On using CloudSim as a Cloud Simulator: The Manual

Research · December 2017

DOI: 10.13140/RG.2.2.30215.91041

Suchintan Mishra
National Institute of Technology Rourkela
6 PUBLICATIONS **17** CITATIONS
SEE PROFILE

On using CloudSim as a Cloud Simulator: The Un-official Manual

Suchintan Mishra · Manmath Narayan Sahoo

Abstract Cloud computing offers cheap, scalable and ubiquitous computing over distributed networks. Evaluation of newly proposed policies in cloud computing are either tested on testbeds that involve real hardware and real infrastructure(e.g. GINI) or through simulation tools. Simulation often saves cost(of using real hardware) and labour(of collecting results from heterogeneous computing elements). CloudSim has emerged as the numero uno cloud simulator and is constantly being developed and fine-tuned constantly. Although CloudSim is widely used in cloud research there is no proper documentation that would guide a novice researcher to understand and implement policies in CloudSim straightaway. This work is an attempt to practically enable researchers to: understand various CloudSim classes and their usage, find classes and their corresponding cloud research domain, use CloudSim as a standalone simulator, modify the CloudSim toolkit to include their policies. As an example we show how to include a new Host load balancing algorithm instead of the default First-Fit load balancing policy.

1 Introduction

Cloud computing is still at a very infant stage, as far as research and practical implementations are concerned. Although, cloud services are provided at commercial levels now, yet there are very few simulators and research platforms that provide the opportunity to researchers to evaluate their policies. Cloud architectures and underlying hardware are often heterogeneous and are mostly proprietary. Thus it becomes difficult to implement policies on real hardware. There are other factors too that limit the use of real cloud infrastructure. Considering the scale of real cloud infrastructure, it is difficult to change the policies on each of the Hosts and across each networking device. It is also difficult to collect the statistics from each individual element in the cloud network. The alternative approach, to testing benchmark applications is through the use of simulation tools. Simulation tools open up the possibility of testing a benchmark theory in a repeatable and controllable manner at low cost. Simulations tools require lesser computing and storage space. They provide a unified interface to collect the results and also provide methods and APIs that model cloud efficiently.

The usefulness of cloud computing is increasing everyday so is the demand for research in the area. Surprisingly, there are very few simulators that actually model cloud resources and allow for repeated and controlled simulation of policies. Grid computing, the predecessor of cloud computing gained attention during the early 2000s. This led to widespread research in the area. A number of Grid simulators like Gridsim[3] SimGrid [4], OptorSim[2] were released. These simulators, although capable of modeling grid applicationsm, fail to realize the multi-layer service model of clouds. These simulators also fail to model virtualization of resources. This gave rise to a compact java based multi-layer cloud simulator namely CloudSim [5]. CloudSim has the following basic properties that make it immensely popular:

1. Provides controlled and a repeatable environment to create and simulate cloud entities.

Suchintan Mishra
Department of Computer Science and Engineering
National Institute of Technology
Rourkela, Odisha-769008
E-mail: suchintan.mishra@gmail.com

Manmath Narayan Sahoo
Department of Computer Science and Engineering
National Institute of Technology
Rourkela, Odisha-769008
E-mail: sahoo.manmath@gmail.com

2. Can be extended to include user defined policies for cloud.
3. Provides changeable infrastructure modeling, changeable network architecture, federated cloud support.
4. Provides VM provisioning, Host provisioning, network provisioning and application provisioning.

CloudSim uses very little overhead as compared to real infrastructure. CloudSim is built on java as a *Maven project* and can be easily extended to include new policies and entities. However, the major problem that hinders researchers from using CloudSim is the lack of proper documentations as to how to use it. The description of classes and a guideline as to when to extend which class is a must in order to encourage researchers to use CloudSim. This paper bridges this gap between researchers and CloudSim to enable swift implementations and comparisons of cloud policies.

In this work, we show how novice researchers can take advantage of CloudSim to compare their theory to existing approaches. We show, how CloudSim can be used as a standalone simulator to compare among various inbuilt algorithms. We show how the simulation toolkit can be modified to include user defined policies. The paper contains steps to implement a new algorithm into the CloudSim toolkit and use it to compare among approaches. As a case study, we compare a newly implemented load balancing algorithm with existing First-Fit policy.

