IMPLEMENTATION OF A DISTRIBUTED MIDDLEWARE FRAMEWORK TO SPAN THE EDGE - FOG - CLOUD TIERS OF A MOBILE DISTRIBUTED COMPUTING IoE PLATFORM REQUIRING REAL-TIME AND NEAR-REAL-TIME RESPONSE

by

Arindrajit Seal

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Electrical Engineering

Charlotte

2020

Approved by:

____
Dr. Arindam Mukherjee

____
Dr. Chen Chen

____
Dr. Tao Han

____
Dr. Fareena Saqib

____
Dr. Susan Sell

ABSTRACT

ARINDRAJIT SEAL. Implementation of a Distributed Middleware Framework to span the Edge - Fog - Cloud tiers of a Mobile Distributed Computing IoE Platform requiring real-time and near-real-time response. (Under the direction of DR. ARINDAM MUKHERJEE).

This report motivates the roles of Cloud computing, Edge computing, and the hierarchically distributed cooperative Fog computing, for the real-time analysis of big-data in Internets-of-Everything (IoEs). IoEs are enhanced Internets of Things (IoTs) which integrate people, process, data and heterogeneous “Things”: compute, storage, and sensor/actuator hardware. The ubiquitousness of IoE devices, the ever-increasing amount of big-data in IoEs, and the need for real-time computing in IoEs have motivated the problem of distributed data storage and analysis. With trillions (big-data scale) of IoE devices on the verge of being deployed in tomorrow’s ever-connected and autonomous society, and with the expected big-data generated by each such IoE device (typically image data of the order of tens and hundreds of gigabytes per day), we are rapidly approaching Big-Squared data dimensions. The power consumption of traditional cloud data centers are already about 70% of all power generated, and it will increase exponentially if Cloud computing is the only solution for tomorrow’s IoEs. Moreover, the Big-Squared-Data from merging IoEs will create network and compute level bottlenecks that will be impractical from a real-time standpoint, especially in case of rapid mobility in IoEs. Hence, the need for distributed hierarchical Fog computing and associated data management. I survey key features of emerging IoEs, the existing big-data computing and storage frameworks, and point out their capabilities and deficiencies. I discuss the design and implementation of my Fog computing architecture. I present my work in Accident Identification and Related Congestion Control. The results show that having a

Fog Computing Layer helps in cutting down data bandwidth to the cloud and reduces total latency by approximately 80%. Finally, I have shown that the integration of Apache Kafka, Apache Cassandra and Spark Streaming in a Fog Computing Architecture has a greater impact on the problem at hand and it is successful in solving both the issues of real-time and near-real-time responses along with containing the Big-Squared-Data to just big-data and conserving the data bandwidth.

ACKNOWLEDGEMENTS

First and foremost I would like to thank Dr. Arindam Mukherjee who allowed me to work with him in his lab at UNC Charlotte and without whom this would not have been possible. Dr. Mukherjee’s ideas and valuable inputs to the research made the entire endeavour quite smooth and an enriching experience for me. True research can only be made possible when your mind is given much needed freedom to do the things in your own comfort zone irrespective of the time of conducting the research. Dr. Mukherjee did just that and I am grateful that he believed in me and put his trust in me. I would also like to thank my committee members like Dr. Tao Han, Dr. Fareena Saqib, Dr. Susan Sell and Dr. Chen Chen for being a part of my dissertation committee and helping me from time to time with my research. The department of Electrical and Computer Engineering at The University of North Carolina at Charlotte also played a pivotal role. I owe my gratitude to the department as it deemed me worthy to utilize its resources and also for funding my entire education – a big contribution indeed.

I would certainly acknowledge the contribution of Mr. Sumanta Bhattacharyya who collaborated with me in my ongoing research. Mr. Bhattacharyya helped in designing an accident identification dataset which was crucial for my research. He managed to collect and assemble the dataset from scratch. A very tiresome but necessary job that needed to be done and for that I am grateful.

I am also indebted to my parents for being a constant source of support during my time as a Ph.D. student. It is only because of their prayers and sacrifices that I have reached where I reached today.

Last but not the least; I would like to thank the Almighty for giving me the opportunity to attain a doctoral degree.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

Cloud technology is generally based on data centers, which handle data storage and analysis. It’s more like a client server paradigm where the server, residing inside the cloud, does all the work and returns the result to the devices or clients. With the ever-increasing number and bandwidth requirements of devices that are connected to the internet however, cloud is not able to handle the big-data in real-time, or even with acceptable delays. In 2012, global commercialization of Internets-of-EveryThing (IoEs) based systems created a profit of $4.8 trillion [1], with CISCO estimating a 21% increase of profits for the IoE sector [2]. However, the current infrastructure for IoE computing in the big-squared data space is inadequate – an IoE with billions of autonomous vehicles that operate in a rapid mobile environment, while sensing and processing data at rates of 10 Gb/day to 100Gb/day in realtime (low latency) with high Quality of Service (QoS), presents an ideal benchmark for my proposed framework. Fog Computing is taken into consideration when we try to elevate the standard of performances of IoE devices that are mobile. By performance we mainly indulge ourselves in realtime output. [3] shows that this term - coined by CISCO, is basically intelligence imparted on edge devices and also other devices in the hierarchy mainly for producing real-time output and reducing latency. Fog Computing is a natural ally of Cloud Computing as is aforementioned already but the essentiality of Fog Computing is realized when we talk about latency resilience and real-time data generated from these network edge devices.

1.1 Motivation

The main storage of cloud computing are the data centers all around the globe. They communicate among themselves. Presently, every day to day activity is reliant on the cloud so eventually the data centers are running constantly. [4] references the mathematical model for Cloud Analysis by Zhang et. al. With the increase in IoE devices, traditional cloud computing will be facing the following challenges:- (i) The International Data Corporation (IDC) foretells that the global market for IoE will grow from $1.9 trillion (2013) to $7.1 trillion (2020) [5]. (ii) The US Environmental Protection Agency(EPA) stated that the data centers of the United States consumed a total power of 61 billion kWh whose expenditure was around $4.5 billion. In 2007, it was seen that 30 million servers worldwide spent 100 TWh of the world’s energy [6] & [7]. (iii) Currently the planet’s energy consumption is around 1500 TWh [8]. By traffic controlling , the relative up-time of the data centers can be reduced which will help in reduction in the overall power consumption. The essentiality of Fog/Edge Computing comes into the picture when we talk about real-time and near-real-time responses. Computation, if done on or very close to the devices from where data gets generated, will reduce the data load over the cloud. By being able to reduce the data bandwidth to the cloud we are in a way able to reduce the total up-time of the cloud and hence we are able to reduce the energy consumption. By having a Fog based architecture we are able to decentralize the computation and storage which previously was centralized in the case of only a cloud. Furthermore, if the Fog Computing instances were equipped with a cluster computing framework, the same ones that are used in a cloud, we will be able to further distribute the data that will give rise to even further parallel processing and real-time outputs. So in conclusion we need a distributed middleware framework over the three tiers of computation Edge-Fog-Cloud that will be helping in the Big-Squared-Data Engineering due to the rapid increase in IoT/IoE devices.

1.2 Contribution

The main contributions of this research are listed below:- (i) This work motivates the need for Fog computing as a computing and big-data storage infrastructure, that is physically closer to the millions of points of data generation and actuation in an IoE. The presence of Fog Computing is expected to enable Real-Time computing for IoEs, by load balancing with existing Cloud computing infrastructure and reducing network bandwidth requirements. (ii) I survey key features of emerging IoEs, the existing big data computing and storage frameworks, and point out their capabilities and deficiencies. (iii) Finally, I discuss the design and implementation of my Fog computing architecture.

1.3 Progression

The following sections have been assigned according to the order of progression of the research. Chapter 2 deals with a literature survey of the previous works that have been done in IoT frameworks. Chapter 3 shows us the existing Big Data Frameworks and categorizes them into either Batch or Stream Processing frameworks. In chapter 4 I talk about the Fog Computing Layer and present my Fog Computing Architecture as the 2nd Tier in the three tiered architecture of Edge, Fog and Cloud. In chapter 5 I talk about using my Fog Architecture performing accident prediction by various deep-learning models namely YOLOv3 and Tiny YOLOv3 over images and related congestion aware navigation based on the results of the image object detection, in near real-time. Chapter 6 talks about the same framework this time only on the Cloud over an Amazon EMR cluster running Apache Spark. Chapter 7 talks about the Ongoing Research and things that are needed to be done before the final defense.

CHAPTER 2: RELATED WORK

2.1 Internet of Things

Presently, IoE comprises of devices that are connected to the Internet enabling machine-machine (M2M) communication[9],[10],[11]. Thus when we talk about IoT/IoE frameworks we focus on data storage, computation and managing the data[12]. Devices relying heavily on the internet are gradually converging towards IoT/IoE[13]. CISCO has estimated that by 2020 around 50 million devices will be connected to the Internet[14]. Thus a surplus of applications needs to be processed from the domain of IoTs/IoEs. The data generated will be of high magnitude with a very rapid velocity. The Big-Data generated from these devices are anticipated to be serviced in real-time[16]-[18] enabling a reduction in the overall latency. Therefore, the performance of IoTs/IoEs should be heavily dependent on cloud[19]-[22], and how well the cloud can service a rainbow of applications requiring services in real-time[12],[23].

2.2 Cloud Computing

Formerly, a number of quality researches have provided the way the processes are delegated in the cloud[24]-[27]. In [28] , Xiao et. al. proposed a work which would focus on optimal positioning of data centers for better QoS in terms of latency and cost efficiency. Tziritas et al. [30] stated migration of processes to better improve the performance of cloud systems. By using job scheduling techniques, Chandio et. al. tried to improve QoS by comparing jobs[31]. Several other scheduling algorithms be it for real-time workload[32] or for energy-efficient scheduling[33] have been proposed on a small-scale . With the increase in IoE devices at an alarming rate, the data centers would be insufficient in handling all of the requests from all these devices. There would be denial of services in the worst case or a high latency in servicing . [34] - [37] states that as we are gradually moving towards technology, we are harming nature, thus making it mandatory to maintain the eco-friendliness of our surroundings.

2.3 Fog Computing

CISCO coined the term Fog Computing, a revolutionary idea which mitigates the limitations of cloud computing [3], [38]. It is a distributed computing infrastructure which is able to handle a lot of internet connected devices. Bonomi et al. [38] showed us the importance of Fog - Cloud interplay. [39] shows the characteristics of the architecture in terms of location, geographical distribution, latency . Hong et al. [40] created a programming model for Fog infrastructure addressing issues like geographical distribution, real-time applications. The importance of Fog was pointed out by Yannuzzi et al. [41] and Preden et al. [42] but only at a top-level. [43] considered various computing models including cloud and worked out the essentiality of building up a Fog Computing Platform. Do et al. [44], Aazam and Huh [45],[46] have worked on various problems in resource allocation in Fog. Few researchers have also worked on security angles in Fog Computing of late [47] - [49].

2.4 Key Features of Emerging IoEs

IoE has revolutionized today’s smart societies by facilitating machine-to-machine (M2M) communications. Whereas IoT focuses on the integration of “Things” or heterogeneous smart devices, IoE (Internet of EveryThing) - a termed coined by CISCO - focuses on the integration of 4 primary tuples, namely People, Process, Data and “Things”. Some of the key features of emerging IoEs are listed below.

2.4.1 Rapid Mobility

Mobile IoE edge devices traverse large geographical distances within short time intervals. A way to interface these devices with the ubiquitous compute and storage access is Mobile Cloud Computing (MCC) [51] in which edge devices communicate directly with the cloud using wireless networks. However, there are strong drawbacks such as (1) long compute latency due to network delays and cloud congestion, (2) very high network bandwidth requirement, and (3) low availability of the cloud instance due to signal attenuation.

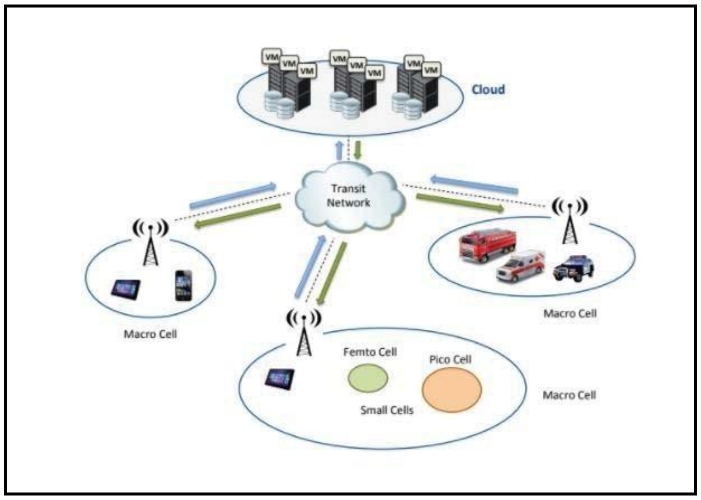


FIGURE 1: MOBILE CLOUD COMPUTING(MCC). THE UPWARD ARROWS INDICATE DATA REQUESTS, UPLOADS AND DOWNWARD ARROWS SIGNIFY COMMANDS OR DOWNLOADS.

Alternatively, Heterogeneous Networks (HetNets) [51] can be used to offer wireless coverage in an area with a variety in wireless coverage zones, ranging from an outdoor environment to indoor office buildings, homes, and underground areas. Such a HetNet would typically be a Wide Area Network with macrocells, picocells, and/or femtocells with complex handshaking between the cells to provide coverage, with handoff capability between network elements as shown in Figure 1. HetNets are defined by Small Cell Forum as a multi-x environment where x denotes technology, domain, spectrum, operator, vendor'. Although a HetNet improves access to the cloud, it does not minimize the network congestion and subsequent latency of service.

The combination of MCC and HetNet is not able to deal with rapid mobility since macro cells are designed to provide signal coverage to larger areas and hence avoid frequent handovers. The drawback lies in the low data rates and high signal instability since macro cells generally contain a surplus of mobile nodes in them which ensures that each device inside the macrocell is bandwidth constrained and they get a low data rate. Pico and femto cells, on the other hand, provide much higher data rates and signal stability since these cells generally contain a lesser number of mobile devices, but their coverage is restricted to a few hundred meters, thus they are not conceived for serving applications under fast mobility.

Yannuzzi et.al [52] conducted experiments considering three different mobility scenarios: (i) Mobile Node Handover, (ii) Process Migration using CRIU(Checkpoint/Restore in Userspace) - a software tool in linux operating system for freezing a running application and checkpointing it, and (iii) Process Migration oriented Mobile Node Handovers - Mobile devices changing access points for a handover towards endpoints. The second case deals with interdomain process migration. In the last situation both the endpoint and the process move together in a completely real-time system. The figure below shows the nature of the service disruptions in three different cases.

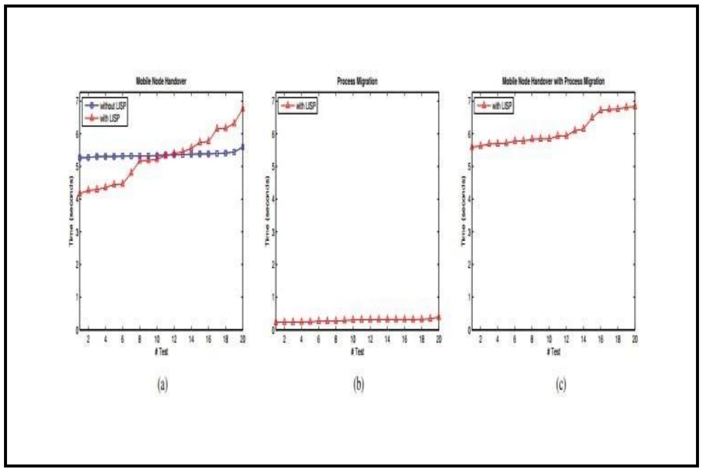


FIGURE 2: DISRUPTIONS IN SERVICES DUE TO: (A) MOBILE NODE HANDOVER; (B) PROCESS MIGRATION (CRIU); (C) MOBILE NODE HANDOVER WITH PROCESS MIGRATION [52].

2.4.2 Seamless Integration of Diverse Nodes and Hardwares

Any device that can connect itself to the internet today and communicate valuable data is a part of the IoE. For this specific reason the importance of sensory data is of valuable importance in the field of energy management, waste management, traffic control, smart transportation, healthcare systems, smart agriculture and smart greenhouse gas monitoring, just to name a few.

[55] discusses various IoT protocols offered by IEEE,IETF and ITU to enable various new devices to join in this virtually real world of IoEs . This paper also includes a discussion on management and security protocols. Kazmi et. al. [54] discusses the modelling of heterogeneous IoE data streams in order to overcome the challenge of heterogeneity. Thus to orchestrate the varied number of jobs based on priority coming in from various heterogeneous edge devices, to establish protocol neutrality(different edge devices communicate by different communication protocols), we need a middleware. Along with the diverse hardwares that take part in the data transportation, storage and analysis, there may also be heterogeneity in the application processes. The middleware

must be smart enough to arbitrate between billions to devices that are connected.

2.4.3 Structured and Unstructured data

Structured data usually refers to the data in a RDBMS(Relational Database Management System), it consists of data fitted into rows and columns with every row identifying a particular entity with various characteristics. We generally use SQL(Structured Query Language) in dealing with structured data. Unstructured data refers to the rest of the data which include videos, images, sensor data, text files, chat data, mobile phone data in general. We use NoSQL(non SQL) in case of unstructured data.

