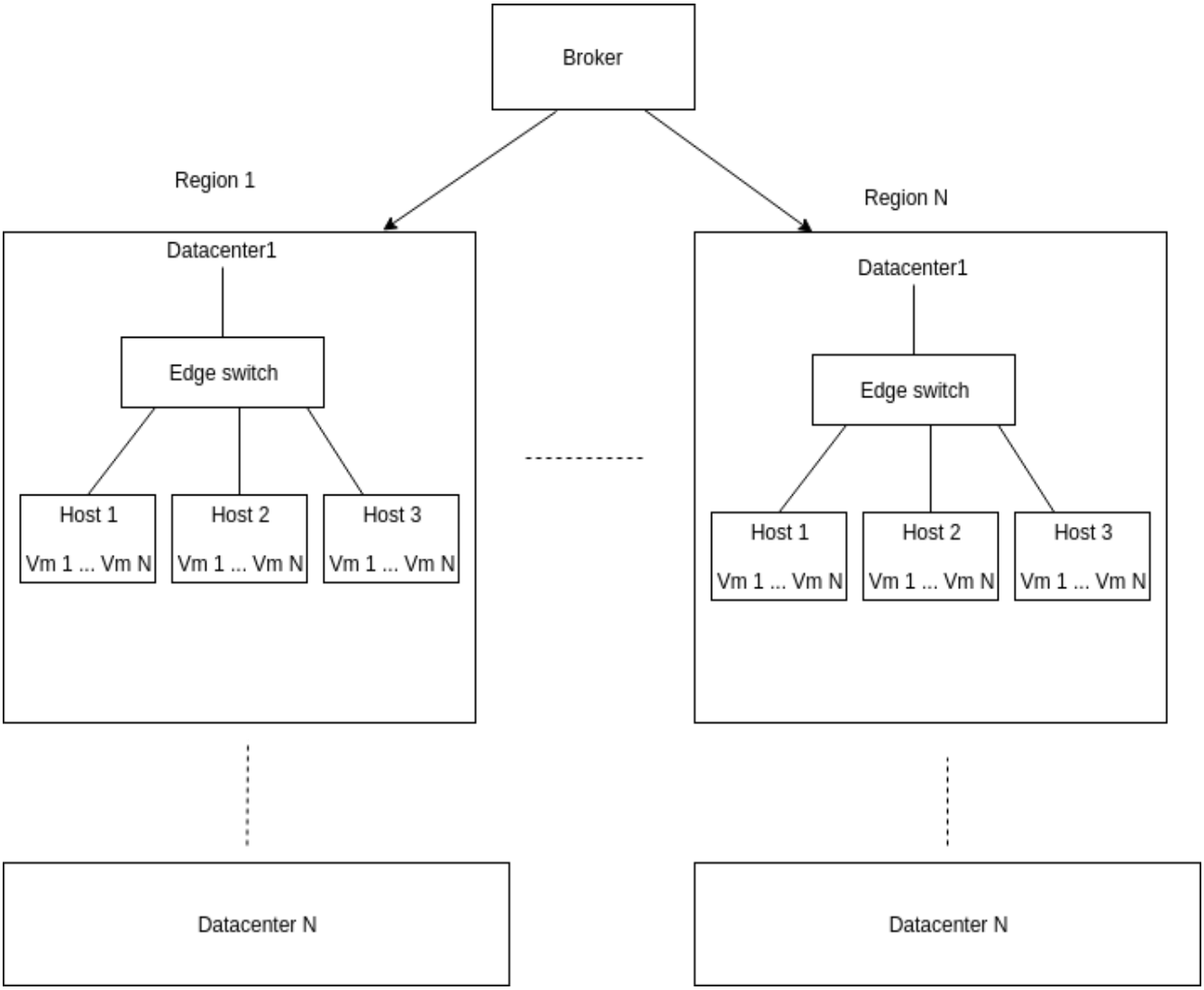
Course Project

Group Members: Adarsh Hegde, Amrish Jhaveri, Chinmay Gangal, Karan Kadakia

Implemented Network Architecture



Architecture Overview:

We have kept 7 regions and each region has a specific number of hosts that we have defined in the configuration file similarly Vm values in the hosts are assigned through configuration file.

