Name: Nikhil B Patil  
Roll No: 35063  
Batch: D

**Assignment No. 3**

1. Constants.java

package <package\_name>;

public class Constants {

    public static final int NO\_OF\_TASKS = 30;

    public static final int NO\_OF\_DATA\_CENTERS = 5;

    public static final int POPULATION\_SIZE = 25;

}

1. **DatacenterCreater.java**
2. import org.cloudbus.cloudsim.\*;
3. import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;
4. import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;
5. import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;
6. import java.util.ArrayList;
7. import java.util.LinkedList;
8. import java.util.List;
9. public class DatacenterCreator {
10. public static Datacenter createDatacenter(String name) {
11. List<Host> hostList = new ArrayList<Host>();
12. List<Pe> peList = new ArrayList<Pe>();
13. int mips = 1000;
14. peList.add(new Pe(0, new PeProvisionerSimple(mips)));
15. int hostId = 0;
16. int ram = 2048; //host memory (MB)
17. long storage = 1000000; //host storage
18. int bw = 10000;
19. hostList.add(
20. new Host(
21. hostId,
22. new RamProvisionerSimple(ram),
23. new BwProvisionerSimple(bw),
24. storage,
25. peList,
26. new VmSchedulerTimeShared(peList)
27. )
28. );
29. String arch = "x86";      // system architecture
30. String os = "Linux";          // operating system
31. String vmm = "Xen";
32. double time\_zone = 10.0;         // time zone this resource located
33. double cost = 3.0;              // the cost of using processing in this resource
34. double costPerMem = 0.05;        // the cost of using memory in this resource
35. double costPerStorage = 0.1;    // the cost of using storage in this resource
36. double costPerBw = 0.1;            // the cost of using bw in this resource
37. LinkedList<Storage> storageList = new LinkedList<Storage>();    //we are not adding SAN devices by now
38. DatacenterCharacteristics characteristics = new DatacenterCharacteristics(
39. arch, os, vmm, hostList, time\_zone, cost, costPerMem, costPerStorage, costPerBw);
40. // 6. Finally, we need to create a PowerDatacenter object.
41. Datacenter datacenter = null;
42. try {
43. datacenter = new Datacenter(name, characteristics, new VmAllocationPolicySimple(hostList), storageList, 0);
44. } catch (Exception e) {
45. e.printStackTrace();
46. }
47. return datacenter;
48. }
49. }

3) **Generatematrices.java**

package <package\_name>;

import java.io.\*;

public class GenerateMatrices {

    private static double[][] commMatrix, execMatrix;

    private File commFile = new File("CommunicationTimeMatrix.txt");

    private File execFile = new File("ExecutionTimeMatrix.txt");

    public GenerateMatrices() {

        commMatrix = new double[Constants.NO\_OF\_TASKS][Constants.NO\_OF\_DATA\_CENTERS];

        execMatrix = new double[Constants.NO\_OF\_TASKS][Constants.NO\_OF\_DATA\_CENTERS];

        try {

            if (commFile.exists() && execFile.exists()) {

                readCostMatrix();

            } else {

                initCostMatrix();

            }

        } catch (IOException e) {

            e.printStackTrace();

        }

    }

    private void initCostMatrix() throws IOException {

        System.out.println("Initializing new Matrices...");

        BufferedWriter commBufferedWriter = new BufferedWriter(new FileWriter(commFile));

        BufferedWriter execBufferedWriter = new BufferedWriter(new FileWriter(execFile));

        for (int i = 0; i < Constants.NO\_OF\_TASKS; i++) {

            for (int j = 0; j < Constants.NO\_OF\_DATA\_CENTERS; j++) {

                commMatrix[i][j] = Math.random() \* 600 + 20;

                execMatrix[i][j] = Math.random() \* 500 + 10;

                commBufferedWriter.write(String.valueOf(commMatrix[i][j]) + ' ');

                execBufferedWriter.write(String.valueOf(execMatrix[i][j]) + ' ');

            }

            commBufferedWriter.write('\n');

            execBufferedWriter.write('\n');

        }

        commBufferedWriter.close();

        execBufferedWriter.close();

    }

    private void readCostMatrix() throws IOException {

        System.out.println("Reading the Matrices...");

        BufferedReader commBufferedReader = new BufferedReader(new FileReader(commFile));

        int i = 0, j = 0;

        do {

            String line = commBufferedReader.readLine();

            for (String num : line.split(" ")) {

                commMatrix[i][j++] = new Double(num);

            }

            ++i;

            j = 0;

        } while (commBufferedReader.ready());

        BufferedReader execBufferedReader = new BufferedReader(new FileReader(execFile));

        i = j = 0;

        do {

            String line = execBufferedReader.readLine();

            for (String num : line.split(" ")) {

                execMatrix[i][j++] = new Double(num);

