Politenico di Milano

Dipartimento Elettronica, Informazione e Bioingegneria

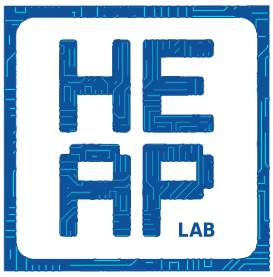
HEAPLab Project Report

EdgeCloudSim Report

Author:
Zhang Qiaolun

Supervisor:
Michele Zanella

February 24, 2019



Abstract

The simulation tool EdgeCloudSim provides some basic features of simulating edge computing scenarios. But it lacks some further features. This project extends basic task application definition to support task-based application.

1 Introduction

EdgeCloudSim is a simulation tool which can simulate Edge Computing scenarios. But this tool can not simulate task-based application. The project adds the following features to the simulation tool:

1. Extend basic task application definition to support task-based application
2. Task migration among the Edge or Cloud VMs
3. Add probabilistic network failure model by considering the congestion or other parameters such as the distance between mobile device and the WiFi access point.
4. Visual tool for displaying the network topology

In the end, this project gives a detailed simulation and provides the simulation results.

2 Relation between EdgeCloudSim and Cloudsim

2.1 Simulation Framework of CloudSim

2.1.1 Simulation Data Flow

CloudSim is a discrete event management framework. 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. So figuring out all the

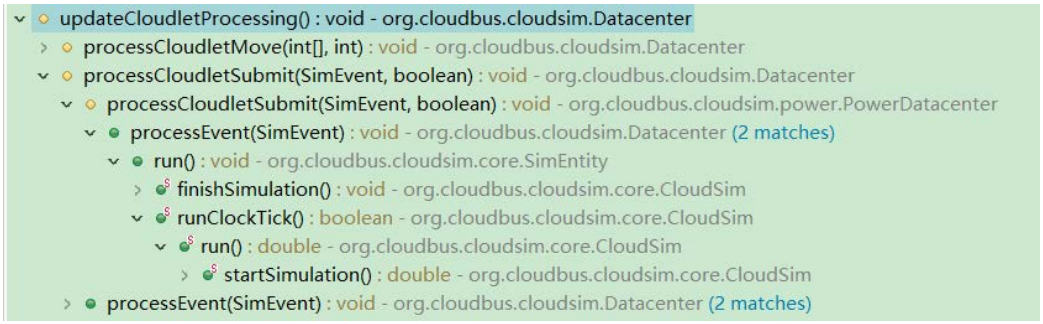


Figure 1: Call hierarchy of function updateCloudletProcessing.

2.1.2 Data center internal processing

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.

2.2 Simulation Framework of EdgeCloudSim

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.

Not sure if I can implement the task-based application here or change the schedule function in CloudSim

2.3 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,

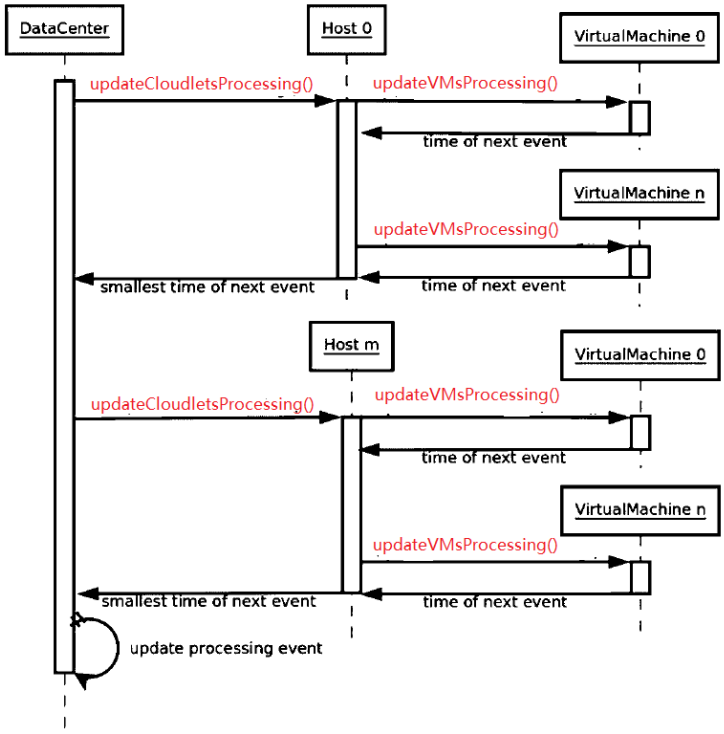


Figure 2: Cloudlet processing update process.

