Simulating Fog Computing Applications Using iFogSim Toolkit

Kamran Sattar Awaisi, Assad Abbas, Samee U. Khan, Redowan Mahmud, and Rajkumar Buyya

Abstract Fog computing is a novel distributed computing paradigm that provides cloud-like services at the edge of the network. It emerges as an efficient paradigm to process the enormous amount of Internet of Things (IoT) data and can address the limitations of cloud-centric IoT models in terms of large end-to-end delays, and huge network bandwidth consumption. Recently, fog computing and IoT have been employed in several domains, including transportation, education, healthcare, and manufacturing industry. To imitate different complex application scenarios for these domains, a notable number of fog computing-based simulators has already been developed. Among them, iFogSim has attained significant attention because of its simplified interface and low complexity. In this article, we present a tutorial on how to use iFogSim toolkit to simulate four real-time case studies for (1) smart car parking, (2) smart waste management system, (3) smart coal mining industry, and (4) sensing as a service. This article is expected to assist the researchers in understanding and implementing various aspects of fog computing using the iFogSim toolkit.

Keywords Fog computing · iFogSim · Smart car parking · Smart waste management system · Smart mining industry

K. S. Awaisi · A. Abbas (✉)
COMSATS University Islamabad, Islamabad, Pakistan
e-mail: assadabbas@comsats.edu.pk

S. U. Khan
Mississippi State University, Mississippi, MS, USA
e-mail: skhan@msstate.edu

R. Mahmud · R. Buyya
Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Melbourne, VIC, Australia
e-mail: mahmudm@student.unimelb.edu.au; rbuyya@unimelb.edu.au

A. Mukherjee et al. (eds.), *Mobile Edge Computing*,
https://doi.org/10.1007/978-3-030-69893-5_22

1 Introduction

Internet of Things (IoT) has connected billions of devices across the world and is consistently promoting the realization of smart cyber-physical environments including smart factories, smart homes, smart transport, and smart healthcare. However, due to limited processing and storage capabilities of IoT devices, cloud computing is often used as the backbone platform to provide computational capacity and storage services to the IoT-enabled environments [1]. Nevertheless, cloud data-centers have some potential challenges, such as large end-to-end delays and huge network bandwidth consumption. These challenges pertinent to cloud computing impact the response time of latency-sensitive real-time applications, for example healthcare systems, traffic management, and fire control systems. Additionally, IoT devices can generate an enormous amount of data within a very short period. When every IoT device initiates sending these data to cloud servers, the performance of cloud services is more likely to degrade.

Fog computing extends the cloud services near the edges of the network and overcomes the challenges of cloud computing [2]. This new distributed computing paradigm has exhibited tremendous potential to effectively process the data generated by millions of IoT devices [3]. Since fog computing brings computations closer to the data generating devices, consequently the latency and network bandwidth utilization are significantly minimized [4]. Compared to the cloud data centers, fog nodes have less computational power and storage capacity. Therefore, fog and cloud computing paradigms work in an integrated manner to provide resources for large-scale IoT systems.

Since the fog computing systems involve fog nodes, cloud data centers, and IoT devices; therefore, the real-world implementation of fog scenarios for research purposes is very expensive [5]. In such situations, simulation and validation of fog scenarios with the help of toolkits are very beneficial. Currently, there are several simulation toolkits available, such as FogNetSim++ [6], Edgecloudsim [7], and iFogSim [8] for modeling and simulating the fog computing environments. Among these toolkits, iFogSim has significantly attracted the attention of the researchers and is being used to model a variety of fog computing cases. In this article, we aim at providing a tutorial on iFogSim to help the researchers quickly understand the fundamental concepts and the advanced implementation steps. To make the study more intriguing, we implement four real-time fog-based scenarios namely, (1) smart car parking system, (2) smart waste management system, (3) smart mining industry, and (4) sensing as a service in the iFogSim. The tutorial not only provides step by step installation guidelines but also contains instructions to simulate the scenarios and create devices, classes, and objects in the iFogSim. Moreover, the tutorial also presents the corresponding code snippets of all the case studies simulated in iFogSim. The remainder of the article is organized as follows: Sect. 2 briefly discusses the installation and setup of iFogSim. Section 3 presents case studies and code snippets whereas Sect. 4 concludes the paper.

2 Installation and Setup of iFogSim

iFogSim is a Java based open-source simulation tool for simulating fog computing scenarios. It is developed by Harshit Gupta and the team at the Cloud Computing and Distributed Systems (CLOUDS) Lab University of Melbourne Australia [8]. The following are the steps to download, install, and setup the iFogSim.

1. Download the iFogSim source code in the zip file from the GitHub https://github.com/Cloudslab/iFogSim.
2. Extract the iFogSim zip file and there will be a folder named *iFogSim-master*.
3. Make sure that you have installed Java Runtime Environment (JRE) or Java Development Kit (JDK) 1.7 or more.
4. Install Eclipse Mars or any latest release on the computer.
5. Define the workspace for the Eclipse Integrated Development Environment (IDE).
6. Create a new folder for the iFogSim in the Eclipse workspace and paste all the files and content of the *iFogSim-master* in this folder or you can simply copy the *iFogSim-master* folder and paste it into the workspace folder
7. Open the Eclipse IDE and create the new Java project.
8. Make sure that the name of the Java project is the same as the name of the folder as you have created in the workspace for iFogSim.
9. Now open the *src* of the project and explore the package *org.fog.test.perfeval*. In this package, you will find three example scenarios of iFogSim.
10. Open any example scenario, explore it, and run it. You will get the results on the console.

3 Case Studies

This section presents the four case studies that are implemented using the iFogSim toolkit. Section 3.1 presents the case study of a smart car parking system, Sect. 3.2 explains the smart waste management system, Sect. 3.3 describes the smart mining industry case study and Sect. 3.4 discusses the sensing as a service case study.

3.1 Smart Car Parking System

Most of the people are moving towards the cities due to better facilities and resources. Owing to the increasing population to the cities, the number of vehicles on the roads have increased enormously as the personal vehicles have become a significant transportation resource nowadays. Consequently, finding the vacant car parking space has become a potential issue in the populated areas. People spend a lot of time finding the vacant car parking space which essentially results in CO2 emission, time wastage, and fuel wastage. Parking problems have attracted more consideration in the past few years and many researches have proposed IoT based car parking solutions. We presented a fog based smart car parking architecture in [9] to solve the car parking issues by using fog computing. The fog-based car parking architecture consists of the following:

- Smart cameras
- Fog nodes
- Light Emitting Diode (LED) display screens
- A cloud server

The smart cameras are deployed in the parking lanes which take the image of the parking lanes and transmit the images to the fog node. On the fog node, we have implemented an image processing algorithm to identify those parking slots which are vacant. After detecting the vacant parking slots, the parking slots information is updated on the LED. The data is stored in the fog node for a limited amount of time, and then it is moved to the cloud server for permanent storage. When the vehicle arrives at the parking gate, the driver finds the vacant car parking space immediately and parks the vehicle on the desired location. The information on the LED is updated after every 5 s interval. The communication between the fog node and the cloud server is enabled through a proxy server. The fog-based car parking system is displayed in Fig. 1.

Building Scenario with iFogSim for Smart Parking System To simulate the smart car parking scenario, we need to create two modules in iFogSim i.e.