            }

            ++i;

            j = 0;

        } while (execBufferedReader.ready());

    }

    public static double[][] getCommMatrix() {

        return commMatrix;

    }

    public static double[][] getExecMatrix() {

        return execMatrix;

    }

}

**4) SJF\_Scheduler.java**

package <package\_name>;

import org.cloudbus.cloudsim.\*;

import org.cloudbus.cloudsim.core.CloudSim;

import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;

import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;

import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;

//import utils.Constants;

//import utils.DatacenterCreator;

//import utils.GenerateMatrices;

import java.text.DecimalFormat;

import java.util.ArrayList;

import java.util.Calendar;

import java.util.LinkedList;

import java.util.List;

public class SJF\_Scheduler {

    private static List<Cloudlet> cloudletList;

    private static List<Vm> vmList;

    private static Datacenter[] datacenter;

    private static double[][] commMatrix;

    private static double[][] execMatrix;

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

        //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(datacenter[i].getId(), userId, mips, pesNumber, ram, bw, size, vmm, new CloudletSchedulerSpaceShared());

            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 fileSize = 300;

        long outputSize = 300;

        int pesNumber = 1;

        UtilizationModel utilizationModel = new UtilizationModelFull();

        Cloudlet[] cloudlet = new Cloudlet[cloudlets];

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

            int dcId = (int) (Math.random() \* Constants.NO\_OF\_DATA\_CENTERS);

            long length = (long) (1e3 \* (commMatrix[i][dcId] + execMatrix[i][dcId]));

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

            // setting the owner of these Cloudlets

            cloudlet[i].setUserId(userId);

            cloudlet[i].setVmId(dcId + 2);

            list.add(cloudlet[i]);

        }

        return list;

    }

    public static void main(String[] args) {

        Log.printLine("Starting SJF Scheduler...");

        new GenerateMatrices();

        execMatrix = GenerateMatrices.getExecMatrix();

        commMatrix = GenerateMatrices.getCommMatrix();

        try {

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

            Calendar calendar = Calendar.getInstance();

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

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

            // Second step: Create Datacenters

            datacenter = new Datacenter[Constants.NO\_OF\_DATA\_CENTERS];

            for (int i = 0; i < Constants.NO\_OF\_DATA\_CENTERS; i++) {

                datacenter[i] = DatacenterCreator.createDatacenter("Datacenter\_" + i);

            }

            //Third step: Create Broker

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

            int brokerId = broker.getId();

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

            vmList = createVM(brokerId, Constants.NO\_OF\_DATA\_CENTERS);

            cloudletList = createCloudlet(brokerId, Constants.NO\_OF\_TASKS, 0);

            broker.submitVmList(vmList);

            broker.submitCloudletList(cloudletList);

            // Fifth step: Starts the simulation

            CloudSim.startSimulation();

            // Final step: Print results when simulation is over

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

            //newList.addAll(globalBroker.getBroker().getCloudletReceivedList());

            CloudSim.stopSimulation();

            printCloudletList(newList);

            Log.printLine(SJF\_Scheduler.class.getName() + " finished!");

        } catch (Exception e) {

            e.printStackTrace();

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

        }

    }

    private static SJFDatacenterBroker createBroker(String name) throws Exception {

        return new SJFDatacenterBroker(name);

    }

    /\*\*

     \* 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" +

                indent + "Waiting Time");

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

        dft.setMinimumIntegerDigits(2);

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

            cloudlet = list.get(i);

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

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

                Log.print("SUCCESS");

                Log.printLine(indent + indent + dft.format(cloudlet.getResourceId()) +

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

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

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

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

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

            }

        }

        double makespan = calcMakespan(list);

        Log.printLine("Makespan using SJF: " + makespan);

    }

    private static double calcMakespan(List<Cloudlet> list) {

        double makespan = 0;

        double[] dcWorkingTime = new double[Constants.NO\_OF\_DATA\_CENTERS];

        for (int i = 0; i < Constants.NO\_OF\_TASKS; i++) {

            int dcId = list.get(i).getVmId() % Constants.NO\_OF\_DATA\_CENTERS;

            if (dcWorkingTime[dcId] != 0) --dcWorkingTime[dcId];

            dcWorkingTime[dcId] += execMatrix[i][dcId] + commMatrix[i][dcId];

            makespan = Math.max(makespan, dcWorkingTime[dcId]);

        }

        return makespan;

    }

}

**5) SJF\_DataCenterBroker.java**

package <package\_name>;

import org.cloudbus.cloudsim.\*;

import org.cloudbus.cloudsim.core.CloudSim;

import org.cloudbus.cloudsim.core.CloudSimTags;

import org.cloudbus.cloudsim.core.SimEvent;

import java.util.ArrayList;

import java.util.List;

public class SJFDatacenterBroker extends DatacenterBroker {

    SJFDatacenterBroker(String name) throws Exception {

        super(name);

