National College of Ireland

Advancement in data locality during data retrieval using community mapping of fog nodes

MSc Research Project
Cloud Computing

Sonia Suhas Ghongadi
Student ID: 20104707

School of Computing
National College of Ireland

Supervisor: Prof. Vikas Sahni

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Sonia Suhas Ghongadi
Student ID:	20104707
Programme:	Cloud Computing
Year:	2021
Module:	MSc Research Project
Supervisor:	Prof. Vikas Sahni
Submission Due Date:	16/08/2021
Project Title:	Advancement in data locality during data retrieval using community mapping of fog nodes
Word Count:	837
Page Count:	11

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	16th August 2021

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.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

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

Advancement in data locality during data retrieval using community mapping of fog nodes

Sonia Suhas Ghongadi
20104707

1 Introduction

Configuration manual consists of a detailed, step-by-step installation of dependencies, libraries, packages required for project implementation. This manual describes the installation of different libraries and packages that were required to build this project. It is shown that to implement the proposed algorithm, it required the various node.js libraries such as 'uuid' to create community hash, message broker queue service 'AMPQHelper(RabbitMQ)' for data transfer and sensor-fog node connection establishment, and 'peerjs' for peer to peer connection between fog nodes. This manual will help to analyze the brief details of community mapping of fog nodes to improve the data locality and the impact on latency and response time. This research project is implemented on the local machine(HP Pavilion).

2 Prerequisites and Configurations of System

To design and evaluate the proposed algorithm, following tools and setup is required:

1. Software Tools and dependencies
    - Operating System : Ubuntu 20.04.2 LTS
    - Integrated Development Environment : Visual Studio Code (*Visual Studio Code* (n.d.))
    - Node.js module
2. Hardware Specifications:
    - System Type: 64-bit, x-64 based processor
    - Processor : Intel® Core™ i7-1065G7 CPU @ 1.30GHz × 8
    - Specifications : 8GB Memory, 512GB SSD

To implement this research project, multiple libraries and dependency packages were required. The detailed installation of libraries is described as following:

2.1 Message Passing Queue Service - AMPQHelper

To establish the connection for data transfer between sensors and fog nodes, as well as between fog nodes and fog nodes to cloud server, we have used the RabbitMQ message-passing queue service named as 'ampqlib' (*AMQP 0-9-1 library and client for Node.JS* (n.d.)) install this library, use the command as shown in following Figure 1. It has installed all the required libraries to establish the connection between sensors and fog nodes.

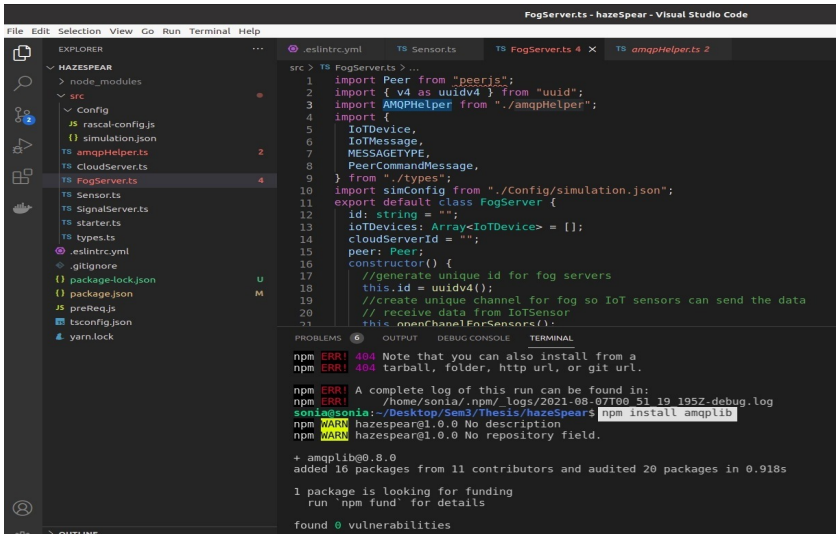

Figure 1: Installation of AMPQLib

2.2 Winston for logging

This tool is used for logging details. Please run the mentioned command as shown in following figure2.