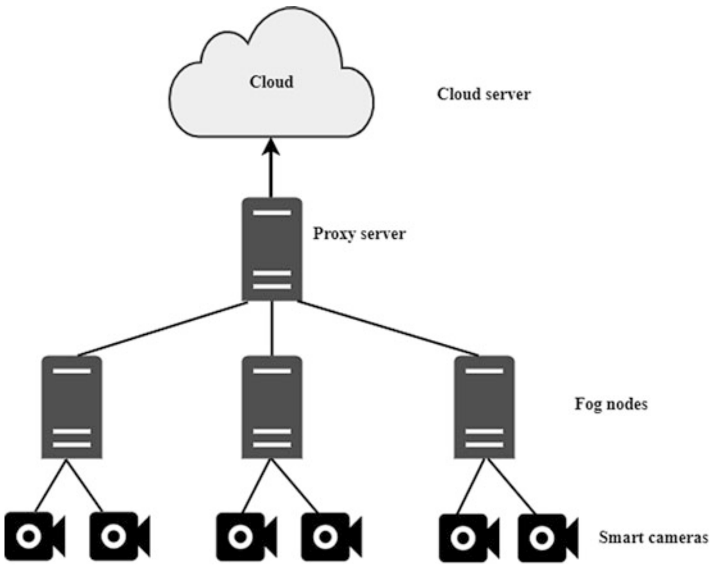

Fig. 1 Fog based smart car parking system

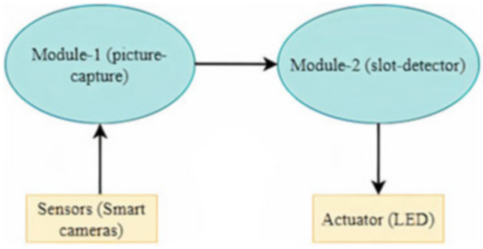

Fig. 2 Fog based smart car parking system application model of iFogSim

picture-capture and *slot-detector*. The *picture-capture* module is embedded in smart cameras. The smart cameras are programmed in such a way that it takes the pictures after a specific time interval of 5 s and transmits the images to the fog node. We can attach the micro-controller device with the smart cameras to establish a connection with the fog node.

Moreover, we create a proxy server and a cloud server. The proxy server enables the communication between the fog node and the cloud server. The cameras here act as the sensor as well. In iFogSim, when we create any device which takes the input to the system for processing, we call it a sensor and any device which receives the output after processing is termed as the Actuator. The sensors, actuators, and fog devices are created in iFogSim using their respective classes. In the smart parking system scenario, the cameras are created and attached to the fog node. Figure 2 depicts the data flow of the smart parking application model. The *picture-capture* module is created in smart cameras. It is programmed to capture the pictures of parking lane after every 5 s.

The pictures are handed over to the second module which is a *slot-detector* and it detects the vacant parking slots. In iFogSim, any computation elements are called modules.

Building Simulation with iFogSim for Smart Car Parking System The iFogSim provides built-in classes to create fog nodes, sensors, and actuators. It also takes care of resource allocation and management policies. The following classes will be used to create a smart car parking scenario in iFogSim.

1. **FogDevice:** This class provides a constructor to create the fog devices and to define the hardware properties of the fog devices i.e. node name (name of the device to be used in simulation), MIPS (Million Instructions Per Second), RAM (main memory of the fog node), uplink bandwidth, downlink bandwidth, level (hierarchy level of the device), ratePerMips (cost rate per MIPS used), busyPower (the amount of power consumed when the fog node is in busy state), and idlePower (the amount of power used when the fog node is in the idle state). When we create the fog device, all these parameters are assigned values. In our implementation of the case studies, all the computational devices are created using the FogDevice class.
2. **Sensor:** By using the sensor class, we create IoT devices in iFogSim. While creating the sensors, we define the gateway device id and setup link latency.

Gateway device is the device with whom the sensor is attached and any devices, such as a router, fog node, or a proxy can serve as the Gateway. Setup link latency is the latency time to create a connection between the sensor and fog device. Normally, we set the setup link latency time between 1 to 3 ms.

3. **Actuator:** Actuator class allows to create the objects in iFogSim that are used to display the output or any information. In the smart car parking scenario, the LED is created actuator because the vacant car parking slot position will be displayed on the LEDs. The actuator needs to be connected with any gateway device. The gateway device sends the data to actuator. Therefore, when we create actuator in iFogSim, we define the gateway device id and setup link latency.

A new class in *org.fog.test.perfeval* package is required to create for simulating this scenario in iFogSim. The *FogDevice* class lets you to create fog nodes with different configurations by providing a constructor. A code snippet to create heterogeneous fog devices is given below:

Code Snippet-1 This code snippet is to be placed in the **main** class.

```
//Here we are creating a list for fog devices.
static List<FogDevice> fogDevices = new ArrayList<FogDevice>();
static List<Sensor> sensors = new ArrayList<Sensor>();
static List<Actuator> actuators = new ArrayList<Actuator>();
static int numOfAreas = 7; //the  number  of  fog  nodes
static int numOfCamerasPerArea1=10;
// the  number  of  cameras per fog node.
static double CAM_TRANSMISSION_TIME = 5; //time interval
private static boolean CLOUD = false;
private static void createFogDevices(int userId, String appId) {
FogDevice cloud = createFogDevice("cloud", 44800, 40000,
100, 10000, 0, 0.01, 16*103, 16*83.25);
cloud.setParentId(-1);
fogDevices.add(cloud);
FogDevice proxy = createFogDevice("proxy-server", 2800, 4000,
10000, 10000, 1, 0.0, 107.339, 83.4333);
proxy.setParentId(cloud.getId());

double costPerStorage
proxy.setUplinkLatency(100);
fogDevices.add(proxy);
for(int i=0;i<numOfAreas;i++){
addArea(i+"", userId, appId, proxy.getId());
}
}

private static FogDevice addArea(String id, int userId,
String appId, int parentId){
FogDevice router = createFogDevice("a-"+id, 2800, 4000,
1000, 10000, 2, 0.0, 107.339,83.4333);
fogDevices.add(router);
router.setUplinkLatency(2);
for(int i=0;i<numOfCamerasPerArea1;i++){
String mobileId = id+"-"+i;
FogDevice camera = addCamera(mobileId, userId,
appId, router.getId());
camera.setUplinkLatency(2);
fogDevices.add(camera);
}
router.setParentId(parentId);
return router;
}

private static FogDevice addCamera(String id, int userId,
String appId, int parentId){
FogDevice camera = createFogDevice("c-"+id, 500, 1000, 10000,
10000, 3, 0, 87.53, 82.44);
camera.setParentId(parentId);
Sensor sensor = new Sensor("s-"+id, "CAMERA", userId, appId, new
DeterministicDistribution(CAM_TRANSMISSION_TIME));
sensors.add(sensor);
Actuator ptz = new Actuator("ptz-"+id, userId,
appId, "PTZ_CONTROL");
actuators.add(ptz);
sensor.setGatewayDeviceId(camera.getId());
sensor.setLatency(40.0);
ptz.setGatewayDeviceId(parentId);
ptz.setLatency(1.0);
return camera;
}
```

