I, Satyajit Bhowmick, hereby submit this original work as part of the requirements for the degree of Master of Science in Computer Science.

It is entitled:
A Fog-based Cloud Paradigm for Time-Sensitive Applications

Student's name: Satyajit Bhowmick

This work and its defense approved by:

Committee chair: Paul Talaga, Ph.D.

Committee member: Karen Davis, Ph.D.

Committee member: Nan Niu, Ph.D.
A Fog-based Cloud Paradigm for Time-Sensitive Applications

A thesis submitted to the Division of Graduate Studies and Research of the University of Cincinnati in partial fulfillment of the requirements for the degree of MASTER OF SCIENCE in the Department of Electrical Engineering and Computing Systems of the College of Engineering and Applied Science

June 8, 2016
by SATYAJIT BHOWMICK Bachelor of Technology West Bengal University of Technology, Kolkata, India, 2013

Thesis Advisor and Committee Chair: Dr. Paul Talaga
Abstract

Fog computing is a recently proposed computing paradigm that extends cloud computing and services to the edge of the network which is the entry point to the core network (e.g., router is located at the edge of a network). The new features offered by fog computing (e.g., storage, distributed analytics and intelligence at the edge of the network), if successfully applied for time-sensitive applications, has great potential to accelerate the discovery of early notification of emergency situations to support smart decision making. While promising, how to design and develop real-world fog computing-based data monitoring systems is still an open question. As a first step to answer this question, in this research, we employ a fog-based cloud paradigm for time-sensitive applications and show the practical applicability and significance of such a system. The ubiquitous deployment of mobile and sensor devices is creating a new environment, namely the Internet of Things (IoT), which enables a wide range of future Internet applications. In this work, we present dynamic fog, a high level programming model for time-sensitive applications that are geospatially distributed, large-scale, and latency-sensitive. We also analyze use cases for the fog model with real-time healthcare data. Our experiments show that our proposed system achieves minimum delay compared to fogless systems, while it also achieves the data accuracy and data consistency which are very important in many applications like medical applications.
Acknowledgement

First and foremost, I would like to express my deepest gratitude to my advisor, Dr. Paul Talaga, for his continuous support and guidance. Dr. Talaga’s patience, motivation, enthusiasm and immense knowledge helped me to overcome many critical situations and added considerably to my graduate experience. I simply could not wish for a better and friendlier advisor. Thank to him for all his time helping me to prepare my thesis.

I would also like to thank my family and friends for the support they have provided me through my entire life. Especially, nothing would have been possible without the unconditional love and patience of my parents and my sister. I owe them all my success.
Contents

List of Tables viii
List of Figures ix

1. Introduction 1
   1.1 Internet of Things ................................................. 1
   1.2 Cloud Computing .................................................. 3
   1.3 What is Fog Computing? .......................................... 4
   1.4 Why Fog Computing? ............................................. 5
      1.4.1 Drawbacks of Traditional Cloud Architecture ............... 5
      1.4.2 Needs for Fog Computing .................................... 6
   1.5 Challenges Ahead for Fog Computing ........................... 7
   1.6 Fog Applications .................................................. 9
   1.7 Overview of the Thesis ........................................... 9

2. Background and Related Work 10
   2.1 Fog Concept ........................................................ 10
   2.2 Fog Computing for Delay-sensitive Applications ............... 11

3. Our Idea 14

4. Architecture 16
   4.1 Our Proposed Fog-based Cloud Paradigm ........................ 16
   4.2 Architectural Explanation ........................................ 16

5. Implementation 19
   5.1 Implementation of Symmetric Structure ........................ 19
   5.2 Implementation Details of Architecture ........................ 20
   5.3 Implementation of File Transmission ............................ 24
   5.4 Software Implementation Details ............................... 24
B.2 Data Transmission and Node Failure ........................................... 69
B.3 Data Flushing ................................................................. 70
B.4 Data Access and Visualization at Client .................................. 72
List of Tables

6.2.3.1 Ping time for data centers .................................................. 41
6.2.3.2 WANem delay input ........................................................... 42
6.4.1 Data center distance and corresponding Round Trip Time .......... 45
6.4.2 Comparison between RTTs of Fog and Cloud ............................... 46
List of Figures

1.2.1 IoT devices interacting with Cloud via Internet ................................. 4
1.2.2 Typical Fog Computing Setup .......................................................... 5
4.2.1 Proposed FOG based cloud architecture ........................................... 17
5.1.1 One Symmetric Structure ................................................................. 20
5.2.1 Simplified Implemented Architecture .............................................. 21
5.5.2.1 Flowchart for Node Failure Handling Mechanism ......................... 28
5.6.2.1 Flowchart for Accuracy check and Data Flushing Mechanism ......... 34
6.1.1 Case 1- Data Transmission Path ....................................................... 37
6.1.2 Case 2- Data Transmission Path ....................................................... 38
6.1.3 Case 3- Data Transmission Path ....................................................... 39
6.3.1 Delay graph ................................................................................. 44
6.5.1 Node failure test case ................................................................... 48
Chapter 1: Introduction

In recent years, there has been a revolution in connectivity with the rapidly increasing number of wireless sensor networks, healthcare services, smart phones and other pervasive real-time monitoring systems. The advent of Internet of Things (IoT) [1] has triggered a ubiquitous connection among people, devices and services almost all the time. This ubiquity is causing a high velocity and high volume of data. Researchers have proposed integration between IoTs and cloud computing for storing and analyzing this high volume data [2][3]. Cloud computing frees the end users and businesses from the specification of maintaining and managing the resources. However, cloud computing becomes a problem for delay-sensitive applications where real-time requirements play an important role in the deployment model. In this research, we discuss the current constraints of using simple cloud systems and propose a better network and system architecture for time-sensitive applications – commonly known as the Fog paradigm [11].

1.1. Internet of Things

Internet of Things (IoT) is the network of physical objects – embedded with sensors, software and network connectivity that enable these objects to collect and exchange data. The term ‘things’ in IoT refers to any object on earth with a network connected sensor attached to it, whether it is a smart communicating device or a dumb object [4] like household item, food item, medicine, etc. that can be a part of the Internet. In an IoT network, these objects, through data communication means, become communicating nodes over the internet. The objective of IoT is to provide a network infrastructure with
interoperable software and internet protocols to allow integration and interaction of physical as well as virtual devices and dumb objects [5].

Machine-to-Machine (M2M) communication is considered the backbone of IoT. In M2M, communication between two machines takes place without human intervention. Many M2M applications will need to deliver and process information in real time, or near-real-time, and many nodes will need to be extremely low powered or self-powered (e.g. solar powered). The key point is that the ‘Things’ in IoT or ‘Machines’ in M2M are increasingly cheap, plentiful, and can communicate either directly with the internet or with internet connected devices. All IoT sensors require some means of relaying data to the outside world. There’s a plethora of short-range, local area, or wireless technologies available, including: RFID, NFC, Wi-Fi, Bluetooth (including Bluetooth Low Energy), XBee, Zigbee, Z-Wave and Wireless M-Bus. There’s no shortage of wired links either, including Ethernet, HomePlug, HomePNA, HomeGrid/G.hn and LonWorks. For long range, or wide-area links, there are existing mobile networks (using GSM, GPRS, 3G, LTE or WiMAX for example) and satellite connections. M2M and the Internet of Things have huge potential, but currently comprise a heterogeneous collection of established and emerging, often competing, technologies and standards.
1.2. Cloud Computing

Cloud computing, a “pay-as-you-go” computing model, takes computing from the desktop to the World Wide Web, and frees the end user from the specification of maintenance and managing the resources. It is an effective alternative to owning and managing private data centers for batch processing and customer facing Web applications. Cloud computing can be considered an extended form of distributed, parallel, and grid computing [6][7][8][9]. Cloud computing enables convenient, on-demand network use of a shared pool of configurable computing resources and provides ubiquitous access to the content. Cloud computing simplifies the client’s computing jobs by removing the hassle of keeping large storage and computing devices.