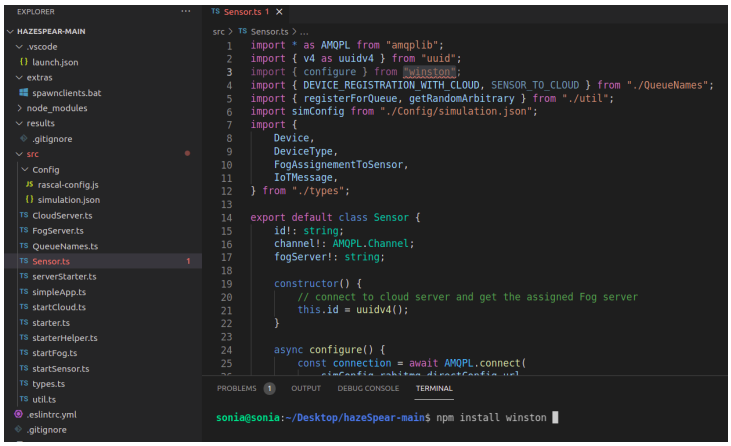

Figure 2: Installation of Winston

2.3 Community Hash Generator : uuid

Once the community is generated, we have assigned the hash value to each community for identification and uniqueness. To do the same, we have used the node.js module named as 'uuidv4'(*UUIDV4* (n.d.)) where v4 stands for Version 4 which generates the random string hash values. To install the uuidv4 library, write the highlighted command as shown in the following Figure 3.

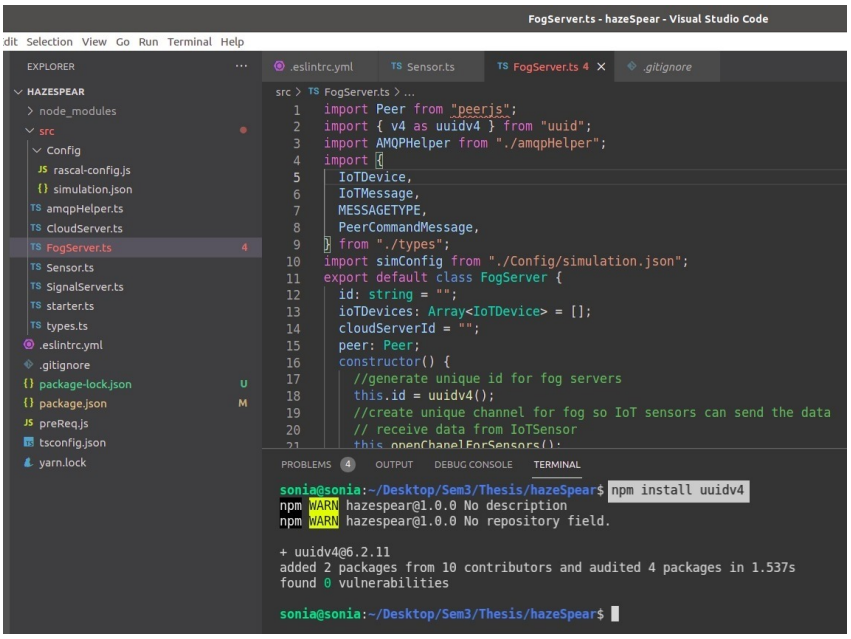

Figure 3: Installation of uuidv4

2.4 Enable the connection and open data channels

As stated earlier, RabbitMQ is used for communication between devices and various queues are created for data transfers. RabbitMQ is maintained by the docker community (*RabbitMQ* (n.d.)). RabbitMQ stores the data based on the node name which is the hostname by default. So, while running the daemon, need to specify the hostname to keep the track of data and not to get the random hostname.

To start the queues and open the data channels for devices, need to run highlighted command in below screenshot 4.

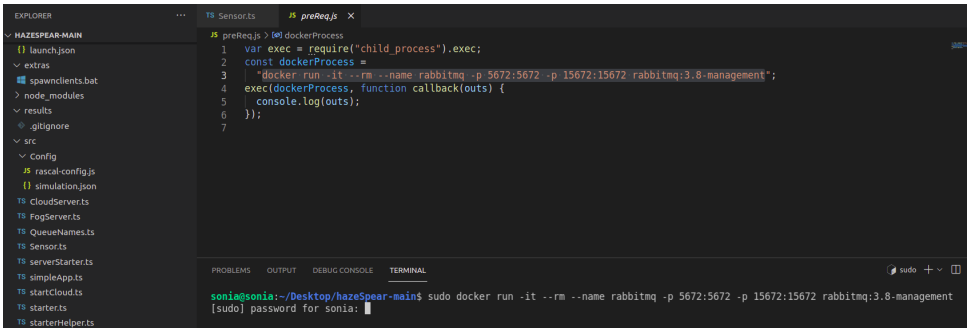

Figure 4: Start RabbitMQ

2.5 Project Setup

Once RabbitMQ is started, run the following command to start the project:

Figure 5: Start the execution

Further, to reset the queue between each run, need to run following commands:

- Find the queue name: Get the name of container 'docker ps, this will print out a table, look for NAMES and copy it.