Code Snippet-2 This code snippet is to be placed in the newly created **main** class. In this code snippet we are creating the modules on fog devices and assigning these modules to fog nodes. Figure 3 illustrates the physical topology of the car parking system in iFogSim that we have created in the code-snippet 1 and code-snippet 2.

```
private static Application createApplication
(String appId, int userId){
Application application =
Application.createApplication(appId, userId);
application.addAppModule("picture-capture", 10);
application.addAppModule("slot-detector", 10);
// adding edge from CAMERA (sensor) to picture-capture module
carrying tuples of type CAMERA
application.addAppEdge("CAMERA", "picture-capture", 1000, 500,
"CAMERA", Tuple.UP,
AppEdge.SENSOR);
application.addAppEdge("picture-capture", "slot-detector",
1000, 500, "slots",Tuple.UP, AppEdge.MODULE);
// adding edge from Slot Detector to PTZ CONTROL (actuator)
application.addAppEdge("slot-detector", "PTZ_CONTROL", 100,
28, 100, "PTZ_PARAMS",
Tuple.UP, AppEdge.ACTUATOR);
application.addTupleMapping("picture-capture", "CAMERA", "slots",
new FractionalSelectivity(1.0));
application.addTupleMapping("slot-detector", "slots",
"PTZ_PARAMS", new FractionalSelectivity(1.0));
```

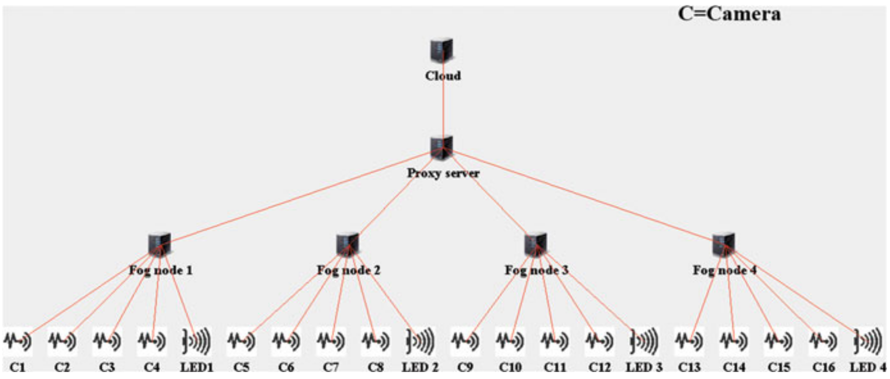

Fig. 3 iFogSim topology of smart car parking system

```
final AppLoop loop1 = new AppLoop(new ArrayList<String>()
{{add("CAMERA");
add("picture-capture");add("slot-detector");
add("PTZ_CONTROL");}});
List<AppLoop> loops = new ArrayList<AppLoop>(){{add(loop1);}};
application.setLoops(loops);
return application;
}
```

3.2 Smart Waste Management System

With the rapid increase in population and urbanization, the waste generation level in the cities is increasing day by day. Waste is generated by humans and by every living thing. Wherever life and human beings are, the waste will be generated there. According to the World Bank Report published in 2012 [10], the solid waste management generation level was about 1.3 billion tons per year and it will reach 2.2 billion tons per year in 2025. Nonetheless, the generation of waste cannot be prevented; however, introducing smart measures to collect and manage the generated waste can help in providing health environments [11]. The timely collection of waste not only prevents the spread of several diseases but also plays its part in keeping the environment green, clean, and healthy. A cloud-based waste management system is presented in [12] where authors used the cloud server to automate the waste management system. The smart waste bins are connected to the cloud server. The waste level information is transmitted to the cloud server after a specific time interval. If the waste level has reached the threshold value, then the waste is collected otherwise no action is taken unless the waste bin generates an alert indicating that the threshold level is reached.

Cloud computing is suffering from many problems like a large end to end delay and huge network bandwidth consumption [13]. In case, if we increase the number of smart waste bins connected with the cloud server, then there will be network congestion and it will not be easy to handle and manage the waste data from all the areas. In a certain community belonging to developing countries, there is a need to place smart waste bins at different points in streets that people can use to throw the waste. In this case, if all the waste bins are connected to the cloud server, it will cause latency and network usage problems. The best possible solution for this is to geographically partitions different areas and subsequently connect the waste bins of a particular area to a specific data management server. Consequently, deployment of fog nodes in the waste management system will make it more efficient and easily manageable for all the concerned stakeholders.

Building Scenario with iFogSim for Smart Waste Management System In the proposed fog-based waste management system, there are different waste bins. Each waste bin is allocated to a different kind of waste, such as kitchen waste, plastic, paper and cardboard, and metal. Smart waste bins will be placed in rural areas to manage and collect waste properly and efficiently. Each waste bin is equipped with the sensor (Ultrasonic sensor HC-SR04) to notify the waste level. Ultrasonic sensors emit the waves at a specific frequency and then wait for the wave to be reflected back. Based on the distance and the time taken back after reflection, we measure the percentage or level of waste in the bin. Figure 4 depicts the fog-based waste management system. The smart waste bins are connected to the fog server via a router device.

Figure 5 shows the data flow application model of the fog-based smart waste management system. In this scenario, five modules will be created. *Waste-info-module* collects the waste level information of the waste bins. The module passes the data to the *master-module* which is basically responsible for managing the waste information on the fog node. We create the separate modules for all the stakeholders, such as healthcare department, recycling unit, and head of the municipal authority to disseminate the waste collection information among the relevant collection staff. These modules represent the logical placement and creation of connection for each stakeholder at the fog node. The location tracking feature of waste collectors can be implemented in real time implementation of smart waste management system.

Building Simulation with iFogSim for Smart Waste Management System To simulate the smart waste management scenario, first make a new class in *org.fog.test.perfeval package*.

Code Snippet-3 This code snippet is to be placed in **main** class. In this code snippet we are adding the modules to the fog devices. Cloud mode is set to FALSE, and all the computational operations will be performed at fog nodes. In case if cloud mode is set to TRUE, then all the modules will be placed on cloud server, and all the computations will be performed on cloud server. There is no need to change the module placement. The code is commented so that you can develop the understanding of code. This code snippet should be added in the main method after initializing the module mapping.