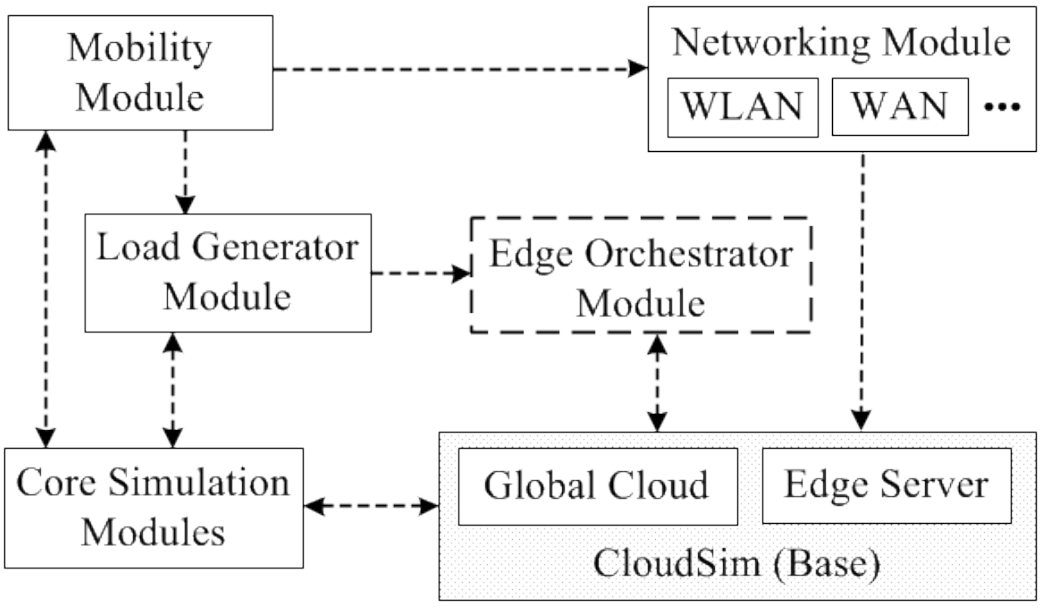


Figure 3: This is a figure caption.

GET_LOAD_LOG, PRINT_PROGRESS, STOP_SIMULATION. SimSettings.java is the class that stores all the configurations.

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

2.4 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

2.5 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

3 Design and Implementation

3.1 Scheduling in EdgeCloudSim

3.2 Task-based Application Design and Implementation

3.2.1 Task Lifecycle

In EdgeCloudSim, task is atomic and cannot be divided into smaller tasks. Task-based application is an application which is composed by several smaller tasks. There may also have dependencies among these tasks. For instance, one sub-task can only starts after another sub-task.

There are two ways to implemented tasked-based application feature in EdgeCloudSim. The first one is to change the code of CloudSim and add task-based features. The second one is to change the code of EdgeCloudSim and add the feature of tasked-based application.

We can define how to divide applications in XML file and store their dependencies.

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 event will be processed.