Design decisions

- We extended cloudsim plus entities to add Region properties so as to improve allocation of cloudlets to vm. Each region has its own set of datacenters.
- Enumeration : We used Enumeration class for defining **regions** for type safety, limiting inputs and quicker comparison
- Functional programming: We made use of streams, fold, flatmap, filters, map, lamda
- Functions are made implicit so that they can be statically imported across the project, Generator
- Policy we extended cloudsim plus policies to define our policies
- Functional interfaces: Our policies are functions itself so functions are passed as parameters

Docker using SBT native packager

SBT native packager lets you build application packages in native formats and offers different archetypes for common configurations, such as simple Java apps or server applications.

Docker images describe how to set up a container for running an application, including what files are present, and what program to run.

Docker commands to build image and container

1. The following generates a docker image locally

sbt docker: publishLocal

2. The following publishes a docker image to the hub (Repository adarsh23)

sbt docker: publish

3. Creates a container when run within the project root directory

docker-compose up

How to run using global docker image

a. Install docker

b. docker pull adarsh23/regionalcloudsim:latest

c. docker run adarsh23/regionalcloudsim

The above command will execute the app

Description of how we designed and implemented the simulation

- We have designed multiple Datacenters in the form of an array in our simulation each region has multiple Datacenters with a particular number of hosts. Region is defined as follows(dcList below is for the list of datacenters):

```
{
  name = "Region1"
  dcList = [{
     noOfHosts = 10
   }]
}
```

- Each datacenter is connected by an Edge switch which can be connected to NetworkHosts which interacts with other Datacenter in order to exchange packets.
- When a datacenter is created the hosts are assigned to it from the configuration file based on the Region value. Each Region is assigned some number of hosts in the configuration file as follows: (For example)

```
hostValues {
  ram = 15000
  storage = 1024
  bw = 50
  noOfPes = 10
  mips = 1000
}
```

- Now as we run the simulation Network cloudlets are assigned on Vms based on a new policy that we created called the VMAllocationPolicyPercentPes which is explained in detail below. CloudletExecutionTask, CloudletSendTask, CloudletReceiveTask are a part of the Network cloudlet execution

- Vm feature values are assigned based on the configuration file

Policies Implemented

- VM Allocation policy: VmAllocationPolicy(PercentPes, Best fit, First fit, Simple)
- VM Scheduler policy: VmScheduler (SpaceShared, TimeShared)
- VM to Cloudlet mapping: VmToCloudletMapping (TimeMinimized, MinPes)
- Cloudlet Scheduler Policy: CloudletScheduler (TimeShared, Space shared)
- Utilization model: UtilizationModel (Dynamic, Full, Stochastic)

Each of these policies have been specified in the configuration file and can be assigned by changing the configuration file.

How we planned the simulation work and how we compared various algorithms

We looked into various vm allocation policies simple, best fit, first fit and tried to compare the various results of vm allocation based on the features of the policy.

Next we tried to understand which policy is suitable for which scenario and started customizing certain features in the policies

Upon research we found that the VmAllocationPolicySimple could be customized and turned into VmAllocationPolicyPercentPes which would allocate vms based on the % of Pes available

VMAllocationPolicyPercentagePes

Details:

We developed our own Vm Allocation Policies called VMAllocationPolicyPercentagePes which assigns vm's based on the maximum percentage of the PE's available in a host.

Here, percentage of PE's available for a host are calculated as follows:

% of free PE's in a host = **{ (Total free PE in a particular host H1)÷(Total no PE in host H1) }*100**

For example:

Suppose there are two hosts:

1) Host A: Pes remaining = 15, Total Pes = 30 ; Thus, % of Pes = 50%

2) Host B: Pes remaining = 4, Total Pes = 6; Thus % of Pes = 66%

In this case VMAllocationPolicyPercentagePes selects Host B(if it is compatible) as it has more % Pes available compared to Host A.

Whereas, VMAllocationPolicySimple would simply select Host A as it has more number of Pes available.