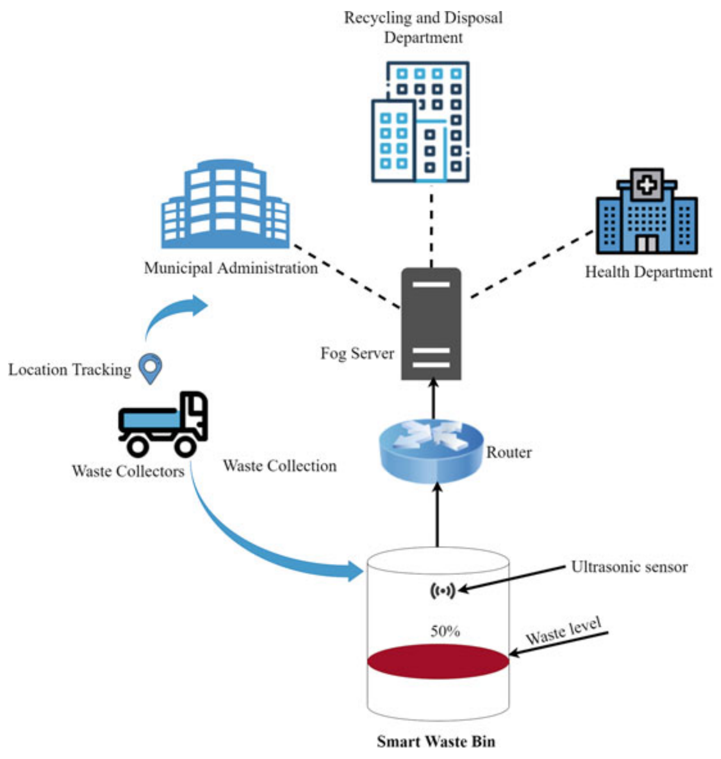

Fig. 4 Fog-based smart waste management system architecture

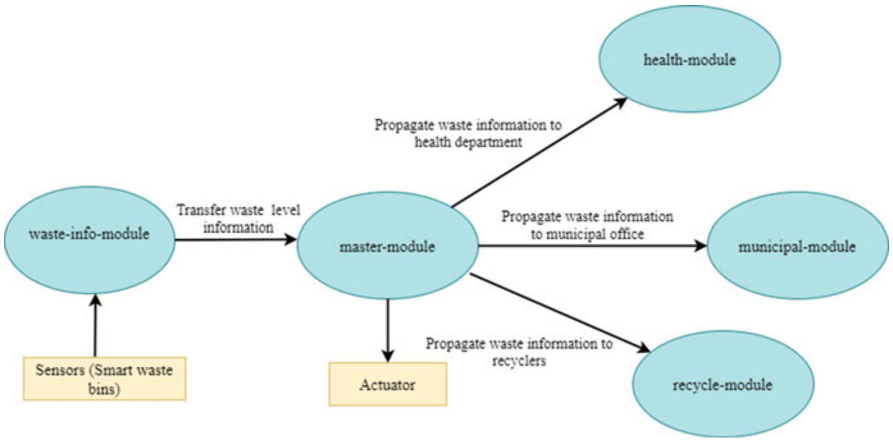
health-module
Propagate waste information to health department
waste-info-module
Transfer waste level information
master-module
Propagate waste information to municipal office
municipal-module
Sensors (Smart waste bins)
Actuator
Propagate waste information to recyclers
recycle-module

Fig. 5 Fog based smart waste management system application model of iFogSim

```
    //Create the list of fog devices
static List<FogDevice> fogDevices = new ArrayList<FogDevice>();
//Create the list of sensors
static List<Sensor> sensors = new ArrayList<Sensor>();
//Create the list of actuators
static List<Actuator> actuators = new ArrayList<Actuator>();
//Define the number of areas
static int numOfTotalAreas = 10;
//Define the number of waste bins with each fog nodes
static int numOfBinsPerArea=1;
//We are using the fog nodes to perform the operations.
//cloud is set to false
private static boolean CLOUD = false;
public static void main(String[] args) {
Log.printLine("Waste Management system...");
try {
Log.disable();
int num_user = 1; // number of cloud users
Calendar calendar = Calendar.getInstance();
boolean trace_flag = false; // mean trace events
CloudSim.init(num_user, calendar, trace_flag);
String appId = "swms"; // identifier of the application
FogBroker broker = new FogBroker("broker");
Application application = createApplication(appId,
broker.getId());
application.setUserId(broker.getId());
createFogDevices(broker.getId(), appId);
Controller controller = null;
ModuleMapping moduleMapping = ModuleMapping.createModuleMapping();
for(FogDevice device : fogDevices){
if(device.getName().startsWith("b")){
// names of all Smart Bins start with 'b'
moduleMapping.addModuleToDevice("waste-info-module",
device.getName());
// mapping
waste information module on waste bins
}
}
for(FogDevice device : fogDevices){
if(device.getName().startsWith("a")){
// names of all fog devices start with 'a'
// mapping master-module on area devices.
moduleMapping.addModuleToDevice("master-module",
device.getName());
// mapping health-module on area devices
moduleMapping.addModuleToDevice("health-module",
device.getName());
// mapping recycle-module on area devices.
moduleMapping.addModuleToDevice("recycle-module",
device.getName());
// mapping municipal-module on area devices.
moduleMapping.addModuleToDevice("municipal-module",
device.getName());
}
}
if(CLOUD){ // if the mode of deployment is cloud-based
// placing all instances of master-module in the Cloud
moduleMapping.addModuleToDevice("master-module", "cloud");
// placing all instances of health-module in the Cloud
moduleMapping.addModuleToDevice("health-module", "cloud");
//placing all instances of recycle-module in the Cloud
moduleMapping.addModuleToDevice("recycle-module", "cloud");
// placing all instances of municipal-module in the Cloud
moduleMapping.addModuleToDevice("municipal-module", "cloud");
}

controller = new Controller("master-controller",
fogDevices, sensors, actuators);
controller.submitApplication(application,
(CLOUD)?(new ModulePlacementMapping(fogDevices,
application, moduleMapping))
:(new ModulePlacementEdgewards(fogDevices, sensors,
actuators, application, moduleMapping)));
TimeKeeper.getInstance().setSimulationStartTime(
Calendar.getInstance().
getTimeInMillis());

CloudSim.startSimulation();

CloudSim.stopSimulation();

Log.printLine("waste management simulation finished!");
}
 catch (Exception e) {
e.printStackTrace();
Log.printLine("Unwanted errors happen");
}
}
```

After adding this code snippet initialize the controller object.

Code Snippet-4 This code snippet is to be placed in **main** class. In this code snippet we are creating the heterogeneous fog devices. The fog nodes will be placed in geographical distributed location and we will also need the location of the smart bin therefore, while creating the fog nodes and smart bins, we are setting the x-coordinate and y-coordinate value of the fog nodes and smart bin. In case of smart bin, the location awareness will help us to know that in which area or street, a particular waste bin is placed. Moreover, the fog node location will help us to be aware of the location of the particular fog node. In code snippet-6, we have created a method which generates the random values and these random value are then assigned as x and y coordinate to the waste bins and fog nodes.

```
private static void createFogDevices(int userId, String appId) {
FogDevice cloud = createFogDevice("cloud", 44800, 40000, 100,
10000, 0, 0.01, 16*103, 16*83.25);
cloud.setParentId(-1);
fogDevices.add(cloud);
FogDevice router = createFogDevice("proxy-server", 7000, 4000,
10000, 10000, 1, 0.0,
107.339, 83.4333);
router.setParentId(cloud.getId());
// latency of connection between proxy server and cloud is 100 ms
router.setUplinkLatency(100.0);
fogDevices.add(router);
for(int i=0;i<numOfTotalAreas;i++){
addArea(i+"", userId, appId, router.getId());
}
}
//creating the fog nodes for each area
private static FogDevice addArea(String id, int userId,
String appId, int parentId){
FogDevice area_fognode = createFogDevice("a-"+id, 5000, 4000,
10000, 10000, 3, 0.0, 107.339, 83.4333);
fogDevices.add( area_fognode);
area_fognode.setUplinkLatency(1.0);
for(int i=0;i<numOfBinsPerArea;i++){
String mobileId = id+"-"+i;
FogDevice bin = addBin(mobileId, userId,
appId, area_fognode.getId());
bin.setUplinkLatency(2.0);
fogDevices.add(bin);
}
//assigning x coordinate value to the fog node
area_fognode.setxCoordinate(getCoordinatevalue(10));
//assigning y coordinate value to the fog node
area_fognode.setyCoordinate(getCoordinatevalue(10));
area_fognode.setParentId(parentId);
return area_fognode;
}
//creating  the smart waste bins
private static FogDevice addBin(String id, int userId,
String appId, int parentId){
FogDevice bin = createFogDevice("b-"+id, 5000, 1000, 10000,
10000, 4, 0, 87.53, 82.44);
bin.setParentId(parentId);
Sensor sensor = new Sensor("s-"+id, "BIN", userId, appId,
new DeterministicDistribution(getCoordinatevalue(5)));
sensors.add(sensor);
Actuator ptz = new Actuator("act-"+id, userId,
appId, "ACT_CONTROL");
actuators.add(ptz);
sensor.setGatewayDeviceId(bin.getId());
sensor.setLatency(1.0);
ptz.setGatewayDeviceId(parentId);
ptz.setLatency(1.0);
//assigning x coordinate value to the smart bin
bin.setxCoordinate(getCoordinatevalue(10));
//assigning y coordinate value to the smart bin
bin.setyCoordinate(getCoordinatevalue(10));
return bin;
}
```

