PFogSim: A Simulator for Evaluating Dynamic and Layered Fog Computing Environments

by

Qian Wang

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
May 4, 2019

Keywords: IoT, Fog Computing, Edge Computing, Cloud Computing, EdgeCloudSim

Approved by

Dr. Sanjeev Baskiyar, Chair, Professor of Computer Science and Software Engineering
Dr. Saad Biaz, Professor of Computer Science and Software Engineering
Dr. Tao Shu, Assistant Professor of Computer Science and Software Engineering

Abstract

The first phase of 5G specification, Release-15, will be completed by April 2019, and our world will truly enter the 5G era. With the advent of 5G, the data transmission cost is expected to reduce significantly. Thus, it will support Edge Computing and Fog Computing feasibility in the future.

To evaluate the cost of the new computing configuration, a new simulator called PFogSim was proposed in [Shehenaz 2019]. As a participant of the PFogSim project, the PFogSim simulator will be discussed in this work. Compared to other simulators which support only a limited number of fog nodes, PFogSim not only supports simulation of a large-scale multi-layered fog design but also supports different orchestrators. It reports the simulation results with metrics. Users can test and evaluate their proposed fog architecture using these metrices accurately and efficiently. Besides, built-in orchestrators, PFogSim supports users to load customized orchestrators and verify their own fog architecture.

To test the simulator, we ran several simulations with different numbers of mobile devices ranging from 500 to 5,000. The test results show that our simulator can provide various feedback information metrics to the user. By using this information, we use several MATLAB scripts to generate result graphs which help users evaluate their fog computing environment effectively and conveniently.

Acknowledgments

The development of the simulator software was a joint work by the following members: Shehenaz Shaik, Jacob Hall, Clayton Johnson, and Qian Wang under the direction of Dr. Sanjeev Baskiyar. As a result of this work, a paper [Shehenaz 2019] was prepared which is in the process of being submitted. As such, there may be material that appears in the paper and this thesis.

I would like to thank my advisor Dr. Sanjeev Baskiyar for his guidance that were very important to this thesis. I would like to thank my team member Shehenaz Shaik for helping me overcoming the difficulty during the work. I would also like to thank my wife Yunfan Rao and my friend Robert Anderson for their effort of reviewing my thesis.

This work was support in part by NSF National Science Foundation award # 1659845 and 096627

Table of Contents

Contents

Abstract ii
Acknowledgments iii
Table of Contents iv
List of Tables vi
List of Figures vii
Chapter 1: Introduction vii
Chapter 2: Related Work 2
Chapter 3: PFogSim Project and My Contribution 4
Chapter 4: Workflow 5
4.1. Pre-simulation 5
4.1.1. Import configuration 5
4.1.2. Create a virtual network system 6
4.1.3. Mobile devices initialization 6
4.2. Simulation 7
4.2.1. Upload task to host 7
4.2.2. Download task to the mobile device 7
4.2.3. Mobile Device receives Task 8
4.2.4. The SimManager stops the simulation 8
4.3. Post-simulation 8
Chapter 5 Components 10
5.1. Service Placement 10
5.1.1. Create Mobile Device List 10
5.1.2. Assign Host 11
5.1.3. Reserve the resource in the host. 12

5.2. Orchestrators 12
5.2.1. Central Orchestrator 13
5.2.2. Edge-Only Orchestrator 13
5.2.3. Cloud-Only and Fixed-Node Orchestrator 14
5.3. Cost Calculation 14
5.3.1. For each task 14
5.3.2. For average cost 14
Chapter 6 Testing and Evaluation 15
6.1. Service Placement: 15
6.1.1. Cloud-Only Orchestrator 15
6.1.2. Fixed-Node Orchestrator 16
6.1.3. Edge-Only Orchestrator 17
6.1.4. Central Orchestrator 18
6.2. Cost evaluation 19
6.3. Result Graphs 20
Chapter 7 Conclusion 26
References 27

List of Tables

Table 1: Cloud-Only Orchestrator (Tasks Distribution) 15
Table 2: Fixed-Node Orchestrator (Tasks Distribution) 16
Table 3: Edge-Only by Distance Orchestrator (Tasks Distribution) 17
Table 4: Edge Only by Latency Orchestrator (Task Distribution) 17
Table 5: Central Orchestrator (Tasks Distribution) 18

List of Figures

Figure 1: Class Interactions Before Simulation Start 5
Figure 2: Class Interactions During Simulation 7
Figure 3: After Simulation 8
Figure 4: PFogSim Major Class Interactions 9
Figure 5: Start Simulation 10
Figure 6: Create Mobile Device List Method 11
Figure 7: Assign Host 11
Figure 8 Good Host Method 12
Figure 9: Make Reservation Method 12
Figure 10: Cloud-Only Orchestrator (Tasks Distribution) 16
Figure 11: Fixed-Node Orchestrator (Tasks Distribution) 16
Figure 12: Edge-Only by Distance Orchestrator (Tasks Distribution) 17
Figure 13: Edge Only by Latency Orchestrator (Task Distribution) 18
Figure 14: Central Orchestrator Tasks Distribution 19
Figure 15: Average Cost for Augmented Reality App 20
Figure 16: Task Distribution Per Layer for Augmented Reality App 20
Figure 17: Average Distance Between Mobile Device and Host 21
Figure 18: Tasks Failure Rate 22
Figure 19: Average Number of Hops Between Mobile Device and Host 23
Figure 20: Average Network Delay Between Mobile Device and Host 24
Figure 21: Average Processing Time 25

Chapter 1: Introduction

With the advent of 5G, the Internet of Things (IoT) systems will become more and more popular. As providing infrastructure for IoT systems, fog computing has its own advantages. Compared to edge computing, which has limited resource in edge systems, fog devices can handle more jobs. Whereas cloud computing systems are limited in their ability to relocate, fog devices can more easily adapt to shifting populations and customer demands. This advantage not only mitigates the delay of data transmission but also improves the performance of some IoT systems such as remote surgery and self-driving cars. It is especially so if the common points between these IoT systems are sensitive to network delay. Furthermore, fog computing is a distributed system and cloud computing is a centralized system; Thus, fog computing can provide more efficient solutions on data storage, processing, and analysis.

Since the demand for IoT systems and fog computing increase rapidly, lots of architects realize they need a tool to help them with testing and analysis of their fog computing architecture design. This is the reason why we started the PFogSim project. In the PFogSim, we use a real-world locations dataset from the City of Chicago [Data Portal, 2018] for simulation. The various simulation reports help the user to fully evaluate their system design.

Chapter 2: Related Work

Since CloudSim [Rodrigo, 2011] was released in 2011, many simulators have been proposed for IoT environments every year. These simulators include EdgeCloudSim [C. Sonmez, 2017], FogTorch [Antonio, 2017], FogTorchII [Antonio, 2017], and iFogSim [Gupta, 2017], etc. As a pioneer, CloudSim divides the cloud computing system to different modules which can be customized by users. Take the advantage of this, researchers extend the scope from cloud computing to edge computing and fog computing. EdgeCloudSim [C. Sonmez, 2017] and iFogSim [Gupta, 2017] are typical examples of these simulators.

