Dynamic Replica Management in Fog-enabled IoT using Enhanced Data Mining Technique

MSc Research Project
Cloud Computing

Rahul Sudhakar Dhande
Student ID:18182852

School of Computing
National College of Ireland

Supervisor: Prof. Vikas Sahni

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Rahul Sudhakar Dhande
Student ID:	18182852
Programme:	Programme Name
Year:	2020
Module:	MSc Research Project
Supervisor:	Prof. Vikas Sahni
Submission Due Date:	23/08/2020
Project Title:	Dynamic Replica Management in Fog- enabled IoT using Enhanced Data Mining Technique (Configuration Manual)
Word Count:	2059
Page Count:	14

Signature:	
Date:	15th August 2020

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Dynamic Replica Management in Fog- enabled IoT using Enhanced Data Mining Technique (Configuration Manual)

Rahul Sudhakar Dhande
18182852

1 Introduction

The significant step of this research paper is the configuration manual. It is shown the installation and execution scenario of the synthetic dataset, implementation and simulation as well as the java code (final output) that is replica placement in Fog-IoT environment. Also, it includes a detailed description of experimental tools and setup for the research project.

1.1 Purpose

The purpose the configuration manual is to understand step by step all procedure for installation and execution scenario of the synthetic dataset, implementation and simulation as well as the java code (final output) that is replica placement in Fog-IoT environment. It acts as a manual guide. It aims to indicates the information.Also, it helps to analyze the effectiveness of the outlined replica placement in fog-IoT environment.

2 Prerequisites: System Configurations

To evaluate this experiment, require the following tools and setup.

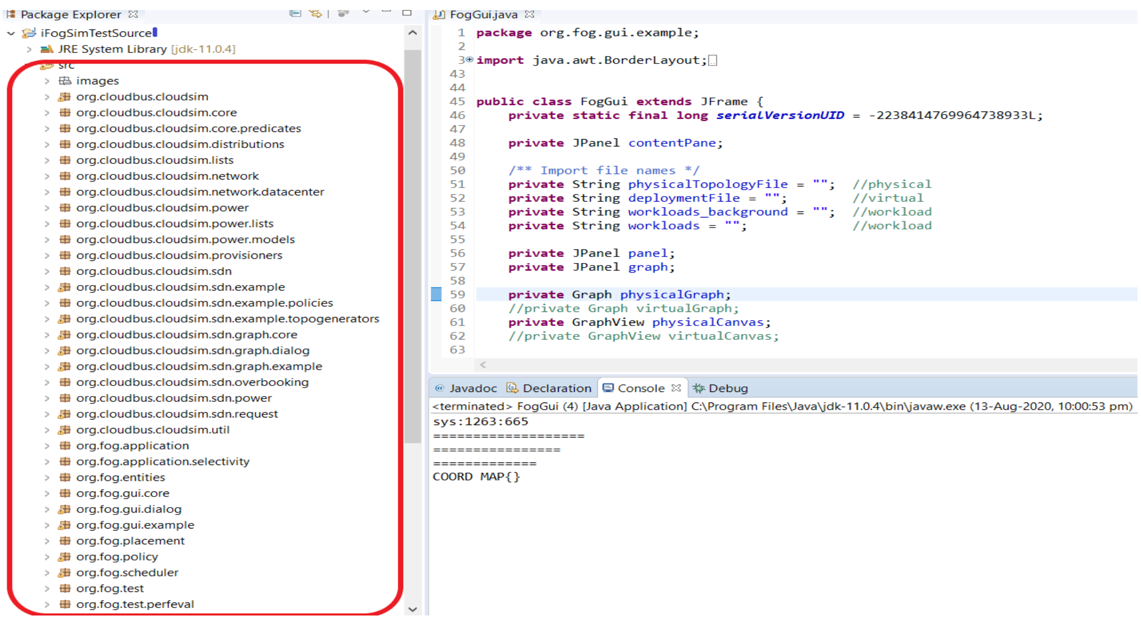
1. Software prerequisites
   - Java development kit (JDK) version 8 [1]
   - Eclipse IDE for Java Developers Version: Oxygen.3a Release (4.7.3a) [2]
   - Benchmarking tool: iFogSim Simulator [3]
   - OS: Windows 10 Home Single Language
2. Hardware prerequisites
   - Processor: Intel (R) Core(TM) i5-8250U CPU 3.4 GHz
   - Specs: 8GB Memory,1 TB HDD and 256GB SSD
   - System Type: 64-bit OS,x-64 based processor

[1]`https://www.oracle.com/in/java/technologies/javase/javase-jdk8-downloads.html`
[2]`https://www.eclipse.org/downloads/`
[3]`https://github.com/Cloudslab/iFogSimTutorials`

2.1 Installation of iFogSim

Once download iFogSim from Cloudslab (**1**) unzip the ifogsim-master into any drive and start Eclipse IDE. Also,followed steps to import ifogsim [4].

1. Click on New to create a new Java Project and give the project name. e.g. ifogsimTestSource.
2. Untick default location and click on browse to select the ifogsim-master from the extracted file drive.
3. Click on finish.


Package Explorer
iFogSimTestSource
JRE System Library [jdk-11.0.4]
src
images
org.cloudbus.cloudsim
org.cloudbus.cloudsim.core
org.cloudbus.cloudsim.core.predicates
org.cloudbus.cloudsim.distributions
org.cloudbus.cloudsim.lists
org.cloudbus.cloudsim.network
org.cloudbus.cloudsim.network.datacenter
org.cloudbus.cloudsim.power
org.cloudbus.cloudsim.power.lists
org.cloudbus.cloudsim.power.models
org.cloudbus.cloudsim.provisioners
org.cloudbus.cloudsim.sdn
org.cloudbus.cloudsim.sdn.example
org.cloudbus.cloudsim.sdn.example.policies
org.cloudbus.cloudsim.sdn.example.topogenerators
org.cloudbus.cloudsim.sdn.graph.core
org.cloudbus.cloudsim.sdn.graph.dialog
org.cloudbus.cloudsim.sdn.graph.example
org.cloudbus.cloudsim.sdn.overbooking
org.cloudbus.cloudsim.sdn.power
org.cloudbus.cloudsim.sdn.request
org.cloudbus.cloudsim.util
org.fog.application
org.fog.application.selectivity
org.fog.entities
org.fog.gui.core
org.fog.gui.dialog
org.fog.gui.example
org.fog.placement
org.fog.policy
org.fog.scheduler
org.fog.test
org.fog.test.perfeval
FogGui.java
Javadoc
Declaration
Console
Debug

```java
package org.fog.gui.example;

import java.awt.BorderLayout;


public class FogGui extends JFrame {
    private static final long serialVersionUID = -2238414769964738933L;

    private JPanel contentPane;

    /** Import file names */
    private String physicalTopologyFile = "";  //physical
    private String deploymentFile = "";        //virtual
    private String workloads_background = "";  //workload
    private String workloads = "";             //workload

    private JPanel panel;
    private JPanel graph;

    private Graph physicalGraph;
    //private Graph virtualGraph;
    private GraphView physicalCanvas;
    //private GraphView virtualCanvas;
```

```
<terminated> FogGui (4) [Java Application] C:\Program Files\Java\jdk-11.0.4\bin\javaw.exe (13-Aug-2020, 10:00:53 pm)
sys:1263:665
===================
================
=============
COORD MAP{}
```

Figure 1: iFogSim in Eclipse IDE

The red highlighted portion figure **1** is shown the ifogSim with packages and classes imported under the Eclipse IDE [5].

Additionally, to build **physical network topology**, used FogGUI.java under the *org.fog.gui.example* package.The GUI developed by Gupta et al. (2016)

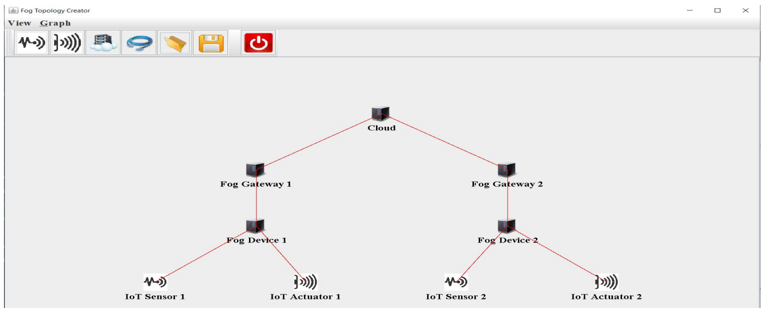


Figure 2: GUI of ifogSim with replica placement network topology

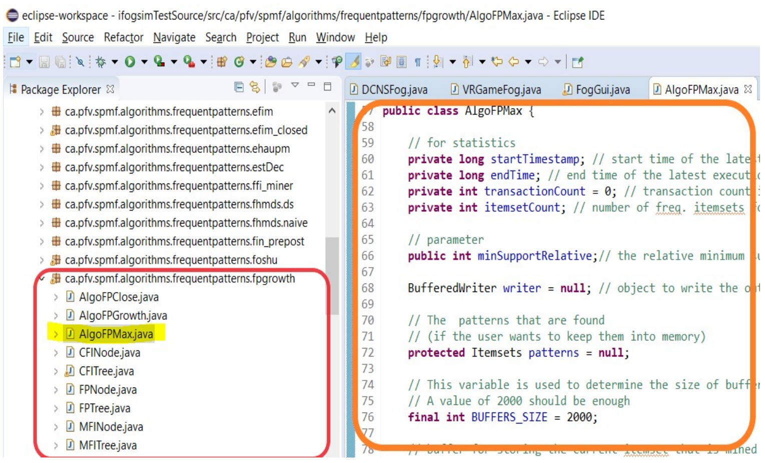
[4] https://youtu.be/uqg7TcOQS5Q

[5] https://www.cloudsimtutorials.online/ifogsim-project-structure-a-beginners-guide/

3 Maximal frequent pattern mining Algorithm

The Hamrouni et al. (2016) introduced data mining techniques for dynamic replication. It includes maximal frequent pattern mining algorithm used for replica placement in data grids environment and simulation using OptorSim. Additionally, the Mansouri et al. (2019) used the same algorithm in a cloud environment and examined results using CloudSim. The algorithm considers data mining techniques such as frequent pattern, correlation measure and frequent correlated pattern. So it is referred to as a data mining algorithm. There are some limitations in the results observed and improved in this approach (EDMDR). Also,evaluated and generated results of latency, response time and network usage by using the same algorithm (MFCPM) in fog enabled IoT environment using iFogSim.