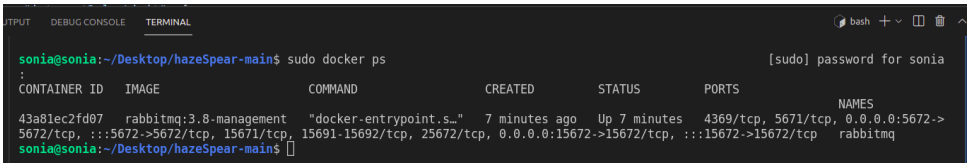

Figure 6: Find the queue name

- Add the queue name in the command as shown in below screenshot.

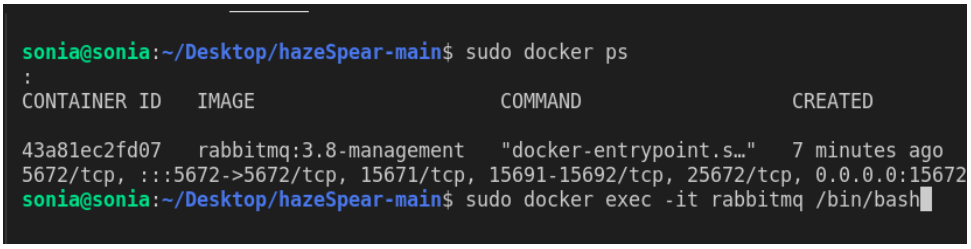

Figure 7: Run for respective container

- Reset the queue with the following commands as shown in below screenshot:

```
sonia@sonia:~/Desktop/hazeSpear-main$ sudo docker ps                                                    [sudo] password for sonia
:
CONTAINER ID   IMAGE                     COMMAND                  CREATED         STATUS         PORTS
                                                                                                    NAMES
43a81ec2fd07   rabbitmq:3.8-management   "docker-entrypoint.s…"   7 minutes ago   Up 7 minutes   4369/tcp, 5671/tcp, 0.0.0.0:5672->
5672/tcp, :::5672->5672/tcp, 15671/tcp, 15691-15692/tcp, 25672/tcp, 0.0.0.0:15672->15672/tcp, :::15672->15672/tcp   rabbitmq
sonia@sonia:~/Desktop/hazeSpear-main$ sudo docker exec -it rabbitmq /bin/bash
root@43a81ec2fd07:/# rabbitmqctl stop_app && rabbitmqctl reset && rabbitmqctl start_app
```

Figure 8: Reset the RMQ

3 Project Implementation

This sections states the code snippet for respective objectives.

3.1 Configurations

Configurations are mentioned as shown in following. While taking the experiments, can modify this file based on the requirement and change the behaviour of device and environment.

simulation.json

src > Config > {} simulation.json > formCommunity

```
    "formCommunity": false,
    "peerJS": {
        "cloudServerName": "CloudServer",
        "host": "localhost",
        "port": 9000,
        "path": "/hazespear"
    },
    "simulation": {
        "sensorParticipationSize": 20,
        "delayBetweenExecutionInSeconds": 2,
        "numberOfFogNode": 5,
        "numberOfSensorNode": 10
    },
    "fog": {
        "retriesBeforeFail": 5,
        "waitForCommunitySync": 200,
        "perDeviceMessageRetentionSize": 1000,
        "retention": false,
        "internetDelayLimit": {
            "upperBound": 100,
            "lowerBound": 200
        }
    },
    "sensor": {
        "delayBetweenSensorAwakeInSeconds": 0.5,
        "tempratureGeneration": {
            "lowestLimit": 5,
            "highestLimit": 20
        }
    },
    "rabitmq": {
        "directConfig": {
            "url": "amqp://guest:guest@localhost:5672"
        },
        "rascalConfig": {
            "vhosts": {
                "v1": {
                    "connection": {
                        "url": "amqp://guest:guest@localhost:5672"
```

Figure 9: Configurations

3.2 Creating the fog infrastructure

To build the fog environment, this research has designed a class named as 'FogServer' that creates fog nodes with the use of different configurations and register the instance to cloud. Further, with the help of RabbitMQ helper, it will capture the data from sensors as shown in following snippet of code 10.

