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:	6447
Page Count:	19

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

Abstract

Because of the exponential expansion of IoT devices, enormous amounts of data are generated and collected. Increased inefficiency and high latency in time-sensitive applications have been seen in the Cloud Data Center-to-IoT method of calculation and storage. To overcome these shortcomings new complementary approach to cloud computing has been introduced named Fog Computing. This decentralized approach is gaining attention in building smart cities and healthcare infrastructure. Data management is one of the key components of fog computing, and control of the data locality is also necessary. When data is created and transmitted to the fog layer, the typical technique saves the data in a hazardous way in the network. Due to this reason, in time-sensitive applications, data retrieval requests or decision-making processes take high response time and latency. To address this issue, this research study has implemented an algorithm of community mapping of fog nodes where the data is stored in a covered area or community of fog nodes as well as replicated within the same community. The experimental results have shown that with community mapping, the time taken to execute the application is reduced by approximately 79% as compared to the traditional approach.

1 Introduction

The Internet of Things (IoT) is revolutionizing how people interact with the physical world by bringing an exceptional level of connection to it. The Internet of Things (IoT) refers to the deployment of a network of interconnected smart devices to assist with everyday tasks (Hu et al. (2017)). By allowing widespread participation of users and, in particular, enhancing machine and sensor/actuator connections, IoT will offer new applications with infinite potential and huge effects. As mentioned by Cisco in their Annual Internet Report updated in March 2020 that, connected home applications and connected cars are the fastest-growing application type in IoT and Machine-to-Machine connections will grow up to 14.7 billion by 2023.[1] The projected large number of IoT devices and vast amounts of accessible data present new potential for developing services that will benefit society, but they also offer major obstacles. By looking at the requirements and behavior of IoT applications, Cloud-IoT device architecture does not provide satisfactory performance for time-sensitive applications where decision making needs to be performed within

[1] https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

a fraction amount of time(Bellavista et al. (2019)). This two-tier(Cloud-IoT) architecture with several sensors detecting and transmitting data to the cloud to be analyzed to determine how the system should behave would barely satisfy the needs of these systems. To overcome these shortcomings, a new paradigm has been introduced to complement cloud computing named 'Fog Computing' (Vaquero and Rodero-Merino (2014)).

Unlike cloud computing, fog computing is geographically distributed architecture that at the edge network, heterogeneous devices are connected to deliver compute, storage, and network services. To solve different difficulties in low latency, high reliability and security, high performance, mobility, and interoperability, fog computing architecture provides an additional resource-rich layer between end devices and the cloud. This computing, with 3 layers of architecture consists of Cloud Data Centers, fog nodes, and IoT device or edge devices. Locally, fog nodes at the edge networks collect data generated by sensors, analyze it, and store it in a local area network. It significantly minimizes data transfer across the Internet and offers endpoints with fast, high-quality localized services. As a result, it provides low latency and satisfies the requirement for real-time interactions.

A large number of connected IoT devices are generating a massive amount of data. Handle this large amount of data is one of the key factors in fog computing. The fog layer acts as a bridge between the IoT and the cloud, facilitating data management (Sadri et al. (2021)). As result, it increases cloud data storage capacity and enhances IoT service usability. For data retrieval in time-sensitive applications, it is crucial to achieve the performance of the application and reflect the decisions. Traditionally, data is stored haphazardly at the network between fog nodes and while data access requests, it takes a long time to complete it (Stavrinides and Karatza (2020)). Due to this, it raises latency issues in time-sensitive IoT applications. For this purpose, data locality needs to be improved and should have control over where the data is going to be stored and which device has the required data.

1.1 Background and Motivations

Data distribution techniques are showing positive results to achieve the performance of the application in fog computing Sadri et al. (2021). Fog resource management (Filiposka et al. (2018)) is based on graphical communities where it will create the network of fog nodes. Dividing the fog nodes within groups or colonies (Lera et al. (2019)) for assigning the required applications which have helped to enhance the service availability and Quality of Service(QoS). A tree-based technique (Confais et al. (2019)) which have been used to distribute the records that save the location of entity copies that helped to locate the data over the network.

However, data locality in fog computing plays an important role. As stated in Steffenel (2018), the algorithm should have control over the data locality and data should be always available for data retrieval. With the help of community mapping of fog nodes, data will be stored within the area and it will be available for easy data retrieval (Skarlat et al. (2017)).

1.2 Specifications of Research Project

This section describes the objectives and contribution of this research study.

1.2.1 Research Question

This research study addresses the following research question:

> Can adoption of community mapping methodology for fog nodes improve the data locality and job response time for data retrieval in fog-based environments?

1.2.2 Objectives and Contributions

- With the use of the proposed algorithm, design and develop the communities of fog nodes for data retrieval requests using the community mapping to improve the data locality.
- Replicate the data within the newly created community for data retrieval with low latency.
- Examine and evaluate the results of the proposed algorithm with and without community mapping of fog nodes.

The contribution consists of:

- Critically analyze and review the previous research studies performed in the same area.
- Design and implement the proposed algorithm with the use of different libraries and packages.
- Design the configuration manual to reproduce the work for others.

2 Related Work

This section covers the previous studies and researches which are done mainly in the fog computing area and how data is distributed over the fog network. Additionally, different approaches of community mapping which have been implemented in fog infrastructure have been discussed in this section.

