Road accident Detection, Prediction and Prevention

DSE CL ZG628T DISSERTATION

A report submitted in partial fulfillment of the requirements for the award of

Master of Technology

in

DATA SCIENCE AND ENGINEERING

by

SURAJ P B(2018AB04032)

Project work carried out at

Vodafone Idea Ltd

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE
Pilani (Rajasthan) India

IV SEMESTER 2018-20

Acknowledgement

On the recollection of so many great favors and blessings, i offer my sincere thanks to the Almighty, the Creator and Preserver.

I sincerely thank my organization Vodafone Idea Ltd (Vi) for facilitating an environment to test and run the prototype.

I am thankful to my Professor Maninder Singh Bawa (Birla Institute of Technology and Science, Pilani) for the full-fledged support, advice and guidance.

Last but not the least I would also like to thank my friends and family who helped me a lot in finalizing this project and supporting me in my endeavours in ways both big and small, but always significant.

Thanking all of you,

SURAJ P B

Connect to My linkedIn

Abstract

The development of a transportation system has been the generative power for human beings to have the highest civilization above creatures in the earth. There are several risk factors such like over speed, driver fatigue in driving. It not only affects the severity of a crash, but also increases risk of being involved in a crash. The main purpose of this project is to develop a prototype of the Vehicle Black Box System (VBBS) that can be installed into any vehicle and control the data flow from such N number vehicles and X number of sensors outfitted with each vehicle. This prototype can be designed with minimum number of circuit elements for the hardware and Big data operations are done in Local machine for avoiding the cloud bill. The project focuses on building an infrastructure which vehicle safety authorities can implement in a massive scale for a social cause to predict road accidents early with a machine learning model that could be implemented in Apache Spark, detect the accidents through the available sensor circuitry and Prevent the accidents by real-time detection of driver fatigue through deep neural networks and alert the driver in real time, the edge device logs also can be used for post-crash analysis and to reduce the time it takes for emergency rescue to arrive at the crash location.

Table of Contents

List of Figures

Chapter 1

Introduction

According to the World Health Organization, more than a million people in the world die each year because of transportation-related accidents. The risk of accident or crash on road has become an unavoidable issue globally and of everyone's concern. They have predicted the figure would reach 2 million casualties by the year 2022 if no action is taken. Those who live, has a high chance of incurring a disability as a result of the impact. 91% of the fatalities on the roads occur in middle and low income countries. Even with so many initiatives in place accidents are still a major factor. Especially in places where road and traffic law enforcement is very low, cause of poor traffic management and also lack of awareness for road safety.

In order to react to this situation, this prototype draws the first step to solve this problem that crosses national boundaries and threatens the safety and health of people worldwide. Introduced to a part of the United States market in 1999, the black box system proved to be efficient. However in the latter case, the system was embedded in the vehicle. Therefore, in addition to improving the treatment of crash victims and the road status in order to decrease the death rate, constructing safer vehicles, and helping insurance companies with their vehicle accidents investigations, the main purpose of this paper is to develop a system that can be installed to any vehicle all over the world

Like flight data recorders in aircraft, "black box" technology can now play a key role in motor vehicle crash investigations. A significant number of vehicles currently on the roads contain electronic systems that record information in the event of a crash. That is why it is so important to have recorders that objectively track what goes on in vehicles before, during and after a crash as a complement to the subjective input that is taken usually from victims, eye witnesses and police reports. The focus of our project is to detect the accidents from various edge device sensors (IoT) and instant notification to the Hospital for an ambulance if the accident is detected and Other concerned authorities to save the life, also in order to build a machine learning model

to predict the dangerous driving patters and violations. If N number of such vehicles are there on the road currently and X number of sensors are outfitted with each vehicle, then N*x events will generated at the same time. We have to develop a data simulator also needed to fetch live weather data from the servers to get the traffic pattern, so we can identify which weather condition is most weighted feature and which may cause more driving violations. So in such a weather we will alert the driver to take more precautions.

Figure 1.1: Accident detection and notification flow diagram

Such a systems deployed in the wider scale will result in much accurate and more accurate accident reports recorded effectively saving more lives. Even if a vehicle crash and the driver is dead or unconscious in an isolated location where no one is around to notify the incident, such system may become a critical factor in life safety. We can utilize existing mobile number network infrastructure already in place and also using GPS to pinpoint the exact location of the crash and inform the data to an emergency rescue authority such as Hospital, Fire force department and police. Despite many efforts taken by different governmental and non-governmental organizations all around the world by various programs to aware against careless driving, yet accidents are taking place every now and then. However many lives could have been saved if emergency service could get crash information on time.

Chapter 2

Literature survey

Over the last couple of years, there has been dramatically increase in the interest in VANETs research. The Vehicle Safety Communications (VSC) project brings Mercedes-Benz, Ford, Nissan, GM, Toyota, BMW and VW together to work cooperatively. The applications for vehicular safety application have widely been discussed by National Highway Traffic Safety Administration [NHTSA] [1][2]. The vehicle safety applications have been developed [3][4]. There are number of work proposed to study DSRC technology that improves safety on road in [5] an over view of vehicle cooperative collision avoidance application based on emerging DSRC device and improve the highway traffic safety along with demonstrating the need for data prioritization for safety critical application.

Vehicular applications are typically classified in (i) active road safety applications, (ii) traffic efficiency and management applications, and (iii) comfort and infotainment applications [11]. The first category aims to avoid the risk of car accidents and make safer driving by distributing information about hazards and obstacles. The basic idea is to broaden the driver's range of perception, allowing him/her to react much quicker, thanks to alerts reception through wireless communications. The second category focus on optimizing flows of vehicles by reducing travel time and avoiding traffic jam situations. Applications like enhanced route guidance/navigation, traffic light optimal scheduling, and lane merging assistance, are intended to optimize routes, while also providing a reduction of gas emissions and fuel consumption. Finally, although the primary purpose of VANETs is to enable safety applications, non-safety applications are expected to create commercial opportunities by increasing the number of vehicles equipped with on-board wireless devices. Comfort and infotainment applications aim to provide the road traveler with information support and entertainment to make the journey more pleasant. In the next subsections we will describe the main aspects of safety and entertainment applications for VANETs

Chapter 3

Benefits to the Organization

The global volume of data is increasing exponentially along the growth of the Internet of Things (IoT) and the ubiquity of connected devices. Much of the newly generated data is valuable; it can be used to optimise manufacturing operations, track assets with increasing accuracy, target existing consumer services with high granularity, and create entirely new services and business models. IoT opens the door to endless possibilities in the world of business. Devices, appliances, and even vehicles can connect and exchange data through a wired or wireless network. The rise of IoT in telecom industry brings a lot of opportunity to operators to generate new value and revenue; allowing Telecommunications to compete not only on the functionality and performance of products and services but the insight created by the use of these products or services, giving operators new ways to differentiate offerings and a new source of value. Going forward, the increasing adoption of cloud computing and applications like smart buildings, vehicle telematics and industrial automation further pushes the demand for the IoT Telecom services market globally.The main opportunities from IoT for Telecoms include:

- Improved Customer Satisfaction Collect data for analysis about processes and offerings in real-time to make the best decisions that impact customers positively.
- Flexible, Cost-Effective Cloud Computing Collect data to transform a smart environment into a decision-making platform and transfer to the cloud through IoT.
- New Business Models Build new business models and revenue streams. As IoT progresses, people and businesses will require new services. IoT enables operators to create new revenue streams and services on top of core services.

Vi™ IoT has a number of products and services addressing the complete Internet of Things (IoT) market – connecting machines, devices and services. Among which I

am planning to expand a prototype to our organization to the project called Smart Mobility, which is in initial stage now. Vi Smart Mobility uses an IoT-enabled 'black box' to make your vehicle a 'connected vehicle'. By enabling wireless communication with transportation systems, it can significantly enhance vehicle safety, mobility and personal convenience. It also opens up new business opportunities in terms of new service offerings and subscription models to customers.

