Politenico di Milano

Dipartimento Elettronica, Informazione e Bioingegneria

HEAPLab Project Report

EdgeCloudSim Report

Author:
Zhang Qiaolun

Supervisor:
Michele Zanella

March 6, 2019

Abstract

The simulation tool EdgeCloudSim provides environment to simulate edge computing scenarios, which can be used to conduct experiments that considers both computational and networking resources. However, the application allocated and scheduled in the system is atomic, which means the application cannot be divided and can only be allocated in one devices. But seperating the application into multiple kernels and distributing them to edge servers in the vicinity really counts because edge server is usually resource limited and may not capable to execute the application. In this project, the atomic application simulation in EdgeCloudSim is extended into thread-based application, which provides a more realistic simulation of computation and networking in edge computing.

1 Introduction

Edge Computing is a new ocmputing paradigm which utilize the resoruces near collabrates with cloud computing to provide better services to the user. EdgeCloudSim is a simulation tool which can simulate edge computing and cloud computing scenarios which combine Edge Computing and Cloud Computing. EdgeCloudSim. EdgeCloudSim is extended from CloudSim and uses the discrete event management framework to simulate the dispatching and scheduling of applications.

The core simulation process is a loop function which checks the events related to all the entities and processes the event. Each entities can process different kinds of events. Figure 1 shows the relations between the EdgeCloudSim modules. The SimSettings class in core simulation modules will load and store all the simulation settings information, which includes the application parameters, the edge server settings, the simulation environment settings. The load generator module will utilize the information in SimSettings instance to generate applications that will be scheduled. When generating applications, mobility module will be utilized to generate the position of each application. Then the simulation of edge computing scenarios starts and the program will loop a clock click cycle. At each clock cycle, the simulation will check whether there are some applications starts at that time. If it is, edge orchestrator module will utilize the specified allocation strategy to calculate the virtual machine in a server to execute the application. Then

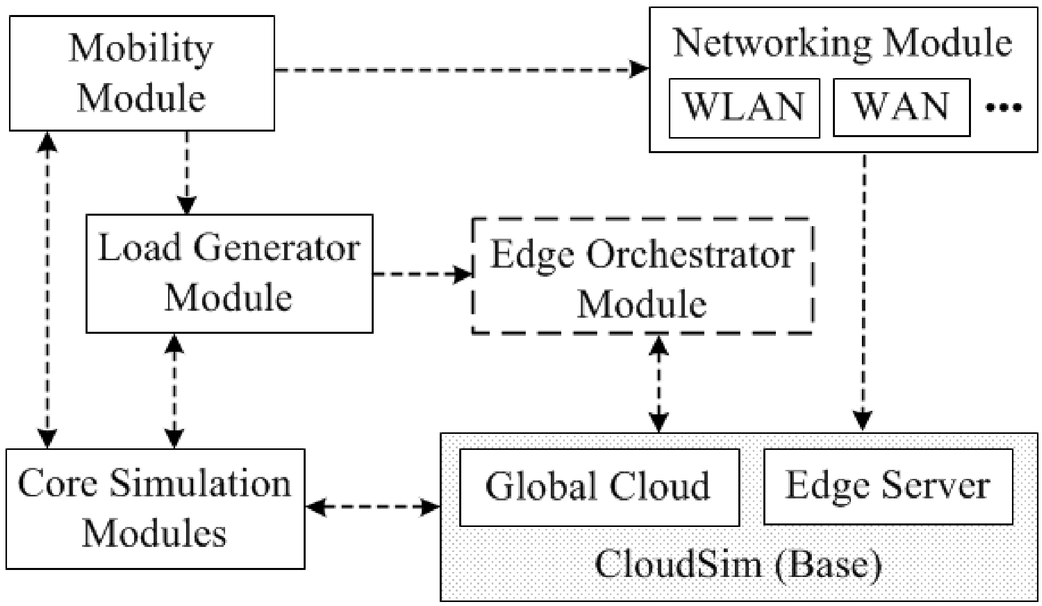


Figure 1: This is a figure caption.

MobileDeviceManager class will utilize the calculated information in edge orchestrator module to submit the application to CloudSim (Base).

2 Problem Statement

In EdgeCloudSim, the application is atomic and can only be executed in one edge devices. However, some applications can be divided into multiple kernels and simulate the execution of these kernels really counts. Since single edge sever is resources-limited and may not have enough resoruces to execute the application. But executing the application in the cloud may lead to time-delay and even miss the deadline of the application. Consequently, application is divided into multiple small kernels, which are distributed into multiple edge server. So it is necessary to provide kernel-based application feature to simulate edge computing scenarios.

3 Design and Implementation

3.1 Kernel-Based Application Design

In EdgeCloudSim, application is atomic and cannot be divided into smaller kernels. In this project, kenel-based application is introduced into this simulation tool. Kernel-based application is composed of a single kernel or multiple kernels. And the kernel is distinguised by kernelId in this implementation. Ma We can define how to divide applications in XML file and store their dependencies.