2 Architecture of CloudSim

In this section, we discuss the layered architecture of CloudSim and also look at some of the distinct classes and observe the dependencies among them. A careful observation of the class diagram reveals the essential entities and multi layered service models of cloud. Cloud computing provides services to the end users over a network. These services are hosted on a geographically distinct platform called a Datacenter. A Datacenter contains several computing nodes known as Hosts that a are responsible to provide the services that are desired by the end user. Hosts are physical computing machines that contain memory, storage and I/O resources. These Hosts might be homogeneous or heterogeneous depending on the cloud service provider. However the user is abstracted from the underlying hardware dependencies. This is achievable through *Virtualization*. Virtual machines are created for each user task that is submitted. These Virtual machines are portable, inter-operable, logically independent. Each VM created is allocated to a Host where all its computational requirements are met. Once the computations are done the results are returned to the user through the network. This ubiquitous nature of cloud services make them far fetched and desirable.

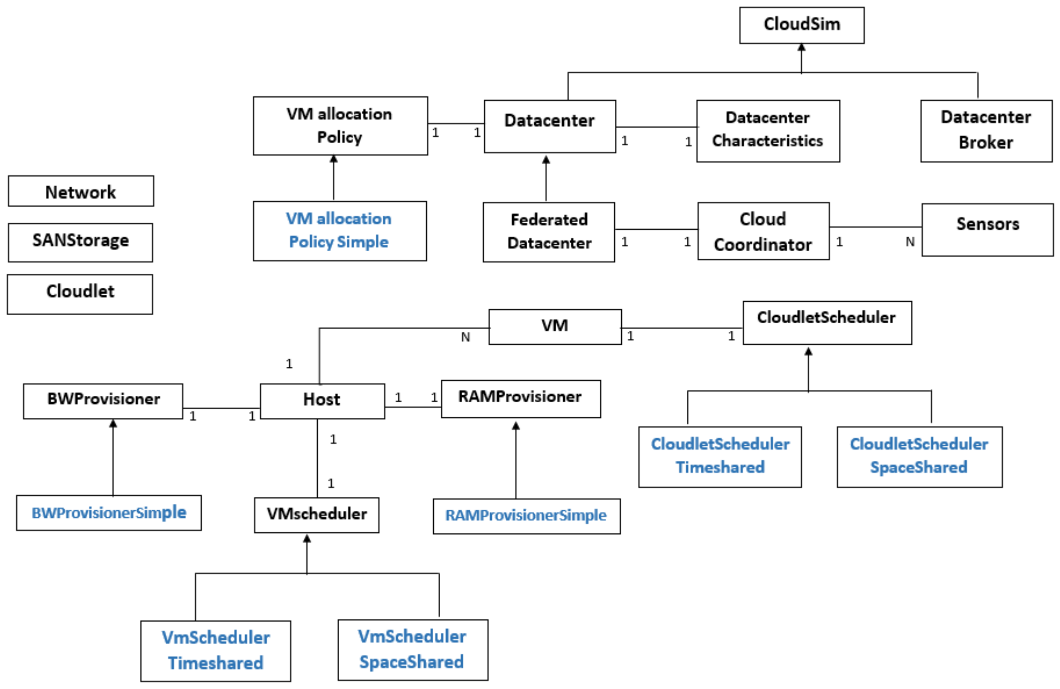


Fig. 1: Class Diagram for CloudSim

Table 1: Finding Cloud Entity characteristics

Cloudlet	**Host**	**Virtual Machine**
getStatus()	allocatePesForVm()	getBw()
getCloudletId()	getAllocatedMipsForVm()	getHost()
getCloudletLength()	getId()	getId()
getCloudletOutputSize()	getDatacenter()	getMips()
getCostPerSec()	getBw()	getSize()
getProccesingCost()	getAvailableMips()	getCurrentAllocatedRam()
getVmId()	getNumberofreePes()	getCurrentAllocatedBw()
getWaitingTime()	getRam()	getCurrentAllocatedMips()
getUserId()	getStorage()	getCurrentAllocatedSize()
getExecStartTime()	getTotalMips()	getCloudletScheduler()
getFinishTime()	getVmList()	getVmm()
isFinished()	getBwProvisioner()	getUserId()
getCloudletFinishedSoFar()	getRamProvisioner()	getNumberOfPes()
getSubmissionTime()	addMigratingInVm()	isInMigration()

CloudSim emulates all these characteristics through its packages and classes as shown in figure 1. There are a total of 12 packages in CloudSim containing several classes. Out of all the packages the most relevant to a cloud researcher is ***org.cloudbus.cloudsim***. This class contains the codes for modeling the various cloud entities like a Datacenter, a Host, a Cloudlet and VMs. Various scheduling and provisioning policies are also defined here. One can extend or overwrite these classes to define newer cloud entities, change existing cloud entities and create new policies. Due to the importance of the package in this work, we discuss the package in more detail.