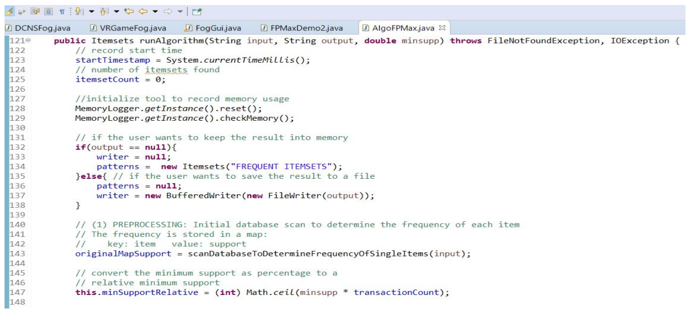
The MFCPM is base the on SPMF(Opensource data mining library). Downloaded SPMF [6] and Unzip into iFogsim src folder for integration to the iFogSim. Right-click on ifogsimTestSource project and Click on Refresh. It is shown SPMF packages and AlgoFPMax java code under the *ca.pfv.spmf.algorithms.frequentpatterns.fpgrowth* package.


eclipse-workspace - ifogsimTestSource/src/ca/pfv/spmf/algorithms/frequentpatterns/fpgrowth/AlgoFPMax.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
DCNSFog.java
VRGameFog.java
FogGui.java
AlgoFPMax.java
ca.pfv.spmf.algorithms.frequentpatterns.efim
ca.pfv.spmf.algorithms.frequentpatterns.efim_closed
ca.pfv.spmf.algorithms.frequentpatterns.ehaupm
ca.pfv.spmf.algorithms.frequentpatterns.estDec
ca.pfv.spmf.algorithms.frequentpatterns.ffi_miner
ca.pfv.spmf.algorithms.frequentpatterns.fhmds.ds
ca.pfv.spmf.algorithms.frequentpatterns.fhmds.naive
ca.pfv.spmf.algorithms.frequentpatterns.fin_prepost
ca.pfv.spmf.algorithms.frequentpatterns.foshu
ca.pfv.spmf.algorithms.frequentpatterns.fpgrowth
AlgoFPClose.java
AlgoFPGrowth.java
AlgoFPMax.java
CFINode.java
CFITree.java
FPNode.java
FPTree.java
MFINode.java
MFITree.java

```
public class AlgoFPMax {

    // for statistics
    private long startTimestamp; // start time of the late
    private long endTime; // end time of the latest execut
    private int transactionCount = 0; // transaction count
    private int itemsetCount; // number of freq. itemsets

    // parameter
    public int minSupportRelative;// the relative minimum

    BufferedWriter writer = null; // object to write the o

    // The  patterns that are found
    // (if the user wants to keep them into memory)
    protected Itemsets patterns = null;

    // This variable is used to determine the size of buff
    // A value of 2000 should be enough
    final int BUFFERS_SIZE = 2000;
```

Figure 3: AlgoFPMax in iFogSim

The red highlighted in the figure 3 indicated the AlgoFPMAx.java under the fpgrowth package, and Orange highlighted shown the actual algorithm in Eclipse IDE GUI. The main function of this algorithm is **runalgorithm()** in figure **4** takes three inputs such as dataset, output and minsupport value.

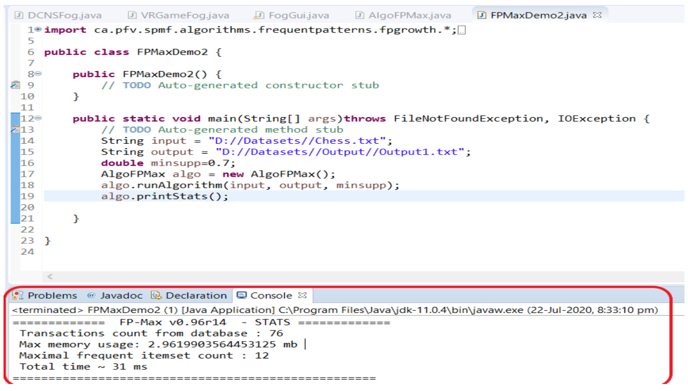


```
    public Itemsets runAlgorithm(String input, String output, double minsupp) throws FileNotFoundException, IOException {
        // record start time
        startTimestamp = System.currentTimeMillis();
        // number of itemsets found
        itemsetCount = 0;

        //initialize tool to record memory usage
        MemoryLogger.getInstance().reset();
        MemoryLogger.getInstance().checkMemory();

        // if the user wants to keep the result into memory
        if(output == null){
            writer = null;
            patterns =  new Itemsets("FREQUENT ITEMSETS");
        }else{ // if the user wants to save the result to a file
            patterns = null;
            writer = new BufferedWriter(new FileWriter(output));
        }

        // (1) PREPROCESSING: Initial database scan to determine the frequency of each item
        // The frequency is stored in a map:
        //    key: item   value: support
        originalMapSupport = scanDatabaseToDetermineFrequencyOfSingleItems(input);

        // convert the minimum support as percentage to a
        // relative minimum support
        this.minSupportRelative = (int) Math.ceil(minsupp * transactionCount);
```

Figure 4: runalgorthm() function of AlgoFPMax

[6]http://www.philippe-fournier-viger.com/spmf/index.php?link=download.php

Additionally,downloaded one sample dataset to test AlgoFPMax algorithm from SPMF sample datasets called Chess.[7]Also, add java class called FPMaxDemo and write an API invocation function to called the runalgorithm() from the package. Moreover , created two individual folders into drive and put this dataset into dataset folder and output folder for outcomes. Finally, give both paths into the program with minsupport value 0.7

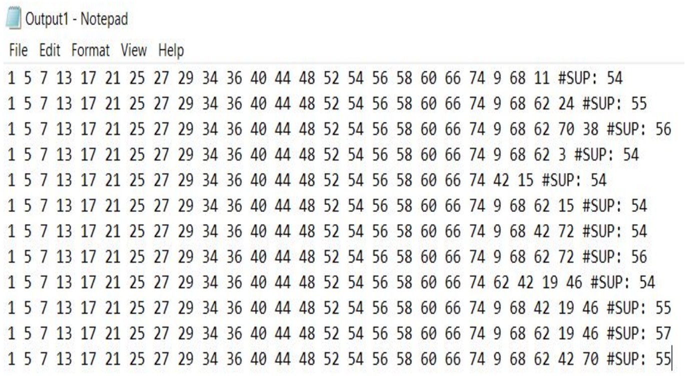


```java
import ca.pfv.spmf.algorithms.frequentpatterns.fpgrowth.*;

public class FPMaxDemo2 {

    public FPMaxDemo2() {
        // TODO Auto-generated constructor stub
    }

    public static void main(String[] args)throws FileNotFoundException, IOException {
        // TODO Auto-generated method stub
        String input = "D://Datasets//Chess.txt";
        String output = "D://Datasets//Output//Output1.txt";
        double minsupp=0.7;
        AlgoFPMax algo = new AlgoFPMax();
        algo.runAlgorithm(input, output, minsupp);
        algo.printStats();

    }

}
```

```
<terminated> FPMaxDemo2 (1) [Java Application] C:\Program Files\Java\jdk-11.0.4\bin\javaw.exe (22-Jul-2020, 8:33:10 pm)
=============  FP-Max v0.96r14  - STATS =============
 Transactions count from database : 76
 Max memory usage: 2.9619903564453125 mb |
 Maximal frequent itemset count : 12
 Total time ~ 31 ms
===================================================
```

Figure 5: Output of FPMaxDemo using Chess dataset(Sample dataset)

