy and/or policy for generation of VM requests and Cloudlets.**

To change default behavior, you can either extend these classes to add the intended behavior.

1.2.2 2. How can I code a periodic behavior to be adopted by entities?

This is done by setting an internal event to be fired periodically. Upon reception of the event, the handler for it is called, and the desired behavior is implemented in such handler method. Below we show how to do it for Datacenter class. The same steps can be used to enable such behavior in Broker as well.

1. Extend DatacenterSimple
2. Define a new tag to describe periodic event
3. Override processOtherEvent, to detect the periodic event and call a handler for it
4. Implement the handler method. Eventually, this method also schedules the next call for the event.

Important: your code must contain a condition for stopping generation of internal events, otherwise simulation will never finish.

```
class NewDatacenter extends DatacenterSimple {
    //choose any unused value you want to represent the tag.
    public static final int PERIODIC_EVENT = 67567;

    @Override
    protected void processOtherEvent(SimEvent ev) {
        if (ev == null){
            Log.println("Warning: "+getSimulation().clock()+" "+this.getName()+" Null_
↪event ignored.");
        } else {
            int tag = ev.getTag();
            switch(tag){
                case PERIODIC_EVENT: processPeriodicEvent(ev); break;
                default: Log.println("Warning: "+getSimulation().clock()+" "+this.getName()+
↪": Unknown event ignored. Tag:" +tag);
            }
        }
    }

    private void processPeriodicEvent(SimEvent ev) {
        //your code here
        float delay; //contains the delay to the next periodic event
        boolean generatePeriodicEvent; //true if new internal events have to be generated
        if (generatePeriodicEvent) {
            send(getId(), delay,PERIODIC_EVENT, data);
        }
    }
}
}
```

1.2.3 3. How can I create my own type of messages? How to make them be received by other entities?

The process is similar to the previous one. First, a new message tag has to be declared somewhere. Then, a handler for this message have to be added in the receiver of the message. The code is similar to the previous, with the exception

of the handler, that will not generate the event internally, but instead it will wait for some entity to send the message.

1.3 Policies and algorithms

1.3.1 1. What are the default scheduling policies and how can I change them?

CloudSim Plus models scheduling of CPU resources at two levels: Host and VM.

At Host level, the host shares fractions of each processor element (PE) to each VM running on it. Because resources are shared among VMs, this scheduler is called VmScheduler. The scheduler must be set to a host after it is instantiated.

In the VM level, each virtual machine divides the resources received from the host among Cloudlets running on it. Because such resources are shared among Cloudlets, this scheduler is called CloudletScheduler. The scheduler must be set to a VM after it is instantiated

In both levels, there are two default policies available: the first policy, xSpaceShared (x stands for VmScheduler or CloudletScheduler), required PEs by Cloudlets/VMs are exclusively allocated. It means that if there are more running elements (VMs or Cloudlets) than available PEs, the last objects to arrive wait on a queue until enough resources are free. In the second policy, xTimeShared, fraction of available PEs are shared among running elements, and all the elements run simultaneously.

Policies for VM and Cloudlet scheduling can be used in any combination. For example, you can use VmSchedulerTimeShared and CloudletSchedulerSpaceShared, or you can use VmSchedulerTimeShared and CloudletSchedulerTimeShared. It is possible even having a host running VMs with different Cloudlet scheduling policies, or a data center with hosts with different VM Scheduling policies.

To define your own policy, you have to extend one of the VmScheduler or CloudletScheduler classes.

1.3.2 2. What scheduling decisions should be implemented at VM level and what should be implemented at broker level?

The VmScheduler models the behavior of scheduling at virtual machine level like VMMs such as Xen and VMware ESX. Therefore, if you want to model behavior of this kind of software regarding distribution of resources among VMs running in the same host, this is the place where your new policy should be implemented.

Similarly, CloudletScheduler models the behavior of scheduling at the guest operating system level: given a number of applications currently running inside a VM, how available CPU resources should be divided among them? If you want to model this behavior, CloudletScheduler is the class to be extended.

There is one point that is not considered by either scheduler: given a number of Cloudlets, which one should start executing first? This kind of decision should be defined at Broker level, that will submit Cloudlets to VMs in the desired order, while it may delay the submission of other Cloudlets according to defined policies. For instance, if the current VMs that the broker is accountable for are overloaded, the submission of new Cloudlets to VMs can be delayed by the broker.

1.3.3 3. What is the default provisioning policy and how can I change it?

The provisioning problem consists of defining, among the available hosts in the data center, which one should receive a new VM requested by a user. Provisioning of hosts to VMs in data centers follows a simple strategy where the host with less running VMs receives the next VM. This behavior is defined in the VMAllocationPolicySimple class. To change this behavior, extend VMAllocationPolicyAbstract to define the new provisioning behavior, and pass this object when instantiating a Datacenter.

1.3.4 4. What class should I modify to implement my algorithm?