EdgeCloudSim migrates cloud computing to edge computing by modifying the broker between mobile devices and hosts. The new broker only deploys the task to edge host. To imitate the real-world environment, EdgeCloudSim is able to assign a location to host and mobile devices. Compared to the single log output of CloudSim, EdgeCloudSim supports the graphical representation of results. Due to the limitation of cloud computing and edge computing, EdgeCloudSim does not have a complex network of nodes in its environment.

iFogSim is a simulator for fog computing. Similar to EdgeCloudSim, iFogSim also extend from CloudSim. Unlike EdgeCloudSim and CloudSim, iFogSim provides a multi-layered fog nodes architecture for its simulation environment. Although iFogSim keeps the modularity of CloudSim, it does not support the deployment for the large number of tasks. This makes iFogSim contains limited mobiles devices in the simulation environment.

PFogSim extends the CloudSim and EdgeCloudSim in terms of inheritance from the advantages of these simulators and launching new features. Take advantage of the different brokers, PFogSim is able to simulate both cloud computing, edge computing, and fog computing. Furthermore, PFogSim supports mobile devices changing their locations during the simulation. By applying the City of Chicago geospatial data into Hierarchical and Autonomous Fog Architecture [Shehenaz, 2018], PFogSim creates a more complex and realistic fog nodes network in its environment Than other simulators mentioned above.

Chapter 3: PFogSim Project and My Contribution

We implemented PFogSim in Java to take advantage of the existing CloudSim [Rodrigo, 2011] Java library and for the ability to extend an existing edge computing simulator called EdgeCloudSim [C. Sonmez, 2017]. PFogSim can imitate the behavior of clients and server by applying different network, application, and orchestrator configurations. PFogSim then uses these configurations and provides a simulation of a distributed large-scale fog environment.

During the simulation, PFogSim collects different types of data, such as the cost of tasks, the processing time of tasks, and the distance between client and host, generated by the whole IoT system. After the simulation, PFogSim generates various result graphs including the average processing time, the average cost of tasks, and the average distance between host and mobile devices. Based on these results, the user can further analyze the efficacy of the fog system and improve their system.

PFogSim is a group project and has required the help of multiple researchers. My main contribution to PFogSim was in implementing critical components in simulation service placement, orchestration, and delay calculation. These are discussed in Chapter 5.

In Chapter 4, the general workflow of PFogSim and interactions of different classes inside PFogSim are discussed. Chapter 6 shows the testing results and various graphs generated by PFogSim. Chapter 7 makes the conclusion and discusses the future work toward PFogSim.

Chapter 4: Workflow

The workflow of PFogSim can be divided into three phases: pre-simulation, simulation, and post-simulation. Figures 1 to 3 show the major class interactions in each phase and Figure 4 shows the major classes interactions for the whole system.

4.1. Pre-simulation

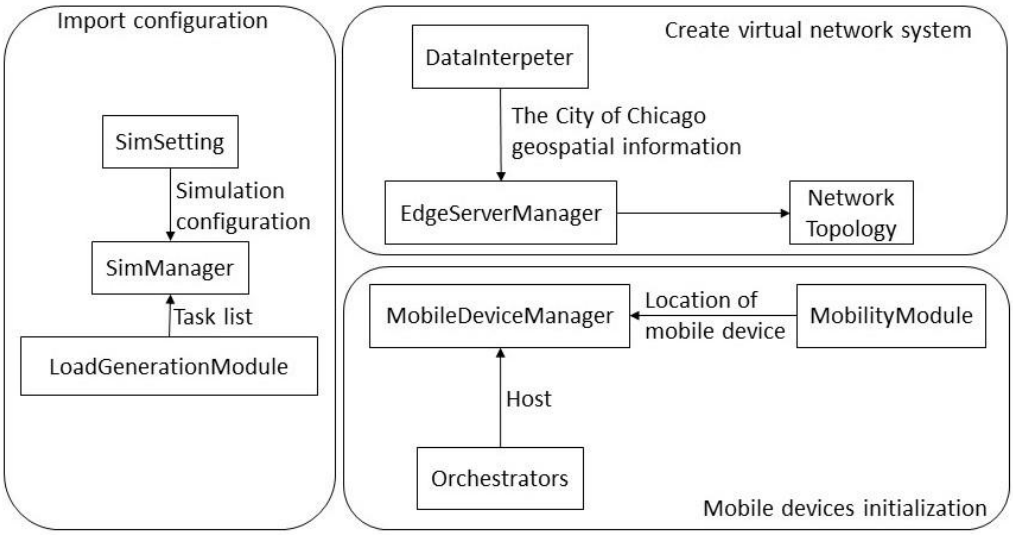

Figure 1: Class Interactions Before Simulation Start

Before the start of simulation, three tasks are completed by several classes as shown in Figure 1.

4.1.1. Import configuration

The DataInterpeter class reads The City of Chicago geospatial information, which is a bunch of points with their latitude and longitude, and generates two XML files which contain the location of fog nodes and links between these nodes. The SimSetting class imports simulator settings from several configuration files.

4.1.2. Create a virtual network system

During this step, an instance of the SimManager class is created. It is a broker class used to handle all the behaviors of the simulator such as start or stop simulation. The LoadGenerationModel class is created and starts to generate tasks for mobile devices. The tasks are generated followed by the Poisson process.

As a manager of fog nodes, the EdgeServerManager class starts all instances of fog nodes and creates the network topology according to fog nodes and links from XML files which were previously created by the DataInterpeter class. In the network topology, we implement hierarchical and autonomous fog architecture [Shehenaz, 2018]. This architecture separates all fog nodes into seven layers with higher layers containing nodes that have more powerful computing capability and bandwidth.

4.1.3. Mobile devices initialization

The MobileDeviceManager is the broker for the mobile device and handles all the behaviors of mobile devices. In this step, the MobileDeviceManager class creates all mobile devices instances and sets the default location of each mobile device. The mobile device changes its location during the simulation according to the rule defined by the MobileModel class. After mobile devices are initialized, MobileDeviceManager assigns the host to each mobile device to follow by the rules defined by various Orchestrator classes. After all the initialization is done, the SimManager module starts the simulation.

4.2. Simulation

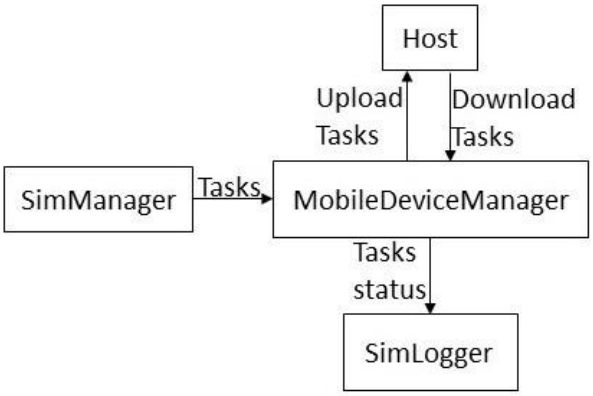

Figure 2: Class Interactions During Simulation

In this period, the mobile device grabs tasks from the SimManager module and uploads tasks to the host. Tasks which have been completed by the host are downloaded to the mobile device. After no task is left in the task list, the simulation is stopped by the SimManager class. This phase can be divided into three steps.

