Ex No : 10 Date: 03/04/23	Implementation of Scheduling Algorithms using Cloudsim Simulator

Aim:

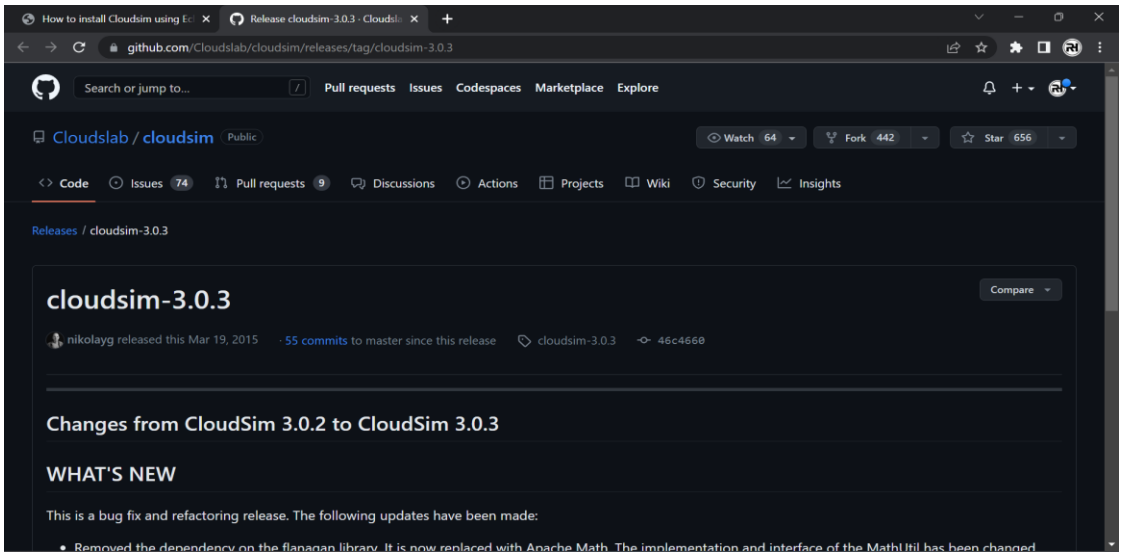
To simulate various cloud scenarios with cloudSim.

Theory:

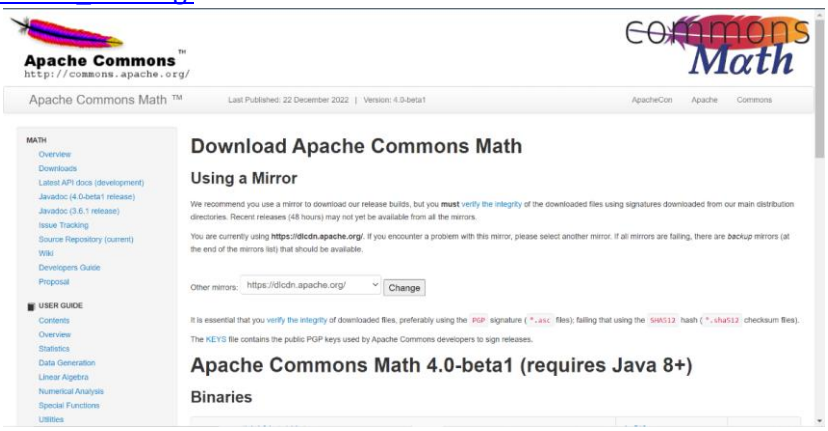
CloudSim is an open-source framework, which is used to simulate cloud computing infrastructure and services. It is developed by the CLOUDS Lab organization and is written entirely in Java. It is used for modelling and simulating a cloud computing environment as a means for evaluating a hypothesis prior to software development to reproduce tests and results.