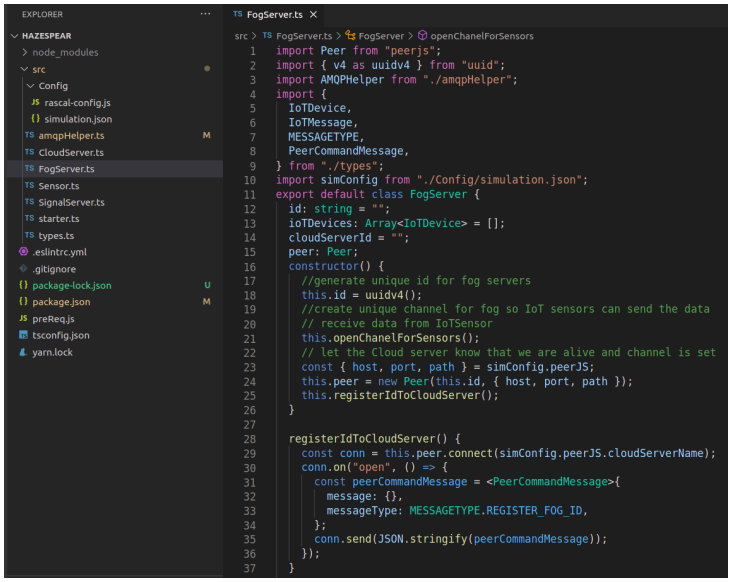

Figure 10: Creating fog environment(FogServer.ts)

3.3 Cloud Server

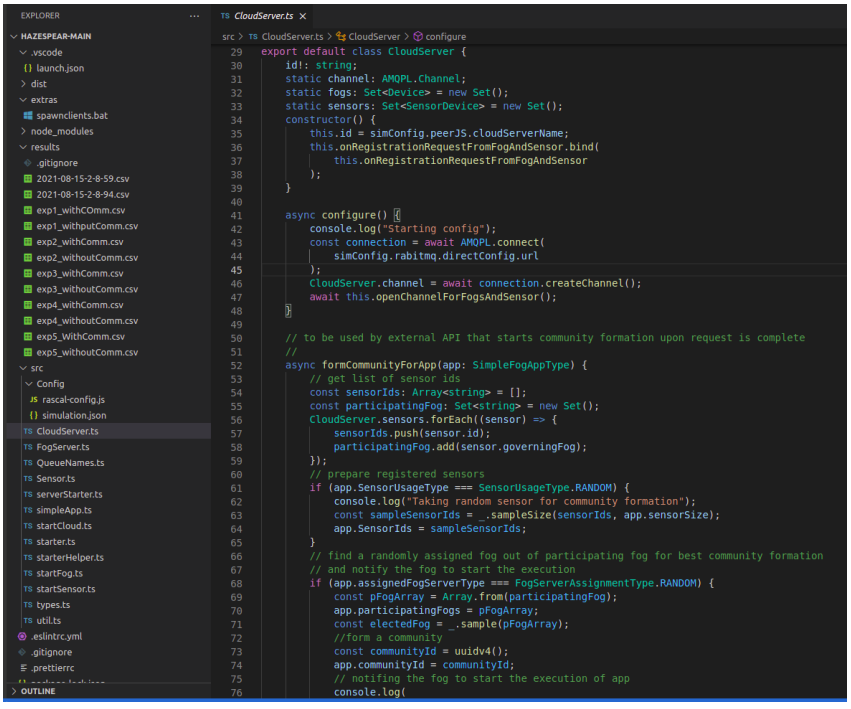

Figure 11: Creating cloud server(CloudServer.ts)