4.2.1. Upload task to host

Before the task is uploaded to the host, the MobileDeviceManager calculates the upload time. If the time of the delay is smaller than the requirement, the MobileDeviceManager uploads task to host. Otherwise, the task is rejected. After the MobileDeviceManager decides the action of the task, the SimLogger class records this action for the task.

4.2.2. Download task to the mobile device

This step is similar to the previous one. The MobileDeviceManager class calculates the download time, and a download action is taken if the time is smaller than the requirement, otherwise, the task is abandoned. Also, the SimLogger records the action for the task.

4.2.3. Mobile Device receives Task

After the mobile device receives the task, the MobileDeviceManager calculates the cost of this task. The cost includes the bandwidth cost and CPU execution cost in the host. The SimLogger class records this cost for the task.

4.2.4. The SimManager stops the simulation

The SimManager class checks task list, when there is no task left in the task list, the SimManager class stops the simulation.

4.3. Post-simulation

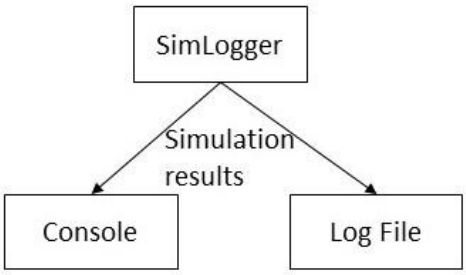

Figure 3: After Simulation

When the simulation is done, all the raw information is captured by the SimLogger class. Thus, the SimLogger class generates result according to this information and writes results to log files and console.

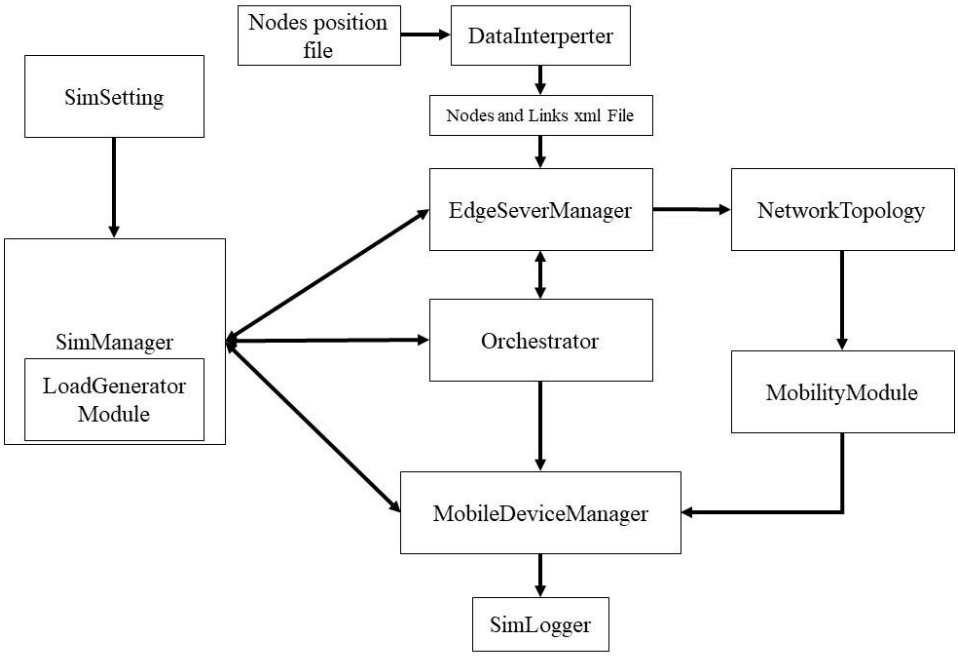

Figure 4: PFogSim Major Class Interactions

Chapter 5 Components

5.1. Service Placement

The Service Placement allocates the tasks to a specific host according to the rule pre-defined in orchestrator. Before the start of simulation, the Service Placement job is finished by the SimManager Class. Service Placement completes three main tasks: create mobile devices list, assign a host to each mobile device, and reserve the resource from hosts which are assigned to mobile devices.

```
/**
 * Triggering CloudSim to start simulation
 */
public void startSimulation() throws Exception{
    //Starts the simulation
    SimLogger.print(super.getName()+" is starting...");

    //Start Edge Servers & Generate VMs
    edgeServerManager.startDatacenters();
    edgeServerManager.createVmList(mobileDeviceManager.getId());
    edgeOrchestrator.initialize();
    SimLogger.print("\n\tCreating device locations...");
    mobilityModel = scenarioFactory.getMobilityModel();
    mobilityModel.initialize();
    SimLogger.printLine("Done,");

    //Qian: added for service replacement
    mobileDeviceManager.creatMobileDeviceList(numOfMobileDevice);
    for (MobileDevice mobile: mobileDeviceManager.getMobileDevices()) {
        edgeOrchestrator.assignHost(mobile);
    }

    CloudSim.startSimulation();
}
```

Figure 5: Start Simulation

5.1.1. Create Mobile Device List

The CreateMobileDeviceList() method belongs to the MobileDeviceManager class. It takes the number of mobile devices as a parameter and generates a list of mobile devices for the MobileDeviceManager class. This method also set the default location information for each mobile device.

```java
/**
 * create mobile devices
 * @author Qian
 *  @param number
 */
public void creatMobileDeviceList(int number) {
    mobileDevices = new ArrayList<>();
    List<EdgeTask> taskList = SimManager.getInstance().getLoadGeneratorModel().getTaskList();
    for (int i =0; i < number; i++) {
        MobileDevice mb = new MobileDevice(i, taskList.get(i).taskType);
        Location lc = ((GPSVectorMobility)SimManager.getInstance().getMobilityModel()).getLastMobileDeviceLocation(i);
        mb.setLocation(lc);
        mobileDevices.add(mb);
    }
}
```

Figure 6: Create Mobile Device List Method

5.1.2. Assign Host

After mobile devices creation, the SimManager assigns a host to each mobile device. The SimManager uses a for loop to iterate over every mobile device and lets the Orchestrator call the assignHost() method to allocate fog host to the mobile device.

```java
    for (MobileDevice mobile: mobileDeviceManager.getMobileDevices()) {
        edgeOrchestrator.assignHost(mobile);
    }
```

Figure 7: Assign Host

Different orchestrators use different ways to allocate fog host to a mobile device. The common thing is that this method verifies whether the host is capable to host tasks by calling the goodHost() method from EdgeOrchestrator. The goodHost() method verifies three important factors for each host: Million instructions per second (MIPS), Bandwidth, and Latency. For MIPS, the host should have enough CPU resource to process the incoming tasks from mobile. Bandwidth indicates that the fog host should have enough bandwidth to transfer data between the mobile device and itself. Network latency should satisfy the minimum latency requirement of the application which is running on the mobile device.