Procedure:

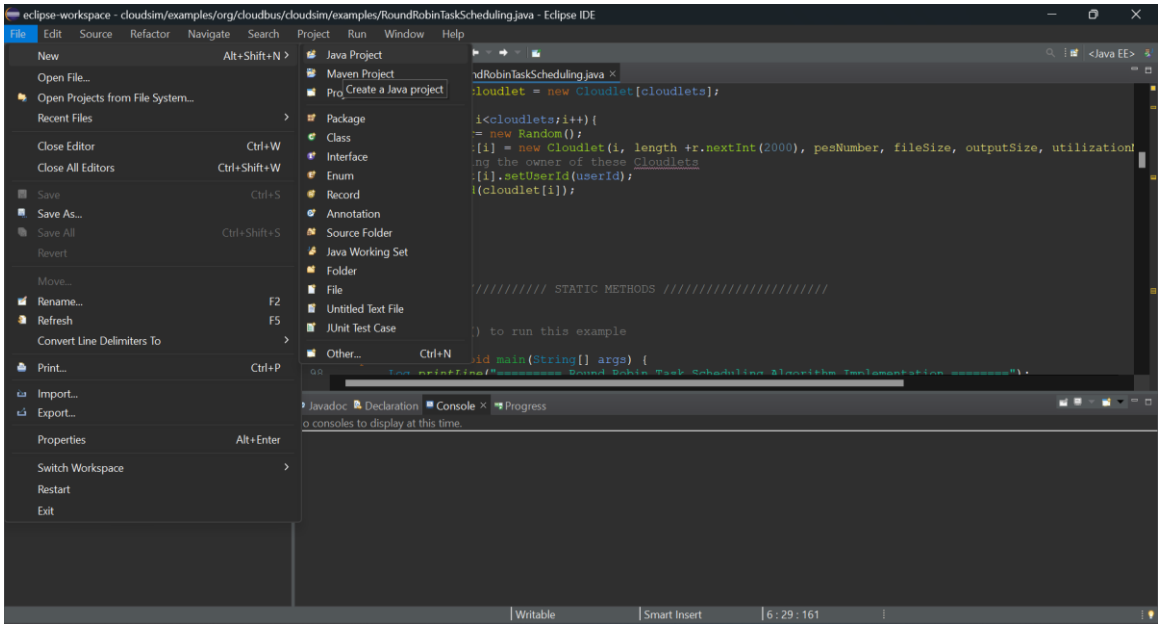
1. Download Cloudsim 3.0 from https://github.com/Cloudslab/cloudsim/releases/tag/5.0



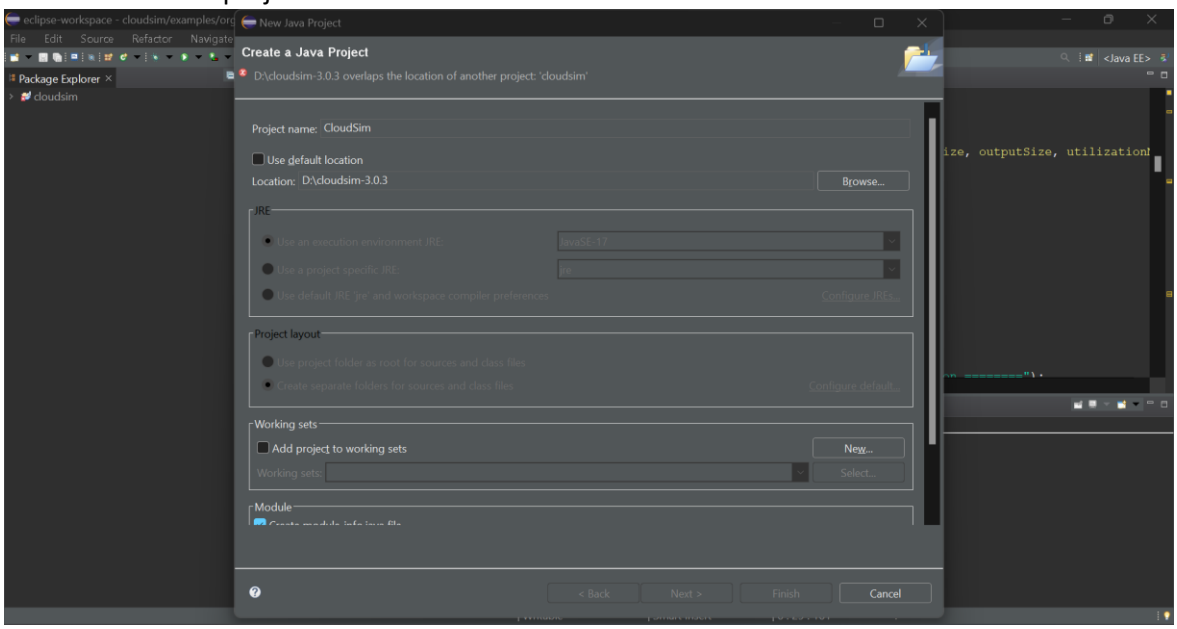
2. Add common math library https://commons.apache.org/proper/commons-math/download_math.cgi