All the classes in the package can be broadly categorized into two basic types: core entity classes and associated classes. The core entities include classes *Cloudlet, Host, Datacenter and VM.* The associated classes can further be categorized as classes that define policies(*VmScheduler, VmAllocationPolicy, CloudletScheduler, UtilizationModel, DatacenterBroker*) and classes that specify certain computational requirements(*Pe, NetworkTopology, SanStorage, Log, File, HarddriveStorage).*

The class Cloudlet, represents a unit of user submitted task in cloud. A constructor call (with parameters) to this class creates a *task* with properties defined by the parameters. This cloudlet (task) is to be assigned a VM(Virtual Machine). A VM in CloudSim is created by extending the VM class. Each VM has certain amount of computing resources dedicated to it. A VM runs on a Host and uses the resources of a Host(these resources are used by a cloudlet when it runs on a VM). A Host has a defined policy for provisioning memory and bandwidth, as well as an allocation policies for allocating, processing elements to virtual machines. A Host is associated to a Datacenter(represented by the Datacenter class).

Table 1 gives a list of functions contained in the major core entity(VM, Cloudlet, Datacenter) classes of CloudSim. cloud entities are created and managed using these functions hence a knowledge of all such functions is useful while finding characteristics of a Host, a VM or a cloudlet. Thus a cloud researcher can make use of these functions while writing policies. For example, a user might want to allocate the next VM to the Host having most available MIPS and this can be accomplished by using a object of Host class and calling the function getAvailableMips() for all such Hosts and making an comparison between the obtained values.

The job of other classes contained in the package are to support communication and simulation among the core entities.Each of the entities in CloudSim are associated with provisioning and regulatory policies. For example how two competing tasks run inside a VM or How to competing VMs run inside a Host or which Host is to be selected when a new VM is created. To address these policies CloudSim provides classes and methods that define how and when scheduling and provisioning should happen. The package org.cloudbus.cloudsim and org.cloudbus.provisoners provide certain classes that define these policies. We enlist some of the major classes and the functions one must overwrite in order to implement user defined policies in Table 2.

2.1 Overview of other Packages in CloudSim

org.cloudbus.cloudsim.core

Contains core functionalities required for simulation layers. Defines DeferredQueue, FutureQueue, SimEvenet and SimEntity.

org.cloudbus.cloudsim.predicate

Defines the various predicates that are used to match events to entities.

Table 2: List of CloudSim Functions to Overwrite and corresponding cloud research domain

Package	**Research Domain**	**Function to Overwrite**
org.cloudbus.cloudsim.VMAllocationPolicy	How to allocate Host to VM	allocateHostForVM()
org.cloudbus.cloudsim.VmScheduler	Proceesing of VMs in a Host	allocatePesForVM()
org.cloudbus.cloudsim.cloudletScheduler	Processing of cloudlets inside a VM	cloudletSubmit()
org.cloudbus.cloudsim.NetworkTopology	Network topology from a BRITE file	
org.cloudbus.cloudsim.DatacenterBroker	How to allocate VM to task	submitCloudlets()
org.cloudbus.cloudsim.provisioner.BwProvisoner	How to allocate Bw to VM	allocateBwForVm()
org.cloudbus.cloudsim.provisioner.RamProvisoner	How to allocate RAM to VM	allocateRamForVm()
org.cloudbus.cloudsim.provisioner.PeProvisoner	How to allocate PEs to VM	allocatePeForVm()

org.cloudbus.cloudsim.distribution

Contains various mathematical distributions like Lomax Distribution, Random Number Generator, Exponential Distribution, Gamma Distribution, etc.

org.cloudbus.cloudsim.lists

The package contains the lists of operations on the list of resources like cloudlets, Hosts, VM, etc.

org.cloudbus.cloudsim.network

The package contains all the classes required for the network topology. It contains delay matrix, routing algorithms, Topological information for the network.

