1. Create a Collaborative learning environment for a particular learning topic using Google Apps. Google Drive, Google Docs and Google Slides must be used for hosting e-books, important articles and presentations respectively. The instructor must use the Google Sheets to convey the timetable for different events and for analyzing the scores for individual assignment submission.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | package org.cloudbus.cloudsim.examples; | |  | | /\* | | \* Title:        CloudSim Toolkit | | \* Description:  CloudSim (Cloud Simulation) Toolkit for Modeling and Simulation | | \*               of Clouds | | \* Licence:      GPL - http://www.gnu.org/copyleft/gpl.html | | \* | | \* Copyright (c) 2009, The University of Melbourne, Australia | | \*/ | |  | | import java.text.DecimalFormat; | | import java.util.ArrayList; | | import java.util.Calendar; | | import java.util.LinkedList; | | import java.util.List; | |  | | import org.cloudbus.cloudsim.Cloudlet; | | import org.cloudbus.cloudsim.CloudletSchedulerTimeShared; | | import org.cloudbus.cloudsim.Datacenter; | | import org.cloudbus.cloudsim.DatacenterBroker; | | 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.VmAllocationPolicySimple; | | import org.cloudbus.cloudsim.VmSchedulerTimeShared; | | import org.cloudbus.cloudsim.core.CloudSim; | | import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple; | | import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple; | | import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple; | |  | | /\*\* | | \* A simple example showing how to create a data center with one host and run one cloudlet on it. | | \*/ | | public class CloudSimExample1 { | | /\*\* The cloudlet list. \*/ | | private static List<Cloudlet> cloudletList; | | /\*\* The vmlist. \*/ | | private static List<Vm> vmlist; | |  | | /\*\* | | \* Creates main() to run this example. | | \* | | \* @param args the args | | \*/ | | @SuppressWarnings("unused") | | public static void main(String[] args) { | | Log.printLine("Starting CloudSimExample1..."); | |  | | try { | | // First step: Initialize the CloudSim package. It should be called before creating any entities. | | int num\_user = 1; // number of cloud users | | Calendar calendar = Calendar.getInstance(); // Calendar whose fields have been initialized with the current date and time. | | boolean trace\_flag = false; // trace events | |  | | /\* Comment Start - Dinesh Bhagwat | | \* Initialize the CloudSim library. | | \* init() invokes initCommonVariable() which in turn calls initialize() (all these 3 methods are defined in CloudSim.java). | | \* initialize() creates two collections - an ArrayList of SimEntity Objects (named entities which denote the simulation entities) and | | \* a LinkedHashMap (named entitiesByName which denote the LinkedHashMap of the same simulation entities), with name of every SimEntity as the key. | | \* initialize() creates two queues - a Queue of SimEvents (future) and another Queue of SimEvents (deferred). | | \* initialize() creates a HashMap of of Predicates (with integers as keys) - these predicates are used to select a particular event from the deferred queue. | | \* initialize() sets the simulation clock to 0 and running (a boolean flag) to false. | | \* Once initialize() returns (note that we are in method initCommonVariable() now), a CloudSimShutDown (which is derived from SimEntity) instance is created | | \* (with numuser as 1, its name as CloudSimShutDown, id as -1, and state as RUNNABLE). Then this new entity is added to the simulation | | \* While being added to the simulation, its id changes to 0 (from the earlier -1). The two collections - entities and entitiesByName are updated with this SimEntity. | | \* the shutdownId (whose default value was -1) is 0 | | \* Once initCommonVariable() returns (note that we are in method init() now), a CloudInformationService (which is also derived from SimEntity) instance is created | | \* (with its name as CloudInformatinService, id as -1, and state as RUNNABLE). Then this new entity is also added to the simulation. | | \* While being added to the simulation, the id of the SimEntitiy is changed to 1 (which is the next id) from its earlier value of -1. | | \* The two collections - entities and entitiesByName are updated with this SimEntity. | | \* the cisId(whose default value is -1) is 1 | | \* Comment End - Dinesh Bhagwat | | \*/ | | CloudSim.init(num\_user, calendar, trace\_flag); | |  | | // Second step: Create Datacenters | | // Datacenters are the resource providers in CloudSim. We need at | | // list one of them to run a CloudSim simulation | | Datacenter datacenter0 = createDatacenter("Datacenter\_0"); | |  | | // Third step: Create Broker | | DatacenterBroker broker = createBroker(); | | int brokerId = broker.getId(); | |  | | // Fourth step: 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) | | long bw = 1000; | | int pesNumber = 1; // number of cpus | | String vmm = "Xen"; // VMM name | |  | | // create VM | | Vm vm = new Vm(vmid, brokerId, mips, pesNumber, ram, bw, size, vmm, new CloudletSchedulerTimeShared()); | |  | | // add the VM to the vmList | | vmlist.add(vm); | |  | | // submit vm list to the broker | | broker.submitVmList(vmlist); | |  | | // Fifth step: Create one Cloudlet | | cloudletList = new ArrayList<Cloudlet>(); | |  | | // Cloudlet properties | | int id = 0; | | long length = 400000; | | long fileSize = 300; | | long outputSize = 300; | | UtilizationModel utilizationModel = new UtilizationModelFull(); | |  | | Cloudlet cloudlet = new Cloudlet(id, length, pesNumber, fileSize, outputSize, utilizationModel, utilizationModel, utilizationModel); | | cloudlet.setUserId(brokerId); | | cloudlet.setVmId(vmid); | |  | | // add the cloudlet to the list | | cloudletList.add(cloudlet); | |  | | // submit cloudlet list to the broker | | broker.submitCloudletList(cloudletList); | |  | | // Sixth step: Starts the simulation | | CloudSim.startSimulation(); | |  | | CloudSim.stopSimulation(); | |  | | //Final step: Print results when simulation is over | | List<Cloudlet> newList = broker.getCloudletReceivedList(); | | printCloudletList(newList); | |  | | Log.printLine("CloudSimExample1 finished!"); | | } catch (Exception e) { | | e.printStackTrace(); | | Log.printLine("Unwanted errors happen"); | | } | | } | |  | | /\*\* | | \* Creates the datacenter. | | \* | | \* @param name the name | | \* | | \* @return the datacenter | | \*/ | | private static Datacenter createDatacenter(String name) { | |  | | // Here are the steps needed to create a PowerDatacenter: | | // 1. We need to create a list to store | | // our machine | | List<Host> hostList = new ArrayList<Host>(); | |  | | // 2. A Machine contains one or more PEs or CPUs/Cores. | | // In this example, it will have only one core. | | List<Pe> peList = new ArrayList<Pe>(); | |  | | int mips = 1000; | |  | | // 3. Create PEs and add these into a list. | | peList.add(new Pe(0, new PeProvisionerSimple(mips))); // need to store Pe id and MIPS Rating | |  | | // 4. Create Host with its id and list of PEs and add them to the list | | // of machines | | int hostId = 0; | | int ram = 2048; // host memory (MB) | | long storage = 1000000; // host storage | | int bw = 10000; | |  | | hostList.add( | | new Host( | | hostId, | | new RamProvisionerSimple(ram), | | new BwProvisionerSimple(bw), | | storage, | | peList, | | new VmSchedulerTimeShared(peList) | | ) | | ); // This is our machine | |  | | // 5. Create a DatacenterCharacteristics object that stores the | | // properties of a data center: architecture, OS, list of | | // Machines, allocation policy: time- or space-shared, time zone | | // and its price (G$/Pe time unit). | | String arch = "x86"; // system architecture | | String os = "Linux"; // operating system | | String vmm = "Xen"; | | double time\_zone = 10.0; // time zone this resource located | | double cost = 3.0; // the cost of using processing in this resource | | double costPerMem = 0.05; // the cost of using memory in this resource | | double costPerStorage = 0.001; // the cost of using storage in this | | // resource | | double costPerBw = 0.0; // the cost of using bw in this resource | | LinkedList<Storage> storageList = new LinkedList<Storage>(); // we are not adding SAN | | // devices by now | |  | | DatacenterCharacteristics characteristics = new DatacenterCharacteristics( | | arch, os, vmm, hostList, time\_zone, cost, costPerMem, | | costPerStorage, costPerBw); | |  | | // 6. Finally, we need to create a PowerDatacenter object. | | Datacenter datacenter = null; | | try { | | datacenter = new Datacenter(name, characteristics, new VmAllocationPolicySimple(hostList), storageList, 0); | | } catch (Exception e) { | | e.printStackTrace(); | | } | |  | | return datacenter; | | } | |  | | // We strongly encourage users to develop their own broker policies, to | | // submit vms and cloudlets according | | // to the specific rules of the simulated scenario | | /\*\* | | \* Creates the broker. | | \* | | \* @return the datacenter broker | | \*/ | | private static DatacenterBroker createBroker() { | | DatacenterBroker broker = null; | | try { | | broker = new DatacenterBroker("Broker"); | | } catch (Exception e) { | | e.printStackTrace(); | | return null; | | } | | return broker; | | } | |  | | /\*\* | | \* Prints the Cloudlet objects. | | \* | | \* @param list list of Cloudlets | | \*/ | | private static void printCloudletList(List<Cloudlet> list) { | | int size = list.size(); | | Cloudlet cloudlet; | |  | | String indent = "    "; | | Log.printLine(); | | Log.printLine("========== OUTPUT =========="); | | Log.printLine("Cloudlet ID" + indent + "STATUS" + indent | | + "Data center ID" + indent + "VM ID" + indent + "Time" + indent | | + "Start Time" + indent + "Finish Time"); | |  | | DecimalFormat dft = new DecimalFormat("###.##"); | | for (int i = 0; i < size; i++) { | | cloudlet = list.get(i); | | Log.print(indent + cloudlet.getCloudletId() + indent + indent); | |  | | if (cloudlet.getCloudletStatus() == Cloudlet.SUCCESS) { | | Log.print("SUCCESS"); | |  | | Log.printLine(indent + indent + cloudlet.getResourceId() | | + indent + indent + indent + cloudlet.getVmId() | | + indent + indent | | + dft.format(cloudlet.getActualCPUTime()) + indent | | + indent + dft.format(cloudlet.getExecStartTime()) | | + indent + indent | | + dft.format(cloudlet.getFinishTime())); | | } | | } | | } | | } |  |  | | --- | |  | |