Data produced by most IoE devices which operate on the network edge is unstructured data. Tingli LI et.al. in [56] talks about storage of unstructured data in IoT through a novel storage solution called IOTMDB based on NoSQL. It is also estimated that the magnitude of the data that is generated by the IoE in future will surpass even the scale of Big-Data .Present Big-Data frameworks on the cloud are not designed to handle such a massive scale of data. IoEs require real-time output with negligible latency and present Big-Data Frameworks do not take latency into consideration.

2.4.4 Scalability and Connectivity

[57] addresses the issue of connectivity and scalability in the Fog Computing Framework(to be discussed elaborately in section V) by easily integrating nodes into either cooperating mode or a task sharing mode. The approach uses a middleware platform for Distributed Cooperative Data Analytics (DCDA) in the fog premises.

2.4.5 Real-Time Computation

The cloud is a physically distributed group of servers, but from a network point of view it is centralized. IoE requires real-time analysis in which case the cloud fails due to two major reasons: 1) high latency of service due to an increased Turn Around Time (TAT) 2) Low Bandwidth and Network Unavailability in regions with poor connectivity. Hazem En Raafat et.al. [58] extracted statistical features from sensor data from the IoEs to minimize latency and storage.

CHAPTER 3: BIG-DATA FRAMEWORKS

Handling the volume, variety, and velocity of IoE big-data requires a new computing model with the following requirements: Reduce latency: In order to meet Real-Time constraints, data analysis has to be made closer to the edge device. Conserve network bandwidth and storage: IoE data will eventually congest any network unless hierarchical computation and data volume reduction is done between the data producers and data analyzers. Address security concerns: The data generated from IoE devices should be secured from source to the destination. Operate reliably: Any framework for big-data analysis must have high availability and reliability. In the following subsections I survey the existing frameworks for big-data analysis and storage, and point out their advantages and deficiencies.

3.1 Big-Data Analysis Frameworks

3.1.1 Batch Processing Frameworks

Apache Hadoop is a batch processing framework. Hadoop was the first big-data framework to gain significant recognition in the open-source community. Based on a lot of facts and reports by Google about how they were dealing with a surplus amount of data at the time, Hadoop tailored the algorithms and made abstraction framework to make large scale batch processing feasible. Present versions of Hadoop comprises of many components or layers, that work together on batch data.

HDFS: HDFS, which stands for Hadoop Distributed File System, is the distributed file system for storage on hadoop cluster nodes using a replication factor. HDFS ensures that the entire hadoop framework is fault-tolerant.

YARN: YARN, which stands for Yet Another Resource Negotiator, is the coordinator in the Hadoop stack which maintains the cluster. It is a job scheduler and a coordinator for the underlying resources. YARN made it possible for a wide variety of workloads to be run on the cluster, than was possible in earlier versions, by acting as an interface to the cluster resources.

MapReduce: MapReduce is Hadoop's engine for batch processing of jobs.

Although Hadoop has some clear advantages in terms of scalability (storing and distributing large datasets across hundreds of inexpensive servers), fault-tolerance(data gets replicated within the cluster of nodes), flexibility(used for a wide range of purposes including fraud-detection, data warehousing, recommendation systems, log processing, market campaign analysis), it lags behind in some key areas which became evident after the advent of IoTs followed by the current IoEs.

The shortcomings of Hadoop includes security(missing encryption at the storage and network levels),vulnerability(the entire framework is written in Java- a widely used but controversial language since it is heavily exploited by cybercriminals), latency(the system was not designed to administer results for real-time and near real-time data analysis).

Apache Spark is a batch processing framework of the next generation that also does stream processing. Inspired by the MapReduce model of Hadoop, Spark can quicken up workloads in batch processing by donating a processing optimization and full in-memory computability.

Spark can act as an alternative to the MapReduce engine by hooking up with Hadoop.

Limitations of Spark include no support for real-time processing(although it does support near real-time processing) , lack of a file management system, in-memory computation creates a bottleneck for cost-effective computation of big data.

3.1.2 Stream Processing Frameworks

Apache Storm is a data stream processing framework that outputs results with extremely low latency and is suited for workloads which require near real-time and realtime analysis There are a lot of similarities that exist between Storm and Hadoop in terms of scalability and fault-tolerance. Storm is extensively used for near real-time and real-time analysis of IoE data.

Apache Samza is a data stream processing framework that is bound to the Apache Kafka messaging system. Kafka may be used by many processing systems.Samza utilizes Kafka's architecture providing fault tolerance and state storage.Samza uses YARN for resource allocation .

Apache Flink is stream framework handling batch tasks. It treats batch processing as a subset of stream processing since it considers large datasets to have finite boundaries thus categorizing the workload automatically as batch. This stream approach unfortunately has side-effects.

This stream-first approach has been named as the Kappa Architecture. The Lambda Architecture is batch-first architecture that occasionally uses streams to faster produce results.

Although stream processing has solved the issue of real-time data analysis of IoEs to an extent, it has a major disadvantage since it only exists on the cloud. [5]-[7] mentions the exponential increase in the number of IoE devices by 2020 and warns us about the increasing power consumption by the data centers with implications of increased CO2 emissions. Computation bottleneck will hence result as an aftermath of network bottleneck [38].

3.2 Big-Data Storage Frameworks

3.2.1 SQL Databases

It is possible to access Big-Data stored in the HDFS of Hadoop through SQL-on-Hadoop. Apache Hive was the first SQL-on-Hadoop engine. Presently, many new engines have been released like Concurrent Lingual, Hadapt, InfiniDB, CitusDB, Cloudera Impala, JethroData, MammothDB, Pivotal HawQ, Progress DataDirect, ScleraDB, Apache Drill, MemSQL, Simba and Splice Machine. All of the aforementioned database engines work on SQL query.

3.2.2 NoSQL Databases

NoSQL is the new breed of the Database Management System (DBMS). These databases typically support horizontal scaling, do not fit their data into tabular form with rows and columns, have a structured storage and they also avoid JOINs . Some of the most widely used NoSQL database engines are MongoDB, Redis, CouchDB, RevenDB, MemcacheDB, Riak, Neo4j, HBASE, Perst, HyperGraphDB, Cassandra, Voldemort, Terrastore, NeoDatis, MyOODB, OrientDB, InfoGrid, Db4objects.

CHAPTER 4: FOG COMPUTING ARCHITECTURE

In this Section I present the fog computing architecture and its details. It is important to mention that fog computing is a non-trivial extension of cloud computing and extends the services of cloud to the network edge.

4.1 Assumptions

Fog paradigm is still in its early stage of research and is yet to shape up. I, therefore, draw few simple, yet realistic assumptions. • Edge devices, also termed here as Terminal Nodes(TNs), are able to share their geospatial location information through technologies such as GPS(Global Positioning System), GIS(Geographic Information System), or GNSS(Global Navigation Satellite System) so that services are provided to the TNs in real-time based on data analysis of its geospatial location. Specific geospatial perimeters inside which these TNs lie are known as Virtual Clusters(VCs), as shown in Figure. 3 and Figure. 4. • Fog Computing devices are “intelligent” based on their storage and computability [38], [40]. Apart from forwarding and routing, they are also in charge of decision making. These devices decide the suitability of the instance(either Cloud or Fog) which is best fitted to the running of an application. Every Fog Instance (FI), see Figure. 3 is in charge of a VC. • Fog Computing Devices support Rapid Mobility(travelling from one virtual cluster to the other) of the TNs through interfog instance communication.

4.2 System Outline

This subsection illustrates the distinct tiers of a generic fog computing architecture. As depicted in Figure. 3, it is essentially three tier architecture. The tiers are discussed below. (a) Tier 1: This is the bottom-most tier of the architecture.This tier is also referred to as the ‘Edge Tier’, see Figure. 3. The tier comprises of several TNs. The TNs are smart, wireless sensor nodes that sense heterogeneous location specific parameters and transfer the same to the immediate upper tier. (b) Tier 2: The tier 2 or the middle layer is also known as the ‘Fog Tier’. Components of this tier are intelligent intermediate devices (such as routers, gateways, switches, and access points, PCs) that possess the ability of data storage, computation, routing, and packet forwarding. (c) Tier 3: The uppermost tier is commonly known as the cloud computing tier. This tier comprises of servers and PCs.

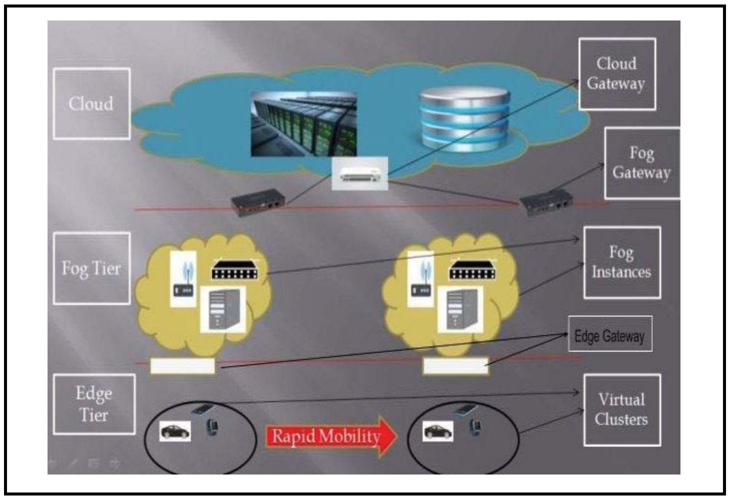


FIGURE 3: FOG COMPUTING ARCHITECTURE

4.3 Architecture Details

Virtual Clusters (VCs) are location based perimetres which consists of IoE devices that are being referred to as TNs. TNs constantly scan their environment and send the data to the Fog. The Fog Tier consists of devices that range from routers,gateways, access points, switches to PCs. A Fog Instance (FI) is in charge of its own VC. As proposed by Bonomi et al., [38] the fog computing architecture can be classified into two sub-parts, viz., (a) the fog abstraction layer and (b) the fog orchestration layer. While the former manages the fog resources, enables virtualization, and preserves tenant privacy, the latter beholds the exclusive fog properties. The fog orchestration layer comprises of a small software agent – foglet which monitors the state of the devices, a distributed database to account for scalability and fault tolerance, and a service orchestration module which is responsible for policy-based routing of application requests.Within the FIs, the data are processed and analyzed to decide whether it needs to be transmitted to the cloud DCs.

Application requests which require storage or historical data based analytics are redirected to the cloud, else, the data are processed within the fog units. The fog devices possess limited semi-permanent storage that allow temporary data storage and serve the latency-sensitive applications in real-time. The cloud computing tier is commonly responsible for permanent storage of huge, voluminous data chunks within its powerful DCs. The DCs are equipped with massive computational ability. However, unlike conventional cloud architecture, the core cloud DCs are not bombarded for every single query. Fog computing enables the cloud tier to be accessed and utilized in an efficient and controlled manner.

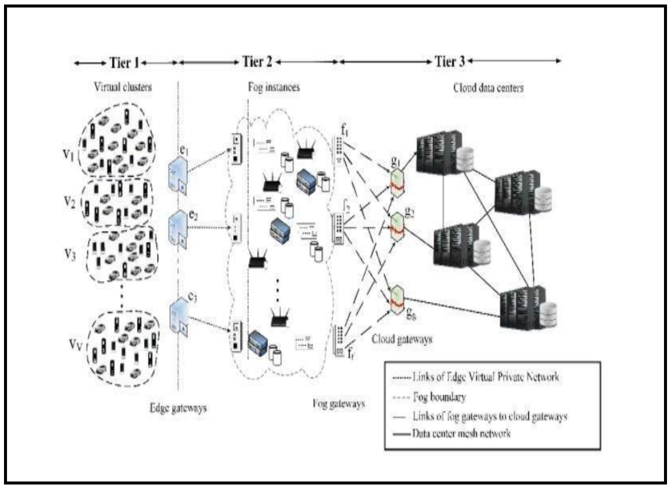


FIGURE 4: NETWORKING LINKS AND COMPONENTS OF FOG COMPUTING [53]

4.4 My Implementation of Fog

(i) **Edge Nodes**: In this work I have used both Raspberry Pi 3 (RPi3) and Google Pixel 2 (Walleye) as my edge nodes or TNs. The RPi has specifications of 1.2 GHz CPU, Armv8 4 cores architecture, Broadcom Video IV GPU, 1GB LPDDR2 RAM, 10/100 Ethernet, 2.4GHz 802.11n wireless and a microSD storage with Raspbian OS. Pixel 2 has a system specification of 1.9 - 2.45 GHz CPU, AARCH64 8 cores architecture, Wi-Fi 2.4G + 5GHz 802.11 a/b/g/n/ac, 4 GB RAM , 52.2 GB Internal Storage with Android 8.0 “Oreo” OS.

(ii) **Fog Instances** : Intel NUCs play the role of Fog Instances or Servers having a system configuration of Intel ® core i7 - 7567U CPU @ 3.50 GHz with a 16 GB RAM DDR4 memory, 512 GB NVME SSD, Intel(R) Dual Band Wireless (802.11ac) and Bluetooth 4.2, Intel ® Gigabit LAN, Micro SD card slot .

(iii) **Cloud Instance**: We have established UNCC’s biggest Hadoop-based research cloud with 39 nodes. The data center is currently available to researchers working with BigData, Mobile Clouds and Internets-of-Everything. In response to a proposal in 2016, Intel made an infrastructure donation of 39 Xeon based business class computing/embedded servers to build a data center in the ECE department at UNCC, with the implied commitment to upgrade machines regularly in future under their “waterfall” program. The Cloud is connected to the Edge Gateways through Fog Gateways.

Following are some key technical specs of the servers and the system: 1) There are currently 39 1U servers, each requiring 350W for computing. The processors are Xeon E3, or 2nd/3rd generation i3 based (about 100 GFLOPS per server node). 2) Each server node is equipped with a 10Gbps NIC card for communication. 3) Each server node has 3TB HDD 2.5in SATA 6Gb/s 7200RPM. The servers are connected by a switching system with the spec: 48-port 10GbE + 12-port 40/56GbE Non-blocking Open Ethernet ToR Switch System.

CHAPTER 5: BENCHMARK DATASET AND APPLICATION IN NEAR REAL-TIME ACCIDENT PREDICTION AND RELATED CONGESTION CONTROL

This work focuses on developing (i) a benchmark application for Real-Time traffic incidence identification and related traffic management, using Real-Time congestion-aware navigation of smart vehicles (Edge nodes) with video feeds, (ii) an image database for Deep Learning used for recognition and classification of traffic incidences such as accidents and congestions, (iii) the System Level Software (or Middleware) required for Distributed Computing in such a heterogeneous Real-Time constrained system with Rapid Mobility - today's Internet-of-Everything (IoE), and (iv) a hardware prototype of the distributed computing and storage infrastructure. The video bandwidth requirement of 10-100 GigaBytes of data per minute per vehicular camera makes it a Big Data problem. With millions of smart vehicles projected to be deployed within the next 5 years, BigData from a single vehicle, multiplied with the large number of vehicles, presents a Big-Squared-Data computing space which will easily overwhelm any Cloud infrastructure with its Real-Time or near Real-Time demands. Hence the need for a Fog tier between the Edge nodes and the Cloud to bring distributed computation (servers) and storage closer to the Edge nodes. Such a Fog consists of multiple Fog instances, each one of which services cells or Virtual Clusters of Edge nodes. Results show that Fog-Cloud computing framework outperforms a Cloud-only platform by 79.7% reduction in total latency or response time.

5.1 Deep learning and Modeling

The objective of this analysis is to predict the kind of traffic incident (or accident) and the associated congestion along with it, from live video feed images from the smart vehicles, and re-route their paths based on the time taken for the clearance. This image analysis for traffic incidence and congestion prediction, and Deep Learning (training), are done continuously in Real-Time, along with congestion-aware re-routing. Fig. 5 shown below presents a block diagram of the underlying workflow of the deep learning and subsequent machine learning based model development.

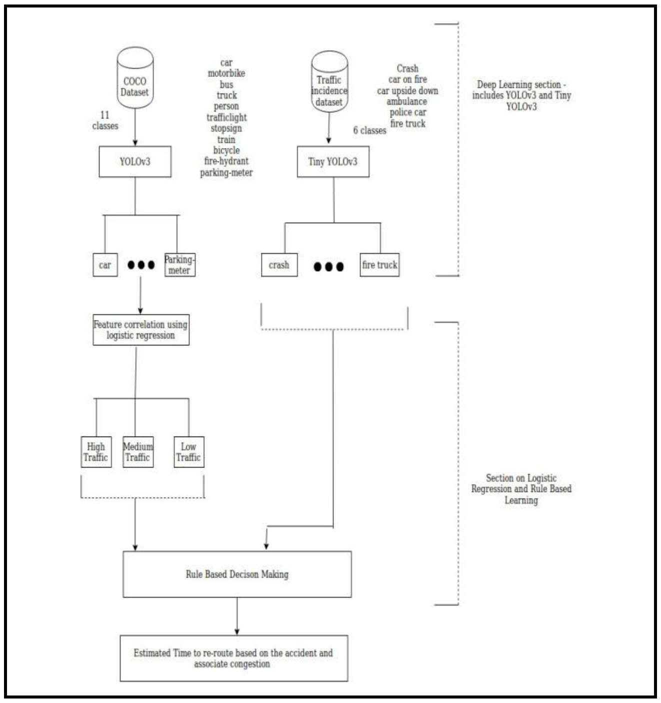

COCO Dataset
11 classes
car
motorbike
bus
truck
person
trafficlight
stopsign
train
bicycle
fire-hydrant
parking-meter
YOLOv3
car
Parking-meter
Traffic incidence dataset
Crash
car on fire
car upside down
ambulance
police car
fire truck
6 classes
Tiny YOLOv3
crash
fire truck
Deep Learning section - includes YOLOv3 and Tiny YOLOv3
Feature correlation using logistic regression
High Traffic
Medium Traffic
Low Traffic
Section on Logistic Regression and Rule Based Learning
Rule Based Decison Making
Estimated Time to re-route based on the accident and associate congestion

FIGURE 5: DEEP LEARNING AND MODEL DEVELOPMENT

In recent years, evolution of convolutional neural networks (CNN) have resulted in significant improvement in object detection and recognition. The original YOLOv3 [59], [60] has been trained on a Microsoft Common Objects in Context (COCO) dataset with 80 different objects, most of which are unrelated to traffic. In this work I am using two CNN architectures to classify the data: YOLOv3.11 (for 11 classes related to traffic congestion) for traffic object identification and tiny YOLOv3.6 (for 6 classes related to traffic incidences).

