Simulating the Cloud using CloudSim with NetBeans

Dr.M.Moorthy
Master of Computer Applications
Muthayammal Engineering College
Rasipuram, India
Director.mca@mec.edu.in

Abstract: Cloud computing is a paradigm of large scale distributed computing which is a repackaging of various existing concepts/technologies such as utility computing, Grid computing, Autonomic computing, Virtualization and Internet technologies. To assess the performance of cloud environment, cloud simulators play an important role as it is a challenging task to perform experiments on the real cloud (which would incur huge cost in terms of currency if experiments are repeated). This paper describes the CloudSim architecture, design and implementation of CloudSim, Simulation data flow, CloudSim code, and experimental results.

***Keywords*— Cloud Computing, CloudSim simulator, Data Center, VM Scheduler, and Host.**

I. INTRODUCTION

Cloud Computing infrastructure is different from Grid Computing. It is the massive deployment of Virtualization technologies and tools. Hence, as compared to Grid, Cloud has an extra layer as Virtualization that acts as an execution and hosting environment for cloud-based application services. To secure data in cloud is really very tough job. To understand the cloud computing we need to first understand how these resources are placed in the cloud. Cloud Computing has basically two parts, the First part is of Client Side and the second part is of Server Side. The Client Side requests to the Servers and the Server responds to the Clients. The request from the client firstly goes to the Master Processor of the Server Side. The Master Processor have many Slave Processors, the master processor sends that request to any one of the Slave Processor which is free at that time. All Processors are busy in their assigned job and none of the Processor gets Idle. Simulation opens the possibility to evaluate the hypothesis prior to actual software development in an environment where one can reproduce tests. Why we need simulation because it provides repeatable and controllable environment to test the services. It tunes the system bottlenecks before deploying on real clouds. For simulation we need a special toolkit named CloudSim. It is basically a Library for Simulation of Cloud Computing Scenarios. It has some features such as it support for modeling and simulation of large scale Cloud Computing infrastructure, including data centers on a single physical computing node. It provides basic classes for describing data centers, virtual machines, applications, users, computational resources, and policies.

CloudSim supports VM Scheduling at two levels: First, at the host level where it is possible to specify how much of the overall processing power of each core in a host will be assigned at each

VM. And the second, at the VM level, where the VMs assign specific amount of the available processing power to the individual task units that are hosted within its execution engine.
The rest of this paper is organized as follows: Related work is discussed in section II, CloudSim Architecture is discussed in section III, Design and Implementation of CloudSim is discussed in section IV, Simulation data flow is discussed in section V. Experimental setup and Result is discussed in section VI. And section VII gives conclusion.

II. RELATED WORK

David S. Linthicum [1] described about the basic information about the Cloud Computing and its various services and models like SaaS, IaaS, PaaS. He also described about the deploying models of Cloud Computing and Virtualization services.

Michael Miller [2] described about the various web based application related to Cloud Computing.

R.Bajaj and D.P. Agrawal [3] described about the Scheduling. They said Scheduling is a process of finding the efficient mapping of tasks to the suitable resources so that execution can be completed such as minimization of execution time as specified by customers. They described various types of Scheduling like Static, Dynamic, Centralized, Hierarchical, Distributed, Cooperative, Non-Cooperative Scheduling. They also described Scheduling problem in Cloud and the types of users like CCU (Cloud Computing Customers) and CCSP (Cloud Computing Service Providers).

R.N.Calheiros, Rajiv Ranjan, Anton Beloglazov, C.A.F. De Rose, Rajkumar Buyya [4] described about the Simulation techniques and the CloudSim. They described the various features of CloudSim like it supports for modelling and simulation for large scale of cloud computing infrastructure including data centers on a single physical computing node.

J.Li, M. Qiu, X. Qin [5] described about the optimization criterion that is used when making scheduling decision and represents the goals of the scheduling process. The criterion is expressed by the value of objective function which allows us to measure the quality of computed solution and compare it with different solution.