The following content is divided into following subsections such as 2.1 explains that how fog computing is developed in recent years and what are the current challenges in fog computing with respect to data management due to the high amount of data is being generated by IoT devices. Subsection 2.2 explains multiple community mapping approaches were being used in the fog environment to achieve the application performance as well as low network latency. Subsection 2.3 discusses the current studies which have implemented the docker container in a fog environment. And last subsection 2.4 describes the effect of the data locality in the fog computing area.

2.1 Survey of Fog Computing

Because of the great computing power and storage capacity, cloud computing has been utilized as an efficient way to process data. The majority of calculations, however, take place in the cloud since the cloud computing paradigm is a centralized computing architecture. This implies that all data and requests must be sent to a centralized cloud

(Bellavista et al. (2019)). As a result, with such a large volume of data, network bandwidth is becoming a barrier in cloud computing. This may cause a significant amount of delay. To overcome this shortcoming, a new paradigm has been introduced as a complementary to cloud computing known as Fog Computing. It is defined as (Vaquero and Rodero-Merino (2014)) the network of heterogeneous (frequently autonomous) ubiquitous and distributed devices which communicate with each other and they perform storage, and computing tasks without the need for third-party intervention.

The internet applications that are emerging as a result of the IoT environment's growth are large-scale, latency-sensitive, and were not designed to function alone, but rather to share infrastructure and resources. New criteria, including mobility support, large-scale global dispersion, location awareness, and low latency, must be met for these applications. As a general rule, it is commonly acknowledged that an architecture paradigm focused solely on direct connectivity between IoT devices and the cloud is insufficient for various IoT applications. A distributed intelligent intermediate layer is needed to provide more functionality to the system, such as data processing while devices collect data and before transmitting it to the network and, eventually, the cloud (Bellavista et al. (2019)).

Therefore, fog computing, which connects network edge devices and cloud centers seamlessly, is touted as a more effective way to overcome these constraints. Fog computing is a distributed computing architecture in which multiple heterogeneous devices at the network's edge are ubiquitously connected to deliver elastic compute, communication, and storage services cooperatively (Hu et al. (2017)). The expansion of cloud services to the network edge is the most notable feature of fog computing. By pooling local resources, it brings processing, communication, control, and storage closer to end-users. Data is used by network edge devices that are dispersed around the globe. As a result, data transfer time and network transmission volume are substantially decreased.

Nowadays, the Internet of Things has invaded many aspects of human life all over the world (suchlike home, health care, etc.). The phrase 'big data is widely used to characterize enormous datasets because of the massive rise of universe data created and gathered by IoT environment sensors (Sadri et al. (2021)). Fog computing, according to the OpenFog Consortium[2], is a horizontal architecture for distributed storage and data computation activities close to consumers. In reality, the fog layer acts as a bridge between the IoT and the cloud, facilitating data management. In a nutshell, it increases cloud data storage capacity and enhances IoT service usability, allowing it to get the most out of any technology. As stated in the survey paper Sadri et al. (2021), data storage is one of the research topics which needs to be explored more in terms to achieve the performance of the time-sensitive IoT applications and distribute the data with low latency for the large datasets. Various methods are described in the survey paper such as data caching, data locality, data placement, data replication, data sharing, and data offloading. The reason for selecting the data locality topic is that we need to control which device is holding the data and which device needs to process the data at a given time. As mentioned in the survey paper Sadri et al. (2021), data distribution is still an open challenge in fog computing and needs to enhance the distributed storage for data algorithms to improve the performance of the time-sensitive IoT applications.

[2]`https://www.openfogconsortium.org`

2.2 Community Mapping in Fog Computing

One of the most important jobs is managing the fog nodes in a distributed network and allocating services from IoT devices to the nearest fog nodes. Many research projects have used the concept of breaking down a complicated network of nodes into smaller pieces. Traditional cloud approaches must be adjusted to support fog computing when moving cloud infrastructure to the edge to deliver low latency and location-based Internet services. Fog-specific resource management challenges differ from traditional cloud resource management problems due to the concept of user location and accessibility. In this case, the choice of virtualized resources is made based on the existing connections to network Access Points (APs) in order to achieve the lowest feasible latency.

As a result, fog management is highly reliant on the location of the fog. Filiposka et al. (2018) have proposed a solution for fog resource management based on the graphical and virtual communities to solve this issue. Communities, according to the author, are the whole network of fog devices, with the lowest atomic layer consisting of smaller communities that may be split into hierarchical dendrograms and connected to one or more co-located wireless APs that cover one specific cell. The research findings of Filiposka et al. (2018) demonstrate that the technique provides for highly optimal solutions for both virtual resource placement and relocation difficulties that follow the wireless user's mobility pattern, but they do not account for intricate fog scenarios with several VMs.

The concept of community mapping was dividing the many research projects that have used it as breaking down a complicated network of nodes into smaller pieces. To the extension of the above work, Lera et al. (2019) has presented a way for creating a fog community for application mapping in order to build device partitions. The suggested policy maps the linked services to the community of fog devices to assure the availability of apps to users. The approach utilized in this case study is to remove the edges from the original path while applying the Newman and Grivan Newman and Girvan (2004) algorithm to find the community. After assigning the application to fog communities and subsequent application service placement on the fog communities, as result, the aforementioned technique for application mapping enhanced service availability and quality of service (QoS), however, the experiments were not assessed with the complicated fog-based environment.

