VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELAGAVI 560018

Report on

"TASK SCHEDULING WITH DEADLINES"

By

Dhruv Agrawal (1BM18CS028) Harsha R(1BM18CS034)

Abhishikat Kumar Soni (1BM18CS004) Arbaz Ahmed(1BM19CS401)

Under the Guidance of

Dr. Pallavi Gb

Assistant Professor, Department of CSE

B.M.S College of Engineering

Work carried out at

Department of Computer Science and Engineering

B.M.S College of Engineering

(Autonomous college under VTU)

P.O. Box No.: 1908, Bull Temple Road, Bangalore-560 019

2020-2021

B.M.S COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Cloud Computing Self Study titled "TASK SCHEDULING WITH DEADLINES" has been carried out successfully by Dhruv Agrawal (1BM18CS028), Harsha R(1BM18CS034), Abhishikat Kumar Soni (1BM18CS004), Arbaz Ahmed(1BM19CS401) during the academic year 2020-2021.

Signature of the guide:

Dr. Pallavi GB

Assistant Professor

Department of Computer Science and Engineering

B.M.S College of Engineering, Bangalore-560019

B.M.S COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

We, Dhruv Agrawal (1BM18CS028), Harsha R(1BM18CS034), Abhishikat Kumar Soni (1BM18CS004), Arbaz Ahmed(1BM19CS401), students of 7th Semester, B.E, Department of Computer Science and Engineering, B.M.S College of Engineering, Bangalore, hereby declare that, this self-study work entitled "TASK SCHEDULING WITH DEADLINES" has been carried out by us under the guidance of Dr. Pallavi GB, Assistant Professor, Department of CSE, B.M.S College of Engineering, Bangalore during the academic semester Oct-Jan 2022. We also declare that to the best of our knowledge and belief, the assignment reported here is not from part of any other report by any other students.

Signature of the Candidates

Dhruv Agrawal (1BM18CS028)

Harsha R(1BM18CS034)

Abhishikat Kumar Soni (1BM18CS004)

Arbaz Ahmed(1BM19CS401)

List of Contents

1. INTRODUCTION TO CLOUD COMPUTING

Cloud computing is the delivery of computing services including servers, storage, databases, networking, software, analytics, and intelligence over the Internet to offer faster innovation, flexible resources, and economies of scale. We can typically pay only for cloud services we use, helping lower our operating costs, run our infrastructure more efficiently and scale as our business needs change.

a. Benefits of using cloud computing

Cost - Cloud computing eliminates the capital expense of buying hardware and software and setting up and running on-site data centers, the racks of servers, the round-the-clock electricity for power and cooling, the IT experts for managing the infrastructure. It adds up fast.

Global Scale - The benefits of cloud computing services include the ability to scale elastically. In cloud speak, that means delivering the right amount of IT resources, for example, more or less computing power, storage, bandwidth, right when it is needed and from the right geographic location.

Performance - The biggest cloud computing services run on a worldwide network of secure data centers, which are regularly upgraded to the latest generation of fast and efficient computing hardware. This offers several benefits over a single corporate data center, including reduced network latency for applications and greater economies of scale.

Security - Many cloud providers offer a broad set of policies, technologies, and controls that strengthen your security posture overall, helping protect your data, apps, and infrastructure from potential threats.

Speed - Most cloud computing services are self-service and on-demand, so even vast amounts of computing resources can be provisioned in minutes, typically with just a few mouse clicks, giving businesses a lot of flexibility and taking the pressure off capacity planning.

Productivity - On-site datacenters typically require a lot of “racking and stacking”, hardware setup, software patching, and other time-consuming IT management chores. Cloud computing removes the need for many of these tasks, so IT teams can spend time on achieving more important business goals.

Reliability - Cloud computing makes data backup, disaster recovery and business continuity easier and less expensive because data can be mirrored at multiple redundant sites on the cloud provider’s network.

Security - Many cloud providers offer a broad set of policies, technologies and controls that strengthen your security posture overall, helping protect your data, apps and infrastructure from potential threats.

b. Types of cloud computing

Not all clouds are the same and not one type of cloud computing is right for everyone. Several different models, types and services have evolved to help offer the right solution for your needs. First, you need to determine the type of cloud deployment or cloud computing architecture that your cloud services will be implemented on.

There are three different ways to deploy cloud services:

Public Cloud - Public clouds are owned and operated by a third-party cloud service provider, which delivers their computing resources like servers and storage over the Internet. Microsoft Azure is an example of a public cloud. With a public cloud, all hardware, software, and other supporting infrastructure is owned and managed by the cloud provider. We can access these services and manage your account using a web browser.

Private Cloud - A private cloud refers to cloud computing resources used exclusively by a single business or organization. A private cloud can be physically located on the company’s on-site data center. Some companies also pay third-party service providers to host their private cloud. A private cloud is one in which the services and infrastructure are maintained on a private network.

Hybrid Cloud - Hybrid clouds combine public and private clouds, bound together by technology that allows data and applications to be shared between them. By allowing data and applications to move between private and public clouds, a hybrid cloud gives your business greater flexibility, more deployment options and helps optimize your existing infrastructure, security and compliance.

Community Cloud - Community clouds is a cloud where services and infrastructure are accessible by a group of organizations.

There are three cloud services -

IaaS - The most basic category of cloud computing services. With IaaS, you rent IT infrastructure—servers and virtual machines (VMs), storage, networks, operating systems—from a cloud provider on a pay-as-you-go basis.

PaaS - Platform as a service refers to cloud computing services that supply an on-demand environment for developing, testing, delivering, and managing software applications. PaaS is designed to make it easier for developers to quickly create web or mobile apps, without worrying about setting up or managing the underlying infrastructure of servers, storage, network, and databases needed for development.

SaaS - Software as a service is a method for delivering software applications over the Internet, on-demand, and typically on a subscription basis. With SaaS, cloud providers host and manage the software application and underlying infrastructure and handle any maintenance, like software upgrades and security patching. Users connect to the application over the Internet, usually with a web browser on their phone, tablet or PC.

Uses of cloud computing

- To create cloud-native applications.
- To test and build applications.
- To store, back up, and recover data.
- To analyze data.
- To stream audio and video.
- To deliver software on demand

2. INTRODUCTION TO CLOUDSIM

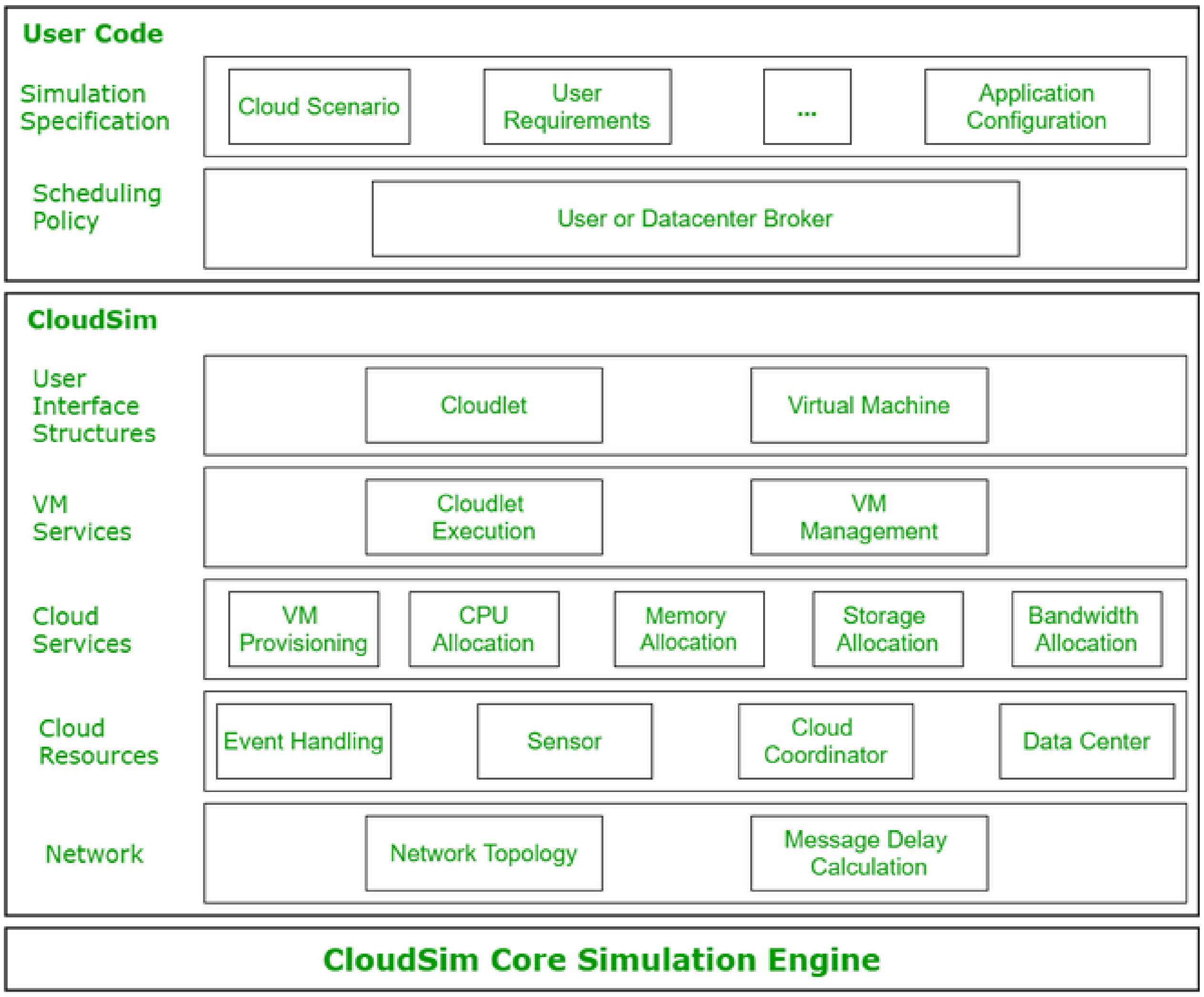
Overview of CloudSim -

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 modeling and simulating a cloud computing environment as a means for evaluating a hypothesis prior to software development in order to reproduce tests and results. For example, if you were to deploy an application or a website on the cloud and wanted to test the services and load that your product can handle and also tune its performance to overcome bottlenecks before risking deployment, then such evaluations could be performed by simply coding a simulation of that environment with the help of various flexible and scalable classes provided by the CloudSim package, free of cost.