In Fig.6, YOLOv3.11 was used for object detection primarily because of its state-of-the-art performance with reasonable accuracy. It has 106 layers, is fully convolutional with residual skip connections and up sampling, and detection is done by applying 1x1 detection kernels on feature maps of three different sizes at three different places in the network. It is trained on the COCO dataset. I used the classes: ‘car’, ‘motorbike’, ‘bus’, ’truck’, ’person’, ‘traffic light’, ‘stop sign’, ‘train’, ‘bicycle’, ‘fire-hydrant’, ‘parking-meter’ out of the 80 classes in the COCO dataset for our object detection module.

The shape of the detection kernel is 1x1x(Bx(5 + C)). Here B is the number of bounding boxes a cell on the feature map can predict, C is the no of classes. For my architecture I took 11 classes, B = 3 and C = 11, so the kernel size is 1 x 1 x 48.

I used feature correlation using logistic regression on the predicted outputs from the first network to assign the input image in any one of these 3 target classes: high congestion, medium congestion and low congestion. These will be our three buckets to measure the degree of congestion. Details of this process is discussed in section below.

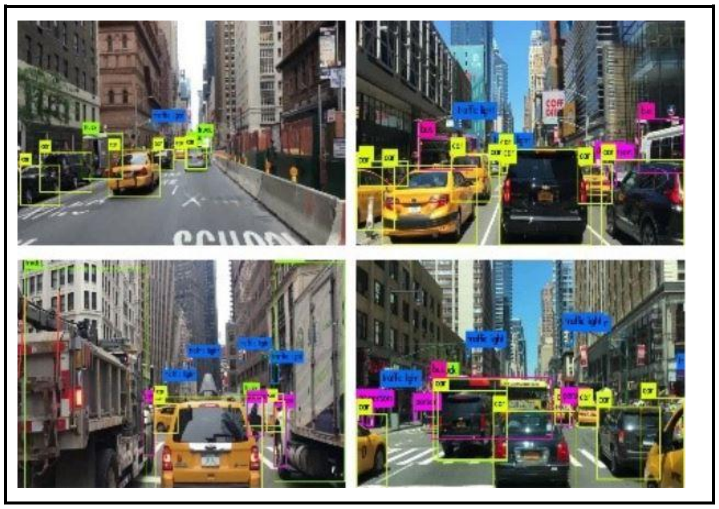


FIGURE 6: RESULT OF YOLOV3.11

Dataset to Test: Since image databases required for this learning are not easily available or open-sourced, we recorded about 20 hours of video, driving around

Charlotte, North Carolina and lower Manhattan, New York city. We extracted 10876 relevant images from video data using a video-to-image converter and tested on them.

For the second network, we chose tiny YOLOv3.6 for detection. It is meant to be used for resource constrained environments. It has 23 layers in a similar architecture to YOLOv3. We used this network due to its better training on smaller datasets. It detects 6 classes: police car, ambulance, crash, car on fire, car upside down, and fire truck. We designed our own annotated dataset containing around 600 images of these classes to train and 150 test images. The following image is the training plot showing the loss vs iteration. We achieve convergence at 50,138 iterations with a current average loss of 0.288 and a learning rate of 0.001 (Fig. 7).

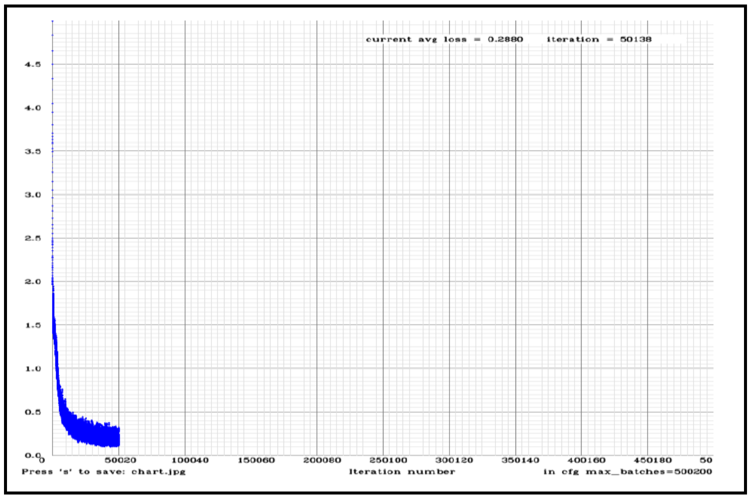


FIGURE 7: CURRENT AVG. LOSS VS ITERATION

Since there is no well defined dataset of accident pictures, we needed to create our own dataset and annotate them properly. Due to the small size of the dataset, we could not train it on YOLOv3 architecture, like the COCO dataset, as there is a strong possibility of loss of accuracy. We found the architecture of Fig. 2 to be more accurate than a single network architecture. We are open-sourcing our dataset for the community.

Accuracy: We used mean Average Precision(mAP) to evaluate our detection model. The idea of Average Precision can be conceptually viewed as finding the area under the precision-recall graph. Table 1 shows us the mAP values for our model.

Precision: Measures how accurate our predictions are, as the ratio of true positives to all positives.

$$\frac{TruePositive}{TruePositive + FalsePositive}$$

Recall: Measures how well the algorithm finds all the positives.

$$\frac{TruePositive}{TruePositive + FalseNegative}$$

TABLE 1: MINIMUM AVERAGE PRECISION

configuration	**mAP**	**Train Dataset**	**Test Dataset**	**classes**	**remark**
YOLOv3	51.5	COCO dataset	COCO dataset	80 classes	106 layers
YOLOv3.11	43.7	COCO dataset	our driving dataset	11 classes	106 layers
Tiny YOLOv3.6	31.5	traffic incidence dataset	traffic incidence dataset	6 classes	23 layers

5.2 Supervised and unsupervised learning

The trained YOLOv3 model detects objects from the images and passes the objects and their confidence scores to a Linear Logistic Regression model (Supervised Learning). The model performs feature correlation and classifies the image into 3 buckets of congestion categorized as high, medium and low.

The trained Tiny YOLOv3 model detects incidences from the same image (if any) and gives confidence scores for the classes categorized as crash, car on fire, fire truck, police car and ambulance, and car upside down. This information is integrated with the output of the Linear Logistic Regression using a rule based approach.

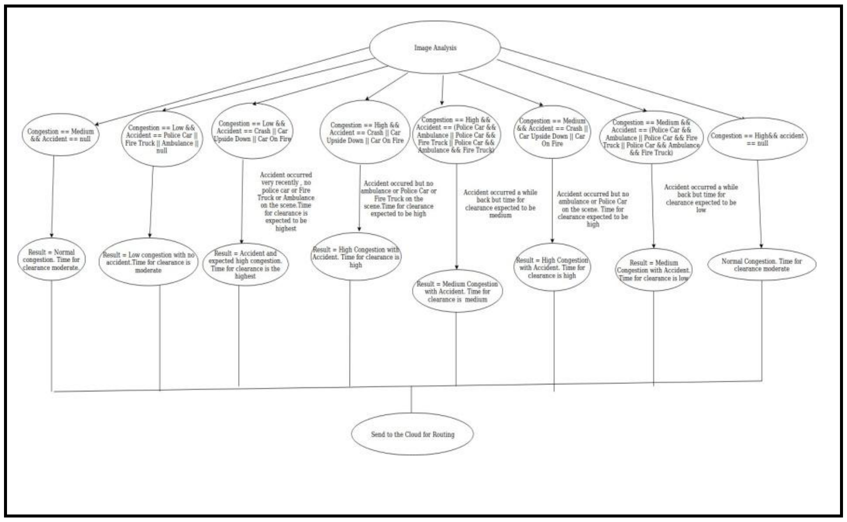


Image Analysis
Congestion == Medium && Accident == null
Congestion == Low && Accident == Police Car || Fire Truck || Ambulance || null
Congestion == Low && Accident == Crash || Car Upside Down || Car On Fire
Congestion == High && Accident == Crash || Car Upside Down || Car On Fire
Congestion == High && Accident == (Police Car && Ambulance || Police Car && Fire Truck || Police Car && Ambulance && Fire Truck)
Congestion == Medium && Accident == Crash || Car Upside Down || Car On Fire
Congestion == Medium && Accident == (Police Car && Ambulance || Police Car && Fire Truck || Police Car && Ambulance && Fire Truck)
Congestion == High&& accident == null
Accident occurred very recently , no police car or Fire Truck or Ambulance on the scene.Time for clearance is expected to be highest
Accident occured but no ambulance or Police Car or Fire Truck on the scene.Time for clearance expected to be high
Accident occurred a while back but time for clearance expected to be medium
Accident occurred but no ambulance or Police Car on the scene. Time for clearance expected to be high
Accident occurred a while back but time for clearance expected to be low
Result = Normal congestion. Time for clearance moderate.
Result = Low congestion with no accident.Time for clearance is moderate
Result = Accident and expected high congestion. Time for clearance is the highest
Result = High Congestion with Accident. Time for clearance is high
Result = Medium Congestion with Accident. Time for clearance is medium
Result = High Congestion with Accident. Time for clearance is high
Result = Medium Congestion with Accident. Time for clearance is low
Normal Congestion. Time for clearance moderate
Send to the Cloud for Routing

FIGURE 8: RULE BASED LEARNING

We correlate the output from the Logistic Regression and the outputs from the Tiny YOLOv3.6 using a Rule Based Learning Approach. Fig. 8 shows us how the model learns and integrates kinds of congestion along with detection of an accident for setting up the Time for clearance of the roads (edges of the graph).

5.3 Traffic Incidence Detection and Congestion-Aware Navigation

I created an undirected weighted graph with the nodes as the intersection for the roads of Lower Manhattan in New York City (NYC), and the roads represented by the edges. When smart vehicles request the route from a given source to a destination, its corresponding FI passes it on to the cloud for routing. The cloud calculates the congestion-aware shortest route, and that information is passed down to the car through its FI. Meanwhile, the car keeps on capturing street videos and presents them to the Fog for interpretation. The aforementioned models derived from deep learning are applied on to the images (in test mode) from the videos frame-by-frame to recognize traffic incidences and congestion. That information is passed on to the cloud (as edge weights) to update its connected routing graph and re-route vehicles.

My framework performs image analysis on the Fog and congestion aware routing on the Cloud. In order to reduce the computation overhead from the increasing number of devices connected to the cloud [1], [2] and also to reduce the consumption of power from the data centers [3], I am limiting the Big Squared Data by processing the bulk of the bandwidth (images from videos) on the FI itself.

The Cloud has in its possession a History Table (HT) which stores the counter value in minutes for each Result Type. The Counter can be thought of as a timer that starts decreasing after it is initially set. When it reaches zero the edge of the graph that was assigned a higher weight because of congestion is reset to the initial weight. Fig. 9 shown below shows us the flowchart of the routing on the cloud along with the HT updation.

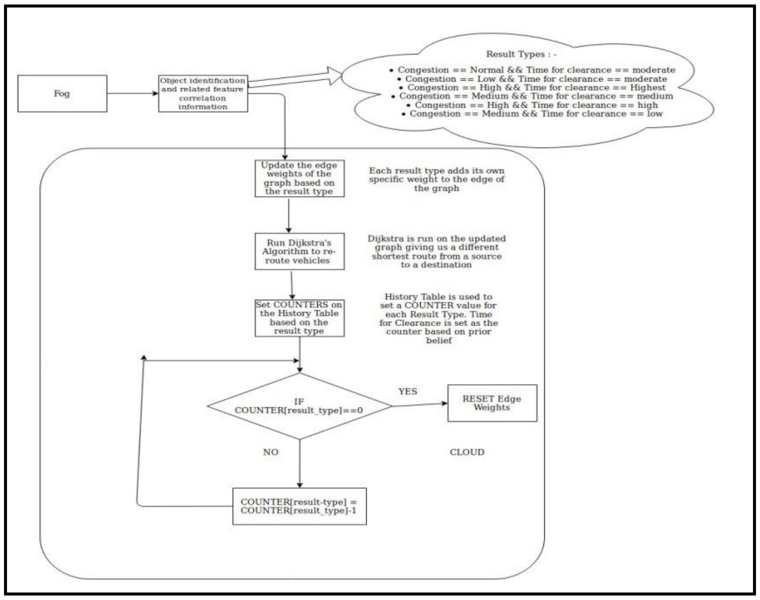

Fog
Object identification and related feature correlation information
Result Types : -
• Congestion == Normal && Time for clearance == moderate
• Congestion == Low && Time for clearance == moderate
• Congestion == High && Time for clearance == Highest
• Congestion == Medium && Time for clearance == medium
• Congestion == High && Time for clearance == high
• Congestion == Medium && Time for clearance == low
Update the edge weights of the graph based on the result type
Each result type adds its own specific weight to the edge of the graph
Run Dijkstra's Algorithm to re-route vehicles
Dijkstra is run on the updated graph giving us a different shortest route from a source to a destination
Set COUNTERS on the History Table based on the result type
History Table is used to set a COUNTER value for each Result Type. Time for Clearance is set as the counter based on prior belief
IF COUNTER[result_type]==0
YES
RESET Edge Weights
NO
CLOUD
COUNTER[result-type] = COUNTER[result_type]-1

FIGURE 9: ROUTING ON THE CLOUD AND HISTORY TABLE UPDATION

5.4 Middleware

Fig. 10 depicts the different functions executed by the Edge nodes and the Fog and Cloud instances. Except for the Image Classifier and Routing functions, which are application codes, all other functions are the system software (middleware) codes. The send or receive functions handle communication between the different tiers and between Fog instances, while the Fog and Cloud Schedulers manage application job executions (specific to different vehicle data and requests). Navigation and image data are stored in the Cloud and Fog databases, which are managed by the corresponding DB_Manager functions. The Execute Application function includes Image Processing (which is forwarded to the Cloud by a busy Fog Instance) and Routing (Global).

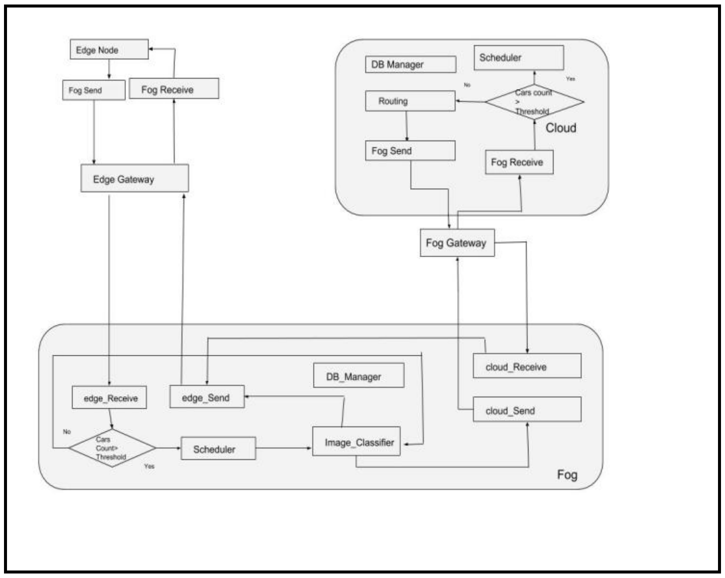


FIGURE 10: MIDDLEWARE IN THE CLOUD, FOG AND EDGE TIERS

Table 2 shows the different middleware functionalities.

TABLE 2 : MIDDLEWARE

TIER	**MIDDLEWARE**	**APPLICATION**
Tier 1 - Edge Node	Edge DB Management Edge Job Scheduling Edge-Fog Communication Handler	Collect videos from cameras
Tier 2 - Fog instance	Fog DB Management Fog Job Scheduling Fog-Edge Communication Handler Fog-Cloud Communication Handler	Local Navigation Image Analysis

	Fog-Fog Communication Handler	
Tier 3 - Cloud instance	Cloud DB Management Cloud Job Scheduling Cloud-Fog Communication Handler	Global Navigation Image Analysis (when Fog is busy)

5.4.1 Job Scheduling

In order to service the requests efficiently I have devised MultiThreaded Fog and Cloud servers which can simultaneously service many client requests at the same time through parallel processing of the data. First-In-First-Out(FIFO) Queue scheduling has been implemented for both Fog servers and the Cloud server. The client requests will only be queued after the thresholding limit of multithreading is exceeded. The threshold limit of multithreading depends on the server configurations. For the Fog servers the thresholding value has been found to be approximately 2000 parallel threads for normal computations which do not involve heavy deep-learning computations like YOLO. Fig. 11 shown below shows the behaviour of the Fog system when I increase the number of threads.