3.2 Allocation of Kernel-Based Application

3.3 Mapping between Kernel-Based Application and it's kernel

3.4 Classes Implementation

3.4.1 SimLogger.java and LogItem.java

3.5 Simulation Workflow

The simulation works in the following steps. First SimManager will use the class SimSettings to load the settings about the application types, edge server, and some other parameters.

The generation of applications are in two phase. In the first phase, SimManager will utilize LoadGenerator to generate the applications that will be scheduled in the simulation. In this process, LoadGenerator will use Possion process to get the arrival time of each applications and use the parameters in SimSettings to generate the positions of each applicaion. Then comes to the second phase, it's a loop to submit the kernels in the application to be scheduled. The loop ends when the simulation comes to the end of the simulation time. In each loop, SimManager

1. lead the settings

2.

3. edge_server: class EdgeServerManager.java has similar functions as CloudServer.

4. mobile_processing: class MobileServerManager.java. This class enables the mobile devices have the ability to process task. We can also create data centers and virtual machines on it.

5. edge_client: MobileDeviceManager.java extends DatacenterBroker class in CloudSim. And it overwrite the processOtherEvent function and add new events: REQUEST_RECEIVED_BY_CLOUD, REQUEST_RECEIVED_BY_EDGE_EDG

4 Relation between EdgeCloudSim and Cloudsim

4.1 Simulation Framework of CloudSim

4.1.1 Simulation Data Flow

Figure 1 shows the call hierarchy of member function updateCloudletProcessing in class DataCenter. From this figure, we can figure out that the Cloudlet processing update process figure in the paper about CloudSim is wrong. Figure 2 is the fixed Cloudlet processing update process. At the end of the function updateCloudletProcessing, it will add a new VM_DATACENTER_EVENT to the future queue using function schedule. When we go back to the loop of Run function, the updateCloudletProcessing function will be triggered again.

Figure1 shows the Cloudlet processing unit in CloudSim. As is shown in the picture1, there are two important classes in core package: ScenarioFactory and SimManager. The ScenarioFactory gets the parameters for the scenarios. And the SimManager receives the object of type ScenarioFactory. Moreover, the SimManager is a class extended from SimEntity class. The SimEntity is a class defined in CloudSim. And it has a function called startEntity(), which will schedule the task.

4.2 Modules in EdgeCloudsim

1. core: three are there important class in this module. ScenarioFactory.java is the class for factory scheme. SimManager.java class extends SimEntity class and it is related to CREATE_TASK, CHECK_ALL_VM, GET_LOAD_LOG, PRINT_PROGRESS, STOP_SIMULATION. SimSettings.java is the class that stores all the configurations.