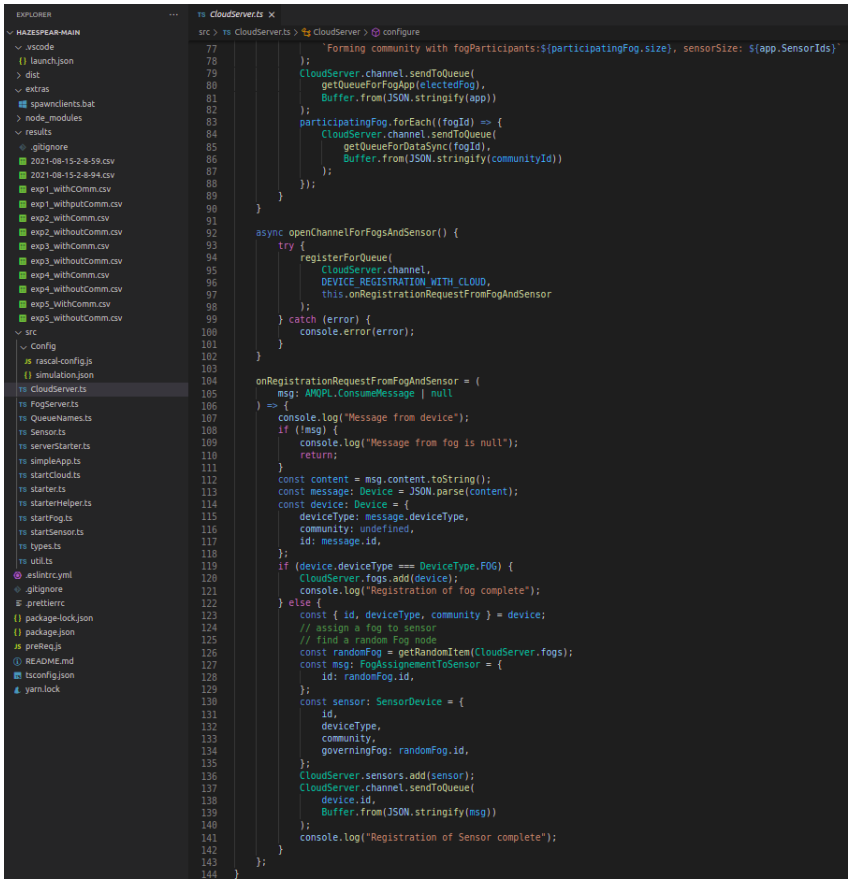

Figure 12: Creating cloud server(CloudServer.ts)

3.4 Connection establishment between Fog nodes and IoT

As mentioned earlier, this research has used the RabbitMQ helper library to transfer the data from IoT to Fog nodes. As shown in the following snippet of code 13 that shows the implementation of data generated at IoT device and sending the data to the fog nodes.

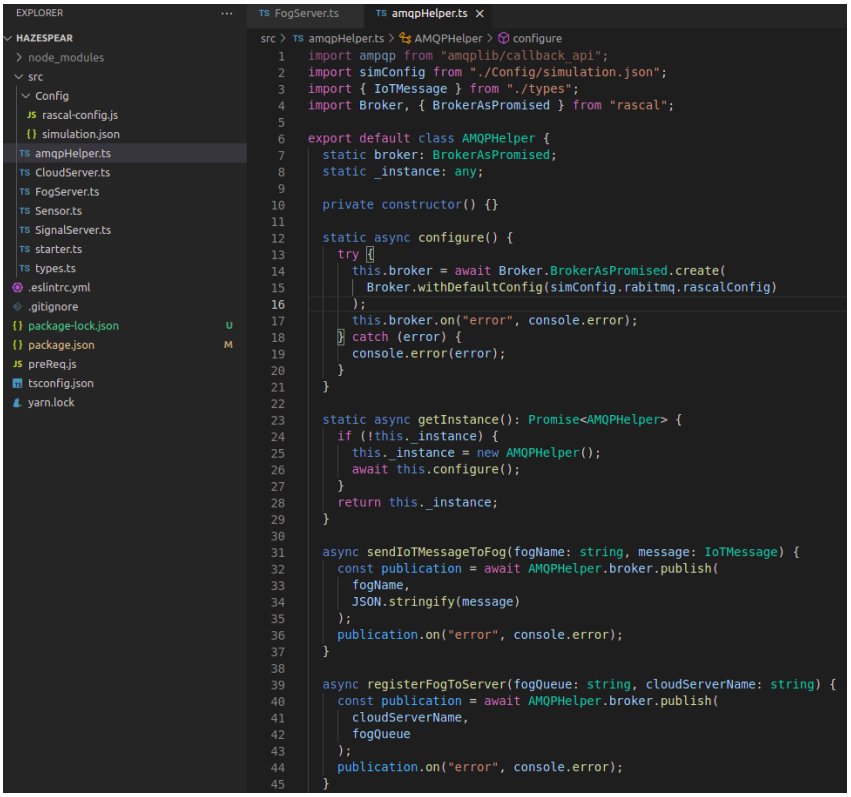

Figure 13: Connection establishment between fog nodes and IoT(AMPQHelper.ts)