Cloud systems are located within the internet, which is heterogeneous in nature. Because of the loosely controlled and non-homogeneous nature of the internet there are many unresolved issues related to the cloud. One such issue is network delay for real time applications. Users could be badly affected by delay and delay jitter caused by the latency in networks. The other major issue confronted with cloud systems is privacy and security. Cloud systems support better operational efficiency, but come with greater risks of data theft attacks, especially insider data theft in data centers [10]. There exists a risk that the integrity and confidentiality of the customer data may be compromised. However, in our research we are more interested in network delay rather than the security and we have not explicitly worked on the security issues. Figure 1.2.1 shows interaction of IoTs with cloud via Internet.
1.3. What is Fog Computing?

The term “Fog Computing” was introduced by Cisco Systems [11] as a new model that could ease wireless data transfer to distributed devices in the Internet of Things (IoT) network paradigm. Fog computing is a new platform that extends the cloud computing paradigm to the edge of the network [12] that is the entry point to the core network. Although both paradigms provide a similar set of services in terms of computation, storage etc.; fog provides an additional advantage due to its proximity to the customer; dense geographic coverage and mobility support [13]. The fundamental element of the fog computing architecture is called a ‘fog node’, which is an extra layer added between an IoT device/data generating node and a remote cloud server. These fog nodes are used in order to speed up time-critical applications. Services are hosted, away from the cloud, closer to the devices, at the fog node devices such as routers, set-up-boxes or access points etc. Data generator nodes, which are typically sensor nodes, generate data.
from these data generator nodes go to the cloud via intermediate fog node(s) and the user, instead of accessing the data from the cloud, can access the data from the fog node. Hence, this reduces service latency, and improves Quality of Service (QoS), resulting in a superior user experience. Therefore, the fog model is a well-suited paradigm for real time data analytics. Figure 1.2.2 shows a typical fog computing structure.

![Figure 1.2.2: Typical Fog Computing Setup [19]](image)

1.4. Why Fog Computing?

1.4.1. Drawbacks of Traditional Cloud Architecture

Cloud computing provides benefits such as low-cost services, large storage, and frees the user from the burden of maintaining large data centers. Therefore, in many data monitoring systems, remote cloud servers are used for storing and processing data collected from a large number of sensor nodes. However, this system has many existing challenges for delay sensitive applications. These are as follows:

1) **Location of Data Storage** - Location matters for critical and delay-sensitive applications. In order to reduce the operating costs, cloud service providers prefer remote areas with low-cost resources to build their data centers. Therefore, this increases the data transmission delay. Satyanarayanan, et al. [14] show that WAN latencies, that can be high and interfere with interactive applications. Hong, et al. [15] show that the cloud
can incur significant amount of delay due to the distance of the cloud from the user. Therefore, it can be vulnerable for latency-sensitive applications. Currently MediaBroker [16], for live sensor stream analysis, and cloudlets [17], for interactive applications, are the two systems that can provide resources for computing at the edge of the network. However, neither of these currently support widely distributed geospatial applications. In short, communication through a distant cloud is not feasible for mission critical and delay sensitive services that require quick response and processing.

2) **Cloud Server Overloading** - Cisco [18] portrays 50 billion IoT devices will be connected to the Internet by 2020. As the amount of data and data generator nodes keep increasing, it results in an increasing burden on the cloud as the data is getting stored on the cloud. Thus the cloud computational effort also increases because the cloud servers get overloaded [19]. This also introduces significant latency. Therefore, a new architecture is needed to offload the cloud by reducing some of the tasks, e.g. storage or processing, usually done on the cloud.

1.4.2. **Needs for Fog Computing**

Fog computing builds upon the capabilities of cloud computing by extending them towards the edge of the network. Putting intelligence in the network, in contrast to the cloud, allows fog computing resources to perform low-latency processing, away from the cloud, near the edge, while latency-tolerant applications can still be performed in the core of the cloud. Fog computing does not replace cloud computing, but supplements the cloud for the most time-critical, delay-sensitive aspects of network operations. We need fog computing for delay-sensitive applications because of the following characteristics:
• **Low latency** – The edge location of fog nodes supports end points with rich services to reduce latency by storing and processing the data close to the source of information [31]. Data can be located at optimal depth in the network; hence it can reduce the transmission delay significantly.

• **Geographic Distribution** – Services and applications targeted by fog demand wide distribution of deployment. Contrary to the more centralized cloud, the wide geographic distribution of fog nodes often holds advantages compared to the cloud deployment [20].

• **Real-time Rapid Interaction** – Time-critical fog based applications need rapid and real-time interaction rather than batch processing. Real-time push notification [21] at smart gateways gives fog computing superiority over the cloud for mission critical and time-sensitive applications.

• **Intelligent end-points** – Fog computing, by providing intelligence to the fog nodes, tries to push some of the computation to the endpoint devices. Hence, highly capable and intelligent fog nodes provide more optimal network deployment [20]. This also reduces the burden on the cloud.

1.5. Challenges Ahead for Fog Computing

Although the current research efforts and user trends are pushing for fog computing, the path is far from being paved. There are many open challenges that must be addressed to make the fog more robust and fault-proof. Here we discuss some challenges ahead:

1. **Compute /storage limitation** – Fog nodes don’t have much storage /computation capacity. Although current trends are improving with more energy-efficient and more
powerful devices, still more improvements are to be made for more efficient operation and critical situation handling [32].

2. **Runtime System Implementation** -- One key challenge in fog research is to develop distributed runtime systems and migrate ‘Mobile Fog Processes’ across different devices in that system [15]. While doing so, providing reliability, security, and performance accuracy is a challenging task to accomplish.

3. **Network Management** -- Unless we can reap the benefits of applying Software Defined Networks and Network Function Virtualization techniques, network management will be a burden for fog computing [22] [23] [24]. However, seamless integration of these techniques in fog computing is not easy and will certainly be a challenging task.

4. **Discovery and Node Failure** -- An efficient node discovery algorithm that can adaptively find near-optimal placement of mobile fog processes is still an open question [15]. Dynamic resource discovery [25] and resource sharing [26] is critical for application performance in the fog. Still, new improvements are to be made in that domain.

5. **Security and Privacy** -- Security and privacy is regarded as one of the biggest challenges for fog computing [22]. Fog has the same security concerns that apply to current virtualized environments and should be properly investigated at every stage of fog computing platform design [27]. To overcome this, access control and intrusion detection system has to be applied, which need support from every layer of the platform.

In our research our goal is to design a fog-based prototype for time-sensitive IoT-based applications that reduces the data access delay, is capable of runtime operation, can effectively handle the storage limitation and does not overburden the fog nodes, and resilient to network/node failure situations.
1.6. Fog Applications

We have outlined a few scenarios that will benefit from the fog computing concept.

- **Connected Vehicles** – A connected vehicles application can be hugely benefited from fog computing. Hong, et al. [15] show the efficacy of mobile fog with such a traffic monitoring and vehicle tracking application.

- **Big Data Analysis in Smart Cities** - Fog computing fits the need of future smart cities where ubiquitous deployment of various kind of sensors requires a new computing paradigm. Tang, et al. [28] shows a hierarchical fog computing architecture for big data analysis in smart cities.

- **Healthcare Services** – The primary objective of fog computing is to ensure low and predictable delay in delay-sensitive applications. One such application where delay is a primary concern is healthcare services. Shi et al. [29] discuss the prospect fog computing can provide in healthcare services.

- **Software Defined Networks (SDN)** – SDN has emerged as a prospective solution to facilitate efficient deployment and management of network services. Truong et al. [30] proposed a novel SDN-based architecture supporting fog computing for both safety and non-safety services.

1.7. Overview of the Thesis

The rest of this thesis is organized as follows: Chapter 2 discusses the background and related works in this domain. Chapter 3 describes our idea and goal for the thesis. Chapter 4 describes our proposed architecture. Chapter 5 talks about implementation details. Chapter 6 describes the experiments that we have done during this research. Finally, in Chapter 7, we conclude our work followed by future work in Chapter 8.
Chapter 2: Background and Related Work

In this thesis, we have brought together different areas of research in the context of delay-sensitive fog computing applications. We provide background information about these areas and also highlight recent work done on topics similar to ours. In this chapter we will discuss that in more details.