As stated earlier, fog nodes provide the storage and computing capabilities, additionally, these nodes are closer to the IoT consumers, they delegate services to lower the application network latency. But one of the challenging tasks is to manage the services among a large range of fog devices. To address this problem, Guerrero et al. (2018) has proposed a method for dividing fog colonies based on the centrality indices of a complicated network. They state that the recommended method will choose the fog node with the greatest values, implying the node's importance. The nodes that remain are subordinated to the controllers closest to them. In addition, as an application makespan indication, they utilized the network distance. The findings indicated that the smallest network distances were produced using the Betweenness centrality index and Barabasi-Albert network topology, on the other hand, this research does not split the fog colonies dynamically and does not examine one of the indications for fog colony resource capabilities.

Skarlat et al. (2017) has said that resource provisioning techniques to allow the utilization of fog-based computational resources are lacking, despite the theoretical underpinnings of fog computing being in place, by continuing the resource management key phrase. To solve this flaw, the author has created a fog computing infrastructure architecture based on fog colonies. The fog colonies are used to execute IoT programs that are made up of several resources, such as a distributed data flow (DDF). As a result, the authors employed the fog colony idea to coordinate the fog nodes and assign appropriate services. The authors assessed the Quality of Service (QoS), cost, and compared the outcomes to the typical cloud-centric strategy through several tests. The results demonstrate that using the evolutionary algorithm reduces network latency, and using the precise optimization approach improves fog resource usage, however, this study does not include resource dependability, cost, or availability of services in relation to the resource provisioning model.

2.3 Communication between fog nodes and sensors

One of the main aims for introducing fog computing is to reduce the latency and response time in real-time applications such as healthcare in which data is generated with a huge amount, needs to be processed in less amount of time, and reached the destination without a packet loss. In terms of fog for IoT includes several strategies for appropriate communication between fog nodes and IoT devices. Further, the peer-to-peer connection at the fog layer is one of the important core functionality in fog computing (Bellavista et al. (2019)). For peer-to-peer, cloud-to-fog, fog-to-IoT, and vice versa, these communications for data transfer need to be established while considering the application type and to achieve the low response time. One of the popular solutions to establish communication between these is with the use of message passing queue services. Ionescu (2015) states the difference between two message queue service such as RabbitMQ and ActiveMQ and their performance over various experiments. In the result, this study has stated that ActiveMQ is faster when a client sends the message to the broker, on the other hand, RabbitMQ is faster when the client receives the messages from the broker.

With the experimental analysis, Chueshev et al. (2018) has designed the cloud robotic platform with the help of fog computing approach and they have used the RabbitMQ as a message-passing queue service for P2P network with the containerization infrastructure with microservices. In their experiments, it has been shown that they have calculated and compared the latencies between Kafka and RabbitMQ for the designed model with different sizes of messages. As a result, studies have shown that RabbitMQ shows low latency and positive marks with multiple tests.

2.4 Effect of Data Locality in Fog Computing

While handling the data in a fog environment, it is necessary to keep data consistency. To ensure efficiency, fog computing platforms need to be located near to end devices and respective computing nodes are assigned as per the capabilities. Peer-to-peer middle-wares are the ones used for fog computing environments but they are missing key components such as data storage control and who will be in charge of computing activities. With the use of the traditional approach, data is distributed and replicated within the network

haphazardly with the use of hashing functions which lack the details about data locality (Steffenel (2018)).

Because data might be stored apart from object copies, the traditional technique of using Distributed Hash Tables to locate data is insufficient. Confais et al. (2019) presented a tree-based technique for distributing records that saves the location of entity copies to address this issue. The recommended technique is organized hierarchically, similar to the DNS (Domain Name System) protocol, but it finds the position of a duplicate of an entity from the bottom up. The researchers created a tree from the actual network architecture using a modified version of Dijkstra's algorithm, the tree is calculated incrementally unlike the normal form were considering each node as a source, and the node with the lowest weight is picked as the "optimal" root node. According to the findings of this research, locating an entity using the tree partition approach was faster than utilizing common protocols such as Distributed Hash Tables (DHTs).

Using the notion of data locality, Stavrinides and Karatza (2020) states that in order to develop effective load balancing and scheduling methods, it is necessary to have a deeper knowledge of how the data locality of the input data impacts the output of particular platforms. The author of this paper assessed the efficiency of a heterogeneous fog system in which a large number of data-intensive jobs came in real-time and analyzed the impact of data localization of workflow input tasks on the system's performance. The technique used in this study revealed that data locality has a negative influence on the deadline miss ratio, i.e., as the quantity of tasks increases, jobs are unable to finish them on time, resulting in more job losses.

The below table 1 depicts the summary of all the previous studies. It states the various methods that have been used by different studies and their limitations.

