

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



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 com-
	munity 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.

<u>ALL</u> 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 R Core $^{\mathbb{M}}$ i7-1065G7 CPU @ 1.30GHz \times 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.

			FogServer.ts - hazeSpear - Visual Studio Code
File Ec	it Selection View Go Run Term	inal Help	
ф			
	 VAZESPEAR > node_modules > src > config > srcsal-config js O simulation.json TB angpHelper.ts TB fogServer.ts TB fogServer.ts TB signalServer.ts <l< td=""><td></td><td><pre>src > 18 FogSeverts > 1 simport Peer from "peer]s"; 2 import {v4 as unidv4 } from "uuid"; 3 import { 3 import { 4 import { 5 introveride; 6 import { 5 introveride; 7 import { 6 import { 7 im</pre></td></l<>		<pre>src > 18 FogSeverts > 1 simport Peer from "peer]s"; 2 import {v4 as unidv4 } from "uuid"; 3 import { 3 import { 4 import { 5 introveride; 6 import { 5 introveride; 7 import { 6 import { 7 im</pre>
8	≰ yarn.lock		<pre>PMOLENS @ OUTUP DEBUGCONSOLE TEXAMUL npm EFRI 404 Note that you can also install from a npm EFRI 404 note that you can also install from a npm EFRI 404 tarball, folder, http url, or git url. npm EFRI A complete log of this run can be found in: npm EFRI A complete log of this run can be found in: npm EFRI A complete log of this run can be found in: npm EFRI A complete log of this run can be found in: npm EFRI A complete log of this run can be found in: npm EFRI A complete log of this run can be found in: npm AFRI hazespeargl.0.0 No description npm AFRI hazespeargl.0.0 No repository field. + amgplib@6.8.6 added 16 packages from 11 contributors and audited 20 packages in 0.918s 1 package is looking for funding run npm fund for details found 0 vulnerabilities</pre>

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 figure 2.



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.

EXPLORER	Tš Sensor.ts Jš preReq.js X	
✓ HAZESPEAR-MAIN		
1 launch.json	<pre>1 var exec = require("child_process").exec;</pre>	
\sim extras	2 const dockerProcess =	
💶 spawnclients.bat	3 GOCKET FUN - 1t - Fm - hame rabbitmg - 50/2:50/2 - p 150/2:150/2 Fabbitmg:3.8-management; 4 gocketProcess functions callback(outp) (
> node_modules	5 console log(outs):	
✓ results	6 });	
.gitignore		
∨ src		
\sim Config		
JS rascal-config.js		
() simulation.json		
TS CloudServer.ts		
TS FogServer.ts		
TS QueueNames.ts		
TS Sensor.ts		
TS serverStarter.ts		
TS simpleApp.ts		
TS startCloud.ts	sonia@sonia:~/Desktop/hazeSpear-main\$ sudo docker run -itrmname rabbitmg -p 5672:5672 -p 15672:15672 rabbitmg:3.8-manageme	ent
T\$ starter.ts	[sudo] password for sonia:	

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.

JTPUT DEBUG CONSOLE	TERMINAL					() t		□ 🕯 ·
sonia@sonia:~/De	esktop/hazeSpear-main\$ suc	lo docker ps				[sudo] passw	ord for	sonia
CONTAINER ID I	IMAGE	COMMAND	CREATED	STATUS	PORTS		urc	
43a81ec2fd07 r 5672/tcp, :::567 sonia@sonia:~/De	rabbitmq:3.8-management 72->5672/tcp, 15671/tcp, 1 esktop/hazeSpear-main\$ []	"docker-entrypoint.s" L5691-15692/tcp, 25672/tcp	7 minutes ago 5, 0.0.0.0:15672	Up 7 minutes ->15672/tcp, ::	4369/tcp, :15672->150	ма 5671/tcp, 0 572/tcp ra	15 .0.0.0:5 obitmq	672->