The above figure **5** is shown the FPMax Java code with output of sample dataset called Chess. It includes the total transactions, memory usage, maximal frequent itemset count and time.



```
1 5 7 13 17 21 25 27 29 34 36 40 44 48 52 54 56 58 60 66 74 9 68 11 #SUP: 54
1 5 7 13 17 21 25 27 29 34 36 40 44 48 52 54 56 58 60 66 74 9 68 62 24 #SUP: 55
1 5 7 13 17 21 25 27 29 34 36 40 44 48 52 54 56 58 60 66 74 9 68 62 70 38 #SUP: 56
1 5 7 13 17 21 25 27 29 34 36 40 44 48 52 54 56 58 60 66 74 9 68 62 3 #SUP: 54
1 5 7 13 17 21 25 27 29 34 36 40 44 48 52 54 56 58 60 66 74 42 15 #SUP: 54
1 5 7 13 17 21 25 27 29 34 36 40 44 48 52 54 56 58 60 66 74 9 68 62 15 #SUP: 54
1 5 7 13 17 21 25 27 29 34 36 40 44 48 52 54 56 58 60 66 74 9 68 42 72 #SUP: 54
1 5 7 13 17 21 25 27 29 34 36 40 44 48 52 54 56 58 60 66 74 9 68 62 72 #SUP: 56
1 5 7 13 17 21 25 27 29 34 36 40 44 48 52 54 56 58 60 66 74 62 42 19 46 #SUP: 54
1 5 7 13 17 21 25 27 29 34 36 40 44 48 52 54 56 58 60 66 74 9 68 42 19 46 #SUP: 55
1 5 7 13 17 21 25 27 29 34 36 40 44 48 52 54 56 58 60 66 74 9 68 62 19 46 #SUP: 57
1 5 7 13 17 21 25 27 29 34 36 40 44 48 52 54 56 58 60 66 74 9 68 62 42 70 #SUP: 55
```

Figure 6: frequent itemsets from FPMaxDemo with support value

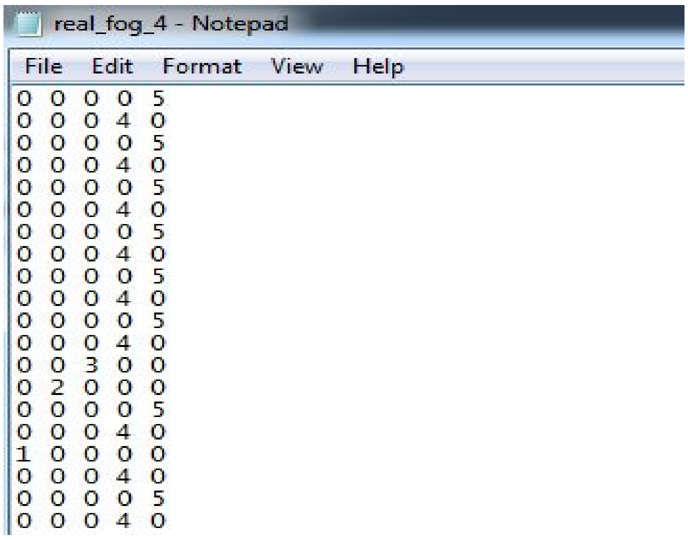
The maximal frequent itemset count referred to as repeated values from chess dataset and support value in the above **6** generated output file.

4 Synthetic dataset

For the evaluation of results, created and used a synthesized dataset that is historical real-time traces (access pattern data) of IoT use case to make replica decisions on fog nodes frequently. It aims to make replica placement. It includes candidate nodes as per physical network topology figure 2. The candidate nodes contain cloud node, fog gateway 1, fog gateway 2, fog device 1, fog device 2. In a dynamic replication algorithm, the candidate nodes referred to as

[7]`http://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php`

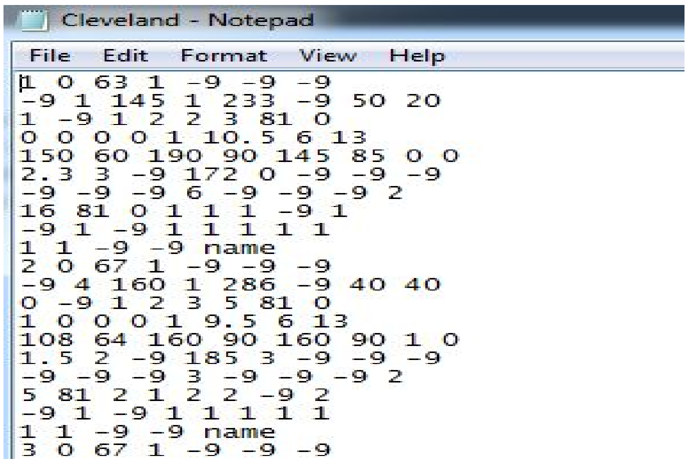
1. Cloud
2. Fog gateway 1
3. Fog gateway 2
4. Fog device 1
5. Fog device 2



```
0 0 0 0 5
0 0 0 4 0
0 0 0 0 5
0 0 0 4 0
0 0 0 0 5
0 0 0 4 0
0 0 0 0 5
0 0 0 4 0
0 0 0 0 5
0 0 0 4 0
0 0 0 0 5
0 0 0 4 0
0 0 3 0 0
0 2 0 0 0
0 0 0 0 5
0 0 0 4 0
1 0 0 0 0
0 0 0 4 0
0 0 0 0 5
0 0 0 4 0
```

Figure 7: Synthesized dataset for EDMDR Simulation

Also, to make replica decisions, used Cleveland health disease training information of healthcare application created by Andras Janosi (M.D.) at the Gottsegen Hungarian Institute of Cardiology, Hungary.[8] The Cleveland dataset contains heart disease patients information, including 7 columns wherein diagnosis column 0 indicates no heart disease and -9 indicates heart disease present. Overall, considered both datasets to make proper replica placement in fog enabled IoT environment.



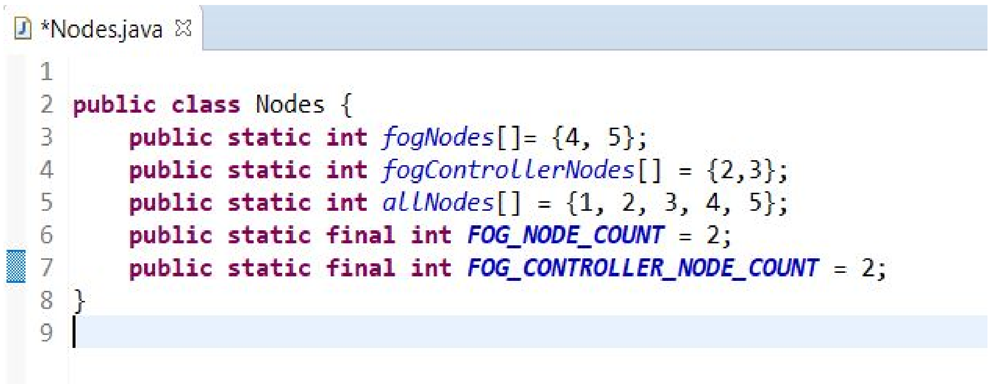
```
1 0 63 1 -9 -9 -9
-9 1 145 1 233 -9 50 20
1 -9 1 2 2 3 81 0
0 0 0 0 1 10.5 6 13
150 60 190 90 145 85 0 0
2.3 3 -9 172 0 -9 -9 -9
-9 -9 -9 6 -9 -9 -9 2
16 81 0 1 1 1 -9 1
-9 1 -9 1 1 1 1 1
1 1 -9 -9 name
2 0 67 1 -9 -9 -9
-9 4 160 1 286 -9 40 40
0 -9 1 2 3 5 81 0
1 0 0 0 1 9.5 6 13
108 64 160 90 160 90 1 0
1.5 2 -9 185 3 -9 -9 -9
-9 -9 -9 3 -9 -9 -9 2
5 81 2 1 2 2 -9 2
-9 1 -9 1 1 1 1 1
1 1 -9 -9 name
3 0 67 1 -9 -9 -9
```