Reference	Main Content	Research Method	Research Results	Limitation
Filiposka et al. (2018)	Management of fog resources	Design virtual and graphical, hierarchical communities	Optimal solution for virtual resources	Not accounted intricate fog scenarios
Lera et al. (2019)	Assignment of applications	Creation of fog communities based on device partitioning	Enhanced service availability	No evaluation with compact scenarios
Guerrero et al. (2018)	Service management between fog nodes	Divide fog colonies using centrality indices	Smaller Network Distances	No dynamic approach for fog colony partition
Skarlat et al. (2017)	Provisioning of resources	Fog computing design based of fog colonies	Decrease in network latency	Does not consider cost/availability of service
Ionescu (2015)	Difference between RabbitMQ and ActiveMQ	NA	ActiveMQ faster to send message, RabbitMQ faster to receive message	NA
Chueshev et al. (2018)	Cloud Robotic Design	Design based on fog and containerization approach of apps	Difference between RabbitMQ and Kafka for latency	Not evaluated complex tests
Steffenel (2018)	Data transfer	Community Mapping	Imrpoved the data locality	Not implemented with real scenarios
Confais et al. (2019)	Handling data locality	Tree-based approach for dividing records	Decrease in time in locating entity	No consideration of dynamic approach
Stavrinides and Karatza (2020)	Affect of data location	Analysis of influence on input workloads	Negative affect on the deadline miss ratio	No consideration of 3-tier architecture

Table 1: Summary of Literature Review

3 Methodology

This section describes the various methodologies, algorithms, and libraries that are used to design the community and communication between different components. This study is divided into different parts for understanding the requirements and analyzing the results. The following parts have been discussed further in this section:

1. For data dissemination between fog nodes and sensors, cloud servers, and fog nodes and peer-to-peer (fog-to-fog) communication has been achieved using the message

passing broker service named as RabbitMQ[3].

2. To handle the sub-tasks between the community, manager node, or control node has been selected using random selection method.

3. For community creation, the application owner needs to provide the instructions with the list of fog nodes to complete the job.

3.1 Communication between fog nodes-sensors, peer-to-peer(fog-to-fog) using RabbitMQ

While creating the community of fog nodes, the application owner needs to provide the instructions. When the application owner mentions the list of required sensors to perform some tasks, communication between fog nodes and sensors needs to be established with the use of lightweight protocols. For this purpose, open-source publish/subscribe message broker RabbitMQ(Ionescu (2015)) is used. It mainly focuses on passing the messages between sensors and fog nodes. Additionally, it transfers the data between different components asynchronously through the work queue. It is faster when the client receives a message from the broker. It is ideal to use when the IoT applications need to receive the required data from the fog nodes after its processing and also it can receive multiple messages. Additionally, RabbitMQ offers easy deployment in the cloud and on-premise as well. The main motivation behind using the RabbitMQ is that it supports the distributed architecture and federated configurations to achieve the scalability and availability of the applications.

In the algorithm 'hazeSpearStarter', all the instances of fog nodes, sensors, and cloud servers have communicated with each other through the RabbitMQ channel. Once the fog nodes and sensors are registered with the cloud server, fog nodes are assigned to the sensors randomly, and the respective data channel is emulated with the use of RabbitMQ. At this point, sensors transfer the generated data to the nearest fog node for computation and storage purposes. Once the decision has been determined, the fog node passes the data to the sensors by the RabbitMQ channel.

Once the communication is established between sensors and fog nodes for data sharing as shown in 1, sensors transfer the data to respective fog nodes for computation. For each community, the manager node has been selected randomly. Each manager node is responsible to manage and monitor the sub-nodes of the community. For replicating the data among a community as well as manager node communication, peer-to-peer synchronization is required between them. It is achieved in the proposed method with the use of the message passing queue service 'RabbitMQ' used in Chueshev et al. (2018). To establish peer-to-peer synchronization, we have used the proposed methodology. Due to peer to peer connection, fog nodes will be replicating the data between their community for data accessibility and high availability. When the sensors request any data, the community manager will communicate with each sub-node to look for the required data.

3.2 Random selection of community manager

Once the community is generated, the manager node for each community is selected randomly. Users can send requests to manager nodes (Skarlat et al. (2017)) for the im-