3. Open Eclipse. Select New > Project > Java Project

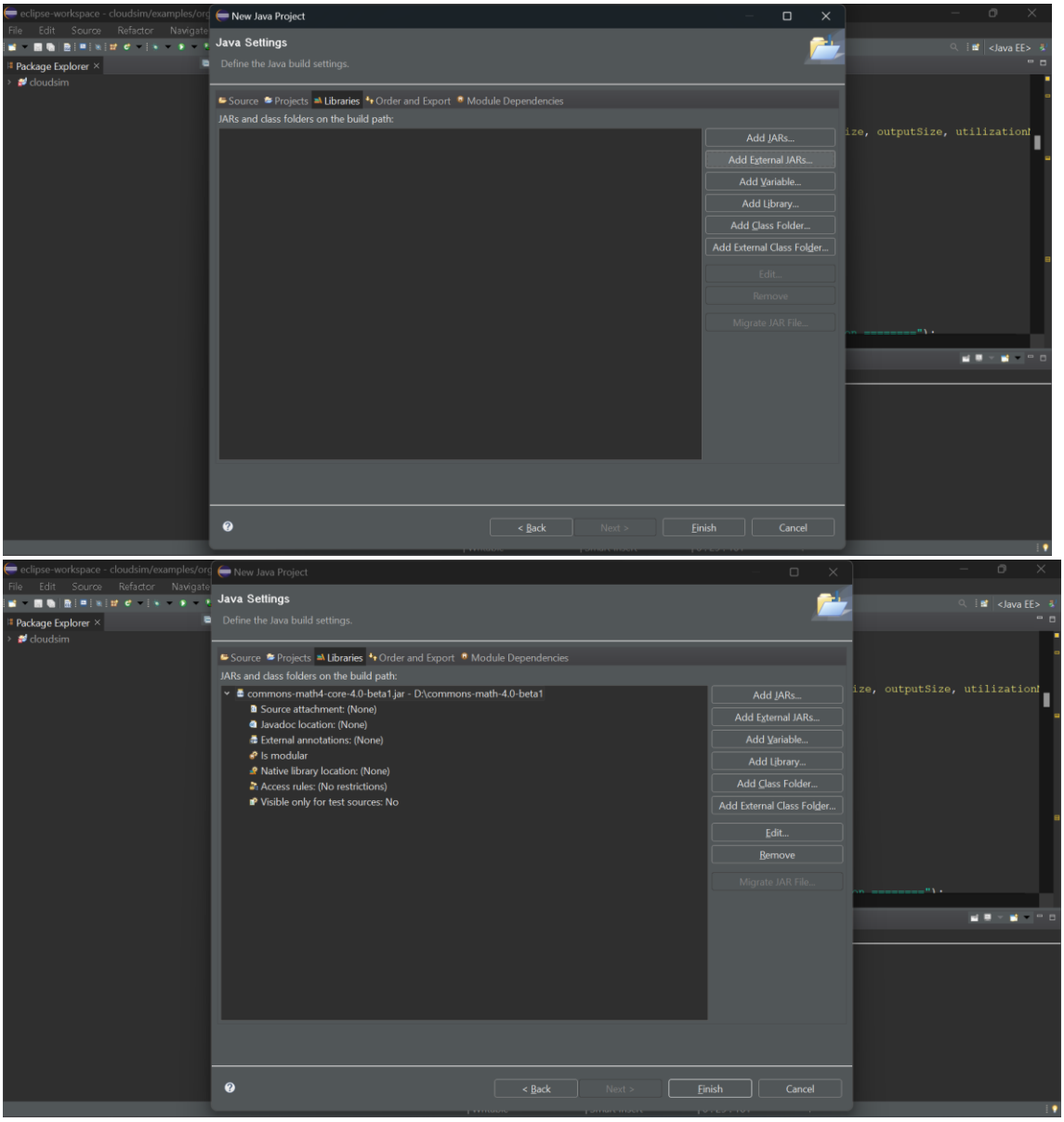


4. Give a name for the project and Select Java Execution environment.

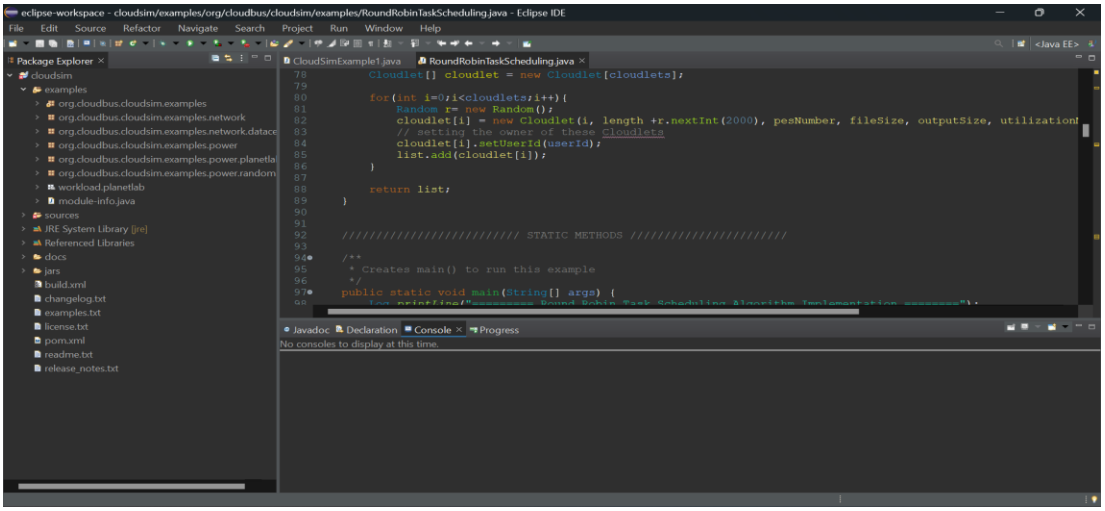


Once on clicking finish, Cloud sim project is viewable in the file explorer.

5. Right click on the project name “Cloudsim” – Select “Properties” – libraries – Add External JARs and select all the JAR files present in the common math folder.