Figure 8: Celeveland information for replica decisions

[8] `https://archive.ics.uci.edu/ml/datasets/heart+Disease`

5 Enhanced data mining dynamic replication algorithm

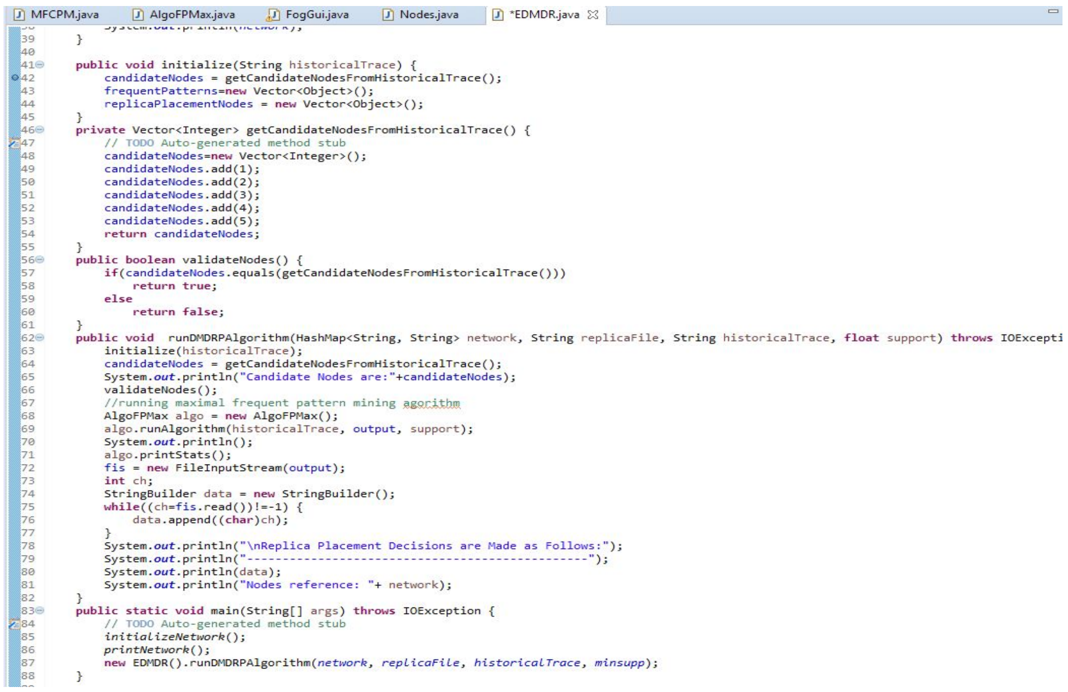
This section includes the implementation of enhancing data mining dynamic replication algorithm (EDMDR) aims to make replica placement based on replica decisions(frequency threshold). The AlgoFPMax algorithm in section 3 invoked using Java API and used SPMF libraries to make proper replica placement.The primary function of this algorithm is runDMDRalgorithm() takes four inputs that include hash network, replica frequency threshold, historical trace and support value, as shown in the figure 10. The replica frequency contains Cleveland training information and historical trace contain synthetic dataset.Also, we provided minsupport value 0.4



```java

public class Nodes {
    public static int fogNodes[]= {4, 5};
    public static int fogControllerNodes[] = {2,3};
    public static int allNodes[] = {1, 2, 3, 4, 5};
    public static final int FOG_NODE_COUNT = 2;
    public static final int FOG_CONTROLLER_NODE_COUNT = 2;
}

```

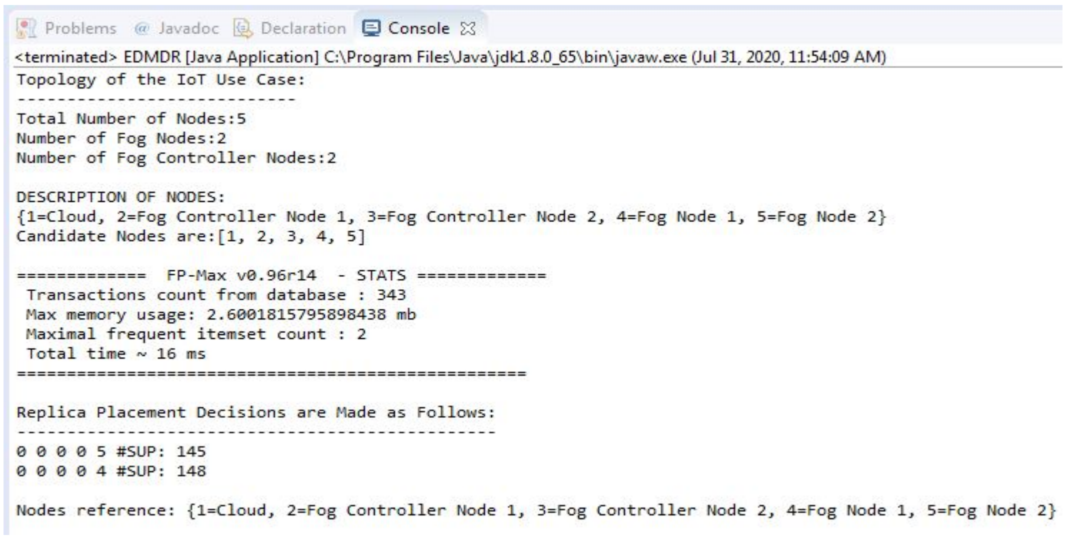
Figure 9: Fog Network: Nodes.Java

For physical network topology, considered another class **Node.java** and integrated into EDMDR algorithm. It calculates total number nodes length, no of fog nodes and no of fog controller nodes. Also, It indicated a description of all fog network used for dynamic replication.



```java
    }

    public void initialize(String historicalTrace) {
        candidateNodes = getCandidateNodesFromHistoricalTrace();
        frequentPatterns=new Vector<Object>();
        replicaPlacementNodes = new Vector<Object>();
    }
    private Vector<Integer> getCandidateNodesFromHistoricalTrace() {
        // TODO Auto-generated method stub
        candidateNodes=new Vector<Integer>();
        candidateNodes.add(1);
        candidateNodes.add(2);
        candidateNodes.add(3);
        candidateNodes.add(4);
        candidateNodes.add(5);
        return candidateNodes;
    }
    public boolean validateNodes() {
        if(candidateNodes.equals(getCandidateNodesFromHistoricalTrace()))
            return true;
        else
            return false;
    }
    public void  runDMDRPAlgorithm(HashMap<String, String> network, String replicaFile, String historicalTrace, float support) throws IOExcepti
        initialize(historicalTrace);
        candidateNodes = getCandidateNodesFromHistoricalTrace();
        System.out.println("Candidate Nodes are:"+candidateNodes);
        validateNodes();
        //running maximal frequent pattern mining agorithm
        AlgoFPMax algo = new AlgoFPMax();
        algo.runAlgorithm(historicalTrace, output, support);
        System.out.println();
        algo.printStats();
        fis = new FileInputStream(output);
        int ch;
        StringBuilder data = new StringBuilder();
        while((ch=fis.read())!=-1) {
            data.append((char)ch);
        }
        System.out.println("\nReplica Placement Decisions are Made as Follows:");
        System.out.println("-----------------------------------------------");
        System.out.println(data);
        System.out.println("Nodes reference: "+ network);
    }
    public static void main(String[] args) throws IOException {
        // TODO Auto-generated method stub
        initializeNetwork();
        printNetwork();
        new EDMDR().runDMDRPAlgorithm(network, replicaFile, historicalTrace, minsupp);
    }
```

Figure 10: Implementation of EDMDR algorithm



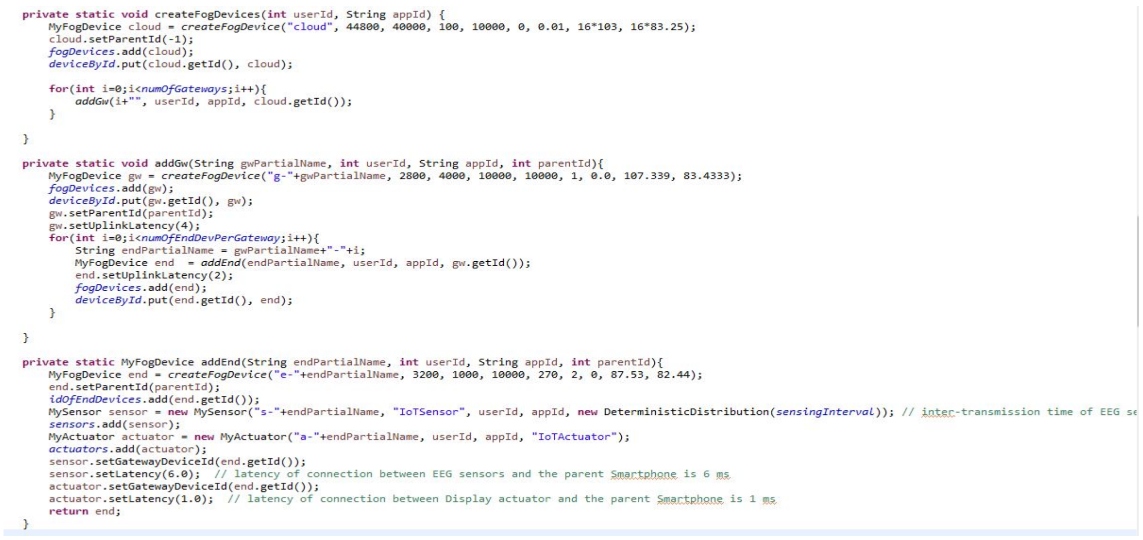
```
Problems  @ Javadoc  Declaration  Console
<terminated> EDMDR [Java Application] C:\Program Files\Java\jdk1.8.0_65\bin\javaw.exe (Jul 31, 2020, 11:54:09 AM)
Topology of the IoT Use Case:
----------------------------
Total Number of Nodes:5
Number of Fog Nodes:2
Number of Fog Controller Nodes:2

DESCRIPTION OF NODES:
{1=Cloud, 2=Fog Controller Node 1, 3=Fog Controller Node 2, 4=Fog Node 1, 5=Fog Node 2}
Candidate Nodes are:[1, 2, 3, 4, 5]

============  FP-Max v0.96r14  - STATS ============
 Transactions count from database : 343
 Max memory usage: 2.6001815795898438 mb
 Maximal frequent itemset count : 2
 Total time ~ 16 ms
===================================================

Replica Placement Decisions are Made as Follows:
------------------------------------------------
0 0 0 0 5 #SUP: 145
0 0 0 0 4 #SUP: 148

Nodes reference: {1=Cloud, 2=Fog Controller Node 1, 3=Fog Controller Node 2, 4=Fog Node 1, 5=Fog Node 2}
```

Figure 11: Dynamic replica placement of fog nodes

The final output of this algorithm in figure 11 is shown the replica placement as per candidate nodes replica decisions and support value. The replica placement contains the removal of duplicates values. In that, five indicates fog node 2 with support value 145 and 4 contains fog node 1 with support value 148. The support value generates as per repeated values in the historical trace. Also, the description of IoT use case (Historical trace), candidates nodes, transaction count, memory usage, total frequent itemset and total time. To recapitulate, if keeping a replica in the cloud, the latency higher than the other nodes as well as in fog gateway latency more and performance is less. As compared to, in fog device latency is less, and performance is high.