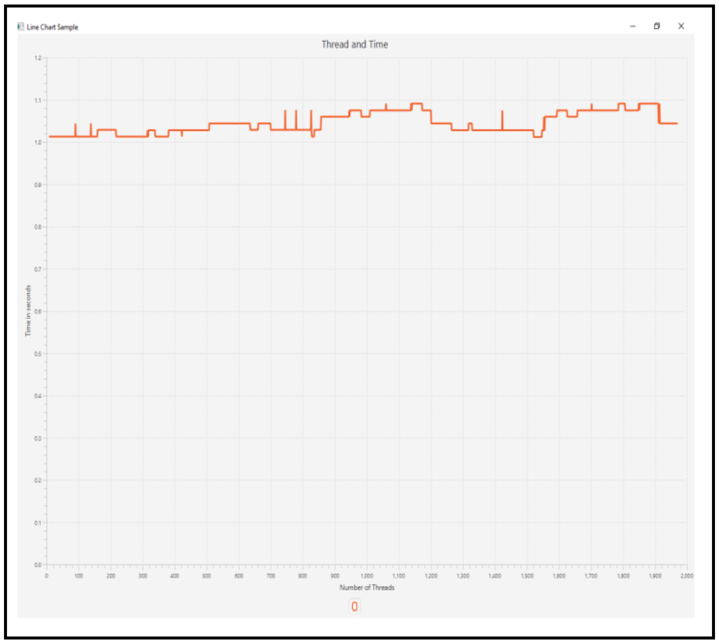


FIGURE 11: THE OPTIMUM NUMBER OF THREADS

5.4.2 Database Management

Database management mainly deals with the saving and deletion of images sent by the smart cars for image analysis and classification. When an image is received by a fog or cloud instance, it saves the image at a particular location in the file system, identified by the car_id, timestamp and geolocation coordinates, and that location is passed to the image analyzer. After the image analysis is completed, the image is placed on a queue for deletion.

5.4.3 Communication Handling

Communication handling functions are under the control of the middleware, but they are not scheduled in the task queues because they run as concurrent threads with the job scheduler. All communications are implemented using TCP/IP sockets.

5.5 Results

Latency for an IoE node is the time interval between the moment when an IoE node sends a service request and when it receives the corresponding response.

TABLE 3: NAVIGATION AND IMAGE PROCESSING ON THE CLOUD

Average Total Image Processing Time (seconds)	**Average Computation Time for Yolo V3 (seconds)**	**Average Computation Time For Tiny Yolo (seconds)**	**Average Computation Time for Rule Based (milliseconds)**	**Average Navigation Time + Network Latency (Edge-Cloud-Edge) (seconds)**
18.93	17.15	1.78	0.03	0.89

Table 3 shows the latencies for all computations (Navigation+Image Processing) done on the Cloud only, in the absence of the Fog instances.

TABLE 4: NAVIGATION ON CLOUD AND IMAGE PROCESSING ON FOG SERIALLY

Average Total Image	**Average Computation**	**Average Computation**	**Average**	**Average Navigation**

Processing Time (in Fog) (seconds)	**Time for Yolo V3(in Fog) (seconds)**	**Time For Tiny Yolo (seconds)**	**ComputationTime for Rule Based (milliseconds)**	**Time + Network Latency (Edge-Cloud-Edge) (seconds)**
8.14	7.5	0.92	0.07	0.18
-56.9%	**-56.26%**	**-48.31%**	**133.33% (time is in the order of ms for this column)**	**-79.77%**

TABLE 5: NAVIGATION ON CLOUD AND IMAGE PROCESSING ON FOG IN PARALLEL

Average Total Image Processing Time (in Fog) (seconds)	**Average Computation Time for Yolo V3 (seconds)**	**Average Computation Time For Tiny Yolo (seconds)**	**Average Computation Time for Rule Based (seconds)**	**Average Navigation Time + Network Latency (Edge-Cloud-Edge) (seconds)**
9.84	9.53	1.05	0.07	0.13

TABLE 6: NAVIGATION ON CLOUD AND IMAGE PROCESSING ON FOG IN PARALLEL WITH QUEUING

Groups	**Average Total Image Processing Time (in Fog) (seconds)**	**Average Computation Time for Yolo V3 (seconds)**	**Average Computation Time For Tiny Yolo (seconds)**	**Average Computation Time for Rule Based (milliseconds)**	**Average Navigation + Queuing + Network Latency (Edge) (seconds)**
Group A high priority requests - low waiting times in queue	17.48	12.69	1.62	0.05	0.12
Group B low priority requests - high waiting times in queue	170.43	17.08	1.59	0.062	42.692

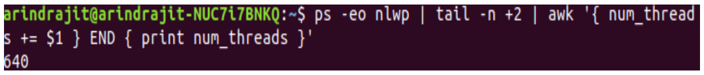
Table 4 shows the average latencies when Image Processing is done on the Fog and the Navigation is done on the cloud (partial data forwarding) in a serial fashion. The average back and forth network latency is found out to be 0.179s. The last row of Table 4 (Navigation in Cloud and Image Processing in Fog) shows the reductions of average latencies in all the columns when compared to those in Table 3 (where all computations are done on the Cloud).

The total wait time of any car request for serial handling is the sum of the latencies of all previous requests that need to be serviced in the time window when the request is active. Hence for the Nth request in an active window, there will be N-1 previous requests that will have to be serviced with a latency (Total Image Processing Time) of 7.5secs per request, on average, for a total wait time of 7.5(N-1) secs before the Nth request will start processing. Hence, the Total Image Processing Time for the Nth request will be 7.5(N) seconds, on average.

Table 5 shows us the values for the average latencies when requests from different cars are handled in parallel. On average, the Total Image Processing Time reduces to 9.84 secs, and this number remains independent of the number of car requests as long as the number of parallel threads does not exceed hardware capacity and start queuing requests.

Table 6 shows the average latencies when the number of requests from cars exceed the multithreading capability of the Fog hardware, so that additional requests are queued. Although each Intel-NUC (used as a Fog node) is capable of handling a huge number threads in parallel (as shown in Fig. 8), when YOLO runs on the Fog instance for Image Processing, around 5 parallel executions of YOLO (corresponding to 5 requests from cars) spawns close to a thousand parallel threads - hence the limit on the number of requests that can be parallely processed, and the increased queuing times for larger number of car requests. This limitation can be overcome using a cloud of NUCs for a particular Fog instance, or using alternate hardware with even greater multithreading capability.

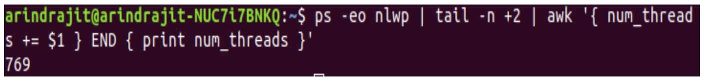
I have prioritized the individual car requests based on the distances between their sources and destinations; fewer the number of intersections in the navigation path, higher the priority of the corresponding car request. I have split the results of Table 6 into 2 groups corresponding to high priority requests (low waiting times in queues) and low priority requests (high waiting times in queues). Note the slight increase in Total average Image Processing time for requests in group A compared to that in Table 5: this is the result of preemptive thread scheduling with waiting threads in the queue, and the larger increase in average Total Image Processing time for requests in group B compared to that in group A: this is the result of both preemptive thread scheduling and waiting times in queues.



```
arindrajit@arindrajit-NUC7i7BNKQ:~$ ps -eo nlwp | tail -n +2 | awk '{ num_thread
s += $1 } END { print num_threads }'
640
```

FIGURE 12: THREAD COUNT BEFORE THE PROCESS IS STARTED

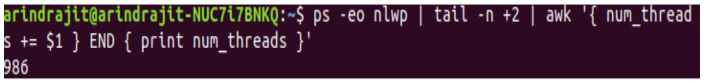
Before the process is started, the total threads running on the system is 640 as shown in Fig. 12.



```
arindrajit@arindrajit-NUC7i7BNKQ:~$ ps -eo nlwp | tail -n +2 | awk '{ num_thread
s += $1 } END { print num_threads }'
769
```

FIGURE 13: THREAD COUNT FOR OPERATION ON A SINGLE CAR

Once YOLO based Image Processing is started for a single car, the thread count is increases to 769 - an increase of about 130 threads per single instance of YOLO.



```
arindrajit@arindrajit-NUC7i7BNKQ:~$ ps -eo nlwp | tail -n +2 | awk '{ num_thread
s += $1 } END { print num_threads }'
986
```

FIGURE 14: THREAD COUNT FOR OPERATION ON EIGHT CARS

Fig. 14 shows that when I simulate 8 cars the thread count goes as high as 986.

CHAPTER 6: BENCHMARK DATASET AND ITS APPLICATION IN NEAR REAL-TIME ACCIDENT IDENTIFICATION AND RELATED CONGESTION CONTROL IN A SPARK AWS EMR CLUSTER

This work focuses on real-time cloud based analytics of live video feeds from the cameras of self-driven autonomous vehicles using the Spark framework on Amazon's Elastic Mapreduce (EMR). We use deep-learning methodologies for real-time object detection and classification on the streamed images, to classify and predict traffic incidences, leading to subsequent congestion control. Results on a benchmark application: traffic congestion aware navigation using 10 self-driving vehicles with their own camera feeds as they drive around in Manhattan; show an 58% improvement in performance on the AWS-EMR based Spark framework, when compared to cloud processing on a single instance of EC2 server on the AWS.

6.1 Streaming Applications with Apache Spark

Spark Streaming is an extension of core Spark that was added to Apache Spark in 2013. It provides scalability, fault-tolerance and high-throughput of data streams that are coming in live. Data consumption is possible through means such as Apache Kafka, Apache Flume, Apache Kinesis or TCP/IP Sockets. The output can be pushed into a file system or any database or via TCP/IP socket.

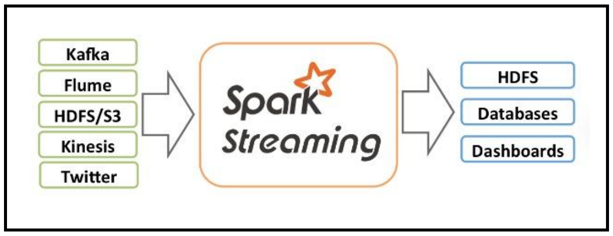


FIGURE 15: SPARK STREAMING

The way this spark streaming works is the data streams are divided into batches and the batches are processed by Spark to generate a final stream of results as shown in Figure 16.

The key abstraction of Spark Streaming is Discretized Stream or DStream which represents a stream of data that is divided into small batches and are built on Spark RDDs. Resilient Distributed Datasets(RDDs) in Spark are collections of elements that can be operated on in parallel. RDDs can be created by parallelizing a collection(data structure) or by referencing a dataset in an external file system like HDFS / HBASE, S3 or Kafka. You will find tabs throughout this guide that let you choose between code snippets of different languages. Fig. 15 and Fig. 16 show an abstract block diagram of Spark Streaming.

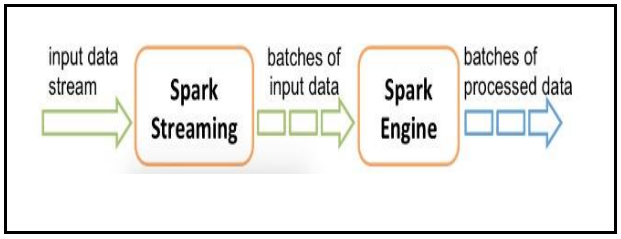


FIGURE 16: DATA STREAMING

6.2 Amazon Web Services

Amazon Web Service(AWS) is basically a service that is provided by amazon as an on-demand platform or more technically a platform as a service(PaaS) to subscribers, companies and governments, on a pay basis. The technology adheres to the needs of the subscribers by providing them with a cluster of servers , available all the time, through the Internet. AWS provides the following : (CPU(s) & GPU(s) for processing, local/RAM memory, hard-disk/SSD storage); operating systems choices; networking; and pre-loaded application software such as web servers, databases etc.

The AWS technology is implemented at amazon data centers throughout the world .The fee is based on the usage of the system and its pay per use. AWS has 6 data centers at North America.

Two of the most popular services of AWS are Elastic Compute Cloud(EC2) and Simple Storage Service (S3). The bulk of AWS services lie in the background and are not exposed to the subscribers and developers, they can avail these services only through API calls.. AWS is accessed over HTTP, using the REST style and SOAP protocol.

Services are billed based on usage, but each service measures usage in different ways. As of 2017, AWS owns a whopping 34% of all cloud (IaaS, PaaS) while the next competitors are namely Microsoft, Google, and IBM have 11%, 8%, 6% respectively according to Synergy Group.

6.2.1 Elastic Map Reduce

Amazon Elastic MapReduce (EMR) is an Amazon Web Services (AWS) service for big data analytics, storage and processing.Amazon EMR offers a scalable, cluster service wherein the subscriber is provided the platform as well as the infrastructure for computation.

Amazon EMR is based on Hadoop which is a mapreduce processing framework. Apache Spark has been a recent add-on over Hadoop. Spark brings in stream processing capabilities for the framework which was initially absent in Apache Hadoop which by default is a batch-processing framework. It was developed at Google for web page indexing and replaced their original indexing algorithms and heuristics in 2004.

Amazon EMR processes big data across a Hadoop based cluster of servers on Amazon Elastic Compute Cloud (EC2) and Amazon Simple Storage Service (S3). in

EMR's name has the term elastic which refers to its dynamic resizing ability, which allows it to ramp up or reduce resource use depending on the demand at any given time. Fig. 17 shows a sample AWS based application which integrates a Spark engine on an EMR cluster with the S3 storage buckets (note: HDFS, RDS, dynamo DB and other AWS storage can also be instantiated to work with EMR).

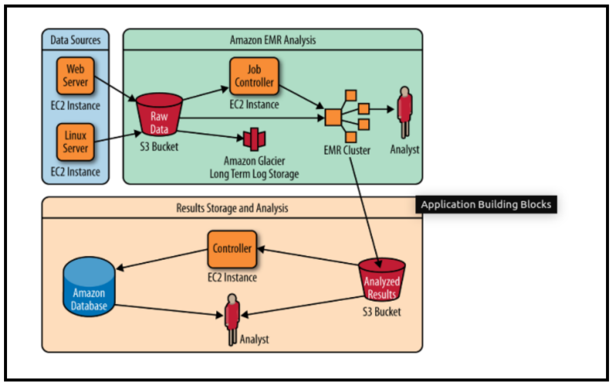


FIGURE 17: AWS EMR

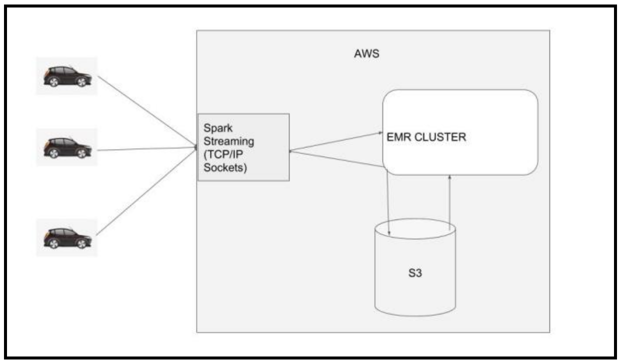


FIGURE 18: PROPOSED APPROACH

Fig. 18. Shown above shows the block diagram of my proposed approach in an abstract manner.

Running the framework mentioned in Chapter 5 over a single Amazon AWS EC2 instance and then over a 2 Node EMR Cluster I have the following results.

6.3 Results

I perform a comparison between running the jobs in an EC2 instance versus running it over a 2 node EMR cluster on Amazon. As an EC2 instance I have chosen the configuration of t2.micro (Variable ECUs, 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1 GiB memory, EBS only)

TABLE 7: NAVIGATION AND IMAGE PROCESSING ON THE EC2 INSTANCE

Average Total Image Processing Time (seconds)	**Average Computation Time for Yolo V3 (seconds)**	**Average Computation Time For Tiny Yolo (seconds)**	**Average Computation Time for Rule Based (milliseconds)**	**Average Navigation Time + Network Latency (Edge-Cloud-Edge) (seconds)**
18.93	17.15	1.78	0.03	0.89

Table 7 shows the benchmark average of the total image processing time, along with processing times for both the deep learning YOLO models, the computation times of the rule based decision tree, as well as the (navigation time + network latency) on the EC2 instance.

TABLE 8: NAVIGATION ON CLOUD AND IMAGE PROCESSING ON A 2 NODE EMR CLUSTER

Average Total Image Processing Time (seconds)	**Average Computation Time for Yolo V3(seconds)**	**Average Computation Time For Tiny Yolo (seconds)**	**Average Computation Time for Rule Based (milliseconds)**
8.00	7.0	0.5	0.01
-57.51%	**-59.18%**	**-71.91%**	**-66.66%**

Table 8 shows the corresponding data for the same algorithm executing in spark deployed on a 2-node AWS based EMR cluster. The nodes in the EMR cluster have the following m4.large configuration: 4 vCore, 8 GiB memory, EBS only Storage:32 GiB. Compared to the single EC2 based computation, the 2-node Spark based EMR cluster reduces the image processing time by around 60%, on average.

CHAPTER 7: REAL-TIME TRAFFIC MANAGEMENT OF AUTONOMOUS VEHICLES USING INTER-FOG COMMUNICATION ON 4G LTE NETWORKS

This paper focuses on developing an Inter-Fog data communication mechanism that was not addressed in [61], [62] & [63] Accident Prediction and Related Congestion Control based on a Fog Computing Architecture. The bandwidth that is actually required for images and video streaming of data from autonomous vehicles to the servers residing on the Fog as well as on the Cloud were only simulated using Wi-Fi internet connectivity as was elaborated in [61], [62], [63]. In this research we simulate the entire application using 4G LTE connectivity and compare the results with the Wi-Fi approach. We also propose two novel approaches - Predictive and Reactive - that control the handover of the data and metadata of the Terminal Nodes (the autonomous vehicles) between the Fog Instances.

7.1 Wi-Fi and LTE NETWORKS

The Internet has become a daily necessity for everyone. Mobile Broadband/LTE and Wi-Fi are mediums to access the internet. Wi-Fi is a wireless networking protocol based on IEEE 802.11 standard. It is the most popular approach in wireless data communication. Wireless networking is often synonymous with Wi-Fi technology but it is a misconception as wireless networking is much broader in nature. Wi-Fi is a trademark of the Wi-Fi Alliance – a consortium involved with wireless LAN technologies and products.