2. Modeling and simulation Cloud computing environments, including Data Centers, Hosts and Cloudlets and perform VM provisioning using CloudSim: Design a host with two CPU cores, which receives request for hosting two VMs, such that each one requires two cores and plans to host four tasks units. More specifically, tasks t1, t2, t3 and t4 to be hosted in VM1, while t5, t6, t7, and t8 to be hosted in VM2. Implement space shared allocation policy and time-shared allocation policy. Compare the results.

/\*

\* Title: CloudSim Toolkit

\* Description: CloudSim (Cloud Simulation) Toolkit for Modeling and Simulation

\* of Clouds

\* Licence: GPL - http://www.gnu.org/copyleft/gpl.html

\*

\* Copyright (c) 2009, The University of Melbourne, Australia

\*/

package org.cloudbus.cloudsim.examples;

import java.text.DecimalFormat;

import java.util.ArrayList;

import java.util.Calendar;

import java.util.LinkedList;

import java.util.List;

import org.cloudbus.cloudsim.Cloudlet;

import org.cloudbus.cloudsim.CloudletSchedulerTimeShared;

import org.cloudbus.cloudsim.Datacenter;

import org.cloudbus.cloudsim.DatacenterBroker;

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.VmAllocationPolicySimple;

import org.cloudbus.cloudsim.VmSchedulerTimeShared;

import org.cloudbus.cloudsim.core.CloudSim;

import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;

import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;

import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;

/\*\*

\* A simple example showing how to create

\* a datacenter with one host and run two

\* cloudlets on it. The cloudlets run in

\* VMs with the same MIPS requirements.

\* The cloudlets will take the same time to

\* complete the execution.