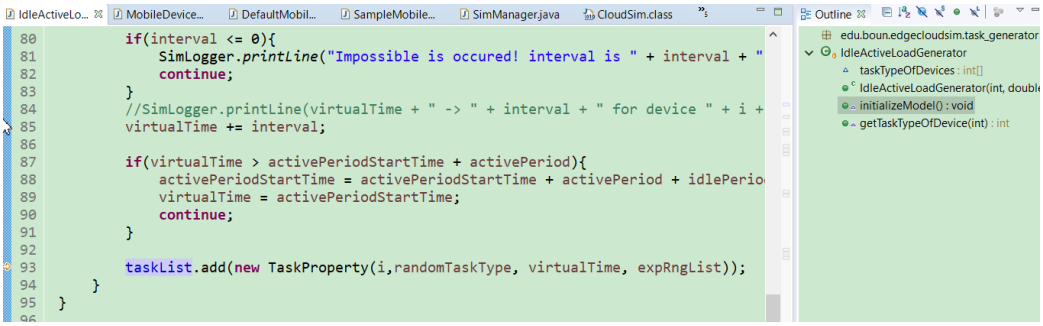


Figure 4: Generating Task List in LoadGenerator class.

Figure 5: Submit The Task Using the Schedule Function.

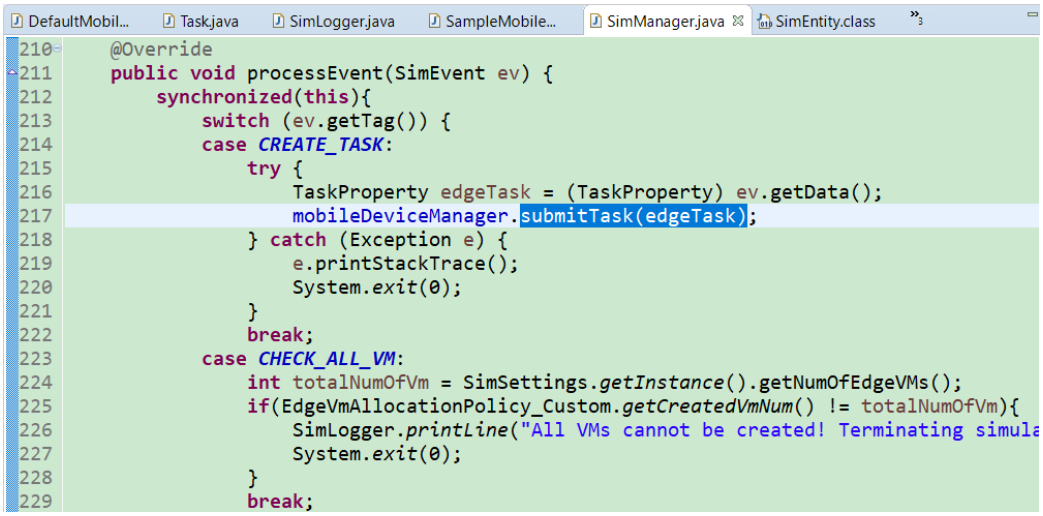


Figure 6: CREATE_TASK event in in SimManager class.

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 schedule function. We implement these functions in MobileDeviceManager

```
282    public void submitTask(TaskProperty edgeTask) {
283        int vmType=0;
284        int nextEvent=0;
285        int nextDeviceForNetworkModel;
286        NETWORK_DELAY_TYPES delayType;
287        double delay=0;
288
289        NetworkModel networkModel = SimManager.getInstance().getNetworkModel();
290
291        //create a task
292        Task task = createTask(edgeTask);
293
294        Location currentLocation = SimManager.getInstance().getMobilityModel().
295                getLocation(task.getMobileDeviceId(), CloudSim.clock());
296
297        //set location of the mobile device which generates this task
298        task.setSubmittedLocation(currentLocation);
299
300        //add related task to log list
301        SimLogger.getInstance().addLog(task.getCloudletId(),
302                task.getTaskType(),
303                (int)task.getCloudletLength(),
304                (int)task.getCloudletFileSize(),
305                (int)task.getCloudletOutputSize());
306
```

Figure 7: Task Submitted to the CloudSim Base in SampleMobileDeviceManager.java class.

class.