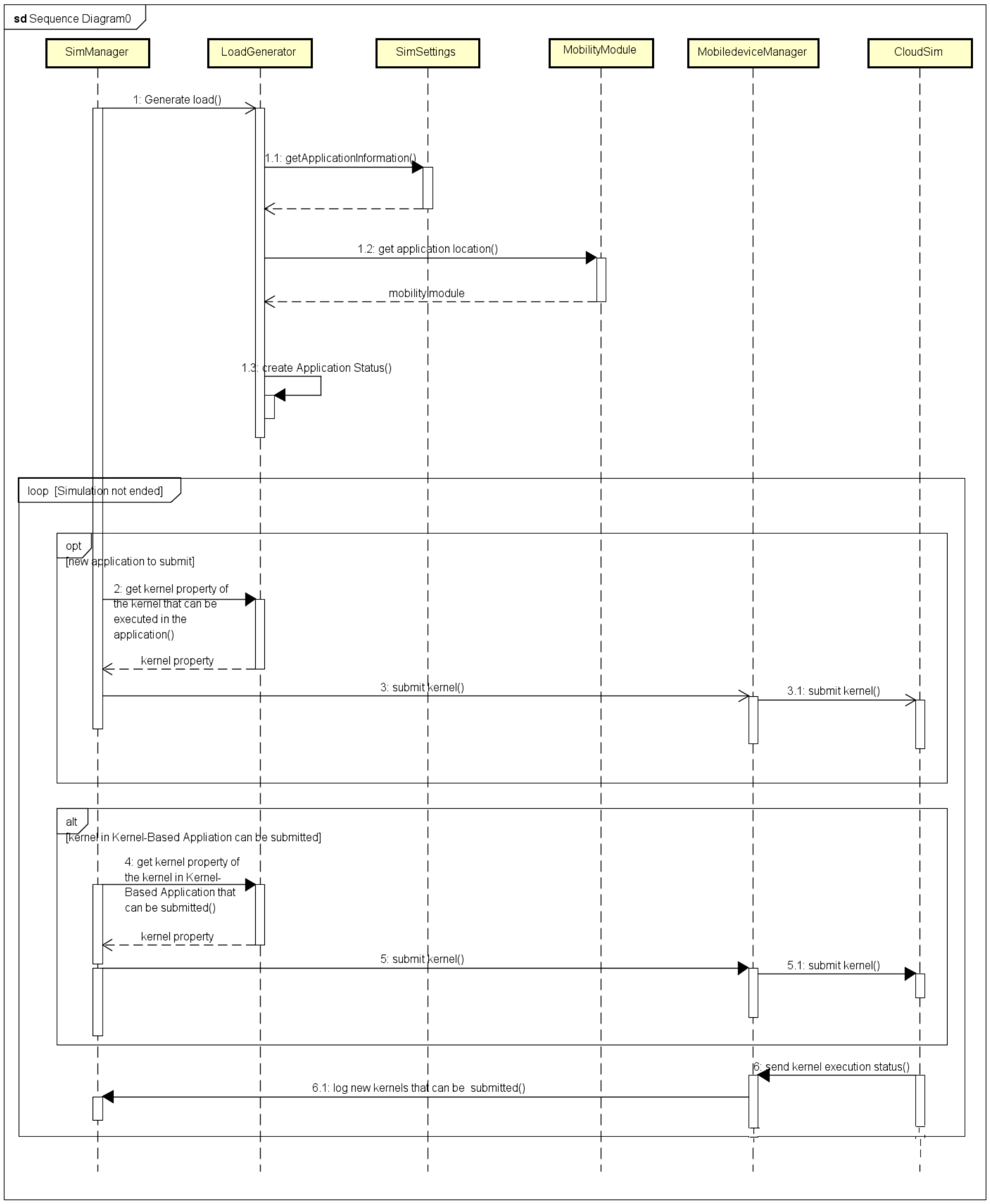


Figure 2: Modules and functions related to create and submit kernel.

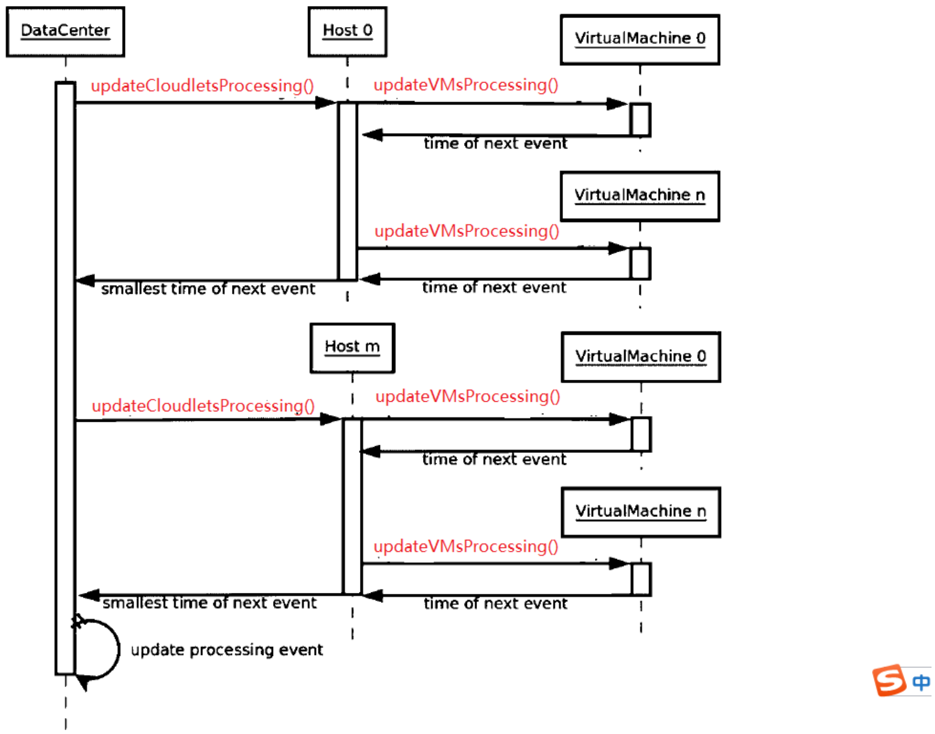


Figure 3: Cloudlet processing update process.

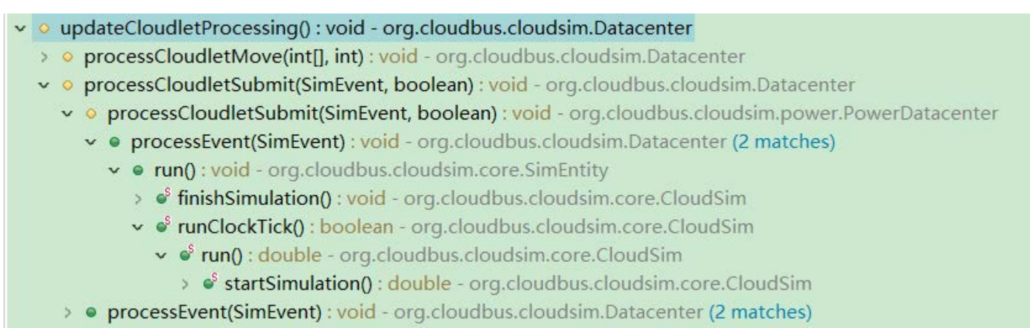


Figure 4: Call hierarchy of function updateCloudletProcessing.

