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()));

}

}

}

}