On a larger scale, the data that you collect from your smart vehicles can be used to streamline operations, improve design and vehicle performance, and get insights into customer behaviour like never before.

The benefits for business includes:

- Enables your vehicles to self-diagnose potential difficulties
- Help reduce road accidents and delays
- Preventing theft and assisting in the recovery of stolen vehicles
- Provides valuable vehicle performance data
- Offering in-car entertainment and other value-added services

Chapter 4

Proposed system

The aim of this project is to implement a system that helps in immediate rescue operations on an accident once the accident is detected, predict road accidents through violations and control the accidents by proactively alerting a snoozing driver.

4.1 Objectives

To create a prototype of this system our objectives include

1. To identify whether the vehicle plunged into nearest river after crash
2. To identify the crash and to inform the incident to rescue team
3. To identify whether the vehicle was moving over speed
4. To predict the road accidents by analyzing the historical dataset through machine learning approach
5. To identify a snoozing driver through deep learning way
6. To build a big data stream processing model for such N number of vehicles and X no. of sensors outfitted with it
7. To architect a Big data platform by forecasting the future demand if we commercialize this product in a wide scale

In order to meet these objectives, we have implanted Raspberry Pi as the edge system processor which can control the edge core hardware, GPS receiver to retrieve the incident latitude and longitude, Predict the road accidents using Apache spark with the dataset and Architect Big data model for massive event processing

Chapter 5

Accident Detection Using Edge Hardware system

5.1 GPIO operations of Raspberry Pi

Raspberry Pi3 is the Main control unit in the hardware part of the Project. The Raspberry Pi is an SOC that can be plugged into your monitor and a keyboard for programming or working with it, which can be used in electronics projects, and for many of the things that one desktop PC does. It also requires a +5V power and all the other sensors and interfaces are connected to this either through USB or through the GPIO ports.

One powerful feature of the Raspberry Pi is the row of GPIO (general purpose input/output) pins along the edge of the board, next to the yellow video out socket. These pins are a physical interface between the Pi and the outside world. At the simplest level, you can think of them as switches that you can turn on or off (input) or that the Pi can turn on or off (output). Out of 40 pins in the row (combined), 8 pins are of Ground, 2 pins are of 5V, 2 pins are of 3.3V and 26 pins are GPIO. Ignoring the Pi for a moment, one of the simplest electrical circuits that you can build is a battery connected to a light source and a switch (The resistor is there to protect the LED)

When we use a GPIO pin as an output, the Raspberry Pi replaces both the switch and the battery in the above diagram. Each pin can turn ON or OFF, or go HIGH or LOW in computing terms. When the pin is HIGH it outputs 3.3volts: when the pin is LOW it is OFF. The above circuit can be implemented using Raspberry Pi.

The LED can be connected to a GPIO pin (Which can output +3.3V – suppose pin no. 1) and a ground pin (which is 0V and acts like the negative terminal of the battery).GPIO OUTPUTs are easy; they are ON or OFF, HIGH or LOW, 3.3V or 0V. INPUTs are bit trickier because of the way that digital device work. Although it might

seem reasonable just to connect a button across an input pin and a ground pin, the Pi can get confused as to whether the button is ON or OFF.

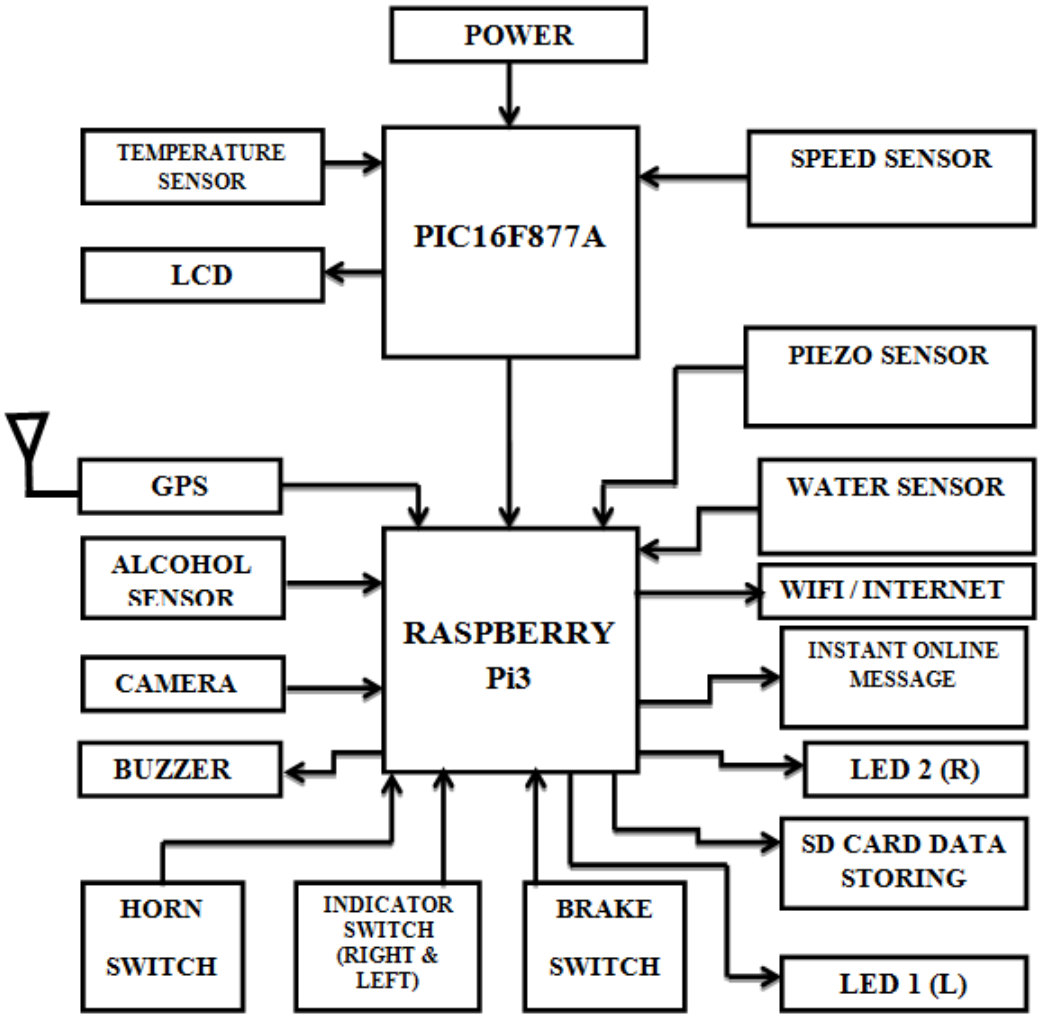

Figure 5.1: Edge hardware system block diagram

So in order to avoid the confusion which may lead to undesirable outputs, the Raspberry Pi has built-in pull-up and pull-down resistors which can be enabled in software. This means we can eliminate our pull-down resistors for the button as long as we enable the internal ones

- GPIO pin 38 is used to interface the Alcohol sensor MQ-3 through a comparator circuit made of Op-Amp LM358 - A standard op-amp operating in open-loop configuration (without negative feedback) may be used as a low-performance comparator. When the non-inverting input (V+) is at a higher voltage than the inverting input (V-), the high gain of the op-amp causes the output to saturate at the highest positive voltage it can output. When the non-inverting input (V+)

drops below the inverting input (V-), the output saturates at the most negative voltage it can output –Whenever there is an alcohol presence, the output of comparator goes LOW and thus will get a LOW at GPIO 38 and it will be considered as there is a presence of Alcohol

- GPIO pin 36 is connected to water sensor- through a comparator circuit made of Op-Amp LM358 - Whenever there is water presence, the output of comparator goes LOW and thus will get a LOW at GPIO 36 and it will be considered as there is a presence of water. The water sensor module is an easy tool for water detection.