2.1. Fog Concept

One of the first related works in the domain of fog computing was done by authors in [31]. They outlined the vision and defined the key characteristics of fog computing. They claimed that fog is the appropriate platform for a number of critical IoT services and applications like Smart Grid, Smart Cities, and Connected Vehicles and in general, Wireless sensors and actuator networks. Authors [32] offers a comprehensive definition of the fog. They provide a broad overview of the fog and highlight some of the main challenges faced by the emerging fog paradigm. Cloudlet [14] [33] was built even before the proposal of the fog model, however its concept coincides with fog computing. Authors in [34] discusses the definition of fog computing, introduces some application scenarios and highlights the opportunities and challenges that might come up while designing fog computing system. Authors in [15] proposed the mobile fog concept. Mobile fog is a high level programming model consisting of set of functions and event handlers for future mobile internet applications.
2.2. Fog Computing for Delay-sensitive Applications

Currently there are a few existing works on delay-sensitive fog based applications. Authors in [15] provides a simplified programming abstraction for latency-sensitive applications. They also analyze two use cases such as vehicle tracking using cameras and traffic monitoring using mobility-driven distributed complex event processing (MCEP) systems [35]. However, the proposal does not enlighten us on how to develop a runtime system that implements the mobile fog programming model on real fog-enabled devices.

There have been many efforts in designing smart gateway-based fog computing platforms for time-sensitive applications where, based on the application, gateways can perform different operations like data filtering, notification generation etc. Authors in [36] introduce a smart gateway-based healthcare system using wireless sensor networks. The gateway acts as a bridge between public communication networks and the wireless sensor network. The gateway has a data decision system, lightweight database, and the ability to generate notifications in case of emergency. Authors in [37] present a smartphone based personal health monitoring gateway. In this proposed gateway, a Bluetooth interface is used to upload gathered data to remote servers. Authors in [38] shows a sensor network system using sensing servers as gateways in their system. Authors in [39] propose a mobile gateway for a pervasive healthcare system using Bluetooth and ZigBee. This gateway serves various purposes such as alarm generation in case of emergency and analysis of medical data. Authors in [40] proposed a gateway based on a cell phone for connecting sensor node devices supporting Bluetooth or CDMA. In [41], authors propose an architecture that acquires data via personal health devices via ZigBee, USB or Bluetooth. The aforementioned systems basically use the gateway for
data collection from the data generating sensor nodes and send this data to remote servers. None of the discussed works have considered taking full advantage of the fog by designing intermediate fog nodes and fetching data from those fog nodes.

Authors in [4] discussed a “Smart Gateway” and presented the architecture of a smart gateway with fog computing. In their proposed system, the smart gateway (e. g., a router that is used for data transportation is a gateway node), which is an intermediate layer between IoT and fog node, performs a number of tasks starting from data collection to preprocessing, filtering, reconstruction, uploading only the necessary data to the cloud, keeping check on IoT nodes, and many others. However, we argue this system lacks the essence of the fog concept. This system puts more intelligence on the gateway rather than the fog node.

Authors in [42] proposed a service oriented fog computing architecture that allows patient health monitoring in non-clinical settings. The architecture employs a single fog node at the fog layer for onsite processing. However, we feel, a single fog computer in the fog layer is not well suited for delay sensitive applications. If the single fog node fails to serve due to system or network issues, then the system loses its efficacy. Hence, in our fog model, we propose multiple fog nodes.

Authors in [43] employ a fall detection application to demonstrate the efficacy of fog computing for health monitoring. In their work, the authors investigated and developed new fall detection algorithms and designed and employed a real-time fall detection system called U-Fall. However, to demonstrate the effectiveness of their proposal, the authors compared the response time of U-fall algorithm with pattern-matching and threshold-based algorithms. As they did not compare the response time of the fog with the response
time of the cloud, their work does not substantiate the efficacy of fog over the cloud for delay sensitive applications.

In [44], authors proposed a fog computing model to easily and quickly notify the caregivers in case of emergency. They only compared the upload delay and synchronization delay of fog and cloud. However, we believe, the lower Round Trip Time (RTT) from data generator node to a nearby client node should be of utmost importance as the client needs to see the data as soon as possible for emergency applications. The authors also did not discuss the cloud data center’s distance from the fog node. Therefore, the effect of the distance on the delay is not properly understandable from their work.

So far, none of the literature discussed have explored a robust and significant use of the fog paradigm in real-world time sensitive applications. There is a lack of proper architectural specification as well as a lack of comprehensive time response comparison between fog and cloud. Also, none of the papers we discussed has focused on the data accuracy that is as important as time for some of the applications like health data. The node breakdown situation, which is the situation where one or multiple fog nodes in the architecture suddenly fail(s) or shut(s) down, can also cause severe crisis for mission critical applications. This situation is overlooked in the literatures. Therefore, an all-inclusive study of the efficacy of the fog paradigm for delay sensitive applications is extremely important.
Chapter 3: Our Idea

Having discussed the background of fog computing, we can state the main questions to be explored in our thesis:

1. How to design an effective fog computing architecture for delay-sensitive applications?

2. How the proposed architecture contributes to reducing the round trip delay (RTT) from a data generator node to a client node, which is the node used by the user to access and visualize the data, compared to the cloud?

In an attempt to answer these questions, we have made two research contributions through our work.

Firstly, we present an architecture of cloud-based fog-paradigm for delay sensitive applications. In this proposed architecture, we aim to design the fog layer in such a manner that can take advantage of the fog concept as well as reduce the delay for mission critical applications.

Secondly, we have performed a set of experiments to substantiate the efficacy of our proposed architecture. In these experiments we aim to show the variation of delay with regard to distance between the client and the cloud. In our research, we used healthcare data (heartbeat data) for all experiments as we believe that healthcare related data is one of the most delay-sensitive data that deals with life and death situations. We could have chosen other delay sensitive data like traffic monitoring data, stock market data etc. However, our choice of data emulates any delay-sensitive data and the end user application of our chosen data could be a mobile heart rate monitor.
We plan to develop a node discovery algorithm and effectively implement that in our architecture. The fog nodes become smart and keep the data flow uninterrupted in the architecture by using this algorithm.

Finally, in addition to the main research contributions, we develop a data flushing algorithm. The fog nodes are very small and smart devices. They have a very limited storage capacity. So, we need to make sure that these nodes are not overburdened with data. Data should be short-lived on these nodes and should be flushed/removed on a regular interval. Thus we call these fog nodes ‘Ephemeral’ nodes. However, a key concern in a distributed system is if you remove data from one node then there might be data inconsistency and inaccuracy. Our data flushing algorithm can effectively flush the data from fog nodes while make sure that this data flushing does not cause any data inconsistency and inaccuracy.
Chapter 4: Architecture

4.1 Our Proposed Fog-based Cloud Paradigm

Fog computing enables a new breed of applications and services, and there is a fruitful interplay between the fog and the cloud, particularly when it comes to data management and analytics. Fog computing is a paradigm that extends cloud computing and services to the edge of the network. In our application, we design the fog system as shown in Figure 4.2.1.

4.2 Architectural Explanation

There are only a few fog computing models proposed in the literature. Therefore, we propose a unique fog architectural design for our application of analysis of time-sensitive data. Here, in Figure 4.2.1, we propose this four-layer architecture.

Layer 1 – This is the data generator layer. This layer consists of sensor nodes/IoT devices. These sensors send their data to the upper layer that is layer 2.

Layer 2 - This layer consists of a sequence of fog nodes in the proposed architecture. These nodes are the local access points. We call it the ephemeral storage. This is the layer where the data is continuously coming from layer 1 sensor nodes. These fog nodes have a small storage capacity. Once the data reaches the cloud in layer 3, the cloud would remotely flush the older data from these fog nodes. These data flushing is based on a predefined time that we will discuss later. When the user wants to access recent data, the data communication would be only between the layer 2 and the client layer and thus reduces the communication cost of going to the cloud.
Figure 4.2.1: Proposed fog based cloud architecture
Layer 3 – The layer 3 is the cloud layer. It is the permanent storage layer. Here, we store all data coming from the entire network and process any requests coming from the client nodes from the client layer. The main advantage of having the cloud is that this is helpful for storing old/empirical data and that data can be accessed by the client whenever needed, thus helps the analysis and interpretation of the empirical data.