Code Snippet-5 This code snippet is to be placed in **main** class.

```
private static Application createApplication
(String appId, int userId){

Application application =
Application.createApplication(appId, userId);
application.addAppModule("waste-info-module", 10);
application.addAppModule("master-module", 10);
application.addAppModule("recycle-module", 10);
application.addAppModule("health-module", 10);
application.addAppModule("municipal-module", 10);

application.addAppEdge("BIN", "waste-info-module",1000, 2000,
"BIN", Tuple.UP,
AppEdge.SENSOR);
application.addAppEdge("waste-info-module", "master-module",
1000, 2000, "Task1",
Tuple.UP, AppEdge.MODULE);
application.addAppEdge("master-module", "municipal-module",
1000, 2000, "Task2",
Tuple.UP, AppEdge.MODULE);
application.addAppEdge("master-module", "recycle-module",
1000, 2000, "Task3",
Tuple.UP, AppEdge.MODULE);
application.addAppEdge("master-module", "health-module",
1000, 2000, "Task4",
Tuple.UP, AppEdge.MODULE);
application.addAppEdge("master-module", "ACT_CONTROL",
100, 28, 100, "ACT_PARAMS",
Tuple.UP, AppEdge.ACTUATOR);
application.addTupleMapping("waste-info-module",
"BIN", "Task1",
new FractionalSelectivity(1.0));
application.addTupleMapping("master-module", "BIN", "Task2",
new FractionalSelectivity(1.0));
application.addTupleMapping("master-module", "BIN", "Task3",
new FractionalSelectivity(1.0));
application.addTupleMapping("master-module", "BIN", "Task4",
new FractionalSelectivity(1.0));
application.addTupleMapping("master-module", "BIN", "ACT_CONTROL",
new FractionalSelectivity(1.0));

final AppLoop loop1 = new AppLoop(new ArrayList<String>()
{{add("BIN");
add("waste-info-module");add("master-module");
add("municipal-module");
add("recycle-module");add("health-module");
add("ACT_CONTROL");}});
List<AppLoop> loops = new ArrayList<AppLoop>(){{add(loop1);}};

application.setLoops(loops);
return application;
}
```

Code Snippet-6 This code snippet is to be placed in **main** class. In this code snippet, we have created a method will generate the random number.

```
private static double getCoordinatevalue(double min)
{
Random rn=new Random();
return rn.nextDouble()+min;
}
```

Code Snippet-7 This code snippet is to be placed in FogDevice class. This code snippet is taken from [5]. We have declared two variables xCoordinate, and yCoordinate to store the value of x and y coordinate respectively.

```
public double xCoordinate;
//specifying the xCoordinate of the fog device
public double yCoordinate;
//specifying the yCoordinate of the fog device
//method to set the value of xCoordinate
public void setxCoordinate(double xCoordinate)
{
this.xCoordinate=xCoordinate;
}
//method to get the value of xCoordinate
public double getxCoordinate()
{
return xCoordinate;
}
//method to set the value of yCoordinate
public void setyCoordinate(double yCoordinate)
{
this.yCoordinate=yCoordinate;
}
//method to get the value of yCoordinate
public double getyCoordinate()
{
return yCoordinate;
}
```

3.3 Smart Mining Industry System

Mining is one of the most important and prominent industries that requires a lot of data analysis. With every passing day, the enormity of mining industry is increasing day by day. According to the IBM research [14], the requirement of mines increasing day by day and every individual requires approximately 3.11 million pounds of fuel, minerals, and metals in his/her life. Despite its significance, mining industry entails multiple risks. During the mineral and coal mining, chemical reactions, hazardous gas emission, suffocation, and rock sliding are among the probable risks that are hazardous for the lives of mining personnel [16]. Therefore, it is important to employ the IoT devices, such as heterogeneous sensors to pick up the

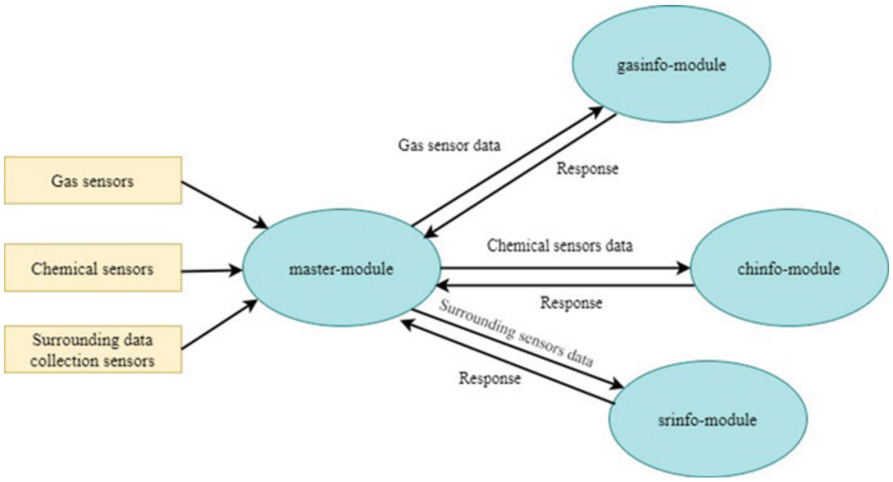

Fig. 6 Fog based smart mining industry application model of iFogSim

gases, chemicals, and the surrounding data and to inform the concerned personnel regarding the undesired and dangerous situations. The surrounding sensors will collect the data before the digging process. This can also reduce cost and save the energy by predicting the probability of finding coal and minerals at certain places before the actual digging process. Gas sensors can be deployed everywhere in the mines that cannot only help in measuring the biological gases value in the mines and tunnels but also control the emission of gases. Moreover, numerous chemical reactions occur in the mines which can be very dangerous for human labors working in the mine. Therefore, collecting and analyzing the surrounding, biological gases and chemical reactions' data is very useful in the mining industry in making predictions about the digging process and hazardous events.

Building Scenario with iFogSim for Smart Mining Industry System To simulate this case study, first make a new class in *org.fog.test.perfeval* package. In the fog based smart mining industry, there would be heterogeneous sensors i.e. surrounding sensors, biological sensors, and chemical sensors that are connected to the fog nodes through a router device. In the fog nodes, we create four modules that include: (1) *master module,* (2) *gasinfo-module,* (3) *chinfo-module,* and (4) *srinfo-module.* The *master-module* will collect the sensors data from all type of sensors. It will categorize the data and send the specific data to the respective modules. For example, the gas sensors data will be sent to the *gasinfo-moudule*. The *gasinfo-module* will process the gas sensors data, analyze the gas values, and it will send the response back to the master-module. The response is basically the action, which will be taken in account of gas sensors values. Figure 6 depicts the data flow of the smart mining industry application model.

Building Simulation with iFogSim for Mining Industry System To simulate this scenario, first make a new class in *org.fog.test.perfeval* package.