The main requirement of Wi-Fi is that there should be a device like a router, phone or computer that can transmit the signal to nearby devices within a range. A router is a device that transmits signals that come from outside the network like an Internet Service Provider(ISP), to nearby devices that are in reach.

LTE stands for Long Term Evolution and is a 4G - 4th generation wireless broadband network standard. It offers a higher bandwidth with faster speeds suitable for voice calls (VoIP) and multimedia streaming. It is very suitable for bandwidth-hungry applications on mobile devices.

LTE is a 4G technology using radio waves, unlike 3G and WiMAX that uses microwaves. LTE has a better penetration in remote areas and has greater coverage span. It uses Single-Carrier Frequency Division Multiple Access Scheme(SC-FDMA) in its uplink and Orthogonal Frequency Division Multiple Access (OFDMA)in downlink and 64 QAM modulation scheme. OFDMA utilises channel resources efficiently therefore increasing the capacity of total number of users. It is an architecture of distributed intelligence among base stations called eNodeB which are interconnected by interface named X2 and connected to the core by interface named S2. This distributed architecture allows User End(Ue) devices in motion to connect to the network with less handover delay and faster connection setup. Enhanced Packet Core(EPC) is the core of the architecture consisting of Mobility Management Entity(MME) that manages authentication, sessions and keeps a track of users. To route the data packets through the user network, Serving Gateway(S-GW) is used. One of the reasons for the success of LTE is that it can support 2G and 3G because it uses Evolved Packet Core which combines voice and data on an Internet Protocol(IP) unlike earlier architectures which had circuit switching for voice and packet switching for data. The interface between the LTE network and other packets is Packet-Data Node Gateway(PGW) which manages Quality of Service(QoS). This architecture is described in Fig. 19 [64] below.

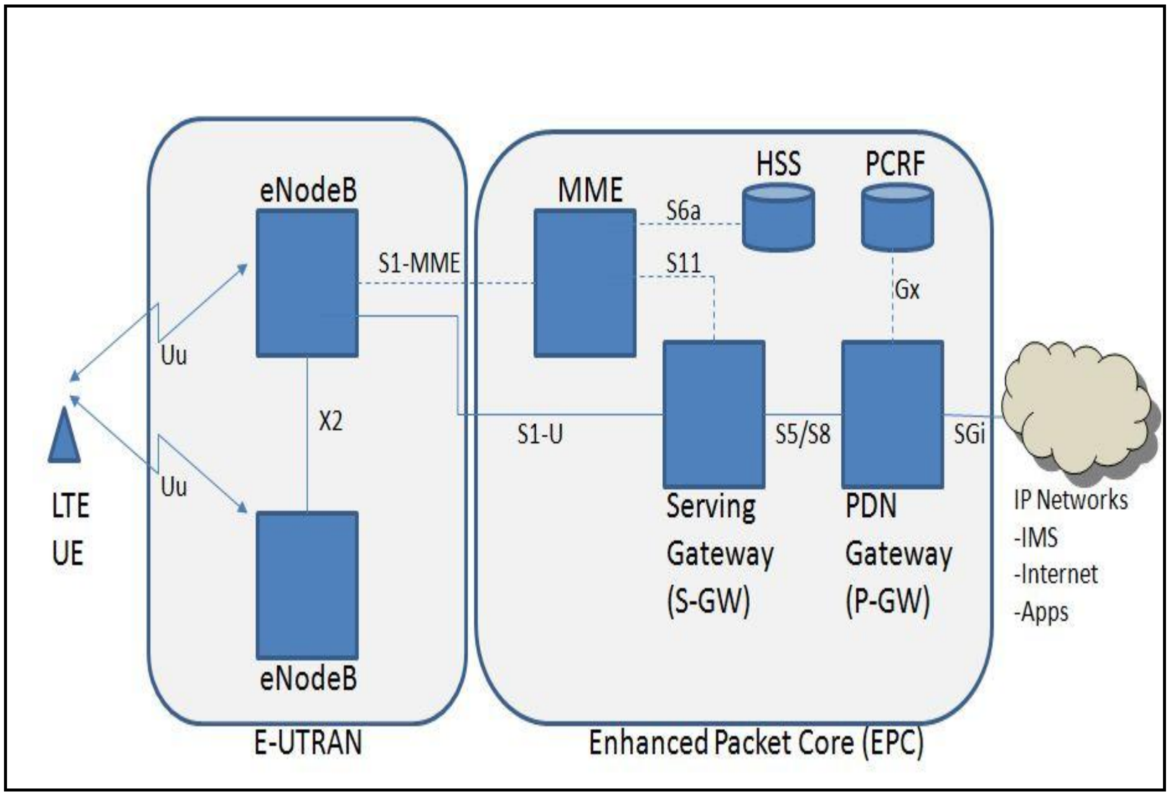


FIGURE 19: LTE ARCHITECTURE [64]

7.2 Related Work

As illustrated in the works [61] and [63], the main goal was to predict accidents and traffic incidences and rerouting based on that received information using live video feeds from autonomous vehicles. The predictions and image object detections are all done in real-time from live video feeds for predicting traffic incidences. Rerouting vehicles based on the received information is done in near and near-real-time. The workflow and model development of the application algorithm has been shown in Figure 20 below [61]. For the Deep Learning part, the methodology that was used was a real-time state-of-the-art Image Objection Detection Algorithm called YOLOv3 [59],[60]. It has 106 layers that are fully-convolutional. It is a modification over normal Convolutional Neural Networks to detect objects from images.

Two instances of the YOLO model were used: one to predict the congestion and second to detect/predict an accident. The database for the accident prediction was assembled and open-sourced [62].

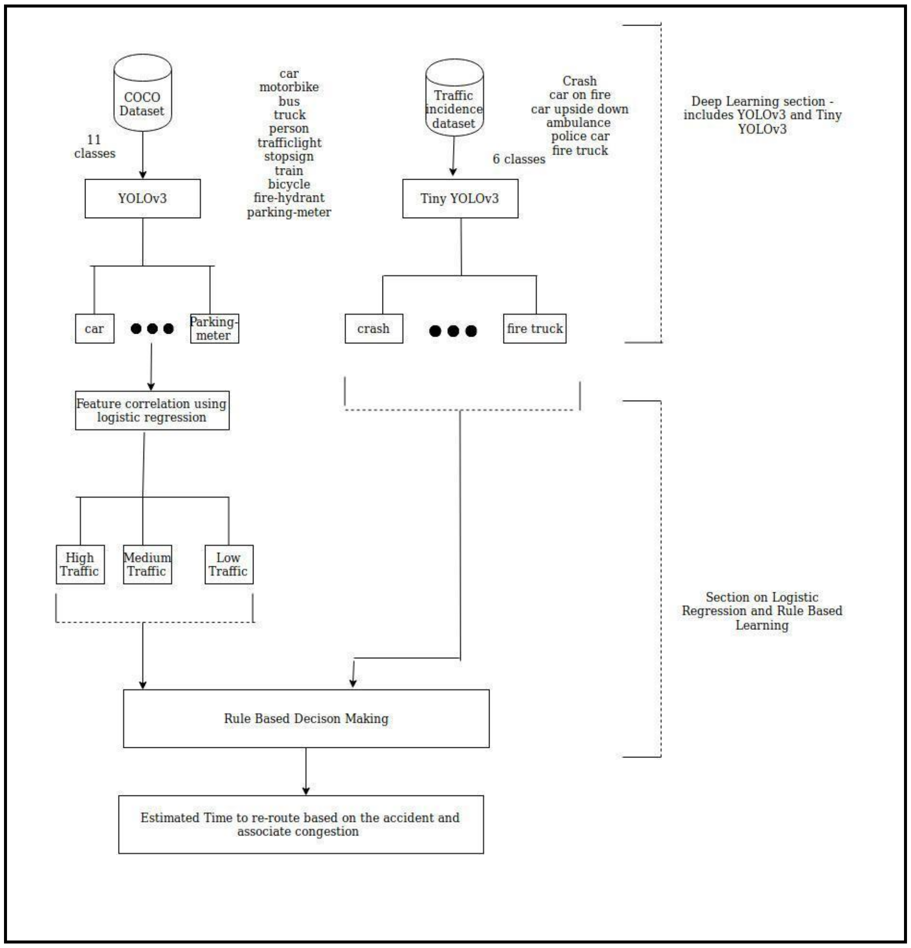

COCO Dataset
11 classes
car
motorbike
bus
truck
person
trafficlight
stopsign
train
bicycle
fire-hydrant
parking-meter
YOLOv3
car
Parking-meter
Feature correlation using logistic regression
High Traffic
Medium Traffic
Low Traffic
Traffic incidence dataset
Crash
car on fire
car upside down
ambulance
police car
fire truck
6 classes
Tiny YOLOv3
crash
fire truck
Deep Learning section - includes YOLOv3 and Tiny YOLOv3
Section on Logistic Regression and Rule Based Learning
Rule Based Decison Making
Estimated Time to re-route based on the accident and associate congestion

FIGURE 20: MODEL AND WORKFLOW DEVELOPMENT [61]

7.3 Our Proposed Algorithm

Predictive Approach

In this approach, a TN recognizes its proximity to the boundary of a VC by evaluating the Received Signal Strength Indicator (RSSI) from its supervising FI. As a result, the TN sends out broadcast packets to all neighboring FIs listening over a dedicated port. The Cloud Server oversees the FIs and is aware of the route of the TN, along with the identity of the VC it would most likely be traversing. The FI in charge of that VC will be notified by the Cloud Server and it will respond back to the TN with its IP address, proclaiming that it is the next server which the TN must communicate with. The data and metadata from one FI are transferred to the next FI through the Cloud. Figure 21 shows the Data Flow Graph of the above approach.

Reactive Approach

We assume that the reactive approach works simultaneously with the predictive approach. Figure 22, explains the reactive approach where a TN starts traversing a VC whose FI is unaware of that TN. This can happen either because of misprediction in the predictive approach above, or because of network failure or high Turn Around Time (TAT) in the cloud implementing the predictive approach. In the reactive approach, each FI runs NMap periodically to discover new MACs in its VC. In this situation, the FI will find the unexpected TN in its VC and inform the Cloud. The Cloud updates the routing information and informs the original FI chosen (as a result of a prior prediction), that the TN won’t be travelling through its supervised VC. This FI then sends the data and metadata of the TN to the Cloud, and deletes the MAC address of the TN from it’s list. Thereafter the cloud sends the data associated with the TN to the new FI along with the new route. The new FI then sends this route to TN.

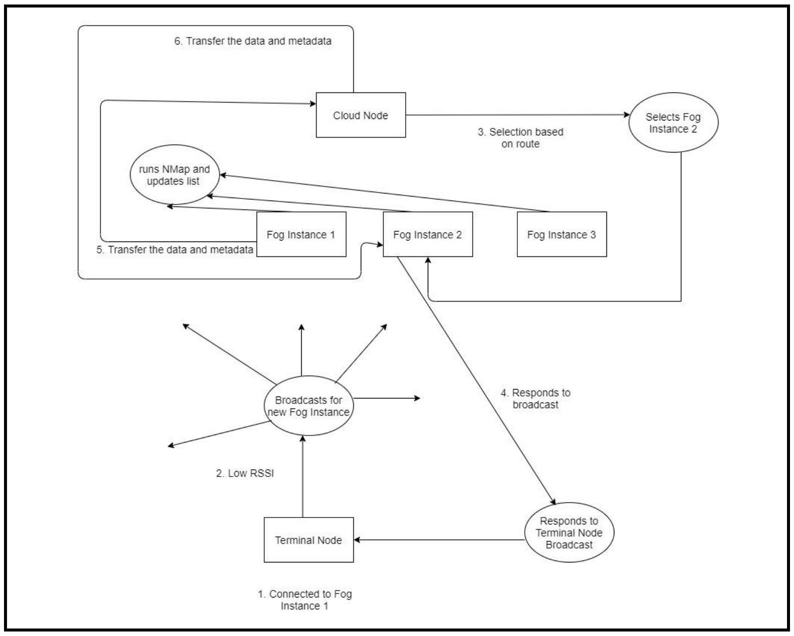

6. Transfer the data and metadata
Cloud Node
3. Selection based on route
Selects Fog Instance 2
runs NMap and updates list
Fog Instance 1
Fog Instance 2
Fog Instance 3
5. Transfer the data and metadata
Broadcasts for new Fog Instance
4. Responds to broadcast
2. Low RSSI
Terminal Node
Responds to Terminal Node Broadcast
1. Connected to Fog Instance 1

FIGURE 21: DFD OF THE PREDICTIVE APPROACH

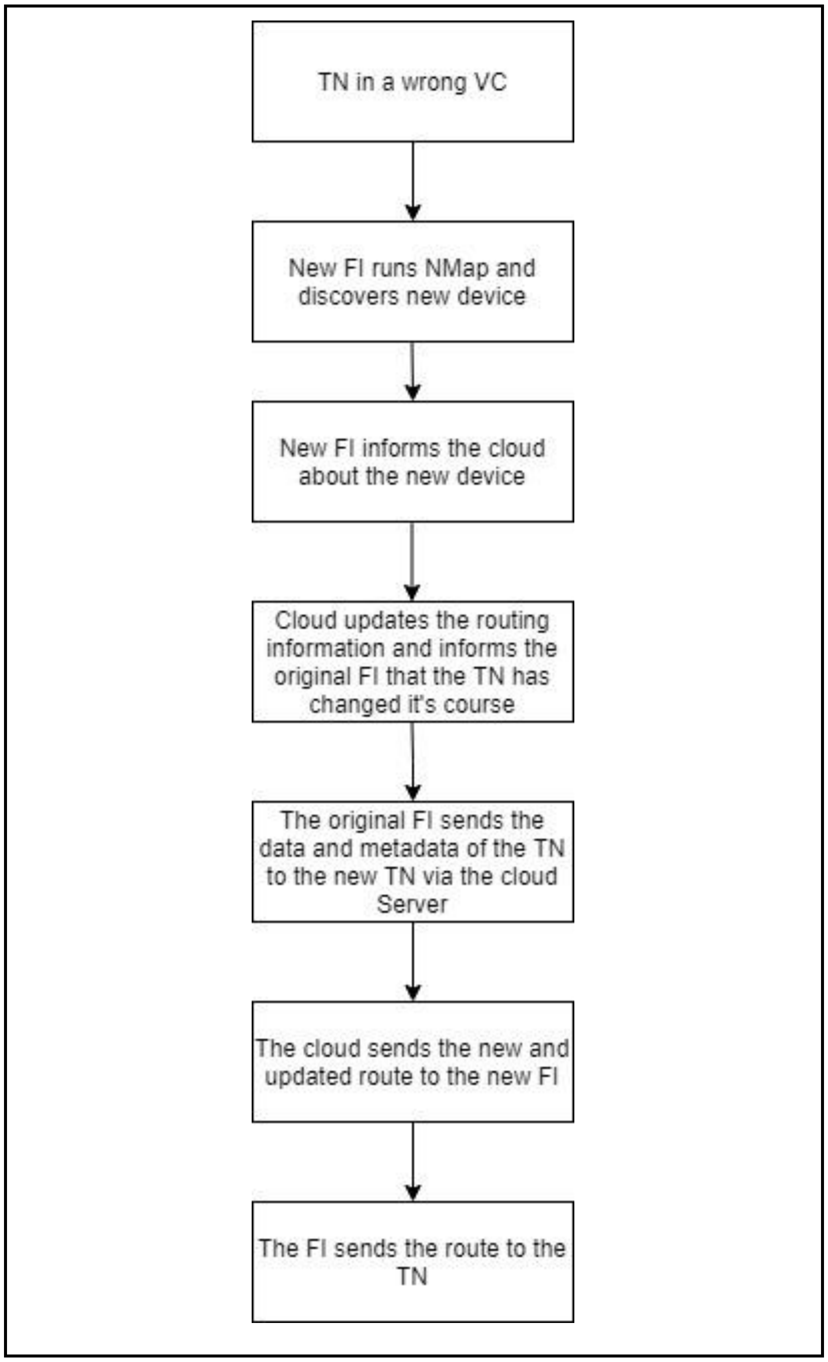

TN in a wrong VC
New FI runs NMap and discovers new device
New FI informs the cloud about the new device
Cloud updates the routing information and informs the original FI that the TN has changed it's course
The original FI sends the data and metadata of the TN to the new TN via the cloud Server
The cloud sends the new and updated route to the new FI
The FI sends the route to the TN

FIGURE 22: BLOCK DIAGRAM OF THE REACTIVE APPROACH

7.4 Results

Experimental Setup

For wifi- internet connectivity setup we have used Raspberry Pi boards running 10 parallel threads as Edge Nodes (TNs). We have used Intel NUCs equipped with 16

GB RAM, DDR4, i7 - 3.5 GHz, 512 GB SSD hard drive facilitating wireless dual band (802.11ac) support, as Fog Instances. We have used the data center at the Cyber Physical Systems Lab at UNCC equipped with 39 nodes as the Cloud.

For simulating the architecture for LTE, we have used NS-3 which is a discrete-event network simulator for Internet-systems[9]. We simulated two network models, (1) Cloud Based Architecture - where the data flows from TN to cloud and back and (2) Fog Based Network Architecture - Data flows across TN -> FI -> Cloud. In the first model, the image processing and computation takes place in the cloud server and in the second case, it takes place in the FI. The simulated architecture consists of LTE and EPC.