\*/

public class CloudSimExample2 {

/\*\* The cloudlet list. \*/

private static List<Cloudlet> cloudletList;

/\*\* The vmlist. \*/

private static List<Vm> vmlist;

/\*\*

\* Creates main() to run this example

\*/

public static void main(String[] args) {

Log.printLine("Starting CloudSimExample2...");

try {

// First step: Initialize the CloudSim package. It should be called

// before creating any entities.

int num\_user = 1; // number of cloud users

Calendar calendar = Calendar.getInstance();

boolean trace\_flag = false; // mean trace events

// Initialize the CloudSim library

CloudSim.init(num\_user, calendar, trace\_flag);

// Second step: Create Datacenters

//Datacenters are the resource providers in CloudSim. We need at list one of them to run a CloudSim simulation

@SuppressWarnings("unused")

Datacenter datacenter0 = createDatacenter("Datacenter\_0");

//Third step: Create Broker

DatacenterBroker broker = createBroker();

int brokerId = broker.getId();

//Fourth step: Create one virtual machine

vmlist = new ArrayList<Vm>();

//VM description

int vmid = 0;

int mips = 250;

long size = 10000; //image size (MB)

int ram = 512; //vm memory (MB)

long bw = 1000;

int pesNumber = 1; //number of cpus

String vmm = "Xen"; //VMM name

//create two VMs

Vm vm1 = new Vm(vmid, brokerId, mips, pesNumber, ram, bw, size, vmm, new CloudletSchedulerTimeShared());

vmid++;

Vm vm2 = new Vm(vmid, brokerId, mips, pesNumber, ram, bw, size, vmm, new CloudletSchedulerTimeShared());

//add the VMs to the vmList

vmlist.add(vm1);

vmlist.add(vm2);

//submit vm list to the broker

broker.submitVmList(vmlist);

//Fifth step: Create two Cloudlets

cloudletList = new ArrayList<Cloudlet>();

//Cloudlet properties

int id = 0;

pesNumber=1;

long length = 250000;

long fileSize = 300;

long outputSize = 300;

UtilizationModel utilizationModel = new UtilizationModelFull();

Cloudlet cloudlet1 = new Cloudlet(id, length, pesNumber, fileSize, outputSize, utilizationModel, utilizationModel, utilizationModel);

cloudlet1.setUserId(brokerId);

id++;

Cloudlet cloudlet2 = new Cloudlet(id, length, pesNumber, fileSize, outputSize, utilizationModel, utilizationModel, utilizationModel);

cloudlet2.setUserId(brokerId);

//add the cloudlets to the list

cloudletList.add(cloudlet1);

cloudletList.add(cloudlet2);

//submit cloudlet list to the broker

broker.submitCloudletList(cloudletList);

//bind the cloudlets to the vms. This way, the broker

// will submit the bound cloudlets only to the specific VM

broker.bindCloudletToVm(cloudlet1.getCloudletId(),vm1.getId());

broker.bindCloudletToVm(cloudlet2.getCloudletId(),vm2.getId());

// Sixth step: Starts the simulation

CloudSim.startSimulation();

// Final step: Print results when simulation is over

List<Cloudlet> newList = broker.getCloudletReceivedList();

CloudSim.stopSimulation();

printCloudletList(newList);

Log.printLine("CloudSimExample2 finished!");

}

catch (Exception e) {

e.printStackTrace();

Log.printLine("The simulation has been terminated due to an unexpected error");

}

. }

private static Datacenter createDatacenter(String name){

// Here are the steps needed to create a PowerDatacenter:

// 1. We need to create a list to store

// our machine

List<Host> hostList = new ArrayList<Host>();

// 2. A Machine contains one or more PEs or CPUs/Cores.

// In this example, it will have only one core.

List<Pe> peList = new ArrayList<Pe>();

int mips = 1000;

// 3. Create PEs and add these into a list.

peList.add(new Pe(0, new PeProvisionerSimple(mips))); // need to store Pe id and MIPS Rating

//4. Create Host with its id and list of PEs and add them to the list of machines

int hostId=0;

int ram = 2048; //host memory (MB)

long storage = 1000000; //host storage

int bw = 10000;

hostList.add(

new Host(

hostId,

new RamProvisionerSimple(ram),

new BwProvisionerSimple(bw),

storage,

peList,

new VmSchedulerTimeShared(peList)

)

); // This is our machine

// 5. Create a DatacenterCharacteristics object that stores the

// properties of a data center: architecture, OS, list of

// Machines, allocation policy: time- or space-shared, time zone

// and its price (G$/Pe time unit).

String arch = "x86"; // system architecture

String os = "Linux"; // operating system

String vmm = "Xen";

double time\_zone = 10.0; // time zone this resource located

double cost = 3.0; // the cost of using processing in this resource

double costPerMem = 0.05; // the cost of using memory in this resource

double costPerStorage = 0.001; // the cost of using storage in this resource

double costPerBw = 0.0; // the cost of using bw in this resource

LinkedList<Storage> storageList = new LinkedList<Storage>(); //we are not adding SAN devices by now

DatacenterCharacteristics characteristics = new DatacenterCharacteristics(

arch, os, vmm, hostList, time\_zone, cost, costPerMem, costPerStorage, costPerBw);

// 6. Finally, we need to create a PowerDatacenter object.

Datacenter datacenter = null;

try {

datacenter = new Datacenter(name, characteristics, new VmAllocationPolicySimple(hostList), storageList, 0);

} catch (Exception e) {

e.printStackTrace();

}

return datacenter;

}

//We strongly encourage users to develop their own broker policies, to submit vms and cloudlets according

//to the specific rules of the simulated scenario

private static DatacenterBroker createBroker(){

DatacenterBroker broker = null;

try {

broker = new DatacenterBroker("Broker");

} catch (Exception e) {

e.printStackTrace();

return null;

}

return broker;

}

/\*\*

\* Prints the Cloudlet objects

\* @param list list of Cloudlets

\*/

private static void printCloudletList(List<Cloudlet> list) {

int size = list.size();

Cloudlet cloudlet;

String indent = " ";

Log.printLine();

Log.printLine("========== OUTPUT ==========");

Log.printLine("Cloudlet ID" + indent + "STATUS" + indent +

"Data center ID" + indent + "VM ID" + indent + "Time" + indent + "Start Time" + indent + "Finish Time");

DecimalFormat dft = new DecimalFormat("###.##");

for (int i = 0; i < size; i++) {

cloudlet = list.get(i);

Log.print(indent + cloudlet.getCloudletId() + indent + indent);

if (cloudlet.getCloudletStatus() == Cloudlet.SUCCESS){

Log.print("SUCCESS");

Log.printLine( indent + indent + cloudlet.getResourceId() + indent + indent + indent + cloudlet.getVmId() +

indent + indent + dft.format(cloudlet.getActualCPUTime()) + indent + indent + dft.format(cloudlet.getExecStartTime())+

indent + indent + dft.format(cloudlet.getFinishTime()));

}

}

}

}

========== OUTPUT ==========

Cloudlet ID STATUS Data center ID VM ID Time Start Time Finish Time

0 SUCCESS 2 0 1000 0.1 1000.1

1 SUCCESS 2 1 1000 0.1 1000.1

CloudSimExample2 finished!