Client Layer - This layer consists of client nodes which might be a stand-by monitoring machine connected with the layer 2 - fog nodes. This machine can be continuously monitored by a human. This layer can access data from the fog layer whenever an emergency arises so that proper actions can be taken without any further delay. The machines in the client layer also have a connection with the layer 3, the cloud layer. If the user wants to see the historical and empirical data, then he/she can access the data from the cloud layer. Therefore, this connection with the cloud would help the user retrieve the older data and bring it back to the client layer.
Chapter 5: Implementation

5.1 Implementation of Symmetric Structure

In Figure 4.2.1, we propose the detailed architecture of a fog-based cloud paradigm. We believe our proposed architecture is very useful for a large organization such as a hospital or manufacturing industry where several patients/manufacturing steps are monitored every second. In case of an emergency, data can be quickly accessed and rapid human actions can be taken.

In Figure 4.2.1, we can see four parallel structures (legs) which are symmetric (see vertically from the client layer to cloud layer). Each of these legs (symmetry) has a client node, data generator node, fog node and the cloud Node. For our implementation purpose, we have used one of the four legs which are all similar to each other. Our plan is to set up our test-bed of one of these structures consisting of computer machines representing a client node, a data generator node, fog nodes and a cloud node respectively. If this one symmetric structure works as per our hypothesis and research plan, which is reducing the delay for time-sensitive applications, then we could extend this implementation in a mass scale for a large healthcare/manufacturing industry environment. The next diagram, in Figure 5.1.1, shows the part of our proposed architecture that we are implementing in our research.
5.2. Implementation Details of Architecture

Figure 5.2.1 shows test-bed implementation for the proposed plan. This test-bed has four primary machines as mentioned before and all of these machines are connected with each other via a wired connection and all the connections are also coupled through network switches.
Figure 5.2.1: Simplified Implemented Architecture
**Base Station / Data Generator Node** – This is the layer 1 node (represented by a black box). This node is the source of generation of data. Through a network switch this generator node is connected to the fog nodes and to the cloud. We have set one Linux machine as the data generator node.

We emulated real-world sensor (health) data generation scenario at the data generator node using a python script. This python scripts generates 1 file/second with some heart rate values inside the file. The heart rate data is the emulation/representation of any time-sensitive data. We could have also chosen any other time-sensitive data. A real-time mobile heart monitoring system could be the end application for our chosen data.

**Fog Layer** - The fog layer, which is the layer 2, is responsible for processing the data at a local server or a network edge device for immediate response purpose. In this hierarchical structure of this design, we have used two Linux machines as two fog machines. These fog machines are connected with each other via a network switch. The fog1 machine, shown in Figure 5.2.1, is the bridge connection between the base station / data generator node and the fog layer. The main reason to design two fog machines in this layer 2 is to provide robustness to the system. If one of the fog nodes breaks down suddenly, then the other one can work with the rest of the system and in the meantime, the system admin can fix the broken fog node. Thus we can get uninterrupted service while using this proposed fog paradigm in time-sensitive applications.

This fog layer pulls data from the immediate lower layer which is the data generator layer. So, as soon as a data file is generated at the data generator node it is pulled by fog1. When the file comes to fog1 it is immediately pulled to fog2. Fog2 acts as a backup...
storage in the fog layer. In our architecture, fog layer is mainly used to store the recent data. They also act as relay nodes in our dataflow architecture where the data flows from one end (data generator) to other end (cloud/client). Fog1 node is always used to access the recent data. If fog1 node fails for some reason, then the data can be pulled to client from the fog2 node. These fog nodes, using the node discovery algorithm, keeps the data flow in the architecture uninterrupted. Different kind of processing can be done at the fog nodes based on user application. However, our primary goal is to effectively store only the recent data (for last few hours/minutes) on these fog nodes.

**Cloud** - We define the layer 3 which is the final destination for data in this structure – as the cloud. This cloud layer stores all data originally generated by the base station / data generator node in layer 1. The cloud is the master node in this architecture which has the information about every node present in this design and can accordingly communicate with every intermediate node if required. The user can use the cloud-stored data for empirical data analysis [61]. In our experiment we have set one Linux machine as the cloud node. Data from fog nodes come to the cloud via an intermediate node acting as a delay box. This delay box, where Wide Area Network Emulator (WANem) which is a software used to emulate a wide area network is installed, adds some artificial delay into the system. We will discuss WANem in more details later.

**Client** – The client node, which is a Linux machine in our architecture, is connected to the fog layer and also to the cloud via a network switch. This connection enables the client to access the data from fog layer as well as from the cloud.
5.3. Implementation of File Transmission

One of the most important parts in the implementation of our architecture is transmission and synchronization of the data over hierarchical chains of nodes. To accomplish this, we have used RSYNC [62] which is a widely used utility for file synchronization and file transfer in UNIX. Rsync uses SSH to connect to a remote host. Once connected, it invokes the remote host's rsync and then the two programs will determine what parts of the file(s) need to be transferred over the connection. Rsync is an extraordinarily versatile and fast file copying tool that offers a large number of options, and copies the changes from source file to target file.

In our model, rsync is running on all nodes except the data generator node to pull the data from the immediate lower node.

5.4. Software Implementation Details

5.4.1. WANem

WANem [45] is Wide Area Network Emulator software running on Knoppix [46], a Linux distribution based on Debian [47]. It is meant to emulate a real experience of a Wide Area Network during application development and testing over a LAN environment by delaying and relatively dropping packets.

In our architectural implementation of this fog prototype, we use WANem which we have set up on a Linux machine between the fog layer and the cloud machine. In our prototype, the cloud machine is a computer which has been represented as the cloud. We are using WANem to emulate a long distance to the cloud.
All traffic between cloud and all other nodes are forced to pass through the WANem node. WANem provides us the opportunity to emulate the distance in terms of delay. It adds an artificial delay between the client and the cloud node. We can configure WANem through a web browser to add any distance or bandwidth limiter needed in our model.

5.4.2. Network Time Protocol (NTP)

NTP [48] is a networking protocol designed to synchronize clocks between computer systems. If communicating programs are running on different computers and the clocks are not synchronized, then switching between these systems would cause time to jump forward and backward leading to a non-desirable effect.

In our fog model, different machines are communicating with each other and transferring data from one machine to another. As the delay in the data transfer is the primary concern of our research, mismatch in the system clocks would lead to a wrong estimation of delay. To do away with this problem, we have installed and configured NTP on all machines in our model architecture so that all machines follow the same time clock.

In our study, we found an average offset value of .055 milliseconds between the nodes. NTP changes the time clock of the nodes accordingly to reduce the offset value to zero. Hence, all the nodes follow the same time clock.

5.5. Node Failure Handling

Cloud-based systems experience failures due to their heterogeneity and large-scale distributed nature [50]. Most cloud computing services are built upon commercial, unreliable off-the-shelf components. The high failure rate in the hardware and software components causes frequent node and application failure. Node failure in the system may
cause the jobs running on the nodes to abort [51]. Node failure has impacts on system performance and operation costs, hence becoming an increasingly important concern for system designers and administrators [52] [53].

Being an extension of the cloud computing service to the edge of the network, the fog computing model suffers from the same issues like the cloud. Node failure in the fog model may lead to potential application failure and performance degradation [54] that we can't afford in critical time-sensitive applications.

In this section, we aim to investigate how to recover from a node failure situation in our proposed fog model. We will discuss our proposed approach to handle node failure in details.

5.5.1. Layered Architecture

Our fog architecture is a layered architecture. Here data moves from one node to another node in a relay fashion. The advantage of using a layered architecture is when the data is transmitted through the intermediate fog nodes, both fog1 and fog2 nodes keep the same copy of the data. So, the same data can be accessed from both the nodes and even from the cloud. If fog1 node breaks down or is unreachable due to network issues, then we can communicate with fog2 node and fetch the data from there. If both the fog nodes are unreachable then we can go to the permanent storage that is the cloud to access the data.