For simplicity, we have used 10 nodes as autonomous vehicles (TNs). The application sending data in NS-3 is working on top of User Datagram Protocol (UDP). The average image processing time is added as delay for both the models. For calculating the average turnaround time, we fixed the positions of the nodes at different positions. The Maximum Transmission Unit (MTU) is set to 1500 bytes which is nothing but the largest packet size. Maximum number of packets in uplink and downlink is 1000000 each with a data rate of 100 Gbps. The distance between two base stations - is 60 meters. Since the average navigation (routing/rerouting) time is negligible, we are ignoring that.

Table 9 shows the average times for all computations and Network Latency when computation is done in Cloud only, in the absence of the Fog instances. We observed that for an ideal case scenario when the terrain - the buildings and the user density in a particular cell is not considered, LTE provided better Turn Around Time (TAT) than wifi internet. The Network Latency is the Turnaround Time for both 802.11ac and LTE. Note the order of magnitude reduction (more than 10 times) in the Network Latency of LTE compared to that of 802.11ac.

TABLE 9: CLOUD ONLY ARCHITECTURE (WITHOUT FOG INSTANCES)

Average Image Object Classification Time (seconds)	**Average Computation Time for Congestion Detection (seconds)**	**Average Computation Time For Accident Detection (seconds)**	**Average Computation Time for Model Output Integration(Decision Tree) (milliseconds)**	**Cloud Turn Around Time (802.11ac) (seconds)**	**Cloud Turn Around Time (LTE) (seconds)**
18.93	17.15	1.78	0.03	0.89	0.063

TABLE 10: INTEGRATION OF THE FOG ARCHITECTURE WITH THE CLOUD

Average Image Object Classification Time in the Fog (seconds)	**Average Computation Time for Congestion Detection (seconds)**	**Average Computation Time For Accident Detection (seconds)**	**Average Computation Time for Model Output Integration(Decision Tree) (milliseconds)**	**Cloud Turn Around Time (802.11ac) (Seconds)**	**Cloud Turn Around Time (LTE) (seconds)**
8.14	7.5	0.92	0.07	0.18	0.082
-56.9%	**-56.26%**	**-48.31%**	**133.33%** **(time is in the order of ms for this column)**	**-79.77%**	

Table 10 shows the average times for all computations and Network Latency when we have a distributed computation utilizing the Fog Computing Framework. On average there is a more than 50% reduction in computation times when using fog, which can be explained by the parallelism realized with multiple FIs. Interestingly, the average computation time for the Rule Based Decision Tree approximately doubled for the FIs. We believe the reason for this could be the process suspension mode of the FI architecture, wherein the main memory is cleared in order to make way for newer processes when run parallely. We also observe that there is an approximate 80% reduction in Network Latency on 802.11ac when we compare with that of Table 1 for a Cloud only framework. The LTE Network Latency is 0.082 secs which is approximately half the 802.11ac latency. This improvement is not as dramatic as the LTE latency improvement over its 802.11ac counterpart for the Cloud only architecture because the Fog architecture’s integration with the Cloud reduces the latency of the 802.11ac network from 0.89secs to 0.18secs (a factor of 5 reduction).

The increase in the TAT when a Fog Computing Architecture is introduced in the LTE mobile broadband network simulation is due to the processing delays that occur on each level of the architecture as a result of the increase in the number of hops.

For the Predictive Approach, the handover time is quite small and mostly equivalent to 0.001 seconds. For the Reactive Approach, the handover time is 1.005 seconds on an average for 802.11ac. The time in seconds for LTE is almost similar to that of 802.11ac.

7.5 Discussion

The results show us that there is (a) an order of magnitude (approximately 10 times) reduction in the TAT for a cloud only architecture when we use LTE over 802.11ac, and (b) a factor of 2 reduction in TAT when we use LTE over 802.11ac for a Fog based architecture. Moreover the Fog-Cloud integration achieves more than 50% reduction in computation time over that of the Cloud only architecture, on average. This paper also highlights two other approaches viz. Predictive and Reactive through which the system software provides Rapid Mobility in a roaming wifi as well as on an LTE environment. Fault tolerance is also achieved as the application provides a mechanism to recover the TN when it falls under an unintended VC.

CHAPTER 8: ACCIDENT PREDICTION AND CONGESTION CONTROL USING SPARK STREAMS THROUGH KAFKA IN A FOG COMPUTING BASED IOT NETWORK ARCHITECTURE

8.1 Spark Streams

Spark Streaming is built on top of core Spark that was added to Apache Spark in 2013. It provides scalability, fault-tolerance and high-throughput of data streams that are coming in live. Data consumption is possible through means such as Apache Kafka, Apache Flume, Apache Kinesis or TCP/IP Sockets. The output can be pushed into a file system or any database or via TCP/IP socket.

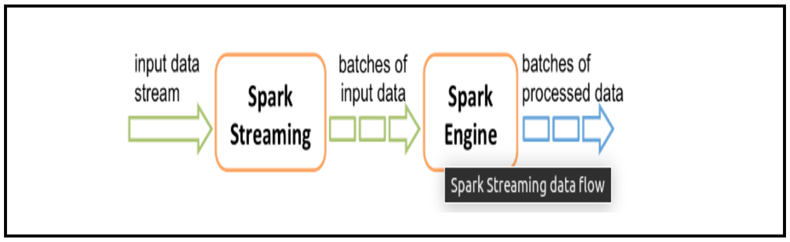
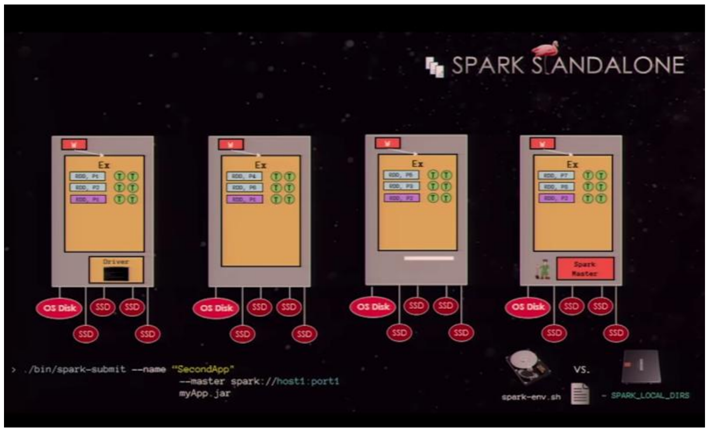


FIGURE 23: SPARK STREAMING DATA FLOW

The way this spark streaming works is the data streams are divided into batches and the processing takes place on these batches to give the final stream of results as shown in Figure 23.

The key abstraction of Spark Streaming is DStream or Discretized Stream which signifies a stream of data that is divided into small batches and are built on Spark RDDs. Resilient Distributed Datasets(RDDs) in Spark are collections of elements that can be operated on in parallel. RDDs can be created by parallelizing a collection(data structure) or by referencing a dataset in an external file system like HDFS / HBASE, S3 or Kafka.

There are worker nodes that work on partitions of an RDD parallely. The spark standalone cluster is picturized in Figure 24.



```
> ./bin/spark-submit --name "SecondApp"
                     --master spark://host1:port1
                     myApp.jar
```

FIGURE 24: SPARK STANDALONE CLUSTER

8.2 Reason for choosing Spark Streaming

For a near-real-time or real-time requirement, a traditional batch processing system like Apache Hadoop is not suitable. For a stream processing system like Apache Storm, a processing of a record is guaranteed if that hasn't been processed but there is an inconsistency in Storm wherein a repetition of a record might be there. Also in Apache Storm the state is lost if a node running Apache Storm goes down. Mostly, people use Apache Hadoop for batch processing and Apache Storm for Stream Processing which cause an increase in the size of the code, increase in the number of bugs to fix, a longer learning curve, a lot of developmental effort and so on.

Spark Streaming is advantageous since it helps in fixing the aforementioned issues and it provides scalability, efficiency, resilience along with a batch processing system. This makes it very easy for a developer as only a particular framework needs to be learnt in order to work with both batch and stream processing.

Spark tasks are assigned dynamically to the worker nodes on the basis of locality of data and resources that are available. In traditional models the load is statically allocated to the worker nodes.

Instead of assigning tasks statically to worker nodes like a continuous operator model does, Spark Streaming is dynamic and assigns jobs to workers based on data locality and available resources. This helps in load-balancing and faster fault recovery.

8.3 Advantages of Spark Streaming

Dynamic load balancing- The data is divided into smaller micro-batches which aids in a more equal allocation of computations. The executor counts can be increased at runtime depending on the application’s computation needs.

Fast failure and straggler recovery- Older systems needed to restart the failed computation on another node. One node was responsible for handling the recomputation which generally stalled the pipeline. Spark creates a more uniform distribution of tasks that can run anywhere on the cluster. Even for failed tasks, they can be distributed evenly on all kinds of nodes which has been proven to be faster than the traditional approach.

Performance- Spark Streaming’s ability to use the spark core engine by batching data leads to higher throughput to other contemporary approaches. Latencies can be as low as a few hundred milliseconds.

8.4 Apache Kafka

Apache Kafka was originally conceived at LinkedIn, later becoming an open-source project at Apache in 2011. Apache Kafka is written in Scala and Java and is a publish-subscribe message queuing system.

Kafka is made for a clustered high throughput system. The advantages that

Apache Kafka provides is that it has a built-in partitioning, replication, fault-tolerant and better throughput.

8.4.1 Publish-Subscribe Messaging System

In this kind of a system, messages are persisted on queues that are called topics. Consumers or Subscribers can actually subscribe to one or many topics and pull messages from that topic. Publishers are also called Producers of the messages.

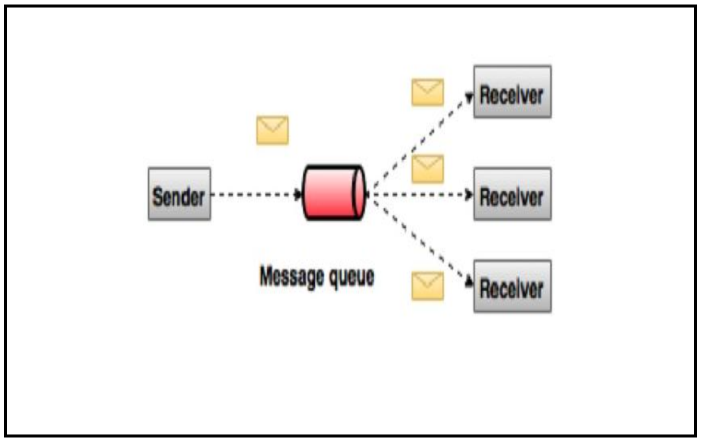


Figure 25: Publish Subscribe Messaging System

Kafka's cluster management system relies on Zookeeper. The messages in Kafka are saved on the disks and replicated within the Kafka cluster. Kafka is suitable for online and offline message consumption as each message is offsetted inside a topic.

The advantages of Apache Kafka lies in the fact that it is blazingly fast performing 2 million writes per sec. Kafka puts everything into the disk which essentially means that all the writes go to the OS(RAM) page cache. The phenomenon of Sequential I/O takes place where since the Kafka messages are in an ordered fashion are read ahead by the OS and persisted on the disk/page cache. So the application doesn't need disk seeking latencies, also no additional logic needs to be written for

Sequential I/O as the OS generally keeps the disk cache on the free memory.

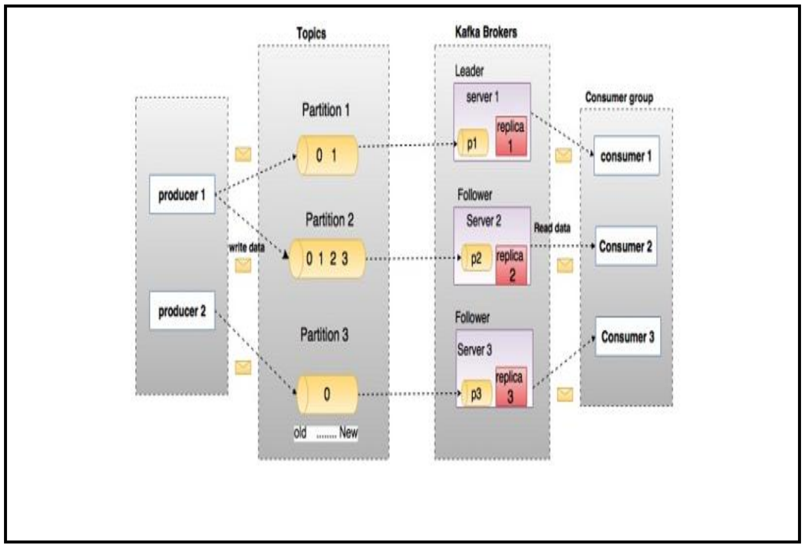


FIGURE 26: APACHE KAFKA'S ARCHITECTURE

In the above Figures 25 and 26, we describe a publish subscribe model along with a proper example of such a model namely Apache Kafka.

In figure 26 a topic is split into three partitions. Partitions have individual offset numbers associated with them.

Kafka creates a replication for each partition of a topic. For load balancing in a cluster, each broker stores one or more of those partitions.

8.5 Apache Cassandra

Apache Cassandra is an open-source, decentralized and distributed NoSQL database. It has no single point of failure.

Cassandra was developed and open-sourced by facebook. Apache accepted it as an incubator project in 2009. It was made a top-level project in 2010. Cassandra provides certain distinct advantages to it's users as :

Scalability - It allows us to add more hardware to accommodate more users and more data.

Always on Architecture - It has no single point of failure and it is best for applications that cannot afford a failure.

Flexible Database - It allows all kinds of data formats to be stored(structured, semi-structured, unstructured).

Replication - Data is replicated across nodes in the cluster, across multiple datacenters.

ACID Property - Cassandra supports properties like Atomicity, Consistency, Isolation and Durability just like a Relational Database System.

Fast Writes - Performing blazingly fast writes, Cassandra can store hundreds of terabytes of data.

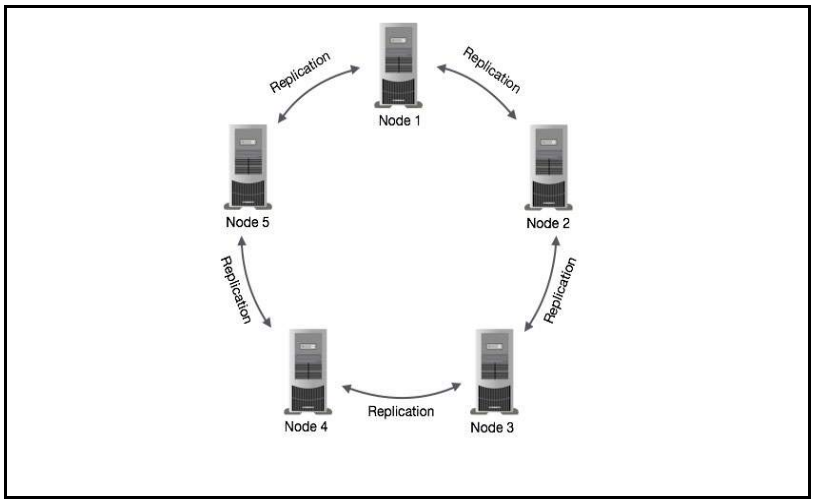


FIGURE 27: APACHE CASSANDRA CLUSTER

As is shown in the figure 27 above, Cassandra has a peer-to-peer distributed system where the data is stored across its nodes. Any node can receive the read/write request regardless of where the data is located in the cluster. Cassandra uses a kind of Gossip Protocol in the background for the nodes to intercommunicate with each other.

8.5.1 Storage in Cassandra

The Storage Components in Cassandra are mentioned above in the figure. The partitions are made in the above figure based on references of the query and they are divided into two groups:-

a) Memory Store - This contains most notably MemTables (it's a caching mechanism for cassandra in the main memory). Any action that has to be performed will be performed on the MemTable first, after that syncing happens to the disk. Bloom Filters(To test whether an element is present in a set in a probabilistic way), Index Summary(Index of the original index that is present on disk), Key Cache(to store primary keys and row offsets), Row Cache(store sets of rows)

b) Disk Store - This contains Commit Logs(crash recovery mechanism in Cassandra, every write operation is written to it after the memory's Commit Log is written to), SSTable (stores Bloom Filters, Indices and the original data). The figure 28 below shows a block diagram of the above theory.

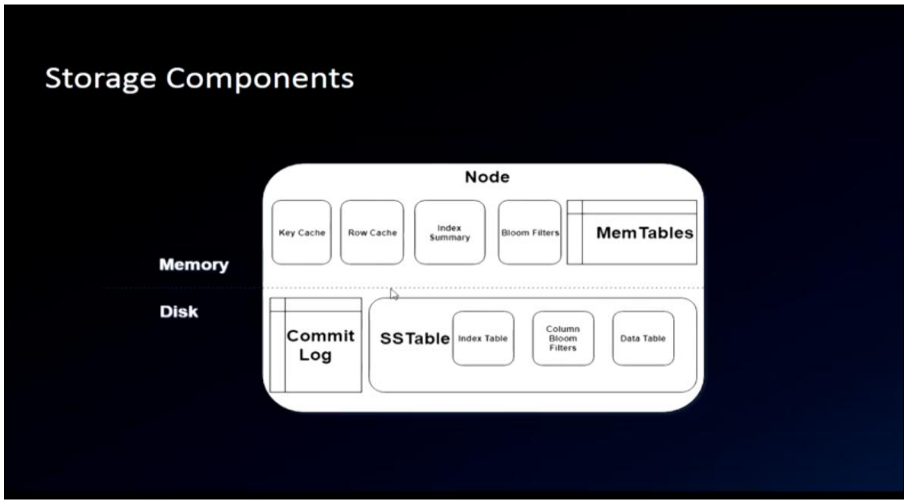


FIGURE 28: STORAGE COMPONENTS IN CASSANDRA

8.6 Our implementation of the Fog Computing Architecture

For our implementation of the Fog Computing Architecture we have simulated autonomous cars in each virtual cluster. The cars supply their initial source and destination and that reaches the Cloud via the Fog. The Cloud handles the routing aspect of the application and the Image Object Detection is performed by the Fog Instances. The figure 29 describes the architecture below.