org.cloudbus.cloudsim.network.datacenter

Defines a network within a Datacenter. Contains classes for networking elemnts such as core switches, aggregate switches, root switches.

org.cloudbus.cloudsim.power

Contains classes that can be extended to simulate a power aware Datacenter. Contains power-aware Datacenter policies. Allows extension and creation of power-aware policies and Datacenter entities.

org.cloudbus.cloudsim.power.lists

Conatins the class PowerVmList which is a collection of operations on lists of power-enabled VMs in the simulation.

org.cloudbus.cloudsim.power.models

Defines various power models for consumption of power. Any one of them can be extended depending on the system utilization.

org.cloudbus.cloudsim.provisioners

Defines policies for allocation of Bandwidth, RAM and PEs competing VMs inside a Host.

org.cloudbus.cloudsim.util

Contains classes that provide specific mathematical functions and contains the class execution time measurer.

3 CloudSim in Action

CloudSim can be used as a standalone simulator to study and analyse various existing policies and cloud entities also, it is possible to change the policies defined for various cloud operations depending on which class to extend. In this section we will discuss both the ways of using CloudSim.

3.1 CloudSim as a Cloud simulator

CloudSim can be run as a standalone simulator to simulate cloud computing scenarios using any popular java IDEs. All the packages and classes are compiled into a jar (in folder CloudSim3.0.3\jars\CloudSim-3.0.3.jar). The following steps can be followed to run CloudSim as a simulator.

- Create a new java application and add the above mentioned jar after unpacking CloudSim zip.
- Add commons-math3-3.2.jar, flanagan.jar to the library folder.
- Goto the folder CloudSim-3.0.3\examples\org\cloudbus\CloudSim\examples and copy the java files into the source package folder of the application u created in step-1.
- Run Example 1 to expect one VM running on one Datacenter with start and finish times.

3.2 Developing the CloudSim Toolkit

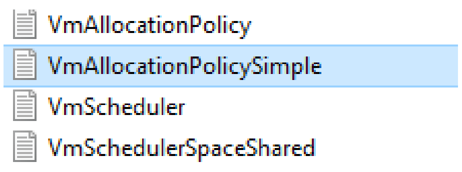
As discussed in the above section the jar files is the heart of the simulation. All the packages and classes are self contained in the jar. Hence to add new policies to the existing toolkit one must overwrite the methods in existing classes. Some of the major classes and their uses are listed in Table 1. One has to compile all the existing classes along with newly written classes. Following steps can be taken to modify the CloudSim simulation toolkit.

- Open the CloudSim folder as a maven project in any IDE. Add CloudSim-3.0.3.jar and commons-math3-3.2.jar to the existing project. Once the dependencies are added the maven project becomes error free.
- Copy any java file to extend in the folder CloudSim-3.0.3\sources\org\cloudbus\CloudSim and rename the newly created file.
- Open the newly created file in the IDE. Now the IDE shows the new file with errors. Edit and correct the errors in the new file. Once the new file is error-free proceed to the next step.
- Apache ANT is a toolt to build projects. Instructions to download and install it is available at the official website [1]
- Go-to the base folder in CloudSim(CloudSim-3.0.3\) and locate the build file. From that location open command prompt and use the command ANT (you should make sure ANT_HOME is set and ANT_HOME\bin is added to the path).
- once build is successful all the classes get compiled and on observation one can find a newly build jar in CloudSim-3.0.3\jars\CloudSim-new.jar this jar contains the newly added classes. One has to add this new package to any project created in section 3.1 and can import the newly created class.

3.3 Case Study:Load Balancing Hosts in CloudSim

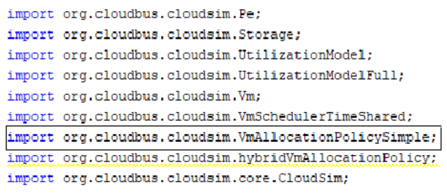
The class org.cloudbus.cloudsim.vmAllocationPolicy can be extended to address user defined policies for Host Load Balancing. Load Balancing is the process of spreading the load across avialbale resources so as to decrease the overall response time of a task and to simultaneously increase the utilization of the resources. By default, CloudSim provides first-fit as the default Host selection policy. Once any task comes all the Hosts are scanned and the first Host that is fit enough is assigned the VM. We run several scenarios using this default approach then change the VM allocation policy to Equally spread current execution and run several scenarios. We collect the overall response time in each of the scenarios and map them to analyze which algorithm performs better. In the process we will also note down the CloudSim steps followed.