2. cloud_server: class CloudServerManager.java. This class actually just generate the hostlist, vmlist, local data center, and function to get the average utilization of all VMS.

3. edge_server: class EdgeServerManager.java has similar functions as CloudServer.

4. mobile_processing: class MobileServerManager.java. This class enables the mobile devices have the ability to process task. We can also create data centers and virtual machines on it.

5. edge_client: MobileDeviceManager.java extends DatacenterBroker class in CloudSim. And it overwrite the processOtherEvent function and add new events: REQUEST_RECEIVED_BY_CLOUD, REQUEST_RECEIVED_BY_EDGE_EDG

4.3 Entities in EdgeCloudsim

1. SimManager: public class SimManager extends SimEntity

2. MobileDeviceManager: public abstract class MobileDeviceManager extends DatacenterBroker

3. EdgeOrchestrator: public abstract class EdgeOrchestrator extends SimEntity

4.4 Relationship of Modules between CloudSim and EdgeCloudSim

Because EdgeCloudSim is implemented on the top of CloudSim, it also relies on the discrete event management framework. MobileDeviceManager class extends DatacenterBroker class. So it implemented the following functions

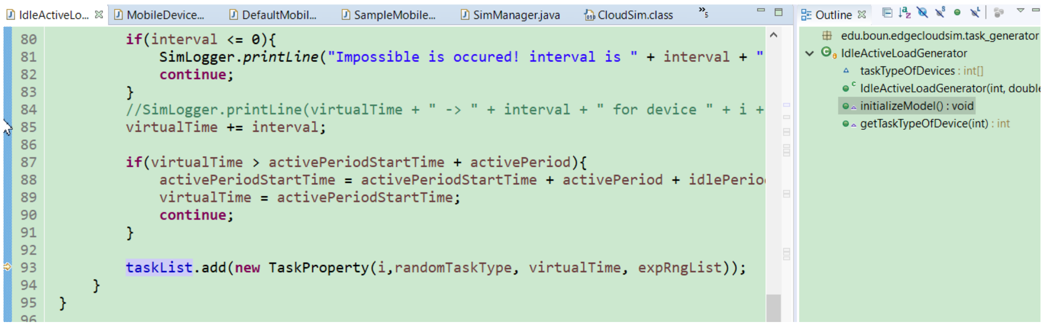


Figure 5: Generating Task List in LoadGenerator class.

Figure 6: Submit The Task Using the Schedule Function.

5 Design and Implementation

5.1 Scheduling in EdgeCloudSim

5.2 Task-based Application Design and Implementation

5.2.1 Task Lifecycle

The process of submit a task is as follows.

Figure 4 shows how generate the task list in LoadGenerator class.

Figure 5 shows how we submit the task using the schedule function.

Figure 6 shows when we come to the clock tick in EdgeCloudSim, the

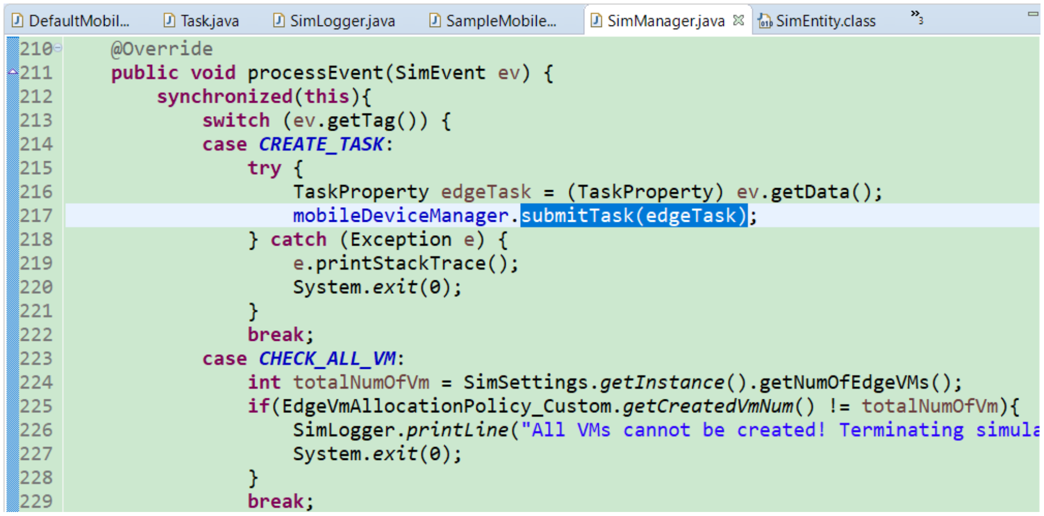


Figure 7: CREATE_TASK event in in SimManager class.

event will be processed.