3.5 Community Creation

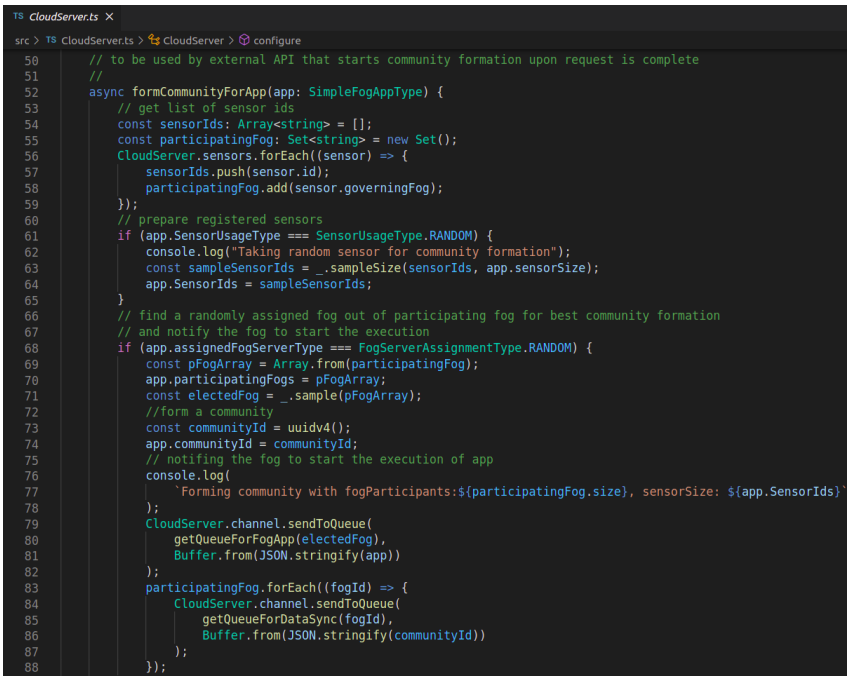

Figure 14: Community Creation

3.6 Data Synchronization

Figure 15: Data Synchronization)

3.7 Traditional Approach

FogServer.ts

src > FogServer.ts > FogServer > startSyncListnerAndSendorForCommunity

```
143     runApp = async () => {
144         const delay = (timeInMilliSec = simConfig.fog.waitForCommunitySync) => {
145             return new Promise((res, rej) => {
146                 setTimeout(res, timeInMilliSec);
147             });
148         };
149         let missedData = 0;
150         const getSensorData: SensorDataProvider = async (
151             sensorId: string,
152             counter: number
153         ): Promise<IoTMessage> => {
154             if (simConfig.formCommunity) {
155                 // see if data is available or wait for 20 millisecond
156                 // wait upto 2 seconds before fetching data from community
157                 for (let i = 0; i < simConfig.fog.retriesBeforeFail; i++) {
158                     if (this.ioTDevices.has(sensorId)) {
159                         const deviceMessages = this.ioTDevices.get(sensorId);
160                         const message = deviceMessages?.find(
161                             (element) => element.counter === counter
162                         );
163                         if (message) return message;
164                     } else {
165                         await delay();
166                     }
167                 }
168                 missedData += 1;
169             }
170             // simulates delay to fetch data from cloud
171             // below limits are taken from 100 to 200 milliseconds
172             // while 100 seconds is on extreme low end
173             const realTimeDelya = getRandomInt(
174                 simConfig.fog.internetDelayLimit.lowerBound,
175                 simConfig.fog.internetDelayLimit.upperBound
176             );
177             await delay(realTimeDelya);
178             const { lowestLimit, highestLimit } =
179                 simConfig.sensor.tempratureGeneration;
180             const simulatedMessage: IoTMessage = {
181                 counter: counter,
182                 deviceId: sensorId,
183                 value: getRandomArbitrary(lowestLimit, highestLimit),
184                 timestamp: "",
185             };
186             return simulatedMessage;
187             // if its a community simply get it from local storage
188         };
189         const sendDataToMonitor: MonitorDataProvider = async (
190             counter: number,
191             timeTaken: number
192         ) => {
193             console.warn(`Counter:${counter},timeTaken: ${timeTaken}`);
194             // dump data to csv file
195             await csvWriter.writeRecords([{ counter, timeTaken }]);
196         };
197         await this.simpleApp.executableAlgorithm(
198             this.app,
199             getSensorData,
200             sendDataToMonitor
```