C.H.Hsu and T. L. Chen [6] described about the Quality of Service that is the ability to provide different jobs and users, or to guarantee a certain level of performance to a job. If the QoS mechanism is supported it allows the user to specify desired performance for their jobs. In system with limited resources the QoS support results in additional cost which is related to the complexity of QoS requests and the efficiency of the scheduler when dealing with them.

Gaetano F.Anastasi, Emanuele Carlini, and Patrizio Dazzi[7] proposed a model of application and cloud resources targeting the contrail platforms using CloudSim simulator.
R.Kannika Devi and S.Sujan[8] discussed about general issues in cloud infrastructure and provided the survey of research works carried out using CloudSim toolkit**.**

S.M.Ranbhise and K.K.Joshi[9] described the CloudSim architecture, need for CloudSim, CloudSim code, and experimental results.

Baptiste Louis, Karan Mithra, Saguna and Saguna[10] proposed a scalable module for modeling and simulation of energy aware storage in cloud systems. Results were produced according to the proposed model using CloudSim simulator.

Law Siew Xue,Nazatul Aini Ahd Majid, and Elangovan A.Sundarajan[11] suggested using PCA and Clustering model to find out the most appropriate and suitable host for allocating VM. Results were obtained using CloudSim simulator. The parameters took into consideration were host RAM,MIPS,storage and bandwidth.

Hanan Ali Al-shehri and Khaoula Hamdi[12] described the optimization of the virtual machines placement is considered with the objective of improving the energy efficiency. An Ant Colony Algorithm is proposed with multiple objectives. Three objectives were optimized: minimizing the number of used physical machines, minimizing the generated network traffic and maximizing the resources utilization. Three resources were considered: CPU, memory and bandwidth. The algorithm was tested on random instances and CloudSim toolkit is used to simulate the results.