5.5.2. Node Discovery Algorithm

We have developed a node discovery algorithm, which is our main contribution apart from defining the prototype, in such a way that can overcome the setback caused by a node
breakdown. Each node \((n)\) starting from the fog 1 to cloud is provided with a list of nodes in its lower hierarchy. We call this list a Privilege list. The immediate next node in the lower hierarchy is given the highest privilege. The node next to the most privilege node in the lower hierarchy is given the second highest privilege. It goes on in a decreasing order until the base station.

Whenever a node \((n)\) tries to pull data, it first tries to communicate to the node \((n-1)\) that is most privileged to it. If it can’t do so then it assumes that there is a network/ node breakdown issue that we term as a node failure situation.

When the node \(n\) detects a failure at node \(n-1\), it tries to communicate to the second most privilege node \((n-2)\). If the node \(n\) can communicate with \(n-2\), then it starts fetching the data from node \(n-2\). During that period of data fetching from node \(n-2\), node \(n\) keeps monitoring the node \(n-1\) and tries to communicate with it. Once the node \(n-1\) becomes operational or the network issues are solved and node \(n\) can communicate with node \(n-1\), node \(n\) switches back to node \(n-1\) and start fetching data from it. If both node \(n-1\) and \(n-2\) don’t work, then node \(n\) goes further down in the hierarchy and continues the same process.

If any intermediate fog node breaks down, data is transmitted to the cloud via other active fog nodes using this node discovery algorithm. If a situation arises where all fog nodes break down, then our algorithm connects the cloud to the base station. Hence, data is directly transmitted to the cloud from the base station.

**Figure 5.5.2.1** shows the flowchart for our proposed node failure handling algorithm.
Figure 5.5.2.1: Flowchart for Node Failure Handling Mechanism
5.6. Ephemeral Storage

Our proposed fog model is aimed to make the process of accessing data faster and to reduce the delay for time-sensitive applications. The fog nodes in our proposed model are ephemeral storage nodes. These nodes are designed to store only recent data, not historical data. So, we have set a threshold value in our system. If the amount of data on the fog nodes exceeds this threshold value, the older data would be remotely flushed from the fog nodes by the cloud, given the data has reached the cloud. However, we have to be careful about data accuracy and data consistency [55] [56] before removing any data from the fog nodes. Inaccurate and inconsistent data may lead to improper analysis of the situation and consequently wrong decision making. In this section, we will discuss about data management and periodical data flushing mechanism in our proposed model.

5.6.1. Data Management – Consistency and Accuracy

Data management, that includes data consistency and accuracy, is one of the key aspects of applications deployed and data stored in the cloud based systems [57] [58]. Strong data accuracy and consistency is crucial for all time-sensitive applications which require access to accurate and consistent data. However, the complexity of data consistency increases in the presence of a source of continuous data generation [29] [59] [60]. Keeping the aforementioned factors in mind, we have developed an algorithm that can ensure the data accuracy, consistency, as well as data flushing in our proposed model.
Working Methodology of the Algorithm

In this part, we will discuss how our developed algorithm can effectively manage the data.

Synchronization and Checksum – At the sender end, we first calculate the checksum value of the file that will be transferred. Then we append this checksum value and prepend previous file name with this file’s name. Then the file is transferred.

For example, after appending checksum value and previous file name the new file names look like:

0_1_4183611209
1_2_4209872455
2_3_4183611205

Here the first digit represents the previous file name; second digit represents the current file name and the last number represents the checksum value of the file.

So, for file named 1 there is no previous file so we prepend zero with it. The checksum value of file 1 is 4183611209. We also append this value. So, finally the file name becomes 0_1_4183611209.

For file named 2, the previous file name is 1. The checksum value of file 2 is 4209872455. So, after prepend and append the file name becomes 1_2_4209872455.

For the file named 3, the previous file name is 2. The checksum value of file 3 is 4183611205. So, after prepend and append the file name becomes 2_3_4183611205.

This way it works for rest of the files.

Our algorithm looks at the storage folder and synchronizes the files in accordance with their creation time at base station. So, it makes sure that all the data files are stored in a
consistent order. The entire data flushing operation is based on this synchronization and checksum calculation. We will discuss the operation later.

**Accuracy Check** – Once a data file reaches the storage folder in fog/cloud, the algorithm calculates the checksum value for that file. Then it compares this newly calculated checksum with the old one appended with the file name at the originating node. If there is a mismatch, then that means a data inaccuracy in that file’s content.

Here, rsync plays an important role in ensuring data accuracy. Rsync has an inbuilt feature that looks for the changes in the content of a file, stored in two different places. So, whenever a data inaccuracy is found in a file’s content from its original version, rsync immediately updates the content of the file at the destination node with the original content of the file at originating/sender node. Thus the file at the destination always contains accurate data even if initially there is a data inaccuracy.

If, for a file, the checksums don’t match due to some hardware/technical faults, the cloud contacts the base station node. The cloud replaces the original file with a blank file at the base station. Then it waits for the file to reach the cloud. If the file reaches the cloud with the original content and correct checksum, then that means the intermediate fog nodes are working properly. However, there may be some issues with the base station node. So, for correct operation in the future the base station node should be replaced.

If the blank file reaches cloud with incorrect content and checksum values, then that means the intermediate fog nodes are not working properly. In that case, the cloud checks the file at individual nodes to find out which node is faulty. When the cloud finds a faulty node, it instructs the nodes next to the faulty node to stop fetching data from the faulty node. Then, instead of the faulty node, the data is fetched from other active nodes. If
cloud finds that none of the fog nodes is working properly, then it pulls the data directly from the base station, bypassing the intermediate fog nodes. Hence, the algorithm provides an effective data accuracy and consistency mechanism for our proposed model.

In our study, no faults happened at the nodes. Therefore, we could not test the faulty node conditions.

5.6.2. Data Flushing

Data at the fog nodes are short-lived as there is a physical limitation of data storage. So, data needs to be flushed from the fog nodes on a regular basis. In our fog model, this data flushing operation is monitored and controlled by the cloud as a supreme authority in the architecture. The following components constitute the entire flushing mechanism:

**Storage in Cloud** – The cloud gives us great storage capacity; hence we use it for long-term storage. In our proposed model, the cloud maintains two types of storage, Temporary and Permanent. These two storages are internally synchronized. Once a file, which comes from any of the lower nodes in the hierarchy, pops up in the Temporary storage, at that instant it gets copied to the Permanent storage. Temporary storage is synchronized with the fog nodes as well. So, the data from the fog nodes directly comes to the Temporary storage.

**Threshold limit** – The data flushing operation is triggered by the predefined threshold limit in the system. In our model, intermediate fog nodes can store the data for the last hour. We define this threshold of one hour in terms of number of files. So, if the file generation rate is 1 file/second, then 3600 files are generated in an hour. This is the threshold value of number of files the fog nodes can store.
**Flushing** – Once the number of files in the Temporary storage of cloud exceeds the threshold (3600 in our experiment), the algorithm starts checking the files which are sequentially stored. At first, it checks the checksum of the file. Then it checks whether the files are in the proper sequence. Since all the files contain its name and the previous file name, it creates a linked-list like structure. Therefore, the sequence can easily be verified.

If both the conditions are satisfied, then the cloud remotely deletes the file from fog nodes. After the file is deleted from the fog nodes, it is also deleted from the Temporary storage of the cloud node. Else if the condition(s) fails, the cloud will wait until the file is updated and properly synchronized and then only it will remove the file. In case, if the checksums never match, the cloud deletes the file, based on its name only, from the fog nodes as well as from the temporary storage in cloud, after replacing the file’s content as described earlier.

**Figure 5.6.2.1** shows the flowchart for this entire data management mechanism.
Figure 5.6.2.1: Flowchart for Accuracy Check and Data Flushing Mechanism
Chapter 6: Experimentation

In this chapter, we present the experiments that we performed in order to validate the effectiveness of our proposed model. We also provide the results of the experiments and analysis of the results.

6.1. Experimental Setup

In our work, we aim to compare the RTT delay when the data is accessed at the client from the fog, with the RTT delay when the data is accessed at the client from the cloud. Here, RTT (Round Trip Time) delay is the time for data transmission from the data generator node to the client node. Our aim is to reduce the cloud RTT delay for time sensitive applications. To achieve this goal, we have done experiments with three different data transfer cases.