Fog Routing Server - The Fog Routing Server is principally concerned with sending the routing information to the Cloud and back to each car.

Cloud Edge Server - The Cloud Edge Server receives the routing information initially from the Fog Routing Server, it pushes the information as a Kafka Producer into a Kafka topic in our architecture. It also opens up another channel for receiving the FI's results and updating the connected graph which is on Cassandra (cloud).

Cassandra Cluster (Fog) - There are two nodes Cassandra Cluster in each Fog Instance. Cassandra cluster is principally concerned here with the storage of images as BLOB (Binary Large Objects) type.

Kafka Queue (Fog) - The message queue in the Fog is concerned with queuing images as byte arrays(message) along with the (MAC + IP) as the key.

Spark Standalone Cluster (Fog) - The cluster that has been set up on each instance of the Fog is a two node Spark standalone cluster. This cluster is concerned with performing image object detection on Spark Streams using two models of a deep-learning algorithm called YOLO trained on different datasets. It also performs feature correlation (Logistic Regression) and rule-based decision making.

Spark Standalone Cluster (Cloud) - The cloud comprises five servers acting in unison to form a five node standalone spark cluster. The cloud is responsible for routing of the autonomous vehicles based on the results of the FI's estimation. Spark is responsible for carrying out the route processing and adjusting the edges of the weighted graph (for the Lower Manhattan map). Spark streaming pulls messages from the Kafka queue on the cloud.

Kafka Queue (Cloud) - The message queue in the cloud is concerned with sending out route information to the Spark Cluster as key and message pairs. Key here is the (MAC + IP) and message is a normal String value with initial source and destination.

Cassandra (Cloud) - Storing of the connected graph as a text file (BLOB) in Cassandra. Spark uses it every time it pulls a new routing information from Kafka.

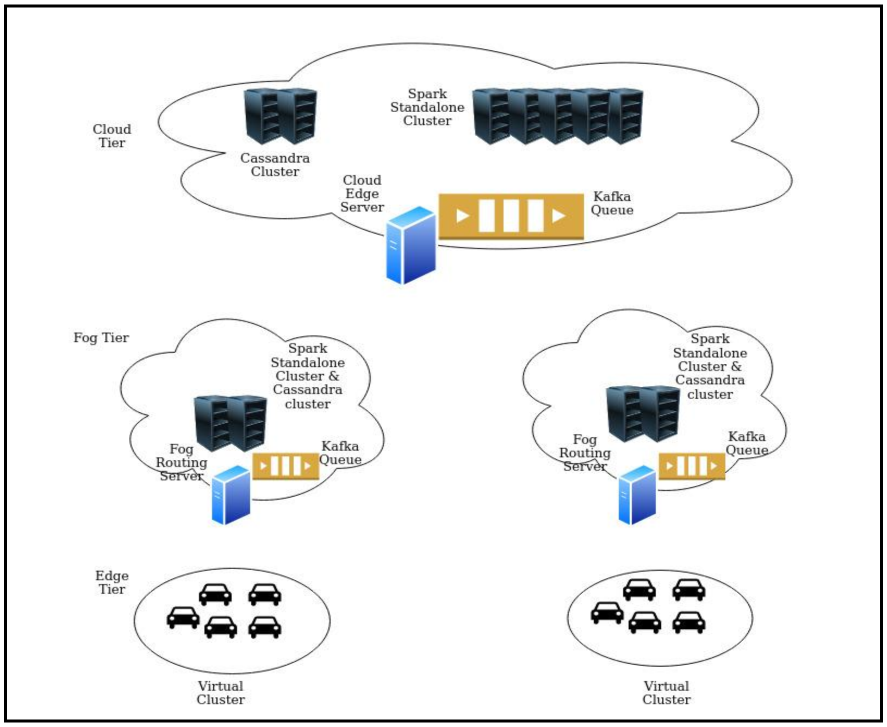


FIGURE 29: FOG COMPUTING ARCHITECTURE

8.7 Deep-Learning and the Application Benchmark

8.7.1 Deep learning and Modeling

The main goal of this analysis is to predict accidents and traffic incidences and rerouting based on that received information using live video feeds from autonomous vehicles. The predictions and image object detections are all done in real-time from live video feeds for predicting traffic incidences. Rerouting vehicles based on the received information is done in near and near-real-time. Figure 30 shown below presents a block diagram of the underlying workflow of the deep learning and subsequent machine learning based model development [61].

Deep Learning : Significant improvements in object detection and object recognition have only been due to the advancement of Convolutional Neural Networks(CNNs). The original YOLOv3 [59], [60] has been trained on a Microsoft Common Objects in Context (COCO) dataset with 80 different objects, mostly unrelated to traffic. To classify the data, we are using two CNN architectures: YOLOv3.11 (for 11 classes related to traffic congestion) for traffic object identification and tiny YOLOv3.6 (for 6 classes related to traffic incidences).

YOLOv3.11 is one of the faster object detection algorithms and due to its fast performance and high accuracy, it has been used in Fig.3 for object detection. It has 106 layers, fully-convolutional with residual skip connections and up sampling. Detection is done by applying 1x1 detection kernels on feature maps of three different sizes at three different places in the network. It is trained on the COCO dataset. We have used the classes: ‘car’, ‘motorbike’, ‘bus’, ’truck’, ’person’, ‘traffic light’, ‘stop sign’, ‘train’, ‘bicycle’, ‘fire-hydrant’, ‘parking-meter’ out of the 80 classes in the COCO dataset for our object detection module.

The shape of the detection kernel is 1x1x(Bx(5 + C)). Here B is the number of bounding boxes a cell on the feature map can predict, C is the no. of classes. For our architecture we took 11 classes, B = 3 and C = 11, so the kernel size is 1 x 1 x 48.

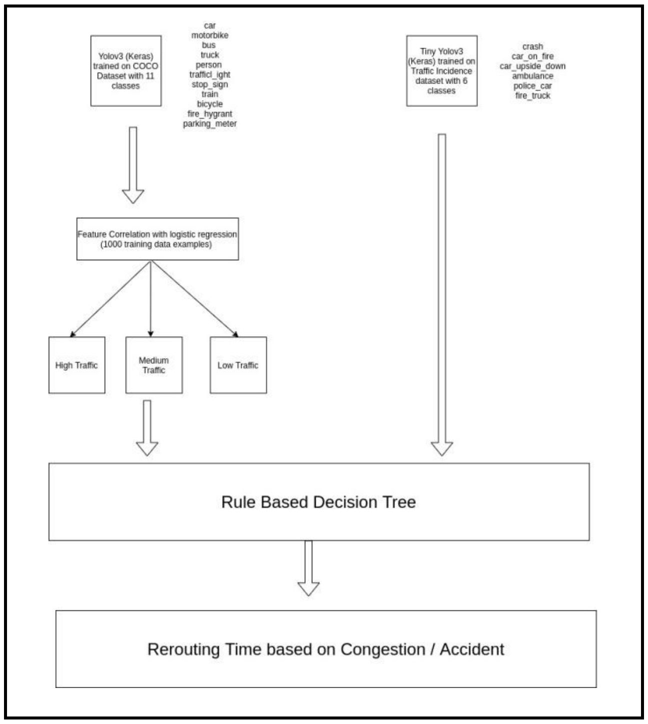

Yolov3 (Keras) trained on COCO Dataset with 11 classes
car
motorbike
bus
truck
person
trafficl_ight
stop_sign
train
bicycle
fire_hygrant
parking_meter
Tiny Yolov3 (Keras) trained on Traffic Incidence dataset with 6 classes
crash
car_on_fire
car_upside_down
ambulance
police_car
fire_truck
Feature Correlation with logistic regression (1000 training data examples)
High Traffic
Medium Traffic
Low Traffic
Rule Based Decision Tree
Rerouting Time based on Congestion / Accident

FIGURE 30 : WORKFLOW OF THE APPLICATION BENCHMARK

On the predicted outputs, we have used feature correlation using logistic regression from the first network to assign the input image in any one of the 3 target classes: high congestion, medium congestion and low congestion. We will measure the degree of congestion from these three classes.

Dataset to Test: For the image database that is required for learning, we have recorded a video of driving for 20 hours around places like Charlotte, North Carolina and Lower Manhattan, New York City since image databases are not easily available or open-sourced. To extract relevant images from the video data, we have used a video-to-image converter. A total of 10876 images were extracted for testing. For resource constrained environments in the second network [3], we chose tiny YOLOv3.6 for detection. The architecture of YOLOv3.6 is similar to YOLOv3 having 23 layers. It has the ability of better training on smaller datasets. It detects 6 classes: police car, ambulance, crash, car on fire, car upside down, and fire truck. Our designed dataset contains approximately 600 images of the above mentioned classes which we have used to train. Apart from these images, there are 150 test images.

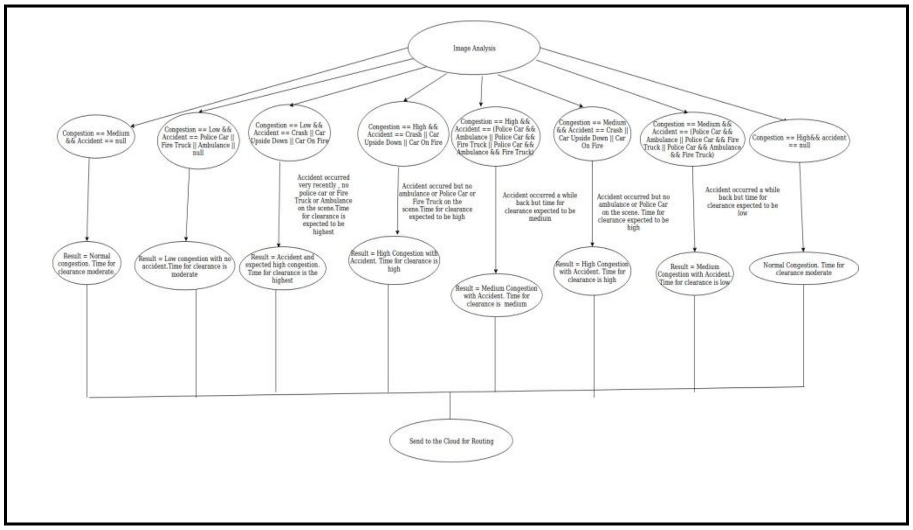


Image Analysis
Congestion == Medium && Accident == null
Congestion == Low && Accident == Police Car || Fire Truck || Ambulance || null
Congestion == Low && Accident == Crash || Car Upside Down || Car On Fire
Congestion == High && Accident == Crash || Car Upside Down || Car On Fire
Congestion == High && Accident == (Police Car && Ambulance || Police Car && Fire Truck || Police Car && Ambulance && Fire Truck)
Congestion == Medium && Accident == Crash || Car Upside Down || Car On Fire
Congestion == Medium && Accident == (Police Car && Ambulance || Police Car && Fire Truck || Police Car && Ambulance && Fire Truck)
Congestion == High&& accident == null
Accident occurred very recently , no police car or Fire Truck or Ambulance on the scene.Time for clearance is expected to be highest
Accident occured but no ambulance or Police Car or Fire Truck on the scene.Time for clearance expected to be high
Accident occurred a while back but time for clearance expected to be medium
Accident occurred but no ambulance or Police Car on the scene. Time for clearance expected to be high
Accident occurred a while back but time for clearance expected to be low
Result = Normal congestion. Time for clearance moderate.
Result = Low congestion with no accident.Time for clearance is moderate
Result = Accident and expected high congestion. Time for clearance is the highest
Result = High Congestion with Accident. Time for clearance is high
Result = Medium Congestion with Accident. Time for clearance is medium
Result = High Congestion with Accident. Time for clearance is high
Result = Medium Congestion with Accident. Time for clearance is low
Normal Congestion. Time for clearance moderate
Send to the Cloud for Routing

FIGURE 31: RULE BASED LEARNING

8.8 Rule Based Decision Tree

This is the second stage of the overall feature correlation by which two model predictions (YoloV3.11 and Tiny YoloV3.6) are combined. The structure is shown in figure 31 above.

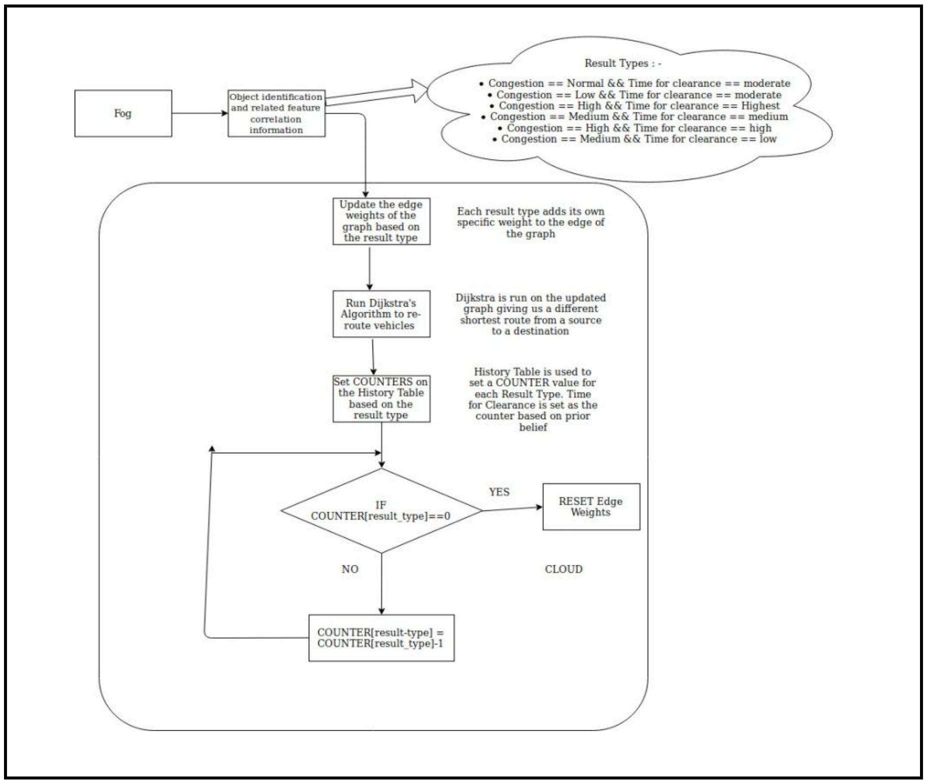

Result Types : -
• Congestion == Normal && Time for clearance == moderate
• Congestion == Low && Time for clearance == moderate
• Congestion == High && Time for clearance == Highest
• Congestion == Medium && Time for clearance == medium
• Congestion == High && Time for clearance == high
• Congestion == Medium && Time for clearance == low
Fog
Object identification and related feature correlation information
Update the edge weights of the graph based on the result type
Each result type adds its own specific weight to the edge of the graph
Run Dijkstra's Algorithm to re-route vehicles
Dijkstra is run on the updated graph giving us a different shortest route from a source to a destination
Set COUNTERS on the History Table based on the result type
History Table is used to set a COUNTER value for each Result Type. Time for Clearance is set as the counter based on prior belief
IF COUNTER[result_type]==0
YES
RESET Edge Weights
NO
CLOUD
COUNTER[result-type] = COUNTER[result_type]-1

FIGURE 32: ROUTING ON THE CLOUD AND HISTORY TABLE UPDATION

8.9 Congestion Awareness and Changes in Routing

The results of the FIs reach the Cloud’s Edge Server. The server makes the necessary change to the BLOB(Binary Large Object) file in the Cassandra cluster on the cloud that stores the overall connected graph for the topology. The consecutive spark jobs that are fired for different cars get the updated value of the overall connected graph. A History Table(HT) is stored into Cassandra database. This portion of Cassandra deals with a more structured form of the data wherein counters are assigned for each edge with a predicted or detected traffic with or without predicted accidents. Leveraging the blazingly high write speeds of Cassandra, this work has been made possible with minimum disk read/write latency.

8.10 Results

TABLE 11: NAVIGATION ON CLOUD AND IMAGE PROCESSING ON THE FOG WITH QUEUING

Groups	Average Total Image Processing Time (in Fog) (seconds)	Average Computation Time for Yolo V3 (seconds)	Average Computation Time For Tiny Yolo (seconds)	Average Computation Time for Rule Based (milliseconds)	Average Computation Time for Logistic Regression (seconds)	Average Navigation + Queuing + Network Latency (Edge) (seconds)
Group A high priority requests - low waiting times in queue	17.48	12.69	1.62	0.05	0.12	0.12
Group B low priority requests - high waiting times in queue	170.43	17.08	1.59	0.062		42.692

TABLE 12: NAVIGATION ON CLOUD AND IMAGE PROCESSING ON THE FOG WITH QUEUING (SPARK - KAFKA - CASSANDRA)

Groups	Average Total Image Processing Time (in Fog) (seconds)	Average Computation Time for Yolo V3 (seconds)	Average Computation Time For Tiny Yolo (seconds)	Average Computation Time for Rule Based (milliseconds)	Average Computation Time for Logistic Regression (seconds)	Average Navigation + Queuing + Network Latency (Edge) (seconds)
Equal Priority Requests	14.81	12.01	2.8	0.07	0.58	2.189

The changes in the result when I have integrated the Spark - Kafka - Cassandra mixture along with my middleware application are due to a myriad of reasons that I am going to explain below.