3. Model a Cloud computing environment having Data center that had 100 hosts. The hosts are to be modeled to have a CPU core (1000 MIPS), 2 GB of RAM and 1 TB of storage. Consider the workload model for this evaluation included provisioning requests for 400 VMs, with each request demanding 1 CPU core (250 MIPS), 256 MB of RAM and 1 GB of storage. Each VM hosts a *web-hosting application service*, whose CPU utilization distribution was generated according to the uniform distribution. Each instance of a webhosting service required 150,000 MIPS or about 10 minutes to complete execution assuming 100% utilization. Simulate Energy-conscious model for power consumption and power management techniques such as Dynamic Voltage and Frequency Scaling (DVFS). Initially, VMs are to be allocated according to

requested parameters (4 VMs on each host). The Cloud computing architecture that is to be considered for studying energy conscious resource management techniques/policies included a data center, CloudCoordinator, and Sensor component. The CloudCoordinator and Sensor perform their usual roles. Via the attached Sensors (which are connected with every host), CloudCoordinator must periodically monitor the performance status of active VMs such as load conditions, and processing share. This real time information is to be passed to VMM, which can use it for performing appropriate resizing of VMs and application of DVFS and soft scaling. CloudCoordinator continuously1 has to adapt allocation of VMs by issuing VM migration commands and changing power states of nodes according to its policy and current utilization of resources.

/\*

\* Title: CloudSim Toolkit

\* Description: CloudSim (Cloud Simulation) Toolkit for Modeling and Simulation

\* of Clouds

\* Licence: GPL - http://www.gnu.org/copyleft/gpl.html

\*

\* Copyright (c) 2009, The University of Melbourne, Australia

\*/

package org.cloudbus.cloudsim.examples.network;

import java.text.DecimalFormat;

import java.util.ArrayList;

import java.util.Calendar;

import java.util.LinkedList;

import java.util.List;

import org.cloudbus.cloudsim.Cloudlet;

import org.cloudbus.cloudsim.CloudletSchedulerTimeShared;

import org.cloudbus.cloudsim.Datacenter;

import org.cloudbus.cloudsim.DatacenterBroker;

import org.cloudbus.cloudsim.DatacenterCharacteristics;

import org.cloudbus.cloudsim.Host;

import org.cloudbus.cloudsim.Log;

import org.cloudbus.cloudsim.NetworkTopology;

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.VmAllocationPolicySimple;

import org.cloudbus.cloudsim.VmSchedulerTimeShared;

import org.cloudbus.cloudsim.core.CloudSim;

import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;

import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;

import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;

/\*\*

\* A simple example showing how to create

\* a datacenter with one host and a network

\* topology and and run one cloudlet on it.

\*/

public class NetworkExample1 {

/\*\* The cloudlet list. \*/

private static List<Cloudlet> cloudletList;

/\*\* The vmlist. \*/

private static List<Vm> vmlist;

/\*\*

\* Creates main() to run this example

\*/

public static void main(String[] args) {

Log.printLine("Starting NetworkExample1...");

try {

// First step: Initialize the CloudSim package. It should be called

// before creating any entities.

int num\_user = 1; // number of cloud users

Calendar calendar = Calendar.getInstance();

boolean trace\_flag = false; // mean trace events

// Initialize the CloudSim library

CloudSim.init(num\_user, calendar, trace\_flag);

// Second step: Create Datacenters

//Datacenters are the resource providers in CloudSim. We need at list one of them to run a CloudSim simulation

Datacenter datacenter0 = createDatacenter("Datacenter\_0");

//Third step: Create Broker

DatacenterBroker broker = createBroker();

int brokerId = broker.getId();

//Fourth step: Create one virtual machine

vmlist = new ArrayList<Vm>();

//VM description

int vmid = 0;

int mips = 250;

long size = 10000; //image size (MB)

int ram = 512; //vm memory (MB)

long bw = 1000;

int pesNumber = 1; //number of cpus

String vmm = "Xen"; //VMM name

//create VM

Vm vm1 = 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);

//Fifth step: Create one Cloudlet

cloudletList = new ArrayList<Cloudlet>();

//Cloudlet properties

int id = 0;

long length = 40000;

long fileSize = 300;

long outputSize = 300;

UtilizationModel utilizationModel = new UtilizationModelFull();

Cloudlet cloudlet1 = new Cloudlet(id, length, pesNumber, fileSize, outputSize, utilizationModel, utilizationModel, utilizationModel);

cloudlet1.setUserId(brokerId);

//add the cloudlet to the list

cloudletList.add(cloudlet1);

//submit cloudlet list to the broker

broker.submitCloudletList(cloudletList);

//Sixth step: configure network

//load the network topology file

NetworkTopology.buildNetworkTopology("topology.brite");

//maps CloudSim entities to BRITE entities

//PowerDatacenter will correspond to BRITE node 0

int briteNode=0;

NetworkTopology.mapNode(datacenter0.getId(),briteNode);

//Broker will correspond to BRITE node 3

briteNode=3;

NetworkTopology.mapNode(broker.getId(),briteNode);

// Seventh step: Starts the simulation

CloudSim.startSimulation();

// Final step: Print results when simulation is over

List<Cloudlet> newList = broker.getCloudletReceivedList();

CloudSim.stopSimulation();

printCloudletList(newList);

Log.printLine("NetworkExample1 finished!");

}

catch (Exception e) {

e.printStackTrace();

Log.printLine("The simulation has been terminated due to an unexpected error");

}

}

private static Datacenter createDatacenter(String name){

// Here are the steps needed to create a PowerDatacenter:

// 1. We need to create a list to store

// our machine

List<Host> hostList = new ArrayList<Host>();

// 2. A Machine contains one or more PEs or CPUs/Cores.

// In this example, it will have only one core.