  It can be used as a switch when water falls through the board and also for measuring waterfall intensity. The module features, a water board and the control board that is separate for more convenience, power indicator LED and an adjustable sensitivity though a potentiometer. The analog output is used in detection of drops in the amount of water. Connected to 5V power supply, the LED will turn on when induction board has no water drops, and DO output is high. When dropping a little amount water, DO output is low, the switch indicator will turn on. Brush off the water droplets, and when restored to the initial state, outputs high level

- GPIO Pin 33 is used to simulate the operation of Brake. If the push button switch is pressed once, Signal at the GPIO pin 33 will go LOW momentarily and the status data may be accordingly recorded

- GPIO Pin 31 is used to simulate the operation of Horn. If the push button switch is pressed once, Signal at the GPIO pin 31 will go LOW momentarily and the status data may be accordingly recorded

- GPIO Pin 37 is used to simulate the operation of Left Indicator. If the SPDT switch is moved towards left side once, Signal at the GPIO pin 37 will go LOW and the status data may be accordingly recorded also the Left indicator LED light on the prototype also will glow.

- GPIO Pin 35 is used to simulate the operation of Right Indicator. If the SPDT switch is moved towards right side once, Signal at the GPIO pin 35 will go LOW and the status data may be accordingly recorded also the Right indicator LED on the prototype also will glow.

- GPIO pin 32 is used to interface the Piezo-Electric Sensor - a device that uses the piezoelectric effect to measure pressure, acceleration or force by converting them

to an electrical charge- through a comparator circuit made of Op-Amp LM358 - A standard op-amp operating in open-loop configuration (without negative feedback) may be used as a low-performance comparator. When the non-inverting input (V+) is at a higher voltage than the inverting input (V-), the high gain of the op-amp causes the output to saturate at the highest positive voltage it can output. When the non-inverting input (V+) drops below the inverting input (V-), the output saturates at the most negative voltage it can output – Whenever there is an accidental pressure or force applied on the Piezo , the output of comparator goes LOW and thus will get a LOW at GPIO 32 and it will be considered as an accident occurred

- GPIO Pin 8 is used as output to simulate the operation of Right indicator LED, and this pin will go HIGH whenever the signal at GPIO pin 35 goes LOW
- GPIO pin 10 is used as output to simulate the operation of Left indicator LED, and this pin will go HIGH whenever the signal at GPIO pin 37 goes LOW
- GPIO pin 40 is used as output to drive the Buzzer, whenever there is a LOW signal appears at GPIO pin 32 or 36, the pin 40 will go HIGH and it makes the Transistor BC547 ON and subsequently the Buzzer may start making sound continuously.

All above mentioned GPIO pins are programmed using python with codes mentioned on [Appendix B.1, B.14]

In addition to this, Whenever the GPIO pin 32 or 36 receiving a LOW signal, the camera will capture the image as well as the Raspberry Pi will send a message to pre-defined mobile number through instant online Twilio messaging to alert the Emergency contact number that there is an accident occurred with the location coordinates. Also will send alert message as “Accident detected” and “plunged into water” with lat-long details incase pin no. 32 and 36 goes low respectively.

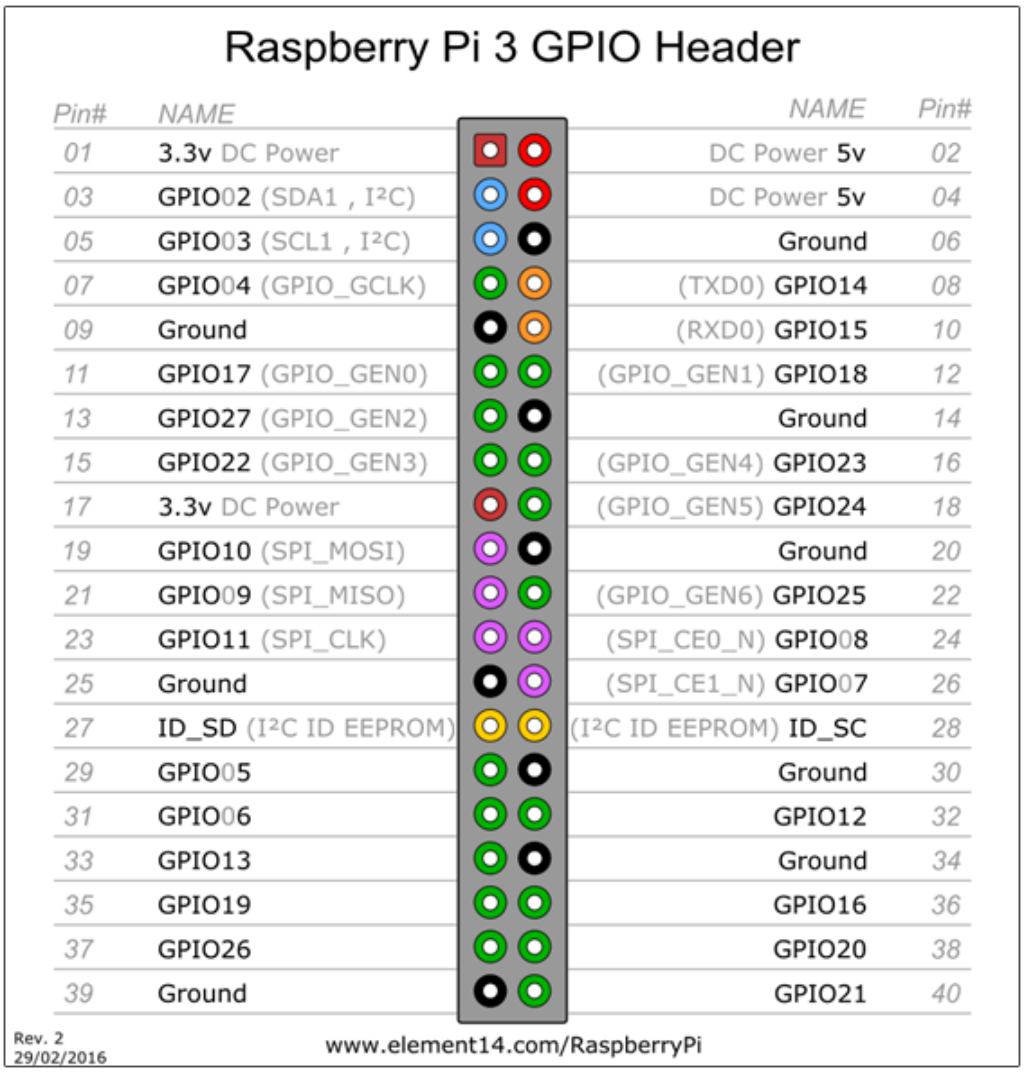

Figure 5.2: GPIO pin outs

5.1.1 Twilio online messaging

Twilio is a Cloud communications online platform, in which we can start building our application in the programming language we already in use (python, in our case). API for Messaging, Voice, Video, and Authentication will build automatically in the server whenever we request for a service (online messaging, in our case) and provide us Twilio number, Account SID, Authentication token and program codes to communicate with their cloud server through programmed API. Some twilio modules should be imported for functioning with this server [Appendix B.2]. Also we can implement a threaded function with the retrieved information from twilio server and the code is as mentioned in between [Appendix B.3]

5.1.2 Emailing notification of the incident

A mail will send to the predefined / prefixed mail id with an alert of "Accident detected" and incident lat-long details and captured image via USB camera whenever 32nd and 36th GPIO pin goes low. Some modules needed to be imported in python for sending the mail [Appendix B.4]. Then we can define a threaded function to send the mail from/to earlier defined mail as given in between [Appendix B.5]. A sample mail mentioned below, captured while leaving from application layer interface which obeys SMTP / MIME protocols to communicate with receiving mail server

5.1.3 Multi-threading in Python and interrupt handling

The Main control unit, Raspberry Pi is functioning as explained above with the help of code program written in Python programming and it is made with such a logic to work as multi-threaded program to meet the project goal through four main function blocks (GPS data, uploading to cloud bucket, pic interface and collection of status from other interfaces).Running several threads is similar to running several different programs concurrently, but with the following benefits

1. Multiple threads within a process share the same data space with the main thread and can therefore share information or communicate with each other more easily than if they were separate processes.