[3] `https://www.npmjs.com/package/amqplib`

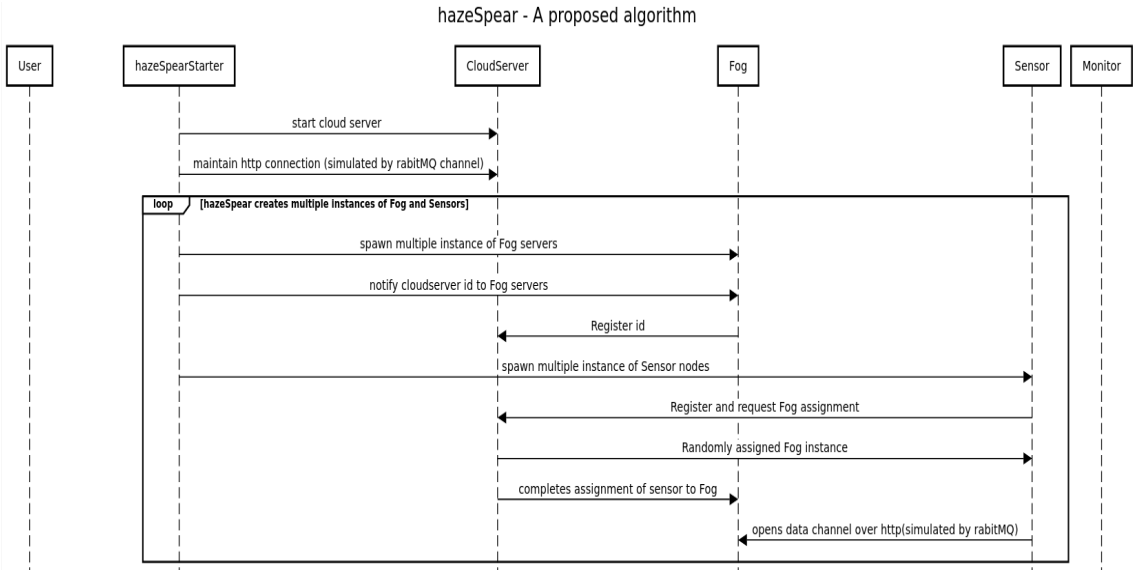

Figure 1: Sequence diagram for communication between fog nodes and sensors

plementation of IoT applications. When the present fog node community is unable to handle service requests, or data access requests the manager node can forward them to the cloud-fog control middleware or to other fog communities through their fog orchestration manager nodes. Manager nodes are responsible for various tasks such as making modifications in the infrastructure of the community, monitoring fog nodes, and looking for a requested data or service in the fog node of a community. By selecting the manager node for each community, it will be easy for data access requests to travel within a community because the manager node would be aware of the data which is requested. On the other hand, without a manager node, data access requests need to travel for multiple hops to look for a requested data or a service.

4 Design Specification

This section specifies the implemented algorithm and its architecture.

4.1 Architecture

As stated earlier, fog-based environments are made up of a large number of fog nodes that are distributed all over the network. As illustrated in Figure 2, the implemented algorithm constructs the community in a static or dynamic way and which would be a subset of fog nodes. A community is nothing more than a collection of related systems or nodes that are linked and communicate with one another. As shown in Figure 2, every IoT device is connected to different fog nodes and transferred the generated data to fog nodes. For time-sensitive applications, data is stored and processed at the fog layer and low priority data is transferred to cloud data centers for archival which would be occasional data transfer. Once data is reached at one of the fog nodes, it creates replicas within a community, and a data synchronization is a place for data retrieval and data consistency.

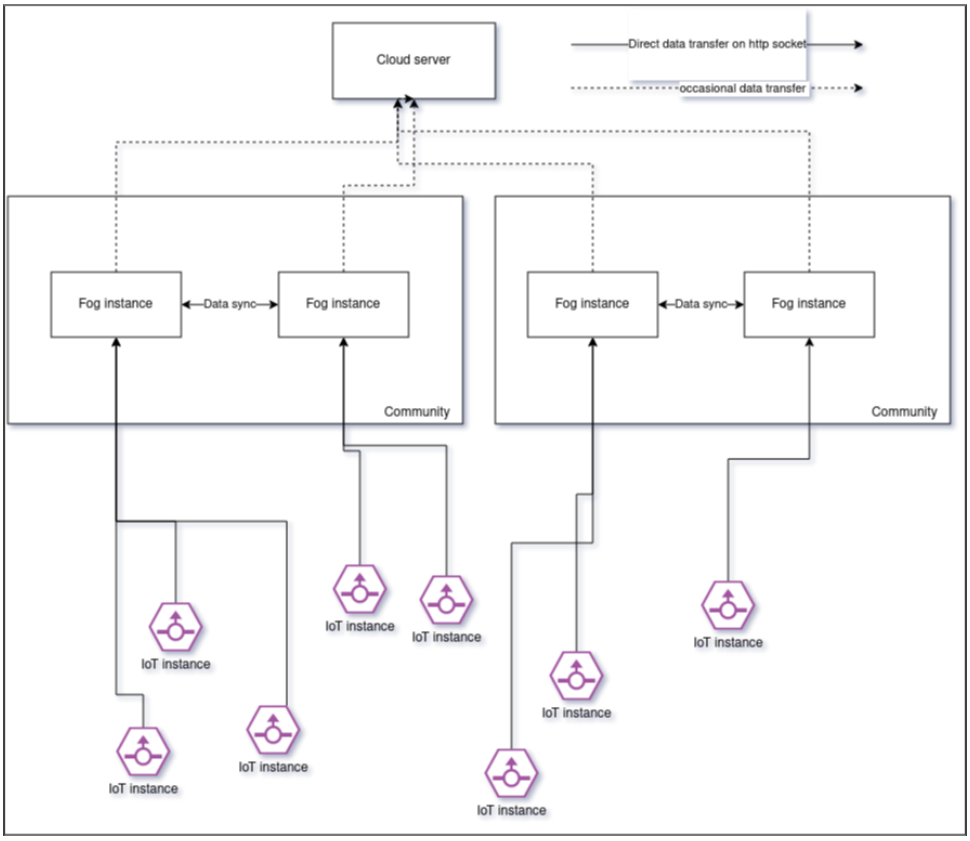

Figure 2: Architecture of implemented algorithm

4.2 Workflow of Implemented Algorithm

Various communities fulfill certain criteria such as node capabilities, security, and placement in both ways such as static and dynamic. Nodes can also be members of one or more communities. The following are two methods for forming a community:

- Static: In this case, a community is formed based on application needs and leveraging the network's prior knowledge to execute certain tasks. Furthermore, in order to maintain the proximity service, this sort of community is made up of nodes that span a specific region. Also, the computational performance of these nodes is similar (CPU type, RAM, etc.).

