



Kazakh-British Technical University

Faculty of Information Technologies  
Information Systems

Task report on the Cloudsim platform

Done by: Sapargali E.E.

Specialty: DS Yandex

Checked by: Akzhalova A.Z.,  
Professor at Faculty of IT

Almaty, 2021

## **Tasks**

Download Cloudsim and code the following problem solution:

Task scheduling depending on current workload and location of requests on available servers located in different data centers.

### **Task scheduling**

CloudletScheduler

Defined in VM level

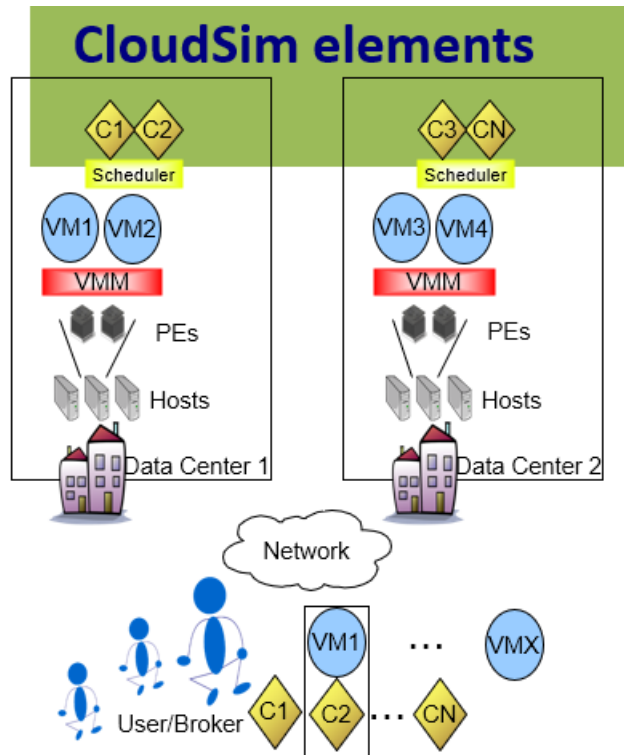
- Different VMs in the same host may have different policies

How to share processing power allocated to a VM among Cloudlets?

- OS scheduling
- Time-shared and space-shared

## Introduction

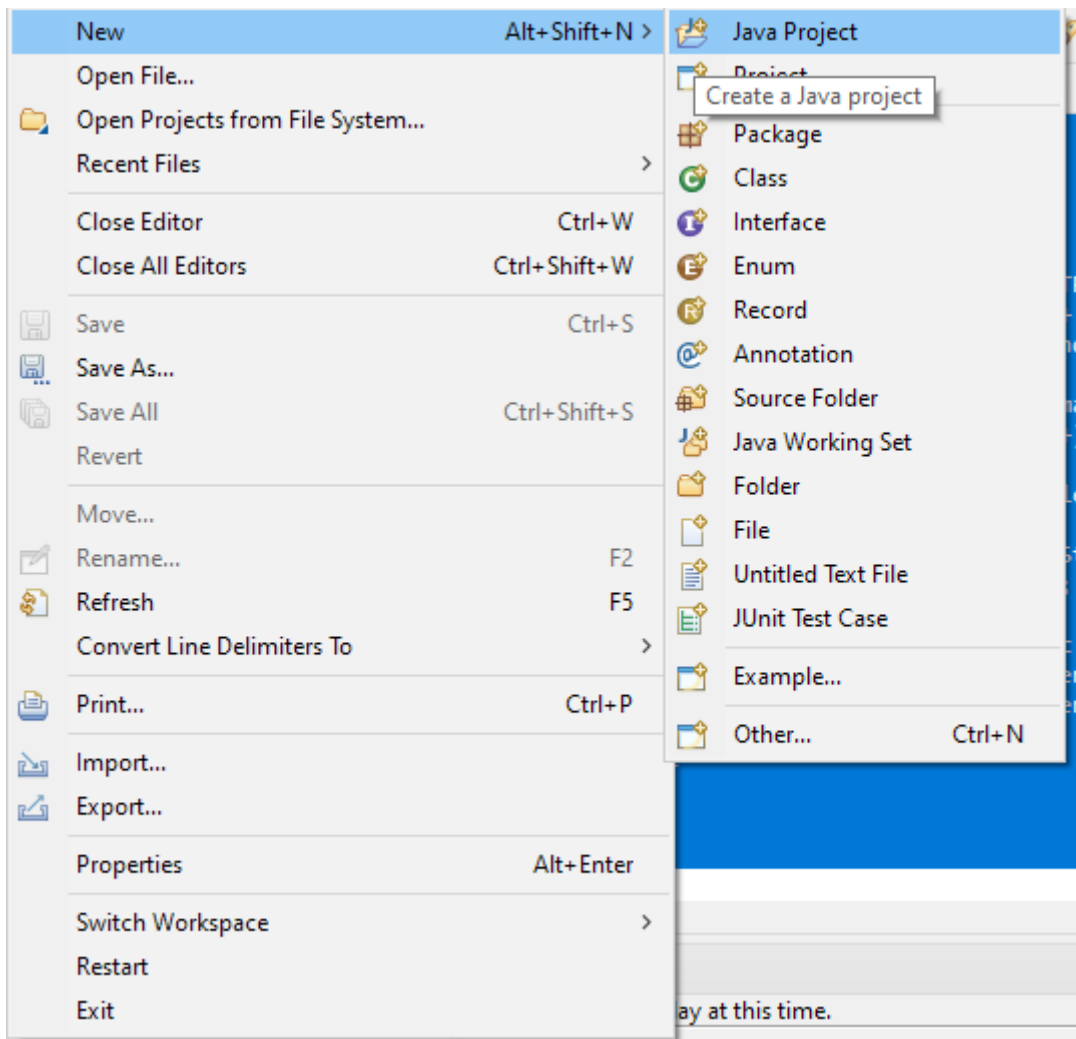
CloudSim is a platform for modeling and simulating cloud computing infrastructures and services . Originally created for work at the Cloud Computing and Distributed Systems Laboratory (CLOUDS) at the University of Melbourne, Australia, CloudSim has gradually become one of the most popular open-source cloud simulators in research. CloudSim is written entirely in Java.