Following are the benefits of CloudSim:

- No capital investment involved. With a simulation tool like CloudSim, there is no installation or maintenance cost.
- Easy to use and Scalable. You can change the requirements such as adding or deleting resources by changing just a few lines of code.
- Risks can be evaluated at an earlier stage. In Cloud Computing utilization of real testbeds limits the experiments to the scale of the testbed and makes the reproduction of results an extremely difficult undertaking. With simulation, you can test your product against test cases and resolve issues before actual deployment without any limitations.
- No need for try-and-error approaches. Instead of relying on theoretical and imprecise evaluations which can lead to inefficient service performance and revenue generation, you can test your services in a repeatable and controlled environment free of cost with CloudSim.

The architecture of cloudSim



CloudSim Core Simulation Engine provides interfaces for the management of resources such as VM, memory and bandwidth of virtualized Datacenters.

CloudSim layer manages the creation and execution of core entities such as VMs, Cloudlets, Hosts, etc. It also handles network-related execution along with the provisioning of resources and their execution and management.

User Code is the layer controlled by the user. The developer can write the requirements of the hardware specifications in this layer according to the scenario.

Some of the most common classes used during simulation are

- Datacenter: used for modeling the foundational hardware equipment of any cloud environment, that is the Datacenter. This class provides methods to specify the functional requirements of the Datacenter as well as methods to set the allocation policies of the VMs etc.
- Host: this class executes actions related to the management of virtual machines. It also defines policies for provisioning memory and bandwidth to the virtual machines, as well as allocating CPU cores to the virtual machines.
- VM: this class represents a virtual machine by providing data members defining a VM's bandwidth, RAM, MIPS (million instructions per second), size while also providing setter and getter methods for these parameters.
- Cloudlet: a cloudlet class represents any task that is run on a VM, like a processing task, or a memory access task, or a file updating task, etc. It stores parameters defining the characteristics of a task such as its length, size, MIPS (million instructions) and provides methods similar to VM class while also providing methods that define a task's execution time, status, cost, and history.
- DatacenterBroker: is an entity acting on behalf of the user/customer. It is responsible for the functioning of VMs, including VM creation, management, destruction, and submission of cloudlets to the VM.
- CloudSim: this is the class responsible for initializing and starting the simulation environment after all the necessary cloud entities have been defined and later stopping after all the entities have been destroyed.

Features of cloudSim

CloudSim provides support for simulation and modeling of:

- Large scale virtualized Datacenters, servers, and hosts.
- Customizable policies for provisioning hosts to virtual machines.
- Energy-aware computational resources.
- Application containers and federated clouds (joining and management of multiple public clouds).
- Datacenter network topologies and message-passing applications.
- Dynamic insertion of simulation entities with stop and resume of simulation.
- User-defined allocation and provisioning policies.

SpaceShared vs TimeShared

Space-sharing refers to sharing memory space (hard disk, RAM, database) by many different users (such as in-place algorithm, executing threads). A space-sharing algorithm would achieve its goal by re-using the space (storage) allocated for the input data structure, in other words, it may not allocate any additional space during its execution. In Space shared VM Scheduling, PE's are explicitly allocated to a VM. It means if a PE is allocated to a VM, it cannot be accessed by another VM until the current VM releases it.

Time-sharing refers to the concept of sharing executing power (such as CPU, logical processor, GPU) by many users (such as OS processes, threads, network requests).In Time shared VM Scheduling, VM's share the PE's. If a PE is capable to fulfill the requirements of multiple VM's simultaneously, then in the case of Time shared VM Scheduling, this can be allowed and multiple VM's will execute in parallel over the same PE or set of PE's

3. PERFORMANCE PARAMETERS

Response Time, Average Response Time, and Formula

Response time is the time spent when the task is in the ready state and gets the resources for the first time.

Response time = Time at which the task gets the resources for the first time - Arrival time

Average response time = Total response time / No. of tasks

Waiting Time, Average Waiting Time and Formula

Waiting time is the total time spent by the task in the ready state waiting for the resources.

Waiting time = Turnaround time - Burst time

Average waiting time = Total wait time/No. of tasks

Execution Time and Formula

Time at which process completes its execution. The execution time or CPU time of a given task is defined as the time spent by the system executing that task, including the time spent executing run-time or system services on its behalf. It is implementation-defined which task, if any, is charged the execution time that is consumed by interrupt handlers and run-time services on behalf of the system.

Turn Around Time and Formula

It is the time Difference between completion time and arrival time.

The formula for Turnaround Time = Execution Time – Arrival Time

Resource Utilization and Formula

Resource utilization tells us how efficient the VM is in scheduling tasks. It can be defined as the number of tasks executed by the VM in a given amount of time.

Resource utilization = Total time taken by each task to get executed/No. of tasks

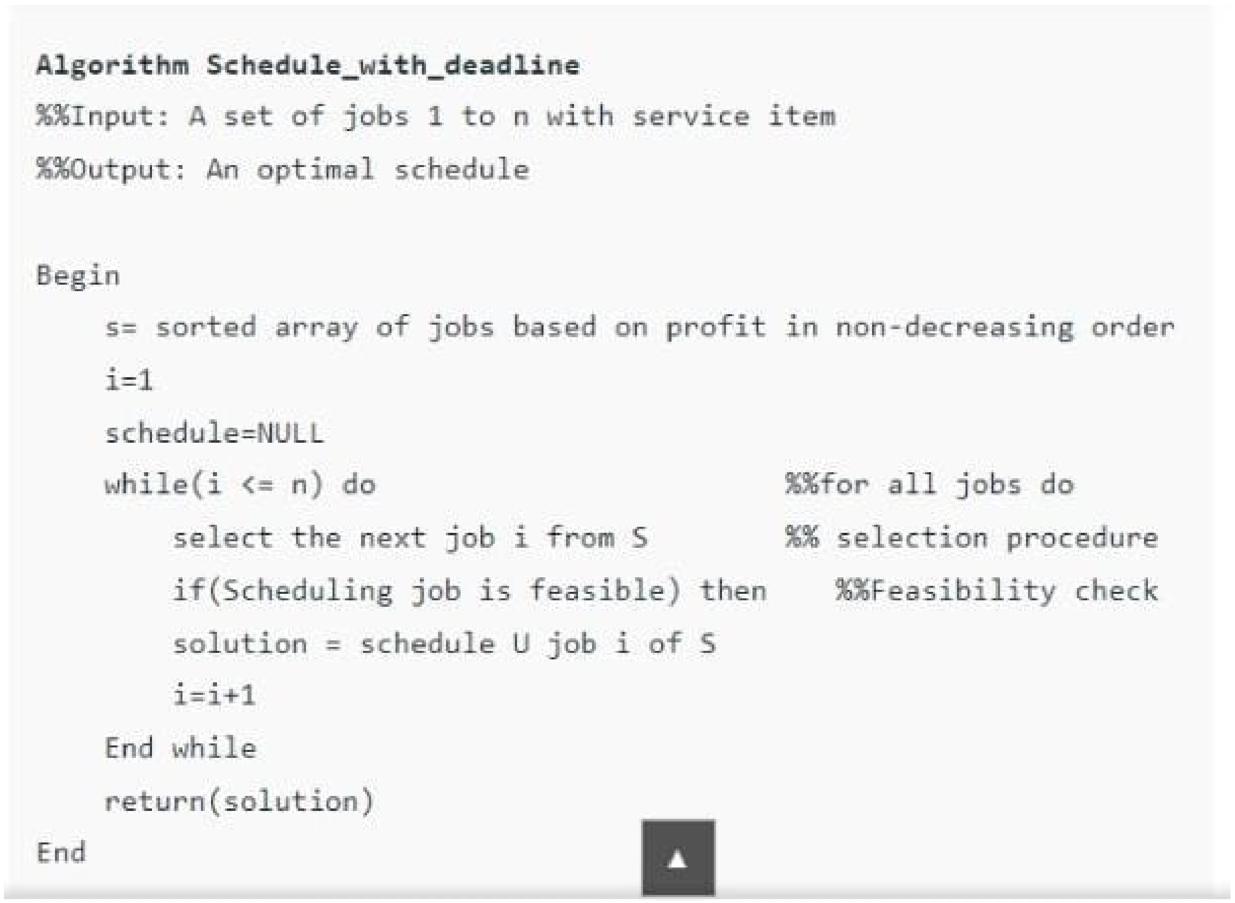
4. ALGORITHM

The goal of a Scheduling problem is to schedule the tasks such that the maximum total profit is obtained.