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\$ su	udo docker ps			[sudo] pa	ssword for sonia
: CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
43a81ec2fd07 5672/tcp, :::5 sonia@sonia:~/ root@43a81ec2f	rabbitmq:3.8-management 672->5672/tcp, 15671/tcp, Desktop/hazeSpear-main\$ su d07:/# rabbitmqctl stop_ag	"docker-entrypoint.s…" 15691-15692/tcp, 25672/tc do docker exec -it rabbit ه ه rabbitmqctl reset ه۵	7 minutes ago p, 0.0.0.0:15672 mq /bin/bash a rabbitmqctl sta	Up 7 minutes ?->15672/tcp, :: urt_app	4369/tcp, 5671/tcp ::15672->15672/tcp	, 0.0.0.0:5672-> rabbitmq

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.



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.

	TS FogServer.ts X
EXFLORGR V AZESPEAR > node, modules > src V Config 35 rascal-configis U simulation, son Ts amgaPreferents TS CloudServerts TS PogServerts TS Sensorts TS Sensorts TS SignalServerts TS Staterts	<pre>TF FogGerverts X src > TB FogGerverts > B FogGerver > @ openChanelForSensors 1 import { v4 as uuidv4 } from "uuid"; 3 import { v4 as uuidv4 } from "uuid"; 4 import { 5 IoTDevice, 6 IoTMessage, 7 MESSAGETYPE, 8 PeerCommandMessage, 9 } from "./types; 10 import simConfig from "./Config/simulation.json"; 11 export default class FogServer { 12 id; string = "; 13 is string = "; 14 is string = "; 15 is string = "; 16 is string = "; 17 is string = "; 18 is string = "; 19 is string = "; 10 is string = "; 10 is string = "; 10 is string = "; 11 is string = "; 11 is string = "; 12 is string = "; 13 is string = "; 14 is string = "; 15 is string = "; 15 is string = "; 16 is string = "; 17 is string = "; 18 is string = "; 19 is string = "; 19 is string = "; 10 is string = "; 11 is string = "; 11 is string = "; 12 is string = "; 13 is string = "; 14 is string = "; 15 is string = "; 15 is string = "; 16 is string = "; 17 is string = "; 18 is string = "; 19 is string = "; 10 is stri</pre>
Is staterts Is bpests @.eslintrcyml or _gitgionore () packagejoon I) packagejoon II) packagejoon II) packagejoon II) packagejoon III) packagejoon III) packagejoon III) packagejoon III) packagejoon IIII) packagejoon IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII	<pre>ioTDevices: Array-CoTDevice> = []; devices: Array-CoTDevi</pre>
	<pre>28 registeriolocloudserver() { 29 const conn = this.peer.connect(simConfig.peerJS.cloudServerName); 30 const peerCommandMessage = <peercommandmessage>{ 31 const peerCommandMessage = <peercommandmessage>{ 32 message(), 33 message(), 34 }; 35 conn.send(JSON.stringify(peerCommandMessage)); 36 }); 37 }</peercommandmessage></peercommandmessage></pre>

Figure 10: Creating fog environment(FogServer.ts)