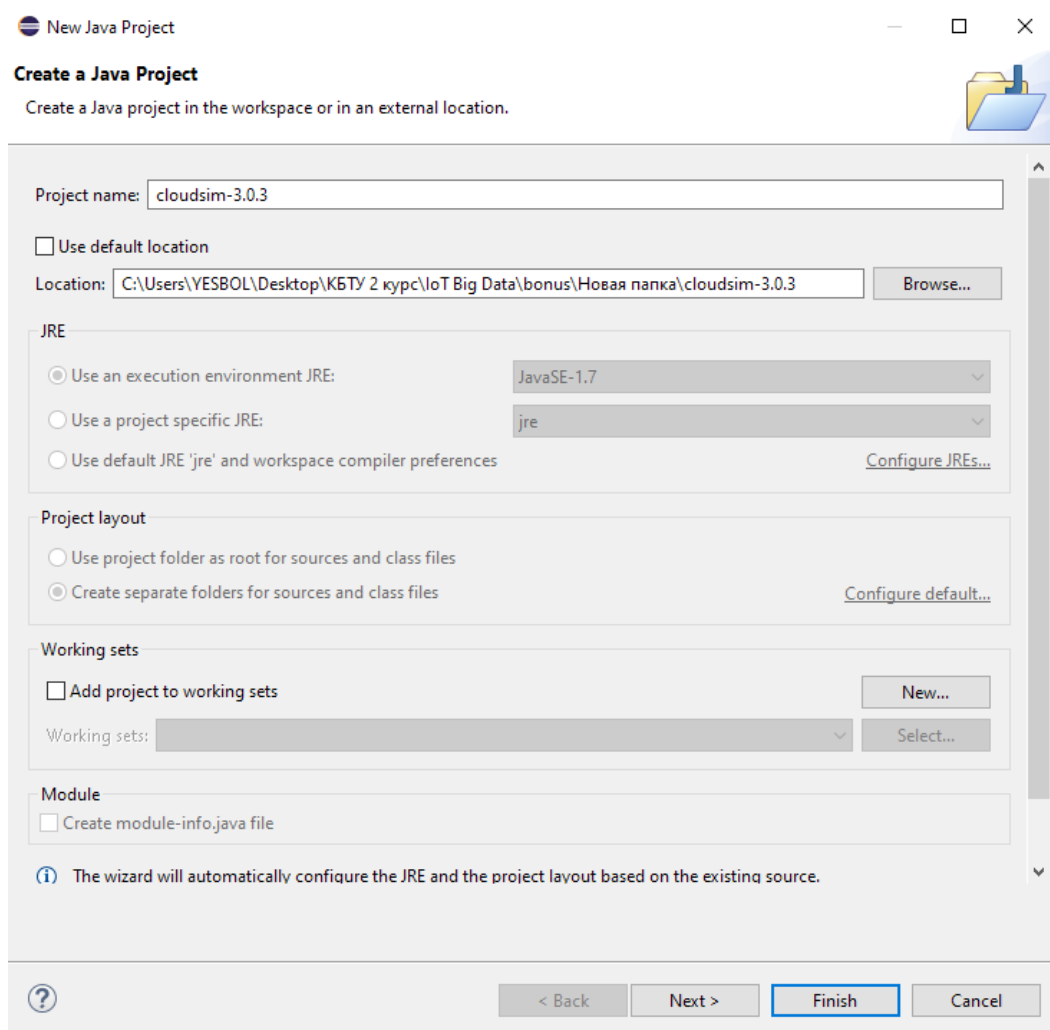
1. Cloudlet: set of tasks ready to submit for processing.
2. VM (Virtual Machine): this term represents a logical image of machine defined with characteristics just like physical machine.
3. Host: This will represent a physical machine with specific characteristics. Each host will consist of minimum one or more processing element.
4. Datacenter: This will represent a physical setup, which will have more than one interconnected host setup using a specific topology (to form a compute cluster).
5. Broker: This will represent a user or machine through which infrastructure (datacenter resources) will be accessed by virtual machines and cloudlets will be allocated to virtual machines for execution.