2. Threads sometimes called light-weight processes and they do not require much memory overhead; they are cheaper than processes.

A thread has a beginning, an execution sequence, and a conclusion. It has an instruction pointer that keeps track of where within its context it is currently running.

1. It can be pre-empted (interrupted)

2. It can temporarily be put on hold (also known as sleeping) while other threads are running - this is called yielding.

To start a new thread, need to call following method available in thread module:

thread.start_new_thread(function, (args[, kwargs]))

This method call enables a fast and efficient way to create new threads in both Linux and Windows.

The method call returns immediately and the child thread starts and calls function with the passed list of args. When function returns, the thread terminates.Here, args

is a tuple of arguments; use an empty tuple to call function without passing any arguments. kwargs is an optional dictionary of keyword arguments, Hence multithreading can be

1. "Smallest unit that can be scheduled in an operating system"
2. Two or more parallel (which is implemented on a single processor by multitasking) tasks
3. More than one thread can exist within the same process

We have implemented the feature of multithreading, using the codes in python [Appendix B.12]. 'import thread' module insertion is mandatory to perform this threading feature

The accident/physical destructions will be sensing by the Piezo-sensor and an interrupt will handle this case irrespective of which thread is under progress [Appendix B.13]

5.1.4 System autorun

The entire program code will let to run in bootstrap programming (The program will execute as soon as the Raspberry Pi powered up). An autorun text file needed to be created with main python code location in it and needed to rename the text file as launcher.sh, 30 second delay needed to be given for starting the processing after every reboot of Raspberry Pi
Sleep 30
cd / home/pi/Desktop
sudo python Main.py
Below mentioned command needed to run using os command prompt,
@reboot sudo sh /home/pi/Desktop/launcher.sh

5.2 Advantages

1. Immediate rescue operations can be ensured on Accident
2. Helps law authorities to decide the ownership of the accident
3. Can be used for proper analysis of accident
4. Live vehicle black box information can be accessed from anywhere at anytime
5. Low cost

6. Maintainable and more flexible

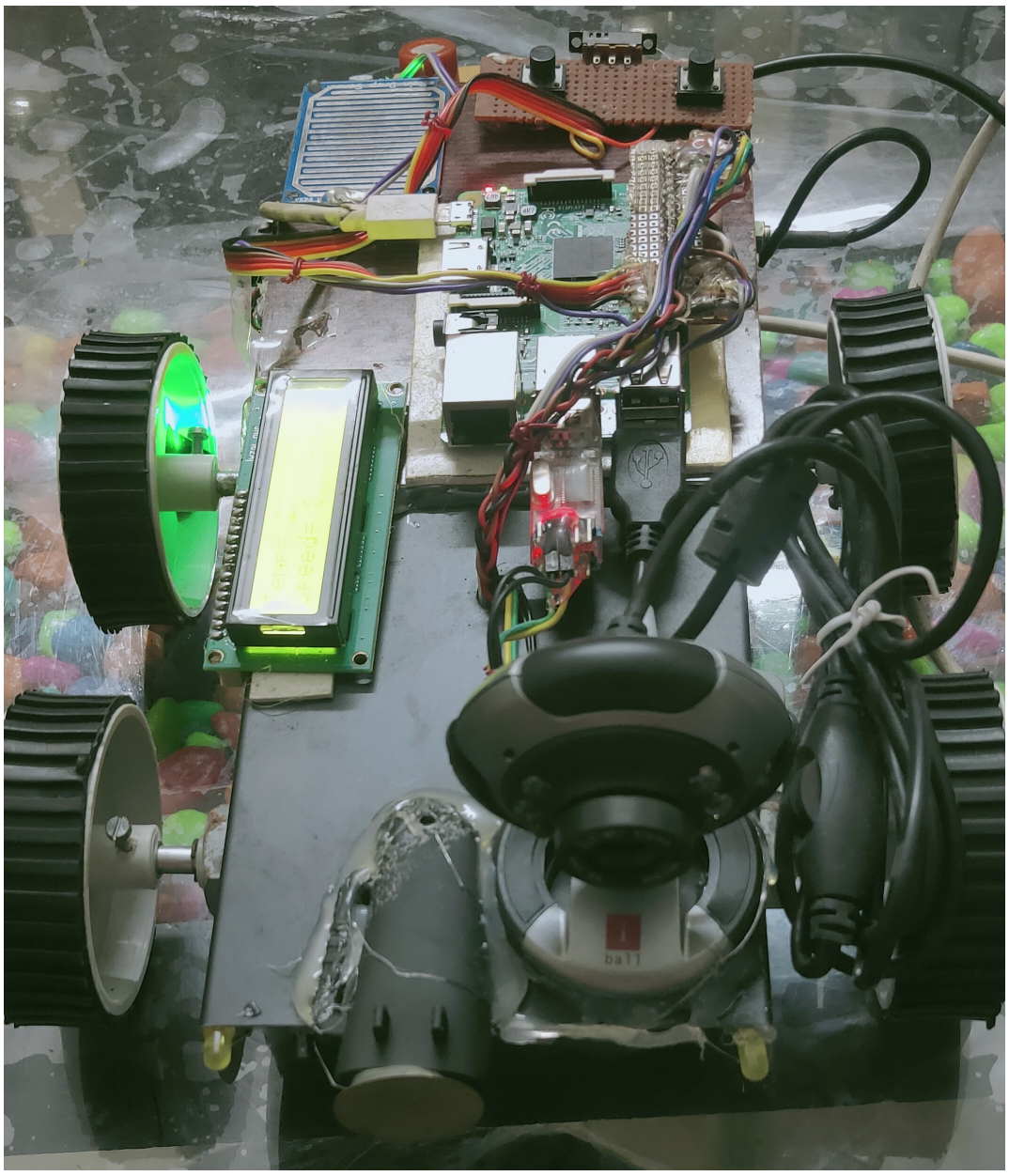

Figure 5.3: Edge Hardware - A prototype

```
MonFeb22:52:47 2021|VehicleNo.:2018AB04032|RI0FF,LI0FF|HnP|BnP|AnD|Temp=27|RPM=0
|Lat=1016.838|Lon=07608.566
```

Figure 5.4: Data output from Edge Hardware System

5.3 Limitations of Edge hardware system

1. Prompt intimation on Accident depends on the data network coverage
2. Industrial level implementation required permission from concerned authorities
3. Full benefits of this project can only be assured after the full-fledged implementation of IoT
4. For implementing Deep neural networks in Raspberry Pi device is a cumbersome activity. More GPU / TPU and CUDA are required for full fledged training of CNN / ANN

Chapter 6

Accident Prevention through Deep learning way

6.1 Drowsiness detection and alerting the Driver to prevent the accidents

Driver's status is crucial because one of the main reasons for motor vehicular accidents is related to driver's inattention or drowsiness. Drowsiness detector on a truck or car can reduce numerous accidents. Accidents occur because of a single moment of negligence, thus driver monitoring system which works in real-time is necessary. This detector should be deployable to an embedded device and perform at high accuracy

6.1.1 Driver fatigue detection using Deep Learning

Recently, deep learning is widely used to resolve difficult problems which cannot be handled properly using conventional methods. Deep learning based on Convolutional Neural Networks (CNNs) makes a breakthrough especially for Computer Vision tasks such as image classification, object detection, emotion recognition, scene segmentation.