Figure 7 shows the submitTask function in class MobileDeviceManager. MobileDeviceManager is extended from Broker. We can notice that the task created is added to log list in this class. The task is distinguished from other tasks by its CloudletId.

Figure 8 what we do when the task finished. EdgeCloudSim will change the parameters concerning with execution time, the status in SimLogger class. Actually, we can submit new subtasks in this function. We can add a new field to Task.java class to determine whetehr the task is a subtask or an atomic task.

In the simulation, so we choose only submit a task to virtual machine when its dependencies has been met. In the SimManager class, the startEntity() function creates all the virtual machines in cloud server, edge server and mobile server. Besides, it also submits all the tasks to virtual machines. But in this project, we only submit the tasks whose dependencies has been met at this time. When a task is finished, it will send a CLOUDLET_RETURN event to DatacenterBroker. In EdgeCloudSim, this event will be sent to MobileDeviceManager. Then processCloudletReturn function will be triggered. So when a task is finished, the tasks that have not been submitted will be checked. If there are tasks whose dependencies has been met now, it will be submitted to the VM at this time. The submit of tasks are based on the

```java
    public void submitTask(TaskProperty edgeTask) {
        int vmType=0;
        int nextEvent=0;
        int nextDeviceForNetworkModel;
        NETWORK_DELAY_TYPES delayType;
        double delay=0;

        NetworkModel networkModel = SimManager.getInstance().getNetworkModel();

        //create a task
        Task task = createTask(edgeTask);

        Location currentLocation = SimManager.getInstance().getMobilityModel().
                getLocation(task.getMobileDeviceId(), CloudSim.clock());

        //set location of the mobile device which generates this task
        task.setSubmittedLocation(currentLocation);

        //add related task to log list
        SimLogger.getInstance().addLog(task.getCloudletId(),
                task.getTaskType(),
                (int)task.getCloudletLength(),
                (int)task.getCloudletFileSize(),
                (int)task.getCloudletOutputSize());

```

Figure 8: Task Submitted to the CloudSim Base in SampleMobileDeviceManager.java class.

schedule function. We implement these functions in MobileDeviceManager class.

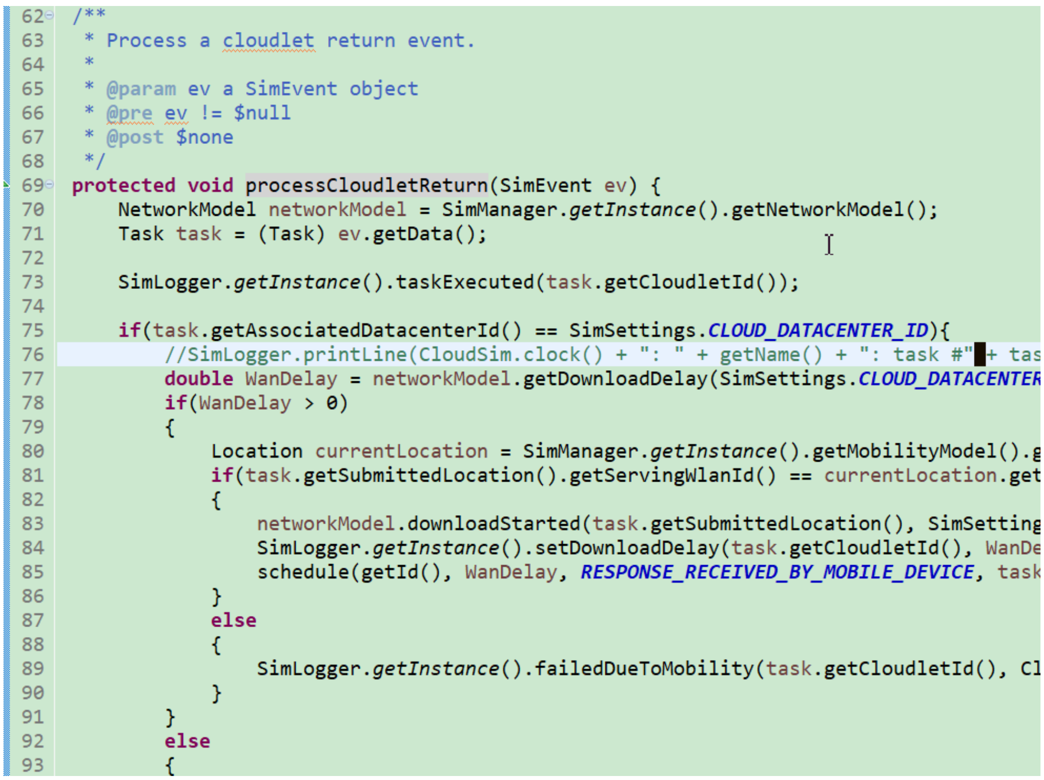
In processCloudletReturn function, it will test the execution place of the task, if it is executed in GENERIC_EDGE_DEVICE_ID, the download delay will be added to the task, if it is executed in mobile device, the delay do not need to be add. We can test whether we need to submit new tasks to virtual machines at the beginning of the function.