III. CLOUDSIM ARCHITECUTURE

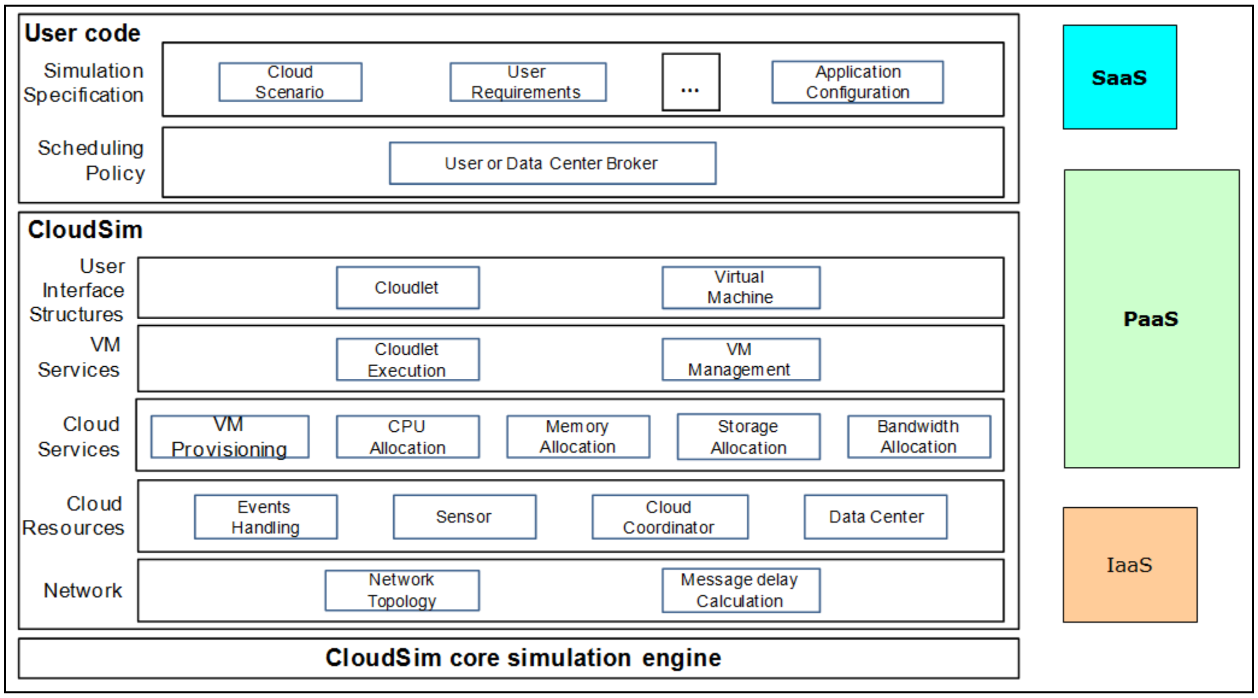

Figure-1: CloudSim layered Architecture

In Figure-1, the CloudSim simulation layer provides support for modeling and simulation of virtualized Cloud-based data center environments including dedicated management interfaces for VMs, memory, storage, and bandwidth. The fundamental issues, such as provisioning of hosts to VMs, managing application execution, and monitoring dynamic system state, are handled by this layer. A Cloud provider, who wants to study the efficiency of different policies in allocating its hosts to VMs (VM provisioning), would need to implement his strategies at this layer. Such implementation can be done by programmatically extending the core VM provisioning functionality. There is a clear distinction at this layer related to provisioning of hosts to VMs. A Cloud host can be concurrently allocated to a set of VMs that execute applications based on SaaS provider's defined QoS levels. This layer also exposes the functionalities that a Cloud application developer can extend to perform complex workload profiling and application performance study. The top-most layer in the CloudSim stack is the User Code that exposes basic entities for hosts (number of machines, their specification, and so on), applications (number of tasks and their requirements), VMs, number of users and their application types, and broker scheduling policies. By extending the basic entities given at this layer, a Cloud application developer can perform the following activities: (i) generate a mix of workload request distributions, application configurations; (ii) model Cloud availability scenarios and perform robust tests based on the custom configurations; and (iii) implement custom application provisioning techniques for clouds and their federation.