In my earlier system, that is in a system without Kafka, Cassandra and Spark, the Fog Instances that were mainly Intel NUCs failed if I ran more than 7 or 8 parallel executions of YOLO which spawned more than 1000 parallel high computation threads. In this spark streaming system however, I have reserved four executors on each worker node to facilitate 4 parallel executions of YOLO at a time. The other machine learning algorithms do not take much time to complete so I am only focussing on YOLO at this time. Due to Spark's Lazy Evaluation and In-Memory processing the entire application has been highly optimized.

The second reason why the queueing time has greatly reduced is that in Table 11's approach while one high priority thread that had started running stalled a corresponding low priority thread for a long time, in Table 12's approach the implementation using Kafka-Spark-Cassandra gave better performance without prioritized threads. As soon as the FI predicts the results of an image by performing Image object detection, feature correlation and decision making (decision tree), it updates the corresponding weight of the connected graph that resides in Cassandra. So the next car doesn't have to wait an entire duration of the time while an earlier car is still in transit (critical section problem in the earlier approach).

The third reason is Apache Cassandra's fast read/write. Owing to the structural components discussed earlier like Commit Logs, Index of an Index and MemTables. Mostly the processing is in-memory again rather than disk reads. This saves a lot of time when compared to the earlier approach.

The fourth reason is Apache Kafka's highly fault-tolerant and fast disk reads owing to the Sequential I/O nature of the OS. Here again it is an in-memory processing. The ordered nature of the messages of Apache Kafka makes the OS prefetch the data from the disk to the disk/page cache which resides in the main memory.

The deep-learning and machine learning portions of the code were actually written in python and that is the reason why the codes were a little slower than that which was written in C/C++ for the earlier chapters.

8.11 Conclusion

The goal of my research was to implement an interconnected smart middleware that spans a three tiered network architecture to facilitate better real-time and near-real-time responses for IoT/IoE devices. While trying to disassemble the application benchmark over the tiers, I found that by limiting the data that travels to the cloud we can prohibit the conversion of BigData to Big-Squared-Data that would have easily overwhelmed any powerful cloud architecture. Thus the reason for Fog Computing is justified as we can find a way to save data bandwidth that is travelling to the cloud. After this was achieved I tried to address the other most important issue of real-time and near-real-time responses. I found out during the course of my research that utilizing a clustered framework on both the Fog instances and the Cloud Servers and leveraging the BigData IoT/IoE framework of Apache Spark, I could reduce the latency of computation by a wide margin as is estimated from the results. By utilizing message queues like Apache Kafka and NoSQL databases like Apache Cassandra, I could improve the results even further as is described earlier. I also successfully managed to deploy deep-learning algorithms as external processes in Spark Streaming jobs thereby managing to maintain a constant task-level parallelism that helped in improving the performance. I would like to conclude this dissertation report by reiterating that in tomorrow’s ever-connected world with ubiquitous smart devices that are constantly sending/receiving data back and forth, Fog Computing is the need of the hour. Such an architecture will help mitigate the problems of Big-Squared-Data, it will help reduce the network latency as the entire goal of Fog Computing is to perform the computation as close to the edge device as possible.

REFERENCES

[1] L. Piras. (2014, March) A brief history of the internet of things [info- graphic]. [Online]. Available: http://www.psfk.com/2014/03/internet-of- things-infographic.html

[2] Internet of things market forecast:. Cisco. [Online]. Available:http://postscapes.com/internet-of-things-market-size

[3] (2014, Jan.) Cisco delivers vision of fog computing to accelerate value from billions of connected devices. Press release. Cisco. [Online]. Available:http://newsroom.cisco.com/release/1334100/Cisco-DeliversVision-of-Fog-Computing-to-Accelerate-Value-from-Billions-of-Connected-Devices-utm-medium-rss

[4] L. Zhang, C. Wu, Z. Li, C. Guo, M. Chen, and F. C. M. Lau, "Moving big data to the cloud: An online cost-minimizing approach," IEEE Journal on Selected Areas in Communications, vol. 31, pp. 2710–2721, Dec. 2013.

[5] D. Lund, C. MacGillivray, V. Turner, and M. Morales, "Worldwide and regional internet of things (iot) 20142020 forecast: A virtuous circle of proven value and demand," International Data Corporation (IDC), Tech. Rep., 2014.

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740-741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[7] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

[8] M. P. Mills, "The cloud begins with coal," Digital Power Group, Tech. Rep., August 2013.

[9] D. He and S. Zeadally, "An analysis of rfid authentication schemes for internet of things in healthcare environment using elliptic curve cryptography," IEEE Internet of Things Journal, vol. 2, no. 1, pp. 72–83, Feb 2015.

[10] I. F. Akyildiz, M. Pierobon, S. Balasubramaniam, and Y. Koucheryavy, "The internet of bio-nano things," IEEE Communications Magazine, vol. 53, no. 3, pp. 32–40, March 2015.

[11] A. Aijaz and A. H. Aghvami, "Cognitive machine-to-machine com- munications for internet-of-things: A protocol stack perspective," IEEE Internet of Things Journal, vol. 2, no. 2, pp. 103–112, April 2015.

[12] A. M. Ortiz, D. Hussein, S. Park, S. N. Han, and N. Crespi, "The cluster between internet of things and social networks: Review and research challenges," IEEE Internet of Things Journal, vol. 1, no. 3, pp. 206– 215, June 2014.

[13] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," Future Generation Computer Systems, vol. 29, no. 7, pp. 1645 – 1660, 2013.

[14] J. Jin, J. Gubbi, S. Marusic, and M. Palaniswami, "An information framework for creating a smart city through internet of things," IEEE Internet of Things Journal, vol. 1, no. 2, pp. 112–121, April 2014.

[15] C. Perera, C. H. Liu, S. Jayawardena, and M. Chen, "A survey on internet of things

from industrial market perspective," IEEE Access, vol. 2, pp. 1660–1679, 2014.

[16] J. Wei, "How wearables intersect with the cloud and the internet of things Considerations for the developers of wearables." IEEE Consumer Electronics Magazine, vol. 3, no. 3, pp. 53–56, July 2014.

[17] L. Wang and R. Ranjan, "Processing distributed internet of things data in clouds," IEEE Cloud Computing, vol. 2, no. 1, pp. 76–80, Jan 2015.

[18] X. Zheng, P. Martin, K. Brohman, and L. D. Xu, "Cloudqual: A quality model for cloud services," IEEE Trans. on Industrial Informatics, vol. 10, no. 2, pp. 1527–1536, May 2014.

[19] H. Yue, L. Guo, R. Li, H. Asaeda, and Y. Fang, "Dataclouds: Enabling community-based data-centric services over the internet of things," IEEE Internet of Things Journal, vol. 1, no. 5, pp. 472–482, Oct 2014.

[20] M. Nitti, R. Girau, and L. Atzori, "Trustworthiness management in the social internet of things," IEEE Trans. on Knowledge and Data Engineering, vol. 26, no. 5, pp. 1253–1266, May 2014.

[21] X. Zheng, P. Martin, K. Brohman, and L. D. Xu, "Cloud service negotiation in internet of things environment: A mixed approach," IEEE Trans. on Industrial Informatics, vol. 10, no. 2, pp. 1506–1515, May 2014.

[22] S. U. Khan, P. Bouvry, and T. Engel, "Energy-efficient high-performance parallel and distributed computing," Journal of Supercomputing, vol. 60, pp. 163–164, 2012.

[23] K. Bilal, S. U. R. Malik, S. U. Khan, and A. Y. Zomaya, "Trends and challenges in cloud datacenters," IEEE Cloud Computing, vol. 1, no. 1, pp. 10–20, May 2014.

[24] Q. Duan, Y. Yan, and A. V. Vasilakos, "A survey on service-oriented network virtualization toward convergence of networking and cloud computing," IEEE Trans. on Network and Service Management, vol. 9, no. 4, pp. 373–392, December 2012.

[25] H. Liang, L. X. Cai, D. Huang, X. Shen, and D. Peng, "An smdp- based service model for interdomain resource allocation in mobile cloud networks," IEEE Trans. on Vehicular Technology, vol. 61, no. 5, pp. 2222–2232, Jun 2012.

[26] T . H. Noor, Q. Z. Sheng, A. H. H. Ngu, and S. Dustdar, "Analysis of web-scale cloud services," IEEE Internet Computing, vol. 18, no. 4, pp. 55–61, July 2014.

[27] A. V. Dastjerdi and R. Buyya, "Compatibility-aware cloud service composition under fuzzy preferences of users," IEEE Trans. on Cloud Computing, vol. 2, no. 1, pp. 1–13, Jan 2014.

[28] J. Xiao, H. Wen, B. Wu, X. Jiang, P.-H. Ho, and L. Zhang, "Joint design on dcn placement and survivable cloud service provision over all-optical mesh networks," IEEE Trans. On Communications, vol. 62, no. 1, pp. 235–245, January 2014.

[29] W. Chen, J. Cao, and Y. Wan, "QoS-aware virtual machine scheduling for video streaming services in multi-cloud," Tsinghua Science and Technology, vol. 18, no. 3, pp. 308–317, June 2013.

[30] N. Tziritas, S. U. Khan, C. Z. Xu, T. Loukopoulos, and S. Lalis, "On minimizing the resource consumption of cloud applications using process migrations," Journal of Parallel and Distributed Computing, vol. 73, pp. 1690–1704, 2013.

[31] A. A. Chandio, K. Bilal, N. Tziritas, Z. Yu, Q. Jiang, S. U. Khan, and C. Z. Xu, "A comparative study on resource allocation and energy efficient job scheduling strategies

in large-scale parallel computing systems," Cluster Computing, vol. 17, 2014.

[32] F. Zhang, J. Cao, W. Tan, S. U. Khan, K. Li, and A. Y. Zomaya, "Evolutionary scheduling of dynamic multitasking workloads for bigdata analytics in elastic cloud," IEEE Trans. on Emerging Topics in Computing, 2013.

[33] D. Kliazovich, S. T. Arzo, F. Granelli, P. Bouvry, and S. U. Khan, "eSTAB: Energy-efficient scheduling for cloud computing applications with traffic load balancing," in IEEE International Conference on and IEEE Cyber, Physical and Social Computing Green Computing and Communications, Aug 2013, pp. 7–13.

[34] L. Xu, C. Li, L. Li, Y. Liu, Z. Yang, and Y. Liu, "A virtual data center deployment model based on the green cloud computing," in IEEE/ACIS 13th International Conference on Computer and Information Science, June 2014, pp. 235–240.

[35] N. N. Naik, K. Kanagala, and J. P. Veigas, "Achieving green computing by effective utilization of cloud resources using a cloud os," in IEEE International Conference on Emerging Trends in Computing, Commu- nication and Nanotechnology, 2013.

[36] T. T. Huu and C.-K. Tham, "An auction-based resource allocation model for green cloud computing," in IEEE International Conference on Cloud Engineering, 2013.

[37] K. Bilal, S. U. R. Malik, O. Khalid, A. Hameed, E. Alvarez, V. Wijay- sekara, R. Irfan, S. Shrestha, D. Dwivedy, M. Ali, U. S. Khan, A. Abbas, N. Jalil, and S. U. Khan, "A taxonomy and survey on green data center networks," Future Generation Computer Systems, vol. 36, no. 0, pp. 189 – 208, 2014.

[38] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog computing: A platform for internet of things and analytics," in Big Data and Internet of Things: A Roadmap for Smart Environments, ser. Studies in Computational Intelligence, N. Bessis and C. Dobre, Eds. Springer International Publishing, 2014, vol. 546, pp. 169–186.

[39] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing. ACM, 2012, pp. 13–16.

[40] K. Hong, D. Lillethun, U. Ramachandran, B. Ottenwalder, and B. Kold- ¨ ehofe, "Mobile fog: A programming model for large-scale applications on the internet of things," in Proceedings of the Second ACM SIGCOMM Workshop on Mobile Cloud Computing, 2013, pp. 15–20.

[41] M. Yannuzzi, R. Milito, R. Serral-Gracia, D. Montero, and M. Ne- mirovsky, "Key ingredients in an iot recipe: Fog computing, cloud computing, and more fog computing," in IEEE 19th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks, Dec 2014, pp. 325–329.

[42] J. Preden, J. Kaugerand, E. Suurjaak, S. Astapov, L. Motus, and R. Pahtma, "Data to decision: pushing situational information needs to the edge of the network," in IEEE International Inter-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support, March 2015, pp. 158–164.

[43] H. Madsen, G. Albeanu, B. Burtschy, and F. L. Popentiu-Vladicescu, "Reliability in the utility computing era: Towards reliable fog computing," in 20th International Conference on Systems, Signals and Image Processing, July 2013, pp. 43–46.

[44] C. T. Do, N. H. Tran, C. Pham, M. G. R. Alam, J. H. Son, and C. S. Hong, "A

proximal algorithm for joint resource allocation and minimizing carbon footprint in geo-distributed fog computing," in 2015 International Conference on Information Networking, Jan 2015, pp. 324–329.
[45] M. Aazam and E.-N. Huh, "Fog computing micro datacenter based dynamic resource estimation and pricing model for iot," in IEEE 29th International Conference on Advanced Information Networking and Applications, March 2015, pp. 687–694.
[46] "Dynamic resource provisioning through fog micro datacenter," in IEEE International Conference on pervasive Computing and Communication Workshops, March 2015, pp. 105 –110.
[47] C. Dsouza, G. J. Ahn, and M. Taguinod, "Policy-driven security management for fog computing: Preliminary framework and a case study," in IEEE 15th International Conference on Information Reuse and Integration, Aug 2014, pp. 16–23.
[48] S. J. Stolfo, M. B. Salem, and A. D. Keromytis, "Fog computing:Mitigating insider data theft attacks in the cloud," in IEEE Symposium on Security and Privacy Workshops, May 2012, pp. 125– 128.
[49] S. Kulkarni, S. Saha, and R. Hockenbury, "Preserving privacy in sensor- fog networks," in 9th International Conference for Internet Technology and Secured Transactions, Dec 2014, pp. 96–99.
[50] Shaoshan Liu, Jie Tang, Chao Wang, Quan Wang, and Jean-Luc Gaudio ,"Implementing a Cloud Platform for Autonomous Driving ",Fellow IEEE
[51] L. Lei, Z. Zhong, K. Zheng, J. Chen, and H. Meng. "Challenges on Wireless Heterogeneous Networks for Mobile Cloud Computing". In IEEE Wireless Communications, June 2013.
[52] M. Yannuzzi, , R. Milito† , R. Serral-Gracia` , D. Montero, and M. Nemirovsky‡,"Key ingredients in an IoT recipe: Fog Computing, Cloud Computing, and more Fog Computing"
[53] Subhadeep Sarkar†, Student Member, IEEE, Subarna Chatterjee*,Student Member, IEEE,Sudip Misra‡, Senior Member, IEEE," Assessment of the Suitability of Fog Computing in the Context f Internet of Things"
[54]" Overcoming the Heterogeneity in the Internet of Things for Smart Cities ", Aqeel Kazmi(B) , Zeeshan Jan, Achille Zappa, and Martin Serrano Insight Centre for Data Analytics, National University of Ireland, Galway, Ireland
[55] " Networking Protocols and Standards for Internet of Things " , Tara Salman
[56] " A Storage Solution for Massive IoT Data Based on NoSQL " , Tingli LI1,2, Yang LIU1,2, Ye TIAN1,, Shuo SHEN1, Wei MAO1. 2012 IEEE International Conference on Green Computing and Communications, Conference on Internet of Things, and Conference on Cyber, Physical and Social Computing
[57] ΈΖΥΦΓΚΣΥΤΚ¸ ΈΣΠΗ ΈΖΣΒΨΖΝΕΕΚ; ΈΘΟΚΥΦΡΞΠ' ΈΘΠ ג ΎΟ ΖΥΟΖΞΖΝ'ΖΤΠ ΤΔΚΥΪΝΒΟΈΒΥΒ¸ΖΧΚΥΒΣΖΡΠΠ' ;ΈΒΣΚΒ ·ΒΝΖΣΠ ΒΧΒΕ ;ΠΙΒΞΞΒΕΡΠΦΣ ΈΕΟΒ ιΚ ΈΘΟΒΪΘΟΒΚΉ ΈΖΟ ΙΒΌΠΟΘ
[58] "Fog Intelligence for Real-Time IoT Sensor Data Analytics Hazem M. Raafati , (Member, IEEE), M. Shamim Hossain 2,5, (Senior Member, IEEE), Ehab Essa3 , Samir Elmougy3 , (Member, IEEE), Ahmed S. Tolba3 , Ghulam Muhammad4 , (Member,

IEEE), and Ahmed Ghoneim5,6, (Member, IEEE)
[59]Joseph Redmon , Santosh Divvala , Ross Girshick , Ali Farhadi,"You Only Look Once: Unified, Real-Time Object Detection" in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)
[60],Joseph Redmon ,Ali Farhadi, "YOLOv3: An Incremental Improvement" in arXiv ,2018
[61]A. Seal, S. Bhattacharya and A. Mukherjee, "Fog Computing for Real-Time Accident Identification and Related Congestion Control," 2019 IEEE International Systems Conference (SysCon), Orlando, FL, USA, 2019, pp. 1-8.
[62]S. Bhattacharyya, A. Seal and A. Mukherjee, "Real-Time Traffic Incidence dataset," 2019 SoutheastCon, Huntsville, AL, USA, 2019, pp. 1-5.
[63]A. Seal and A. Mukherjee, "Real Time Accident Prediction and Related Congestion Control Using Spark Streaming in an AWS EMR cluster," 2019 SoutheastCon, Huntsville, AL, USA, 2019, pp. 1-7.
[64]https://www.rcrwireless.com/20140509/evolved-packet-core-epc/lte-network-diagram