CLOUD COMPUTING AND SERVICES

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
UNIVERSITY OF WEST ATTICA

DEPARTMENT OF INFORMATION AND COMPUTER ENGINEERING

PROJECT #1

SCENARIOS IN CLOUDSIM

WORK DETAILS

THEORY RESPONSIBLE: KALLERGIS DIMITRIOS – MAMALIS VASILEIOS
DELIVERY DATE : 16/6/2024
SUBMISSION DEADLINE : 16/06/2024

CLOUD COMPUTING AND SERVICES

STUDENT DETAILS

STUDENT PHOTO:

NAME : ATHANASIOU VASILEIOS EVANGELOS
STUDENT ID : 19390005
STUDENT SEMESTER : 10
STUDENT STATUS : UNDERGRADUATE
STUDY PROGRAM : UNIWA

CLOUD COMPUTING AND SERVICES

CONTENTS

Theoretical Part 5

1. How many Datacenters are being created? How many Hosts are created in each Datacenter ? ..5

2. How many processors (PEs) and what other features does each Host have (mips , memory, etc.)? 6

3. What other characteristics does each Datacenter have ? In which class are they represented? 7

4. How many VMs are created? What characteristics does each one have? 9

5. What parameters determine how many VMs each Host can accommodate ? And how (with what algorithm) is the attempt to assign them to the available Hosts (that is, how is it decided whether and on which Host each VM will run)? 10

6. How many VMs will eventually run and on which Host ? Why will some VMs not be assigned to any Host ? How would the above change (and why) if each VM was initially created with 2 PEs of 250 MIPS and 256 MB RAM ? 12

7. How many Cloudlets are created? With what characteristics each one? 15

8. How is it decided for each Cloudlet in which VM it will run (that is, with what algorithm is Cloudlets allocated to the available VMs)? How many Cloudlets will eventually run (both in total across all VMs and on each individual VM)? 16

9. With what policy (space sharing or time sharing – and where in the code is this seen / specified) is (a) routing the Cloudlets to the VMs they are ultimately assigned to run on? and (b) the routing of the VMs to the PEs (or otherwise, the delegation of the PEs to the VMs) of the Host on which they run? 19

10. Explain in detail the final part of the results (output table at the end) of the example - what each column gives etc, and in particular among others the start and end times of the Cloudlets (in direct correlation with the routing policies you observed being followed in question 9). 21

Practical Part 26

A. Run the example again with the rest of the alternatives provided to you (space/time sharing) for points 9(a) and 9(b) above, i.e. (a) for routing Cloudlets to VMs, and (b) for routing VMs to Hosts. Study the output again in each case, see whether or not there are differences, and explain why. Also run the example again applying a time policy to route the VMs sharing with over - subscription and describe-explain the output again 26

B. Make the minimum required changes to the available infrastructure (hosts , PEs and their characteristics) so that all VMs can run (without over - subscription) . Give three versions, (a) one without increasing the number of hosts and PEs (i.e. increasing only their attributes), (b) one increasing the number of PEs (without increasing the number of hosts), and (c) one increasing the number of hosts . Study the output again, see if there are any differences or not, and explain why. 38

C. Where in the code (simulator implementation) and how would you intervene (a) to change the policy / decision algorithm for 5. and 8., and (b) to add some other policy of your own for 9 (a) and 9(b); In particular with regard to 5. (the policy followed i.e. for assigning VMs to Hosts), explain in more detail how you would implement another policy of your choice – search and select another specific policy (consult, among others, the attached links and papers – see folder ' VM Allocation Policies '), describe it briefly, and then try to implement it. 44

Theoretical Part

1. How many Datacenters are being created? How many Hosts are created in each Datacenter ?

CloudSimExample 6 example simulation are created:

1. Datacenters

2 Datacenters (Datacenter_0, Datacenter_1), with entity_id=2 and entity_id=3 respectively

```
CloudSimExample6/main(String[] args)
```

```
// Second step: Create Datacenters
                    // Datacenters are the resource providers in CloudSim. We need to
list one of them to run a CloudSim simulation
                    @SuppressWarnings ( "unused" )
                    Datacenter datacenter0 = createDatacenter ( "Datacenter_0" );
                    @SuppressWarnings ( "unused" )
                    Datacenter datacenter1 = createDatacenter ( "Datacenter_1" );
```

2. Hosts (Physical hosting machines)

2 Hosts respectively in each Datacenter, with entity_id=0 and entity_id=1 respectively

```
CloudSimExample6/createDatacenter(String name) : Datacenter
```

```
             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
```

2. How many processors (PEs) and what other features does each Host have (mips , memory, etc.)?

Each Host has the following features :

1. **Processing Elements (PEs)**

   Host with entity _ id =0 has 4 processing elements, while Host with entity _ id = 1 has 2 processing elements.

2. **Processing Element Performance (MIPS)**

   The performance of a processing element of each Host is 1000 MIPS , which means that each processing element can execute 1000 * 1000000 = 1000000000 commands per second.

3. **Memory (RAM)**

   Both Hosts have 2048 MB of memory.

4. **Storage**

   Both Hosts have 1000000 MB (1 TB) of storage space.

5. **Network Bandwidth (BW)**

   Both Hosts have from 10000 MB / s network bandwidth.

```
CloudSimExample6/createDatacenter(String name) : Datacenter
```

```
// 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;
```

3. What other characteristics does each Datacenter have ? In which class are they represented?

The characteristics of each Datacenter are as follows:

1. **architecture system (arch)**

   Each Datacenter uses x86 based processors for hardware architecture system.

2. **Operating system (OS)**

   The operating system installed on the machines of each Datacenter is Linux .

3. **Virtual Machine Management (VMM)**

   The software that manages the virtual machines on the computers of each Datacenter is Xen .

4. **Time zone (time _zone)**

   The time zone in which each Datacenter is located is 10 (GMT + 10).

5. **Cost parameters**

   - **K cost of processing (cost)**

     The cost spent on the processing resources of each Datacenter is $ 3G / Pe .

   - **Memory cost (costPerMem)**

     The cost spent on the memory resources of each Datacenter is 0.05 G $/ Pe .

   - **Cost of space (costPerStorage)**

The cost spent on the storage resources of each Datacenter is 0.1 G$/Pe.

- **Bandwidth cost (costPerBw)**

Datacenter 's network bandwidth resources is $0.1 G / Pe .

```
CloudSimExample6/createDatacenter(String name) : Datacenter
```

```
// 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
```

Datacenters are represented in class **Datacenter.java** which is located at packet **org.cloudbus.cloudsim.**

```
CloudSimExample6/createDatacenter(String name) : Datacenter
```

```
// 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();
             }
```

4. How many VMs are created? What characteristics does each one have?

CloudSimExample 6 example simulation are created:

1. **VM (Virtual Machines)**

20 VMs with entity_id=0 to entity_id=19.

```
CloudSimExample6/main(String[] args)
```

```
//Fourth step: Create VMs and Cloudlets and send them to broker
                       vmlist = createVM ( brokerId ,20); // creating 20 vms
```

Each virtual machine has the following features:

1. **Image size**

   Each virtual machine is 10000 MB (10 GB) in size.

2. **Memory (RAM)**

   Each virtual machine has 512 MB of RAM .

3. **Processing Element Performance (MIPS)**

   The performance of a processing unit of each virtual machine is 1000 MIPS , which means that a processing unit can execute 1000 * 1000000 = 1000000000 instructions per second.

4. **Network Bandwidth (bw)**

   The network bandwidth available to each virtual machine is 1000 MB / s .

5. **Number of processing elements (PEs)**

   Each virtual machine has 1 processing unit.

6. **Virtual Machine Manager (VMM)**

   The software that manages each virtual machine is Xen .

```
CloudSimExample6/ createVM(int userId, int vms) : List<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
```

5. What parameters determine how many VMs each Host can accommodate ? And how (with what algorithm) is the attempt to assign them to the available Hosts (that is, how is it decided whether and on which Host each VM will run)?

The parameters that determine how many VMs each Host can host are:

1. **Processor performance (MIPS)**

   Each virtual machine requires a certain number of PEs with a certain MIPS (millions of instructions per second). The total number of available PEs and their performance (MIPS) on the Host determines how many virtual machines (VMs) it can support.

2. **Memory (RAM)**

   Each VM requires a certain amount of RAM . The total available RAM on the Host, along with the memory available to the VMs it hosts, determines the number of additional VMs the Host can host.

CloudSimExample6/Output

```
[VmScheduler.vmCreate] Allocation of VM #6 to Host #0 failed by RAM
[VmScheduler.vmCreate] Allocation of VM #6 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #7 to Host #0 failed by RAM
[VmScheduler.vmCreate] Allocation of VM #7 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #8 to Host #0 failed by RAM
[VmScheduler.vmCreate] Allocation of VM #8 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #9 to Host #0 failed by RAM
[VmScheduler.vmCreate] Allocation of VM #9 to Host #1 failed by MIPS
```

1. **Storage capacity**

   Each VM requires a certain amount of storage space. The total available storage on the Host together with the storage available to the VMs it hosts determines the number of additional VMs the Host can host.

2. **Network Bandwidth (BW)**

   Each VM may require a certain amount of network bandwidth. The total available bandwidth and the bandwidth available to existing VMs may limit the number of additional VMs .

Attempting to allocate VMs to Hosts is managed by the **VmAllocationPolicySimple class . java** located in the **org package . cloudbus . cloudsim** . The Host with the fewest processing elements (PEs) in use by other VMs will be chosen to host the next VM requesting it in ascending order of entity_id . The VM allocation policy is not the best possible and hence it is called **Worst - Fit** policy.

```
CloudSimExample6/createDatacenter(String name) : Datacenter
```

```
// 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();
             }
```

6. How many VMs will eventually run and on which Host ? Why will some VMs not be assigned to any Host ? How would the above change (and why) if each VM was initially created with 2 PEs of 250 MIPS and 256 MB RAM ?

In total, 20 VMs are created, as answered in question 4 , and their assignment status to Datacenter Hosts (for the number of Datacenters and Hosts in each Datacenter created during the simulation, see question 1) is as follows:

1. **Datacenter_0 (#2)**

   - **Host #0**

     VM #0, #1, #2, #4 (Total 4 VMs running on this Host).

   - **Host #1**

     VM #3, #5 (Total 2 VMs running on this Host).

```
CloudSimExample6/Output
```

```
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
```

1. **Datacenter_1 (#3)**

   - **Host #0**

     VM #6, #7, #8, #10 (Total 4 VMs running on this Host).

- **Host #1**

  VM #9, #11 (Total 2 VMs running on this Host)

CloudSimExample6/Output

```
0.2: Broker: VM #6 has been created in Datacenter #3, Host #0
0.2: Broker: VM #7 has been created in Datacenter #3, Host #0
0.2: Broker: VM #8 has been created in Datacenter #3, Host #0
0.2: Broker: VM #9 has been created in Datacenter #3, Host #1
0.2: Broker: VM #10 has been created in Datacenter #3, Host #0
0.2: Broker: VM #11 has been created in Datacenter #3, Host #1
```

The 20 virtual machines have the same characteristics (see question 4).

Hosts also have the same characteristics with the only difference being the number of PEs (see query 2).

The reasons why some VMs will not be assigned to any Host (for parameters that determine how many VMs a Host can host , see question 5) are as follows:

1. **Processor performance (MIPS)**

   Host #1 can host up to 2 VMs due to processor performance (MIPS) . Host #1 has 2 PEs processors , while each VM has 1. Given that the performance of the processors is the same (1000 MIPS) , it does not make it possible to Host #1 to host more than 2 VMs .

2. **Memory (RAM)**

   Host #0 can host up to 4 VMs due to RAM . Specifically, Host #0 has 2048 MB of memory, while each VM has 512 MB of memory (512 * 4 = 2048). Therefore, the hosting of a 5 th VM does not make it possible, as by hosting four VMs , the available memory of Host #0 is exhausted.