IV. DESIGN AND IMPLEMENTATION OF CLOUDSIM

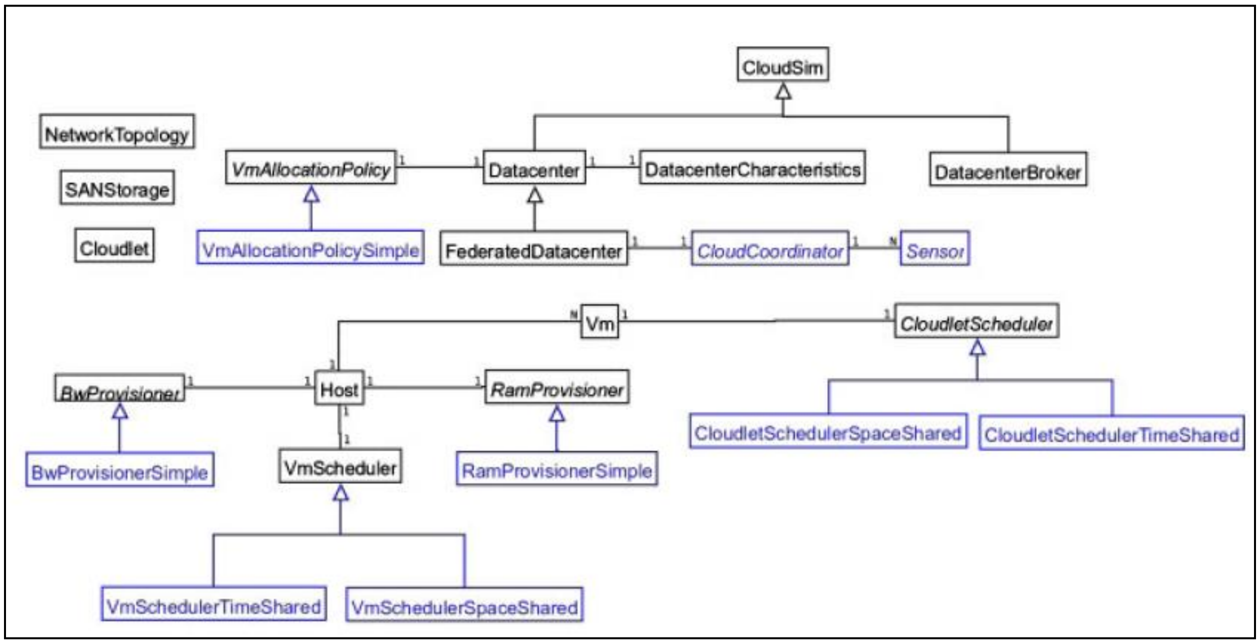

Figure-2: CloudSim class design diagram

In Figure-2, we provide the finer details related to the fundamental classes of CloudSim, which are also the building blocks of the simulator. The overall Class design diagram for CloudSim is shown in Figure x.

BwProvisioner: This is an abstract class that models the policy for provisioning of bandwidth to VMs. The main role of this component is to undertake the allocation of network bandwidths to a set of competing VMs that are deployed across the data center. Cloud system developers and researchers can extend this class with their own policies (priority, QoS) to reflect the needs of their applications. The *BwProvisioningSimple* allows a VM to reserve as much bandwidth as required;however, this is constrained by the total available bandwidth of the host.

CloudCoordinator: This abstract class extends a Cloud-based data center to the federation. It is responsible for periodically monitoring the internal state of data center resources and based on that it undertakes dynamic load-shredding decisions. Concrete implementation of this component includes the specific sensors and the policy that should be followed during load-shredding.

Monitoring of data center resources is performed by the updateDatacenter*()* method by sending queries Sensors. Service/Resource Discovery is realized in the *setDatacenter()*abstract method that can be extended for implementing custom protocols and mechanisms (multicast, broadcast, peer-to-peer). Further, this component can also be extended for simulating Cloud-based services such as the Amazon EC2 Load-Balancer. Developers aiming to deploy their application services across multiple clouds can extend this class for implementing their custom inter-cloud provisioning policies.

Cloudlet: This class models the Cloud-based application services (such as content delivery, social networking, and business workflow). CloudSim orchestrates the complexity of an application in terms of its computational requirements. Every application service has a pre-assigned instruction length and data transfer (both pre and post fetches) overhead that it needs to undertake during its life cycle. This class can also be extended to support modeling of other performance and composition metrics for applications such as transactions in database-oriented applications.

CloudletScheduler: This abstract class is extended by the implementation of different policies that determine the share of processing power among Cloudlets in a VM. As described previously,two types of provisioning policies are offered: space-shared (*CloudetSchedulerSpaceShared*) and time-shared (*CloudletSchedulerTimeShared*).

Datacenter: This class models the core infrastructure-level services (hardware) that are offered by Cloud providers (Amazon, Azure, and App Engine). It encapsulates a set of compute hosts that can either be homogeneous or heterogeneous with respect to their hardware configurations (memory, cores, capacity, and storage). Furthermore, every Datacenter component instantiates a generalized application provisioning component that implements a set of policies for allocating bandwidth, memory, and storage devices to hosts and VMs.

DatacenterBroker or Cloud Broker: This class models a broker, which is responsible for mediating negotiations between SaaS and Cloud providers; and such negotiations are driven by QoS requirements. The broker acts on behalf of SaaS providers. It discovers suitable Cloud service providers by querying the CIS and undertakes online negotiations for allocation of resources/services that can meet the application's QoS needs. Researchers and system developers must extend this class for evaluating and testing custom brokering policies. The difference between the broker and the CloudCoordinator is that the former represents the customer (i.e. decisions of these components are made in order to increase user-related performance metrics), whereas the latter acts on behalf of the data center, i.e. it tries to maximize the overall performance of the data center, without considering the needs of specific customers.

DatacenterCharacteristics: This class contains configuration information of data center resources.

Host: This class models a physical resource such as a compute or storage server. It encapsulates important information such as the amount of memory and storage, a list and type of processing cores (to represent a multi-core machine), an allocation of policy for sharing the processing power among VMs, and policies for provisioning memory and bandwidth to the VMs.