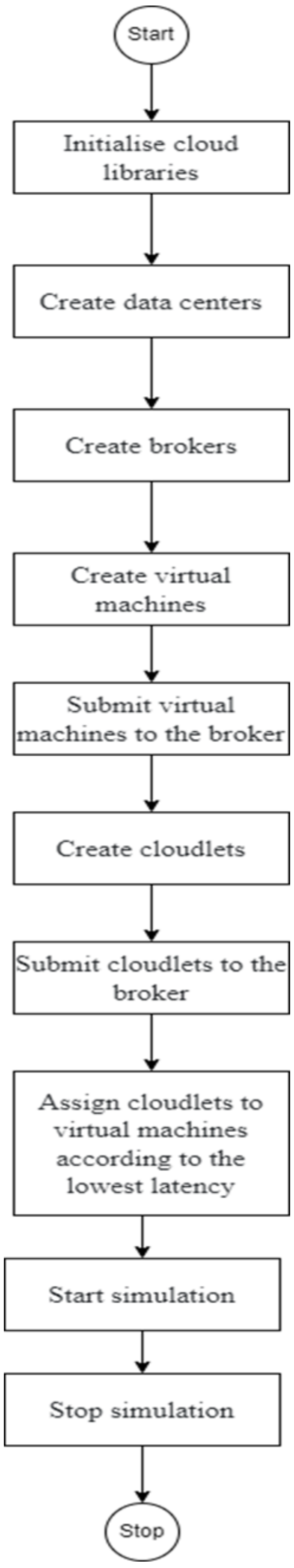
This algorithm for scheduling with a deadline is different from scheduling without a deadline because task completion here is associated with profit.

The objective of this problem is to construct a feasible sequence that gives the maximum profit.

The optimal sequence is found.



```
Algorithm Schedule_with_deadline
%%Input: A set of jobs 1 to n with service item
%%Output: An optimal schedule

Begin
    s= sorted array of jobs based on profit in non-decreasing order
    i=1
    schedule=NULL
    while(i <= n) do                        %%for all jobs do
        select the next job i from S        %% selection procedure
        if(Scheduling job is feasible) then    %%Feasibility check
        solution = schedule U job i of S
        i=i+1
    End while
    return(solution)
End
```


Start
Initialise cloud libraries
Create data centers
Create brokers
Create virtual machines
Submit virtual machines to the broker
Create cloudlets
Submit cloudlets to the broker
Assign cloudlets to virtual machines according to the lowest latency
Start simulation
Stop simulation
Stop

5. CODE

```java
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.Collections;
import java.util.Comparator;
import java.util.LinkedList;
import java.util.List;
import java.util.Random;

import org.cloudbus.cloudsim.Cloudlet;
import org.cloudbus.cloudsim.CloudletSchedulerSpaceShared;
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 one
 * cloudlet on it.
 */
public class TaskSchedulingWithDeadlines {

	/** The cloudlet list. */
	private static float timeSlice = (float)8;
	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 deadline = 1000;
                long profit = 1000;
                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();
                        Random r1= new Random();

                        cloudlet[i]  =  new  Cloudlet(i,  length  +r.nextInt(2000),deadline  +
r.nextInt(2000),profit + 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;
	}


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

	/**
	 * Creates main() to run this example
	 */
	public static void main(String[] args) {
		Log.printLine("Starting CloudSimExample6...");

		try {
			// 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);

			// Second step: Create Datacenters
			//Datacenters are the resource providers in CloudSim. We need at least
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 VMs and Cloudlets and send them to broker
			vmlist = createVM(brokerId,6); //creating 10 vms
			cloudletList = createCloudlet(brokerId,10); // creating 40 cloudlets
			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();

			CloudSim.stopSimulation();

			printCloudletList(newList);

			//Print the debt of each user to each datacenter
//			datacenter0.printDebts();
//			datacenter1.printDebts();

			Log.printLine("CloudSimExample6 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 = 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


		//To create a host with a space-shared allocation policy for PEs to VMs:
		//hostList.add(
	//		new Host(
	//			hostId,
	//			new CpuProvisionerSimple(peList1),
	//			new RamProvisionerSimple(ram),
	//			new BwProvisionerSimple(bw),
	//			storage,
	//			new VmSchedulerSpaceShared(peList1)
	//		)
	//	);

		//To create a host with a oportunistic space-shared allocation policy for PEs
to VMs:
		//hostList.add(
	//		new Host(
	//			hostId,
	//			new CpuProvisionerSimple(peList1),
	//			new RamProvisionerSimple(ram),
	//			new BwProvisionerSimple(bw),
	//			storage,
	//			new VmSchedulerOportunisticSpaceShared(peList1)
	//		)
	//	);
```

```
            // 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(){


		DatacenterBroker broker = null;
		try {
			broker = new DatacenterBroker("Broker");
		} catch (Exception e) {
			e.printStackTrace();
			return null;
		}
		return broker;
	}

	private static void printProfit(List<Cloudlet> arr, int t) {
		int size = arr.size();
		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];
		Cloudlet cloudlet;
		Log.print("WIT PROFIT:::::::::::");
		int n = arr.size();
		   for (Cloudlet jb : arr)
		  {
		      System.out.print("rr1 : "+ jb + " ");
		  }
	  Collections.sort(arr,
				new Comparator<Cloudlet>() {
							@Override
							public int compare(Cloudlet a, Cloudlet b) {
								System.out.println("PROFIT:
"+a.getCloudProfit());
								return   (int)   (b.getCloudProfit()   -
a.getCloudProfit());
```

```
                                    }
                                });
for (Cloudlet jb : arr)
{
    System.out.print("rr : "+ jb + " ");
}

boolean result[] = new boolean[t];
int job[] = new int[t];
int sum1 = 0;
for (int i = 0; i < n; i++)
{
    for (int j
         = (int) Math.min(t - 1, arr.get(i).getCloudDeadline() - 1);
         j >= 0; j--) {
        if (result[j] == false)
        {
            result[j] = true;
            System.out.println(arr.get(i).getCloudletId());
            job[j] = arr.get(i).getCloudletId();
            sum1 += arr.get(i).getCloudProfit();
            break;
        }
    }
}
for (int jb : job)
{
    System.out.print(jb + " ");
}
System.out.println();
System.out.println("Total Profit: " + sum1);


// printing

String indent = "    ";
     DecimalFormat dft = new DecimalFormat("###.##");
```

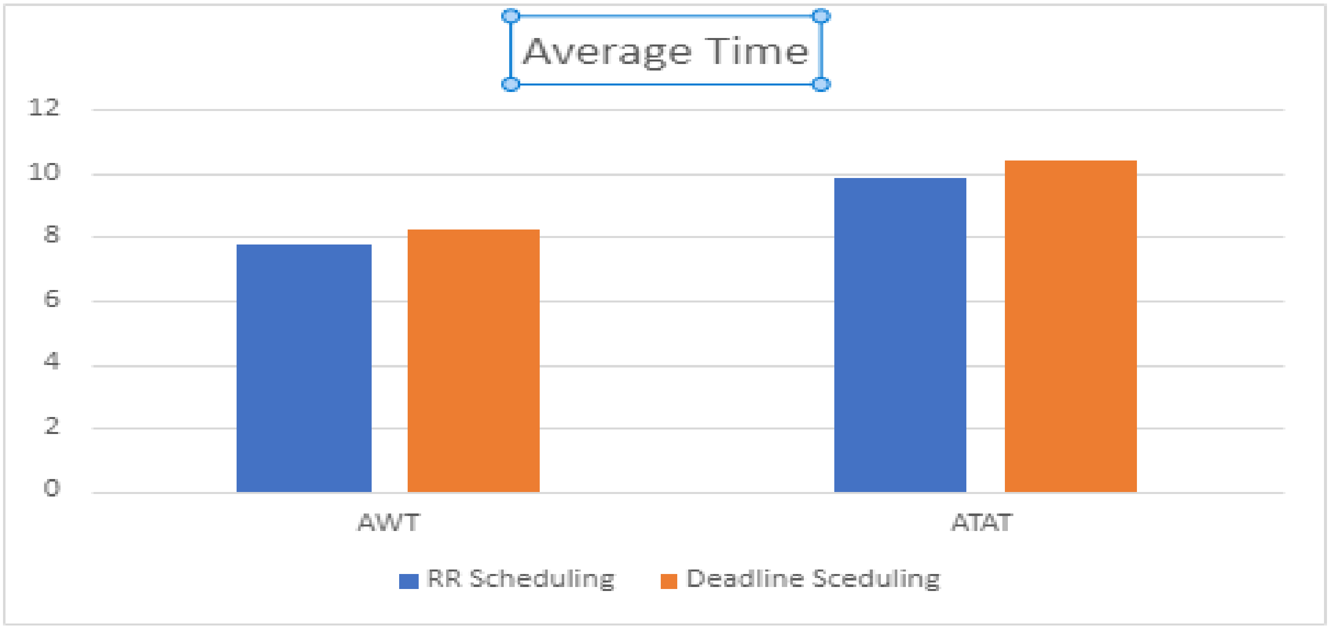
```
        for(int i = 0; i<job.length; i++) {
            cloudlet = arr.get(i);
            //We get the cpu time for each cloudlet
            String cpuTime = dft.format(cloudlet.getActualCPUTime());
            float convertedCPUTime = (float) Double.parseDouble(cpuTime);
            burstTime[i]  =  convertedCPUTime;  //burst  time  is  equal  to  execution
time.
        }
        for(int i=0; i<job.length; i++) {
            a[i] = burstTime[i];
        }
        for(int i=0; i<job.length; i++) {
            waitingTime[i] = 0;
        }
        do {
            for(int i=0; i<job.length; i++) {
                if(burstTime[i]>timeSlice) {
                    burstTime[i] -= timeSlice;
                    for(int j=0; j<job.length; j++) {
                        if((j != i) && (burstTime[j] != 0)) {
                            waitingTime[j] += timeSlice;
                        }
                    }
                }
                else {
                    for(int j=0; j<job.length; j++) {
                        if((j != i) && (burstTime[j] != 0)) {
                            waitingTime[j] += burstTime[i];
                        }
                    }
                    burstTime[i] = 0;
                }
            }
            sum = 0;
            for(int k=0; k<job.length; k++) {
                sum += burstTime[k];
            }
            }while(sum != 0);
            for(int i=0; i<job.length; 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<job.length; i++) {
                  cloudlet = arr.get(i);
                  pes = arr.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<job.length; j++) {
                  averageWaitingTime += waitingTime[j];
            }
            for(int j=0; j<job.length; j++) {
                  averageTurnAroundTime += turnAroundTime[j];
            }
            System.out.println("Average Waiting Time on Total: "
+(averageWaitingTime/size)+ "\nAverage Turn Around Time on Total: "
+(averageTurnAroundTime/size));


      }
      /**
       * Prints the Cloudlet objects
       * @param list  list of Cloudlets
       */
      @SuppressWarnings("deprecation")
```