Additionally, the goodHost() method verifies the bandwidth factor for each node in the path between the mobile device and fog node host. It guarantees the successful data transferring between the mobile device and fog host.

```
protected static boolean goodHost(EdgeHost host, MobileDevice mb) {
    System.out.print(host.getId()+" ");
    if (!host.isMIPSAvailable(mb) || !host.isBWAvailable(mb) || !host.isLatencySatisfactory(mb)) {
        return false;
    }
    LinkedList<NodeSim> path = ((ESBModel)SimManager.getInstance().getNetworkModel()).findPath(host, mb);
    for (NodeSim node: path) {
        EdgeHost tempHost = SimManager.getInstance().getLocalServerManager().findHostByWlanId(node.getLocation().getServingWlanId());
        if (!tempHost.isBWAvailable(mb)) {
            return false;
        }
    }
    System.out.println(" is good.");
    return true;

}
```

Figure 8 Good Host Method

5.1.3. Reserve the resource in the host.

Once the host is assigned to the mobile device, it calls the makeReservation() method to book resources in the host for the execution of tasks. These resources include bandwidth and CPU resource. The mobile device also reserves the bandwidth from the intermediary fog nodes. The reserved bandwidth should be sufficient for successful data transmission between nodes.

```
public void makeReservation() {
    host.makeReservation(this);
    for (NodeSim node: path) {
        EdgeHost interHost = SimManager.getInstance().getLocalServerManager().findHostByWlanId(node.getLocation().getServingWlanId());
        interHost.reserveBW(this);
    }
    setAssignHostStatus(SimLogger.TASK_STATUS.ASSIGNED_HOST);
}
```

Figure 9: Make Reservation Method

5.2. Orchestrators

In PFogSim we pre-define several orchestrators for testing and verifying the simulation. These orchestrators are Centralize, Cloud Only, Edge Only, Fixed Node, and Local Only.

Generally, an orchestrator is a strategy that defines which fog nodes to handle the tasks. For example, if we apply Cloud-Only orchestrator, the system would only send tasks to the cloud.

5.2.1. Central Orchestrator

Central Orchestrator deploys the tasks with optimized cost. Generally, it predicts the cost for every fog node and picks the lowest-cost fog node to execute tasks.

At the initialization stage, Central Orchestrator creates a table. This table contains the optimized path from one fog node to another.

At the host assignment stage, Central Orchestrator calculates cost from the mobile device to every fog node. Then all fog nodes are sorted in ascending order of their cost. Next, the orchestrator checks every fog node in the list until the first available optimized cost found. Then the orchestrator assigns this fog node to the mobile device.

5.2.2. Edge-Only Orchestrator

Edge-Only Orchestrator deploys tasks only to edge nodes. We store all edge nodes in a list and sort them by distance or latency between the mobile device and hosts using Radix sort algorithm. Based on distance and latency, we propose two orchestrators: Edge-By-Distance and Edge-By-Latency.

Edge-By-Distance Orchestrator

This Edge-Only Orchestrator sorts edge nodes by the distance between the mobile device and edge node. It assigns the shortest distance edge node to the mobile device.

Edge-By-Latency Orchestrator

This Edge-Only Orchestrator selects the smallest latency edge node as host for the mobile device. The latency is the network latency between the mobile device and the edge node. The network latency contains congestion delay and propagation delay.

5.2.3. Cloud-Only and Fixed-Node Orchestrator

Cloud-Only and Fixed-Node Orchestrator both deploy tasks to one host. In Cloud-Only Orchestrator, all the tasks are sent to the public cloud. For Fixed-Node Orchestrator, tasks are sent to City Center of Chicago which is a fog node defined in the network topology.

5.3. Cost Calculation

In the cost calculation module, first we calculate the cost of every single task, then we calculate the average cost of all tasks.

5.3.1. For each task

After a finished task is transferred back to the mobile device, the MobileDeviceManager class calculates the cost of this task. The cost includes execution cost and bandwidth cost. Execution cost is defined by money spent on fog node which processes tasks. Bandwidth cost is the data transmission cost. It includes all fog nodes through which the data actually passes.

$$Cost_{task} = Cost_{execute} + \sum Cost_{transfer}$$

5.3.2. For average cost

After the task is finished, the SimLogger class records the cost information of each task. When the simulation is done, the SimLogger computes the average cost. The average cost is defined by the sum of all task cost divided by the number of successful tasks.

$$Cost_{average} = \frac{\sum Cost_{task}}{n}$$

Chapter 6 Testing and Evaluation

6.1. Service Placement:

The following graphs show the summary of the fog layer on which tasks were executed. All tasks were generated by 500 mobile devices.

6.1.1. Cloud-Only Orchestrator

For the Cloud-Only Orchestrator, the simulator is expected to deploy all the tasks to the cloud node. The testing result shows that all tasks were executed on the cloud. This proves that PFogSim works correctly for Cloud-Only Orchestrator.

Table 1: Cloud-Only Orchestrator (Tasks Distribution)

Fog Layer	Executed Tasks	Percentage
Layer 1	0	0
Layer 2	0	0
Layer 3	0	0
Layer 4	0	0
Layer 5	0	0
Layer 6	0	0
Layer 7	3315378	1

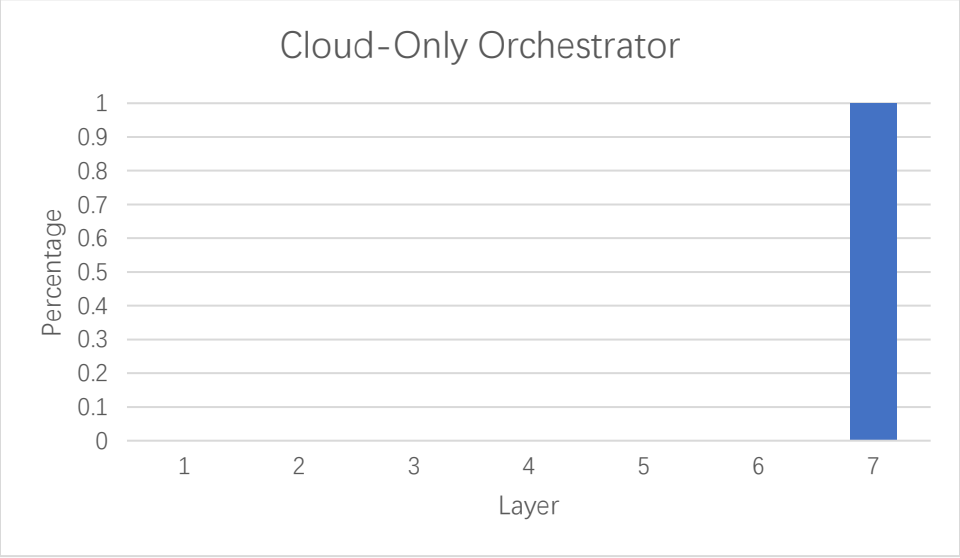

Figure 10: Cloud-Only Orchestrator (Tasks Distribution)

6.1.2. Fixed-Node Orchestrator

On Fix-Node Orchestrator, we assigned the Fog Node of City Hall as the only host for all tasks. The City Node is located on Layer 6. In the result data, it shows that all tasks were executed at layer 6. This is what we expected and verifies PFogSim accomplishing the service placement for Fixed-Node Orchestrator.