CloudSimExample6/Output

```
[VmScheduler.vmCreate] Allocation of VM #6 to Host #0 failed by RAM
[VmScheduler.vmCreate] Allocation of VM #6 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #7 to Host #0 failed by RAM
[VmScheduler.vmCreate] Allocation of VM #7 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #8 to Host #0 failed by RAM
[VmScheduler.vmCreate] Allocation of VM #8 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #9 to Host #0 failed by RAM
[VmScheduler.vmCreate] Allocation of VM #9 to Host #1 failed by MIPS
```

As answered in question 5 , the policy of assigning VMs to Hosts is the **Worst - Fit** policy. The resulting changes in the parameters of the VMs are in terms of processor performance which was reduced from 1 processor of 1000 MIPS to 2 processors of 250 MIPS . Also, the RAM memory was reduced from 512 MB to 256 MB . Therefore, by reducing these parameters, Hosts can host more VMs and the new situation is as follows:

1. **Datacenter_0 (#2)**

- **Host #0**

  VM #0, #1, #3, #5, #7, #9, #10, #11 (Total 8 VMs running on this Host).

- **Host #1**

  VM #2, #4, #6, #8 (Total 4 VMs running on this Host).

CloudSimExample6/Output

```
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 #1
0.1: Broker: VM #3 has been created in Datacenter #2, Host #0
0.1: Broker: VM #4 has been created in Datacenter #2, Host #1
0.1: Broker: VM #5 has been created in Datacenter #2, Host #0
0.1: Broker: VM #6 has been created in Datacenter #2, Host #1
0.1: Broker: VM #7 has been created in Datacenter #2, Host #0
0.1: Broker: VM #8 has been created in Datacenter #2, Host #1
0.1: Broker: VM #9 has been created in Datacenter #2, Host #0
0.1: Broker: VM #10 has been created in Datacenter #2, Host #0
0.1: Broker: VM #11 has been created in Datacenter #2, Host #0
```

2. **Datacenter_1 (#3)**

- **Host #0**

  VM #12, #13, #15, #17, #19 (Total 5 VMs running on this Host)

- **Host #1**

  VM #14, #16, #18 (Total 3 VMs running on this Host)

CloudSimExample6/Output

```
0.2: Broker: VM #12 has been created in Datacenter #3, Host #0
0.2: Broker: VM #13 has been created in Datacenter #3, Host #0
0.2: Broker: VM #14 has been created in Datacenter #3, Host #1
0.2: Broker: VM #15 has been created in Datacenter #3, Host #0
0.2: Broker: VM #16 has been created in Datacenter #3, Host #1
0.2: Broker: VM #17 has been created in Datacenter #3, Host #0
0.2: Broker: VM #18 has been created in Datacenter #3, Host #1
0.2: Broker: VM #19 has been created in Datacenter #3, Host #0
```

As we can see in the simulation output , all 20 VMs are created and distributed to the Hosts .

O Host #0 with 4 processing elements of 4000 MIPS total performance, can host a maximum of 8 VMs with 2 processing elements of 500 MIPS total performance.

Host #1 with 2 processing elements of 2000 MIPS total performance, can host a maximum of 4 VMs with 2 processing elements of 500 MIPS total performance.

The available memory is used up in the 1st ^{case} as it was in the initial parameters of the VMs .

7. How many Cloudlets are created? With what characteristics each one?

CloudSimExample 6 example simulation are created:

1. **Cloudlets (Processes)**

   40 Cloudlets with entity_id=0 to entity_id=39

```
CloudSimExample6/main(String[] args)

//Fourth step: Create VMs and Cloudlets and send them to broker
                        vmlist = createVM ( brokerId ,20); //creating 20 vms
                        cloudletList = createCloudlet ( brokerId ,40); // creating 40
cloudlets
```

Cloudlets have the following features:

1. **Size (length)**

   The size of the Cloudlet is determined by the number of commands it must execute. Each Cloudlet requires 1000 MI (Millions Instructions) to complete its execution. The processor performance (measured in MIPS) and the size of the Cloudlet determine the execution time of the Cloudlet in the VM .

2. **Input file size (fileSize)**

   The input file size that each Cloudlet needs to start running is 300 bytes .

3. **Output File Size (outputSize)**

   The size of the output file that each Cloudlet will produce upon completion of its execution is 300 bytes .

4. **Number of processing elements (PEs)**

   Each Cloudlet requires 1 PE to run.

```
CloudSimExample6/createCloudlet(int userId, int cloudlets ) : List<Cloudlet>
```

```
// cloudlet parameters
             long length = 1000;
             long fileSize = 300;
             long outputSize = 300;
             int pesNumber = 1;
```

8. How is it decided for each Cloudlet in which VM it will run (that is, with what algorithm is Cloudlets allocated to the available VMs)? How many Cloudlets will eventually run (both in total across all VMs and on each individual VM)?

Each Cloudlet is assigned to the first virtual machine (VM) that satisfies its computing requirements. All VMs have the same characteristics and satisfy the requirements of each Cloudlet which also have the same characteristics (for VM characteristics see question 4 , for Cloudlets characteristics see question 7). This policy is called **First - Fit** .

```
CloudSimExample6/Output
```

```
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
0.2: Broker: Sending cloudlet 4 to VM #4
0.2: Broker: Sending cloudlet 5 to VM #5
0.2: Broker: Sending cloudlet 6 to VM #6
0.2: Broker: Sending cloudlet 7 to VM #7
0.2: Broker: Sending cloudlet 8 to VM #8
0.2: Broker: Sending cloudlet 9 to VM #9
0.2: Broker: Sending cloudlet 10 to VM #10
0.2: Broker: Sending cloudlet 11 to VM #11
0.2: Broker: Sending cloudlet 12 to VM #0
0.2: Broker: Sending cloudlet 13 to VM #1
0.2: Broker: Sending cloudlet 14 to VM #2
0.2: Broker: Sending cloudlet 15 to VM #3
0.2: Broker: Sending cloudlet 16 to VM #4
0.2: Broker: Sending cloudlet 17 to VM #5
0.2: Broker: Sending cloudlet 18 to VM #6
0.2: Broker: Sending cloudlet 19 to VM #7
0.2: Broker: Sending cloudlet 20 to VM #8
0.2: Broker: Sending cloudlet 21 to VM #9
0.2: Broker: Sending cloudlet 22 to VM #10
0.2: Broker: Sending cloudlet 23 to VM #11
0.2: Broker: Sending cloudlet 24 to VM #0
0.2: Broker: Sending cloudlet 25 to VM #1
0.2: Broker: Sending cloudlet 26 to VM #2
0.2: Broker: Sending cloudlet 27 to VM #3
0.2: Broker: Sending cloudlet 28 to VM #4
0.2: Broker: Sending cloudlet 29 to VM #5
0.2: Broker: Sending cloudlet 30 to VM #6
0.2: Broker: Sending cloudlet 31 to VM #7
```

```
0.2: Broker: Sending cloudlet 32 to VM #8
0.2: Broker: Sending cloudlet 33 to VM #9
0.2: Broker: Sending cloudlet 34 to VM #10
0.2: Broker: Sending cloudlet 35 to VM #11
0.2: Broker: Sending cloudlet 36 to VM #0
0.2: Broker: Sending cloudlet 37 to VM #1
0.2: Broker: Sending cloudlet 38 to VM #2
0.2: Broker: Sending cloudlet 39 to VM #3
```

In total, all 40 Cloudlets assigned to the 12 VMs created were executed. In detail, the status of Cloudlets is as follows:

1. **VM #0**

   Cloudlet #0, #12, #24, #36 (Total 4 Cloudlets run on this VM)

2. **VM #1**

   Cloudlet #1, #13, #25, #37 (Total 4 Cloudlets ran on this VM)

3. **VM #2**

   Cloudlet #2, #14, #26, #38 (Total 4 Cloudlets ran on this VM)

4. **VM #3**

   Cloudlet #3, #15, #27, #39 (Total 4 Cloudlets ran on this VM)

5. **VM #4**

   Cloudlet #4, #16, #28 (Total 3 Cloudlets run on this VM)

6. **VM #5**

   Cloudlet #5, #17, #29 (A total of 4 Cloudlets ran on this VM)

7. **VM #6**

   Cloudlet #6, #18, #30 (Total 3 Cloudlets run on this VM)

8. **VM #7**

   Cloudlet #7, #19, #31 (A total of 3 Cloudlets were run on this VM)

9. **VM #8**

Cloudlet #8, #20, #32 (A total of 3 Cloudlets were run on this VM)

10. VM #9

Cloudlet #9, #21, #33 (Total 3 Cloudlets run on this VM)

11. VM #10

Cloudlet #10, #22, #34 (A total of 3 Cloudlets were run on this VM)

12. VM #11

Cloudlet #11, #23, #35 (Total 3 Cloudlets ran on this VM)

CloudSimExample6/Output

```
3.1980000000000004: Broker: Cloudlet 4 received
3.1980000000000004: Broker: Cloudlet 16 received
3.1980000000000004: Broker: Cloudlet 28 received
3.1980000000000004: Broker: Cloudlet 5 received
3.1980000000000004: Broker: Cloudlet 17 received
3.1980000000000004: Broker: Cloudlet 29 received
3.1980000000000004: Broker: Cloudlet 6 received
3.1980000000000004: Broker: Cloudlet 18 received
3.1980000000000004: Broker: Cloudlet 30 received
3.1980000000000004: Broker: Cloudlet 7 received
3.1980000000000004: Broker: Cloudlet 19 received
3.1980000000000004: Broker: Cloudlet 31 received
3.1980000000000004: Broker: Cloudlet 8 received
3.1980000000000004: Broker: Cloudlet 20 received
3.1980000000000004: Broker: Cloudlet 32 received
3.1980000000000004: Broker: Cloudlet 10 received
3.1980000000000004: Broker: Cloudlet 22 received
3.1980000000000004: Broker: Cloudlet 34 received
3.1980000000000004: Broker: Cloudlet 9 received
3.1980000000000004: Broker: Cloudlet 21 received
3.1980000000000004: Broker: Cloudlet 33 received
3.1980000000000004: Broker: Cloudlet 11 received
3.1980000000000004: Broker: Cloudlet 23 received
3.1980000000000004: Broker: Cloudlet 35 received
4.198: Broker: Cloudlet 0 received
4.198: Broker: Cloudlet 12 received
4.198: Broker: Cloudlet 24 received
4.198: Broker: Cloudlet 36 received
4.198: Broker: Cloudlet 1 received
4.198: Broker: Cloudlet 13 received
4.198: Broker: Cloudlet 25 received
4.198: Broker: Cloudlet 37 received
4.198: Broker: Cloudlet 2 received
4.198: Broker: Cloudlet 14 received
4.198: Broker: Cloudlet 26 received
```

```
4.198: Broker: Cloudlet 38 received
4.198: Broker: Cloudlet 3 received
4.198: Broker: Cloudlet 15 received
4.198: Broker: Cloudlet 27 received
4.198: Broker: Cloudlet 39 received
4.198: Broker: All Cloudlets executed. Finishing...
```

9. With what policy (space sharing or time sharing – and where in the code is this seen / specified) is (a) routing the Cloudlets to the VMs they are ultimately assigned to run on? and (b) the routing of the VMs to the PEs (or otherwise, the delegation of the PEs to the VMs) of the Host on which they run?

CloudSimExample 6 simulation , the algorithm that:

*(a) routes Cloudlets to virtual machines (VMs) is **Time - Shared** scheduling **Cloudlets Scheduling**).*

In time-sharing Cloudlets routing, multiple Cloudlets can run in parallel on a VM . This is possible provided the VM has sufficient computing resources to serve multiple Cloudlets in parallel. The class representing this algorithm is **CloudletSchedulerTimeShared . java** located in the **org package . cloudbus . cloudsim** .

```
CloudSimExample6/createVM(int userId, int vms) : List<Vm>

//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 CloudletSchedulerTimeShared());
                    //for creating a VM with a space shared scheduling policy for
cloudlets :
                    // vm [i] = Vm (i, userId, mips , pesNumber, ram, bw , size,
priority, vmm , new CloudletSchedulerSpaceShared());

                    list . add ( vm [ i ]);
             }
```

*(b) routes the VMs to the processing elements (PEs) of the Hosts is **Time - Shared** scheduling **VM Scheduling**).*

In time-shared VM scheduling, virtual machines share the Processing Elements (PEs) of the Host hosting them. Since a processing element (PE) can satisfy the requirements of several VMs simultaneously, then those with time sharing will run in parallel on the same PE of the Host or on a set of PEs of the same Host . The class representing this algorithm is **VmSchedulerTimeShared . java** located in the **org package . cloudbus . cloudsim** .