Case 1: Data is generated at the base station. Then it is transmitted to the cloud through the fog layer. When the user tries to access the data, it sends the request to the fog1 node and fetches the data from the fog1 node. This gives us the Round Trip Time (RTT delay) for data transfer from base station to client node via fog1. Figure 6.1.1 shows the architecture and data flow path for case 1.

Case 2: Data is generated at the base station. Then it is transmitted to the cloud through fog layer. When the user tries to access the data, it sends the request to the cloud node and fetches the data from the cloud node. This gives us the Round Trip Time (RTT delay) for data transfer from base station to client node via cloud in the presence of intermediate fog nodes. Here the fog nodes act as the intermediate data relay/hop nodes. Figure 6.1.2 shows the architecture and data flow path for case 2.
**Case 3:** Data is generated at the base station. Then it is transmitted to the cloud through the delay (WANem) node only. There is no intermediated fog node in the system. When the user tries to access the data, it sends the request to the cloud node and fetches the data from the cloud node. This gives us the Round Trip Time (RTT delay) for data transfer from base station to client node via cloud in the absence of intermediate fog nodes. Figure 6.1.3 shows the architecture and data flow path for case 3.
Figure 6.1.1: Case 1 - Data Transmission Path from Data Generator to Client via Fog1
Figure 6.1.2: Case 2 - Data Transmission Path from Data Generator to Client via Cloud in the presence of Intermediate Fog nodes
Figure 6.1.3: Case 3 - Data Transmission Path from Data Generator to Client via Cloud in the absence of Intermediate Fog nodes
6.2. Experiment Methodology

In our study, we have used health (heart rate) data for experiments. Heart rate data can be a few bytes to a few kilobytes long in size [49]. However, to show the generic nature of our architecture, we experimented with files of different sizes such as 10Kb, 100Kb and 1Mb. As we aim to compare the performance of fog with cloud in terms of delay, working with data of different size over different distances to travel also gives us an insight into the efficacy of our fog model.

6.2.1. Data File Generation

At the base station node, we generate dummy data files every second using a python script. These data files contain heartbeat data. A continuous generation of data files gives us a real sensor data generation-like environment. As we already discussed, for our experiment we generated files of size 10Kb, 100Kb and 1Mb using the python script. Experimentation with different file sizes helps us to understand the generic nature of our model irrespective of the file size.

6.2.2. File Transmission

Data files are transmitted from the base station to the cloud via the intermediate fog layer. To successfully accomplish this transmission, we have used Rsync which is embedded within a Python script.

The Python-Rsync script is continuously running on all nodes except the data generator node. Once the data is generated at the generator node, fog1 node pulls the data from the generator node. Then as soon as the data file pops up at fog1, fog2 node pulls the
Finally, when the data file arrives at fog2, Cloud node pulls the data from fog2. In this way, the data transmission in the proposed model takes place.

### 6.2.3. Emulation of Distance

To emulate a real data center distance in our network, we have examined the ping time to three Amazon data centers, North Virginia, Oregon and Ireland. From the base station node, located at Cincinnati, we pinged these three data centers. The average round trip time is calculated for 100 data samples which are 64 bytes each. It was done during daytime on weekdays between 10am to 5pm. Table 6.2.3.1 shows the ping time for these three data centers from our base station at Cincinnati.

<table>
<thead>
<tr>
<th>Data Center</th>
<th>N. Virginia</th>
<th>Oregon</th>
<th>Ireland</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delay (RTT in ms)</td>
<td>28</td>
<td>64</td>
<td>124</td>
</tr>
</tbody>
</table>

**Table 6.2.3.1: Ping time for data centers**

In our model, WANem is used to emulate the distance between the fog nodes and the data centers. The ping time values we got are the Round Trip Time (RTT) values. So, these values are \(2\) (one-way traversal time). Therefore, to emulate the actual distance we need to provide the one-way delay value to the WANem as input. Therefore, we have given half of the RTT (ping) values as the input to the WANem using the web interface. When pinging our emulated cloud node from the base station, the ping times matched our actual values from AWS (Amazon Web Services). Table 6.2.3.2 shows the WANem inputs.
<table>
<thead>
<tr>
<th>Data Center</th>
<th>N. Virginia</th>
<th>Oregon</th>
<th>Ireland</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delay (ms) Input to WANem</td>
<td>(28/2)=14</td>
<td>(64/2)=32</td>
<td>(124/2)=62</td>
</tr>
</tbody>
</table>

**Table 6.2.3.2: WANem Delay input**

6.2.4. Access Data from fog and Cloud

In our study, we aim to get the data at a client node in optimal time. We wish to reduce the data access time at a client using the fog nodes. Hence, we need to compare the performance of the cloud and the fog in terms of delay. Our experiment has three cases:

1. Accessing data from fog1,
2. Accessing data from cloud in the presence of intermediate fog nodes,
3. Accessing data from cloud in the absence of intermediate fog nodes.

We have already discussed these three cases in Experimental setup section.

To get the data at the client node, we run the Python-Rsync script at the client to pull the data from fog1/cloud. For case 1, this script runs at the client and fetches the data from the fog1 node. So for case 1, the data transmission path from data generator to client is: Data generator node $\rightarrow$ fog1 $\rightarrow$ client.

For case 2, the script runs at the client and fetches the data from the cloud node. So for case 2, the data transmission path from data generator to client is: Data generator node $\rightarrow$ fog1 $\rightarrow$ fog2 $\rightarrow$ cloud $\rightarrow$ client.

For case 3, the script runs at the client and fetches the data from the cloud node. No fog node is present in the system. So for case 3, the data transmission path from data generator to client is: Data generator node $\rightarrow$ cloud $\rightarrow$ client.
In all three cases, 100 data files are transferred from source to destination and the average time for the transfer is calculated. The architecture is physically implemented with machines at our home.

6.3. Results

The results of the experiments on different file sizes for case1, case2 and case 3 are shown graphically in Figure 6.3.1. In this graph the x-axis represents the delay input to the WANem web interface and y-axis represents the Round Trip Time (RTT delay). Data points on the plot show the RTT values with respect to the delay inputs for three data centers. The plot shows the change in the delay due to the fog and the cloud respectively. It also shows the change in the delay in the presence and absence of fog nodes, while pulling the data from cloud. All X and Y units are in milliseconds. The legends show the line specifications.
Figure 6.3.1: Delay graph
6.4. Analysis
The results show that using fog nodes does reduce the delay of accessing data by a significant amount compared to the cloud. In this section, we will take a closer look at the results and analyze their implications.

- **Data Center Distance vs RTT** – We have already discussed the emulation of a real WAN-like environment using WANem. We have emulated the WAN environment in terms of ping time delay. The delay, the cloud incurs, is mainly due to the distance of the cloud from the client. Therefore, we would also like to analyze the RTT delay in terms of distance. It would help us better understand the relation of the delay with the distance. Table 6.4.1 shows the corresponding details.

<table>
<thead>
<tr>
<th>Data Center</th>
<th>Distance (approx. in miles)</th>
<th>RTT(ms)</th>
<th>10k</th>
<th>100k</th>
<th>1Mb</th>
</tr>
</thead>
<tbody>
<tr>
<td>North Virginia</td>
<td>509</td>
<td>219</td>
<td>259</td>
<td>281</td>
<td></td>
</tr>
<tr>
<td>Oregon</td>
<td>2217</td>
<td>307</td>
<td>362</td>
<td>405</td>
<td></td>
</tr>
<tr>
<td>Ireland</td>
<td>3588</td>
<td>369</td>
<td>441</td>
<td>493</td>
<td></td>
</tr>
</tbody>
</table>

**Table 6.4.1: Data center distance and corresponding Round Trip Time**

- **Fog-RTT vs Cloud-RTT** – In our experiment we pulled data to the client from the fog and the cloud respectively. We compare the RTT for these two scenarios. It shows the reduction of the delay, while using fog to access the data, compared to the cloud. It helps us better understand the importance of having fog.
We also tried to observe the effects of intermediate fog nodes on the delay, when the data is pulled from the cloud. The data takes lesser time to reach the cloud, from the base station, when it travels directly to the cloud via only the WANem node, as opposed to the transmission via fog nodes and WANem node. The delay increases due to multiple hops. The following table gives us a clear picture of the improvement in delay while using fog.