List<Pe> peList = new ArrayList<Pe>();

int mips = 1000;

// 3. Create PEs and add these into a list.

peList.add(new Pe(0, new PeProvisionerSimple(mips))); // need to store Pe id and MIPS Rating

//4. Create Host with its id and list of PEs and add them to the list of machines

int hostId=0;

int ram = 2048; //host memory (MB)

long storage = 1000000; //host storage

int bw = 10000;

hostList.add(

new Host(

hostId,

new RamProvisionerSimple(ram),

new BwProvisionerSimple(bw),

storage,

peList,

new VmSchedulerTimeShared(peList)

)

); // This is our machine

// 5. Create a DatacenterCharacteristics object that stores the

// properties of a data center: architecture, OS, list of

// Machines, allocation policy: time- or space-shared, time zone

// and its price (G$/Pe time unit).

String arch = "x86"; // system architecture

String os = "Linux"; // operating system

String vmm = "Xen";

double time\_zone = 10.0; // time zone this resource located

double cost = 3.0; // the cost of using processing in this resource

double costPerMem = 0.05; // the cost of using memory in this resource

double costPerStorage = 0.001; // the cost of using storage in this resource

double costPerBw = 0.0; // the cost of using bw in this resource

LinkedList<Storage> storageList = new LinkedList<Storage>(); //we are not adding SAN devices by now

DatacenterCharacteristics characteristics = new DatacenterCharacteristics(

arch, os, vmm, hostList, time\_zone, cost, costPerMem,

costPerStorage, costPerBw);

// 6. Finally, we need to create a PowerDatacenter object.

Datacenter datacenter = null;

try {

datacenter = new Datacenter(name, characteristics, new VmAllocationPolicySimple(hostList), storageList, 0);

} catch (Exception e) {

e.printStackTrace();

}

return datacenter;

}

//We strongly encourage users to develop their own broker policies, to submit vms and cloudlets according

//to the specific rules of the simulated scenario

private static DatacenterBroker createBroker(){

DatacenterBroker broker = null;

try {

broker = new DatacenterBroker("Broker");

} catch (Exception e) {

e.printStackTrace();

return null;

}

return broker;

}

/\*\*

\* Prints the Cloudlet objects

\* @param list list of Cloudlets

\*/

private static void printCloudletList(List<Cloudlet> list) {

int size = list.size();

Cloudlet cloudlet;

String indent = " ";

Log.printLine();

Log.printLine("========== OUTPUT ==========");

Log.printLine("Cloudlet ID" + indent + "STATUS" + indent +

"Data center ID" + indent + "VM ID" + indent + "Time" + indent + "Start Time" + indent + "Finish Time");

for (int i = 0; i < size; i++) {

cloudlet = list.get(i);

Log.print(indent + cloudlet.getCloudletId() + indent + indent);

if (cloudlet.getCloudletStatus() == Cloudlet.SUCCESS){

Log.print("SUCCESS");

DecimalFormat dft = new DecimalFormat("###.##");

Log.printLine( indent + indent + cloudlet.getResourceId() + indent + indent + indent + cloudlet.getVmId() +

indent + indent + dft.format(cloudlet.getActualCPUTime()) + indent + indent + dft.format(cloudlet.getExecStartTime())+

indent + indent + dft.format(cloudlet.getFinishTime()));

}

}

}

}

========== OUTPUT ==========

Cloudlet ID STATUS Data center ID VM ID Time Start Time Finish Time

0 SUCCESS 2 0 160 0.1 160.1

NetworkExample1 finished!

4. Model and simulate the environment consisting of a data center with 10,000 hosts where each host was modeled to have a single CPU core (1200MIPS), 4GB of RAM memory and 2TB of storage. Consider the provisioning policy for VMs as space-shared, which allows one VM to be active in a host at a given instance of time. Make a request from the end-user (through the Datacenter Broker) for creation and instantiation of 50 VMs that had following constraints: 1024MB of physical memory, 1 CPU core and 1GB of storage. The application granularity was modeled to be composed of 300 task units, with each task unit requiring 1,440,000 million instructions (20 minutes in the simulated hosts) to be executed on a host. Minimal data transfer (300 KB) overhead can be considered for the task units (to and from the data center). After the creation of VMs, task units were submitted in small groups of 50 (one for each VM) at inter-arrival delay of 10 minutes.

/\*

\* Title: CloudSim Toolkit

\* Description: CloudSim (Cloud Simulation) Toolkit for Modeling and Simulation

\* of Clouds

\* Licence: GPL - http://www.gnu.org/copyleft/gpl.html

\*

\* Copyright (c) 2009, The University of Melbourne, Australia

\*/

package org.cloudbus.cloudsim.examples.network;

import java.text.DecimalFormat;

import java.util.ArrayList;

import java.util.Calendar;

import java.util.LinkedList;

import java.util.List;

import org.cloudbus.cloudsim.Cloudlet;

import org.cloudbus.cloudsim.CloudletSchedulerTimeShared;

import org.cloudbus.cloudsim.Datacenter;

import org.cloudbus.cloudsim.DatacenterBroker;

import org.cloudbus.cloudsim.DatacenterCharacteristics;

import org.cloudbus.cloudsim.Host;

import org.cloudbus.cloudsim.Log;

import org.cloudbus.cloudsim.NetworkTopology;

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.VmAllocationPolicySimple;

import org.cloudbus.cloudsim.VmSchedulerSpaceShared;

import org.cloudbus.cloudsim.core.CloudSim;

import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;

import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;

import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;

/\*\*

\* A simple example showing how to create

\* two datacenters with one host each and

\* run cloudlets of two users with network

\* topology on them.

\*/

public class NetworkExample3 {

/\*\* The cloudlet list. \*/

private static List<Cloudlet> cloudletList1;

private static List<Cloudlet> cloudletList2;

/\*\* The vmlist. \*/

private static List<Vm> vmlist1;

private static List<Vm> vmlist2;

/\*\*

\* Creates main() to run this example

\*/

public static void main(String[] args) {

Log.printLine("Starting NetworkExample3...");

try {

// First step: Initialize the CloudSim package. It should be called

// before creating any entities.

int num\_user = 2; // number of cloud users

Calendar calendar = Calendar.getInstance();

boolean trace\_flag = false; // mean trace events

// Initialize the CloudSim library

CloudSim.init(num\_user, calendar, trace\_flag);

// Second step: Create Datacenters

//Datacenters are the resource providers in CloudSim. We need at list one of them to run a CloudSim simulation

Datacenter datacenter0 = createDatacenter("Datacenter\_0");

Datacenter datacenter1 = createDatacenter("Datacenter\_1");