CloudSimExample6/createDatacenter(String name) : Datacenter

```
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
```

10. Explain in detail the final part of the results (output table at the end) of the example - what each column gives etc, and in particular among others the start and end times of the Cloudlets (in direct correlation with the routing policies you observed being followed in question 9).

CloudSimExample 6 simulation results shows a table with the following columns:

1. **Cloudlet ID**

   This is the Cloudlet assigned to the corresponding VM to run. Its identification is determined by the entity _ id given to it during its creation. The ids range from #0 to #39, as a total of 40 Cloudlets were created (see question 7).

2. **Status**

   This is the execution state of Cloudlets . SUCCESS means that the Cloudlet was executed successfully, while FAIL means the opposite.

3. **Data center ID**

   This is the Datacenter in which the VM that executed the corresponding Cloudlet is located . Its identification is determined by the entity _ id given to it during its creation. The ids range from #2 to #3, as a total of 2 Datacenters were created (see question 1).

4. **VM ID**

This is the VM in which the corresponding Cloudlet is running . Its identification is determined by the entity _ id given to it during its creation. The ids range from #0 to #19, as a total of 20 VMs were created (see query 4).

5. **Time**

This is the time spent by the VM 's processor to run the corresponding Cloudlet . This time is measured in **seconds** and its calculation comes out of Eq

$$\text{Time} = \text{Finish Time} - \text{Start Time}$$

6. **Start Time**

This is the start time of the execution of the corresponding Cloudlet (measured in seconds).

7. **Finish Time**

This is the completion time of the execution of the corresponding Cloudlet (measured in seconds).

CloudSimExample6/Output

```
========== OUTPUT ==========
Cloudlet ID STATUS Data center ID VM ID Time Start Time Finish Time
4 SUCCESS 2 4 3 0.2 3.2
16 SUCCESS 2 4 3 0.2 3.2
28 SUCCESS 2 4 3 0.2 3.2
5 SUCCESS 2 5 3 0.2 3.2
17 SUCCESS 2 5 3 0.2 3.2
29 SUCCESS 2 5 3 0.2 3.2
6 SUCCESS 3 6 3 0.2 3.2
18 SUCCESS 3 6 3 0.2 3.2
30 SUCCESS 3 6 3 0.2 3.2
7 SUCCESS 3 7 3 0.2 3.2
19 SUCCESS 3 7 3 0.2 3.2
31 SUCCESS 3 7 3 0.2 3.2
8 SUCCESS 3 8 3 0.2 3.2
20 SUCCESS 3 8 3 0.2 3.2
32 SUCCESS 3 8 3 0.2 3.2
10 SUCCESS 3 10 3 0.2 3.2
22 SUCCESS 3 10 3 0.2 3.2
34 SUCCESS 3 10 3 0.2 3.2
9 SUCCESS 3 9 3 0.2 3.2
21 SUCCESS 3 9 3 0.2 3.2
33 SUCCESS 3 9 3 0.2 3.2
11 SUCCESS 3 11 3 0.2 3.2
23 SUCCESS 3 11 3 0.2 3.2
35 SUCCESS 3 11 3 0.2 3.2
0 SUCCESS 2 0 4 0.2 4.2
12 SUCCESS 2 0 4 0.2 4.2
24 SUCCESS 2 0 4 0.2 4.2
```

```
36 SUCCESS 2 0 4 0.2 4.2
1 SUCCESS 2 1 4 0.2 4.2
13 SUCCESS 2 1 4 0.2 4.2
25 SUCCESS 2 1 4 0.2 4.2
37 SUCCESS 2 1 4 0.2 4.2
2 SUCCESS 2 2 4 0.2 4.2
14 SUCCESS 2 2 4 0.2 4.2
26 SUCCESS 2 2 4 0.2 4.2
38 SUCCESS 2 2 4 0.2 4.2
3 SUCCESS 2 3 4 0.2 4.2
15 SUCCESS 2 3 4 0.2 4.2
27 SUCCESS 2 3 4 0.2 4.2
39 SUCCESS 2 3 4 0.2 4.2
CloudSimExample 6 finished !
```

The times displayed in the **Time column** they vary between 3 and 4 seconds . If we consider that the routing policy of Cloudlets to VMs and VMs to PEs of Hosts is **Time - Shared** policy (see question 9 for details), times are expectedly short. This is not only due to the routing policy, but also to the number of Cloudlets each VM has , as well as to the corresponding characteristics of the Host hosting the VM , the characteristics of the VM and the running Cloudlet .

As answered in question 8 , the Cloudlets assigned to VMs #0, #1, #2, #3 took 4 seconds each to run, while the remaining VMs (#4 ... #19) took 3 seconds each . The answer to the small time difference has to do with the creation of the 40 Cloudlets and their distribution to the 12 VMs that were successfully created on the respective Hosts . The policy for assigning Cloudlets to VMs is **First - Fit** , therefore, the first 4 VMs will have one more Cloudlet than the remaining 16 VMs .

Let's take for example Host #0 of Datacenter #2.

Host #0 is a physical 4-core machine with a performance of 1000 MIPS per core. It hosts a total of 4 VMs that are 1-core with a performance of 1000 MIPS each. Based on the **First - Fit policy** , VM #0, #1, #2 are assigned 4 Cloudlets of size 1000 MI each, while VM #4 is assigned 3 Cloudlets of size 1000 MI each.

Time - Shared routing policy **Cloudlets Scheduling** and **Time - Shared VM Scheduling** , Cloudlets run simultaneously on VMs , and VMs use the same Host core in parallel , as shown in **Figure 10.1.**

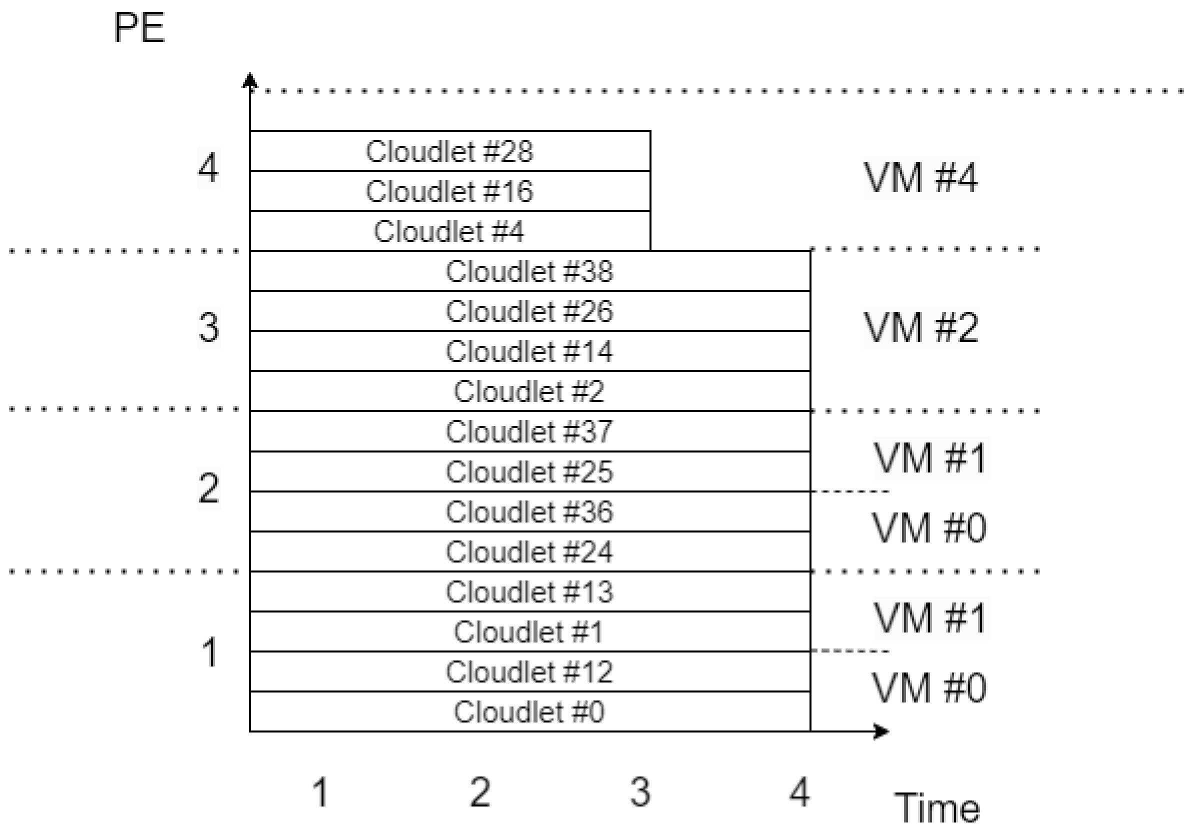


Figure 10.1 Time-Shared Cloudlet Scheduling and Time-Shared VM Scheduling on Host #0 of Datacenter #2

Each Host core is used simultaneously by 1-2 VMs running 2-3 Cloudlets each. With the size of each Cloudlet at 1000 MI and the performance of each Host core at 1000 MIPS , each Cloudlet runs at

$$\text{Rate} = \text{MIPS} / \#\text{Cloudlets}$$

, where MIPS is the throughput of the Host core running the Cloudlet , and # Cloudlets is the number of Cloudlets running on the Host core . In detail we have:

- **Core 1**

  $$\text{Rate} = \text{MIPS} / \#\text{Cloudlets} \rightarrow \text{Rate} = 1000 / 4 \rightarrow \text{Rate} = 250 \text{ MIPS}$$

  Each Cloudlet of 1000 MI runs at a rate of 250 MIPS , so it will take 4 seconds to run them.

- **Core 2**

  Similarly, as it applies to Core 1

- **Core 3**

  Similarly, as it applies to Core 1

- **Core 4**

Rate = MIPS / #Cloudlets → Rate = 1000 / 3 → Rate ≅ 333 MIPS

Each Cloudlet of 1000 MI runs roughly at a rate of about 333 MIPS , so it will take 3 seconds to run.

Finally, we notice that the startup times of the Cloudlets are from 0.2 second instead of 0.0 second , as one time is spent **for** the creation of the VMs and their assignment to the Hosts of the Datacenters , as well as the assignment of the Cloudlets to the VMs . The entity that manages these processes is called **a Broker** and are represented in the **DatacenterBroker class . java** located in the package **org . cloudbus . cloudsim** .

As we can see in the simulation output , 0.1 seconds were needed to create the VMs and assign them to the Hosts of Datacenter #2.

CloudSimExample6/Output

```
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
```

Likewise, we see in the output of the simulation, another 0.1 seconds were needed to create the VMs and assign them to the Hosts of Datacenter #3, as well as to create the Cloudlets and assign them to the VMs .

CloudSimExample6/Output

```
0.2: Broker: VM #6 has been created in Datacenter #3, Host #0
0.2: Broker: VM #7 has been created in Datacenter #3, Host #0
0.2: Broker: VM #8 has been created in Datacenter #3, Host #0
0.2: Broker: VM #9 has been created in Datacenter #3, Host #1
0.2: Broker: VM #10 has been created in Datacenter #3, Host #0
0.2: Broker: VM #11 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
0.2: Broker: Sending cloudlet 4 to VM #4
```

Practical Part

A. Run the example again with the rest of the alternatives provided to you (space/time sharing) for points 9(a) and 9(b) above, i.e. (a) for routing Cloudlets to VMs, and (b) for routing VMs to Hosts. Study the output again in each case, see whether or not there are differences, and explain why. Also run the example again applying a time policy to route the VMs sharing with over - subscription and describe-explain the output again .