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

        int size = list.size();
        printProfit(list, 5);
        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;
                        }
                    }
```

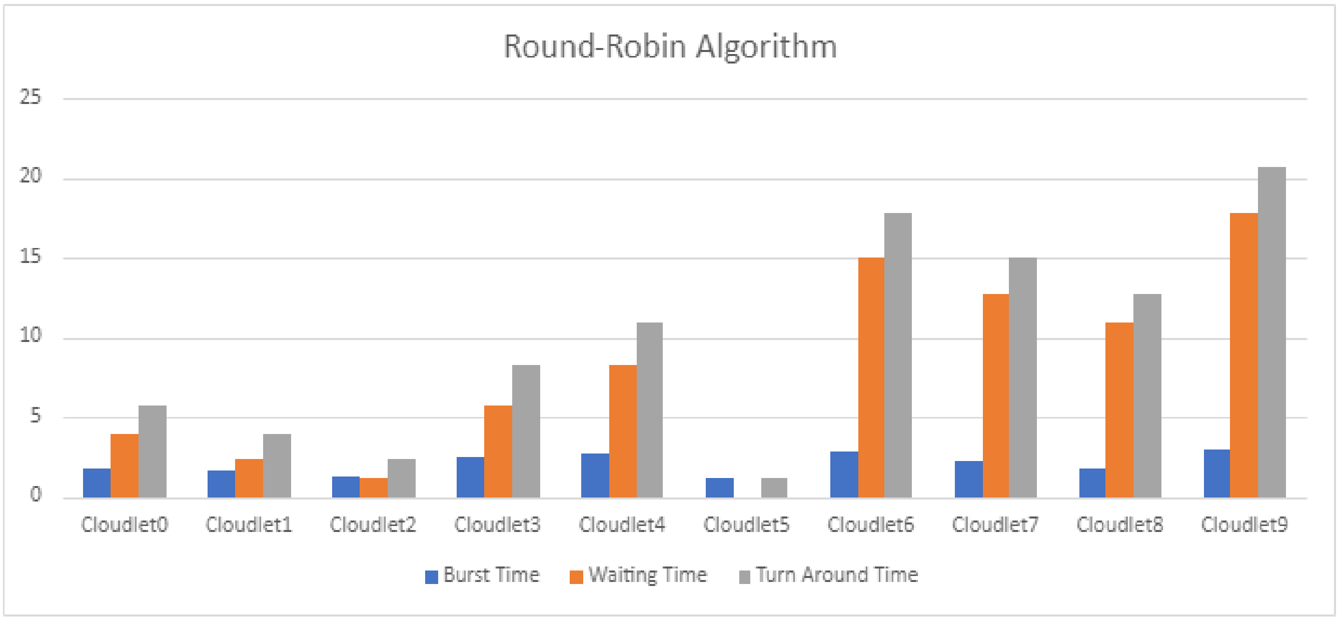
```
                        }
                        else {
                              for(int j=0; j<size; j++) {
                                    if((j != i) && (burstTime[j] != 0)) {
                                          waitingTime[j] += burstTime[i];
                                    }
                              }
                              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));

            String indent1 = "    ";
            Log.printLine();
            Log.printLine("========== OUTPUT ==========");
            Log.printLine("Cloudlet ID" + indent1 + "STATUS" + indent1 +
                    "Data center ID" + indent1 + "VM ID" + indent1 + indent1 + "Time"
+ indent1 + "Start Time" + indent1 + "Finish Time" +indent1+"user id"+indent1);

//          DecimalFormat dft = new DecimalFormat("###.##");
            for (int i = 0; i < size; i++) {
                cloudlet = list.get(i);
                Log.print(indent1 + cloudlet.getCloudletId() + indent1 + indent1);

                if (cloudlet.getCloudletStatus() == Cloudlet.SUCCESS){
                    Log.print("SUCCESS");

                    Log.printLine(  indent1  +  indent1  +  cloudlet.getResourceId()  +
indent1 + indent1 + indent1 + cloudlet.getVmId() +
                                        indent1     +     indent1     +     indent1     +
dft.format(cloudlet.getActualCPUTime()) +
                                        indent1             +             indent1             +
dft.format(cloudlet.getExecStartTime())+    indent1    +    indent1    +    indent1    +
dft.format(cloudlet.getFinishTime())+indent1 +cloudlet.getUserId());


