UNIVERSITÀ DI PISA

DIPARTIMENTO DI INFORMATICA

Orchestration and collocation of application for autonomous drive

Candidato: Tommaso Lencioni
Relatori: Prof. Patrizio Dazzi, Prof. Antonio Brogi

Anno Accademico: 2020/2021

Contents

Chapter 1

Introduction to the problem

1.1 Application allocation

Chapter 2

State of the art

2.1 Cloud Brokering

Chapter 3

Metodologies

3.1 Application allocation

3.2 Tools

Chapter 4

Thesis's goal

Chapter 5

Pure Edge Sim

5.1 Features and limitations

Pure Edge Sim (*PES*) is a task oriented simulator. This trait is inherited from its parent tool (Cloud Sim).
Although this approach to simulations is suitable for testing task scheduling on Cloud and Edge, such decoupling represents a major barrier for the implementation of consequentials events involving more than one architecture layer. An insight of the structure can be take in the paper proposed by the authors [LINK BIBLIOGRAFIA].

5.2 Start a simulation

Starting from main in MainApplication the method launchSimulation is called.
Configuration files are parsed.
Scenarios (combinations of parameters) are loaded into Iterations. The iterations are executed one by one in a for loop.
Before starting the simulation the models are loaded:

- The data centers are generated (this includes the edge devices).
- The tasks are generated at random for the devices that can generate them (every device have an APPLICATION_LIST.
- An orchestrator is set.
- A network model is set.

The method startInternal of SimulationManager is called.
Before scheduling the task as stated below there is a check for the flag ENABLE_ORCHESTRATORS. If it is false the task's orchestrator is the generating device itself. All the task are scheduled to "this" (the Simulation Manager) with a delay based on the time established during their generation and with the tag SEND_TO_ORCH.
Tasks regarding the simulation (log printing, charts updating and progress bar) are scheduled.
All the classes that can receive a scheduled task (not necessary an edge device's task) implements the method processEvent that evaluate the tag of the event in a switch, trying to match it to a known case.

5.3 Task's life

Every step in a task's life is handled by either Simulation Manager or Network Model. The initial scheduling to itself done for every task as stated in the previous section causes the Simulation Manager to catch the tasks in the method **process event**.
Due to the SEND_TO_ORCH flag set before the method **sendTaskToOrchestrator** is called. Check if the task is failed. If the orchestrators are enabled the closest device among the non-cloud orchestrators is set as such for the task. Then the task is scheduled immediately with **scheduleNow** to the Network Model with tag SEND_REQUEST_FROM_DEVICE_TO_ORCH.

As stated before the Network Model of the simulation receive the task and in its processEvent method the right case is caught, calling the method **sendRequestFromDeviceToOrch** right away.
If the orchestrator of the task is the generating device itself the task is immediately scheduled to the Simulation Manager with SEND_TASK_FROM_ORCH_TO_DESTINATIO tag. Otherwise a new transfer with

- file size = the number of bits of the task's file
- type = REQUEST

is added to **transferProgressList** in order to simulate the request traveling through the network from the source to the orchestrator. Let's consider this case.

Network transfers

The **transferProgressList** is an ArrayList initialized at the creation of the Network Model object.

It is loaded with transfers objects to simulate any kind of displacement of information.

In the method startInternal of Network Model an event with tag UPDATE_PROGRESS is scheduled and repeated every NETWORK_UPDATE_INTERVAL as we can see in processEvent.

Here updateTasksProgress is called every update cycle.

In this method transferProgressList is scanned and, for every transfer in queue, the number of transfers on the same network (both WAN and LAN) and their utilization are evaluated.

- WAN is used when the transfer involves the Cloud.
- LAN is used when two transfers share an endpoint.

Those exchanges undergo the bandwidth limit that is set to the minimum between LAN and WAN. The bandwidth is updated (updateBandwidth) as well as the transfer (updateTransfer).
In this part a crucial role is played by the flag REALISTIC_NETWORK_MODEL. In fact, if true, the remaining file size of the transfer is set after a subtraction between the old remaining file size and the NETWORK_UPDATE_INTERVAL times the current bandwidth. This implies that a simulation done in this way reflects how the network delay is impacting the performance of the whole system.
This is essential to point out because if the flag is false the remaining file size is simply set to 0. Considering the fact that this happens every NETWORK_UPDATE_INTERVAL, if the latter is set to a low value the simulation could be not faithful.
Upon finishing the transfer (therefore the remaining file size is 0) **transferFinished** is called. Here the network usage is updated and the transfer is removed from the queue. According to the type of the transfer (request, task, container, result to orchestrator or result to device) a different "if guard" is satisfied and the corresponding method is called. After that call the energy model is always updated
In our case (REQUEST as type) the method offloadingRequestRecievedByOrchestrator is called.