As answered in question 9 , the routing policies followed by the **CloudSimExample 6 simulation** are

(a) Time-Shared Cloudlet Scheduling

(b) Time-Shared VM Scheduling

Let's examine the other alternatives available to us:

(a) Space-Shared Cloudlet Scheduling & Time-Shared VM Scheduling (SC-TV)

Cloudlet scheduling with space sharing (**Space - Shared Cloudlets Scheduling**), if a Cloudlet is assigned to a VM for execution, then no other Cloudlet will be executed from that VM until the current Cloudlet is released from the VM . Multiple Cloudlets cannot run in parallel in a VM , even if the VM is capable enough to do so. The class representing this algorithm is **CloudletSchedulerSpaceShared . java** located in the **org package . cloudbus . cloudsim** .

CloudSimExample6/createVM(int userId, int vms) : List<Vm>

```
//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 CloudletSchedulerTimeShared());
                    //for creating a VM with a space shared scheduling policy for
cloudlets :
                    vm [ i ] = new Vm( i , userId , mips , pesNumber , ram , bw , size ,
vmm , new CloudletSchedulerSpaceShared());

                    list . add ( vm [ i ]);
             }
```

Time-Shared VM scheduling (**Time - Shared VM Scheduling**), is clearly mentioned in question 9 .

The final part of the **CloudSimExample 6 simulation results** shows a table with the following columns (for an explanation of the simulation columns, see query 10)

CloudSimExample6/Output

```
========== OUTPUT ==========
Cloudlet ID STATUS Data center ID VM ID Time Start Time Finish Time
0 SUCCESS 2 0 1 0.2 1.2
1 SUCCESS 2 1 1 0.2 1.2
2 SUCCESS 2 2 1 0.2 1.2
4 SUCCESS 2 4 1 0.2 1.2
3 SUCCESS 2 3 1 0.2 1.2
5 SUCCESS 2 5 1 0.2 1.2
6 SUCCESS 3 6 1 0.2 1.2
7 SUCCESS 3 7 1 0.2 1.2
8 SUCCESS 3 8 1 0.2 1.2
10 SUCCESS 3 10 1 0.2 1.2
9 SUCCESS 3 9 1 0.2 1.2
11 SUCCESS 3 11 1 0.2 1.2
12 SUCCESS 2 0 1 1.2 2.2
13 SUCCESS 2 1 1 1.2 2.2
14 SUCCESS 2 2 1 1.2 2.2
16 SUCCESS 2 4 1 1.2 2.2
15 SUCCESS 2 3 1 1.2 2.2
17 SUCCESS 2 5 1 1.2 2.2
18 SUCCESS 3 6 1 1.2 2.2
19 SUCCESS 3 7 1 1.2 2.2
20 SUCCESS 3 8 1 1.2 2.2
22 SUCCESS 3 10 1 1.2 2.2
21 SUCCESS 3 9 1 1.2 2.2
23 SUCCESS 3 11 1 1.2 2.2
24 SUCCESS 2 0 1 2.2 3.2
25 SUCCESS 2 1 1 2.2 3.2
26 SUCCESS 2 2 1 2.2 3.2
28 SUCCESS 2 4 1 2.2 3.2
27 SUCCESS 2 3 1 2.2 3.2
29 SUCCESS 2 5 1 2.2 3.2
30 SUCCESS 3 6 1 2.2 3.2
31 SUCCESS 3 7 1 2.2 3.2
32 SUCCESS 3 8 1 2.2 3.2
34 SUCCESS 3 10 1 2.2 3.2
33 SUCCESS 3 9 1 2.2 3.2
35 SUCCESS 3 11 1 2.2 3.2
36 SUCCESS 2 0 1 3.2 4.2
37 SUCCESS 2 1 1 3.2 4.2
38 SUCCESS 2 2 1 3.2 4.2
39 SUCCESS 2 3 1 3.2 4.2
CloudSimExample 6 finished !
```

The difference in the execution times of the Cloudlets of the **SC - TV** policy with the corresponding times of the **TC - TV policy** that we saw in question 10 is obvious, because each Cloudlet the processing component of the VM is exclusively committed , not being allowed to be committed by another until its execution is complete. Therefore, each Cloudlet of size 1000 MI will exclusively commit the core of the VM performing 1000 MIPS , which justifies the execution time of each Cloudlet at 1 second .

We also notice a difference in the startup and completion times of each Cloudlet , as no priority has been set and for this reason the Cloudlets bind their VM cores in ascending order of entity_id . Therefore, the startup time of the next Cloudlet depends solely on the completion time of the immediately preceding Cloudlet that bound the VM core .

Let's take for example Host #0 of Datacenter #2.

Host #0 is a physical 4-core machine with a performance of 1000 MIPS per core. It hosts a total of 4 VMs that are 1-core with a performance of 1000 MIPS each. Based on the **First - Fit policy** , VM #0, #1, #2 are assigned 4 Cloudlets of size 1000 MI each, while VM #4 is assigned 3 Cloudlets of size 1000 MI each.

With the **Space - Shared routing policy Cloudlets Scheduling (SC)** and **Time - Shared VM Scheduling (TV)** , Cloudlets do not run simultaneously on VMs in contrast to VMs that use the same core of the Host in parallel , as shown in **Figure A.1**

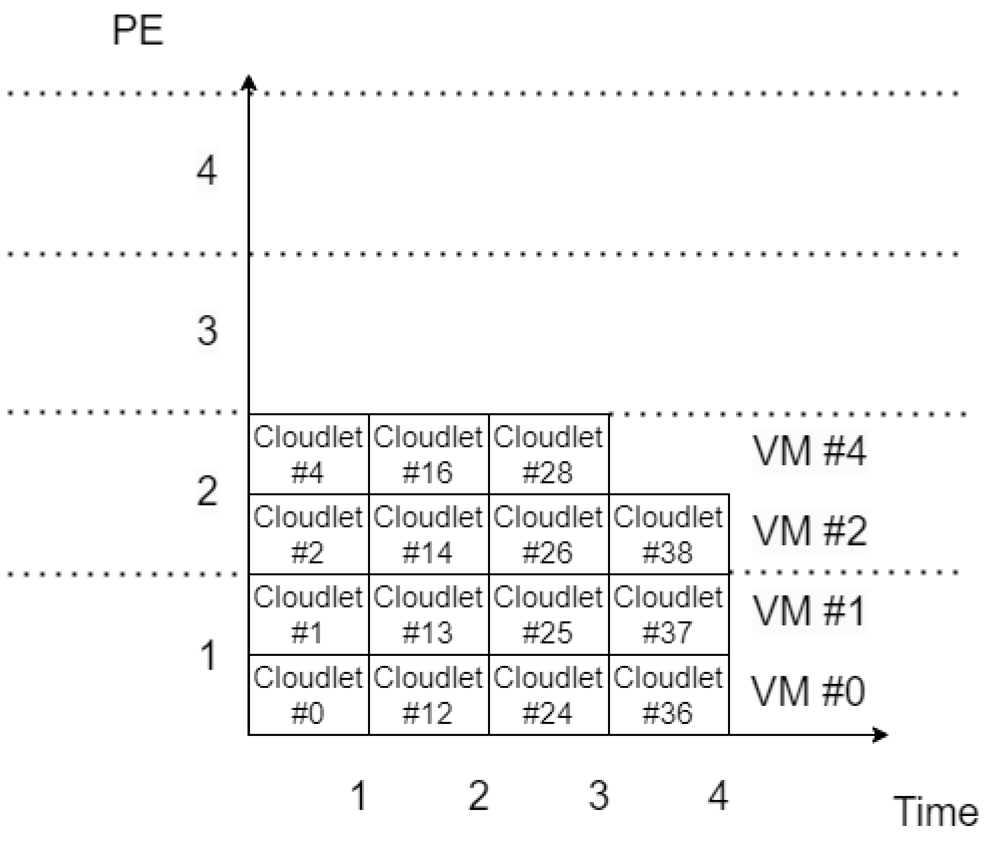


Figure A.1 Space-Shared Cloudlet Scheduling and Time-Shared VM Scheduling on Host #0 of Datacenter #2

(b) Space-Shared Cloudlet Scheduling & Space-Shared VM Scheduling (SC-SV)

Cloudlet scheduling with space sharing (**Space - Shared Cloudlets Scheduling**), is clearly mentioned in case a.

VM scheduling with space sharing (**Space - Shared VM Scheduling**), processing elements (PE) are distributed across a single VM . Strictly speaking, it means that if a PE is registered to a VM , it cannot be accessed by another VM until it is released. The class representing this algorithm is **VmSchedulerSpaceShared . java** located in the **org package . cloudbus . cloudsim** .

```
CloudSimExample6/createDatacenter(String name) : Datacenter
```

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

             hostId ++;

             //To create a host with a space-shared allocation policy for PEs to VMs:
                           hostList .add(
                                  new Host(
                                         hostId ,
                                         new RamProvisionerSimple( ram ),
                                         new BwProvisionerSimple( bw ),
                                         storage ,
                                         peList2 ,
                                         new VmSchedulerSpaceShared( peList2 )
                                  )
                           );
```

CloudSimExample 6 simulation results shows a table with the following columns (see query 10)