NetworkTopology: This class contains the information for inducing network behavior (latencies) in the simulation. It stores the topology information, which is generated using the BRITE topology generator.

RamProvisioner: This is an abstract class that represents the provisioning policy for allocating primary memory (RAM) to the VMs. The execution and deployment of VM on a host is feasible only if the RamProvisioner component approves that the host has the required amount of free memory. The *RamProvisionerSimple* does not enforce any limitation on the amount of memory that a VM may request. However, if the request is beyond the available memory capacity, then it is simply rejected.

SanStorage: This class models a storage area network that is commonly ambient in Cloud-based data centers for storing large chunks of data (such as Amazon S3, Azure blob storage). SanStorage implements a simple interface that can be used to simulate storage and retrieval of any amount of data, subject to the availability of network bandwidth. Accessing files in a SAN at run-time incurs additional delays for task unit execution; this is due to the additional latencies that are incurred in transferring the data files through the data center internal network.

Sensor: This interface must be implemented to instantiate a sensor component that can be used by a CloudCoordinator for monitoring specific performance parameters (energy-consumption, resource utilization). Recall that, CloudCoordinator utilizes the dynamic performance information for undertaking load-balancing decisions. The methods defined by this interface are: (i) set the minimum and maximum thresholds for performance parameter and (ii) periodically update the measurement. This class can be used to model the real-world services offered by leading Cloud providers such as Amazon's CloudWatch and Microsoft Azure's FabricController. One data center may instantiate one or more Sensors, each one responsible for monitoring a specific data center performance parameter.

Vm: This class models a VM, which is managed and hosted by a Cloud host component. Every VM component has access to a component that stores the following characteristics related to a VM: accessible memory, processor, storage size, and the VM's internal provisioning policy that is extended from an abstract component called the *CloudletScheduler*.

VmmAllocationPolicy: This abstract class represents a provisioning policy that a VM Monitor utilizes for allocating VMs to hosts. The chief functionality of the VmmAllocationPolicy is to select the available host in a data center that meets the memory, storage, and availability requirement for a VM deployment.

VmScheduler: This is an abstract class implemented by a Host component that models the policies (space-shared, time-shared) required for allocating processor cores to VMs. The functionalities of this class can easily be overridden to accommodate application-specific processor sharing policies.

V. SIMULATION DATA FLOW

Figure-3 depicts the flow of communication among core CloudSim entities. At the beginning of a simulation, each Datacenter entity registers with the CIS Registry. CIS then provides information registry-type functionalities, such as match-making services for mapping user/brokers, requests to suitable Cloud providers. Next, the DataCenter brokers acting on behalf of users consult the CIS service to obtain the list of cloud providers who can offer infrastructure services that match application's QoS, hardware, and software requirements. In the event of a match, the DataCenter broker deploys the application with the CIS suggested cloud. The communication flow described so far relates to the basic flow in a simulated experiment. Some variations in this flow are possible depending on policies. For example, messages from Brokers to Datacenters may require a confirmation from other parts of the Datacenter, about the execution of an action, or about the maximum number of VMs that a user can create.