- Dynamic: Different criteria are examined in this scenario, such as the node's state and context, as well as if the nodes are entering or departing the network, to establish a transient community. The suggested abstract function allows the fog application to define data needs at runtime and then builds a temporary community of fog nodes that participate.

The implemented algorithm for community generation is as shown in Figure 3 and explains the working as following: The algorithm focuses on creating the community of fog nodes on the basis of the requirement and data transfer between fog nodes and sensors. Initially, once all the fog nodes are selected, a community manager or control node for each community has been selected to control the various operations such as transferring the data between communities, checking which node has the required data, and also making changes in the infrastructure of the community such as adding or removing the fog

Algorithm 1 Community Generator

```
1: Input: available fog nodes
2: Output: Generate community of nodes
3: GenerateCommunity(fog_nodes)
4: if fog_nodes.size less than 2 then Return
5: end if
6: Comm_manager = randomly_selected(fog_nodes)
7: Comm_hash = community_hash_generator(fog_nodes)
8: resources = Initialize empty_array
9: for each node in fog_nodes do
10:     if node is Comm_manager then Continue
11:     end if
12:     node.hash_node = sha2(node)
13:     resources.add(node.resources)
14: end for
```

Figure 3: Algorithm for Community Mapping

node from the community. Further, each community has been assigned to a unique hash key for identification and to avoid creating duplicate fog nodes community. In the end, a community is created with multiple fog nodes and each fog node has been assigned to various sensors for data transfer and replicating the data between communities.

5 Implementation

The implementation of the proposed algorithm has been completed with the sequence as illustrated in Figure 4. The algorithm 'hazespear' starts connection between cloud server, fog nodes, and sensors with the use of RabbitMQ as stated earlier. Once all the required fog nodes are collected, the algorithm notifies the fog nodes with the available cloud server-id. This step is required whenever the application is in need to store their archival data or respective task is considered as a non-priority task. Further, all the fog nodes register themselves to the cloud server. In the next step, the algorithm collects the instances of sensor nodes and sensors register themselves to fog nodes. Once registration is completed, sensors have been assigned randomly to several fog nodes. In the end, all the data channels have been opened via RabbitMQ and sensors transfer their data to fog nodes.

When a job has started to perform some task, the algorithm provides the configurations of job to cloud server and further cloud server informs fog nodes about the task or a job. This algorithm has implemented both scenarios such as with and without a community of fog nodes. The implementation difference between these two scenarios is, whenever the data is generated through sensors and transferred to fog nodes, data gets stored over the network haphazardly where there is no control over the fog nodes and where exactly data should store. With the use of community mapping, data is storing within a covered area and replicating the data within a particular community. Fog nodes do not need to ask for missing data to cloud servers because required data is stored within a community via data synchronization.

The below sub-sections explain how the fog environment is created and the connection is

established between cloud servers, fog nodes, and sensors pragmatically.

5.1 Configurations for devices

To define the topology and evaluate the parameter metrics, different configurations are required. The simulation configurations are mentioned in the JSON format where several fog nodes and sensors, RabbitMQ connection URL are added. While taking experiments, configurations can be modified as per the requirements.

5.2 Device Setup

While implementing this research study, the following devices; fog nodes, IoT devices(Sensors), and Cloud Server were created. They are defined using TypeScript language. Each device is created with its respective functionalities. RabbitMQ queues are started further for each device to establish the connection between them. Further, sensors and fog devices are registered to the cloud server. Additionally, each fog device has been assigned with a unique id with the use of 'uuid' for identification. Once the data channels are opened through the queue, IoT devices are ready to send the data to the fog devices and sensors are assigned to several fog nodes randomly.

5.3 Community Creation

Communities of fog nodes are created with two way i.e. static and dynamic approach. When selection type of devices is random, communities are created dynamically and if it is specified then communities are created statically. Firstly, cloud server is started with the spawning the list of available sensors and their governing fog nodes which are mentioned in configurations. Out of available sensors, algorithm have selected the sensors randomly. Further, it has looked for the randomly selected fog nodes out of the available fog nodes to form a community. In the end, once all nodes are elected, community id is generated and notified the fog nodes to start the execution of application and started the data synchronization within a newly created community.

5.4 Application - Calculate Temperature

The application is created in such a way that it is calculating the temperature values from a sensor for every 2 minutes and takes the average of the all the sensor values. The delay of 20 millisecond is created between 2 sensor values.

5.5 Data Synchronization between communities

Once the community has created, application has started to execute on fog nodes and generated temperature values, it sent the sensor data to fog nodes. Thus, the one of the objective is to replicate the same data in a newly generated community. The data synchronization is performed via the work queue where message is published in a community and each node will consume the same message. Then, algorithm has checked the respective community id. While executing the application with community id, program has checked whether data sync is started or not. Next, data sync is started and send all the data to all fog nodes who all have the same message.