```
CloudSimExample6/Output
```

```
========== OUTPUT ==========
Cloudlet ID STATUS Data center ID VM ID Time Start Time Finish Time
0 SUCCESS 2 0 1 0.2 1.2
1 SUCCESS 2 1 1 0.2 1.2
2 SUCCESS 2 2 1 0.2 1.2
4 SUCCESS 2 4 1 0.2 1.2
3 SUCCESS 2 3 1 0.2 1.2
5 SUCCESS 2 5 1 0.2 1.2
6 SUCCESS 3 6 1 0.2 1.2
7 SUCCESS 3 7 1 0.2 1.2
8 SUCCESS 3 8 1 0.2 1.2
10 SUCCESS 3 10 1 0.2 1.2
9 SUCCESS 3 9 1 0.2 1.2
11 SUCCESS 3 11 1 0.2 1.2
12 SUCCESS 2 0 1 1.2 2.2
13 SUCCESS 2 1 1 1.2 2.2
```

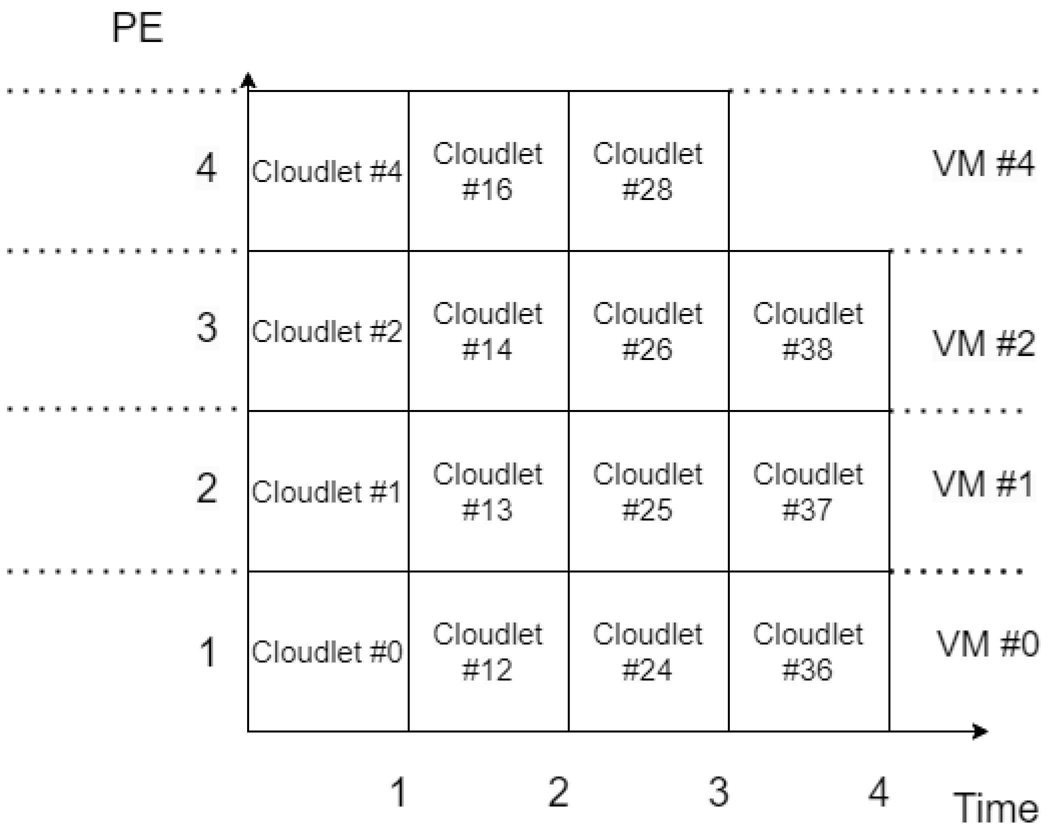
```
14 SUCCESS 2 2 1 1.2 2.2
16 SUCCESS 2 4 1 1.2 2.2
15 SUCCESS 2 3 1 1.2 2.2
17 SUCCESS 2 5 1 1.2 2.2
18 SUCCESS 3 6 1 1.2 2.2
19 SUCCESS 3 7 1 1.2 2.2
20 SUCCESS 3 8 1 1.2 2.2
22 SUCCESS 3 10 1 1.2 2.2
21 SUCCESS 3 9 1 1.2 2.2
23 SUCCESS 3 11 1 1.2 2.2
24 SUCCESS 2 0 1 2.2 3.2
25 SUCCESS 2 1 1 2.2 3.2
26 SUCCESS 2 2 1 2.2 3.2
28 SUCCESS 2 4 1 2.2 3.2
27 SUCCESS 2 3 1 2.2 3.2
29 SUCCESS 2 5 1 2.2 3.2
30 SUCCESS 3 6 1 2.2 3.2
31 SUCCESS 3 7 1 2.2 3.2
32 SUCCESS 3 8 1 2.2 3.2
34 SUCCESS 3 10 1 2.2 3.2
33 SUCCESS 3 9 1 2.2 3.2
35 SUCCESS 3 11 1 2.2 3.2
36 SUCCESS 2 0 1 3.2 4.2
37 SUCCESS 2 1 1 3.2 4.2
38 SUCCESS 2 2 1 3.2 4.2
39 SUCCESS 2 3 1 3.2 4.2
CloudSimExample 6 finished !
```

The timings are exactly the same as case 2 of the SC-TV policy. This is due to the **Space-Shared VM Scheduling** policy , where a VM occupies a processing element until it has completed all of its Cloudlets. If there is no more free processing item, the other VMs must wait until all Cloudlets of the current VM have finished. The Space-Shared Cloudlet Scheduling policy further increases the wait time for the next VM waiting to use the Host's processing element.

Let's take for example Host #0 of Datacenter #2.

Host #0 is a physical 4-core machine with a performance of 1000 MIPS per core. It hosts a total of 4 VMs that are 1-core with a performance of 1000 MIPS each. Based on the **First - Fit policy** , VM #0, #1, #2 are assigned 4 Cloudlets of size 1000 MI each, while VM #4 is assigned 3 Cloudlets of size 1000 MI each.

With the **Space - Shared routing policy Cloudlets Scheduling (SC)** and **Space - Shared VM Scheduling (SV)** , the Cloudlets do not run simultaneously on the VMs , nor do the VMs use the same core of the Host at the same time , as shown in **Figure A.2**



Picture A .2 Space-Shared Cloudlet Scheduling and Space-Shared VM Scheduling on Host #0 of Datacenter #2

(c) Time-Shared Cloudlet Scheduling & Space-Shared VM Scheduling (TC-SV)

Time-Shared Cloudlet Scheduling (**Time - Shared Cloudlets Scheduling**), is clearly mentioned in question 9 .

VM scheduling with space sharing (**Space - Shared VM Scheduling**), is clearly mentioned in case b.

The final part of the **CloudSimExample 6 simulation results** shows a table with the following columns (for an explanation of the simulation columns, see query 10)

CloudSimExample6/Output

```
========== OUTPUT ==========
Cloudlet ID STATUS Data center ID VM ID Time Start Time Finish Time
4 SUCCESS 2 4 3 0.2 3.2
16 SUCCESS 2 4 3 0.2 3.2
28 SUCCESS 2 4 3 0.2 3.2
5 SUCCESS 2 5 3 0.2 3.2
17 SUCCESS 2 5 3 0.2 3.2
29 SUCCESS 2 5 3 0.2 3.2
```

```
6 SUCCESS 3 6 3 0.2 3.2
18 SUCCESS 3 6 3 0.2 3.2
30 SUCCESS 3 6 3 0.2 3.2
7 SUCCESS 3 7 3 0.2 3.2
19 SUCCESS 3 7 3 0.2 3.2
31 SUCCESS 3 7 3 0.2 3.2
8 SUCCESS 3 8 3 0.2 3.2
20 SUCCESS 3 8 3 0.2 3.2
32 SUCCESS 3 8 3 0.2 3.2
10 SUCCESS 3 10 3 0.2 3.2
22 SUCCESS 3 10 3 0.2 3.2
34 SUCCESS 3 10 3 0.2 3.2
9 SUCCESS 3 9 3 0.2 3.2
21 SUCCESS 3 9 3 0.2 3.2
33 SUCCESS 3 9 3 0.2 3.2
11 SUCCESS 3 11 3 0.2 3.2
23 SUCCESS 3 11 3 0.2 3.2
35 SUCCESS 3 11 3 0.2 3.2
0 SUCCESS 2 0 4 0.2 4.2
12 SUCCESS 2 0 4 0.2 4.2
24 SUCCESS 2 0 4 0.2 4.2
36 SUCCESS 2 0 4 0.2 4.2
1 SUCCESS 2 1 4 0.2 4.2
13 SUCCESS 2 1 4 0.2 4.2
25 SUCCESS 2 1 4 0.2 4.2
37 SUCCESS 2 1 4 0.2 4.2
2 SUCCESS 2 2 4 0.2 4.2
14 SUCCESS 2 2 4 0.2 4.2
26 SUCCESS 2 2 4 0.2 4.2
38 SUCCESS 2 2 4 0.2 4.2
3 SUCCESS 2 3 4 0.2 4.2
15 SUCCESS 2 3 4 0.2 4.2
27 SUCCESS 2 3 4 0.2 4.2
39 SUCCESS 2 3 4 0.2 4.2
CloudSimExample 6 finished !
```

We note that the times are exactly the same as question 9 of the TC - TV policy . This is due to the **Space - Shared policy VM Scheduling** , where a VM uses a processing component until it has completed all of its Cloudlets . If no other processor is available, the other VMs must wait until the current VM finishes . In addition, the **Time - Shared policy Cloudlets Scheduling** makes the wait even longer for the next VM waiting to use the Host 's processing component .

Let's take for example Host #0 of Datacenter #2.

Host #0 is a physical 4-core machine with a performance of 1000 MIPS per core. It hosts a total of 4 VMs that are 1-core with a performance of 1000 MIPS each. Based on the **First - Fit policy** , VM #0, #1, #2 are assigned 4 Cloudlets of size 1000 MI each, while VM #4 is assigned 3 Cloudlets of size 1000 MI each.

Time - Shared routing policy **Cloudlets Scheduling (SC)** and **Space - Shared VM Scheduling (SV)** , the Cloudlets are executed simultaneously in the VMs , but the VMs do not simultaneously use the same core of the Host , as shown in **Figure A.3**

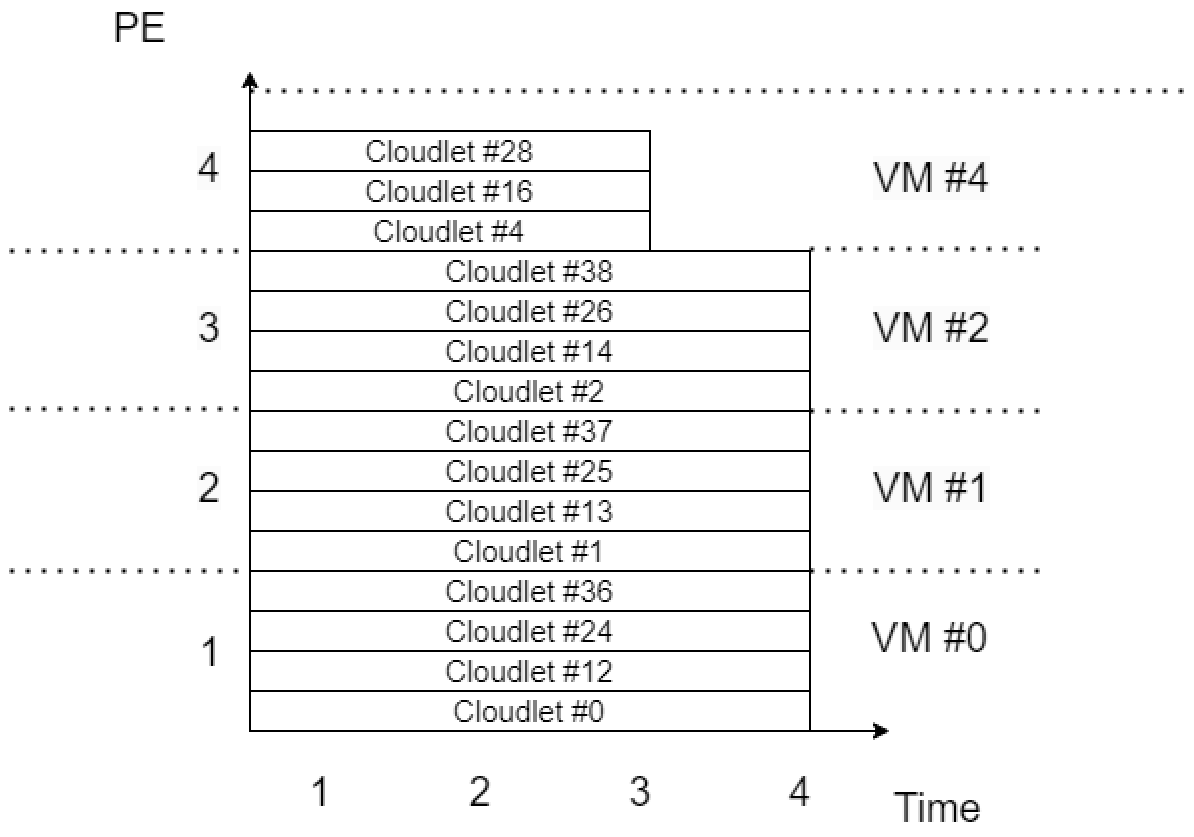


Figure A.3 Time-Shared Cloudlet Scheduling and Space-Shared VM Scheduling on Host #0 of Datacenter #2

(d) Time-Shared Cloudlet Scheduling & Time-Shared VM Scheduling with over-subscription

Time-Shared Cloudlet Scheduling (**Time - Shared Cloudlets Scheduling**), is clearly mentioned in question 9 .

Time-Shared VM scheduling (**Time - Shared VM Scheduling**), is clearly mentioned in question 9 . With the **over - subscription feature in the scheduler,** VM allocation to the Host is allowed even when it requires more processing capacity than is available. The class representing this algorithm is **VmSchedulerTimeSharedOverSubscription . java** located in the **org package . cloudbus . cloudsim** .