6 Simulation of EDMDR

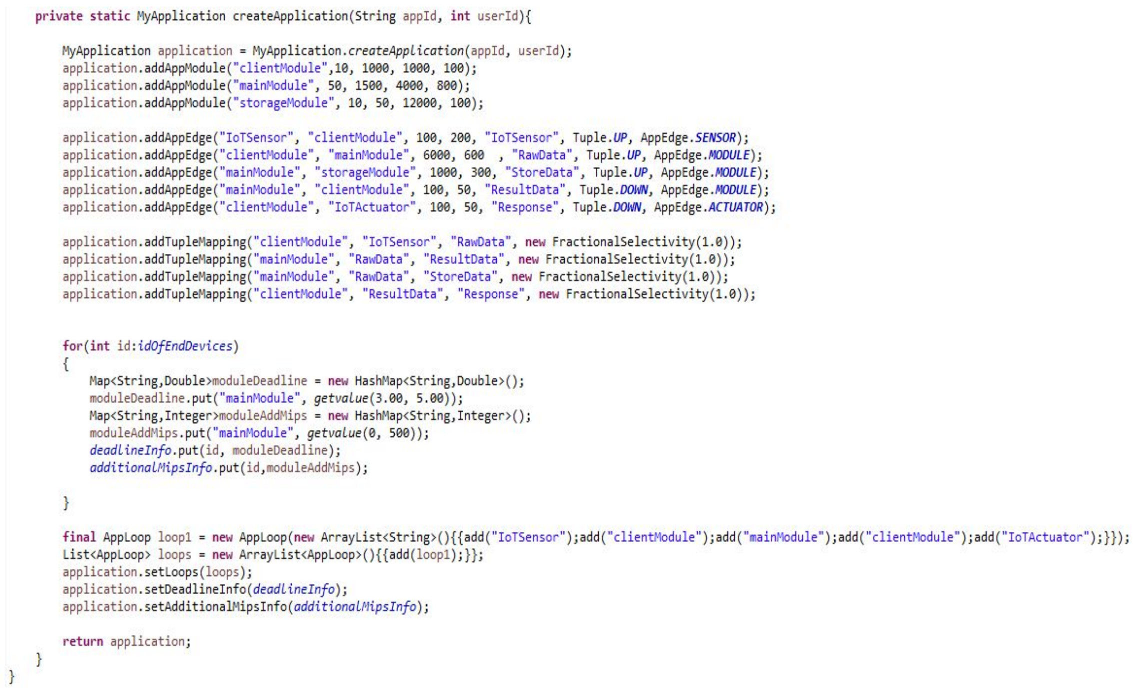
6.1 Structure of fog network nodes environment



```java
private static void createFogDevices(int userId, String appId) {
    MyFogDevice cloud = createFogDevice("cloud", 44800, 40000, 100, 10000, 0, 0.01, 16*103, 16*83.25);
    cloud.setParentId(-1);
    fogDevices.add(cloud);
    deviceById.put(cloud.getId(), cloud);

    for(int i=0;i<numOfGateways;i++){
        addGw(i+"", userId, appId, cloud.getId());
    }

}

private static void addGw(String gwPartialName, int userId, String appId, int parentId){
    MyFogDevice gw = createFogDevice("g-"+gwPartialName, 2800, 4000, 10000, 10000, 1, 0.0, 107.339, 83.4333);
    fogDevices.add(gw);
    deviceById.put(gw.getId(), gw);
    gw.setParentId(parentId);
    gw.setUplinkLatency(4);
    for(int i=0;i<numOfEndDevPerGateway;i++){
        String endPartialName = gwPartialName+"-"+i;
        MyFogDevice end  = addEnd(endPartialName, userId, appId, gw.getId());
        end.setUplinkLatency(2);
        fogDevices.add(end);
        deviceById.put(end.getId(), end);
    }

}

private static MyFogDevice addEnd(String endPartialName, int userId, String appId, int parentId){
    MyFogDevice end = createFogDevice("e-"+endPartialName, 3200, 1000, 10000, 270, 2, 0, 87.53, 82.44);
    end.setParentId(parentId);
    idOfEndDevices.add(end.getId());
    MySensor sensor = new MySensor("s-"+endPartialName, "IoTSensor", userId, appId, new DeterministicDistribution(sensingInterval)); // inter-transmission time of EEG se
    sensors.add(sensor);
    MyActuator actuator = new MyActuator("a-"+endPartialName, userId, appId, "IoTActuator");
    actuators.add(actuator);
    sensor.setGatewayDeviceId(end.getId());
    sensor.setLatency(6.0);  // latency of connection between EEG sensors and the parent Smartphone is 6 ms
    actuator.setGatewayDeviceId(end.getId());
    actuator.setLatency(1.0);  // latency of connection between Display actuator and the parent Smartphone is 1 ms
    return end;
}
```

Figure 12: Network typology nodes configuration in TestApplication Class

To work in this replica placement strategy in Fog enabled IoT environment, implemented three-tier hierarchical order physical network topology in the final TestApplication (2) and iFogSim GUI as per section 2.1. Mahmud and Buyya (2018) As shown in the figure 12, we created fog devices (fog nodes) that are connected to IoT actuators and sensors as well as fog gateway controllers are connected to a cloud data centre with different links for the execution of replica placement.



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

        MyApplication application = MyApplication.createApplication(appId, userId);
        application.addAppModule("clientModule",10, 1000, 1000, 100);
        application.addAppModule("mainModule", 50, 1500, 4000, 800);
        application.addAppModule("storageModule", 10, 50, 12000, 100);

        application.addAppEdge("IoTSensor", "clientModule", 100, 200, "IoTSensor", Tuple.UP, AppEdge.SENSOR);
        application.addAppEdge("clientModule", "mainModule", 6000, 600  , "RawData", Tuple.UP, AppEdge.MODULE);
        application.addAppEdge("mainModule", "storageModule", 1000, 300, "StoreData", Tuple.UP, AppEdge.MODULE);
        application.addAppEdge("mainModule", "clientModule", 100, 50, "ResultData", Tuple.DOWN, AppEdge.MODULE);
        application.addAppEdge("clientModule", "IoTActuator", 100, 50, "Response", Tuple.DOWN, AppEdge.ACTUATOR);

        application.addTupleMapping("clientModule", "IoTSensor", "RawData", new FractionalSelectivity(1.0));
        application.addTupleMapping("mainModule", "RawData", "ResultData", new FractionalSelectivity(1.0));
        application.addTupleMapping("mainModule", "RawData", "StoreData", new FractionalSelectivity(1.0));
        application.addTupleMapping("clientModule", "ResultData", "Response", new FractionalSelectivity(1.0));


        for(int id:idOfEndDevices)
        {
            Map<String,Double>moduleDeadline = new HashMap<String,Double>();
            moduleDeadline.put("mainModule", getvalue(3.00, 5.00));
            Map<String,Integer>moduleAddMips = new HashMap<String,Integer>();
            moduleAddMips.put("mainModule", getvalue(0, 500));
            deadlineInfo.put(id, moduleDeadline);
            additionalMipsInfo.put(id,moduleAddMips);

        }

        final AppLoop loop1 = new AppLoop(new ArrayList<String>(){{add("IoTSensor");add("clientModule");add("mainModule");add("clientModule");add("IoTActuator");}});
        List<AppLoop> loops = new ArrayList<AppLoop>(){{add(loop1);}};
        application.setLoops(loops);
        application.setDeadlineInfo(deadlineInfo);
        application.setAdditionalMipsInfo(additionalMipsInfo);