When a driver does not sleep properly or sleep while driving it gives a warning and start alarm which can prevent accidents. A camera in front of the driver face which detects the driver's face and using deep learning identify the driver eyes closed or opened with the help of this data system detect driver drowsiness.

Convolutional Neural Network used for classification of the images, rather person eye in image close or open. Tensorflow / Keras API used for building a CNN. A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The

pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

6.1.2 Neural Net Architecture

In our proposed fatigue detection system, we used a model composed of two double convolutional layers followed by max-pooling layers, leading to two fully connected hidden layers with a final Softamx layer for classification as illustrated in Fig.5.1

1. Convolutional Layer: The convolutional layer is the core of any convolutional network accomplishing the main role of extracting input image features. It is usually the first layer and consists of multiple learn-able filters to produce an activation map. It accepts an input size of (W x H x D) which correspond to our data image size of (100 x 100 x 1) since we are converting RGB images into grey-scale to focus on facial features rather than skin tone. We use double convolutional layers as this has experimentally produced a balance between accuracy and computations.

2. Max-Pooling Layer: A common non-linear downsampling method to reduce the spatial size of the representation in order to decrease the number of parameters, hence reduce computation time and control over-fitting.

3. Dropout: The dropout layer can be used as an image noise reduction technique when applied after max-pooling layer. When it used with fully connected layers it deactivates part of the neurons in order to improve generalization by allowing the layer to learn the same concept with different neurons and avoid over-fitting.

4. Fully-Connected Layer: Dense hidden layers take an input volume from the output of the last max-pooling layer to produce an N-dimensional vector, where N corresponds to the number of classes. In our case, the model has to choose from two classes (fatigued and control) for classification.

5. Model Parameters: We build our sequential model by adding a stack of linear layers. The system is trained after defining an optimizer and a loss function. We use the Adam optimizer, which is a stochastic optimization technique. For the loss function, the objective function that the training tries to minimize, we use the categorical cross entropy. For performance evaluation, we use the accuracy.

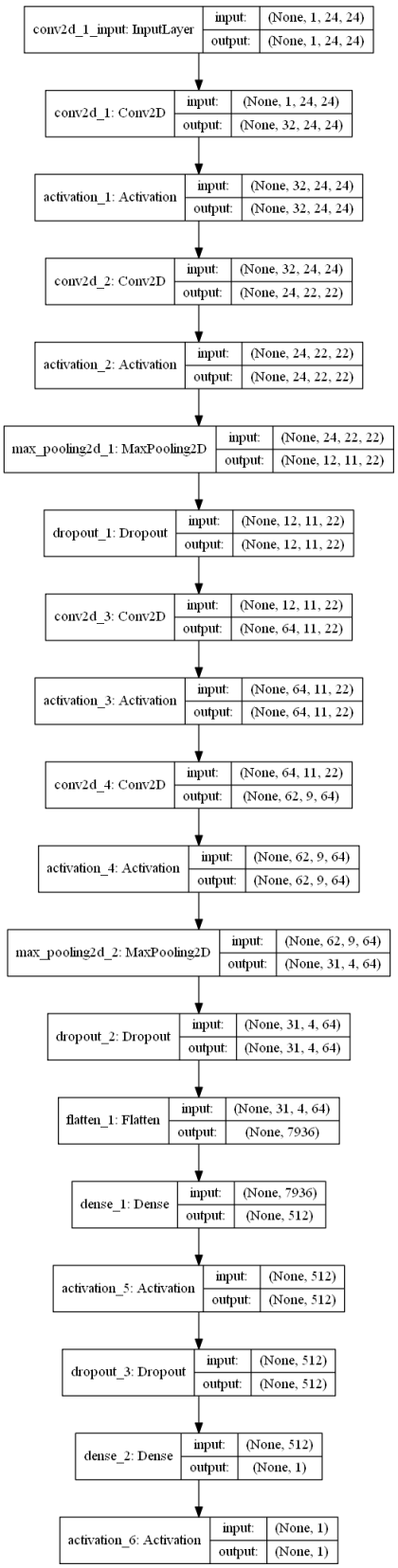

Figure 6.1: CNN Model

6.1.3 System Operation

The proposed real-time fatigue detection system start by loading weights from a pre-trained network to transfer the learning and reduce dependency on large datasets. We then capture a frame from the camera and apply face detection to establish a region of interest (In our case, it is left eye). The eye image is then run through the classifier to label the frame as having signs of drowsy or not. If the label persists over time, a warning is delivered to smart phone / buzzer module of edge device.

```
[1]: import tensorflow as tf
     from tensorflow import keras
     from tensorflow.keras import layers
     from tensorflow.keras.preprocessing.image import ImageDataGenerator
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
     from tensorflow.keras.layers import Convolution2D, MaxPooling2D
     from tensorflow.keras.models import load_model
     from tensorflow.keras.optimizers import SGD, Adadelta, Adagrad
     from  tensorflow.keras.callbacks import ModelCheckpoint
     from six.moves import cPickle as pickle
     import numpy as np
     from sklearn.metrics import accuracy_score
     from sklearn.model_selection import train_test_split

     pickle_files = ['open_eyes.pickle', 'closed_eyes.pickle']
     i = 0
     for pickle_file in pickle_files:
         with open(pickle_file, 'rb') as f:
             save = pickle.load(f)
             if i == 0:
                 train_dataset = save['train_dataset']
                 train_labels = save['train_labels']
                 test_dataset = save['test_dataset']
                 test_labels = save['test_labels']
             else:
                 print("here")
                 train_dataset = np.concatenate((train_dataset,␣
      ↪save['train_dataset']))
                 train_labels = np.concatenate((train_labels, save['train_labels']))
                 test_dataset = np.concatenate((test_dataset, save['test_dataset']))
                 test_labels = np.concatenate((test_labels, save['test_labels']))
             del save  # hint to help gc free up memory
         i += 1

     print('Training set', train_dataset.shape, train_labels.shape)
     print('Test set', test_dataset.shape, test_labels.shape)
```

Figure 6.2: CNN Model - Training phase

```
batch_size = 30
nb_classes = 1
epochs = 12

X = train_dataset
#X_train = X_train.reshape((X_train.shape[0], X_train.shape[3]) + X_train.
 ↪shape[1:3])
#X_train = X_train.reshape((X_train.shape[0], X_train.shape[3]) + X_train.
 ↪shape[1:3])
Y = train_labels
X_train, X_val, Y_train, Y_val = train_test_split(X, Y, test_size=0.20,␣
 ↪random_state=42)

X_test = test_dataset
#X_test = X_test.reshape((X_test.shape[0], X_test.shape[3]) + X_test.shape[1:3])
Y_test = test_labels

# print shape of data while model is building
print("{1} train samples, {4} channel{0}, {2}x{3}".format("" if X_train.shape[1]␣
 ↪== 1 else "s", *X_train.shape))
print("{1}  test samples, {4} channel{0}, {2}x{3}".format("" if X_val.shape[1]␣
 ↪== 1 else "s", *X_val.shape))

# input image dimensions
_,img_rows, img_cols,img_channels = X_train.shape

model = Sequential()

model.add(Convolution2D(32, (3, 3), padding='same',
                        input_shape=(img_rows, img_cols,img_channels),))
model.add(Activation('relu'))
model.add(Convolution2D(24, (3, 3),))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Convolution2D(64, (3, 3), padding='same',))
model.add(Activation('relu'))
model.add(Convolution2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
```