3.3 Cloud Server

	TS CloudServer.ts ×
✓ HAZESPEAR-MAIN	
✓ .vscode	
{} launch.json	30 id!: string;
	31 static channel: AMQPL.Channel;
✓ extras	<pre>32 static fogs: Set<device> = new Set();</device></pre>
snawnclients hat	<pre>33 static sensors: Set<sensordevice> = new Set();</sensordevice></pre>
) node modules	34 constructor() {
> node_modules	35 (n15.10 = Simconig.peer35.coudservername;
* Tesuics	30 this unkey is a first at the page of th
2021-08-15-2-8-59.csv	
E 2021-08-15-2-8-94.csv	
exp1_withCOmm.csv	41 async configure() 🛛
exp1_withputComm.csv	<pre>42 console.log("Starting config");</pre>
exp2_withComm.csv	<pre>43 const connection = await AMQPL.connect(</pre>
exp2_withoutComm.csv	44 simConfig.rabitmq.directConfig.url
exp3_withComm.csv	45);
exp3_withoutComm.csv	<pre>46 CloudServer.channel = await connection.createChannel(); </pre>
exp4_withComm.csv	4/ await this.openchannetForFogsAndSensor();
exp4_withoutComm.csv	
exp5 WithComm.csv	79 50 // to be used by external API that starts community formation upon request is complete
exp5 withoutComm.csv	
× src	52 async formCommunityForApp(app: SimpleFogAppType) {
× Confin	
15 rarcal-confin in	
() simulation ison	
	56 CloudServer.sensors.forEach((sensor) => {
TS Cloudservents	57 sensorids.push(sensor.id);
TS FogServer.ts	58 participatingrog.add(sensor.governingrog);
TS QueueNames.ts	
TS Sensor.ts	61 if (an SensorlisaneType === SensorlisaneType RANDOM) {
TS serverStarter.ts	62 console.log("Taking random sensor for community formation"):
TS simpleApp.ts	63 const sampleSensorIds = .sampleSize(sensorIds, app.sensorSize);
TS startCloud.ts	64 app.SensorIds = sampleSensorIds;
TS starterHelper.ts	
TS startFog.ts	
TS startSensor.ts	<pre>68 if (app.assignedFogServerType === FogServerAssignmentType.RANDOM) {</pre>
TS types.ts	by const program a reaver comparticipatingFog);
TS util.ts	70 app.participartingrogs = programay;
Seslintrc.yml	72 //form a computity
 aitianore 	73 const communityId = uuidy4();
E .prettierrc	74 app.communityId = communityId;
> OUTLINE	76 console.log(

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.

	TS FogServer.ts TS amqpHelper.ts X
✓ HAZESPEAR	src > 18 amqpHelper.ts > 😫 AMQPHelper > 🏵 configure
 > node_modules > src < Config J\$ rascal-config.js {} simulation.json T\$ amgpHelperts T\$ CloudServerts T\$ FogServerts 	<pre>import ampapp from "ampplib/callback api"; import simConfig from "./Config/simulation.json"; import { IoTMessage } from "./types"; import Broker, { BrokerAsPromised } from "rascal"; export default class AMQPHelper { static broker: BrokerAsPromised; static _instance: any; }</pre>
 T3 Sensor.ts T3 SignalServer.ts T3 starter.ts T5 types.ts Ø .eslintrc.yml 	<pre>10 private constructor() {} 11 12 static async configure() { 13 try [] 14 this.broker = await Broker.BrokerAsPromised.create(15 Broker.withDefaultConfig(simConfig.rabitmq.rascalConfig)</pre>
.gitignore	16); 17 this broker on("error" console error);
 () package-lock.json () package.json Js preReq.js Itsconfig.json 4 yarn.lock 	<pre>1/ catch (error) { 20 } 21 } 22</pre>
	<pre>23 static async getInstance(): Promise<amqphelper> { 24</amqphelper></pre>
	<pre>44 publication.on("error", console.error); 45 }</pre>