        return application;
    }
}
```

Figure 13: Implementation of IoT application model for EDMDR

Additionally, to simulate different scenarios of EDMDR implemented an IoT application model as shown in figure 13. It includes various logical components such as IoT sensors, actuators, ClientModule, MainModule placed in fog devices and StorageModule placed in the cloud data centre to reduce end to end latency. Also, used and invoked AppEdge, AppModule and AppLoop classes from org.fog.application in TestApplication.The IoT sensors and actuators include rawdata,resultdata and Storedata for data transmission between logical components. Also, used tuple class for data stream communication between various fog entities as per physical network topology. Mahmud and Buyya (2018) and Gupta et al. (2016)

For the simulation and evaluation of EDMDR in iFogSim, considered different classes from existing packages significantly. Moreover, add enhancements in some java classes like MyApplication, MyActuator, MySensor, and MyFogDevice and MyPlacement is somewhat similar to Application, Actuator, Sensor, FogDevice and ModulePlacement. For the implementation of application placement logic in iFogSim, the two major classes are considered that is ModulePlacement and Controller.Gupta et al. (2016) In EDMDR simulation, applied replica placement logic in MyModulePlacement and MyController.

Subsequently, added EDMDR and Nodes classes in the final testing package (org.fog.test.perfeval) for integration with iFogSim. Finally, the major for testing EDMDR approach create an TestApplication where all the above classes are added through API invocation explicitly. Overall, implemented a physical network topology of EDMDR in FogGUI and TestApplication class.

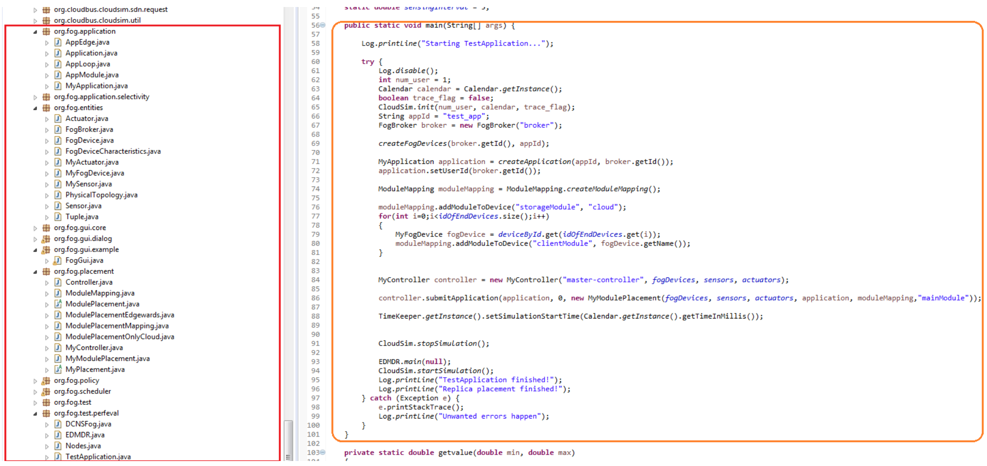
iFogSim Packages	Java Classes
org.fog.application	MyApplication
org.fog.entities	MyActuator,MySensor and MyFogDevice
org.fog.placement	MyController,MyModulePlacement,MyPlacement
org.fog.test.perfeval	TestApplication,EDMDR,Nodes
org.fog.gui.example	FogGUI

Table 1: Structure of classes in iFogSim Simulation

Class	Description
TestApplication	Test dynamic replica placement (EDMDR).
MySensor	It encapsulated the functionality of a physical sensor
MyPlacement	It is used for placement of application.
MyModulePlacement	It is used to have placement of modules
MyFogDevice	It holded capabilities of storage and processing.
MyController	It coordinated the functions of the simulation.
MyApplication	Encapsulated the IoT application is deployed in a fog environment.
MyActuator	Actuator functionality.
AlgoFPMax	It is used for generation of Maximal Frequent Itemsets count.
EDMDR	Replica placement decisions .
Nodes	Physical network topology components.

Table 2: Description of classes used in iFogSim Simulation

The detailed description of classes used for simulation of EDMDR indicated in the table 2. These all classes are invoked in TestApplication used for the final simulation.Also, for the evaluation of various experiments, specifically used some existing classes of iFogSim in the table 3.



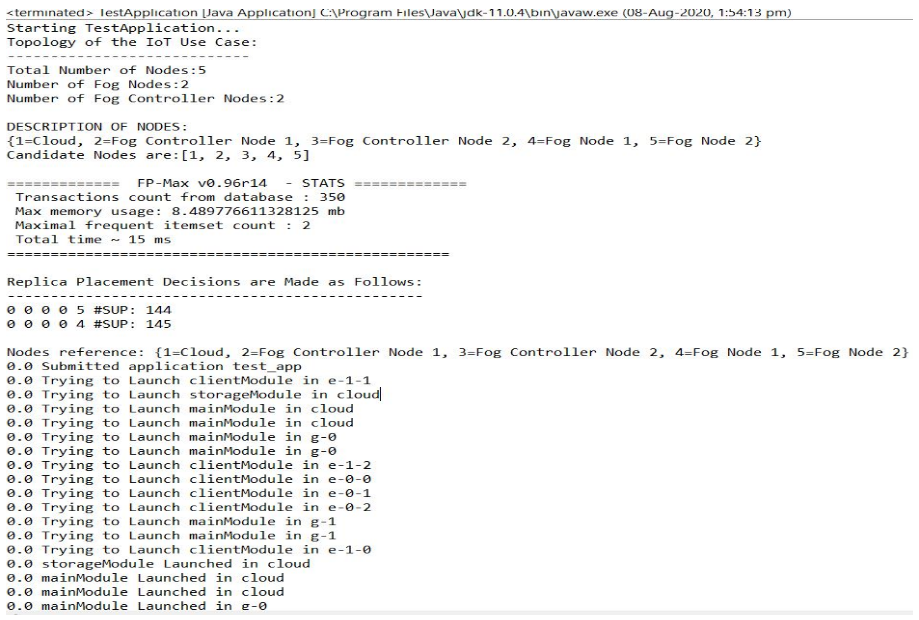
```
org.cloudbus.cloudsim.sdn.request
org.cloudbus.cloudsim.util
org.fog.application
    AppEdge.java
    Application.java
    AppLoop.java
    AppModule.java
    MyApplication.java
org.fog.application.selectivity
org.fog.entities
    Actuator.java
    FogBroker.java
    FogDevice.java
    FogDeviceCharacteristics.java
    MyActuator.java
    MyFogDevice.java
    MySensor.java
    PhysicalTopology.java
    Sensor.java
    Tuple.java
org.fog.gui.core
org.fog.gui.dialog
org.fog.gui.example
    FogGui.java
org.fog.placement
    Controller.java
    ModuleMapping.java
    ModulePlacement.java
    ModulePlacementEdgewards.java
    ModulePlacementMapping.java
    ModulePlacementOnlyCloud.java
    MyController.java
    MyModulePlacement.java
    MyPlacement.java
org.fog.policy
org.fog.scheduler
org.fog.test
org.fog.test.perfeval
    DCNSFog.java
    EDMDR.java
    Nodes.java
    TestApplication.java
```

```java
public static void main(String[] args) {

    Log.printLine("Starting TestApplication...");

    try {
        Log.disable();
        int num_user = 1;
        Calendar calendar = Calendar.getInstance();
        boolean trace_flag = false;
        CloudSim.init(num_user, calendar, trace_flag);
        String appId = "test_app";
        FogBroker broker = new FogBroker("broker");

        createFogDevices(broker.getId(), appId);

        MyApplication application = createApplication(appId, broker.getId());
        application.setUserId(broker.getId());

        ModuleMapping moduleMapping = ModuleMapping.createModuleMapping();

        moduleMapping.addModuleToDevice("storageModule", "cloud");
        for(int i=0;i<idOfEndDevices.size();i++)
        {
            MyFogDevice fogDevice = deviceById.get(idOfEndDevices.get(i));
            moduleMapping.addModuleToDevice("clientModule", fogDevice.getName());
        }


        MyController controller = new MyController("master-controller", fogDevices, sensors, actuators);

        controller.submitApplication(application, 0, new MyModulePlacement(fogDevices, sensors, actuators, application, moduleMapping,"mainModule"));

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


        CloudSim.stopSimulation();

        EDMDR.main(null);
        CloudSim.startSimulation();
        Log.printLine("TestApplication finished!");
        Log.printLine("Replica placement finished!");
    } catch (Exception e) {
        e.printStackTrace();
        Log.printLine("Unwanted errors happen");
    }
}

private static double getvalue(double min, double max)
```

Figure 14: Simulation of EDMDR(TestApplication.java)

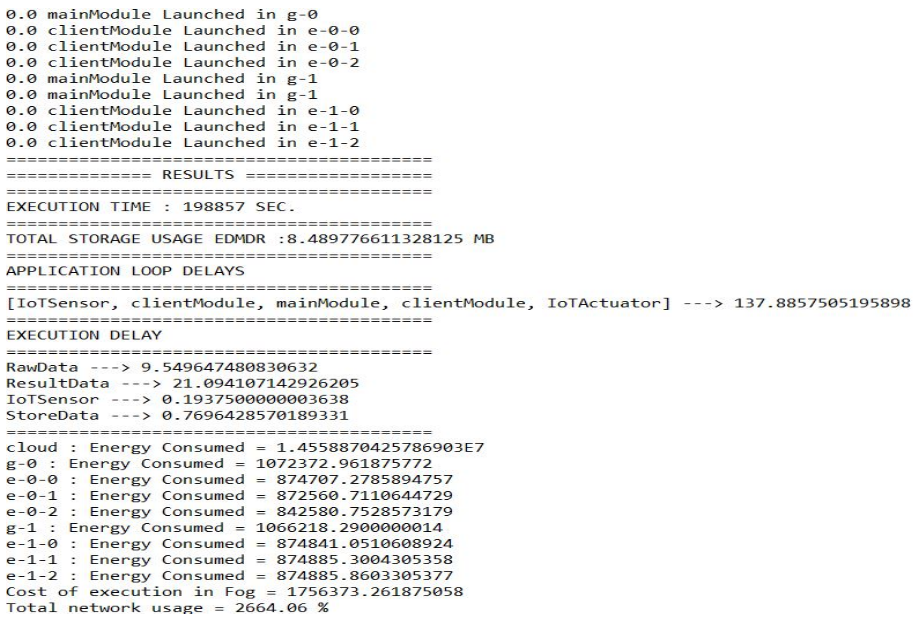
As shown in the figure 14, left side highlighted packages used in EDMDR for simulation. Subsequently, right side, the main simulation function used for the execution of EDMDR approach. Moreover,used historical trace, for replication frequency threshold Cleveland file (for replica decisions) and min.support value 0.4 as an input.In the main function, invoked EDMDR algorithm for replica placement.

The final replica placement decisions with support value using candidate nodes (real-time historical traces) in different text files as per physical network topology 2 are shown in the figure 15 . Also, it shown total transactions count, frequent itemset count, memory usage and total time(ms) with a description of candidate nodes for IoT.