Table 2: Fixed-Node Orchestrator (Tasks Distribution)

Fog Layer	Executed Tasks	Percentage
Layer 1	0	0
Layer 2	0	0
Layer 3	0	0
Layer 4	0	0
Layer 5	0	0
Layer 6	3329544	1
Layer 7	0	0

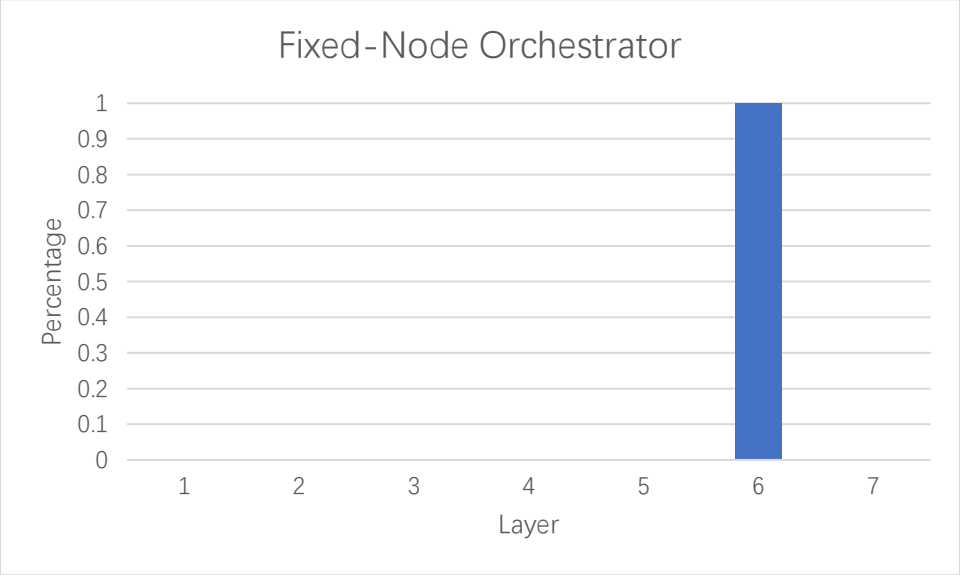

Figure 11: Fixed-Node Orchestrator (Tasks Distribution)

6.1.3. Edge-Only Orchestrator

For Edge-Only Orchestrator, all tasks should be executed by edge nodes on layer 1. The testing result shows that all tasks are submitted to layer 1 on Edge-Only Orchestrator. Table 4 and Figure 12 show the distribution of tasks of Edge-Only by Distance. Table 5 and Figure 13 show the result of Edge-Only by Latency.

Table 3: Edge-Only by Distance Orchestrator (Tasks Distribution)

Fog Layer	Executed Tasks	Percentage
Layer 1	856421	1
Layer 2	0	0
Layer 3	0	0
Layer 4	0	0
Layer 5	0	0
Layer 6	0	0
Layer 7	0	0

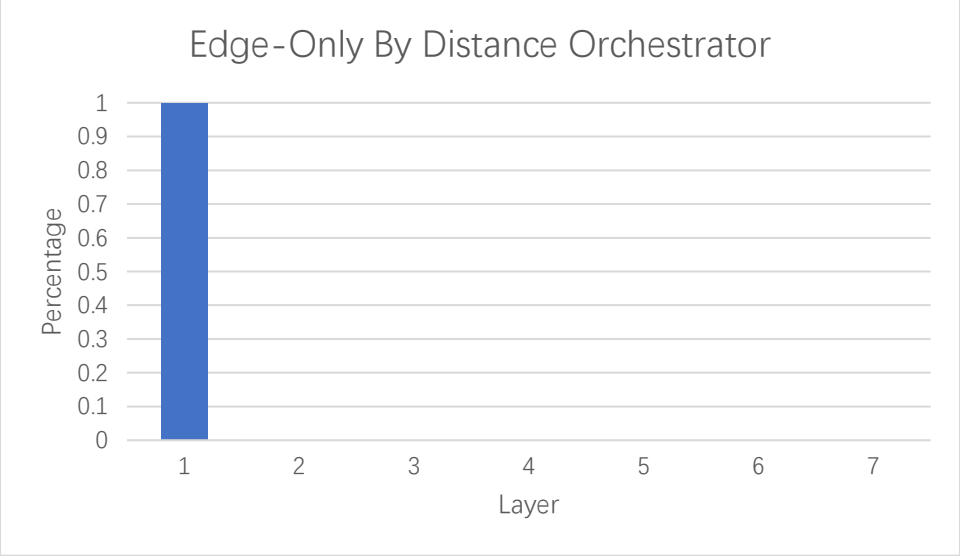

Figure 12: Edge-Only by Distance Orchestrator (Tasks Distribution)

Table 4: Edge Only by Latency Orchestrator (Task Distribution)

Fog Layer	Executed Tasks	Percentage
Layer 1	858027	1
Layer 2	0	0

Layer 3	0	0
Layer 4	0	0
Layer 5	0	0
Layer 6	0	0
Layer 7	0	0

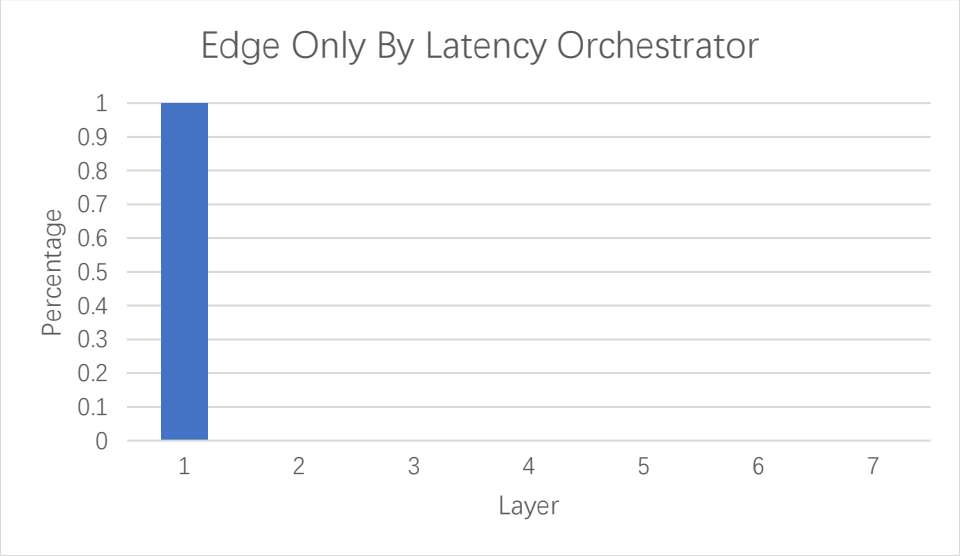

Figure 13: Edge Only by Latency Orchestrator (Task Distribution)

6.1.4. Central Orchestrator

Central Orchestrator allocates the task to a cost optimized fog node. Figure 14 shows 99% of tasks were executed between layers 2 to 7. Figure 15 shows that Central Orchestrator produces the cost-optimized result.