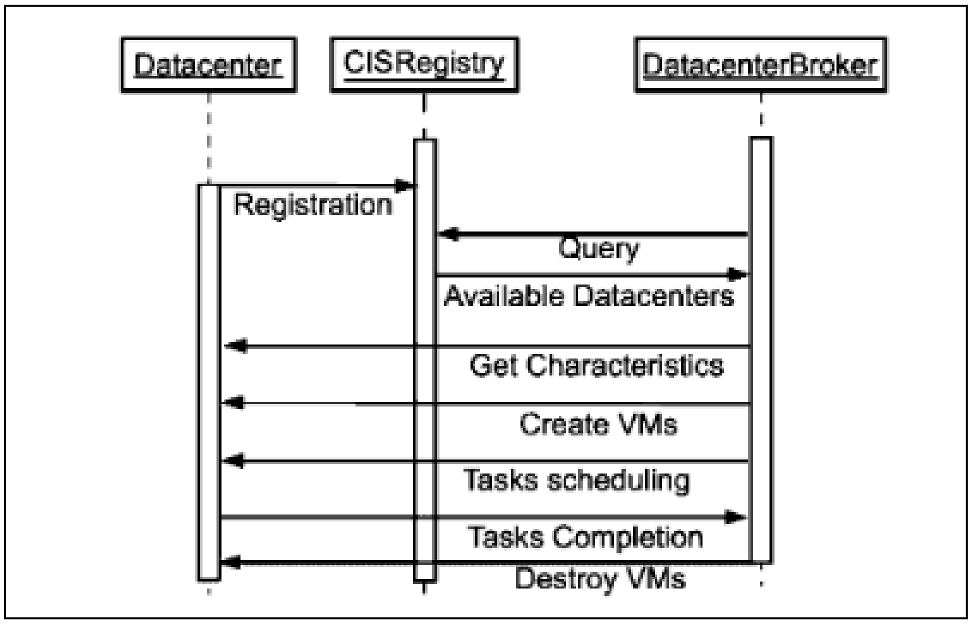

Figure-3: Simulation data flow

VI. EXPERIMENTAL SETUP AND RESULT

To evaluate the performance of Cloud, results were simulated in Window 7 basic (32-bit), Intel ® Core ™ Duo E6550 Processor, 2.33 GHz of speed with memory of 1 GB and the language used is Java. First code of Simulation is tested on one Data Center with one Host and run on one Cloudlet. [Softwares used: JDK version: 1.7, NetBeans 7.0, and CloudSim 3.0.3]

Initialize the CloudSim Package :
int num_user = 1; //number of cloud users
Calendar calendar = Calender.getInstance();
boolean trace_flag = false; //mean trace events

Initialize the CloudSim Library :
CloudSim.init(num_user, calendar, trace_flag);
Create Datacenters : These are the resource providers in CloudSim. We need at least one of them to run a CloudSim simulation.

Datacenter datacenter0 =createDatacenter("Datacenter_0");

Create Broker :
DatacenterBroker broker = createBroker();
int brokerId = broker.getId();
Create one Virtual machine:
Vmlist = new ArrayList<Vm>();
VM description :
int vmid = 0;
int mips= 1000;
long size = 10000; //image size (MB)
int ram = 512; //vm memory (MB)
iong bw = 1000;
int pesNumber = 1; //number of cpu
String vmm = "Xen"; // VMM name