```
<terminated> TestApplication [Java Application] C:\Program Files\Java\jdk-11.0.4\bin\javaw.exe (08-Aug-2020, 1:54:13 pm)
Starting TestApplication...
Topology of the IoT Use Case:
----------------------------
Total Number of Nodes:5
Number of Fog Nodes:2
Number of Fog Controller Nodes:2

DESCRIPTION OF NODES:
{1=Cloud, 2=Fog Controller Node 1, 3=Fog Controller Node 2, 4=Fog Node 1, 5=Fog Node 2}
Candidate Nodes are:[1, 2, 3, 4, 5]

============  FP-Max v0.96r14  - STATS ============
 Transactions count from database : 350
 Max memory usage: 8.489776611328125 mb
 Maximal frequent itemset count : 2
 Total time ~ 15 ms
===================================================

Replica Placement Decisions are Made as Follows:
------------------------------------------------
0 0 0 0 5 #SUP: 144
0 0 0 0 4 #SUP: 145

Nodes reference: {1=Cloud, 2=Fog Controller Node 1, 3=Fog Controller Node 2, 4=Fog Node 1, 5=Fog Node 2}
0.0 Submitted application test_app
0.0 Trying to Launch clientModule in e-1-1
0.0 Trying to Launch storageModule in cloud
0.0 Trying to Launch mainModule in cloud
0.0 Trying to Launch mainModule in cloud
0.0 Trying to Launch mainModule in g-0
0.0 Trying to Launch mainModule in g-0
0.0 Trying to Launch clientModule in e-1-2
0.0 Trying to Launch clientModule in e-0-0
0.0 Trying to Launch clientModule in e-0-1
0.0 Trying to Launch clientModule in e-0-2
0.0 Trying to Launch mainModule in g-1
0.0 Trying to Launch mainModule in g-1
0.0 Trying to Launch clientModule in e-1-0
0.0 storageModule Launched in cloud
0.0 mainModule Launched in cloud
0.0 mainModule Launched in cloud
0.0 mainModule Launched in g-0
```

Figure 15: Final EDMDR Simulation Result 1



```
0.0 mainModule Launched in g-0
0.0 clientModule Launched in e-0-0
0.0 clientModule Launched in e-0-1
0.0 clientModule Launched in e-0-2
0.0 mainModule Launched in g-1
0.0 mainModule Launched in g-1
0.0 clientModule Launched in e-1-0
0.0 clientModule Launched in e-1-1
0.0 clientModule Launched in e-1-2
=========================================
============== RESULTS ==================
=========================================
EXECUTION TIME : 198857 SEC.
=========================================
TOTAL STORAGE USAGE EDMDR :8.489776611328125 MB
=========================================
APPLICATION LOOP DELAYS
=========================================
[IoTSensor, clientModule, mainModule, clientModule, IoTActuator] ---> 137.8857505195898
=========================================
EXECUTION DELAY
=========================================
RawData ---> 9.549647480830632
ResultData ---> 21.094107142926205
IoTSensor ---> 0.1937500000003638
StoreData ---> 0.7696428570189331
=========================================
cloud : Energy Consumed = 1.4558870425786903E7
g-0 : Energy Consumed = 1072372.961875772
e-0-0 : Energy Consumed = 874707.2785894757
e-0-1 : Energy Consumed = 872560.7110644729
e-0-2 : Energy Consumed = 842580.7528573179
g-1 : Energy Consumed = 1066218.2900000014
e-1-0 : Energy Consumed = 874841.0510608924
e-1-1 : Energy Consumed = 874885.3004305358
e-1-2 : Energy Consumed = 874885.8603305377
Cost of execution in Fog = 1756373.261875058
Total network usage = 2664.06 %
```

Figure 16: Final EDMDR Simulation Result 1.1

The figure 16 is shown the evaluation metrics results include execution time, application loop delays, tuple execution delay, cloud energy consumption, cost of execution in fog and total network usage. The results are generated based on different java classes directly invoked in TestApplication.java. Also, the source code for the above metrics is shown in experiments section 7.

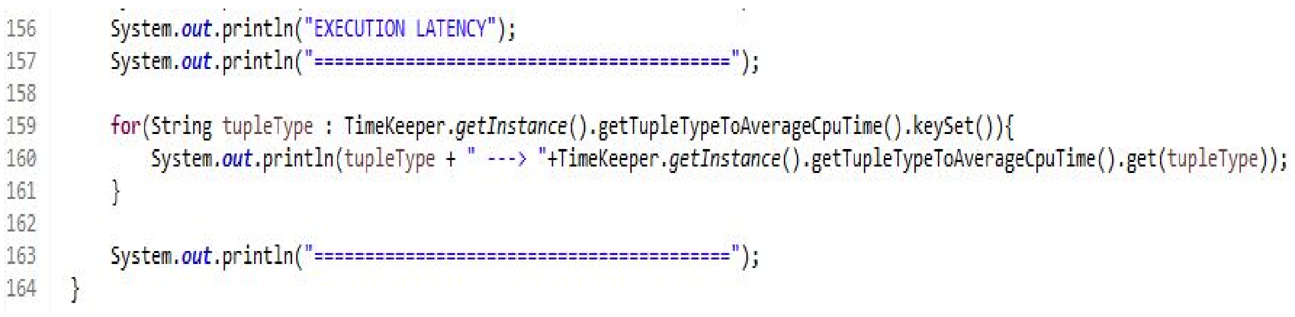
7 Experiments

This section contains implemented source codes for calculation of various experiments of EDMDR in fog enabled IoT.

Class	Description
TimeKeeper	Calculation of total execution delay
Calender	Calculation of execution time
NetworkUsageMonitor	Calculation of total network usage.
PowerDatacenter	Calculation of Energy Consumption.
MemeoryLogger	Calculation of Storage usage.

Table 3: Description of existing classes used in experiments

7.1 Exp1: Execution delay

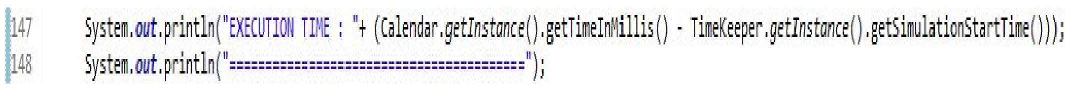


```
    System.out.println("EXECUTION LATENCY");
    System.out.println("=========================================");

    for(String tupleType : TimeKeeper.getInstance().getTupleTypeToAverageCpuTime().keySet()){
        System.out.println(tupleType + " ---> "+TimeKeeper.getInstance().getTupleTypeToAverageCpuTime().get(tupleType));
    }

    System.out.println("=========================================");
}
```

Figure 17: Implementation of Overall latency of EDMDR(MyController.java)

To calculate overall latency of EDMDR in fog enabled IoT, primarily used TimeKeeper class from org.fog.utils and MyController from org.fog.placement invoked in TestApplication.java The TimeKepper class already available in iFogSim. For this experiment,implemented logic in Mycontroller class. As show in the figure 17, the getInstance() function invoked from TimeKeeper class for average CPU time calculation.

7.2 Exp2: Response Time

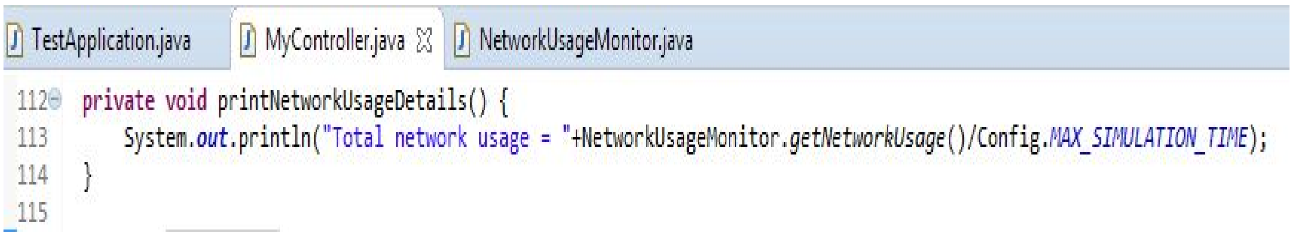