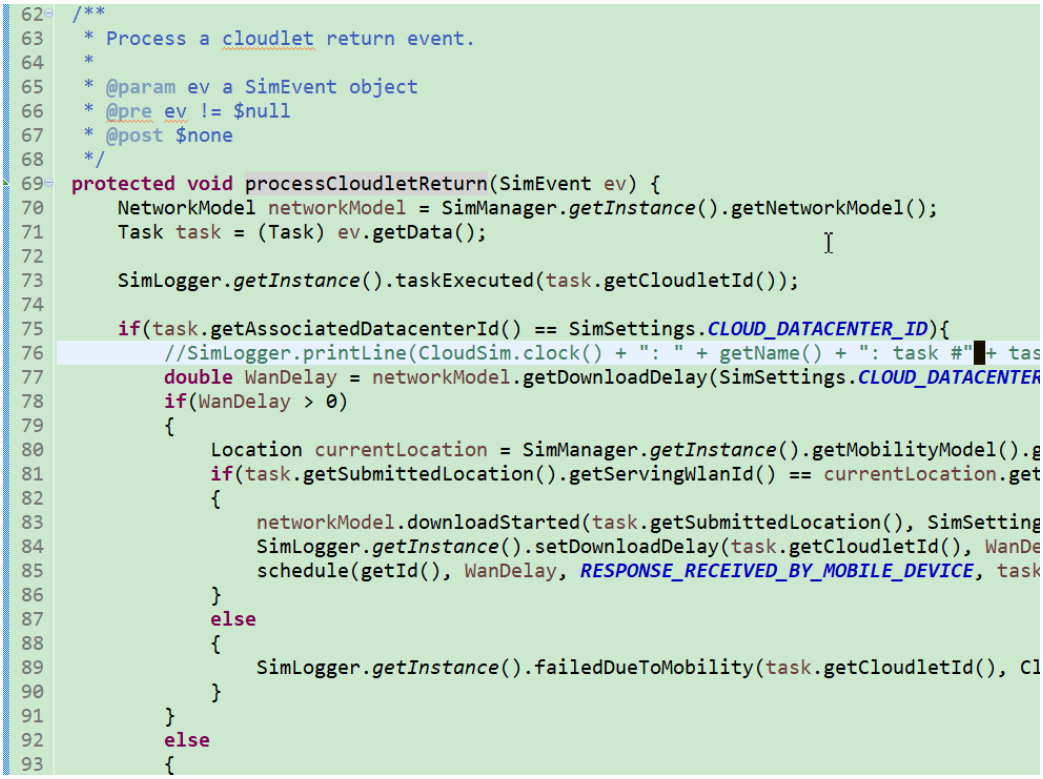
In processCloudletReturn function, it will test the execution place of the task, if it is executed in GENERIC_EDGE_DEVICE_ID, the download delay will be added to the task, if it is executed in mobile device, the delay do not need to be add. We can test whether we need to submit new tasks to virtual machines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

3.2.2 Task-based Application Design

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

3.2.3 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 meaning of these parameters. Figure 9 is an example of configurations of task-



```
/**
 * 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 8: ProcessCloudletReturn Function is Executed in MobileDeviceManager class.

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.

3.2.4 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 a sub-task. If it is a sub-task. Then it will check whether there are other

```
<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 9: Example of XML file of Task-based applications.

```
<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 10: Example of XML file of Atomic Applications.

```
// 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 11: exponential number generator for sub-task.

```
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 12: Generate task properties for atomic task and task-based task.

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.

3.2.5 Scheduling Algorithm in EdgeCloudSim

Which module can we modify to achieve task-based application? How to deal with task-dependency graph

3.2.6 Component Diagram of the Simulation tool

After adding the feature, the component diagram has changed.

3.3 Task Migration

3.4 Probabilistic Network Failure Model

4 Experimental Results

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

5 Conclusions

6 Some LaTeX Examples

6.1 Sections

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

6.2 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:

This is an inline comment.

Here's a comment in the margin!

6.3 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

Figure 13: This is a figure caption.

Item	Quantity
Widgets	42
Gadgets	13

Table 1: An example table.

menu. To include it in your document, use the includegraphics command as in the code for Figure 13 below.

6.4 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 $\mathcal{N}(0, \sigma^2)$.

6.5 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.