5.2.2 Task-based Application Design

In this section introduces the design of task-based application based on EdgeCloudSim.

5.2.3 Structure and Loading of Applications.xml

The Applications.xml describes the types and parameters of applications in the simulation. Parameters in the original Applications.xml file of EdgeCloudSim is not explained, which makes it difficult to understand the mean-



```
/**
 * Process a cloudlet return event.
 *
 * @param ev a SimEvent object
 * @pre ev != $null
 * @post $none
 */
protected void processCloudletReturn(SimEvent ev) {
    NetworkModel networkModel = SimManager.getInstance().getNetworkModel();
    Task task = (Task) ev.getData();

    SimLogger.getInstance().taskExecuted(task.getCloudletId());

    if(task.getAssociatedDatacenterId() == SimSettings.CLOUD_DATACENTER_ID){
        //SimLogger.printLine(CloudSim.clock() + ": " + getName() + ": task #"+ tas
        double WanDelay = networkModel.getDownloadDelay(SimSettings.CLOUD_DATACENTER
        if(WanDelay > 0)
        {
            Location currentLocation = SimManager.getInstance().getMobilityModel().g
            if(task.getSubmittedLocation().getServingWlanId() == currentLocation.get
            {
                networkModel.downloadStarted(task.getSubmittedLocation(), SimSetting
                SimLogger.getInstance().setDownloadDelay(task.getCloudletId(), WanDe
                schedule(getId(), WanDelay, RESPONSE_RECEIVED_BY_MOBILE_DEVICE, task
            }
            else
            {
                SimLogger.getInstance().failedDueToMobility(task.getCloudletId(), Cl
            }
        }
        else
        {
```

Figure 9: ProcessCloudletReturn Function is Executed in MobileDeviceManager class.

ing of these parameters. Figure 9 is an example of configurations of task-based applications. Figure 10 shows the original configurations of atomic applications. The meanings of the parameters are listed as follows:

1. usage_percentage: The usage percentage here is used to decide the percentage of this type of application in all of the tasks generated. So add up usage_percentage field in the applications.xml, we can get 100 because we have sumed up all the percentages of all applications.

2. prob_cloud_selection: The probability that we choose the cloud to execute the task.

3. poisson_interarrival:

4. active_period: In active period, the edge devices can generate new tasks.

5. idle_period: In idle period, the edge devices can not generate new tasks.

The parameters are loaded in SimSettings.java. A new class called TaskBasedApplication.java is also implemented. The parameters of the applications and sub-applications are stored in the following two variables.

1. private double[][] taskLookUpTable: store the parameters of applications.

2. private double[][] subtaskLookUpTable: store the parameters of sub-applications.

3. private TaskBasedApplication[] dependencyLookUpTable: store the dependency of each task-based application.

5.2.4 Sub-task Submission Using Topological Order

In the class LoadGenerator, we generate the list properties of all tasks including all the sub tasks. The dependencies between all the sub-tasks are maintained in a list of objects of TaskBasedTask.java. The dependencies of them are maintained in this class. The reference of the LoadGenerator instance will be added to the instance of MobileDeviceManager. When MobileDeviceManager knows some tasks ends. It will check whether the task is

```xml
<application name="TASK_BASED_APP">
    <usage_percentage>30</usage_percentage>
    <prob_cloud_selection>20</prob_cloud_selection>
    <poisson_interarrival>2</poisson_interarrival>
    <active_period>40</active_period>
    <idle_period>20</idle_period>
    <sub_applications>
        <sub_application name="SUB_APP1">
            <data_upload>1500</data_upload>
            <data_download>250</data_download>
            <task_length>12000</task_length>
            <required_core>1</required_core>
            <vm_utilization_on_edge>8</vm_utilization_on_edge>
            <vm_utilization_on_cloud>0.8</vm_utilization_on_cloud>
            <vm_utilization_on_mobile>20</vm_utilization_on_mobile>
            <sub_application_number>0</sub_application_number>
            <num_dependency>0</num_dependency>
        </sub_application>
        <sub_application name="SUB_APP2">
            <data_upload>1500</data_upload>
            <data_download>250</data_download>
            <task_length>12000</task_length>
            <required_core>1</required_core>
            <vm_utilization_on_edge>8</vm_utilization_on_edge>
            <vm_utilization_on_cloud>0.8</vm_utilization_on_cloud>
            <vm_utilization_on_mobile>20</vm_utilization_on_mobile>
            <sub_application_number>1</sub_application_number>
            <num_dependency>0</num_dependency>
        </sub_application>
    </sub_applications>
</application>
```