Figure 16: Traditional Approach

3.8 Results

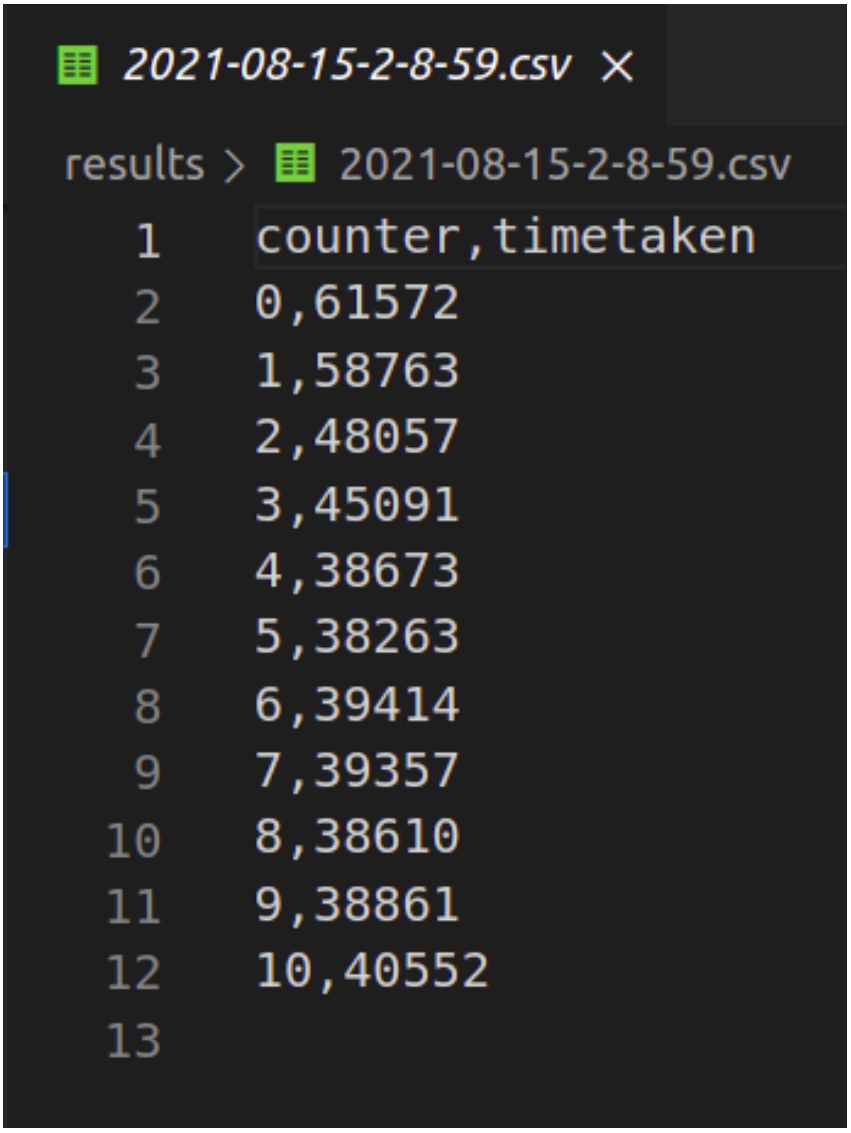

Figure 17: Generated Output

References

AMQP 0-9-1 library and client for Node.JS (n.d.).
URL: *https://www.npmjs.com/package/amqplib*

RabbitMQ (n.d.).
URL: *https://hub.docker.com/_/rabbitmq*

UUIDV4 (n.d.).
URL: *https://www.npmjs.com/package/uuidv4*

Visual Studio Code (n.d.).
URL: *https://code.visualstudio.com/download*