Table 5: Central Orchestrator (Tasks Distribution)

Fog Layer	Executed Tasks	Percentage
Layer 1	0	0
Layer 2	552789	0.165956
Layer 3	1156897	0.347319
Layer 4	964069	0.289429
Layer 5	652089	0.195767
Layer 6	2136	0.000641
Layer 7	2958	0.000888

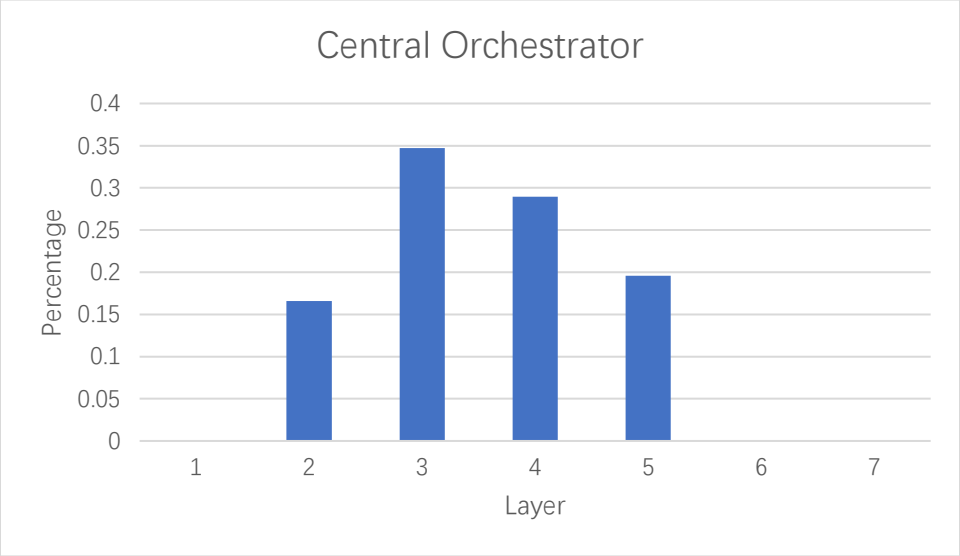

Figure 14: Central Orchestrator Tasks Distribution

6.2. Cost evaluation

In this experiment, we set up six test groups. Each group implements one type of orchestrator. We execute ten simulations for every group. The number of mobile devices was increased by 500 for each simulation.

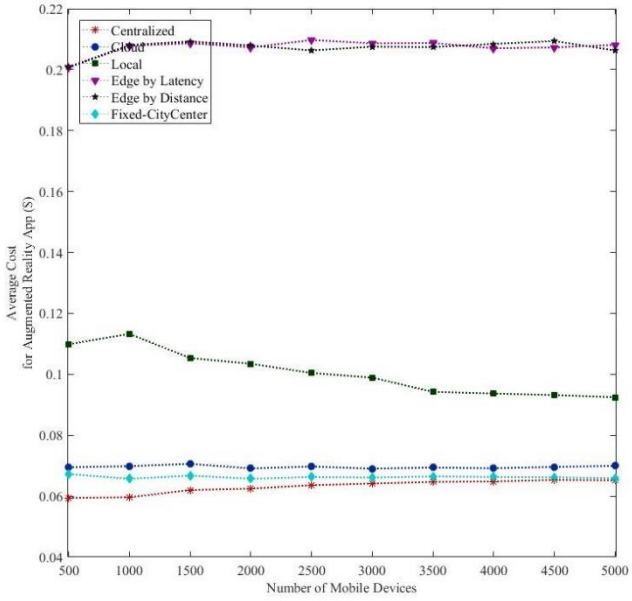

Figure 15: Average Cost for Augmented Reality App

Figure 15 indicates that Central Orchestrator has the lowest cost and Edge Only Orchestrators have the highest cost. The cost of Fixed-Node and Cloud-Only Orchestrator is close to Central Orchestrator and a little bit higher than Central Orchestrator. Local-Only Orchestrator lies between Edge-Only Orchestrator and Cloud-Only Orchestrator. All these results match the theoretical expectation. This indicates that the cost calculation function works correctly for our simulator.

6.3. Result Graphs

Figure 16 to Figure 22 were generated by PFogSim. These graphs contain most of the information which is generated by the system during the simulation. This information includes task distribution, cost (Figure 15), distance, hops, tasks failure rate, network delay, task processing time, etc.

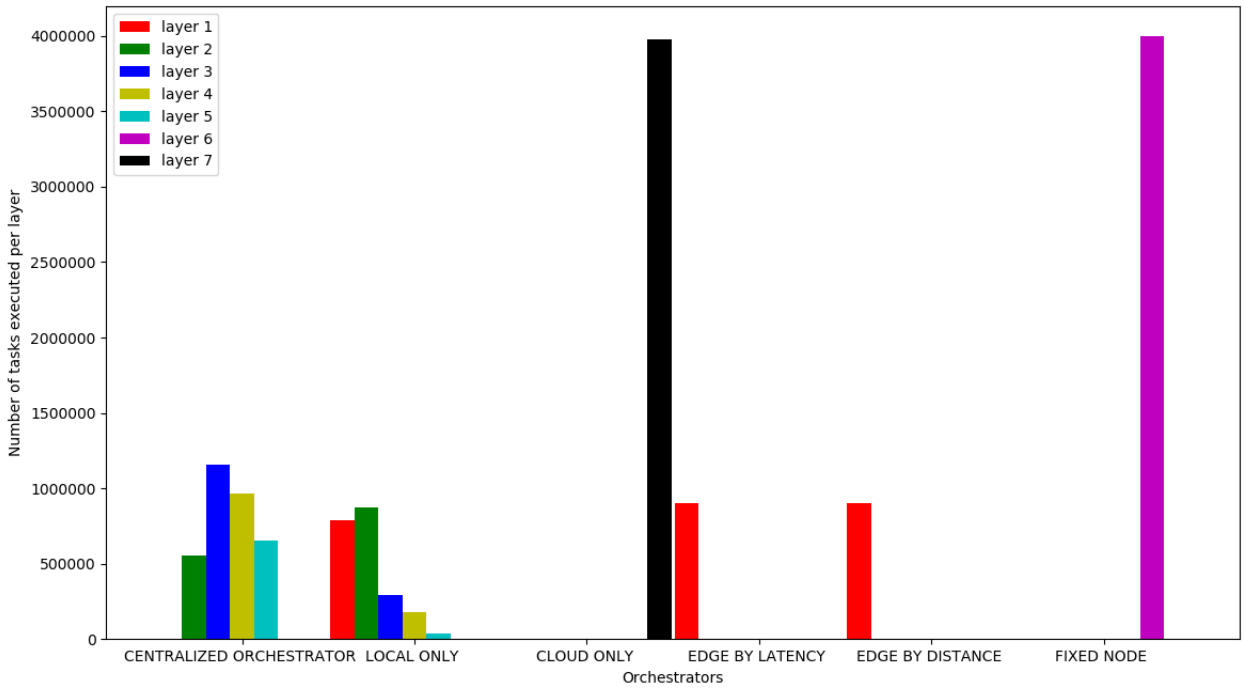

Figure 16: Task Distribution Per Layer for Augmented Reality App

Figure 16 shows the task distribution for Augmented Reality App when applying different orchestrators. We set the mobile device number to 500.