## 1 Performance of work

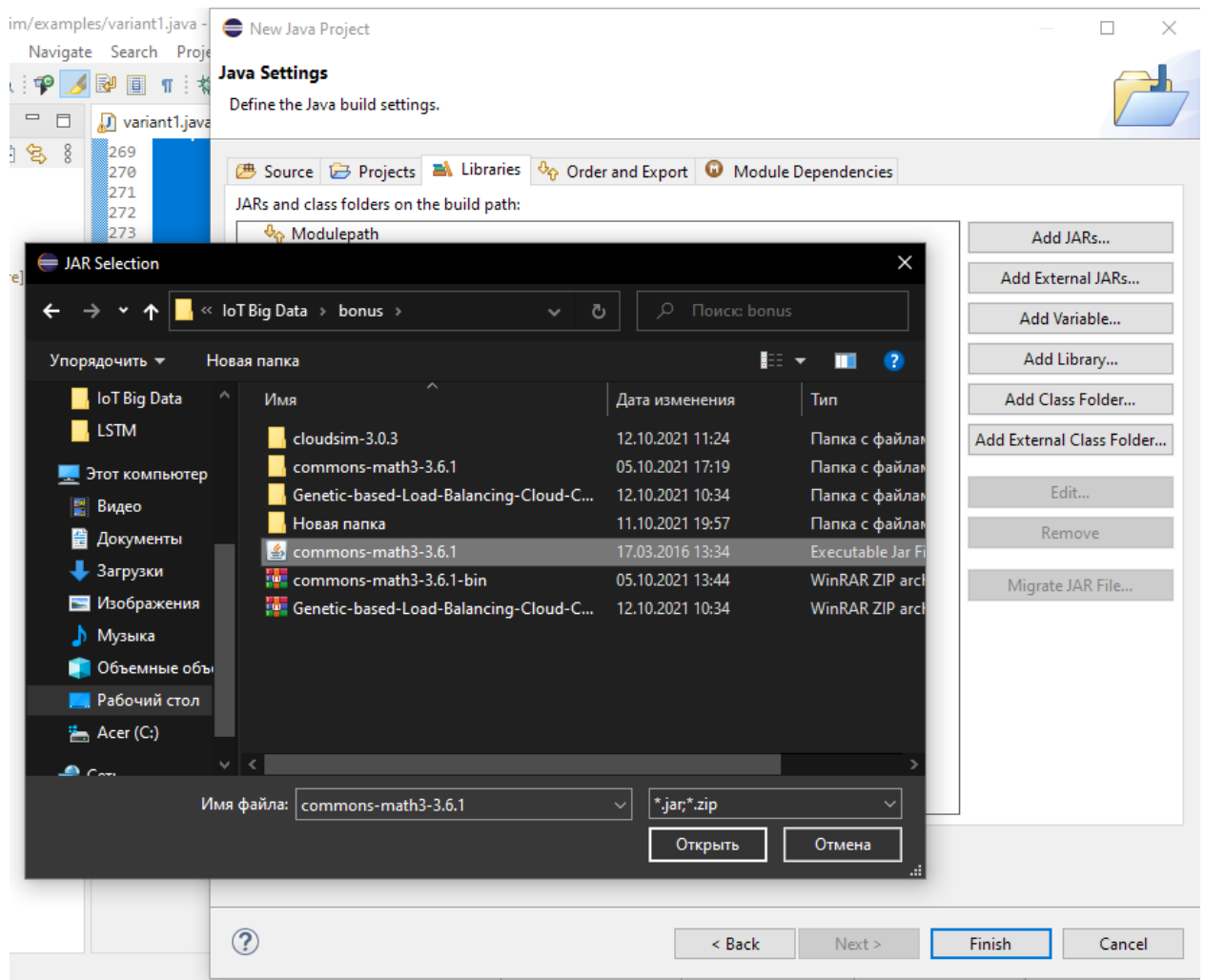
First go into Eclipse, go to File-> New -> Java Project to create a project.