Code Snippet-8 This code snippet is to be placed in the **main** class. In this code snippet we created the variables for fog devices and sensors. Three type of sensors variables are created for biological, chemical and surrounding sensors. Moreover, the modules are placed on the fog nodes. This code snippet should be added in the main method after initializing the module mapping.

```
//Create the list of fog devices
static List<FogDevice> fogDevices = new ArrayList<FogDevice>();
//Create the list of sensors
static List<Sensor> sensors = new ArrayList<Sensor>();
//Create the list of actuators
static List<Actuator> actuators = new ArrayList<Actuator>();
//Define the number of fog nodes will be deployed
static int numOfFogDevices = 10;
//Define the number of gas sensors with each fog nodes
static int numOfGasSensorsPerArea=1;
//Define the number of chemical sensors with each fog nodes
static int numOfChSensorsPerArea=1;
//Define the number of surrounding sensors with each fog nodes
static int numOfSrSensorsPerArea=1;
//We are using the fog nodes to perform the operations.
//cloud is set to false
private static boolean CLOUD = false;
public static void main(String[] args) {
Log.printLine("Waste Management system...");
try {
Log.disable();
int num_user = 1; // number of cloud users
Calendar calendar = Calendar.getInstance();
boolean trace_flag = false; // mean trace events
CloudSim.init(num_user, calendar, trace_flag);
String appId = "mins"; // identifier of the application
FogBroker broker = new FogBroker("broker");
Application application =
createApplication(appId, broker.getId());
application.setUserId(broker.getId());
createFogDevices(broker.getId(), appId);
Controller controller = null;
// initializing a module mapping
ModuleMapping moduleMapping =
ModuleMapping.createModuleMapping();
for(FogDevice device : fogDevices){
if(device.getName().startsWith("a")){
moduleMapping.addModuleToDevice("master-module",
device.getName());
if(device.getName().startsWith("g")){
moduleMapping.addModuleToDevice("gasinfo-module",
device.getName());   }
if(device.getName().startsWith("c")){
moduleMapping.addModuleToDevice
("chemicalinfo-module", device.getName());   }
if(device.getName().startsWith("s")){
moduleMapping.addModuleToDevice
("srinfo-module", device.getName());
}}}
// if the mode of deployment is cloud-based
if(CLOUD){
// placing all instances of master-module in Cloud
addModuleToDevice("mastermodule", "cloud");
moduleMapping.addModuleToDevice("gasinfo-module", "cloud");
moduleMapping.addModuleToDevice("chinfo-module", "cloud");
moduleMapping.addModuleToDevice("srinfo-module", "cloud");}

controller = new Controller("master-controller", fogDevices,
sensors, actuators);
controller.submitApplication(application,
(CLOUD)?(new ModulePlacementMapping(fogDevices, application,
moduleMapping))
:(new ModulePlacementEdgewards(fogDevices, sensors, actuators,
application, moduleMapping)));

 TimeKeeper.getInstance().setSimulationStartTime(
   Calendar.getInstance().
getTimeInMillis());

CloudSim.startSimulation();

CloudSim.stopSimulation();

Log.printLine("mining industry simulation finished!");
} catch (Exception e) {
e.printStackTrace();
Log.printLine("Unwanted errors happen");
}
}
```

Code Snippet-9 It is to be placed in the **main** class. In this code snippet we are creating cloud server, proxy server, fog nodes, gas sensors, chemical sensors, and surrounding sensors.

```
private static void createFogDevices(int userId, String appId) {
FogDevice cloud = createFogDevice("cloud", 44800, 40000, 100,
10000, 0, 0.01, 16*103,
16*83.25);
cloud.setParentId(-1);
fogDevices.add(cloud);
FogDevice router = createFogDevice("proxy-server", 7000, 4000,
10000, 10000, 1, 0.0, 107.339, 83.4333);
router.setParentId(cloud.getId());
router.setUplinkLatency(100.0);
fogDevices.add(router);
for(int i=0;i<numOfFogDevices;i++){
addFogNode(i+"", userId, appId, router.getId());
}
```

```
}
private static FogDevice addFogNode(String id,
int userId, String appId, int parentId)
{
FogDevice fognode = createFogDevice("a-"+id, 5000, 4000, 10000,
10000, 3, 0.0, 107.339, 83.4333);
fogDevices.add( fognode);
fognode.setUplinkLatency(1.0);
for(int i=0;i<numOfGasSensorsPerArea;i++){
addGasSensors(i+"", userId, appId, fognode.getId());
}
for(int i=0;i<numOfChSensorsPerArea;i++){
addChSensors(i+"", userId, appId, fognode.getId());
}
for(int i=0;i<numOfSrSensorsPerArea;i++){
addSrSensors(i+"", userId, appId, fognode.getId());
}
return fognode;
}
private static FogDevice addGasSensors(String id, int userId,
String appId, int parentId){
FogDevice gasSensor = createFogDevice("g-"+id, 5000, 1000, 10000,
10000, 4, 0, 87.53, 82.44);
gasSensor.setParentId(parentId);
Sensor sensor = new Sensor("s-"+id, "GAS", userId, appId, new
DeterministicDistribution(5));
sensors.add(sensor);
Actuator ptz = new Actuator("act-"+id, userId,
appId, "ACT_CONTROL");
actuators.add(ptz);
sensor.setGatewayDeviceId(gasSensor.getId());
sensor.setLatency(1.0);
ptz.setGatewayDeviceId(parentId);
ptz.setLatency(1.0);
return gasSensor;
}
private static FogDevice addChSensors(String id, int userId,
String appId, int parentId){
FogDevice chSensor = createFogDevice("c-"+id, 5000, 1000, 10000,
10000, 4, 0, 87.53, 82.44);
chSensor.setParentId(parentId);
Sensor sensor = new Sensor("sch-"+id, "CH", userId, appId,
new DeterministicDistribution(5));
sensors.add(sensor);
Actuator ptzch = new Actuator("actch-"+id, userId,
appId, "ACT_CONTROLCH");
actuators.add(ptzch);
sensor.setGatewayDeviceId(chSensor.getId());
sensor.setLatency(1.0);
ptzch.setGatewayDeviceId(parentId);
ptzch.setLatency(1.0);
return chSensor;
}
private static FogDevice addSrSensors(String id, int userId,
String appId, int parentId){
FogDevice srSensor = createFogDevice("s-"+id, 5000, 1000, 10000,
10000, 4, 0, 87.53, 82.44);
srSensor.setParentId(parentId);
Sensor sensor = new Sensor("ssr-"+id, "SR", userId, appId,
new DeterministicDistribution(5));
sensors.add(sensor);
Actuator ptzch = new Actuator("actsr-"+id, userId,
appId, "ACT_CONTROLSR");
actuators.add(ptzch);
sensor.setGatewayDeviceId(srSensor.getId());
sensor.setLatency(1.0);
ptzch.setGatewayDeviceId(parentId);
ptzch.setLatency(1.0);
return srSensor;
}
```