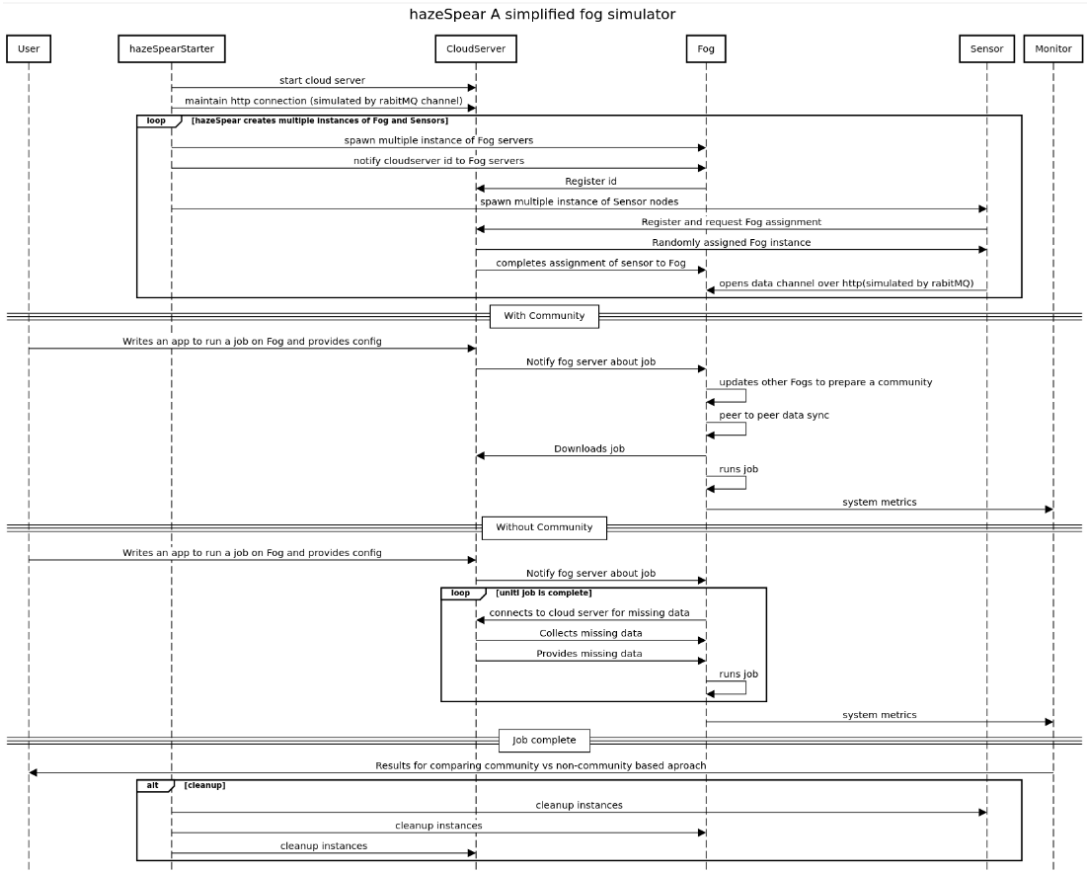

Figure 4: Sequence diagram of proposed algorithm

6 Evaluation

This section discusses the results and the analysis of the experiments which have been conducted on the proposed algorithm and traditional algorithm.

As shown in Figure 4, if the community mapping is in place at the fog layer, data has stored within a number of fog nodes as well as data has replicated through data synchronization in a community. On the other hand, without community mapping, the fog node has to communicate with the cloud server and look for the missing data because fog nodes do not have enough storage capacity or computation power as compared to cloud servers.

This research study has taken multiple experiments for evaluation and calculation of response time to run the application. As per the configurations shown below in Table 2, we have taken the total number of fog nodes, participating Sensors and their fog nodes, and total sensors as parameters and calculated the time taken to run the application several times.

Config No	No of Fog Nodes	Total Sensors	Participating Sensors
1	20	60	10
2	40	80	30
3	40	130	80
4	10	80	40

Table 2: Configurations for experiments

6.1 Experiment 1

In experiment 1, selected configurations are as per the table above 2. As shown in Figure 5, we can state that less time has taken when the number of participating devices are less. When the number of application runs is increasing, the total execution time is increasing ultimately in both scenarios. Overall, the response time is enhanced and improved by approximately 76% with the use of community mapping as compared to without community mapping.

Figure 5: Experiment 1

6.2 Experiment 2

In experiment 2, systems are configured as stated in Table 2 for Config.2. As illustrated in Figure 6, it is clear that when the number of devices has increased and community mapping is not in place, the latency has increased to the highest as compared to with community mapping approach. Additionally, when the number of counters (the number of times the application will run) is increased, execution time is reduced by 79% overall with the community mapping approach.

Figure 6: Experiment 2

6.3 Experiment 3

In experiment 3, several devices are selected as stated in Table 2. Also, we have changed the behavior of the fog node were added the upper and lower bound for internet delay limit in configurations. Even after changing the behavior of devices and their configurations, the traditional approach is taking more than 21 seconds in each iteration as depicted in Figure 7. The experimental result has shown roughly 89% of improvement in response time for application execution with the community mapping approach in comparison to the conventional method.

Figure 7: Experiment 3

6.4 Experiment 4

In experiment 4, when the number of fog nodes is fewer, and the number of participating nodes is more than half of the fog nodes, the results are showing the same output as shown in previous experiments. With the use of traditional behavior, latency has increased exponentially in contrast to community mapping and is not a suitable approach for time-sensitive applications. From Figure 8, it is clear that the response time is enhanced by about 75% if the community mapping approach is in place.