//Third step: Create Brokers

DatacenterBroker broker1 = createBroker(1);

int brokerId1 = broker1.getId();

DatacenterBroker broker2 = createBroker(2);

int brokerId2 = broker2.getId();

//Fourth step: Create one virtual machine for each broker/user

vmlist1 = new ArrayList<Vm>();

vmlist2 = new ArrayList<Vm>();

//VM description

int vmid = 0;

long size = 10000; //image size (MB)

int mips = 250;

int ram = 512; //vm memory (MB)

long bw = 1000;

int pesNumber = 1; //number of cpus

String vmm = "Xen"; //VMM name

//create two VMs: the first one belongs to user1

Vm vm1 = new Vm(vmid, brokerId1, mips, pesNumber, ram, bw, size, vmm, new CloudletSchedulerTimeShared());

//the second VM: this one belongs to user2

Vm vm2 = new Vm(vmid, brokerId2, mips, pesNumber, ram, bw, size, vmm, new CloudletSchedulerTimeShared());

//add the VMs to the vmlists

vmlist1.add(vm1);

vmlist2.add(vm2);

//submit vm list to the broker

broker1.submitVmList(vmlist1);

broker2.submitVmList(vmlist2);

//Fifth step: Create two Cloudlets

cloudletList1 = new ArrayList<Cloudlet>();

cloudletList2 = new ArrayList<Cloudlet>();

//Cloudlet properties

int id = 0;

long length = 40000;

long fileSize = 300;

long outputSize = 300;

UtilizationModel utilizationModel = new UtilizationModelFull();

Cloudlet cloudlet1 = new Cloudlet(id, length, pesNumber, fileSize, outputSize, utilizationModel, utilizationModel, utilizationModel);

cloudlet1.setUserId(brokerId1);

Cloudlet cloudlet2 = new Cloudlet(id, length, pesNumber, fileSize, outputSize, utilizationModel, utilizationModel, utilizationModel);

cloudlet2.setUserId(brokerId2);

//add the cloudlets to the lists: each cloudlet belongs to one user

cloudletList1.add(cloudlet1);

cloudletList2.add(cloudlet2);

//submit cloudlet list to the brokers

broker1.submitCloudletList(cloudletList1);

broker2.submitCloudletList(cloudletList2);

//Sixth step: configure network

//load the network topology file

NetworkTopology.buildNetworkTopology("topology.brite");

//maps CloudSim entities to BRITE entities

//Datacenter0 will correspond to BRITE node 0

int briteNode=0;

NetworkTopology.mapNode(datacenter0.getId(),briteNode);

//Datacenter1 will correspond to BRITE node 2

briteNode=2;

NetworkTopology.mapNode(datacenter1.getId(),briteNode);

//Broker1 will correspond to BRITE node 3

briteNode=3;

NetworkTopology.mapNode(broker1.getId(),briteNode);

//Broker2 will correspond to BRITE node 4

briteNode=4;

NetworkTopology.mapNode(broker2.getId(),briteNode);

// Sixth step: Starts the simulation

CloudSim.startSimulation();

// Final step: Print results when simulation is over

List<Cloudlet> newList1 = broker1.getCloudletReceivedList();

List<Cloudlet> newList2 = broker2.getCloudletReceivedList();

CloudSim.stopSimulation();

Log.print("=============> User "+brokerId1+" ");

printCloudletList(newList1);

Log.print("=============> User "+brokerId2+" ");

printCloudletList(newList2);

Log.printLine("NetworkExample3 finished!");

}

catch (Exception e) {

e.printStackTrace();

Log.printLine("The simulation has been terminated due to an unexpected error");

}

}

private static Datacenter createDatacenter(String name){

// Here are the steps needed to create a PowerDatacenter:

// 1. We need to create a list to store

// our machine

List<Host> hostList = new ArrayList<Host>();

// 2. A Machine contains one or more PEs or CPUs/Cores.

// In this example, it will have only one core.

List<Pe> peList = new ArrayList<Pe>();

int mips = 1000;

// 3. Create PEs and add these into a list.

peList.add(new Pe(0, new PeProvisionerSimple(mips))); // need to store Pe id and MIPS Rating

//4. Create Host with its id and list of PEs and add them to the list of machines

int hostId=0;

int ram = 2048; //host memory (MB)

long storage = 1000000; //host storage

int bw = 10000;

//in this example, the VMAllocatonPolicy in use is SpaceShared. It means that only one VM

//is allowed to run on each Pe. As each Host has only one Pe, only one VM can run on each Host.

hostList.add(

new Host(

hostId,

new RamProvisionerSimple(ram),

new BwProvisionerSimple(bw),

storage,

peList,

new VmSchedulerSpaceShared(peList)

)

); // This is our machine

// 5. Create a DatacenterCharacteristics object that stores the

// properties of a data center: architecture, OS, list of

// Machines, allocation policy: time- or space-shared, time zone

// and its price (G$/Pe time unit).

String arch = "x86"; // system architecture

String os = "Linux"; // operating system

String vmm = "Xen";

double time\_zone = 10.0; // time zone this resource located

double cost = 3.0; // the cost of using processing in this resource

double costPerMem = 0.05; // the cost of using memory in this resource

double costPerStorage = 0.001; // the cost of using storage in this resource

double costPerBw = 0.0; // the cost of using bw in this resource

LinkedList<Storage> storageList = new LinkedList<Storage>(); //we are not adding SAN devices by now

DatacenterCharacteristics characteristics = new DatacenterCharacteristics(

arch, os, vmm, hostList, time\_zone, cost, costPerMem,

costPerStorage, costPerBw);

// 6. Finally, we need to create a PowerDatacenter object.