Code Snippet-10 It is to be placed in the **main** class. In this code snippet we are creating the modules and mapping it to the fog devices.

```
private static Application createApplication(String appId,
int userId){
Application application = Application.createApplication(appId,
userId);
application.addAppModule("gasinfo-module", 10);
application.addAppModule("master-module", 10);
application.addAppModule("chinfo-module", 10);
application.addAppModule("srinfo-module", 10);
application.addAppEdge("GAS", "master-module",1000,
2000, "GAS", Tuple.UP, AppEdge.SENSOR);
application.addAppEdge("CH", "chinfo-module", 1000,
2000, "CH", Tuple.UP, AppEdge.SENSOR);
application.addAppEdge("SR", "srinfo-module", 1000, 2000,
"SR",
Tuple.UP, AppEdge.SENSOR);
application.addAppEdge("master-module", "gasinfo-module",
1000, 2000,
"gasTask", Tuple.UP, AppEdge.MODULE);
application.addAppEdge("master-module", "chinfo-module", 1000,
2000, "chTask", Tuple.UP, AppEdge.MODULE);
application.addAppEdge("master-module", "srinfo-module", 1000,
2000, "srTask", Tuple.UP, AppEdge.MODULE);
//Response
application.addAppEdge("gasinfo-module", "master-module",
1000, 2000, "gasResponse", Tuple.UP, AppEdge.MODULE);
application.addAppEdge("chinfo-module", "master-module",
1000, 2000, "chResponse", Tuple.UP, AppEdge.MODULE);
application.addAppEdge("srinfo-module", "master-module",
1000, 2000,"srResponse", Tuple.UP, AppEdge.MODULE);
application.addTupleMapping("master-module", "GAS", "gasTask",
new FractionalSelectivity(1.0));
application.addTupleMapping("master-module", "CH", "chTask",
new FractionalSelectivity(1.0));
application.addTupleMapping("master-module", "SR", "srTas",
new FractionalSelectivity(1.0));
application.addTupleMapping("gasinfo-module",
"gasTask", "gasResponse",
new FractionalSelectivity(1.0));
application.addTupleMapping("chinfo-module",
"chTask", "chResponse",
new FractionalSelectivity(1.0));
application.addTupleMapping("srinfo-module",
"srTask", "srResponse",
new FractionalSelectivity(1.0));

final AppLoop loop1 = new AppLoop(new ArrayList<String>()
{{add("GAS");
add("master-module");add("gasinfo-module");add("gasTask");
add("gasResponse");}});
final AppLoop loop2 = new AppLoop(new
ArrayList<String>(){{add("CH");
add("master-module");add("chinfo-module");
add("chTask")
;add("chResponse");}});
final AppLoop loop3 = new AppLoop(new
ArrayList<String>(){{add("SR");
add("master-module");add("srinfo-module")
;add("srTask");add("srResponse");}});
List<AppLoop> loops = new
ArrayList<AppLoop>(){{add(loop1);add(loop2)
;add(loop3);}};

application.setLoops(loops);
return application;
}
```

3.4 *Sensing as a Service*

Nowadays Unmanned Aerial Vehicles (UAVs) are widely used to support on-demand IoT services [15]. The sensors and actuators associated with an UAV helps in perceiving the external environments and triggering physical actions respectively where the structured deployment of IoT devices is infeasible and costly. However, UAVs are mobile in nature and most of them are energy constrained and are equipped with limited processing capabilities [17]. Therefore, the data generated by the UAVs requires assistance from Fog or Cloud computing paradigms to be processed. It also demands faster networking support for real-time interactions [18]. Considering these requirements, we model an application case scenario and build a simulation setup to illustrate the UAV-based sensing as a service in integrated Fog-Cloud environments.

Building Scenario with iFogSim for UAV-Based Sensing as a Service Figure 7 depicts the conceptual integration of UAVs with gateways, Fog nodes and Cloud

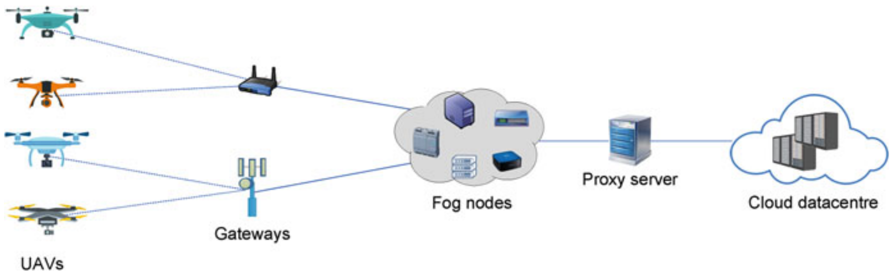

Fig. 7 Prospective computing environments for UAV-based sensing as a service

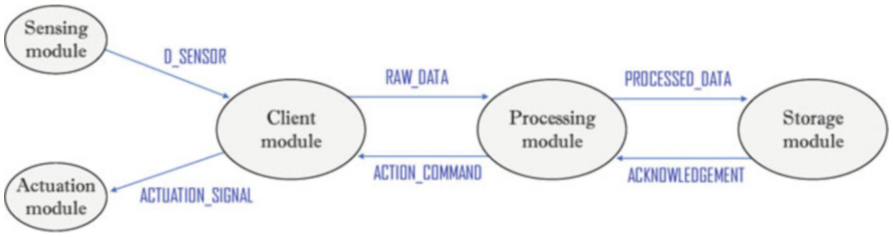

Fig. 8 Application model for UAV-based sensing as a service

data centers. Additionally, the data-driven interactions among these heterogeneous components can be realized through a distributed application as shown in Fig. 8 where the Sensing and Actuation module operate on the UAV-based sensor and actuator, respectively. The Sensing module forwards *D_SENSOR* to the Client module which is more likely to be placed in the UAV. Later, the Client module performs pre-processing of the sensor generated data and dispatches to the Processing module. This module incorporates data analytic that help in transforming the *RAW_DATA* to a meaningful information suitable for evaluation, comparison and making actuation decisions. After performing such operations, the Processing module generates two types of data namely *PROCESSED_DATA* and *ACTION_COMMAND* that are directed to Storage and Client module respectively. The Storage module preserves the outcome of Processing module for location-independent and scalable distribution whereas the Client module digest the outcome for generating the *ACTUATION_SIGNAL* for the Actuation module.

Building Simulation with iFogSim for UAV-Based Sensing as a Service To simulate the prospective UAV-based sensing as a service scenario, a new class in *org.fog.test.perfeval* package is required to be created.

Code Snippet-11 This code snippet helps in creating the computing environments for UAV-based sensing as a service and it should be placed in the main class.

```
private static void createFogDevices(int userId, String appId) {

FogDevice cloud = createFogDevice("cloud", 44800, 40000, 100,
10000,0.01, 16*103, 16*83.25);
cloud.setParentId(-1);
locator.setLevel(cloud, 0);
FogDevice proxy = createFogDevice("proxy-server", 2800, 4000,
10000, 10000, 0.0, 107.339, 83.4333);
proxy.setParentId(cloud.getId());
proxy.setUplinkLatency(100);
locator.setLevel(proxy, 1);

fogDevices.add(cloud);
fogDevices.add(proxy);

for(int i=0;i<numOfGatewayDevices;i++){
FogDevice gateway = addGw("gateway_"+i,
userId, appId, proxy.getId());
gateway.setUplinkLatency(4);
locator.setLevel(gateway, 2);
fogDevices.add(gateway);
}

for(int i=0;i<numOfIoTDrones;i++){
FogDevice drone = addDrone("drone_"+i, userId, appId, -1);
drone.setUplinkLatency(2);
locator.setLevel(drone, 3);
fogDevices.add(drone);
}
}

private static FogDevice addGw(String name, int userId,
String appId, int parentId){
FogDevice gateway = createFogDevice(name, 2800, 4000, 10000,
10000, 0.0, 107.339, 83.4333);
//locator.setInitialLocation(name,gateway.getId());
gateway.setParentId(parentId);
return gateway;
}

private static FogDevice addDrone(String name, int userId,
String appId, int parentId){
FogDevice drone = createFogDevice(name, 500, 20,
1000, 270, 0, 87.53, 82.44);
drone.setParentId(parentId);
//locator.setInitialLocation(name,drone.getId());
Sensor droneSensor = new Sensor("sensor-"+name, "D-SENSOR",
userId, appId,
new DeterministicDistribution(SENSOR_TRANSMISSION_TIME));
sensors.add(droneSensor );
Actuator dronedisplay = new Actuator("actuator-"+name, userId,
appId, "D-DISPLAY");
actuators.add(dronedisplay);
droneSensor.setGatewayDeviceId(drone.getId());
droneSensor.setLatency(6.0);
dronedisplay.setGatewayDeviceId(drone.getId());
dronedisplay.setLatency(1.0);
```

```
return drone;
}
```