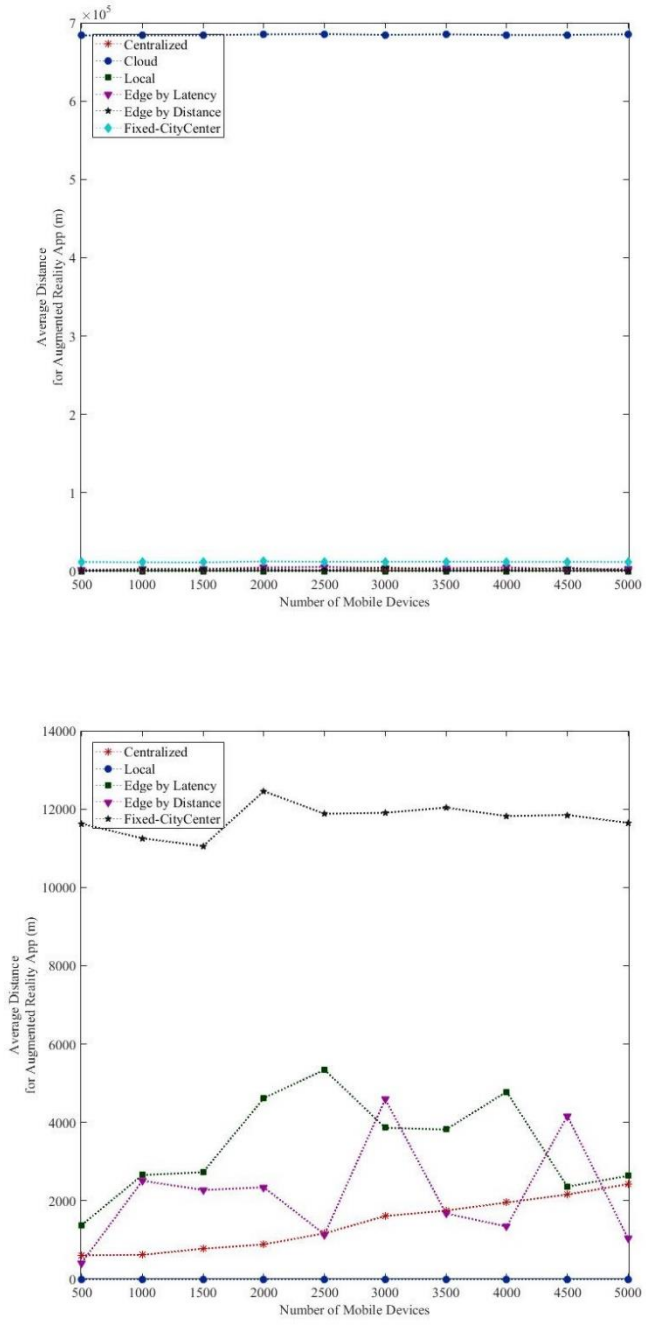

Figure 17: Average Distance Between Mobile Device and Host

Figure 17 shows the average distance between mobile devices and fog nodes which processes the task for the mobile device. The graph demonstrates that the distance between the cloud node and mobile device is about 700 km and the distance between City Hall and mobile device is about 12 km. This correctly matches the real data imported from the configuration file.

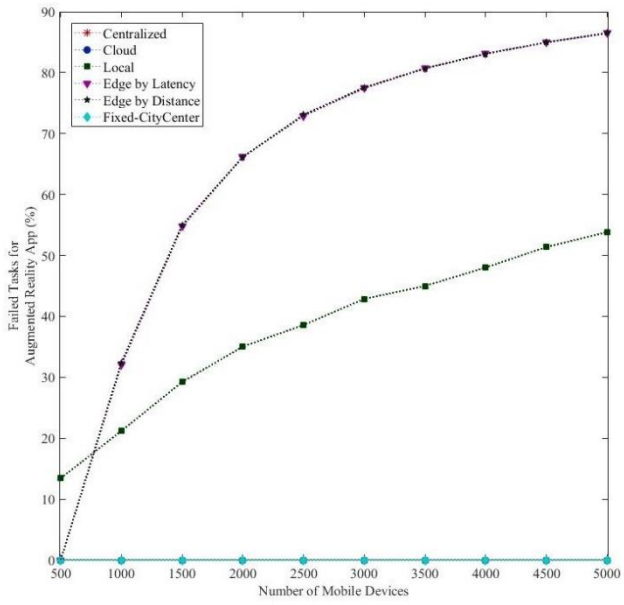

Figure 18: Tasks Failure Rate

Figure 18 shows the failure rate of tasks in different orchestrators. Two major causes for task failure are lack of computing resources and network resources (bandwidth). The Edge-Only and Local-Only Orchestrators tend to deploy tasks to lower-layer fog nodes which have limited resources of computing and bandwidth. As the number of mobile devices increases, the number of tasks raises. While the lower-layer fog nodes cannot handle all these tasks, the rate of task failure raises.

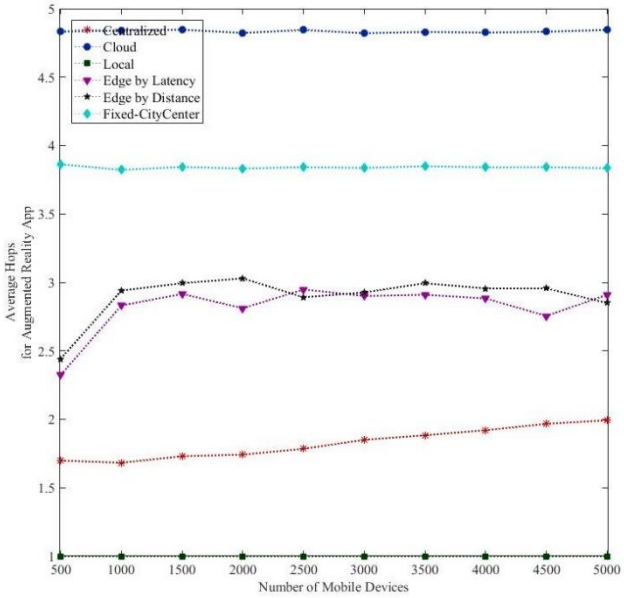

Figure 19: Average Number of Hops Between Mobile Device and Host

Figure 19 shows the number of average hops between mobile devices and fog nodes hosting the application service. The hops are fog nodes where data passes through during the transmission between the mobile device and the host. This is justified in case a longer distance between the mobile device and the host usually introduces the higher number of hops.