Datacenter datacenter = null;

try {

datacenter = new Datacenter(name, characteristics, new VmAllocationPolicySimple(hostList), storageList, 0);

} catch (Exception e) {

e.printStackTrace();

}

return datacenter;

}

//We strongly encourage users to develop their own broker policies, to submit vms and cloudlets according

//to the specific rules of the simulated scenario

private static DatacenterBroker createBroker(int id){

DatacenterBroker broker = null;

try {

broker = new DatacenterBroker("Broker"+id);

} catch (Exception e) {

e.printStackTrace();

return null;

}

return broker;

}

/\*\*

\* Prints the Cloudlet objects

\* @param list list of Cloudlets

\*/

private static void printCloudletList(List<Cloudlet> list) {

int size = list.size();

Cloudlet cloudlet;

String indent = " ";

Log.printLine();

Log.printLine("========== OUTPUT ==========");

Log.printLine("Cloudlet ID" + indent + "STATUS" + indent +

"Data center ID" + indent + "VM ID" + indent + "Time" + indent + "Start Time" + indent + "Finish Time");

for (int i = 0; i < size; i++) {

cloudlet = list.get(i);

Log.print(indent + cloudlet.getCloudletId() + indent + indent);

if (cloudlet.getCloudletStatus() == Cloudlet.SUCCESS){

Log.print("SUCCESS");

DecimalFormat dft = new DecimalFormat("###.##");

Log.printLine( indent + indent + cloudlet.getResourceId() + indent + indent + indent + cloudlet.getVmId() +

indent + indent + dft.format(cloudlet.getActualCPUTime()) + indent + indent + dft.format(cloudlet.getExecStartTime())+

indent + indent + dft.format(cloudlet.getFinishTime()));

}

}

}

}

=============> User 4

========== OUTPUT ==========

Cloudlet ID STATUS Data center ID VM ID Time Start Time Finish Time

0 SUCCESS 2 0 160 0.1 160.1

=============> User 5

========== OUTPUT ==========

Cloudlet ID STATUS Data center ID VM ID Time Start Time Finish Time

0 SUCCESS 3 0 160 0.2 160.2

NetworkExample3 finished!

5. Implement Map Reduce concept for

a. Strassen’s Matrix Multiplication for a huge matrix.

b. Computing the average number of citation index a researcher has according to age among some 1 billion journal articles. Consider a network of entities and relationships between them. It is required to calculate a state of each entity on the basis of properties of the other entities in its neighborhood. This state can represent a distance to other nodes, indication that there is a neighbor with the certain properties, characteristic of neighborhood density and so on. A network is stored as a set of nodes and each node contains a list of adjacent node IDs. Mapper emits messages for each node using ID of the adjacent node as a key. Reducer must re compute state and rewrite node with the new state. Implement this scenario.

/\*

\* Title: CloudSim Toolkit

\* Description: CloudSim (Cloud Simulation) Toolkit for Modeling and Simulation

\* of Clouds

\* Licence: GPL - http://www.gnu.org/copyleft/gpl.html

\*

\* Copyright (c) 2009, The University of Melbourne, Australia

\*/

package org.cloudbus.cloudsim.examples;

import java.text.DecimalFormat;

import java.util.ArrayList;

import java.util.Calendar;

import java.util.LinkedList;

import java.util.List;

import org.cloudbus.cloudsim.Cloudlet;

import org.cloudbus.cloudsim.CloudletSchedulerTimeShared;

import org.cloudbus.cloudsim.Datacenter;

import org.cloudbus.cloudsim.DatacenterBroker;

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.VmAllocationPolicySimple;

import org.cloudbus.cloudsim.VmSchedulerTimeShared;

import org.cloudbus.cloudsim.core.CloudSim;

import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;

import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;

import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;

/\*\*

\* An example showing how to pause and resume the simulation,

\* and create simulation entities (a DatacenterBroker in this example)

\* dynamically.

\*/

public class CloudSimExample7 {

/\*\* The cloudlet list. \*/

private static List<Cloudlet> cloudletList;

/\*\* The vmlist. \*/

private static List<Vm> vmlist;

private static List<Vm> createVM(int userId, int vms, int idShift) {

//Creates a container to store VMs. This list is passed to the broker later

LinkedList<Vm> list = new LinkedList<Vm>();

//VM Parameters

long size = 10000; //image size (MB)

int ram = 512; //vm memory (MB)

int mips = 250;

long bw = 1000;

int pesNumber = 1; //number of cpus

String vmm = "Xen"; //VMM name

//create VMs

Vm[] vm = new Vm[vms];

for(int i=0;i<vms;i++){

vm[i] = new Vm(idShift + i, userId, mips, pesNumber, ram, bw, size, vmm, new CloudletSchedulerTimeShared());

list.add(vm[i]);

}

return list;

}

private static List<Cloudlet> createCloudlet(int userId, int cloudlets, int idShift){

// Creates a container to store Cloudlets

LinkedList<Cloudlet> list = new LinkedList<Cloudlet>();

//cloudlet parameters

long length = 40000;

long fileSize = 300;

long outputSize = 300;

int pesNumber = 1;

UtilizationModel utilizationModel = new UtilizationModelFull();

Cloudlet[] cloudlet = new Cloudlet[cloudlets];

for(int i=0;i<cloudlets;i++){

cloudlet[i] = new Cloudlet(idShift + i, length, pesNumber, fileSize, outputSize, utilizationModel, utilizationModel, utilizationModel);

// setting the owner of these Cloudlets

cloudlet[i].setUserId(userId);

list.add(cloudlet[i]);

}

return list;

}

////////////////////////// STATIC METHODS ///////////////////////

/\*\*

\* Creates main() to run this example

\*/

public static void main(String[] args) {

Log.printLine("Starting CloudSimExample7...");

try {

// First step: Initialize the CloudSim package. It should be called

// before creating any entities.

int num\_user = 2; // number of grid users

Calendar calendar = Calendar.getInstance();

boolean trace\_flag = false; // mean trace events

// Initialize the CloudSim library

CloudSim.init(num\_user, calendar, trace\_flag);

// Second step: Create Datacenters

//Datacenters are the resource providers in CloudSim. We need at list one of them to run a CloudSim simulation

@SuppressWarnings("unused")

Datacenter datacenter0 = createDatacenter("Datacenter\_0");

@SuppressWarnings("unused")

Datacenter datacenter1 = createDatacenter("Datacenter\_1");

//Third step: Create Broker

DatacenterBroker broker = createBroker("Broker\_0");

int brokerId = broker.getId();

//Fourth step: Create VMs and Cloudlets and send them to broker

vmlist = createVM(brokerId, 5, 0); //creating 5 vms

cloudletList = createCloudlet(brokerId, 10, 0); // creating 10 cloudlets

broker.submitVmList(vmlist);

broker.submitCloudletList(cloudletList);

// A thread that will create a new broker at 200 clock time

Runnable monitor = new Runnable() {

@Override

public void run() {

CloudSim.pauseSimulation(200);

while (true) {

if (CloudSim.isPaused()) {

break;