6.Click apply and close, click on modules/cloudsim-examples and click run button.



```
Starting CloudSimExample1...
Initialising...
Starting CloudSim version 3.0
Datacenter_0 is starting...
Broker is starting...
Entities started.
0.0: Broker: Cloud Resource List received with 1 resource(s)
0.0: Broker: Trying to Create VM #0 in Datacenter_0
0.1: Broker: VM #0 has been created in Datacenter #2, Host #0
0.1: Broker: Sending cloudlet 0 to VM #0
400.1: Broker: Cloudlet 0 received
400.1: Broker: All Cloudlets executed. Finishing...
400.1: Broker: Destroying VM #0
Broker is shutting down...
Simulation: No more future events
CloudInformationService: Notify all CloudSim entities for shutting down.
Datacenter_0 is shutting down...
Broker is shutting down...
Simulation completed.
Simulation completed.

========== OUTPUT ==========
Cloudlet ID    STATUS    Data center ID    VM ID    Time    Start Time    Finish Time
    0        SUCCESS        2            0        400        0.1        400.1
CloudSimExample1 finished!
```

RoundRobinScheduling:

```
public class RoundRobinTaskScheduling {
        private static float timeSlice = (float) 8;
        /** The cloudlet list. */
        private static List<Cloudlet> cloudletList;
        /** The vmlist. */
        private static List<Vm> vmlist;
        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 = 1000;
                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(i, userId, mips, pesNumber, ram, bw, size, vmm, new CloudletSchedulerSpaceShared());
			//for creating a VM with a space shared scheduling policy for cloudlets:
			//vm[i] = Vm(i, userId, mips, pesNumber, ram, bw, size, vmm, new CloudletSchedulerSpaceShared());
			list.add(vm[i]);
		}
		return list;
	}
	private static List<Cloudlet> createCloudlet(int userId, int cloudlets){
		// Creates a container to store Cloudlets
		LinkedList<Cloudlet> list = new LinkedList<Cloudlet>();
		//cloudlet parameters
		long length = 1000;
		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++){
			Random r= new Random();
			cloudlet[i] = new Cloudlet(i, length +r.nextInt(2000), pesNumber, fileSize, outputSize, utilizationModel, utilizationModel, utilizationModel);
			// setting the owner of these Cloudlets
			cloudlet[i].setUserId(userId);
			list.add(cloudlet[i]);
		}
```

```
        return list;
    }
    public static void main(String[] args) {
        Log.printLine("========= Round Robin Task Scheduling Algorithm
Implementation ========");
        try {
            Log.printLine("======== Starting Execution ========");
            // First step: Initialize the CloudSim package. It should be called
            // before creating any entities.
            int num_user = 3;   // 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);
            @SuppressWarnings("not used")
            Datacenter datacenter0 = createDatacenter("Datacenter_0");
            Datacenter datacenter1 = createDatacenter("Datacenter_1");
            DatacenterBroker broker = createBroker();
            int brokerId = broker.getId();
            vmlist = createVM(brokerId,10); //creating 10 vms
            cloudletList = createCloudlet(brokerId,40); // creating 40 cloudlets
            broker.submitVmList(vmlist);
            broker.submitCloudletList(cloudletList);
            CloudSim.startSimulation();
            List<Cloudlet> newList = broker.getCloudletReceivedList();
            CloudSim.stopSimulation();
            printCloudletList(newList);
            Log.printLine("Round Robin has finished executing!");
        }catch (Exception e){
```

```
                e.printStackTrace();
                Log.printLine("The simulation has been terminated due to an
unexpected error");
        }}private static Datacenter createDatacenter(String name){
        List<Host> hostList = new ArrayList<Host>();
        List<Pe> peList1 = new ArrayList<Pe>();
        int mips = 1000;
        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)));
        List<Pe> peList2 = new ArrayList<Pe>();
        peList2.add(new Pe(0, new PeProvisionerSimple(mips)));
        peList2.add(new Pe(1, new PeProvisionerSimple(mips)));
        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,
                        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
        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;
	}
	private static DatacenterBroker createBroker(){