This clearly distributes the load evenly across the hosts.

Advantages of using this policy:
1. It can be helpful to reduce the no of active hosts at a time and thus increase availability
2. It can help divide the load amongs the hosts equally
2. Instead of just assigning a host with higher Pes available it assigns the one with enough maximum % PE available and thus assigns vms on that host

Code Snippets:

VMAllocationPolicyPercentagePes Code snippet:

- It can be seen in this policy that the percentage of the free Pes are calculated with the help of host1.getFreePesNumber / host1.getNumberOfPes as compared to the VMAllocationPolicySimple which simply assigns based on the maximum free Pes available.

```
override def defaultFindHostForVm(vm: Vm): Optional[Host] = {
  Optional.ofNullable(getHostList[Host].asScala.toList
    .filter(host => host.isSuitableForVm(vm))
    .sortBy(host => host.isActive)
    .reverse
    .sortWith((host1, host2) => (host1.getFreePesNumber / host1.getNumberOfPes)
      > (host2.getFreePesNumber / host2.getNumberOfPes)).headOption.orNull)
}
```

VMAllocationPolicySimple Code snippet:

```
protected Optional<Host> defaultFindHostForVm(final Vm vm) {
    final Comparator<Host> comparator = Comparator.comparing(Host::isActive)
                                        .thenComparingLong(Host::getFreePesNumber);
    final Stream<Host> stream = isParallelHostSearchEnabled() ? getHostList().stream().parallel() :
getHostList().stream();
    return stream
            .filter(host -> host.isSuitableForVm(vm))
            .max(comparator);
}
```

Test cases

The code has been thoroughly tested with the help of Scala test which involves unit tests as well as Integration test

GeneratorTest: Unit Tests

ConfigReaderTest: Unit Tests

AppTest: Integration test which tests the simulation end to end

Limitations we encountered in NetworkCloudlets

Cloudsim plus working of the NetworkCloudlets is limited in terms of the design. Network cloudlets consist of list of ExecutionTasks, CloudletSendTask and CloudletReceiveTask. The limitation is that each cloudlet needs to be assigned a VM before tasks can be assigned to the cloudlet. Since we wanted to use the logic of sending and receiving cloudlet tasks, we had to provide a mapping of Vms to cloudlets before starting the simualtion. This restricted our design to static assignment of Vms to clodulets.

Screenshots of Execution

1. VmAllocationPercentPes

It can be seen below that PercentPes considers the percentage of Pes and thus assigns vm to host 1 first then since host 2 has more % Pes it assign vm to host 2 and then again host 1 and host 2:

Thus, host 1 → host 2 → host 1→ host 2

```
18:44:56.318 [main] INFO  VmAllocationPolicyAbstract - 0.0: VMAllocationPolicyPercentagePes: Vm 0/Broker 2 has been allocated to Host
 1/DC 1
18:44:56.326 [main] INFO  VmAllocationPolicyAbstract - 0.0: VMAllocationPolicyPercentagePes: Vm 1/Broker 2 has been allocated to Host
 0/DC 1
18:44:56.328 [main] INFO  VmAllocationPolicyAbstract - 0.0: VMAllocationPolicyPercentagePes: Vm 2/Broker 2 has been allocated to Host
 1/DC 1
18:44:56.329 [main] INFO  VmAllocationPolicyAbstract - 0.0: VMAllocationPolicyPercentagePes: Vm 3/Broker 2 has been allocated to Host
 0/DC 1
```

2. Output of running the app on Docker

docker run -it adarsh23/regionalcloudsim:latest

SIMULATION RESULTS

Cloudlet ID	Status	DC ID	Host ID	Host PEs CPU cores	VM ID	VM PEs CPU cores	CloudletLen MI	CloudletPEs CPU cores	StartTime Seconds	FinishTime Seconds	ExecTime Seconds
3	SUCCESS	1	6	10	4	2	1000	1	0	2	2
4	SUCCESS	1	6	10	4	2	1000	1	0	2	2
6	SUCCESS	1	6	10	4	2	1000	1	0	2	2
5	SUCCESS	1	8	10	2	2	1000	1	0	2	2