Figure 6.3: CNN Model - Activation function

```
model.add(Dense(nb_classes))
model.add(Activation('sigmoid'))


filepath="Models/weights-improvement-{epoch:02d}-{val_acc:.2f}.hdf5"
checkpoint =␣
 ↪ModelCheckpoint(filepath,monitor='val_acc',mode='max',save_best_only=True,verbose=1)
callbacks_list = [checkpoint]
# let's train the model using SGD + momentum (how original).
sgd = SGD(lr=0.01,decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='binary_crossentropy', optimizer=sgd, metrics=['accuracy'])
model.fit(X_train, Y_train, batch_size=batch_size, epochs=epochs, verbose=2,␣
 ↪validation_data=(X_val, Y_val))
yhat_classes = model.predict_classes(X_test, verbose=0)
accuracy = accuracy_score(Y_test, yhat_classes)
print("Test accuracy:",accuracy)
# STEP_SIZE_TRAIN=train_generator.n//train_generator.batch_size
# STEP_SIZE_VALID=valid_generator.n//valid_generator.batch_size
# model.fit_generator(generator=train_generator,
#                     steps_per_epoch=STEP_SIZE_TRAIN,
#                     validation_data=valid_generator,
#                     validation_steps=STEP_SIZE_VALID,
#                     epochs=10,callbacks=callbacks_list
# )
```

```
C:\Users\ANNA\anaconda3\envs\tf\lib\site-
packages\tensorflow\python\framework\dtypes.py:516: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\ANNA\anaconda3\envs\tf\lib\site-
packages\tensorflow\python\framework\dtypes.py:517: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\ANNA\anaconda3\envs\tf\lib\site-
packages\tensorflow\python\framework\dtypes.py:518: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\ANNA\anaconda3\envs\tf\lib\site-
packages\tensorflow\python\framework\dtypes.py:519: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:\Users\ANNA\anaconda3\envs\tf\lib\site-
packages\tensorflow\python\framework\dtypes.py:520: FutureWarning: Passing
```

Figure 6.4: CNN Model - Optimizer

```
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:\Users\ANNA\anaconda3\envs\tf\lib\site-
packages\tensorflow\python\framework\dtypes.py:525: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
C:\Users\ANNA\anaconda3\envs\tf\lib\site-
packages\tensorboard\compat\tensorflow_stub\dtypes.py:541: FutureWarning:
Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\ANNA\anaconda3\envs\tf\lib\site-
packages\tensorboard\compat\tensorflow_stub\dtypes.py:542: FutureWarning:
Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\ANNA\anaconda3\envs\tf\lib\site-
packages\tensorboard\compat\tensorflow_stub\dtypes.py:543: FutureWarning:
Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\ANNA\anaconda3\envs\tf\lib\site-
packages\tensorboard\compat\tensorflow_stub\dtypes.py:544: FutureWarning:
Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:\Users\ANNA\anaconda3\envs\tf\lib\site-
packages\tensorboard\compat\tensorflow_stub\dtypes.py:545: FutureWarning:
Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:\Users\ANNA\anaconda3\envs\tf\lib\site-
packages\tensorboard\compat\tensorflow_stub\dtypes.py:550: FutureWarning:
Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])

here
Training set (3876, 24, 24, 1) (3876, 1)
Test set (970, 24, 24, 1) (970, 1)
3100 train samples, 1 channels, 24x24
776  test samples, 1 channels, 24x24
WARNING:tensorflow:From C:\Users\ANNA\anaconda3\envs\tf\lib\site-
packages\tensorflow\python\ops\init_ops.py:1251: calling
VarianceScaling.__init__ (from tensorflow.python.ops.init_ops) with dtype is
deprecated and will be removed in a future version.
```

Figure 6.5: CNN Model - Epochs

6.1.4 Real-time fatigue detection

As shown in below Fig, the detection system is capable of differentiating between alert and fatigued drivers. During the real-time detection process, an alarm can be sent immediately to the driver's smart phone or buzzer can be alarmed from the edge hardware system once fatigue symptoms are detected to alert the driver.

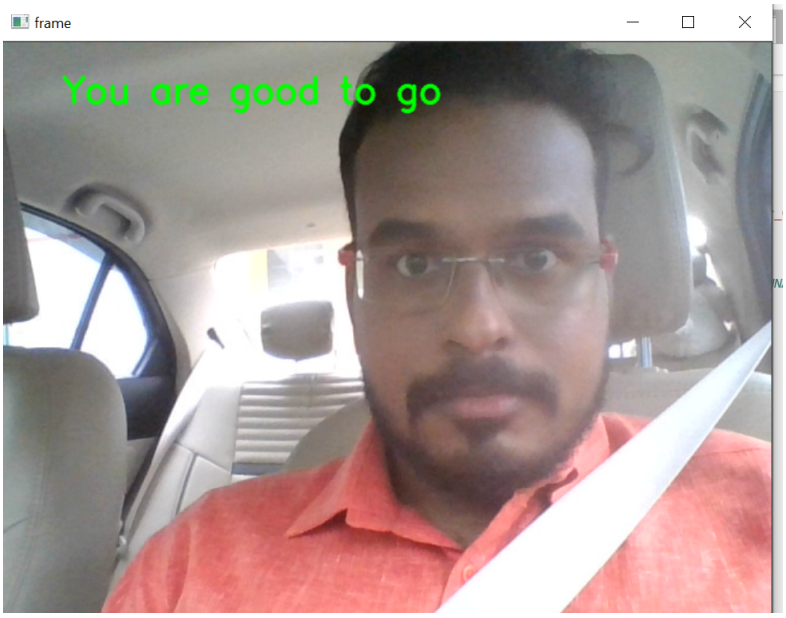

Figure 6.6: Detection of Open eyes.

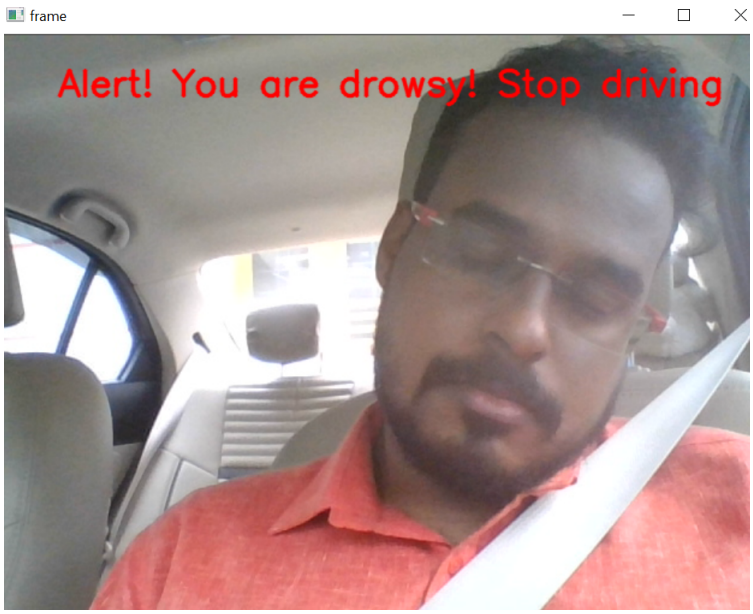

Figure 6.7: Detection of closed eyes.