                }
            }


        }
}
```

6. RESULTS AND GRAPH

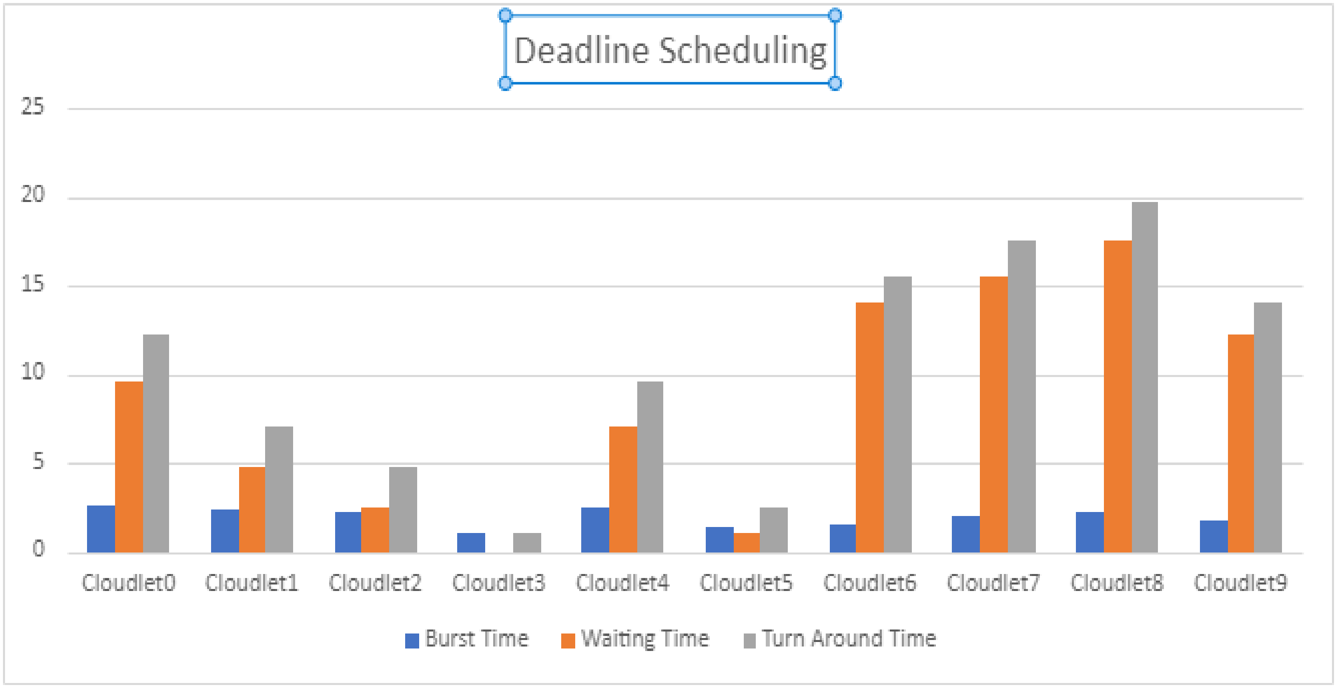
GRAPHS:



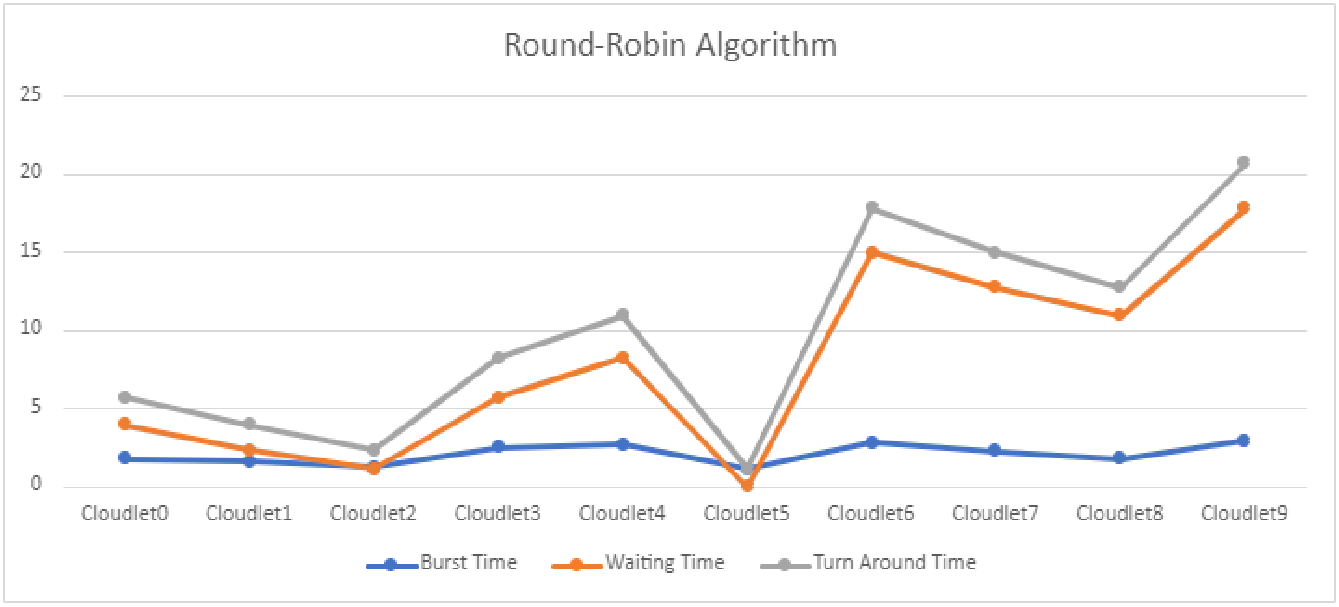
Graph-1



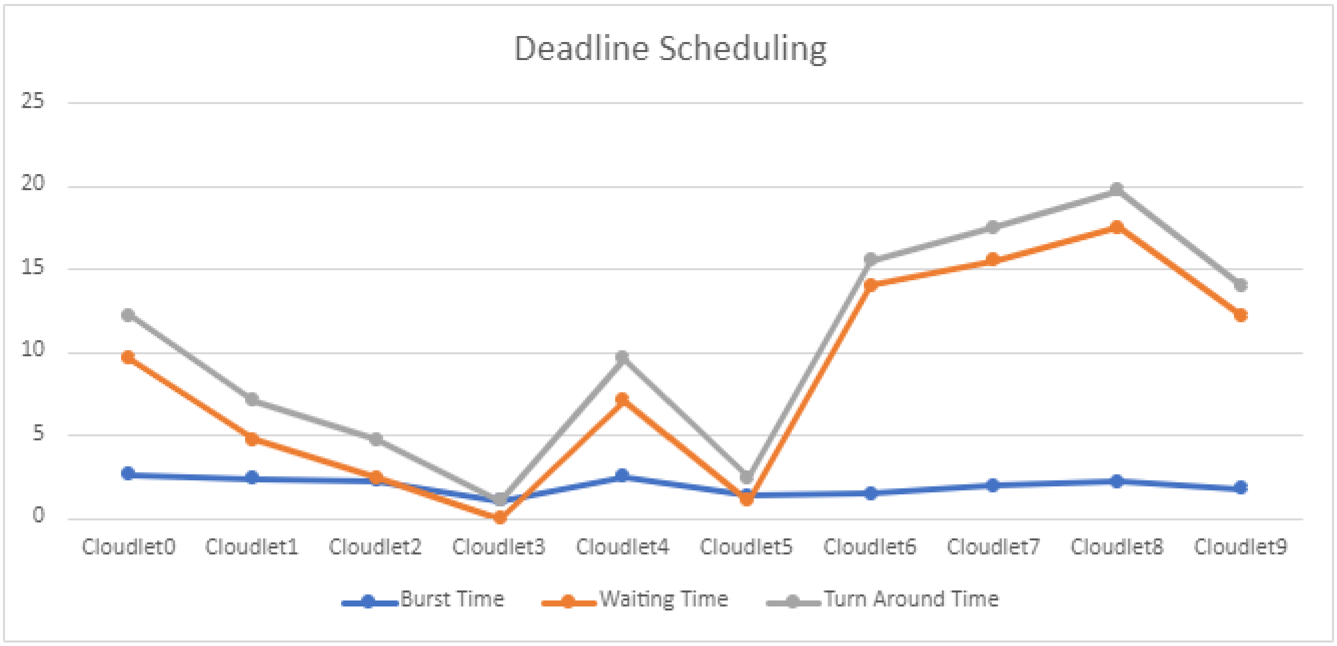
Graph-2



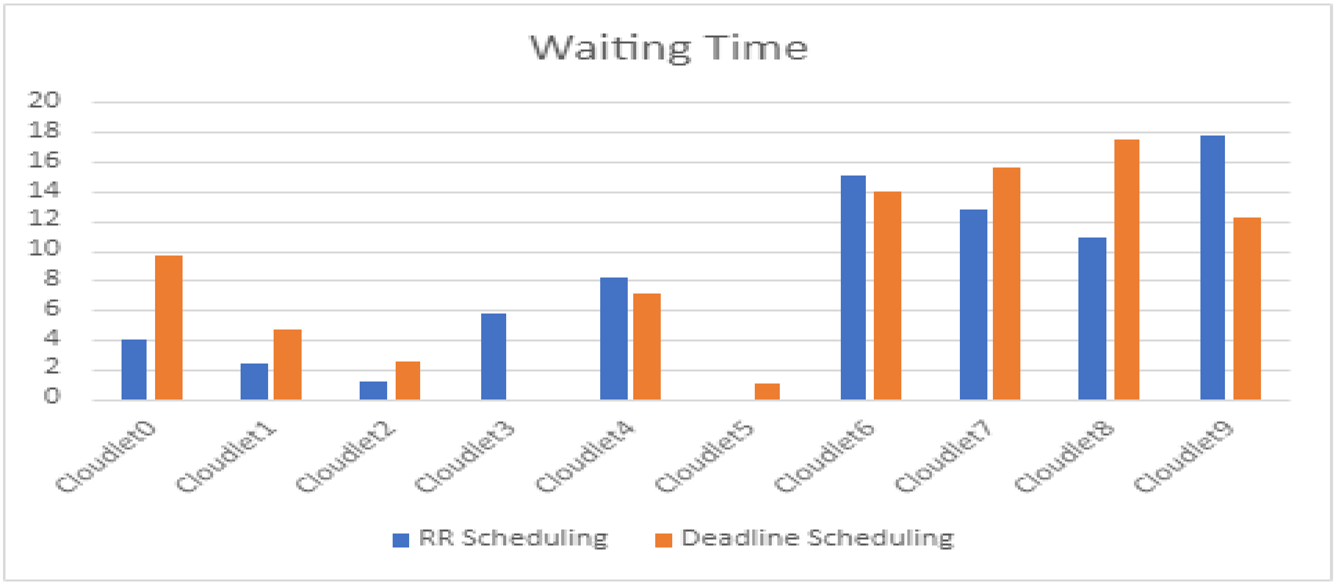
Graph-3



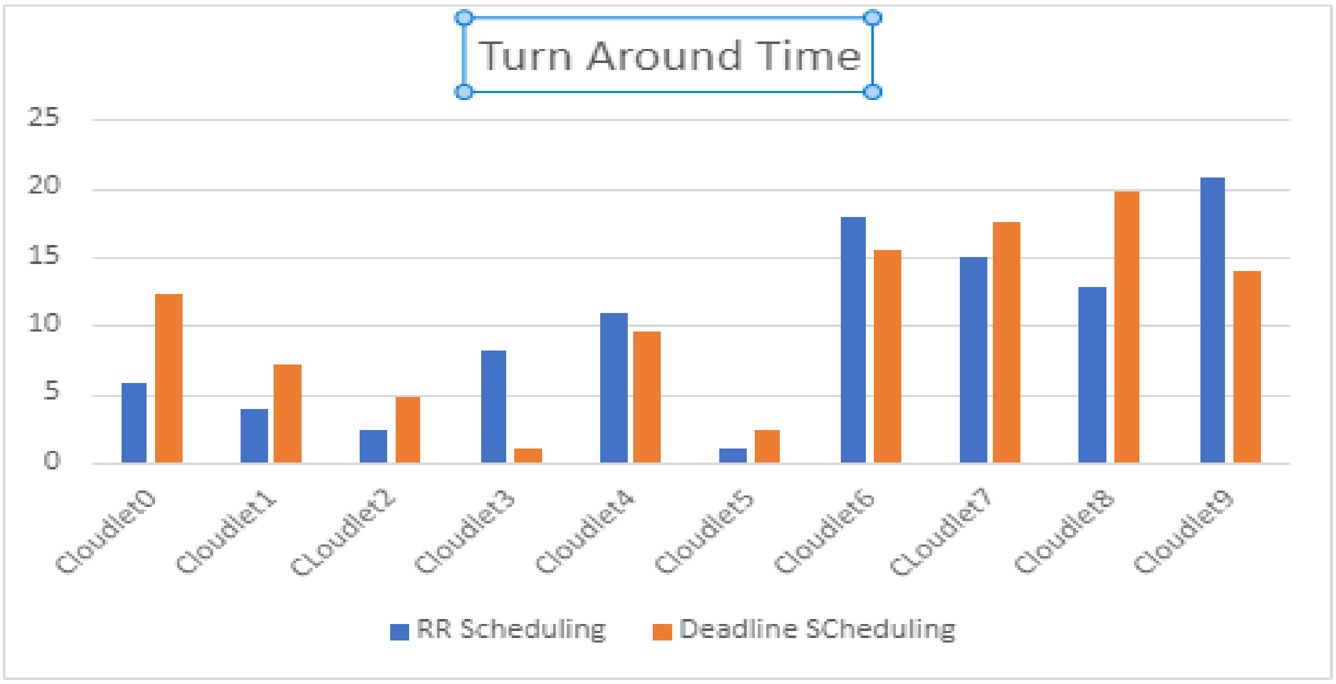
Graph-4



Graph-5

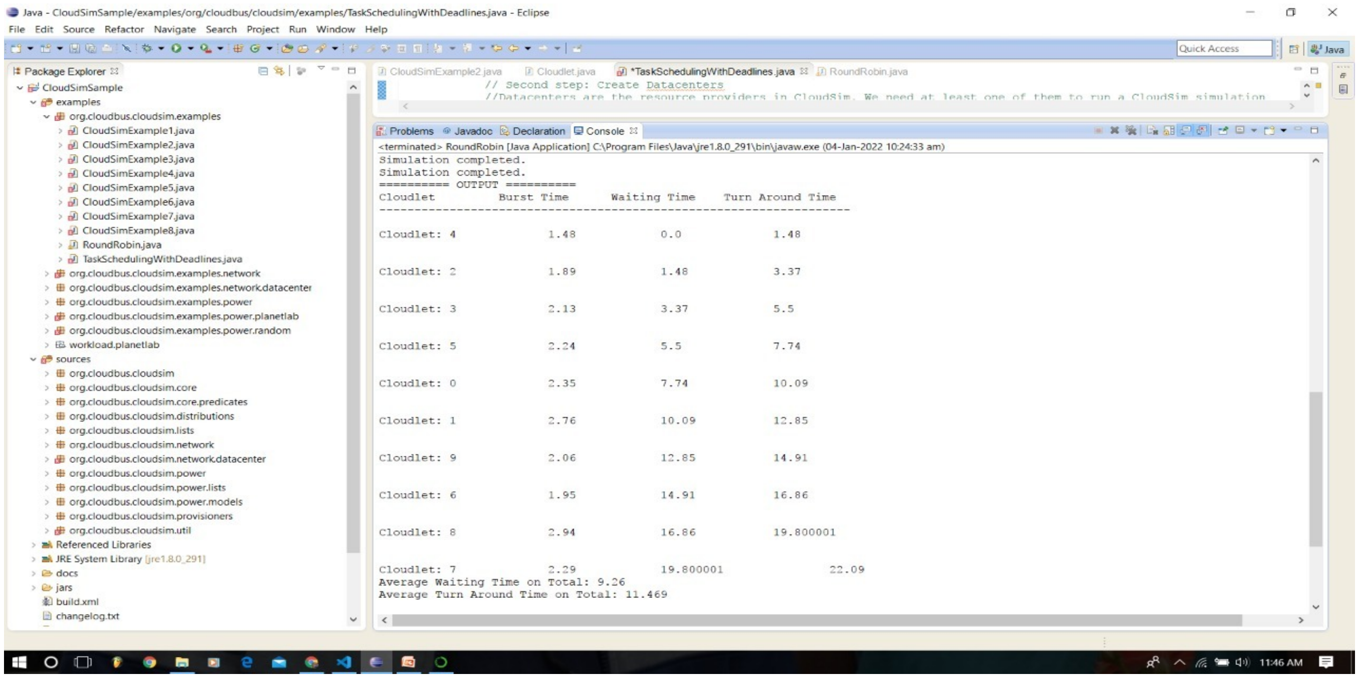


Graph-6


Turn Around Time
25
20
15
10
5
0
Cloudlet0
Cloudlet1
CLoudlet2
Cloudlet3
Cloudlet4
Cloudlet5
Cloudlet6