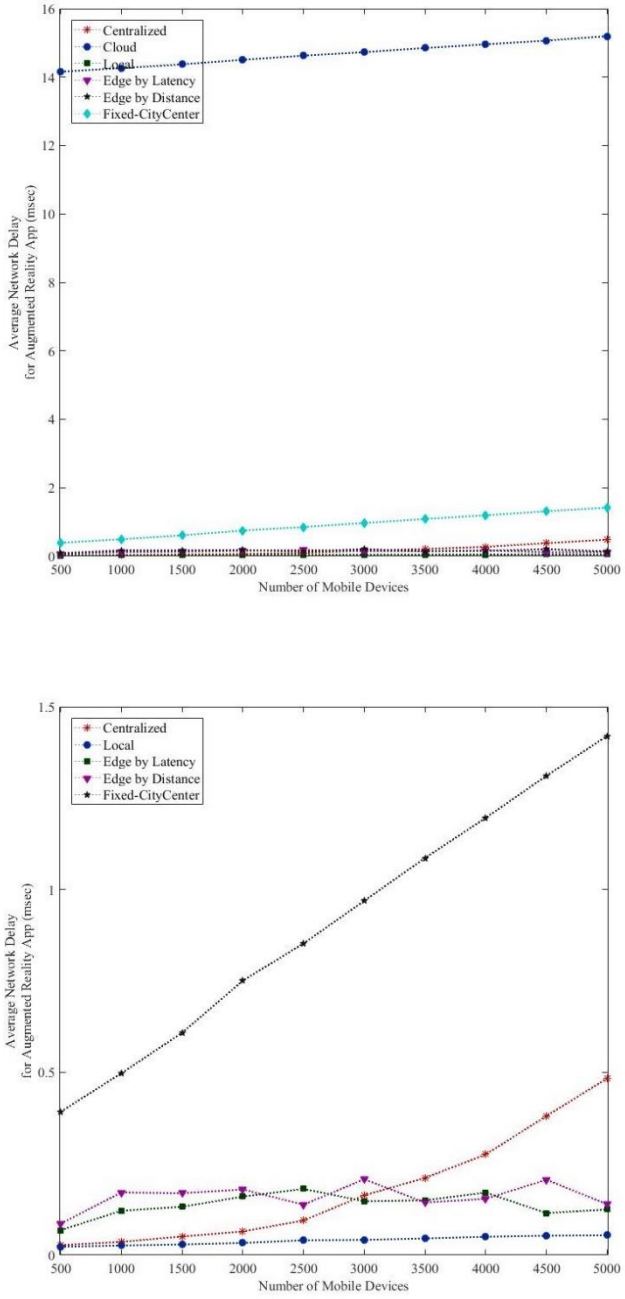

Figure 20: Average Network Delay Between Mobile Device and Host

Normally, In the real-world a long distance causes high network delay. Figure 20 shows this, e.g., the network delay of the cloud and the City Hall which are higher than others as their

longest distances between them and mobile devices.

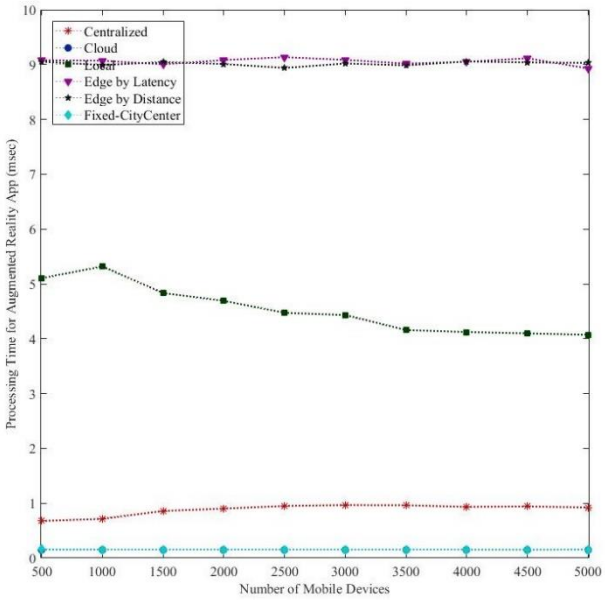

Figure 21: Average Processing Time

Figure 21 illustrates the average time of processing tasks in every host. Because lower-layer fog nodes have fewer computing resources than higher-layer fog nodes, the time consumed by edge nodes (first layer fog nodes) is extremely higher than that of other nodes.

Chapter 7 Conclusion

This thesis discusses a Java-based fog computing simulator called PFogSim. The workflow and general components of PFogSim have been introduced. PFogSim supports real-world large-scale fog node architecture. Our simulator is fully tested and evaluated by using multiple orchestrators across cloud computing, edge computing, and fog computing.

During the test, we identify some improvements for PFogSim. In general, to reduce the randomness of the single result, multiple simulations are required. The running time of a single simulation should not take too much time. However, with the current version of PFogSim, the running time of single simulation is longer than 12 hours with 5000 mobile devices on Centralized Orchestrator. Also, when multiple simulations are running at the same time, they consume a large amount of memory. To address these issues, we may optimize the architecture of the simulator.

Another improvement could be completed is that we need to separate the sender and receiver of data. In the real world, the sender and receiver of data may not be the same mobile device. For example, when you are working, you want to use your smartphone to remotely control the A/C in your home. In this situation, the data sender is your smart device, but the receiver is your A/C at home. Current PFogSim only supports the same data sender and receiver. In the future, we will add this feature to PFogSim.

References

[Shehenaz, 2019] Shehenaz Shaik, Qian Wang, Jacob Hall, and Clayton Johnson Sanjeev Baskiyar, “PFogSim: A Simulator for Performance Evaluation of Mobile and Hierarchical Fog Computing Environments”

[Data Portal, 2018] Chicago Data Portal, “City of Chicago”, https://data.cityofchicago.org/, 1-Jun-2018.

[Rodrigo, 2011] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, Cesar A. F. De Rose, and Rajkumar Buyya, “CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms”, *Software: Practice and Experience*, Volume 41, Number 1, Pages: 23-50, ISSN: 0038-0644, Wiley Press, New York, USA, January 2011.

[Antonio, 2017] Antonio Brogi and Stefano Forti, “QoS-aware Deployment of IoT Applications Through the Fog”, *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1185-1192, Oct. 2017.

[Antonio, 2017] Antonio Brogi, Stefano Forti, Ahmad Ibrahim, “How to Best Deploy Your Fog Applications, Probably”. In 2017 IEEE *1st International Conference on Fog and Edge Computing (ICFEC)*, 10.1109/ICFEC.2017.8

[Gupta, 2017] Gupta H, Vahid Dastjerdi A, Ghosh SK, Buyya R. “iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge, and Fog computing environments”, *Softw Pract Exper*. 2017;47: 1275–1296.

[C. Sonmez, 2017] C. Sonmez, A. Ozgovde, and C. Ersoy, "EdgeCloudSim: An environment for performance evaluation of Edge Computing systems," *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, Valencia, 2017, pp. 39-44. doi: 10.1109/FMEC.2017.7946405

[Shehenaz, 2018] Shehenaz Shaik and Sanjeev Baskiyar, “Hierarchical and Autonomous Fog Architecture”. *In ICPP 2018: 47th International Conference on Parallel Processing Companion Proceedings*, August 13-16, 2018, Eugene, OR, USA. ACM, New York, NY, USA, 8 pages. https://doi.org/ 10.1145/3229710.3229740