- Goto folder CloudSim-3.0.3\sources\org\cloudbus\CloudSim and loacte the file VmAllocationPolicysimple.java this file contains the methods to allocate Hosts for VM shown in figure 2.a. The Host having most number of free Processing elements is chosen as the candidate Host to run the VM.
- Create a project as shown in Section 3.1 and run example 4 using the VmAllocationPolicySimple as the VM allocation policy as shown in figure 2.b.
- Run the example using increasing number of tasks and record the average response time .
- Copy the file CloudSim-3.0.3\sources\org\cloudbus\CloudSim\VmAllocationPolicysimple.java and rename is to hybridVmAllocationPolicy.java
- Open hybridVmAllocationPolicy.java and change the class anme and constructor name from VmAllocationpolicySimple to hybridVmAllocationPolicy. Once changed it becomes error free.
- In hybridVmAllocationPolicy locate allocateHostForVm() method and change it according to algorithm 1.
- ANT the CloudSim toolkit as explained in Section 3.2 and figure 2.c to build CloudSim-new.jar add this jar (shown in figure 2.d) to the project created in step-1.



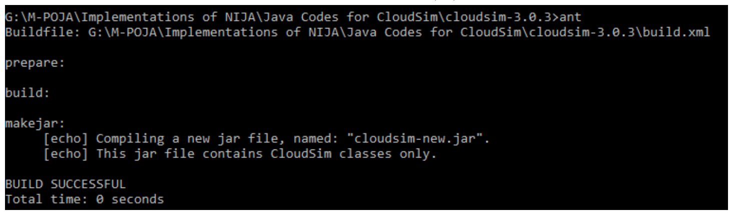
```
VmAllocationPolicy
VmAllocationPolicySimple
VmScheduler
VmSchedulerSpaceShared
```

(a) Simple VM Allocation Policy



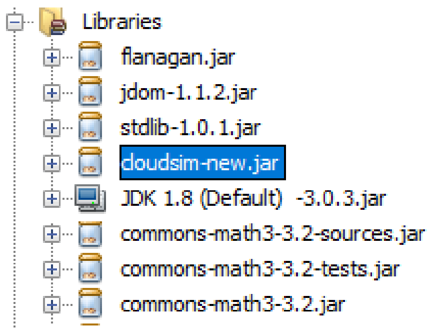
```
import org.cloudbus.cloudsim.Pe;
import org.cloudbus.cloudsim.Storage;
import org.cloudbus.cloudsim.UtilizationModel;
import org.cloudbus.cloudsim.UtilizationModelFull;
import org.cloudbus.cloudsim.Vm;
import org.cloudbus.cloudsim.VmSchedulerTimeShared;
import org.cloudbus.cloudsim.VmAllocationPolicySimple;
import org.cloudbus.cloudsim.hybridVmAllocationPolicy;
import org.cloudbus.cloudsim.core.CloudSim;
```

(b) Using Simple VM alocation policy



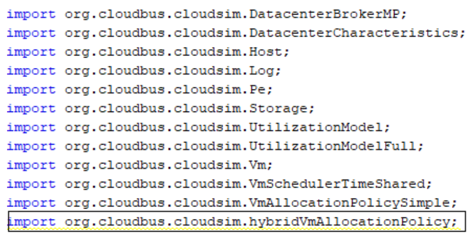
```
G:\M-POJA\Implementations of NIJA\Java Codes for CloudSim\cloudsim-3.0.3>ant
Buildfile: G:\M-POJA\Implementations of NIJA\Java Codes for CloudSim\cloudsim-3.0.3\build.xml

prepare:

build:

makejar:
     [echo] Compiling a new jar file, named: "cloudsim-new.jar".
     [echo] This jar file contains CloudSim classes only.

BUILD SUCCESSFUL
Total time: 0 seconds
```

(c) Building the CloudSim Toolkit containing new policy



```
Libraries
    flanagan.jar
    jdom-1.1.2.jar
    stdlib-1.0.1.jar
    cloudsim-new.jar
    JDK 1.8 (Default)  -3.0.3.jar
    commons-math3-3.2-sources.jar
    commons-math3-3.2-tests.jar
    commons-math3-3.2.jar
```