```
CloudSimExample6/createDatacenter(String name) : Datacenter
```

```
hostList .add(
                new Host(
                        hostId ,
                        new RamProvisionerSimple( ram ),
                        new BwProvisionerSimple( bw ),
                        storage ,
                        peList1 ,
                        new VmSchedulerTimeSharedOverSubscription( peList1 )
                )
        ); // This is our first machine

        hostId ++;

        hostList .add(
                new Host(
                        hostId ,
                        new RamProvisionerSimple( ram ),
                        new BwProvisionerSimple( bw ),
                        storage ,
                        peList2 ,
                        new VmSchedulerTimeSharedOverSubscription( peList2 )
                )
        ); // Second machine
```

The new assignment status of the 16 VMs that were successfully committed to the Datacenter Hosts is as follows:

2. **Datacenter_0 (#2)**

   - **Host #0**

     VM #0, #1, #2, #4 (Total 4 VMs running on this Host).

   - **Host #1**

     VM #3, #5, #6, #7 (Total 4 VMs running on this Host).

CloudSimExample6/Output

```
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: VM #6 has been created in Datacenter #2, Host #1
0.1: Broker: VM #7 has been created in Datacenter #2, Host #1
```

2. **Datacenter_1 (#3)**

   - **Host #0**

VM #8, #9, #10, #12 (Total 4 VMs running on this Host).

- **Host #1**

VM #11, #13, #14, #15 (Total 4 VMs running on this Host)

CloudSimExample6/Output

```
0.2: Broker: VM #8 has been created in Datacenter #3, Host #0
0.2: Broker: VM #9 has been created in Datacenter #3, Host #0
0.2: Broker: VM #10 has been created in Datacenter #3, Host #0
0.2: Broker: VM #11 has been created in Datacenter #3, Host #1
0.2: Broker: VM #12 has been created in Datacenter #3, Host #0
0.2: Broker: VM #13 has been created in Datacenter #3, Host #1
0.2: Broker: VM #14 has been created in Datacenter #3, Host #1
0.2: Broker: VM #15 has been created in Datacenter #3, Host #1
```

CloudSimExample 6 simulation results shows a table with the following columns (see query 10)

CloudSimExample6/Output

```
========== OUTPUT ==========
Cloudlet ID STATUS Data center ID VM ID Time Start Time Finish Time
8 SUCCESS 3 8 2 0.2 2.2
24 SUCCESS 3 8 2 0.2 2.2
9 SUCCESS 3 9 2 0.2 2.2
25 SUCCESS 3 9 2 0.2 2.2
10 SUCCESS 3 10 2 0.2 2.2
26 SUCCESS 3 10 2 0.2 2.2
12 SUCCESS 3 12 2 0.2 2.2
28 SUCCESS 3 12 2 0.2 2.2
0 SUCCESS 2 0 3 0.2 3.2
16 SUCCESS 2 0 3 0.2 3.2
32 SUCCESS 2 0 3 0.2 3.2
1 SUCCESS 2 1 3 0.2 3.2
17 SUCCESS 2 1 3 0.2 3.2
33 SUCCESS 2 1 3 0.2 3.2
2 SUCCESS 2 2 3 0.2 3.2
18 SUCCESS 2 2 3 0.2 3.2
34 SUCCESS 2 2 3 0.2 3.2
4 SUCCESS 2 4 3 0.2 3.2
20 SUCCESS 2 4 3 0.2 3.2
36 SUCCESS 2 4 3 0.2 3.2
11 SUCCESS 3 11 4 0.2 4.2
27 SUCCESS 3 11 4 0.2 4.2
13 SUCCESS 3 13 4 0.2 4.2
29 SUCCESS 3 13 4 0.2 4.2
14 SUCCESS 3 14 4 0.2 4.2
30 SUCCESS 3 14 4 0.2 4.2
15 SUCCESS 3 15 4 0.2 4.2
31 SUCCESS 3 15 4 0.2 4.2
3 SUCCESS 2 3 6 0.2 6.2
19 SUCCESS 2 3 6 0.2 6.2
35 SUCCESS 2 3 6 0.2 6.2
```

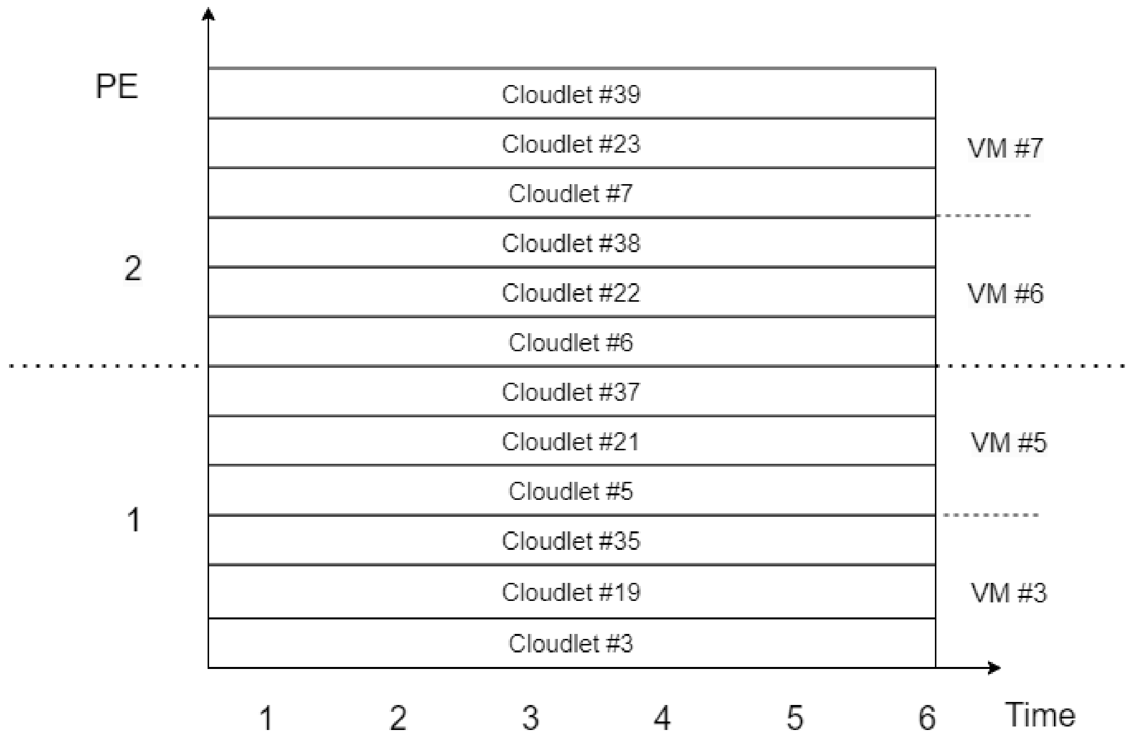
```
5 SUCCESS 2 5 6 0.2 6.2
21 SUCCESS 2 5 6 0.2 6.2
37 SUCCESS 2 5 6 0.2 6.2
6 SUCCESS 2 6 6 0.2 6.2
22 SUCCESS 2 6 6 0.2 6.2
38 SUCCESS 2 6 6 0.2 6.2
7 SUCCESS 2 7 6 0.2 6.2
23 SUCCESS 2 7 6 0.2 6.2
39 SUCCESS 2 7 6 0.2 6.2
CloudSimExample 6 finished !
```

The times for the Cloudlets of the VMs assigned to Host #0 of Datacenter #2 and #3, remain the same as also documented in question 9 for the **TC - TV** policy . We notice, however, that the execution times for the Cloudlets of the VMs assigned to Host #1 of Datacenter #2 and #3 have increased to 6.2 seconds . This is due to the increased allocation of Cloudlets to VMs and VMs to cores on Host #2, which hosts more VMs than it has processing elements (4 VMs and 2 PEs).

Let's take for example Host #1 of Datacenter #2.

Host #1 is a physical 2-core machine with a performance of 1000 MIPS per core. It hosts a total of 4 VMs that are 1-core with a performance of 1000 MIPS each. Based on the **First - Fit policy** , VMs #3, #5, #6, #7 are assigned 3 Cloudlets of size 1000 MI each.

Time - Shared routing policy **Cloudlets Scheduling** and **Time - Shared VM Scheduling over - subscription** , the Cloudlets run simultaneously on the VMs , and the VMs simultaneously use the same core of the Host , as shown in **Figure A.4.**



Picture A .4 Time-Shared Cloudlet Scheduling and Time-Shared VM Scheduling with over-subscription on Host #1 of Datacenter #2

B. Make the minimum required changes to the available infrastructure (hosts , PEs and their characteristics) so that all VMs can run (without over - subscription) . Give three versions, (a) one without increasing the number of hosts and PEs (i.e. increasing only their attributes), (b) one increasing the number of PEs (without increasing the number of hosts), and (c) one increasing the number of hosts . Study the output again, see if there are any differences or not, and explain why.

In order to be able to run all 20 VMs that are created, in the Hosts of the Datacenters there are 3 versions where we make minimal required changes to the available infrastructure (Hosts , PEs and their characteristics).

(a) change in characteristics of Hosts and PEs;

The changes we made to the characteristics of hosts and PEs were as follows:

1. We doubled the performance of PEs from 1000 MIPS to 2000 MIPS
2. RAM memory of each Host from 2048 MB (2 GB) to 4096 MB (4 GB)

```
CloudSimExample6/createDatacenter(String name) : Datacenter

int mips = 2000;

             // 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 = 4096; //host memory (MB)
             long storage = 1000000; //host storage
             int bw = 10000;
```

However, as answered in question 5 , not only the increase in RAM and processor capacity (MIPS) is sufficient, but also in other characteristics such as capacity (storage) and network bandwidth (bw).

The final part of the **CloudSimExample 6 simulation results** shows a table with the following columns (for an explanation of the simulation columns, see query 10)

CloudSimExample6/Output

```
========== OUTPUT ==========
Cloudlet ID STATUS Data center ID VM ID Time Start Time Finish Time
0 SUCCESS 2 0 2 0.2 2.2
20 SUCCESS 2 0 2 0.2 2.2
1 SUCCESS 2 1 2 0.2 2.2
21 SUCCESS 2 1 2 0.2 2.2
2 SUCCESS 2 2 2 0.2 2.2
22 SUCCESS 2 2 2 0.2 2.2
4 SUCCESS 2 4 2 0.2 2.2
24 SUCCESS 2 4 2 0.2 2.2
6 SUCCESS 2 6 2 0.2 2.2
26 SUCCESS 2 6 2 0.2 2.2
8 SUCCESS 2 8 2 0.2 2.2
28 SUCCESS 2 8 2 0.2 2.2
10 SUCCESS 2 10 2 0.2 2.2
30 SUCCESS 2 10 2 0.2 2.2
11 SUCCESS 2 11 2 0.2 2.2
31 SUCCESS 2 11 2 0.2 2.2
3 SUCCESS 2 3 2 0.2 2.2
23 SUCCESS 2 3 2 0.2 2.2
5 SUCCESS 2 5 2 0.2 2.2
25 SUCCESS 2 5 2 0.2 2.2
7 SUCCESS 2 7 2 0.2 2.2
27 SUCCESS 2 7 2 0.2 2.2
9 SUCCESS 2 9 2 0.2 2.2
29 SUCCESS 2 9 2 0.2 2.2
12 SUCCESS 3 12 2 0.2 2.2
32 SUCCESS 3 12 2 0.2 2.2
13 SUCCESS 3 13 2 0.2 2.2
33 SUCCESS 3 13 2 0.2 2.2
14 SUCCESS 3 14 2 0.2 2.2
34 SUCCESS 3 14 2 0.2 2.2
16 SUCCESS 3 16 2 0.2 2.2
36 SUCCESS 3 16 2 0.2 2.2
18 SUCCESS 3 18 2 0.2 2.2
38 SUCCESS 3 18 2 0.2 2.2
15 SUCCESS 3 15 2 0.2 2.2
35 SUCCESS 3 15 2 0.2 2.2
17 SUCCESS 3 17 2 0.2 2.2
37 SUCCESS 3 17 2 0.2 2.2
19 SUCCESS 3 19 2 0.2 2.2
39 SUCCESS 3 19 2 0.2 2.2
CloudSimExample 6 finished !
```

On the occasion of the allocation of Cloudlets to VMs with the **First - Fit method** (see question 8) and **Time - Shared** routing methods **Cloudlets** and **Time - Shared VM** , each VM has 2 Cloudlets of 1000 MI . VMs run Cloudlets in parallel and share the Host PE that hosts them, and with other VMs . Therefore, with parallel execution, the performance of the PE is shared for the execution of the Cloudlets and given that each Cloudlet needs 1 second for its execution in the PE of the VM , and each VM has 2 Cloudlets , the execution time of each Cloudlet is well justified in 2 seconds .

(b) change in the number of PEs

The changes we made to the number of PEs were to increase the corresponding number of Host #1 from 2 PEs to 4 PEs .

CloudSimExample6/createDatacenter(String name) : Datacenter

```
// 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 )));
             peList2 .add( new Pe(2, new PeProvisionerSimple( mips )));
             peList2 .add( new Pe(3, new PeProvisionerSimple( mips )));
```

However, it does not make it possible to run all the VMs on the Host , even if we increase the number of PEs , and this is due to the limited RAM memory (2048 MB). So the maximum we can achieve is 16/20 VMs running on Hosts .

CloudSimExample6/Output