CLoudlet7
Cloudlet8
Cloudlet9
RR Scheduling
Deadline SCheduling

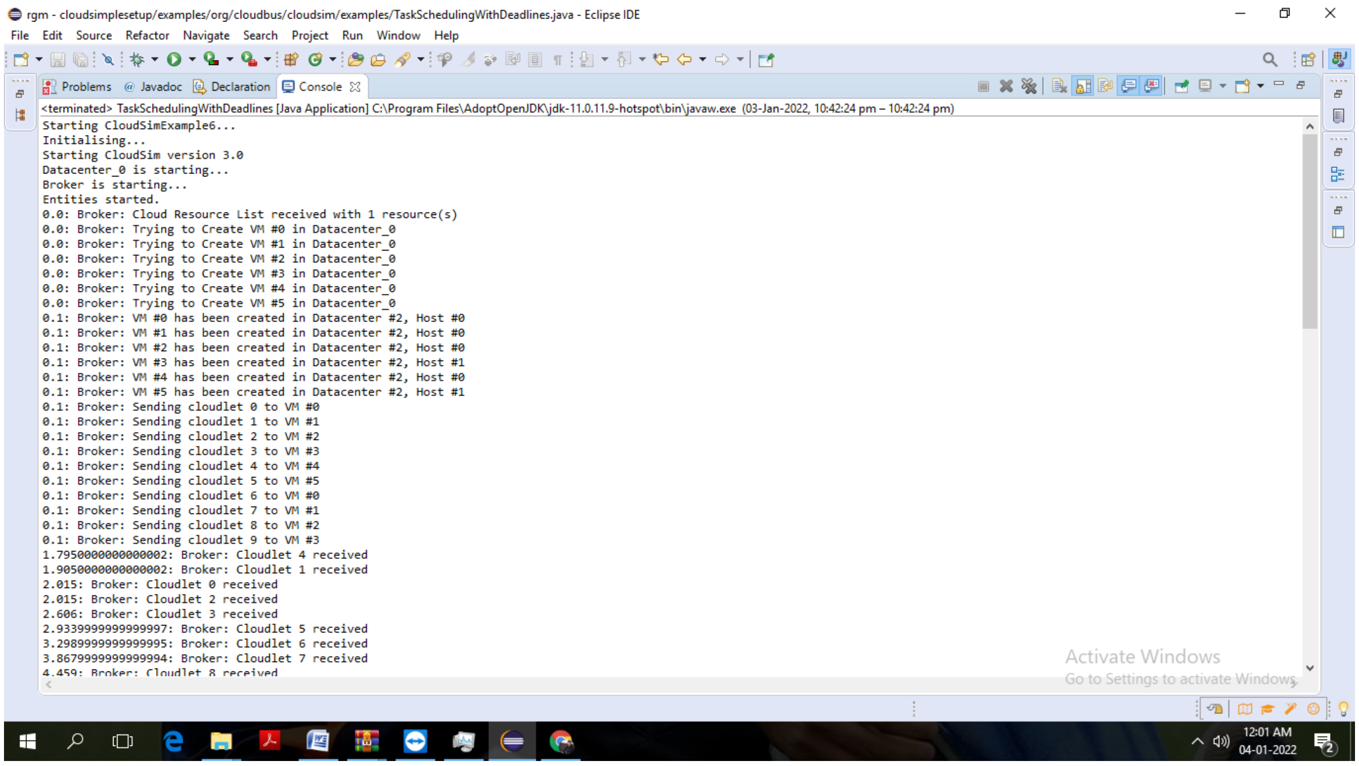
Graph-7

Output



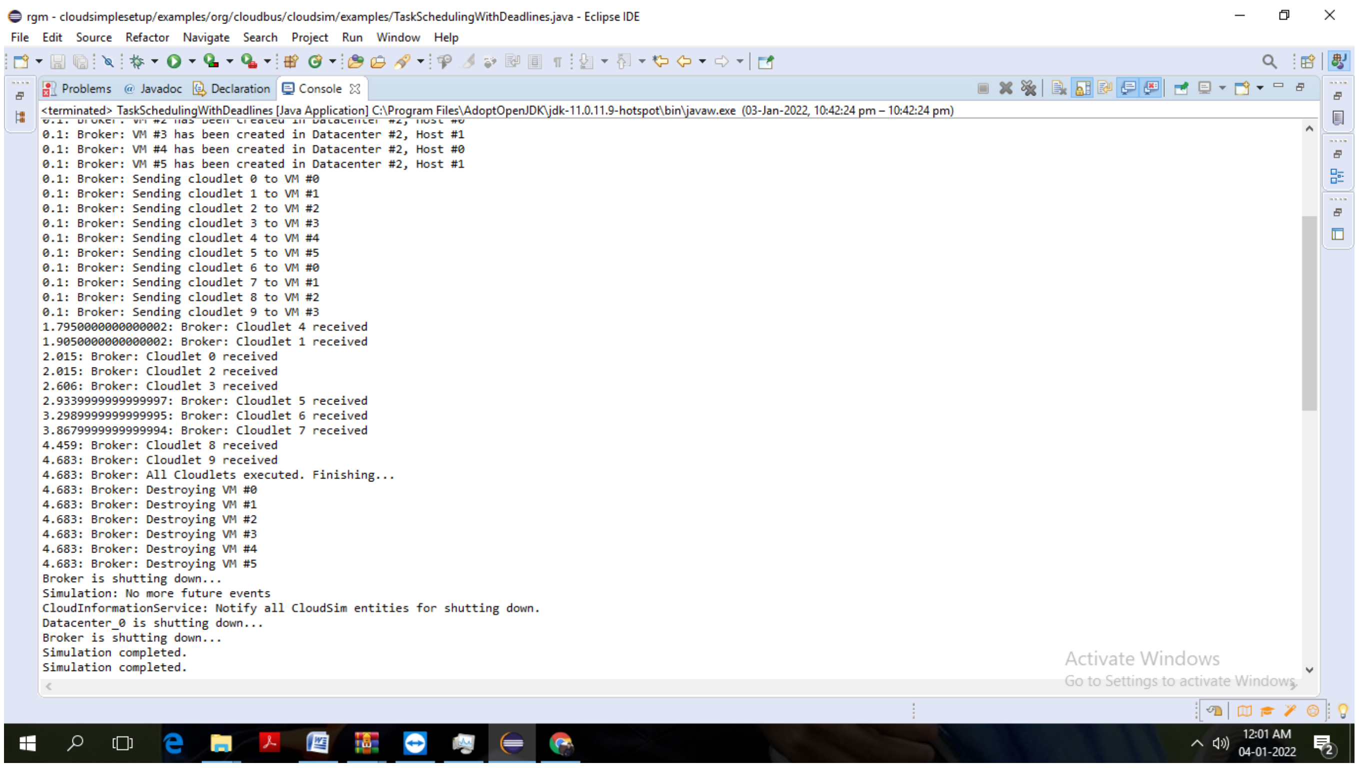
```
<terminated> RoundRobin [Java Application] C:\Program Files\Java\jre1.8.0_291\bin\javaw.exe (04-Jan-2022 10:24:33 am)
Simulation completed.
Simulation completed.
========== OUTPUT ==========
Cloudlet          Burst Time      Waiting Time     Turn Around Time
----------------------------------------------------------------------