There are several places in CloudSim Plus where you can implement your algorithm depending on what the algorithm is intended to do. Usually you may start by extending some abstract class or even extending a concrete class. Below are several examples of classes that you may need to extend:

1. `DatacenterBrokerAbstract` – to define the way VM provisioning requests are submitted to data centers and the way cloudlets are submitted and assigned to VMs.
2. `VmAllocationPolicyAbstract` – to implement your own algorithms for deciding which host a new VM should be placed on. You can also implement dynamic VM reallocation algorithms (VM migration) by extending the `optimizeAllocation` method, which is called at every time frame and receives the full set of current VMs in the data center.
3. `PowerVmAllocationPolicyMigrationAbstract` – to implement power-aware dynamic VM consolidation algorithms that use VM live migration to dynamically reallocate VMs at every time frame. The main method to be overridden is `optimizeAllocation`.
4. `VmSchedulerAbstract` – to implement algorithms for resource allocation to VMs within a single host.
5. `CloudletSchedulerAbstract` – to implement algorithms for scheduling cloudlets within a single VM.

1.4 Advanced features

1.4.1 1. How can I code VM migration inside a data center?

VM migrations are triggered inside the data center, by an internal data center event. Therefore, triggering a migration means receiving and processing a `VM_MIGRATION` event. Such event is sent by a `Datacenter` to itself when it receives a list of VMs to migrate from the `VmAllocationPolicy`.

The datacenter sends the migration request message using a call such as:

```
send(this.getId(), delay, CloudSimTags.VM_MIGRATE, vm);
```

The `delay` field contains the estimated migration completion time. Therefore, when using it, the method that starts the migration process has to provide estimated completion time. After the delay, the event is received by the data center, which is interpreted as migration completed: therefore, from this time on the VM is available in the destination host.

1.5 Getting help

1.5.1 1. I have a question. What should I do?

The first thing you should do is reading this FAQ and the [documentation](#). If you are trying to implement some feature, check the examples. They usually implement the most required features. Try reading the source code of the classes involved in the feature you may need to implement. By understanding how such classes work you may get your answers.

If your question is not answered, you should try next previous discussions from [CloudSim Plus Group](#). Fragments of code that solve typical problems can be found there.

Finally, if you can't find an answer for your problem, send an e-mail to the discussion group. Please, try to be clear about your question, use appropriate English and show that you have tried by yourself to fix the issue. Such recommendations are likely to speed up the answer. If you are not getting meaningful answers or any answer at all, maybe it's time to read the [How To Ask Questions The Smart Way](#).

1.5.2 2. How do I report bugs, desirable features, unexpected behavior and other issues?

Please, use the [issue tracker](#) for that. This helps to speed up update process. Issues reported in the discussion group may take longer time to be added to the issue tracker.

1.5.3 3. Can you implement the specific feature X, required by my project/assignment?

Because we are a small team of developers, we can't add support to every scenario envisioned by users. But this is our intention: we provide generic classes and features that can be broadly used, and users develop case-specific behavior. Suggestion for new features that may be useful for significant number of users are welcomed and can be posted in the [issue tracker](#). Classes and features that are narrow in applicability and are intended to solve specific problems, though, are unlikely to be developed.

CHAPTER 2

Publications

Information about CloudSim Plus publications is available at the [official web site](#).

Syncing your fork or clone with the latest version from CloudSim Plus repository

3.1 Syncing your fork

If you want to contribute to CloudSim Plus, you have to fork the project so that you can make changes at your own copy of the repository. Once you have forked CloudSim Plus, this quick tutorial shows you how to sync your fork with the official repository in order to get the latest development version.

Open a terminal inside the directory where you cloned the CloudSim Plus repository and execute the following commands:

1. Add a remote to the official repository in order to track changes on it (this step has to be performed just once, if you already have execute it and try to execute again, git will just tell you that a remote called “upstream” already exists): `git remote add upstream https://github.com/manoelcampos/cloudsim-plus`
2. Get the latest version from the upstream remote (that is pointing to the official repository): `git fetch upstream`
3. Switch to your master branch to merge the changes from the upstream: `git checkout master`
4. Merge the changes from the master branch got from the upstream remote into your master: `git merge upstream/master`
5. As CloudSim Plus is a Maven project, usually it is worth to execute `mvn clean install` to clean the build directory, re-build the project and install the new updated CloudSim Plus jars into your local maven repository. This procedure is useful to avoid errors when trying to use a new version.
6. Push the changes to your fork at github: `git push origin master`

More information on syncing forks can be found [here](#).

3.2 Syncing your clone

If you just want to download CloudSim Plus source code so that you can use it into your project, study its code, documentation and examples, you can simply clone it locally at your computer. The clone doesn't create a copy of the repository into your GitHub account. However, if you are planning to make changes and request their inclusion in CloudSim Plus, you have to fork the project.

If you have just cloned the repository, to update your local copy with the official repository it is as simple as executing `git pull` at the command line.

CHAPTER 4

Indices and tables

- `genindex`
- `search`