```
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: 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: VM #6 has been created in Datacenter #2, Host #0
0.1: Broker: VM #7 has been created in Datacenter #2, Host #1
...
0.2: Broker: VM #8 has been created in Datacenter #3, Host #0
0.2: Broker: VM #9 has been created in Datacenter #3, Host #1
0.2: Broker: VM #10 has been created in Datacenter #3, Host #0
0.2: Broker: VM #11 has been created in Datacenter #3, Host #1
0.2: Broker: VM #12 has been created in Datacenter #3, Host #0
0.2: Broker: VM #13 has been created in Datacenter #3, Host #1
0.2: Broker: VM #14 has been created in Datacenter #3, Host #0
0.2: Broker: VM #15 has been created in Datacenter #3, Host #1
```

Assume that each Host has sufficient processing capacity to host multiple VMs . Each VM has 512 MB of RAM and each Host has 2048 MB , therefore each Host can host a maximum of 4 VMs . In total, 4 Hosts are created , 2 for each Datacenter . Therefore, the RAM limitation leads to the conclusion that the maximum number of hosted VMs cannot exceed 16, no matter how many PEs we add.

CloudSimExample6/Output

```
[VmScheduler.vmCreate] Allocation of VM #16 to Host #0 failed by RAM
[VmScheduler.vmCreate] Allocation of VM #16 to Host #1 failed by RAM
[VmScheduler.vmCreate] Allocation of VM #17 to Host #0 failed by RAM
[VmScheduler.vmCreate] Allocation of VM #17 to Host #1 failed by RAM
[VmScheduler.vmCreate] Allocation of VM #18 to Host #0 failed by RAM
[VmScheduler.vmCreate] Allocation of VM #18 to Host #1 failed by RAM
[VmScheduler.vmCreate] Allocation of VM #19 to Host #0 failed by RAM
[VmScheduler.vmCreate] Allocation of VM #19 to Host #1 failed by RAM
```

The final part of the **CloudSimExample 6 simulation results** shows a table with the following columns (for an explanation of the simulation columns, see query 10)

CloudSimExample6/Output

```
========== OUTPUT ==========
Cloudlet ID STATUS Data center ID VM ID Time Start Time Finish Time
8 SUCCESS 3 8 2 0.2 2.2
24 SUCCESS 3 8 2 0.2 2.2
10 SUCCESS 3 10 2 0.2 2.2
26 SUCCESS 3 10 2 0.2 2.2
12 SUCCESS 3 12 2 0.2 2.2
28 SUCCESS 3 12 2 0.2 2.2
14 SUCCESS 3 14 2 0.2 2.2
30 SUCCESS 3 14 2 0.2 2.2
9 SUCCESS 3 9 2 0.2 2.2
25 SUCCESS 3 9 2 0.2 2.2
11 SUCCESS 3 11 2 0.2 2.2
27 SUCCESS 3 11 2 0.2 2.2
13 SUCCESS 3 13 2 0.2 2.2
29 SUCCESS 3 13 2 0.2 2.2
15 SUCCESS 3 15 2 0.2 2.2
31 SUCCESS 3 15 2 0.2 2.2
0 SUCCESS 2 0 3 0.2 3.2
16 SUCCESS 2 0 3 0.2 3.2
32 SUCCESS 2 0 3 0.2 3.2
2 SUCCESS 2 2 3 0.2 3.2
18 SUCCESS 2 2 3 0.2 3.2
34 SUCCESS 2 2 3 0.2 3.2
4 SUCCESS 2 4 3 0.2 3.2
20 SUCCESS 2 4 3 0.2 3.2
36 SUCCESS 2 4 3 0.2 3.2
6 SUCCESS 2 6 3 0.2 3.2
22 SUCCESS 2 6 3 0.2 3.2
38 SUCCESS 2 6 3 0.2 3.2
1 SUCCESS 2 1 3 0.2 3.2
17 SUCCESS 2 1 3 0.2 3.2
33 SUCCESS 2 1 3 0.2 3.2
3 SUCCESS 2 3 3 0.2 3.2
19 SUCCESS 2 3 3 0.2 3.2
35 SUCCESS 2 3 3 0.2 3.2
5 SUCCESS 2 5 3 0.2 3.2
21 SUCCESS 2 5 3 0.2 3.2
37 SUCCESS 2 5 3 0.2 3.2
7 SUCCESS 2 7 3 0.2 3.2
23 SUCCESS 2 7 3 0.2 3.2
39 SUCCESS 2 7 3 0.2 3.2
CloudSimExample 6 finished !
```

On the occasion of the allocation of Cloudlets to VMs with the **First - Fit method** (see question 8) and **Time - Shared** routing methods **Cloudlets** and **Time - Shared VM** , each VM has 2 Cloudlets of 1000 MI except for 4 which have 3 Cloudlets . VMs run Cloudlets in parallel and share the Host PE that hosts them, and with other VMs . Therefore, with parallel execution, the performance of the PE is shared for the execution of the Cloudlets and given that each Cloudlet needs 1 second for its execution in the PE of the VM , and each VM has 2 Cloudlets , the execution time of each Cloudlet is justified correctly in 2 seconds and correspondingly 3 seconds in the VMs that have 3 Cloudlets .

(c) change in the number of Hosts

The changes we made were in the increase of Hosts from 2 to 3. It is worth noting that the 3rd $^{\text{Host}}$ has 4 PEs of 1000 MIPS . So , then , the each Datacenter has from 3 hosts.

```
CloudSimExample6/createDatacenter(String name) : Datacenter


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

             hostId ++;

             hostList .add(
                    new Host(
                           hostId ,
                           new RamProvisionerSimple( ram ),
                           new BwProvisionerSimple( bw ),
                           storage ,
                           peList1 ,
                           new VmSchedulerTimeShared( peList1 )
                    )
             ); // This is our third machine
```

It is also worth noting that the increase in the number of Hosts was not sufficient in the event that the 3rd Host owned 2 PEs of 1000 MIPS instead of 4, which points out the direct dependency of Hosts with PEs .

CloudSimExample6/Output

```
[VmScheduler.vmCreate] Allocation of VM #6 to Host #2 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #7 to Host #2 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #8 to Host #2 failed by MIPS
```

The final part of the **CloudSimExample 6 simulation results** shows a table with the following columns (for an explanation of the simulation columns, see query 10)

CloudSimExample6/Output

```
========== OUTPUT ==========
Cloudlet ID STATUS Data center ID VM ID Time Start Time Finish Time
0 SUCCESS 2 0 2 0.2 2.2
20 SUCCESS 2 0 2 0.2 2.2
2 SUCCESS 2 2 2 0.2 2.2
22 SUCCESS 2 2 2 0.2 2.2
4 SUCCESS 2 4 2 0.2 2.2
24 SUCCESS 2 4 2 0.2 2.2
7 SUCCESS 2 7 2 0.2 2.2
27 SUCCESS 2 7 2 0.2 2.2
5 SUCCESS 2 5 2 0.2 2.2
25 SUCCESS 2 5 2 0.2 2.2
8 SUCCESS 2 8 2 0.2 2.2
28 SUCCESS 2 8 2 0.2 2.2
1 SUCCESS 2 1 2 0.2 2.2
21 SUCCESS 2 1 2 0.2 2.2
3 SUCCESS 2 3 2 0.2 2.2
23 SUCCESS 2 3 2 0.2 2.2
6 SUCCESS 2 6 2 0.2 2.2
26 SUCCESS 2 6 2 0.2 2.2
9 SUCCESS 2 9 2 0.2 2.2
29 SUCCESS 2 9 2 0.2 2.2
10 SUCCESS 3 10 2 0.2 2.2
30 SUCCESS 3 10 2 0.2 2.2
12 SUCCESS 3 12 2 0.2 2.2
32 SUCCESS 3 12 2 0.2 2.2
14 SUCCESS 3 14 2 0.2 2.2
34 SUCCESS 3 14 2 0.2 2.2
17 SUCCESS 3 17 2 0.2 2.2
37 SUCCESS 3 17 2 0.2 2.2
15 SUCCESS 3 15 2 0.2 2.2
35 SUCCESS 3 15 2 0.2 2.2
18 SUCCESS 3 18 2 0.2 2.2
38 SUCCESS 3 18 2 0.2 2.2
11 SUCCESS 3 11 2 0.2 2.2
31 SUCCESS 3 11 2 0.2 2.2
13 SUCCESS 3 13 2 0.2 2.2
33 SUCCESS 3 13 2 0.2 2.2
16 SUCCESS 3 16 2 0.2 2.2
36 SUCCESS 3 16 2 0.2 2.2
19 SUCCESS 3 19 2 0.2 2.2
39 SUCCESS 3 19 2 0.2 2.2
```

```
CloudSimExample 6 finished !
```

The output of the simulation remains the same as in case a.

C. Where in the code (simulator implementation) and how would you intervene (a) to change the policy / decision algorithm for 5. and 8., and (b) to add some other policy of your own for 9 (a) and 9(b); In particular with regard to 5. (the policy followed i.e. for assigning VMs to Hosts), explain in more detail how you would implement another policy of your choice – search and select another specific policy (consult, among others, the attached links and papers – see folder ' VM Allocation Policies '), describe it briefly, and then try to implement it.

(a) changing the assignment policy of VMs to Hosts , and Cloudlets to VMs

The decision policy that assigns VMs to Hosts is implemented in the **VmAllocationPolicySimple** class **. java** located in the package **org . cloudbus . cloudsim** .

```
CloudSimExample6/createDatacenter(String name) : Datacenter

// 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();
             }
```

If we wanted to change this policy we would create a new class that would have another decision policy for assigning VMs to Hosts . The new class would inherit the **VmAllocationPolicy class** that uses the **allocateHostForVm () method** and thus we would be able to implement the new idea of VM allocation to Hosts , in this method.

```
VmAllocationPolicySimple/allocateHostForVm(Vm vm) : boolean

public boolean allocateHostForVm(Vm vm ) {
             int requiredPes = vm .getNumberOfPes();
             boolean result = false ;
             int tries = 0;
             List<Integer> freePesTmp = new ArrayList<Integer>();
             for (Integer freePes : getFreePes()) {
                    freePesTmp .add( freePes );
             }

             if (!getVmTable().containsKey( vm .getUid())) { // if this vm was not
created
```

```
                    do { // we still trying until we find a host or until we try all of
them
                          int moreFree = Integer. MIN_VALUE ;
                          int idx = -1;

                          // we want the host with less pes in use
                          for ( int i = 0; i < freePesTmp .size(); i ++) {
                                if ( freePesTmp .get( i ) > moreFree ) {
                                      moreFree = freePesTmp .get( i );
                                      idx = i ;
                                }
                          }

                          Host host = getHostList().get( idx );
                          result = host .vmCreate( vm );

                          if ( result ) { // if vm were successfully created in the host
                                getVmTable().put( vm .getUid(), host );
                                getUsedPes().put( vm .getUid(), requiredPes );
                                getFreePes().set( idx , getFreePes().get( idx ) -
requiredPes );
                                result = true ;
                                break
                          } else {
                                freePesTmp .set( idx , Integer. MIN_VALUE );
                          }
                          tries ++;
                    } while (! result && tries < getFreePes().size());

              }

              return result ;
       }
```

OR policy decision where assigns Cloudlets on VMs is implemented in class **DatacenterBroker.java** which is located at packet **org.cloudbus.cloudsim** .