In the new project window that opens, enter a project name and path where you unzipped the source code of the CloudSim project.



Next open the 'Libraries' tab and click on add external jar (Math common jar will be included in the project from this step).



I set about writing the code. The first thing to do is to create a container to store the VMs, which will be handed over to the broker later.

Create a VM and Set parameters to create a VM with a shared space scheduling policy for the cloud.

```
private static List<Vm> createVM(int userId, int vms) {
    LinkedList<Vm> list = new LinkedList<Vm>();

    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

    Vm[] vm = new Vm[vms];
    for(int i=0;i<vms;i++){
        vm[i] = new Vm(i, userId, mips+(i*10), pesNumber, ram, bw, size, vmm, new CloudletSchedulerTimeShared());
        list.add(vm[i]);
    }
    return list;
}
```

Creates a container to store Cloudlets and cloudlet parameters.

```

private static List<Cloudlet> createCloudlet(int userId, int cloudlets){
    LinkedList<Cloudlet> list = new LinkedList<Cloudlet>();

    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++){
        cloudlet[i] = new Cloudlet(i, (length + 2*i*10), pesNumber, fileSize, outputSize);
        cloudlet[i].setUserId(userId);
        list.add(cloudlet[i]);
    }

    return list;
}

```

Create a main() function to run with a few steps.

First step: Initialize the CloudSim package. It should be called before creating any entities. Initialize the CloudSim library.

```

public static void main(String[] args) {
    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.

```

@SuppressWarnings("unused")
Datacenter datacenter0 = createDatacenter("Datacenter_0");
@SuppressWarnings("unused")
Datacenter datacenter1 = createDatacenter("Datacenter_1");

```

Third step: Create Broker

```

DatacenterBroker broker = createBroker();
int brokerId = broker.getId();

```

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

```

broker.submitVmList(vmlist);
broker.submitCloudletList(cloudletList);

```

Starts the simulation and Print results when simulation is over

```

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

CloudSim.stopSimulation();

printCloudletList(newList);

```

To create a PowerDatacenter, you need a storage list for one or more machines.

The machine contains one or more PE or CPU/cores. Therefore a list should be created to store these PEs before creating the machine.

```
List<Pe> peList1 = new ArrayList<Pe>();  
  
int mips = 1000;
```

Create PEs and add them to the list.

```
// for a given core machine, a list of PEs as required  
peList1.add(new Pe(0, new PeProvisionerSimple(mips + 500)));  
peList1.add(new Pe(1, new PeProvisionerSimple(mips + 1500)));
```

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

```
int hostId=0;  
int ram = 4096; //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++;
```

Create a DatacenterCharacteristics object that stores:

- datacenter properties: architecture, OS, list
- machines, distribution policy: time or space sharing, time zone
- and its cost (time unit G\$/Pe).