```
System.out.println("EXECUTION TIME : "+ (Calendar.getInstance().getTimeInMillis() - TimeKeeper.getInstance().getSimulationStartTime()));
System.out.println("=========================================");
```

Figure 18: Implementation of Response time of EDMDR(MyController.java)

To calculate the average response time of EDMDR in fog environment, used Calendar class from java.util and Timer class from org.fog.utils and MyController from org.fog.placement invoked in TestApplication.java. The Calendar and Timer class are already present in iFogSim. For this experiment, implemented logic in Mycontroller class. As shown in the figure 18, the getTimeInMillis() function from invoked from the calendar for timezone and getSimulationStartTime() function invoked from TimeKeeper class to return simulation time.

7.3 Exp3: Network Usage

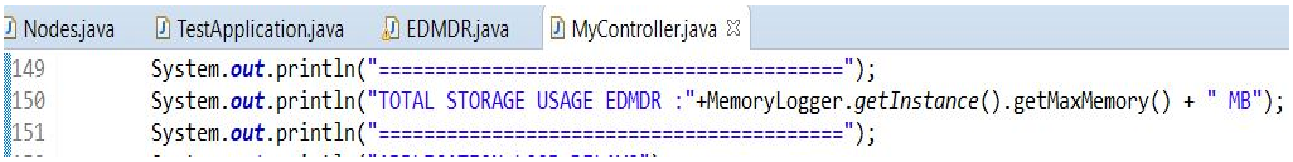

TestApplication.java | MyController.java | NetworkUsageMonitor.java

```
private void printNetworkUsageDetails() {
    System.out.println("Total network usage = "+NetworkUsageMonitor.getNetworkUsage()/Config.MAX_SIMULATION_TIME);
}
```

Figure 19: Implementation of Network Usage of EDMDR(MyController.java)

To evaluate the effective network usage of EDMDR in fog environment, used NetworkUsageMonitor class and Config class from org.fog.utils and MyController from org.fog.placement invoked in TestApplication.java. The NetworkUsageMonitor class and Config class are already present in iFogSim. For this experiment, implemented logic in Mycontroller class. As shown in the figure 19, the sendingTuple() function invoked from the NetworkUsageMonitor for network usage and set MAXSIMULATIONTIME=1000 invoked from Config class.

7.4 Exp4: Storage Usage

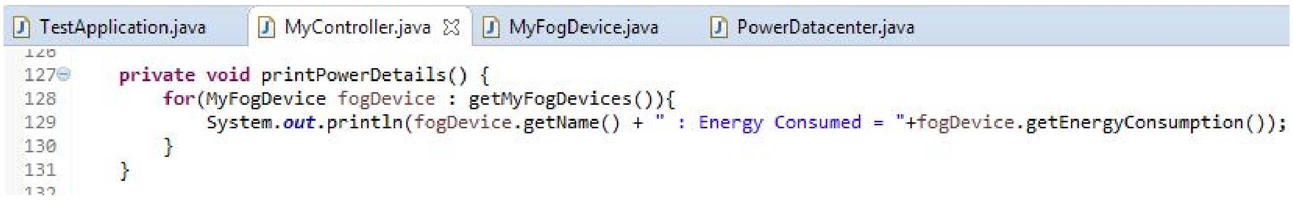

Nodes.java | TestApplication.java | EDMDR.java | MyController.java

```
        System.out.println("========================================");
        System.out.println("TOTAL STORAGE USAGE EDMDR :"+MemoryLogger.getInstance().getMaxMemory() + " MB");
        System.out.println("========================================");
```

Figure 20: Implementation of Storage Usage of EDMDR (MyController.java)

To evaluate the total RAM consumption or Memory Usage of EDMDR in fog environment,used MemoryLogger class from ca.pfv.spmf.tool and MyController from org.fog.placement invoked in TestApplication.java. The MemoryLogger are already present in SPMF libraries and extended in iFogSim. For this experiment, implemented logic in MyController class. As shown in the figure 20, getMaxMemory() function invoked from the MemoryLogger class to indicate exact RAM consumption of EDMDR. For the memory consumption calculation the HeapMemory was used, it shown the total memory consumption in megabytes.

7.5 Exp5: Energy Consumption


TestApplication.java | MyController.java | MyFogDevice.java | PowerDatacenter.java

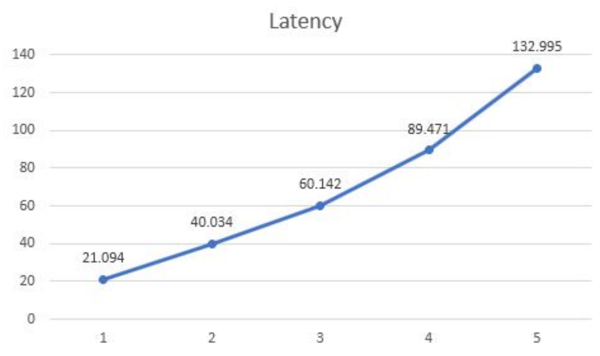
```
    private void printPowerDetails() {
        for(MyFogDevice fogDevice : getMyFogDevices()){
            System.out.println(fogDevice.getName() + " : Energy Consumed = "+fogDevice.getEnergyConsumption());
        }
    }
```

Figure 21: Implementation of Energy Consumption of EDMDR(MyController.java)

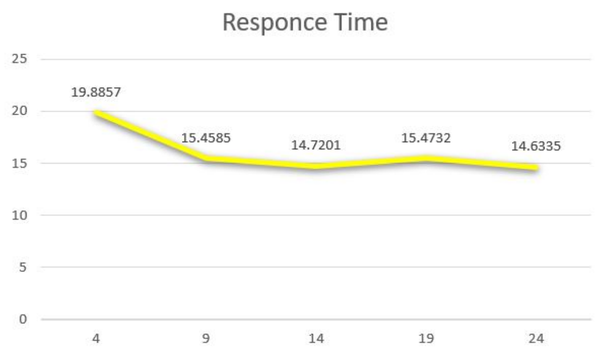
To calculate the total energy consumption of cloud of EDMDR in fog environment, used PowerDatacenter class from org.cloudbus.cloudsim.power and MyFogDevice class from org.fog.entities and MyController from org.fog.placement invoked in TestApplication.java. The PowerDatacenter class are already present in iFogSim. For this experiment,implemented logic in MyFogDevice and MyController class. As shown in the figure 21, thegetEnergyConsumption() function invoked from the MyFogDevice which extends PowerDatacenter class to show exact energy consumption of data centre.

8 Discussion

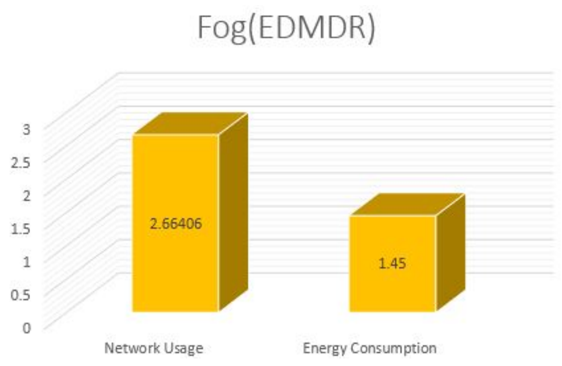
In this manual,successfully implemented simulation and examined results of various evaluation metrics such as Total latency, Response time, Network Usage, Storage Usage and Energy Consumption. Also, created fog devices,fog gateways and cloud datacenter to reduce overall latency of IoT sensors and actuators using EDMDR model. For the evaluation, used number of files and size of files contains candidate nodes (synthetic dataset) as an input. The figure 22 is shown the graphical visualization of the final results in Eclipse Console using iFogSim.



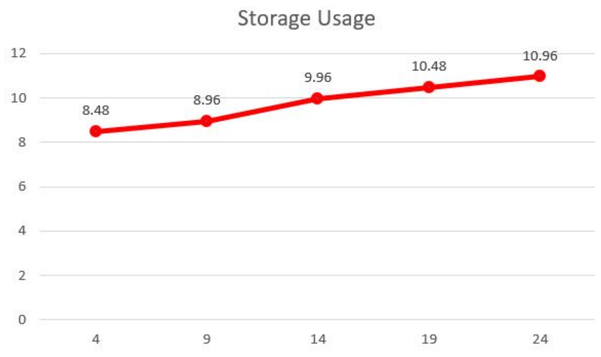
(a) Latency



(b) Response Time



(c) Network and Energy Consumption



(d) Storage Usage

Figure 22: Evaluation Metrics results of EDMDR

References

Gupta, H., Dastjerdi, A., Ghosh, S. and Buyya, R. (2016). ifogsim: A toolkit for modeling and simulation of resource management techniques in internet of things, edge and fog computing environments, *Software: Practice and Experience* . JCR Impact Factor:1.931:(2017).

Hamrouni, T., Slimani, S. and Charrada, F. B. (2016). A survey of dynamic replication and replica selection strategies based on data mining techniques in data grids, *Eng. Appl. Artif. Intell.* **48**: 140–158. JCR Impact Factor:2.819:(2016).
URL: *https://doi.org/10.1016/j.engappai.2015.11.002*

Mahmud, M. and Buyya, R. (2018). *Modelling and Simulation of Fog and Edge Computing Environments using iFogSim Toolkit.*

Mansouri, N., Javidi, M. and Mohammad Hasani Zade, B. (2019). Using data mining techniques to improve replica management in cloud environment, *Soft Computing* . JCR Impact Factor:2.237:(2019).