```
CloudsimExample6/main(String[] args) : void

//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 ,20); //creating 20 vms
                    cloudletList = createCloudlet ( brokerId ,40); // creating 40
cloudlets

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

If we wanted to change the policy of assigning Cloudlets to VMs we would intervene in the **submitCloudlets () method of the DatacenterBroker** class . The broker entity is an entity that manages the creation of VMs , the assignment of Cloudlets to VMs , and the destruction of VMs after the simulation is complete.

```
DatacenterBroker/submitCloudlets() : void
```

```
protected void submitCloudlets () {
                int vmIndex = 0;
                List<Cloudlet> successfullySubmitted = new ArrayList<Cloudlet>();
                for (Cloudlet cloudlet : getCloudletList()) {
                        Vm vm ?
                        // if user didn't bind this cloudlet and it hasn't been executed yet
                        if ( cloudlet .getVmId() == -1) {
                                vm = getVmsCreatedList().get( vmIndex );
                        } else { // submit to the specific vm
                                vm = VmList. getById (getVmsCreatedList(), cloudlet
.getVmId());
                                if ( vm == null ) { // vm was not created
                                        if (!Log. isDisabled ()) {
                                        Log. printConcatLine (CloudSim. clock (), ": " ,
getName(), ": Postponing execution of cloudlet " ,
                                                        cloudlet .getCloudletId(), ": bount VM not
available" );
                                        }
                                        continue ;
                                }
                        }

                        if (!Log. isDisabled ()) {
                        Log. printConcatLine (CloudSim. clock (), ": " , getName(), ":
Sending cloudlet " ,
                                        cloudlet .getCloudletId(), " to VM #" , vm .getId());
                        }

                        cloudlet .setVmId( vm .getId());
                        sendNow(getVmsToDatacentersMap().get( vm .getId()), CloudSimTags.
CLOUDLET_SUBMIT , cloudlet );
                        cloudletsSubmitted ++;
                        vmIndex = ( vmIndex + 1) % getVmsCreatedList().size();
                        getCloudletSubmittedList().add( cloudlet );
                        successfullySubmitted .add( cloudlet );
                }

                // remove submitted cloudlets from waiting list
                getCloudletList().removeAll( successfullySubmitted );
        }
```

(b) addition new policy routing of VMs on Host PEs and of Cloudlets to PEs of VMs

The routing policy that schedules VMs on Host PEs is implemented in the **VmScheduler** class **. java** located in the package **org . cloudbus . cloudsim** .

```
CloudSimExample6/createDatacenter(String name) : Datacenter
```

```
hostList .add(
                        new Host(
                                hostId ,
                                new RamProvisionerSimple( ram ),
                                new BwProvisionerSimple( bw ),
                                storage ,
                                peList1 ,
                    new VmSchedulerTimeShared( peList1 )
               )
          ); // This is our first machine
```

If we wanted to add a new routing policy we would create a new class that would inherit **VmScheduler** which uses the **allocatePesForVm() method** and thus we would have the possibility to implement the new idea of VM routing to Host PEs , in this method.

```
VmSchedulerTimeShared/allocatePesForVm(String vmUid, List<Double> mipsSharedRequested) :
boolean

protected boolean allocatePesForVm(String vmUid , List<Double> mipsShareRequested ) {
          double totalRequestedMips = 0;
          double peMips = getPeCapacity();
          for (Double mips : mipsShareRequested ) {
               // each virtual PE of a VM must require no more than the capacity of
a physical PE
               if ( mips > peMips ) {
                    return false ;
               }
               totalRequestedMips += mips ;
          }

          // This scheduler does not allow over-subscription
          if (getAvailableMips() < totalRequestedMips ) {
               return false ;
          }

          getMipsMapRequested().put( vmUid , mipsShareRequested );
          setPesInUse(getPesInUse() + mipsShareRequested .size());

          if (getVmsMigratingIn().contains( vmUid )) {
               // the destination host only experiences 10% of the migrating VM's
MIPS
               totalRequestedMips *= 0.1;
          }

          List<Double> mipsShareAllocated = new ArrayList<Double>();
          for (Double mipsRequested : mipsShareRequested ) {
               if (getVmsMigratingOut().contains( vmUid )) {
                    // performance degradation due to migration = 10% MIPS
                    mipsRequested *= 0.9;
               } else if (getVmsMigratingIn().contains( vmUid )) {
                    // the destination host only experiences 10% of the migrating
VM's MIPS
                    mipsRequested *= 0.1;
               }
               mipsShareAllocated .add( mipsRequested );
          }

          getMipsMap().put( vmUid , mipsShareAllocated );
          setAvailableMips(getAvailableMips() - totalRequestedMips );

          return true ;
     }
```

The routing policy that schedules Cloudlets to VM PEs is implemented in the **CloudletScheduler** class **. java** located in the package **org . cloudbus . cloudsim** .

```
CloudSimExample6/createVM(int userId, int vms) : List<Vm>
```

```
//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 CloudletSchedulerTimeShared());
			//for creating a VM with a space shared scheduling policy for
cloudlets :
			// vm [i] = new Vm (i, userId, mips , pesNumber, ram, bw , size, vmm
, new CloudletSchedulerSpaceShared());

			list . add ( vm [ i ]);
		}
```

Cloudlets routing policy to VMs we would create a new class that inherits **CloudletScheduler** which uses the **updateVmProcessing () method** and thus we would have the possibility to implement the new idea of routing Cloudlets to the PEs of the VMs , in this method.

```
CloudletSchedulerTimeShared/updateVmProcessing(double currentTime, List<Double>
mipsShare) : double
```

```
@Override
	public double updateVmProcessing ( double currentTime , List<Double> mipsShare ) {
		setCurrentMipsShare( mipsShare );
		double timeSpam = currentTime - getPreviousTime();

		for (ResCloudlet rcl : getCloudletExecList()) {
			rcl .updateCloudletFinishedSoFar(( long ) (getCapacity( mipsShare )
* timeSpam * rcl .getNumberOfPes() * Consts. MILLION ));
		}

		if (getCloudletExecList().size() == 0) {
			setPreviousTime( currentTime );
			return 0.0;
		}

		// check finished cloudlets
		double nextEvent = Double. MAX_VALUE ;
		List<ResCloudlet> toRemove = new ArrayList<ResCloudlet>();
		for (ResCloudlet rcl : getCloudletExecList()) {
			long remainingLength = rcl .getRemainingCloudletLength();
			if ( remainingLength == 0 ) { // finished: remove from the list
				toRemove .add( rcl );
				cloudletFinish( rcl );
				continue ;
			}
		}
		getCloudletExecList().removeAll( toRemove );

		// estimate finish time of cloudlets
		for (ResCloudlet rcl : getCloudletExecList()) {
```

```
                    double estimatedFinishTime = currentTime
                                 + ( rcl .getRemainingCloudletLength() / (getCapacity(
mipsShare ) * rcl .getNumberOfPes()));
                    if ( estimatedFinishTime - currentTime < CloudSim .
getMinTimeBetweenEvents ()) {
                          estimatedFinishTime = currentTime + CloudSim .
getMinTimeBetweenEvents ();
                    }

                    if ( estimatedFinishTime < nextEvent ) {
                          nextEvent = estimatedFinishTime ;
                    }
              }

              setPreviousTime( currentTime );
              return nextEvent ;
       }
```

As mentioned above, to change the allocation policy of VMs to Hosts , we will create a new class that inherits **VmAllocationPolicy** , in order to intervene in the **allocateHostForVm () method** that handles the policy for assigning VMs to Hosts .

The new policy we will use is **Round - Robin** policy. This policy assigns VMs to Hosts in round robin order. Each VM request is assigned to the next entity_id in order Host that accepts the VM , as long as its available resources are sufficient for hosting. The **Round - Robin policy** ensures an even distribution of VMs across Hosts preventing any individual Host from being overloaded .

To implement the policy, we first added the **CircularHostList class . java** in package **org . cloudbus . cloudsim** from the repository found at the link below:

https://github.com/AnanthaRajuC/CloudSim-Round-Robin/tree/master/src/cloudsim_round_robin

This class inherits the **Iterable < T > class** which enables the object (in our case the Host) to be the target of the enhanced for command . Thus, having the Hosts stored in a linked list (**LinkedList < Host >**) we can retrieve them by accessing them in circular order.

```
CircularHostList
```

```
public final class CircularHostList implements Iterable<Host> {

      private final List<Host> host_list = new LinkedList<Host>();

      private int ini ;

      public CircularHostList(List<? extends Host> hosts ) {
            this . host_list .addAll( hosts );
      }

      public boolean add(Host host ) {
            return this . host_list .add( host );
      }

      public boolean remove(Host host2Remove ) {
            return this . host_list .remove( host2Remove );
      }
```

```
	public Host next() {
		Host host = null ;

		if (! host_list . isEmpty()) {
			int index = ( this . ini ++ % this . host_list . size());
			host = this . host_list .get( index );
		}

		return host ;
	}
```

At so on , we added her class **RoundRobinVmAllocationPolicy** at packet **org.cloudbus.cloudsim** from the repository where is located to link :

https://gist.github.com/alessandroleite/4598040

The class adheres to what we mentioned above, that is, in the **allocateHostForVm () method** that inherits from the **VmAllocationPolicy class, it applies the Round - Robin** policy for assigning VMs to Hosts .

```
RoundRobinVmAllocationPolicy/allocateHostForVm(Vm vm, Host host) : boolean

public boolean allocateHostForVm(Vm vm , Host host )
	{
		if ( host != null && host .vmCreate( vm ))
		{
			vm_table .put( vm .getUid(), host );
			Log. formatLine ( "%.4f: VM #" + vm .getId() + " has been allocated 
to the host#" + host .getId() +
					" datacenter #" + host .getDatacenter().getId() + "(" + 
host .getDatacenter().getName() + ") #" ,
					CloudSim. clock ());
			return true ;
		}
		return false ;
	}
```

The simulation results for Datacenter #2 Hosts are as follows :

```
CloudSimExample6/Output

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: 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: Creation of VM #5 failed in Datacenter #2
0.1: Broker: VM #6 has been created in Datacenter #2, Host #0
```

The simulation results for Datacenter #3 Hosts are as follows :

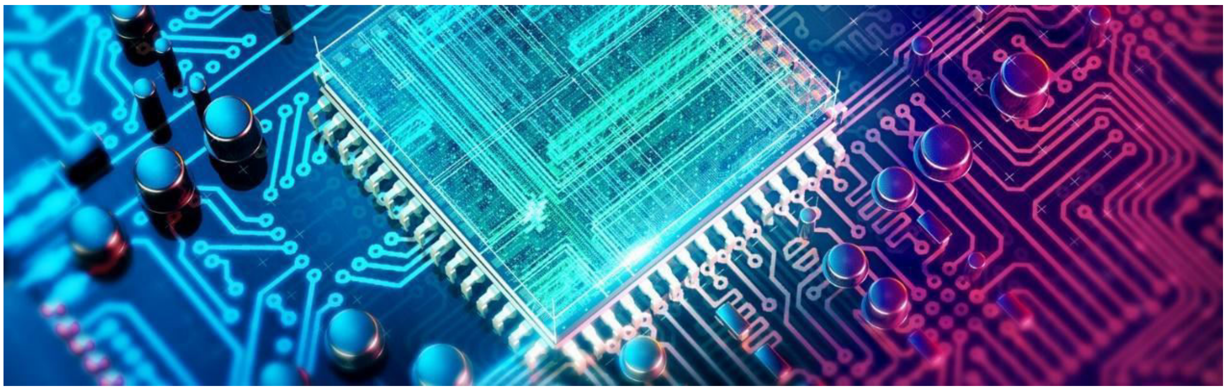
```
CloudSimExample6/Output

0.2: Broker: VM #5 has been created in Datacenter #3, Host #0
```

```
0.2: Broker: VM #7 has been created in Datacenter #3, Host #1
0.2: Broker: VM #8 has been created in Datacenter #3, Host #0
0.2: Broker: VM #9 has been created in Datacenter #3, Host #1
0.2: Broker: VM #10 has been created in Datacenter #3, Host #0
0.2: Broker: Creation of VM #11 failed in Datacenter #3
0.2: Broker: VM #12 has been created in Datacenter #3, Host #0
```

As we can see, the VMs are alternately distributed to the available Hosts of each Datacenter , due to the circular order adopted by the **Round - Robin policy** . It is worth noting that in the event that a Host cannot serve the VM 's request due to limited available resources, the next VM sends a request to the next Host in the round robin.

bpy.context.scene.objects.active = modifier_ob
print("Selected" + str(modifier_ob)) # modifier

Thank you for your attention.



CLOUD COMPUTING AND SERVICES