```
String arch = "x86"; // system architecture  
String os = "Linux"; // operating system  
String vmm = "Xen";  
double time_zone = 6.0; // 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);
```

Finally, we need to create the PowerDatacenter object itself.



```

Datacenter datacenter = null;
try {
    datacenter = new Datacenter(name, characteristics, new VmAllocationPolicySimple(hostList), storageList, 0);
} catch (Exception e) {
    e.printStackTrace();
}

return datacenter;

```

Here is a simulation with four computers, two in each datacenter. And with their own timeshared and spaceshared policies.

```

variant1.java X variant2.java
140 private static Datacenter createDatacenter(String name){

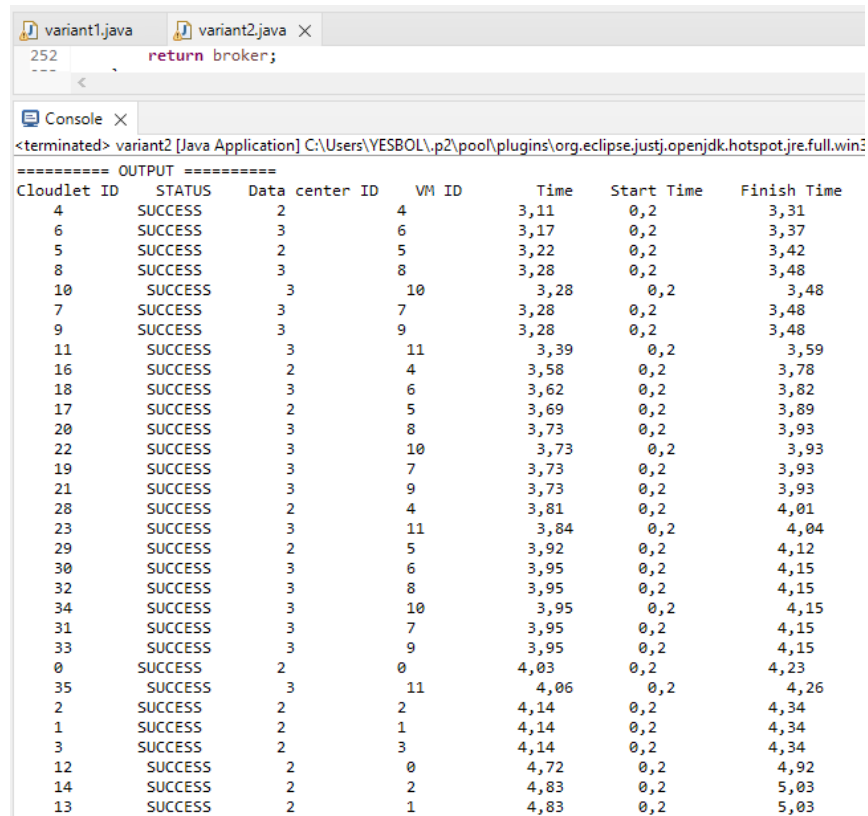
Console X
<terminated> variant1 [Java Application] C:\Users\YESBOL\.p2\pool\plugins\org.eclipse.justi
Initialising...
Starting CloudSim version 3.0
Datacenter_0 is starting...
Datacenter_1 is starting...
Broker is starting...
Entities started.
0.0: Broker: Cloud Resource List received with 2 resource(s)
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
[VmScheduler.vmCreate] Allocation of VM #2 to Host #0 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #2 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #3 to Host #0 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #3 to Host #1 failed by MIPS
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 #1
0.1: Broker: Creation of VM #2 failed in Datacenter #2
0.1: Broker: Creation of VM #3 failed in Datacenter #2
0.1: Broker: Trying to Create VM #2 in Datacenter_1
0.1: Broker: Trying to Create VM #3 in Datacenter_1
0.2: Broker: VM #2 has been created in Datacenter #3, Host #0
0.2: Broker: VM #3 has been created in Datacenter #3, Host #1
0.2: Broker: Sending cloudlet 0 to VM #0
0.2: Broker: Sending cloudlet 1 to VM #1
0.2: Broker: Sending cloudlet 2 to VM #2
0.2: Broker: Sending cloudlet 3 to VM #3
1.2: Broker: Cloudlet 0 received
1.2196078431372548: Broker: Cloudlet 2 received
1.31: Broker: Cloudlet 1 received
1.3296078431372549: Broker: Cloudlet 3 received
1.3296078431372549: Broker: All Cloudlets executed. Finishing...