3. Screenshot of results

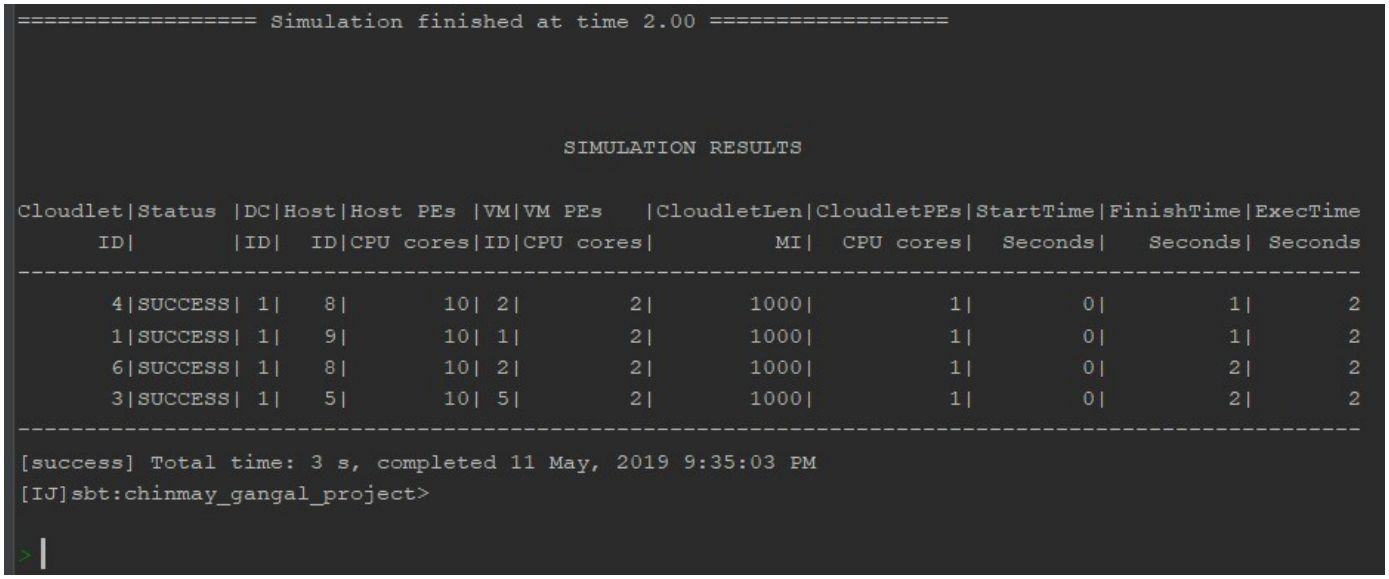
Made use of the following policies for this result:

vmAllocationPolicyPercentagePes

CloudletToVmMappingRegionFit

CloudletSchedulerTimeShared

vmScheduler = "SpaceShared"



```
================== Simulation finished at time 2.00 ==================



                                    SIMULATION RESULTS

Cloudlet|Status |DC|Host|Host PEs |VM|VM PEs   |CloudletLen|CloudletPEs|StartTime|FinishTime|ExecTime
      ID|       |ID|  ID|CPU cores|ID|CPU cores|         MI|  CPU cores|  Seconds|    Seconds| Seconds
-------------------------------------------------------------------------------------------------------
       4|SUCCESS| 1|   8|       10| 2|        2|       1000|          1|        0|          1|       2
       1|SUCCESS| 1|   9|       10| 1|        2|       1000|          1|        0|          1|       2
       6|SUCCESS| 1|   8|       10| 2|        2|       1000|          1|        0|          2|       2
       3|SUCCESS| 1|   5|       10| 5|        2|       1000|          1|        0|          2|       2
-------------------------------------------------------------------------------------------------------
[success] Total time: 3 s, completed 11 May, 2019 9:35:03 PM
[IJ]sbt:chinmay_gangal_project>

>
```

4. Integration test ouput:The integration test below shows that the simulation works correctly.