}

try {

Thread.sleep(100);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

Log.printLine("\n\n\n" + CloudSim.clock() + ": The simulation is paused for 5 sec \n\n");

try {

Thread.sleep(5000);

} catch (InterruptedException e) {

e.printStackTrace();

}

DatacenterBroker broker = createBroker("Broker\_1");

int brokerId = broker.getId();

//Create VMs and Cloudlets and send them to broker

vmlist = createVM(brokerId, 5, 100); //creating 5 vms

cloudletList = createCloudlet(brokerId, 10, 100); // creating 10 cloudlets

broker.submitVmList(vmlist);

broker.submitCloudletList(cloudletList);

CloudSim.resumeSimulation();

}

};

new Thread(monitor).start();

Thread.sleep(1000);

// Fifth step: Starts the simulation

CloudSim.startSimulation();

// Final step: Print results when simulation is over

List<Cloudlet> newList = broker.getCloudletReceivedList();

CloudSim.stopSimulation();

printCloudletList(newList);

Log.printLine("CloudSimExample7 finished!");

}

catch (Exception e)

{

e.printStackTrace();

Log.printLine("The simulation has been terminated due to an unexpected error");

}

}

private static Datacenter createDatacenter(String name){

// Here are the steps needed to create a PowerDatacenter:

// 1. We need to create a list to store one or more

// Machines

List<Host> hostList = new ArrayList<Host>();

// 2. A Machine contains one or more PEs or CPUs/Cores. Therefore, should

// create a list to store these PEs before creating

// a Machine.

List<Pe> peList1 = new ArrayList<Pe>();

int mips = 1000;

// 3. Create PEs and add these into the list.

//for a quad-core machine, a list of 4 PEs is required:

peList1.add(new Pe(0, new PeProvisionerSimple(mips))); // need to store Pe id and MIPS Rating

peList1.add(new Pe(1, new PeProvisionerSimple(mips)));

peList1.add(new Pe(2, new PeProvisionerSimple(mips)));

peList1.add(new Pe(3, new PeProvisionerSimple(mips)));

//Another list, for a dual-core machine

List<Pe> peList2 = new ArrayList<Pe>();

peList2.add(new Pe(0, new PeProvisionerSimple(mips)));

peList2.add(new Pe(1, new PeProvisionerSimple(mips)));

//4. Create Hosts with its id and list of PEs and add them to the list of machines

int hostId=0;

int ram = 16384; //host memory (MB)

long storage = 1000000; //host storage

int bw = 10000;

hostList.add(

new Host(

hostId,

new RamProvisionerSimple(ram),

new BwProvisionerSimple(bw),

storage,

peList1,

new VmSchedulerTimeShared(peList1)

)

); // This is our first machine

hostId++;

hostList.add(

new Host(

hostId,

new RamProvisionerSimple(ram),

new BwProvisionerSimple(bw),

storage,

peList2,

new VmSchedulerTimeShared(peList2)

)

); // Second machine

// 5. Create a DatacenterCharacteristics object that stores the

// properties of a data center: architecture, OS, list of

// Machines, allocation policy: time- or space-shared, time zone

// and its price (G$/Pe time unit).

String arch = "x86"; // system architecture

String os = "Linux"; // operating system

String vmm = "Xen";

double time\_zone = 10.0; // time zone this resource located

double cost = 3.0; // the cost of using processing in this resource

double costPerMem = 0.05; // the cost of using memory in this resource

double costPerStorage = 0.1; // the cost of using storage in this resource

double costPerBw = 0.1; // the cost of using bw in this resource

LinkedList<Storage> storageList = new LinkedList<Storage>(); //we are not adding SAN devices by now

DatacenterCharacteristics characteristics = new DatacenterCharacteristics(

arch, os, vmm, hostList, time\_zone, cost, costPerMem, costPerStorage, costPerBw);

// 6. Finally, we need to create a PowerDatacenter object.

Datacenter datacenter = null;

try {

datacenter = new Datacenter(name, characteristics, new VmAllocationPolicySimple(hostList), storageList, 0);

} catch (Exception e) {

e.printStackTrace();

}

return datacenter;

}

//We strongly encourage users to develop their own broker policies, to submit vms and cloudlets according

//to the specific rules of the simulated scenario

private static DatacenterBroker createBroker(String name){

DatacenterBroker broker = null;

try {

broker = new DatacenterBroker(name);

} catch (Exception e) {

e.printStackTrace();

return null;

}

return broker;

}

/\*\*

\* Prints the Cloudlet objects

\* @param list list of Cloudlets

\*/

private static void printCloudletList(List<Cloudlet> list) {

int size = list.size();

Cloudlet cloudlet;

String indent = " ";

Log.printLine();

Log.printLine("========== OUTPUT ==========");

Log.printLine("Cloudlet ID" + indent + "STATUS" + indent +

"Data center ID" + indent + "VM ID" + indent + indent + "Time" + indent + "Start Time" + indent + "Finish Time");

DecimalFormat dft = new DecimalFormat("###.##");

for (int i = 0; i < size; i++) {

cloudlet = list.get(i);

Log.print(indent + cloudlet.getCloudletId() + indent + indent);

if (cloudlet.getCloudletStatus() == Cloudlet.SUCCESS){

Log.print("SUCCESS");

Log.printLine( indent + indent + cloudlet.getResourceId() + indent + indent + indent + cloudlet.getVmId() +

indent + indent + indent + dft.format(cloudlet.getActualCPUTime()) +

indent + indent + dft.format(cloudlet.getExecStartTime())+ indent + indent + indent + dft.format(cloudlet.getFinishTime()));

}

}

}

}

========== OUTPUT ==========

Cloudlet ID STATUS Data center ID VM ID Time Start Time Finish Time

0 SUCCESS 2 0 320 0.1 320.1

5 SUCCESS 2 0 320 0.1 320.1

1 SUCCESS 2 1 320 0.1 320.1

6 SUCCESS 2 1 320 0.1 320.1

2 SUCCESS 2 2 320 0.1 320.1

7 SUCCESS 2 2 320 0.1 320.1

4 SUCCESS 2 4 320 0.1 320.1

9 SUCCESS 2 4 320 0.1 320.1

3 SUCCESS 2 3 320 0.1 320.1

8 SUCCESS 2 3 320 0.1 320.1

CloudSimExample7 finished!