Code Snippet-12 This code snippet creates the application model for UAV-based sensing as a service and it is also required to be placed in the **main** class.

```
private static Application createApplication(String appId,
int userId){

Application application =
Application.createApplication(appId, userId);

application.addAppModule("clientModule", 10);
application.addAppModule("processingModule", 10);
application.addAppModule("storageModule", 10);


if(SENSOR_TRANSMISSION_TIME==5.1)
application.addAppEdge("D-SENSOR", "clientModule",
2000, 500, "D-SENSOR", Tuple.UP, AppEdge.SENSOR);
else
application.addAppEdge("D-SENSOR", "clientModule",
3000, 500, "D-SENSOR", Tuple.UP, AppEdge.SENSOR);
application.addAppEdge("clientModule",
"processingModule", 3500, 500, "RAW_DATA",
Tuple.UP, AppEdge.MODULE);
application.addAppEdge("processingModule",
"storageModule", 1000, 1000, "PROCESSED_DATA",
Tuple.UP, AppEdge.MODULE);
application.addAppEdge("processingModule",
"clientModule", 14, 500, "ACTION_COMMAND",
Tuple.DOWN, AppEdge.MODULE);
application.addAppEdge("clientModule", "D-DISPLAY",
1000, 500, "ACTUATION_SIGNAL", Tuple.DOWN,
AppEdge.ACTUATOR);

application.addTupleMapping("clientModule",
"D-SENSOR", "RAW_DATA",
new FractionalSelectivity(1.0));
application.addTupleMapping("processingModule",
"RAW_DATA", "PROCESSED_DATA",
new FractionalSelectivity(1.0));
application.addTupleMapping("processingModule",
"RAW_DATA", "ACTION_COMMAND",
new FractionalSelectivity(1.0));
application.addTupleMapping("clientModule",
"ACTION_COMMAND", "ACTUATION_SIGNAL",
new FractionalSelectivity(1.0));


final AppLoop loop1 = new AppLoop(new ArrayList<String>()
{{add("D-SENSOR");
add("clientModule");add("processingModule");
add("clientModule");
add("D-DISPLAY");}});
List<AppLoop> loops = new ArrayList<AppLoop>(){{add(loop1);}};
application.setLoops(loops);

return application;
}
```

4 Conclusions

In this article, we described the key features of iFogSim along with a step by step installation and simulation guide to help researchers model and simulate difference IoT and fog-based scenarios. To help readers gain a better understanding of the iFogSim toolkit, we modeled four real-time case studies related to smart car parking, smart waste management system, the smart mining industry and UAV-based sensing as a service. Moreover, we provided the corresponding code snippets of every case study. The simulation source codes of the case studies can be accessed from the link: https://sites.google.com/site/assadabbasciit/.

References

1. R. Mahmud, K. Ramamohanarao, and R. Buyya, “Edge Affinity-based Management of Applications in Fog Computing Environments,” in Proceedings - 12th IEEE/ACM International Conference on Utility and Cloud Computing, UCC 2019, 2019, pp. 61–70.
2. S. Yi, Z. Hao, Z. Qin, and Q. Li, “Fog computing: Platform and applications,” in Proceedings - 3rd Workshop on Hot Topics in Web Systems and Technologies, HotWeb 2015, 2016, pp. 73–78.
3. I. Stojmenovic and S. Wen, “The Fog computing paradigm: Scenarios and security issues,” in 2014 Federated Conference on Computer Science and Information Systems, FedCSIS 2014, 2014, pp. 1–8.
4. M. Afrin, M. R. Mahmud, and M. A. Razzaque, “Real time detection of speed breakers and warning system for on-road drivers,” in Proceedings - IEEE International WIE Conference on Electrical and Computer Engineering, WIECON-ECE 2015, 2015, pp. 495–498.
5. R. Mahmud and R. Buyya, “Modeling and Simulation of Fog and Edge Computing Environments Using iFogSim Toolkit,” in Fog and Edge Computing, 2019, pp. 433–465.
6. T. Qayyum, A. W. Malik, M. A. K. Khattak, O. Khalid, and S. U. Khan, “FogNetSim++: A Toolkit for Modeling and Simulation of Distributed Fog Environment,” IEEE Access, vol. 6, pp. 63570–63583, 2018.
7. C. Sonmez, A. Ozgovde, and C. Ersoy, “EdgeCloudSim: An environment for performance evaluation of edge computing systems,” Trans. Emerg. Telecommun. Technol., vol. 29, no. 11, Nov. 2018.
8. H. Gupta and R. Buyya, “iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things , Edge,” no. October 2016, pp. 1275–1296, 2017.
9. K. S. Awaisi et al., “Towards a Fog Enabled Efficient Car Parking Architecture,” IEEE Access, vol. 7, no. 1, pp. 159100–159111, 2019.

10. D. Hoornweg and P. Bhada-Tata, “What a waste: a global review of solid waste management,” 2012.
11. M. Aazam, S. Zeadally, and K. A. Harras, “Deploying Fog Computing in Industrial Internet of Things and Industry 4.0,” IEEE Trans. Ind. Informatics, vol. 14, no. 10, pp. 4674–4682, 2018.
12. M. Aazam, M. St-Hilaire, C. H. Lung, and I. Lambadaris, “Cloud-based smart waste management for smart cities,” in IEEE International Workshop on Computer Aided Modeling and Design of Communication Links and Networks, CAMAD, 2016, pp. 188–193.
13. R. Mahmud, A. N. Toosi, K. Ramamohanarao, and R. Buyya, “Context-aware Placement of Industry 4.0 Applications in Fog Computing Environments,” IEEE Transactions on Industrial Informatics, vol. 16, no. 11, pp. 7004–7013, 2020.
14. “IBM”,https://www.ibm.com/blogs/internet-of-things/mining-industry-benefits/, Accessed on August 14, 2020 .
15. M. Afrin, J. Jin, and A. Rahman, “Energy-delay co-optimization of resource allocation for robotic services in cloudlet infrastructure,” in International Conference on Service-Oriented Computing, 2018, pp. 295–303.
16. M. Afrin, J. Jin, A. Rahman, Y. Tian, and A. Kulkarni, “Multi-objective resource allocation for Edge Cloud based robotic workflow in smart factory,” Future Generation Computer Systems, vol. 97, pp. 119–130, 2019.
17. R. Mahmud, S. N. Srirama, K. Ramamohanarao, and R. Buyya, “Quality of Experience (QoE)-aware placement of applications in Fog computing environments,” Journal of Parallel and Distributed Computing, vol. 132, pp. 190–203, 2019.
18. A. N. Toosi, R. Mahmud, Q. Chi, and R. Buyya, “Management and Orchestration of Network Slices in 5G, Fog, Edge and Clouds,” in Fog and Edge Computing, 2019, pp. 79–102.