    }

    public void scheduleTaskstoVms() {

        int reqTasks = cloudletList.size();

        int reqVms = vmList.size();

        Vm vm = vmList.get(0);

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

            bindCloudletToVm(i, (i % reqVms));

            System.out.println("Task" + cloudletList.get(i).getCloudletId() + " is bound with VM" + vmList.get(i % reqVms).getId());

        }

        //System.out.println("reqTasks: "+ reqTasks);

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

        for (Cloudlet cloudlet : getCloudletReceivedList()) {

            list.add(cloudlet);

        }

        //setCloudletReceivedList(null);

        Cloudlet[] list2 = list.toArray(new Cloudlet[list.size()]);

        //System.out.println("size :"+list.size());

        Cloudlet temp = null;

        int n = list.size();

        for (int i = 0; i < n; i++) {

            for (int j = 1; j < (n - i); j++) {

                if (list2[j - 1].getCloudletLength() / (vm.getMips() \* vm.getNumberOfPes()) > list2[j].getCloudletLength() / (vm.getMips() \* vm.getNumberOfPes())) {

                    //swap the elements!

                    //swap(list2[j-1], list2[j]);

                    temp = list2[j - 1];

                    list2[j - 1] = list2[j];

                    list2[j] = temp;

                }

                // printNumbers(list2);

            }

        }

        ArrayList<Cloudlet> list3 = new ArrayList<Cloudlet>();

        for (int i = 0; i < list2.length; i++) {

            list3.add(list2[i]);

        }

        //printNumbers(list);

        setCloudletReceivedList(list);

        //System.out.println("\n\tSJFS Broker Schedules\n");

        //System.out.println("\n");

    }

    public void printNumber(Cloudlet[] list) {

        for (int i = 0; i < list.length; i++) {

            System.out.print(" " + list[i].getCloudletId());

            System.out.println(list[i].getCloudletStatusString());

        }

        System.out.println();

    }

    public void printNumbers(ArrayList<Cloudlet> list) {

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

            System.out.print(" " + list.get(i).getCloudletId());

        }

        System.out.println();

    }

    @Override

    protected void processCloudletReturn(SimEvent ev) {

        Cloudlet cloudlet = (Cloudlet) ev.getData();

        getCloudletReceivedList().add(cloudlet);

        Log.printLine(CloudSim.clock() + ": " + getName() + ": Cloudlet " + cloudlet.getCloudletId()

                + " received");

        cloudletsSubmitted--;

        if (getCloudletList().size() == 0 && cloudletsSubmitted == 0) {

            scheduleTaskstoVms();

            cloudletExecution(cloudlet);

        }

    }

    protected void cloudletExecution(Cloudlet cloudlet) {

        if (getCloudletList().size() == 0 && cloudletsSubmitted == 0) { // all cloudlets executed

            Log.printLine(CloudSim.clock() + ": " + getName() + ": All Cloudlets executed. Finishing...");

            clearDatacenters();

            finishExecution();

        } else { // some cloudlets haven't finished yet

            if (getCloudletList().size() > 0 && cloudletsSubmitted == 0) {

                // all the cloudlets sent finished. It means that some bount

                // cloudlet is waiting its VM be created

                clearDatacenters();

                createVmsInDatacenter(0);

            }

        }

    }

    @Override

    protected void processResourceCharacteristics(SimEvent ev) {

        DatacenterCharacteristics characteristics = (DatacenterCharacteristics) ev.getData();

        getDatacenterCharacteristicsList().put(characteristics.getId(), characteristics);

        if (getDatacenterCharacteristicsList().size() == getDatacenterIdsList().size()) {

            distributeRequestsForNewVmsAcrossDatacenters();

        }

    }

    protected void distributeRequestsForNewVmsAcrossDatacenters() {

        int numberOfVmsAllocated = 0;

        int i = 0;

        final List<Integer> availableDatacenters = getDatacenterIdsList();

        for (Vm vm : getVmList()) {

            int datacenterId = availableDatacenters.get(i++ % availableDatacenters.size());

            String datacenterName = CloudSim.getEntityName(datacenterId);

            if (!getVmsToDatacentersMap().containsKey(vm.getId())) {

                Log.printLine(CloudSim.clock() + ": " + getName() + ": Trying to Create VM #" + vm.getId() + " in " + datacenterName);

                sendNow(datacenterId, CloudSimTags.VM\_CREATE\_ACK, vm);

                numberOfVmsAllocated++;

            }

        }

        setVmsRequested(numberOfVmsAllocated);

        setVmsAcks(0);

    }

}