Figure 10: Example of XML file of Task-based applications.

```xml
<application name="AUGMENTED_REALITY">
    <usage_percentage>30</usage_percentage>
    <prob_cloud_selection>20</prob_cloud_selection>
    <poisson_interarrival>2</poisson_interarrival>
    <active_period>40</active_period>
    <idle_period>20</idle_period>
    <data_upload>1500</data_upload>
    <data_download>250</data_download>
    <task_length>12000</task_length>
    <required_core>1</required_core>
    <vm_utilization_on_edge>8</vm_utilization_on_edge>
    <vm_utilization_on_cloud>0.8</vm_utilization_on_cloud>
    <vm_utilization_on_mobile>20</vm_utilization_on_mobile>
</application>
```

Figure 11: Example of XML file of Atomic Applications.

```java
// create random number generator for each place for subtask
for(int i=0; i<SimSettings.getInstance().getSubtaskLookUpTable().length; i++) {
    if(SimSettings.getInstance().getSubtaskLookUpTable()[i][0]==0)
        continue;

    subtaskExpRngList[i][0] = new ExponentialDistribution(SimSettings.getInstance().getSubtaskLookUpTable()[i][5]);
    subtaskExpRngList[i][1] = new ExponentialDistribution(SimSettings.getInstance().getSubtaskLookUpTable()[i][6]);
    subtaskExpRngList[i][2] = new ExponentialDistribution(SimSettings.getInstance().getSubtaskLookUpTable()[i][7]);
}
```

Figure 12: exponential number generator for sub-task.

```java
if (SimSettings.getInstance().isTaskBasedApplication(randomTaskType)) {
    // create an object of TaskBasedTask
    int subtaskNum = SimSettings.getInstance().getsubTaskNum(randomTaskType);
    for (int subTaskIndex=0; subTaskIndex<subtaskNum; subTaskIndex++) {
        // add the subtask to taskList

        // map the subtask to TaskBasedTask
        int subRandomTaskType = SimSettings.getInstance().getsubTaskIndex(randomTaskType, randomTaskType);
        taskList.add(new TaskProperty(i, subRandomTaskType, virtualTime, subtaskExpRngList));
    }
}
else {
    taskList.add(new TaskProperty(i,randomTaskType, virtualTime, expRngList));
}
```

Figure 13: Generate task properties for atomic task and task-based task.

a sub-task. If it is a sub-task. Then it will check whether there are other sub-tasks can be executed after the ending of this sub-task. If there are such tasks, the tasks will be send to datacenter using the function schedule.

The following picture shows how this project generate task properties for atomic task and bask-based task. After generating the random task type for edge devices, we judge whether it is a task-based task. If it is a task-based task, we will generate the properties of these sub-tasks.

The dependencies of sub-task in task-based applications are handled in class TaskBasedApplication. This class can check the status of each task using two private variables called submitted and dependency.

In this project, a class called TaskBasedTask.java is implemented to store the information of task-based task. This class can store the dependencies between each sub-tasks. We can trigger a function in the class to record the tasks that have finished. Moreover, it can calculate the sub-tasks that can be executed at this time. The object of TaskBasedTask will utilize the information stored in TaskBasedApplication to generate the dependencies of sub-tasks.

Now I think add a new class TaskBasedTask.java with static variables. Each object can acccess the static instance in this class. This class can be used to log the status of all the task-based applications.

```java
    if (SimSettings.getInstance().isTaskBasedApplication(randomTaskType)) {
        // create an object of TaskBasedTask
        int subtaskNum = SimSettings.getInstance().getsubTaskNum(randomTaskType);

        int[][] dependency_task = SimSettings.getInstance().getsubTaskDependency(randomTaskType);

        TaskBasedTaskStatus.getInstance().addTaskBasedTask(subtaskNum, taskBasedTaskId);

        int[] id_subtask_list = new int[subtaskNum];

        for (int subTaskIndex=0; subTaskIndex<subtaskNum; subTaskIndex++) {
            // map the subtask to TaskBasedTask
            int subRandomTaskType = SimSettings.getInstance().getsubTaskIndex(randomTaskType, subTaskIndex);
            TaskProperty taskProperty = new TaskProperty(i, subRandomTaskType, virtualTime, subtaskExpRngList, ta
            //int taskId = taskProperty.getCloud
            taskList.add(taskProperty);
            id_subtask_list[subTaskIndex] = taskPropertyId;
            taskPropertyId++;
        }

        // add the taskPropertyId list
        TaskBasedTaskStatus.getInstance().addSubTaskIdList(id_subtask_list, taskBasedTaskId);
        // add the dependency
        for (int id=0; id<dependency_task.length; id++) {
            for (int id_dependency=0; id_dependency < dependency_task[id].length; id_dependency++)
                if (dependency_task[id][id_dependency] ==  1) {
                    // pass id_subtask_list[id] because what we pass is the property_id, not the index
                    TaskBasedTaskStatus.getInstance().addDependency(id_subtask_list[id], id_subtask_list[id_depen
                }
        }
```

Figure 14: Initialize TaskBasedTaskStatus in IdleActiveLoadGenerator.