Simulation completed.

===== OUTPUT =====
Cloudlet ID   STATUS   Data center ID   VM ID   Time   Start Time   Finish Time
0            SUCCESS   2                0        1        0,2          1,2
2            SUCCESS   3                2        1,02     0,2          1,22
1            SUCCESS   2                1        1,11     0,2          1,31
3            SUCCESS   3                3        1,13     0,2          1,33

```

The second case includes more machines compared to the first as it might in a real situation.



The screenshot shows an Eclipse IDE with two tabs: 'variant1.java' and 'variant2.java'. The 'variant2.java' tab is active, showing a line of code: `return broker;`. Below the code editor is a 'Console' window. The console output starts with a header '===== OUTPUT =====' followed by a table of execution results. The table has seven columns: 'Cloudlet ID', 'STATUS', 'Data', 'center ID', 'VM ID', 'Time', 'Start Time', and 'Finish Time'. The table contains 35 rows of data, all with a 'SUCCESS' status. The 'Time' column values range from 3,11 to 4,83. The 'Start Time' column values are mostly 0,2, with some 0,0. The 'Finish Time' column values range from 3,31 to 5,03.

Cloudlet ID	STATUS	Data	center ID	VM ID	Time	Start Time	Finish Time
4	SUCCESS	2	4	4	3,11	0,2	3,31
6	SUCCESS	3	6	6	3,17	0,2	3,37
5	SUCCESS	2	5	5	3,22	0,2	3,42
8	SUCCESS	3	8	8	3,28	0,2	3,48
10	SUCCESS	3	10	10	3,28	0,2	3,48
7	SUCCESS	3	7	7	3,28	0,2	3,48
9	SUCCESS	3	9	9	3,28	0,2	3,48
11	SUCCESS	3	11	11	3,39	0,2	3,59
16	SUCCESS	2	4	4	3,58	0,2	3,78
18	SUCCESS	3	6	6	3,62	0,2	3,82
17	SUCCESS	2	5	5	3,69	0,2	3,89
20	SUCCESS	3	8	8	3,73	0,2	3,93
22	SUCCESS	3	10	10	3,73	0,2	3,93
19	SUCCESS	3	7	7	3,73	0,2	3,93
21	SUCCESS	3	9	9	3,73	0,2	3,93
28	SUCCESS	2	4	4	3,81	0,2	4,01
23	SUCCESS	3	11	11	3,84	0,2	4,04
29	SUCCESS	2	5	5	3,92	0,2	4,12
30	SUCCESS	3	6	6	3,95	0,2	4,15
32	SUCCESS	3	8	8	3,95	0,2	4,15
34	SUCCESS	3	10	10	3,95	0,2	4,15
31	SUCCESS	3	7	7	3,95	0,2	4,15
33	SUCCESS	3	9	9	3,95	0,2	4,15
0	SUCCESS	2	0	0	4,03	0,2	4,23
35	SUCCESS	3	11	11	4,06	0,2	4,26
2	SUCCESS	2	2	2	4,14	0,2	4,34
1	SUCCESS	2	1	1	4,14	0,2	4,34
3	SUCCESS	2	3	3	4,14	0,2	4,34
12	SUCCESS	2	0	0	4,72	0,2	4,92
14	SUCCESS	2	2	2	4,83	0,2	5,03
13	SUCCESS	2	1	1	4,83	0,2	5,03

## **Conclusion**

In this optional task, a platform for modelling and simulating cloud computing infrastructures and services was explored.

The parameters specified in the task have been created. In the final simulation policies were inserted whereby the computers were started with a certain interval and allocated to two datacenters and according to a list of cloudlets.

It was also understood that the start time parameter in the simulation can be adjusted simultaneously for all machines or use a global broker.

## Resources

1 E. Rani and H. Kaur, "Study on fundamental usage of CloudSim simulator and algorithms of resource allocation in cloud computing," *2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, 2017, pp. 1-7, doi: 10.1109/ICCCNT.2017.8203998.

2 Calheiros, Rodrigo Neves et al. "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms." *Software: Practice and Experience* 41 (2011): n. pag.