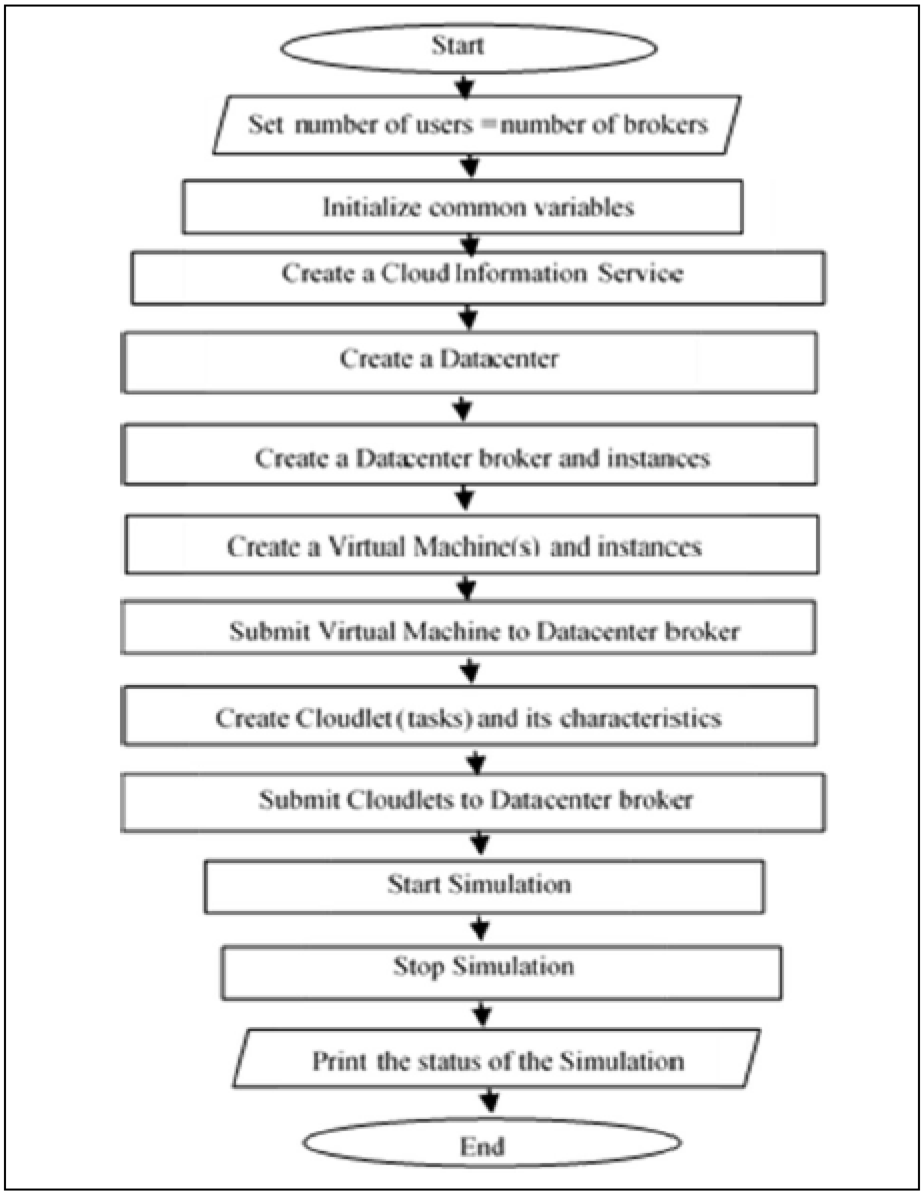
Start
Set number of users =number of brokers
Initialize common variables
Create a Cloud Information Service
Create a Datacenter
Create a Datacenter broker and instances
Create a Virtual Machine(s) and instances
Submit Virtual Machine to Datacenter broker
Create Cloudlet (tasks) and its characteristics
Submit Cloudlets to Datacenter broker
Start Simulation
Stop Simulation
Print the status of the Simulation
End

Figure-4: The main steps in using CloudSim for the simulation.

Figure-4 shows the main steps involved in cloud simulation
Create VM :

```
Vm vm = new Vm(vmid, brokerId, mips, pesNumber,ram, bw, size, vmm,
            new CloudletSchedulerTimeShared());
```

Add the VM to the vmList :

```
vmList.add(vm1);
```

Submit vm list to the broker :

```
Broker.submitVmList(vmList);
```

Creation of Cloudlets :

```
cloudLetList = new ArrayList<Cloudlet>();
```

Cloudlet Properties :

```
int id = 0;
pesNumber = 1;
long length = 400000;
long fileSize = 300;
long outputSize = 300;
UtilizationModel utilizationModel = newUtilizationModelFull();
Cloudlet cloudlet = new Cloudlet(id, length,pesNumber, fileSize, outputSize, utilizationModel,
                         utilizationModel1, utilizationModel1);
Cloudlet.setUserId(brokerId);
Cloudlet.setVmId(vmid);
```

Add the Cloudlets to the list :

```
cloudLetList.add(cloudlet1);
```

Submit cloudlet list to the broker :

```
Broker.submitCloudletList(cloudLetList);
```

Start Simulation :

```
CloudSim.startSimulation();
CloudSim.stopSimulation();
```

Final step : Print result when simulation is over

```
List<Cloudlet> newList = broker.getCloudletReceivedList();
printCloudletList(newList);
// Print the debt of each user to each datacenter
Datacenter0.printDebts();
```