Figure 8: Experiment 4

6.5 Discussion

This work helps to address the time-sensitive latency problems through the creation of community of fog nodes and the replication of data throughout the community. Moreover, the location of the data has been enhanced by synchronizing and replicating needed data within a community. The aforementioned testing findings and analysis showed that, even if the device configuration changes, the community mapping method has helped to achieve latency. The findings of figures 5, 6, 7 and 8 have demonstrated that, in contrast to conventional approaches, the suggested algorithm is trustworthy by decreasing overall delay. The RabbitMQ implementation has shown to be a dependable open source message queue service, which enabled to create data transmission and connectivity. The method has overall helped improve the latency and response time for the execution of IoT applications by around 79%. In addition, by addressing the data miss ratio for community mapping, the outcomes may be enhanced. With the present technique deployed, few communities lack the data to replicate inside a community while conducting synchronization. Overall, experimental findings are evaluated and displayed utilizing community-based mapping and finally addressed in order to reach the data locality. The implementation has assisted in the fog-enabled IoT environment to fulfill the main objective.

7 Conclusion and Future Work

A community mapping technique is proposed in this research study to optimize the locality of the data and reduce the response time in the fog-based environment. Fog node communities with participating IoT devices are established on the fog layer. Simulating devices and their settings implement the suggested method. Additionally, an IoT application is developed which has helped to generate the data.

In general, the technique used has contributed substantially to improving the data location and execution time by establishing fog node communities. The test results have revealed that the time taken using the traditional technique is significantly longer than the community mapping methodology with multiple rounds. It is also noted that the time needed for calculating the program to execute significantly decreases system performance when the number of participating devices is increased. In addition, the data miss rates in communities is affected.

In future, this technique can be deployed on the fog-compatible infrastructure in real-time applications with improved device versions. Also, to achieve 100% data consistency between communities, NoSQL scalable database can be used which ultimately helps to accomplish the data locality.

References

Bellavista, P., Berrocal, J., Corradi, A., Das, S. K., Foschini, L. and Zanni, A. (2019). A survey on fog computing for the internet of things, *Pervasive and Mobile Computing* **52**: 71–99.
URL: *https://www.sciencedirect.com/science/article/pii/S1574119218301111*

Chueshev, A., Melekhova, O. and Meshcheryakov, R. (2018). Cloud robotic platform on basis of fog computing approach, *in* A. Ronzhin, G. Rigoll and R. Meshcheryakov (eds), *Interactive Collaborative Robotics*, Springer International Publishing, Cham, pp. 34–43.

Confais, B., Parrein, B. and Lebre, A. (2019). Data location management protocol for object stores in a fog computing infrastructure, *IEEE Transactions on Network and Service Management* **16**(4): 1624–1637.

Filiposka, S., Mishev, A. and Gilly, K. (2018). Community-based allocation and migration strategies for fog computing, *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, Barcelona, pp. 1–6.

Guerrero, C., Lera, I. and Juiz, C. (2018). On the influence of fog colonies partitioning in fog application makespan, *2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud)*, Barcelona, pp. 377–384.

Hu, P., Dhelim, S., Ning, H. and Qiu, T. (2017). Survey on fog computing: architecture, key technologies, applications and open issues, *Journal of Network and Computer Applications* **98**: 27–42. JCR Impact Factor: 5.570(2019).
URL: *https://www.sciencedirect.com/science/article/pii/S1084804517302953*

Ionescu, V. M. (2015). The analysis of the performance of rabbitmq and activemq, *2015 14th RoEduNet International Conference - Networking in Education and Research (RoEduNet NER)*, pp. 132–137.

Lera, I., Guerrero, C. and Juiz, C. (2019). Availability-aware service placement policy in fog computing based on graph partitions, *IEEE Internet of Things Journal* **6**(2): 3641–3651.

Newman, M. E. J. and Girvan, M. (2004). Finding and evaluating community structure in networks, *PHYSICAL REVIEW E* **69**: 026113.
URL: *https://link.aps.org/doi/10.1103/PhysRevE.69.026113*

Sadri, A. A., Rahmani, A. M., Saberikamarposhti, M. and Hosseinzadeh, M. (2021). Fog data management: A vision, challenges, and future directions, *Journal of Network and Computer Applications* **174**: 102882. JCR Impact Factor : 5.570(2019).
URL: *https://www.sciencedirect.com/science/article/pii/S1084804520303465*

Skarlat, O., Nardelli, M., Schulte, S., Borkowski, M. and Leitner, P. (2017). Optimized iot service placement in the fog, *Service Oriented Computing and Applications* **11**: 427–443.

Stavrinides, G. L. and Karatza, H. D. (2020). Orchestration of real-time workflows with varying input data locality in a heterogeneous fog environment, *2020 Fifth International Conference on Fog and Mobile Edge Computing (FMEC)*, Paris, pp. 202–209.

Steffenel, L. A. (2018). Improving the performance of fog computing through the use of data locality, *2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, Lyon, pp. 217–224.

Vaquero, L. M. and Rodero-Merino, L. (2014). Finding your way in the fog: Towards a comprehensive definition of fog computing, *SIGCOMM Comput. Commun. Rev.* **44**(5): 27–32.
URL: *https://doi.org/10.1145/2677046.2677052*