Cloudlet: 4            1.48            0.0              1.48


Cloudlet: 2            1.89            1.48             3.37


Cloudlet: 3            2.13            3.37             5.5


Cloudlet: 5            2.24            5.5              7.74


Cloudlet: 0            2.35            7.74             10.09


Cloudlet: 1            2.76            10.09            12.85


Cloudlet: 9            2.06            12.85            14.91


Cloudlet: 6            1.95            14.91            16.86


Cloudlet: 8            2.94            16.86            19.800001


Cloudlet: 7            2.29            19.800001                22.09
Average Waiting Time on Total: 9.26
Average Turn Around Time on Total: 11.469
```

Fig-1



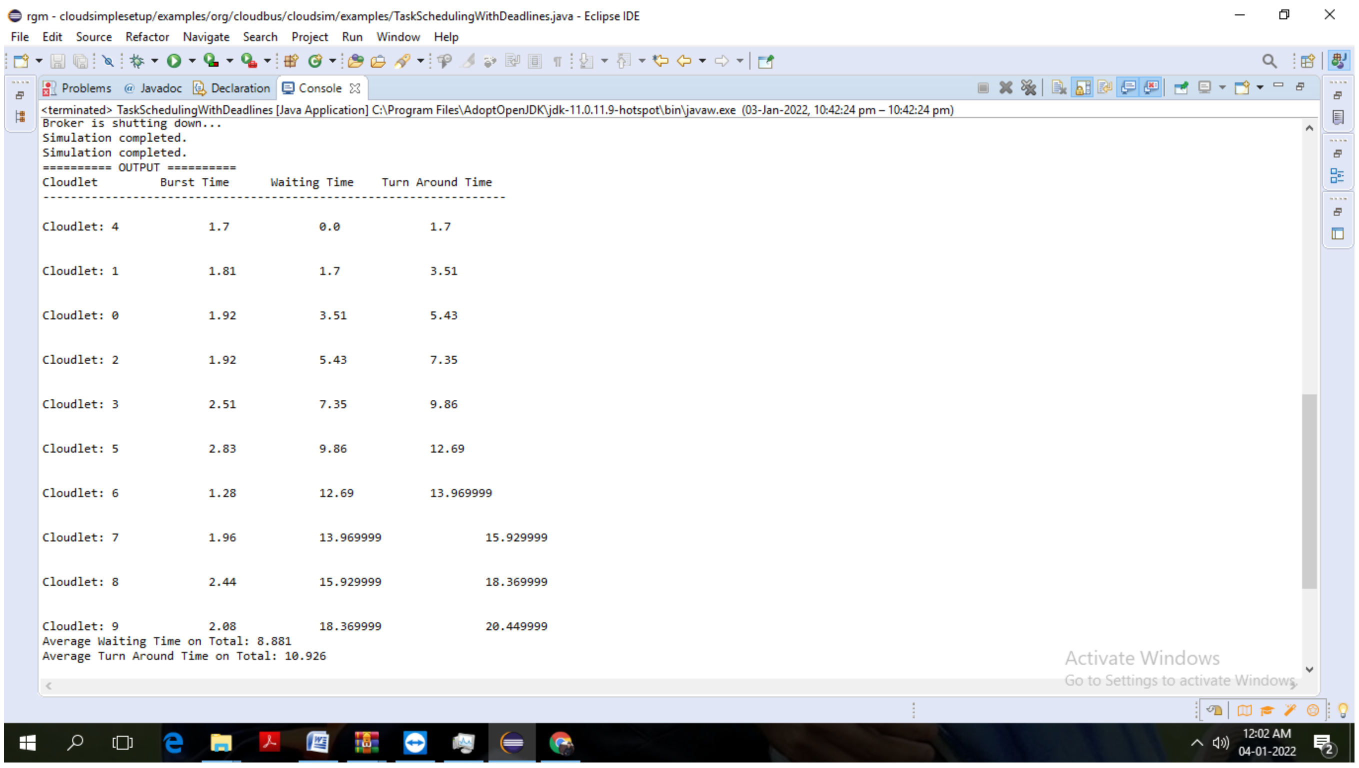
```
<terminated> TaskSchedulingWithDeadlines [Java Application] C:\Program Files\AdoptOpenJDK\jdk-11.0.11.9-hotspot\bin\javaw.exe (03-Jan-2022, 10:42:24 pm – 10:42:24 pm)
Starting CloudSimExample6...
Initialising...
Starting CloudSim version 3.0
Datacenter_0 is starting...
Broker is starting...
Entities started.
0.0: Broker: Cloud Resource List received with 1 resource(s)
0.0: Broker: Trying to Create VM #0 in Datacenter_0
0.0: Broker: Trying to Create VM #1 in Datacenter_0
0.0: Broker: Trying to Create VM #2 in Datacenter_0
0.0: Broker: Trying to Create VM #3 in Datacenter_0
0.0: Broker: Trying to Create VM #4 in Datacenter_0
0.0: Broker: Trying to Create VM #5 in Datacenter_0
0.1: Broker: VM #0 has been created in Datacenter #2, Host #0
0.1: Broker: VM #1 has been created in Datacenter #2, Host #0
0.1: Broker: VM #2 has been created in Datacenter #2, Host #0
0.1: Broker: VM #3 has been created in Datacenter #2, Host #1
0.1: Broker: VM #4 has been created in Datacenter #2, Host #0
0.1: Broker: VM #5 has been created in Datacenter #2, Host #1
0.1: Broker: Sending cloudlet 0 to VM #0
0.1: Broker: Sending cloudlet 1 to VM #1
0.1: Broker: Sending cloudlet 2 to VM #2
0.1: Broker: Sending cloudlet 3 to VM #3
0.1: Broker: Sending cloudlet 4 to VM #4
0.1: Broker: Sending cloudlet 5 to VM #5
0.1: Broker: Sending cloudlet 6 to VM #0
0.1: Broker: Sending cloudlet 7 to VM #1
0.1: Broker: Sending cloudlet 8 to VM #2
0.1: Broker: Sending cloudlet 9 to VM #3
1.7950000000000002: Broker: Cloudlet 4 received
1.9050000000000002: Broker: Cloudlet 1 received
2.015: Broker: Cloudlet 0 received
2.015: Broker: Cloudlet 2 received
2.606: Broker: Cloudlet 3 received
2.9339999999999997: Broker: Cloudlet 5 received
3.2989999999999995: Broker: Cloudlet 6 received
3.8679999999999994: Broker: Cloudlet 7 received
4.459: Broker: Cloudlet 8 received
```

Fig-2



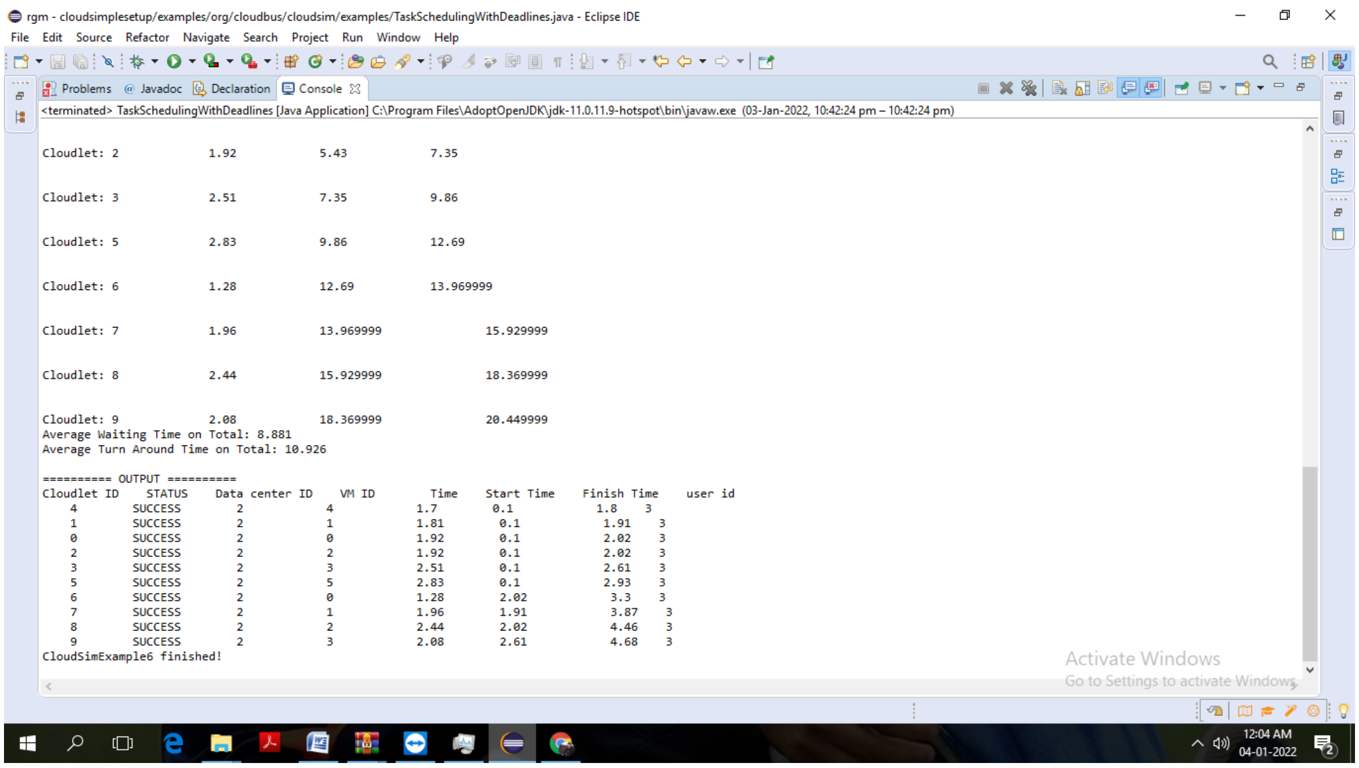
```
<terminated> TaskSchedulingWithDeadlines [Java Application] C:\Program Files\AdoptOpenJDK\jdk-11.0.11.9-hotspot\bin\javaw.exe  (03-Jan-2022, 10:42:24 pm – 10:42:24 pm)
0.1: Broker: VM #2 has been created in Datacenter #2, Host #0
0.1: Broker: VM #3 has been created in Datacenter #2, Host #1
0.1: Broker: VM #4 has been created in Datacenter #2, Host #0
0.1: Broker: VM #5 has been created in Datacenter #2, Host #1
0.1: Broker: Sending cloudlet 0 to VM #0
0.1: Broker: Sending cloudlet 1 to VM #1
0.1: Broker: Sending cloudlet 2 to VM #2
0.1: Broker: Sending cloudlet 3 to VM #3
0.1: Broker: Sending cloudlet 4 to VM #4
0.1: Broker: Sending cloudlet 5 to VM #5
0.1: Broker: Sending cloudlet 6 to VM #0
0.1: Broker: Sending cloudlet 7 to VM #1
0.1: Broker: Sending cloudlet 8 to VM #2
0.1: Broker: Sending cloudlet 9 to VM #3
1.7950000000000002: Broker: Cloudlet 4 received
1.9050000000000002: Broker: Cloudlet 1 received
2.015: Broker: Cloudlet 0 received
2.015: Broker: Cloudlet 2 received
2.606: Broker: Cloudlet 3 received
2.9339999999999997: Broker: Cloudlet 5 received
3.2989999999999995: Broker: Cloudlet 6 received
3.8679999999999994: Broker: Cloudlet 7 received
4.459: Broker: Cloudlet 8 received
4.683: Broker: Cloudlet 9 received
4.683: Broker: All Cloudlets executed. Finishing...
4.683: Broker: Destroying VM #0
4.683: Broker: Destroying VM #1
4.683: Broker: Destroying VM #2
4.683: Broker: Destroying VM #3
4.683: Broker: Destroying VM #4
4.683: Broker: Destroying VM #5
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.
```

Fig-3



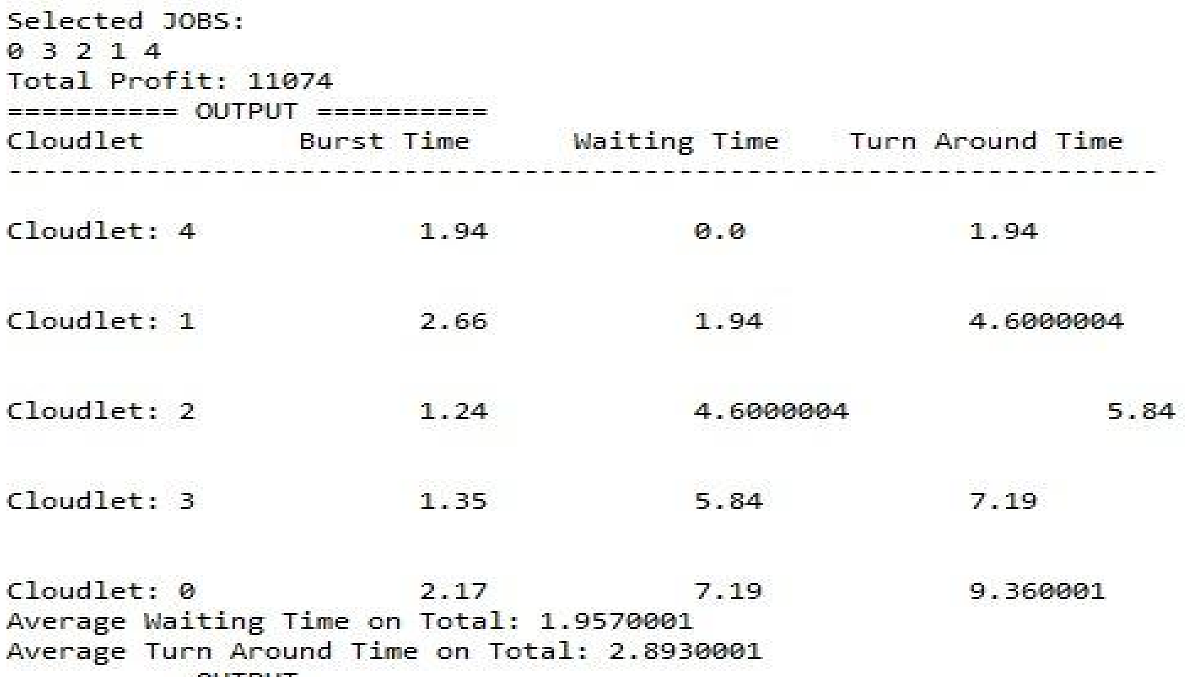
```
<terminated> TaskSchedulingWithDeadlines [Java Application] C:\Program Files\AdoptOpenJDK\jdk-11.0.11.9-hotspot\bin\javaw.exe  (03-Jan-2022, 10:42:24 pm – 10:42:24 pm)
Broker is shutting down...
Simulation completed.
Simulation completed.
========== OUTPUT ==========
Cloudlet        Burst Time      Waiting Time    Turn Around Time
--------------------------------------------------------------------