Here there is a check for the type of task's orchestrator:

- If it's Cloud then the task is scheduled to Simulation Manager with a delay of WAN_PROPAGATION_DELAY and tag SEND_TASK_FROM_ORCH_TO_DESTINAT
- otherwise same as above except the immediate schedule (no WAN propagation delay if the orchestrator is an Edge Device).

In Simulation Manager the corresponding switch case is caught in processEvent and sendFromOrchToDestination is called. Here there is a check if the task is failed. After that the a suitable VM is searched for the task.

VM Choice

The method initialize of the Simulation Manager's orchestrator is called. The architecture of the simulation is evaluated and the respective method is called. Whatever method is called the only difference is the array of strings Architecture (containing the names of the supported architectures in this simulation) passed to the method nesting of assignTaskToVM and findVM.

The latter is up to implementation and utilizes the simulation algorithm to find the suitable VM.

[INSERIRE PERSONALIZZAZIONE DELL'ALGORITMO]

assignTaskToVM evaluates the VM (-1 results in a failure for lack of resources) and assign it to the task.

There is a second evaluation for the VM=Null (this time in the Simulation Manager) then there is a check:

- If the device that generated the task isn't the same that contains the assigned VM and the orchestrator of the task isn't the data center that contains the assigned VM then a task with tag SEND_REQUEST_FROM_ORCH_TO_DESTIN and destination the Network model is scheduled immediately.
- Otherwise the task is scheduled to this (Simulation Model) with tag EXECUTE_TASK.

Let's consider the firts case.

In network model the corresponding case is caught and sendRequestFromOrchToDest is called.

A transfer is added to the queue with tag TASK. The transfer behavior is the same as mentioned in the Network transfer subsection. Upon finishing

the transfer the corresponding if is caught and executeTaskOrDownloadContainer is called.

Here there is a check for the ENABLE_REGISTRY, in that case a new task with tag DOWNLOAD_CONTAINER is scheduled.

Otherwise the task is scheduled to the Simulation Manager according to the type of data center of the VM assigned to the task (schedule with WAN_PROPAGATION_DELAY for Cloud and schedule immediately for the Edge) with the tag EXECUTE_TASK.

In the simulation manager the corresponding branch is caught. There is a check for failure, then submitCloudlet is called on the broker (created at the initialization). The CPU utilization of the VM is increased accordingly and the energy model too.

The task is executed by the Cloud Sim part of the project.

At its termination (CLOUDLET_RETURN case in processEvent of the DatacenterBrokerSimple class extension) the task is immediately scheduled to the simulation manager with tag TRANSFER_RESULTS_TO_ORCH.

In Simulation Manager TRANSFER_RESULTS_TO_ORCH case is taken. The CPU percentage is remove and the method sendResultsToOchestrator is called.

There is a check if the task is failed. Then:

- If the source device isn't equal to the data center that contains the associated VM the task is scheduled immediately to the network model with tag SEND_RESULT_TO_ORCH.
- otherwise the task is immediately scheduled to this (Simulation Manager) with the tag RESULT_RETURN_FINISHED.

Let's consider the first case. In network model the corresponding case is caught and the method sendResultFromDevToOrch is called. Here there is a check:

- if the orchestrator of the task isn't the device that generated the task then a file transfer is started with tag RESULTS_TO_ORCH
- otherwise the task is immediately scheduled to this (Network Model) with the tag SEND_RESULT_FROM_ORCH_TO_DEV)

Let's consider the first case.

Upon finishing the transfer returnResultToDevice is called. A check whether the orchestrator or the data center of the VM assigned to the task are Cloud. In that case the task with tag SEND_RESULT_FROM_ORCH_TO_DEV is scheduled to this (Network Model) with WAN_PROPAGATION_DELAY.

Otherwise the same is done without delay.

The corresponding case is caught and the method sendResultFromOrch-ToDev is called. Here a new file transfer is added with tag RESULTS_TO_DEV.

Upon finishing the transfer the last else is caught and the method resultsReturnedToDevice is called. This schedules immediately the task for the Simulation Model with tag RESULT_RETURN_FINISHED.

In Simulation Model the corresponding case i caught. A failure check is done, then a customizable method of the simulation orchestrator is called (resultsReturned) and the number of task is increased.