6.1.5 Experimental Results and Analysis

While training, the system reported 93.81% of cross-validation accuracy. We also tested our proposed system's performance by applying on never seen before test subjects and obtained an average accuracy of 94.53%. We obtain testing accuracy from testing the model on subjects outside the training data. If the curves of testing and training accuracy start to depart consistently, then we need to stop the training at an earlier epoch to avoid over-fitting. We face the problem of overfitting when the model is over-learned the training data set. We can observe from the graph that 12 epochs in our model are sufficient for accurate results.

```
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the
constructor
WARNING:tensorflow:From C:\Users\ANNA\anaconda3\envs\tf\lib\site-
packages\tensorflow\python\ops\nn_impl.py:180:
add_dispatch_support.<locals>.wrapper (from tensorflow.python.ops.array_ops) is
deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
Train on 3100 samples, validate on 776 samples
Epoch 1/12
3100/3100 - 8s - loss: 0.6673 - acc: 0.5877 - val_loss: 0.6163 - val_acc: 0.6753
Epoch 2/12
3100/3100 - 7s - loss: 0.6077 - acc: 0.6726 - val_loss: 0.5873 - val_acc: 0.6894
Epoch 3/12
3100/3100 - 7s - loss: 0.5885 - acc: 0.6816 - val_loss: 0.5453 - val_acc: 0.7139
Epoch 4/12
3100/3100 - 7s - loss: 0.5205 - acc: 0.7377 - val_loss: 0.4347 - val_acc: 0.8054
Epoch 5/12
3100/3100 - 7s - loss: 0.4148 - acc: 0.8100 - val_loss: 0.3855 - val_acc: 0.8106
Epoch 6/12
3100/3100 - 7s - loss: 0.3038 - acc: 0.8761 - val_loss: 0.2461 - val_acc: 0.8905
Epoch 7/12
3100/3100 - 7s - loss: 0.2506 - acc: 0.8997 - val_loss: 0.2241 - val_acc: 0.9059
Epoch 8/12
3100/3100 - 7s - loss: 0.2179 - acc: 0.9126 - val_loss: 0.2147 - val_acc: 0.9072
Epoch 9/12
3100/3100 - 7s - loss: 0.1750 - acc: 0.9323 - val_loss: 0.1805 - val_acc: 0.9304
Epoch 10/12
3100/3100 - 7s - loss: 0.1648 - acc: 0.9358 - val_loss: 0.1816 - val_acc: 0.9304
Epoch 11/12
3100/3100 - 7s - loss: 0.1509 - acc: 0.9377 - val_loss: 0.1827 - val_acc: 0.9407
Epoch 12/12
3100/3100 - 7s - loss: 0.1364 - acc: 0.9500 - val_loss: 0.1773 - val_acc: 0.9381
Test accuracy: 0.945360824742268
```

Figure 6.8: Epochs and analysis

```
[2]: from sklearn.metrics import classification_report
     import matplotlib.pyplot as plt
     import seaborn as sns
     target_names = ['Open','Close']
     Y_pred=model.predict(X_test)
     Y_pred = (Y_pred>0.2)
     print(classification_report(Y_test, Y_pred, target_names=target_names))
     from sklearn.metrics import confusion_matrix
     def plot_cm(labels, predictions, p=0.5):
       cm = confusion_matrix(labels, predictions > p)
       plt.figure(figsize=(5,5))
```

Figure 6.9: Performance analysis of the model

```
  sns.heatmap(cm, annot=True, fmt="d")
  plt.title('Confusion matrix @{:.2f}'.format(p))
  plt.ylabel('Actual label')
  plt.xlabel('Predicted label')

plot_cm(Y_test,Y_pred)
```

```
              precision    recall  f1-score   support

        Open       0.99      0.84      0.91       477
       Close       0.87      0.99      0.93       493

    accuracy                           0.92       970
   macro avg       0.93      0.92      0.92       970
weighted avg       0.93      0.92      0.92       970
```

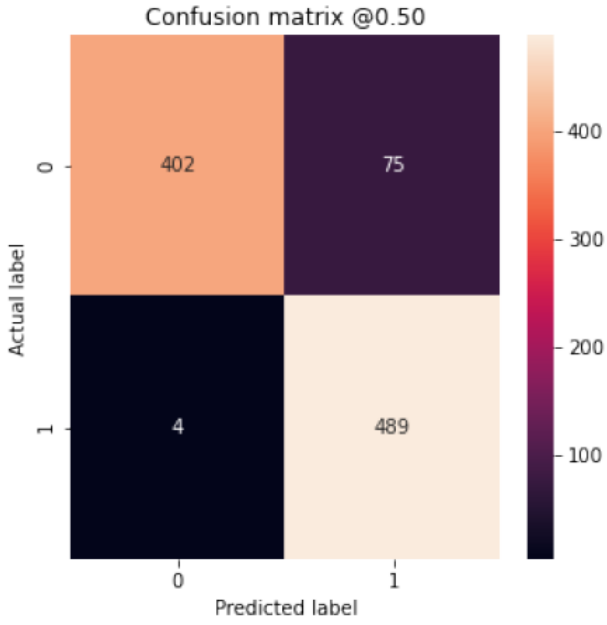

Figure 6.10: Confusion matrix

Chapter 7

Accident Prediction using Apache Spark

7.1 Machine Learning and Exploratory data analysis using Apache Zeppelin

Labels we are predicting are Road traffic violations occur or don't occur. Feature input variables are 'no. of hours driven in one stretch, km driven, weather conditions like foggy,rainy, windy'. For example, "Driver is drowsy and a road violation has occurred, In a windy weather, a lane cross violation occurred" are labelled examples. So utilizing the feature variables we can apply the training model to unlabelled example by introducing the machine learning techniques in accident prediction. In regression model, we can predict continuous values such as "What's the probability of accident in a particular scenario? " and in classification model we can learn the model to classify whether accident has occured or not.

We can do some exploratory data analysis (EDA) with the available dataset before jumping into Machine learning. We can perform some SQL Queries against the RDD such as

1. Do certified Drivers have less violations?
2. What's impact of fog/rain/wind on driving?
3. Does location has any impact on Incident?

So the features we have to create are

- Fatigue by Hours

- Fatigue by miles
- Weather conditions such as Foggy, Rainy and Windy
- Driver certified or not
- Driver wage plan (payment scheme)

So we have to fetch the data from HDFS, we have used Zeppelin notebook to fetch the data from HDP sandbox HDFS directory. This is a training model for data at rest (Hortonworks Data Platform).

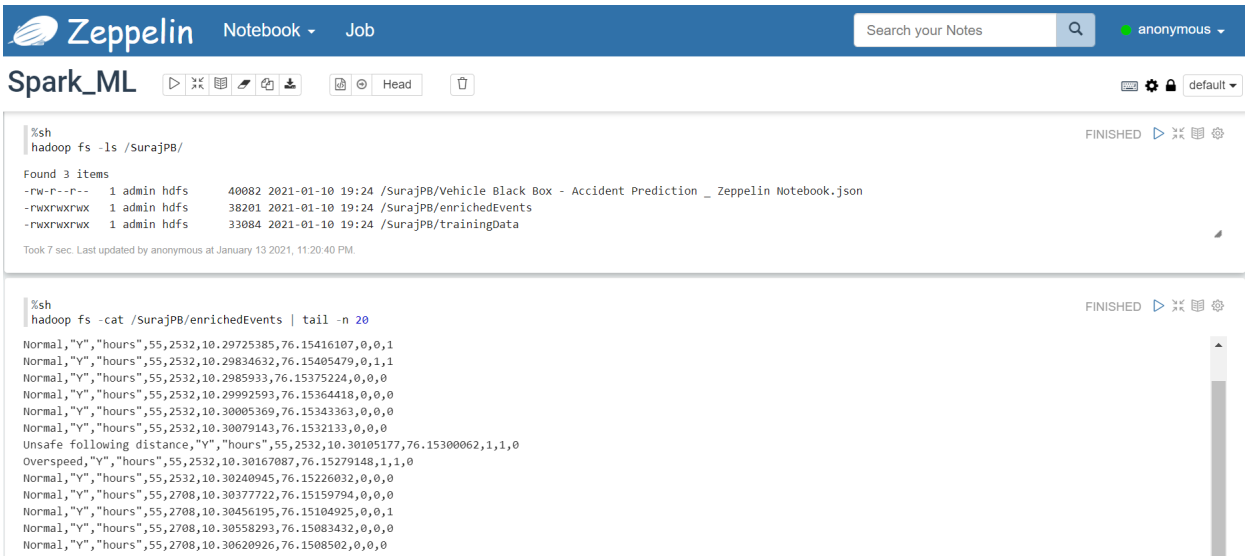

Figure 7.1: Inserted Dataset into HDFS

We have used sample features like eventType (eventType attribute column contains different driving events like Normal, Lane Departure, Over speed, Unsafe following distance, Unsafe tail distance), iscertified (Whether driver is certified or Not - Y / N), paymentscheme (wage plan : per miles or per hour), hoursdriven (in digits), milesdriven (in digits), isfoggy (foggy weather or not, Yes – 1, No - 0), israiny (Rainy weather or not, Yes – 1, No - 0),iswindy (Windy weather or not, Yes – 1, No - 0), latitude and longitude.