The class TaskBasedTaskStatus.java and the class TaskBasedTask.java are fully tested using a test Java file. Moreover, in debug mode, the TaskBased-TaskStatus instance is as follows.

5.2.5 New Task Submission

Since in EdgeCloudSim, when we submit a task, the task will be scheduled. So when it comes to task-based task, the sub-task can only be submitted after its dependencies have been met. So when each sub-task finishes, the tasks that belong to the same task-based application with it will be checked. When a task finishes in EdgeCloudSim will execute the processCloudReturn function in MobileDeviceManager class. Actually, CloudSim will execute the processCloudReturn function in Broker. But the MobileDeviceManager cannot access the variables in object LoadGeneratorModel. But the the event of submitting new tasks can be done by sending an event to SimManager. In the meantime, the sub-task need to be set as submitted. In the class SimManager, a new label called CREATE_SUB_TASK is added. And the event is handled in the function processEvent in SimManager. Since we need to submit the new sub-tasks immediately, we use scheduleNow function to

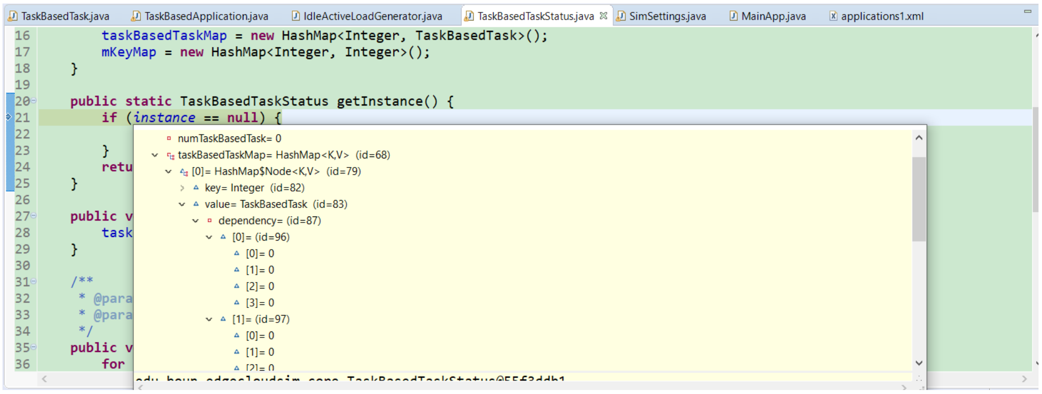


Figure 15: TaskBasedTaskStatus in Debug Mode.

send the event of creating new sub-tasks in the system.

5.2.6 Component Diagram of the Simulation tool

After adding the feature, the component diagram has changed.

5.3 Task Migration

5.4 Probabilistic Network Failure Model

6 Experimental Results

In this section, we did some experiment. We test the failure rate.

7 Conclusions

8 Future Works

9 Some LaTeX Examples

9.1 Sections

Use section and subsection commands to organize your document. LaTeX handles all the formatting and numbering automatically. Use ref and label

Figure 16: This is a figure caption.

Item	Quantity
Widgets	42
Gadgets	13

Table 1: An example table.

commands for cross-references.

9.2 Comments

Comments can be added to the margins of the document using the todo command, as shown in the example on the right. You can also add inline comments too:

Here's a comment in the margin!

This is an inline comment.

9.3 Tables and Figures

Use the table and tabular commands for basic tables — see Table 1, for example. You can upload a figure (JPEG, PNG or PDF) using the files menu. To include it in your document, use the includegraphics command as in the code for Figure 16 below.

9.4 Mathematics

LaTeX is great at typesetting mathematics. Let $X_1, X_2, \ldots, X_n$ be a sequence of independent and identically distributed random variables with $\mathrm{E}[X_i] = \mu$ and $\mathrm{Var}[X_i] = \sigma^2 < \infty$, and let

$$S_n = \frac{X_1 + X_2 + \cdots + X_n}{n} = \frac{1}{n}\sum_{i}^{n} X_i$$

denote their mean. Then as n approaches infinity, the random variables $\sqrt{n}(S_n - \mu)$ converge in distribution to a normal $\mathcal{N}(0, \sigma^2)$.

9.5 Lists

You can make lists with automatic numbering …

1. Like this,
2. and like this.

… or bullet points …

- Like this,
- and like this.

We hope you find writeLaTeX useful, and please let us know if you have any feedback using the help menu above.