		DatacenterBroker broker = null;
		try {
			broker = new DatacenterBroker("Broker");
		} catch (Exception e) {
			e.printStackTrace();
			return null;
		}
		return broker;
	}
	@SuppressWarnings("deprecation")
	private static void printCloudletList(List<Cloudlet> list) {
		int size = list.size();
		Cloudlet cloudlet;
		int pes =0;
		float sum = 0;
		float burstTime[] = new float[size];
		float waitingTime[] = new float[size];
		float turnAroundTime[] = new float[size];
		float a[] = new float[size];
		String indent = "    ";
		DecimalFormat dft = new DecimalFormat("###.##");
		for(int i = 0; i<size; i++) {
			cloudlet = list.get(i);
			//We get the cpu time for each cloudlet
```

```
                String cpuTime = dft.format(cloudlet.getActualCPUTime());
                float convertedCPUTime = (float) Double.parseDouble(cpuTime);
                burstTime[i] = convertedCPUTime; //burst time is equal to execution
time.
        }
        for(int i=0; i<size; i++) {
                a[i] = burstTime[i];
        }
        for(int i=0; i<size; i++) {
                waitingTime[i] = 0;
        }
        do {for(int i=0; i<size; i++) {
                        if(burstTime[i]>timeSlice) {
                                burstTime[i] -= timeSlice;
                                for(int j=0; j<size; j++) {
                                        if((j != i) && (burstTime[j] != 0)) {
                                                waitingTime[j] += timeSlice;}}}
                        else {
                                for(int j=0; j<size; j++) {
                                        if((j != i) && (burstTime[j] != 0)) {
                                                waitingTime[j] += burstTime[i];}       }
                                burstTime[i] = 0;}}
                sum = 0;
                for(int k=0; k<size; k++) {
                        sum += burstTime[k];
                }
        }while(sum != 0);
        for(int i=0; i<size; i++) {
                turnAroundTime[i] = waitingTime[i] + a[i];
```

```
                }
                Log.printLine("========== OUTPUT ==========");
                Log.print("Cloudlet \t Burst Time \t Waiting Time \t Turn Around Time");
                Log.printLine();
                Log.print("-------------------------------------------------------------------");
                for(int i=0; i<size; i++) {
                        cloudlet = list.get(i);
                        pes = list.get(i).getNumberOfPes();
                        System.out.println("\n");
                        System.out.println("Cloudlet: "+cloudlet.getCloudletId()+ "\t\t" +a[i]+
"\t\t" +waitingTime[i]+ "\t\t" +turnAroundTime[i]);
                }
                /* Average waiting and turn around time */
                float averageWaitingTime = 0;
                float averageTurnAroundTime = 0;
                for(int j=0; j<size; j++) {
                        averageWaitingTime += waitingTime[j];
                }
                for(int j=0; j<size; j++) {
                        averageTurnAroundTime += turnAroundTime[j];
                }
                System.out.println("Average Waiting Time on Total: "
+(averageWaitingTime/size)+ "\nAverage Turn Around Time on Total: "
+(averageTurnAroundTime/size));

                Log.printLine();
                Log.printLine("Cloudlet ID" + indent + "STATUS" + indent +
                                "Data center ID" + indent + "VM ID" + indent + indent + "Time"
+ indent + "Start Time" + indent + "Finish Time" +indent+ "User ID" +indent+ "Waiting Time"
+indent+ indent + "Turn Around 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");
                                Log.printLine( indent + indent + indent+
cloudlet.getResourceId() + indent + indent + indent + cloudlet.getVmId() +
                                                indent + indent + indent +
dft.format(cloudlet.getActualCPUTime()) +
                                                indent + indent  +
dft.format(cloudlet.getExecStartTime())+ indent + indent  +
dft.format(cloudlet.getFinishTime())+indent+indent + indent +cloudlet.getUserId() + indent
+ indent + indent + waitingTime[i] + indent + indent + indent + turnAroundTime[i]);
                        }
                }
        }
}
```

Result:

Therefore, the simulation of various cloud scenarios has been executed with cloudsim successfully.