Cloudlet: 4             1.7             0.0             1.7


Cloudlet: 1             1.81            1.7             3.51


Cloudlet: 0             1.92            3.51            5.43


Cloudlet: 2             1.92            5.43            7.35


Cloudlet: 3             2.51            7.35            9.86


Cloudlet: 5             2.83            9.86            12.69


Cloudlet: 6             1.28            12.69           13.969999


Cloudlet: 7             1.96            13.969999               15.929999


Cloudlet: 8             2.44            15.929999               18.369999


Cloudlet: 9             2.08            18.369999               20.449999
Average Waiting Time on Total: 8.881
Average Turn Around Time on Total: 10.926
```

Fig-4



```
<terminated> TaskSchedulingWithDeadlines [Java Application] C:\Program Files\AdoptOpenJDK\jdk-11.0.11.9-hotspot\bin\javaw.exe  (03-Jan-2022, 10:42:24 pm – 10:42:24 pm)

Cloudlet: 2          1.92          5.43          7.35


Cloudlet: 3          2.51          7.35          9.86


Cloudlet: 5          2.83          9.86          12.69


Cloudlet: 6          1.28          12.69         13.969999


Cloudlet: 7          1.96          13.969999             15.929999


Cloudlet: 8          2.44          15.929999             18.369999


Cloudlet: 9          2.08          18.369999             20.449999
Average Waiting Time on Total: 8.881
Average Turn Around Time on Total: 10.926

========== OUTPUT ==========
Cloudlet ID    STATUS    Data center ID    VM ID        Time    Start Time    Finish Time    user id
    4        SUCCESS        2            4           1.7       0.1            1.8    3
    1        SUCCESS        2            1           1.81       0.1            1.91    3
    0        SUCCESS        2            0           1.92       0.1            2.02    3
    2        SUCCESS        2            2           1.92       0.1            2.02    3
    3        SUCCESS        2            3           2.51       0.1            2.61    3
    5        SUCCESS        2            5           2.83       0.1            2.93    3
    6        SUCCESS        2            0           1.28       2.02            3.3    3
    7        SUCCESS        2            1           1.96       1.91            3.87    3
    8        SUCCESS        2            2           2.44       2.02            4.46    3
    9        SUCCESS        2            3           2.08       2.61            4.68    3
CloudSimExample6 finished!
```

Fig-5



```
Selected JOBS:
0 3 2 1 4
Total Profit: 11074
========== OUTPUT ==========
Cloudlet            Burst Time        Waiting Time     Turn Around Time
------------------------------------------------------------------

Cloudlet: 4            1.94            0.0             1.94


Cloudlet: 1            2.66            1.94            4.6000004


Cloudlet: 2            1.24            4.6000004               5.84


Cloudlet: 3            1.35            5.84            7.19


Cloudlet: 0            2.17            7.19            9.360001
Average Waiting Time on Total: 1.9570001
Average Turn Around Time on Total: 2.8930001
```

Fig-6

7. CONCLUSION

Cloud computing is a service that delivers computer resources such as data storage, data access, and a platform through a virtual environment rather than real hard discs. Cloud computing has exploded in popularity and altered the way people do business. The Internet is the only requirement for cloud computing. Enterprises have benefited considerably from the pay-as-you-use cloud payment model since they pay based on resource utilization, which decreases the capital spent by businesses on hardware and software maintenance.It is also adaptable to the needs of consumers because it allows for quick changes in hardware and software requirements, as well as improved security through the use of stronger data encryption techniques.

CloudSim is a very efficient Cloud task scheduling tool. It has many libraries that make the creation of virtual machines, data centers, brokers, and cloudlets very simple and easy. The Task Scheduling With Deadline scheduling algorithm created in our AAT took advantage of the same.

We can observe that the Task Scheduling With Deadline scheduling algorithm has high waiting time for each task as well as high average waiting time when compared to round-robin task scheduling and normal Task scheduling with a deadline. The Task Scheduling With Deadline scheduling algorithm even has a high response time by each of the virtual machines when compared to round-robin task scheduling and normal Task scheduling with a deadline.

Using these observations we can conclude that the Task Scheduling With Deadline scheduling algorithm finishes each task slower than the standard Round-Robin Algorithm.

8. REFERENCES

- https://www.geeksforgeeks.org/what-is-cloudsim/
- https://azure.microsoft.com/en-in/overview/what-is-cloud-computing/#benefits
- https://stackoverflow.com/questions/16590723/difference-between-time-shared-and-space-shared-algorithms
- https://github.com/Cloudslab/cloudsim/wiki/FAQ
- https://www.opensourceforu.com/2014/03/cloudsim-framework-modelling-simulating-cloud-environment/
- http://www.cloudbus.org/reports/CloudSim-ICPP2009.pdf
- http://www.ijeter.everscience.org/Manuscripts/Volume-4/Issue-12/Vol-4-issue-12-M-09.p df
-