(d) Include CloudSim-new.jar into library



```
import org.cloudbus.cloudsim.DatacenterBrokerMP;
import org.cloudbus.cloudsim.DatacenterCharacteristics;
import org.cloudbus.cloudsim.Host;
import org.cloudbus.cloudsim.Log;
import org.cloudbus.cloudsim.Pe;
import org.cloudbus.cloudsim.Storage;
import org.cloudbus.cloudsim.UtilizationModel;
import org.cloudbus.cloudsim.UtilizationModelFull;
import org.cloudbus.cloudsim.Vm;
import org.cloudbus.cloudsim.VmSchedulerTimeShared;
import org.cloudbus.cloudsim.VmAllocationPolicySimple;
import org.cloudbus.cloudsim.hybridVmAllocationPolicy;
```

(e) Using Hybrid VM Allocation policy

Fig. 2: Steps in Host Load Balancing

– Run example 4 using the hybridVmAllocationPolicy as the VM allocation policy as shown in figure 2.e

Results obtained using both the approaches are as shown in Figure 3. Clearly hybrid Vm Allocation policy performs slightly better as compared to first fit. Although they both perform equally to a certain point. Naive First-Fit algorithms fails to allocate tasks fairly and hence performs poorly in comparison to the Enhanced First-Fit algorithm when the number of tasks are large.

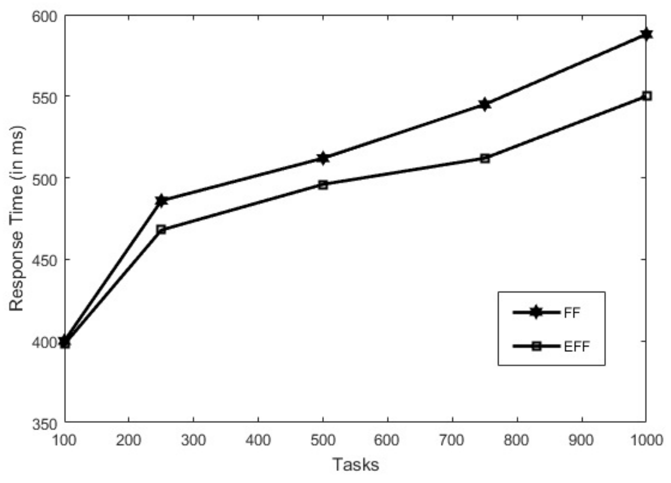


Fig. 3: Comparison in Response Time of First Fit and Enhanced First Fit

```
Input:
A Virtual Machine
Output:
Best Host to choose from the set of Hosts
Begin Algorithm
if $B_{ti} > \sum_{i=1}^{|hi|} B_i$ and $B_{ti} > \sum_{i=1}^{|hi|} B_i$ and $B_{ti} > \sum_{i=1}^{|hi|} B_i$ then
    Queue Task $t$ until resources are free.
else
    Find the First Host which suffices the Compute requirement.
    if the Host was not allocated the last VM. then
        Allocate the VM to this Host
    else
        Find the Next Host which suffices the Compute requirements and allocate the VM
    end
end
End Algorithm
```

Algorithm 1: Enhanced First-Fit

4 Conclusion and Scope for Future

Although cloud is gaining immense popularity, the lack of cloud simulators and proper documentation of existing simulators has led blocked innovation quality. Researchers are unable to test their algorithms on real infrastructure due to large scale and unavailabilty of unified interfaces to collect results. In this work we give the complete steps of running and modifying CloudSim. This will not only enable the researchers to test and analyse their algorithms but also pave way for innovation in simulation tools for cloud. As a scope for future SDN based cloud simulation tools can be discussed to provide insight and innovation into SDN-based cloud networking.

Acknowledgement

This work is funded by MediaLabAsia (Visvesvaraya Ph.D. Scheme for Electronics and IT, Project Code-CS-VSI) under MeitY, Government of India.

References

1. http://ant.apache.org/.
2. W. H. Bell, D. G. Cameron, A. P. Millar, L. Capozza, K. Stockinger, and F. Zini. Optorsim: A grid simulator for studying dynamic data replication strategies. *The International Journal of High Performance Computing Applications*, 17(4):403–416, 2003.
3. R. Buyya and M. Murshed. Gridsim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and Computation: Practice and Experience*, 14(13-15):1175–1220, 2002.
4. R. Buyya and M. Murshed. Gridsim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and Computation: Practice and Experience*, 14(13-15):1175–1220, 2002.
5. R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, 2011.