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

3.5 Community Creation

TS CloudS	erver.ts ×						
	CloudServer.ts > 😫 CloudServer > 😚 configure						
	// to be used by external API that starts community formation upon request is complete						
	async formCommunityForApp(app: SimpleFogAppType) {						
	// get list of sensor lds						
54	const sensoriads: Array(string) = []; const participation for set $()$;						
	CloudServer sensors for Fach((sensor) => {						
	sensorIds.push(sensor.id):						
	<pre>participatingFog.add(sensor.governingFog);</pre>						
	3);						
	// prepare registered sensors						
	if (app.SensorUsageType === SensorUsageType.RANDOM) {						
	<pre>console.log("Taking random sensor for community formation");</pre>						
	<pre>const sampleSensorIds =sampleSize(sensorIds, app.sensorSize);</pre>						
64	app.SensorIds = sampleSensorIds;						
65	}						
67	// ind a random y assigned tog out of participating fog for best community formation						
	// and notify the roy to start the execution if (ann essignedFonServerTure === FonServerAssignmentType PANDOM) {						
69	const p Englarray = A rray from (naticipationEngl):						
	app.participatindFour pFonArray;						
	<pre>const electedFog = .sample(pFogArray);</pre>						
	//form a community						
	<pre>const communityId = uuidv4();</pre>						
	app.communityId = communityId;						
	// notifing the fog to start the execution of app						
	console.log(
	Forming community with fogParticipants:\${participatingFog.size}, sensorSize: \${app.SensorIds}						
); [] condication channel conditionuous(
81	Buffer, from (JSON string fv(app))						
):						
	<pre>participatingFog.forEach((fogId) => {</pre>						
	CloudServer.channel.sendToQueue(
	getQueueForDataSync(fogId),						
	Buffer.from(JSON.stringify(communityId))						
88	});						

Figure 14: Community Creation

3.6 Data Synchronization



Figure 15: Data Synchronization)

3.7 Traditional Approach

TS FogServer.ts X			
src > TS FogServer.ts > 😫 FogServer > 🗘 startSyncListnerAndSendorForCommunity			
143	<pre>runApp = async () => {</pre>		
144	<pre>const delay = (timeInMilliSec = simConfig.fog.waitForCommunitySync) => {</pre>		
145	return new Promise((res, rej) => {		
146	setTimeout(res, timeinMittiSec);		
147			
148	}; lot miccodData		
	const getSonserData: SonserDataProvider - asvec /		
150	const getsensorbata. Sensorbatariovider = async (
151	counter: number		
152): Promise <totmessages ==""> {</totmessages>		
154	if (simConfig_formCommunity) {		
155	// see if data is available or wait for 20 millisecond		
156	<pre>// wait upto 2 seconds before fetching data from community</pre>		
157	<pre>for (let i = 0: i < simConfig.fog.retriesBeforeFail: i++) {</pre>		
158	if (this.ioTDevices.has(sensorId)) {		
159	<pre>const deviceMessages = this.ioTDevices.get(sensorId);</pre>		
160	<pre>const message = deviceMessages?.find(</pre>		
161	(element) => element.counter === counter		
162			
	if (message) return message;		
	} else {		
	await delay();		
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		
174	const reallimeDelya = getRandomint(
175	simConfig fog internetDelayLimit upperBound		
176).		
177	await delav(realTimeDelva):		
178	const { lowestLimit, highestLimit } =		
179	simConfig.sensor.tempratureGeneration:		
180	const simulatedMessage: IoTMessage = {		
181	counter: counter,		
182	deviceId: sensorId,		
183	<pre>value: getRandomArbitrary(lowestLimit, highestLimit),</pre>		
	timestamp: "",		
	};		
186	return simulatedMessage;		
187	// if its a community simply get it from local storage		
188	};		
189	const sendDataloMonitor: MonitorDataProvider = async (
190	counter: number,		
102) -> /		
102	/ \ console warn(`Counter:\$/counter] timeTaken: \${timeTaken}`).		
104	// dump data to csv file		
194	await csyWriter writeRecords/[/ counter_timeTaken_]).		
195	ware covaries write ecolos ([{ counter, timeraken }]);		
197	await this simpleApp executableAlgorithm(
198	this.app.		
199	getSensorData,		
200	sendDataToMonitor		

Figure 16: Traditional Approach

3.8 Results

■ 2021-08-15-2-8-59.csv ×			
results	> 🖽 2021-08-15-2-8-59.csv		
1	counter,timetaken		
2	0,61572		
3	1,58763		
4	2,48057		
5	3,45091		
6	4,38673		
7	5,38263		
8	6,39414		
9	7,39357		
10	8,38610		
11	9,38861		
12	10,40552		
13			

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