<table>
<thead>
<tr>
<th>Data Coming From</th>
<th>Data Size</th>
<th>RTT(ms)</th>
<th>10k</th>
<th>100k</th>
<th>1Mb</th>
</tr>
</thead>
<tbody>
<tr>
<td>fog1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>North Virginia</td>
<td>Case 2</td>
<td></td>
<td>219</td>
<td>259</td>
<td>281</td>
</tr>
<tr>
<td></td>
<td>Case 3</td>
<td></td>
<td>195</td>
<td>222</td>
<td>240</td>
</tr>
<tr>
<td>Oregon</td>
<td>Case 2</td>
<td></td>
<td>307</td>
<td>362</td>
<td>405</td>
</tr>
<tr>
<td></td>
<td>Case 3</td>
<td></td>
<td>236</td>
<td>283</td>
<td>310</td>
</tr>
<tr>
<td>Ireland</td>
<td>Case 2</td>
<td></td>
<td>369</td>
<td>441</td>
<td>493</td>
</tr>
<tr>
<td></td>
<td>Case 3</td>
<td></td>
<td>275</td>
<td>325</td>
<td>376</td>
</tr>
</tbody>
</table>

**Data Transmission Path**

- Case 2: Data generator node → Fog1 → Fog2 → cloud → client
- Case 3: Data generator node → cloud → client

**Table 6.4.2: Comparison between RTTs of fog and cloud**
Increase in Delay – Our results clearly show that using cloud to access the data, instead of fog, does increase the delay (RTT). When we try to access the data from the Ireland data center, which is almost 3,588 miles away from our current location, the delay actually becomes more than twice as compared to the fog, in the presence of intermediate hop (fog) nodes. In the absence of intermediate hop (fog) nodes, the delay becomes almost 1.5 times of the delay caused by the fog. The graphical representation of the results shows a clear pattern in the delay increment.

Also the delay increases with the size of the data. This is also an important point to note. In real world scenario we might be trying to access data that is a few gigabytes. If delay increases in this linear manner, then that can cause a huge delay for those big data files that we can’t afford for delay-sensitive applications. In the existing IoT-cloud interaction models, which are currently used for time-sensitive applications, the data generated by the IoT device is stored in the cloud and from there it comes to the client. However, this existing model is not suitable for real-time applications like critical health monitoring, traffic monitoring etc., where even a few milliseconds delay can be detrimental. Cloud incurs a significant delay when data is accessed from the cloud in this kind of time-sensitive applications. Fog can effectively reduce the delay by a significant amount that is proven by our experiments.

Therefore, the outcomes of our experiment strongly substantiate our claim of superiority of the proposed fog architecture over cloud for delay-sensitive applications.

6.5. Node Failure Experiment

We have tested the node failure algorithm on the implemented physical structure of our proposed architecture. We randomly turned off machines and unplugged network cables
in the middle of data transmission. We noticed that our algorithm has been able to successfully recover from the setback caused by the node failure. Figure 6.5.1 shows one of the test cases for our node failure experiment.

Figure 6.5.1: Node failure test case

In this experiment of figure 6.5.1, cloud pulls data from the fog2 which is the most privileged node for the cloud. In the middle of the transmission fog2 is suddenly turned
off. This might have caused data loss in the destination of the data. However, using the node discovery algorithm that we have developed, cloud immediately finds fog1 that is the next most privileged node to it and starts pulling the data from fog1. In that way data transmission remains uninterrupted in the system. From the figure it can be seen that cloud pulls up to file number 27 (file name format is: previous filename _ current file name_ current file's checksum) from fog2 and then when it starts pulling files from fog1 after the breakdown of fog2, files starting from number 28 are pulled. Thus, the data transmission remains consistent in the system in spite of a node breakdown.

We ran the same experiment several times with different file sizes that we already have discussed. Each time the cloud received all the data files generated at the base station, although one or all the intermediate fog nodes were turned off. This shows the effectiveness of our node failure algorithm.

Screenshots of some of the outputs of our algorithm are shown in Appendix A.

6.6. Data Flushing Experiment

We have designed our data flushing algorithm in such a way that each fog node can store the data for the last hour only. Our algorithm determines this timeframe by the number of data files. In our study, 3600 data files are generated every hour. So, when the number of files on the fog nodes exceeds this value, the cloud starts removing the old files from the fog nodes given that the files have successfully reached the cloud.

In our experiment with data flushing, at the sender node we continuously generated data for two hours. We noticed, at the end of the first hour, old files are automatically deleted from the fog nodes as well as from the temporary storage of the cloud. Fog nodes keep
the data which are most recent, in our case this is the data generated in the last hour only. We also noticed, though the fog nodes keep the recent data only, the permanent storage folder of cloud stores all the empirical data. Hence, at the end of two hours, fog nodes keep the data for the second hour only but the permanent storage folder of cloud keeps the data for last two hours. When we run our experiments, we notice that the fog nodes are never overburdened with data. These nodes store only the recent data and whenever the amount of data on these nodes exceeds the defined threshold value, data flushing algorithm starts removing the oldest data file first. Then the second oldest data file is removed and this way it moves on until the number of files on the fog nodes become less than the threshold value. Thus the fog nodes are able to store the recent data only while all the empirical/old data is stored on the cloud. Hence, this experiment shows the efficacy of our data flushing algorithm and proves the ephemeral characteristics of fog nodes in our architecture where fog nodes store only the recent data.
Chapter 7: Conclusions

Fog computing performs better than cloud computing in meeting the demands of quick transmission of the data to the client and smart decision making in emergency situations for time sensitive applications. The biggest advantage that fog computing provides us is reduced delay, compared to the cloud, that is evident from our experimental results. This is crucial for time-sensitive applications where the data needs to reach the client in the least possible time. In our research, we have successfully built and physically implemented and tested a fog-based prototype for time-sensitive applications that can reduce the delay by a significant amount. Based on this prototype, which is the first in its type in terms of physical implementation with real hardware devices, we can implement real-world applications and can add more computational capability to fog nodes. Our experimental results show that when the cloud is relatively far from the user, it incurs a significant amount of delay and that could be detrimental for delay-sensitive applications like healthcare where the delay could be life threatening.

Our proposed architecture is one of the very few prototypes to answer the open question – how to design and develop a real-world fog computing based pervasive data monitoring system. Our work discusses the expansion of IoTs and their integration with fog based cloud computing for enhanced and more useful service provisioning to the user. We pay to use cloud bandwidth. Therefore, if we can access data from the fog, instead of going to the cloud, then that is economically beneficial as well.

However, fog computing cannot totally replace cloud computing, as it will still be preferred for storage and high end computation intensive jobs that are very common in the business
world. Hence, we can come to the conclusion that fog computing and cloud computing will complement each other while having their own advantages and disadvantages. Edge computing plays a crucial role in Internet of Things (IoT). Studies related to security, confidentiality and system reliability in the fog computing is an open research problem. Fog computing helps the emerging network paradigms that require fast processing with minimal delay, whereas cloud computing would serve the business community meeting their high end computing demands for big data processing based on a utility pricing model.
Chapter 8: Future Work

Our work contributes a simple framework for implementing a fog based data movement paradigm for time-sensitive applications. Our work also provides experimental results that justify the framework’s efficiency to reduce the delay for time-sensitive applications as well as its capability of handling adverse scenarios like node failure and assuring data accuracy. While our framework and experiments give a good insight into a fog computing model for time-sensitive applications, there are opportunities for further research and several improvements can be made.

- **More Intelligence to the fog** – In our work, the fog nodes work as ephemeral storage and data relay nodes. However, the number of emergency situations in different sectors like healthcare, manufacturing, etc. is growing. Generating notifications for such emergency events in a quick and easy manner is becoming increasingly important. Therefore, we would like to extend our work to provide intelligence and computational authority to the fog nodes and to develop a quick and effective way of notifying emergency events by fog nodes.