The output of this simulation is :

```
========== OUTPUT ==========
Cloudlet ID    STATUS      Data center ID  VM ID      Time Start     Time Finish   Time
    0          SUCCESS           2            0           400          0            400

*****PowerDatacenter: Datacenter_0*****
User id                Debt
3              35.6
```

```
**********************************
```

After increasing the datacenter with two hosts and run two cloudlets on it. The cloudlets run in VMs with different MIPS requirements. The cloudlets will take different time to complete the execution depending on the requested VM performance. It prints the following output:

```
========== OUTPUT ==========
Cloudlet ID    STATUS       Data center ID   VM ID      Time Start     Time Finish    Time
     1         SUCCESS           2             1           80              0          80
     0         SUCCESS           2             0          160              0         160

*****PowerDatacenter: Datacenter_0*****
User id        Debt
3              224.8
**********************************
```

VII. CONCLUSION

In this paper, I have proposed the code for simulation and shown the various outputs in which the information about the Cloudlets, Status, Datacenter ID, Virtual Machine ID, Start Time, and Finish Time is given. And by changing the number of Host, Datacenters and Cloudlets, I have observed the difference. Cloudsim is the perfect solution for modeling the cloud against scaling in and scaling out of the infrastructure requirement. Cloud reports are the perfect solution for costing of the infrastructure, resource utilization, and power consumption of the customized environment.

REFERENCES

[1] David S. Linthicum “Cloud Computing and SOA Convergence in your Enterprise”, 2011.
[2] Michael Miller, “Cloud Computing Web Based Application” 2012.
[3] R. Bajaj and D.P. Agrawal, “Improving Scheduling of Tasks in a Heterogeneous Environment, IEEE Transaction on parallel and Distributed Systems”, p. 107-118, 2004.
[4] R. N. Calheiros, Rajiv Ranjan, Anton Beloglazov, C.A.F. De Rose, Rajkumar Buyya, “CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms”, Software Practice and Experience, Wiley publishers, 2010.
[5] J.Li, M.Qiu, X.Qin, “Feedback Dynamic Algorithms for Preemptable Job Scheduling in Cloud System”, IEEE, 2010.
[6] C.H.Hsu, T.L.Chen, “Adaptive Scheduling based on QoS in Heterogeneous Environment”, IEEE, 2010.
[7] Gaetano F.Anastasi, Emanuele Carlini, and Patrizio Dazzi,”Smart Cloud Federation Simulations with CloudSim”,ACM-ARMACloud’13”,June 17,2013.
[8] R.Kannika Devi and S.Sujan,”A Survey on application of CloudSim toolkit in Cloud Computing”,IJSRSET Vol. 3,Issue 6, June 2014.
[9] S.M.Ranbhise and K.K.Joshi,”Simulation and Analysis of Cloud Environment”,IJARCST Vol. 2, Issue 4,Oct-Dec. 2014.
[10] Baptiste Louis, Karan Mithra, Saguna and Saguna,”CloudSimDisk:Energy-Aware storage simulation in CloudSim”,IEEE/ACM DOI 10.1109/UCC 2015.

[11] Law Siew Xue,Nazatul Aini Ahd Majid, and Elangovan A.Sundarajan,"Quality of Service evaluation of IaaS modeler allocation",ACM ISBN 978-1-4503-4868-2/2017, Malaysia.
[12] Hanan Ali Al-shehri and Khaoula Hamdi," Energy-Aware Multi-Objective Placement of Virtual Machines in Cloud Data Centers", ACM-*ICBBS '18,* June 23–25, 2018, Shenzhen, China.

M. Moorthy has received his Ph.D. from Anna University, Chennai in Computer Science and MCA from Manonmaniam Sundaranar University, Tirunelveli. He is Professor and Head of Department in MCA, Muthayammal Engineering College, Namakkal, Tamil Nadu, India. His research interest includes Computer Networking, Network Security, Wireless Network, Data Mining, and Artificial Intelligence. He is a life member of ISTE, CSI, and IAENG.

Bibliography of the author