- **Data aggregation and compression** – Currently we have not employed any data aggregation and compression scheme in our work. However, there might be other types of data from different sectors such as manufacturing and social media that could be significantly larger in size.Trimming and pre-processing the data before sending to the cloud is very important for a better and quick service provisioning. To help lessen the burden on the cloud and the network we would like to implement an efficient data aggregation and compression algorithm between fog and cloud while dealing with large data samples.
• **More efficient Node Discovery algorithm** – The architecture we have proposed here, which is a very simplified structure, has only two intermediate fog nodes. So, maintaining a list of nodes on those nodes is very convenient. However, in a large real-world scenario there might be thousands of intermediate nodes. Maintaining a list of those nodes and going through the list to find an optimal one could be computationally expensive and might incur some delay that we cannot afford in delay-sensitive applications. Therefore, we would like to work on a more efficient node discovery algorithm that could find the near-optimal node much more quickly and effectively without causing much delay. The algorithm would be dynamic as well that can effectively handle node addition and deletion from the network.

• **Improvements in Data Flushing** – As fog nodes are ephemeral storage nodes, so flushing the data from those nodes on a regular basis are of huge importance. In our work, we have shown how the cloud being the central authority can flush the data from fog nodes. However, one limitation is that if a fog node goes down, the cloud waits for that node to be operational again to flush the data. The cloud does not delete the data from other fog nodes which are currently active until the failed node starts running properly. Hence the volume of data starts increasing on the fog nodes as new data comes into the nodes. If there is a very strict space constraint for data storage on the fog nodes, then our current algorithm might not be a good fit. In that case we would like to investigate more about the data flushing algorithm and would like to come up with a more efficient and dynamic one that would be the best fit for large and strict space-constrained data repository.
- **Parallel Architecture Connected to Real Data Centers** – One limitation of this work is availability of number of machines to implement a large, real-world like scenario. We have implemented only one symmetric structure from our proposed detailed architecture. The future goal would be to incorporate real sensors and a large number of machines to implement and test the original parallel architecture that we proposed. In addition to that, instead of emulating the data center distance, we would like to connect our system to the real data centers and rerun our experiments.
Bibliography


[59] R. Rawson and J. Gray, "HBase at Hadoop World NYC." R. Rawson and J. Gray, "HBase at Hadoop World NYC."


Appendix A

Node Failure Test Cases

A.1 Test Case I

Cloud node tries to connect with fog2 which is the most privileged node for cloud to pull the data. Fog2 node works perfectly fine, that’s why cloud is able to connect with fog1 and can pull the data from fog2. This is a test case for normal operation when there is no node/network breakdown.
A.2 Test Case II

Node Fog2 works perfectly and node Fog1 breaks down at the very beginning of transmission and Fog2 can't communicate with Fog1. Hence Node Fog2 starts fetching data from the base station.
Appendix B

Codes

B.1 Data File Generation

#!/usr/bin/python

#---------------------------------------------------------
---

#Author: Satyajit Bhowmick
#Date: January, 2016
#Generate files and append previous file_name, checksum value
#---------------------------------------------------------

#import modules
import shutil
import os
import time
import subpr
occess

src="/home/base/source/"
dest="/home/base/data/"

#get the list of files in the source folder
listOfFiles = os.listdir(src)
listOfFiles = map(str,sorted(map(int,listOfFiles)))

for filename in listOfFiles:
    #calculate checksum and append file name, cheksum value
subprocess.call(['cat /home/base/source/{} | cksum > ~/check.txt', shell=True])

f=open('/home/base/check.txt')

f_list=f.readlines()

split=f_list[0].split()

if listofFiles.index(filename)==0:
    #for the first file, prepend 0 as the previous file name
    newname='0'+'.'+filename+'.'+split[0]
else:
    newname=listOfFiles[(listOfFiles.index(filename))-1]+'.'+filename+'.'+split[0]

os.rename(src+filename, src+newname)

shutil.copy(src+newname, dest)

# include time delay of 1 sec between two file generation

time.sleep(1)
B.2 Data Transmission and Node Failure

#!/usr/bin/python

#------------------------------------------------------------
#Author: Satyajit Bhowmick
#Date: January,2016
#Pull data & handle node failure
#------------------------------------------------------------

#import module to embed shell commands in python
import subprocess

cloud = '/home/cloud/temp_store/

#list of nodes
#to cloud - FOG2 is of highest priority
nodes = ['fog2@192.168.200.117:/home/fog2/data/','fog1@192.168.200.88:/home/fog1/data/','base@192.168.200.152:/home/base/data/']

#pull data
while True:
    try:
        subprocess.check_call(['rsync -az -e ssh ' + nodes[0] + ' ' + cloud], shell=True)
    except subprocess.CalledProcessError:
        subprocess.call(['rsync -az -e ssh ' + nodes[1] + ' ' + cloud], shell=True)
    except :
        subprocess.call(['rsync -az -e ssh ' + nodes[2] + ' ' + cloud], shell=True)
    except :
        pass
B.3 Data Flushing

#!/usr/bin/python

#----------------------------------------------
#Author: Satyajit Bhowmick
#Date: February, 2016
#Data flushing
#----------------------------------------------

#import modules
import os
import subprocess

#files are getting stored in cloud
path="/home/cloud/temp_store/

#set the threshold for flushing data
n=200  #though originally 3600, to better understand used 200
while 1:
    #files are sorted according to the creation/last modification time
    files = sorted(os.listdir(path), key=lambda x: os.path.getctime(path+x))

    if len(files)>n:
        #calculate the checksum in cloud
        subprocess.call(['cat /home/cloud/temp_store/' + files[0] + ' | cksum > ~/check.txt'], shell=True)
        f=open('/home/cloud/check.txt')
        f_list=f.readlines()
        f.close()
try:

# get the appended values
    split=f_list[0].split() #split=[hashvalue,'no of bits']
    l1=files[0].split('_') #l(n)=['prev file name','this file name','hash val']
    l2=files[1].split('_')

# check the checksum value and the file sequence
    if split[0]==l1[2] and l1[1]==l2[0]:
        try:
            # flush the data from fog nodes
            subprocess.call(['ssh fog1@192.168.200.88 '+'rm /home/fog1/data/' + files[0]], shell=True)
            subprocess.call(['ssh fog2@192.168.200.117 '+'rm /home/fog2/data/' + files[0]], shell=True)
            # flush the data from temporary storage in cloud
            os.remove(path+files[0])

        except subprocess.CalledProcessError:
            subprocess.call(['rsync -az -e ssh base@192.168.200.152:/home/base/data/ /home/cloud/temp_store'], shell=True)
        except OSError:
            subprocess.call(['rsync -az -e ssh base@192.168.200.152:/home/base/data/ /home/cloud/temp_store'], shell=True)
        except:
            pass
B.4 Data Access and Visualization at Client

#!/usr/bin/python

#---------------------------------------------------------------
#Author: Satyajit Bhowmick
#Date: February,2016
#Real-time data visualization at client
#---------------------------------------------------------------

#import modules
import os
import time
import matplotlib.pyplot as plt
import subprocess

#get the data from fog1
subprocess.call(['rsync -az -e ssh fog1@192.168.200.88:/home/fog1/data/ /home/client/data/'], shell=True)

#maintain a counter of number of files in the previous iteration
number_of_files=0

#specify the data location
src="/home/client/data/

while 1:
    try:
        open("bigfile.txt", 'w').close()
        #list of sorted files based on creation time
        listOfFiles = sorted(os.listdir(src), key=lambda x: os.path.getctime(src+x))

        #when the data size decreases
if number_of_files<=len(listOfFiles):
    number_of_files=len(listOfFiles)
    #append the values
    f = open("bigfile.txt", "a")
    for tempfile in listOfFiles:
        f.write((open(src+str(tempfile),'r')).read())

    #real-time graph generation
    #clear current figure
    plt.clf()
    #read the content of appended data
    Read=open("bigfile.txt", "r").read()

    #split the line contents
    R=Read.splitlines()
    time.sleep(1)

    #convert R from string to int
    R=(map(int,R))

    #set the range of x axis
    x=[i for i in range(len(R))]

    #plot the values
    plt.figure(figsize=(20,10))
    plt.plot(x, R)
    plt.draw()
    plt.savefig("myfig.png")
except:
    pass