The entry point into all functionality in Spark SQL is the SQLContext class, or one of its descendants. To create a basic SQLContext, all you need is a SparkContext.

val sqlContext = new org.apache.spark.sql.SQLContext(sc) // sc is an existing SparkContext

val eventsFile = sc.textFile("hdfs:/SurajPB/enrichedEvents") // Create an RDD

```
val sqlContext = new org.apache.spark.sql.SQLContext(sc)

val eventsFile = sc.textFile("hdfs:/SurajPB/enrichedEvents")

case class Event(eventType: String,
                 isCertified: String,
                 paymentScheme: String,
                 hoursDriven: Int,
                 milesDriven: Int,
                 lati: Float,
                 longi: Float,
                 isFoggy: Int,
                 isRainy: Int,
                 isWindy: Int)

val eventsRDD = eventsFile.map(s => s.split(",")).map(
    s => Event(s(0),        //eventType
            s(1).replaceAll("\"", ""),  //isCertified
            s(2).replaceAll("\"", ""),  //paymentScheme
            s(3).toInt,     //hoursDriven
            s(4).toInt,     //milesDriven
            s(5).toFloat,   //latitude
            s(6).toFloat,   //longitude
            s(7).toInt,     //isFoggy
            s(8).toInt,     //isRainy
            s(9).toInt      //isWindy
        )
    )


eventsRDD.count

eventsRDD.toDF().registerTempTable("enrichedEvents")
```

Figure 7.2: Created events RDD and pushed

```
%sql
select * from enrichedEvents order by hoursDriven desc limit 10
```

READY

eventType	isCertified	paymentScheme	hoursDriven	milesDriven	lati	longi	isFoggy	isRainy	isWindy
Normal	N	miles	90	4300	10.191698	76.20155	0	0	1
Lane Departure	N	miles	90	4300	10.202711	76.20025	1	1	1
Normal	N	miles	90	4300	10.192757	76.20156	0	0	0
Overspeed	N	miles	90	4300	10.192967	76.20139	1	0	0
Unsafe tail distance	N	miles	90	4300	10.194767	76.200935	1	1	1
Normal	N	miles	90	4300	10.197733	76.200645	0	0	0
Normal	N	miles	90	4300	10.198903	76.200455	0	0	1
Normal	N	miles	90	4300	10.199647	76.200165	0	0	0
Lane Departure	N	miles	90	4300	10.20092	76.20019	1	0	0

Figure 7.3: enriched Events

Whether certified drivers have less violations?

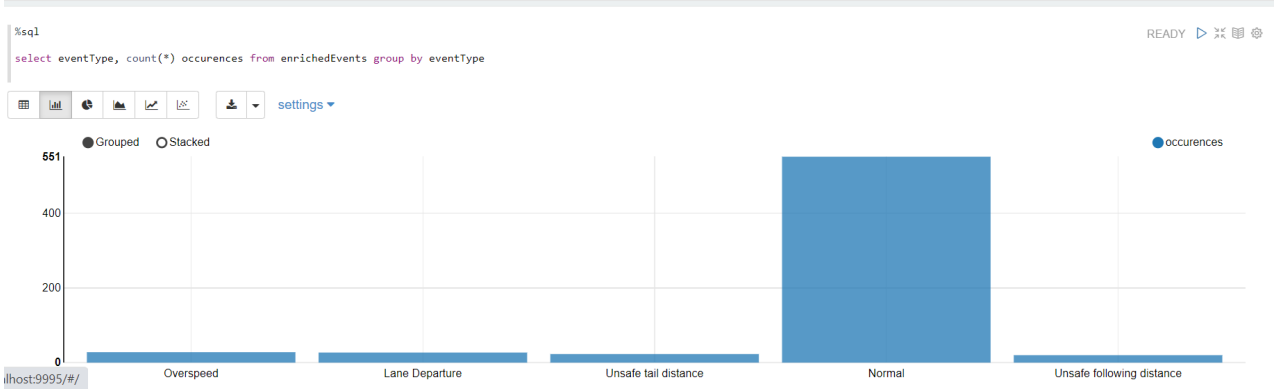

Figure 7.4: Impact of Certified Drivers in violation

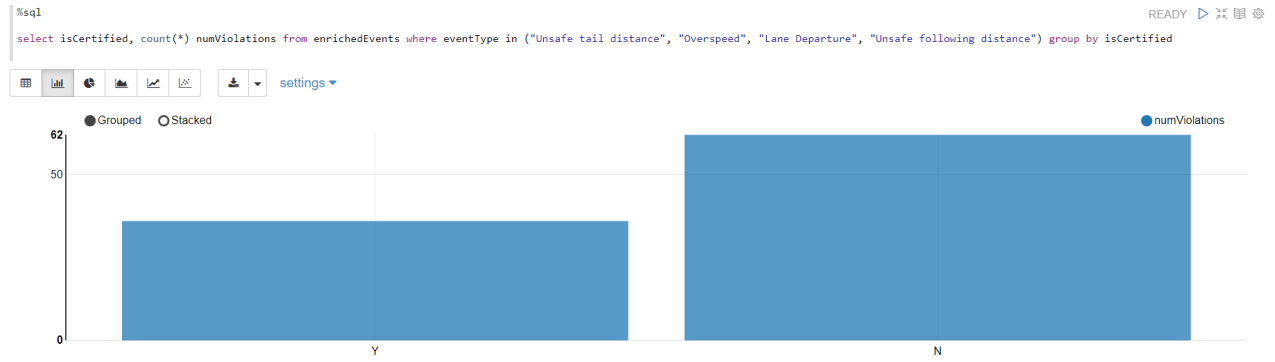

Figure 7.5: Concludes Certified Drivers deoesnt have much impacts in violation

Do driver fatigue cause violations? - Pending to implement CNN based fatigue detection and fed into model

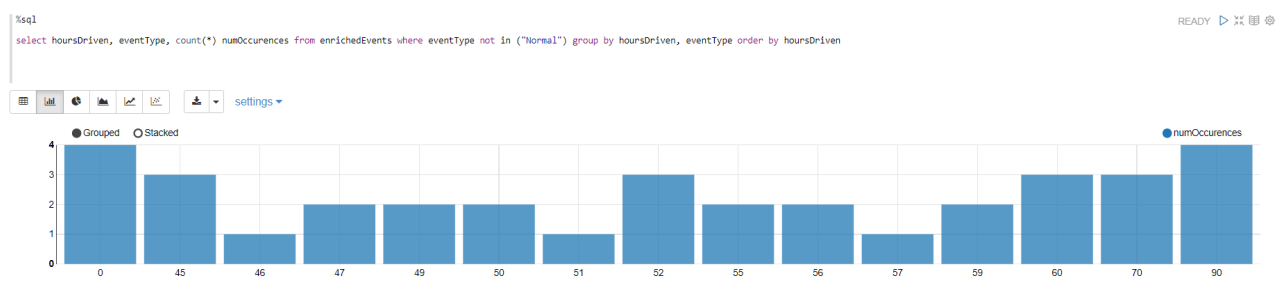

Figure 7.6: Driver fatigue Vs violations

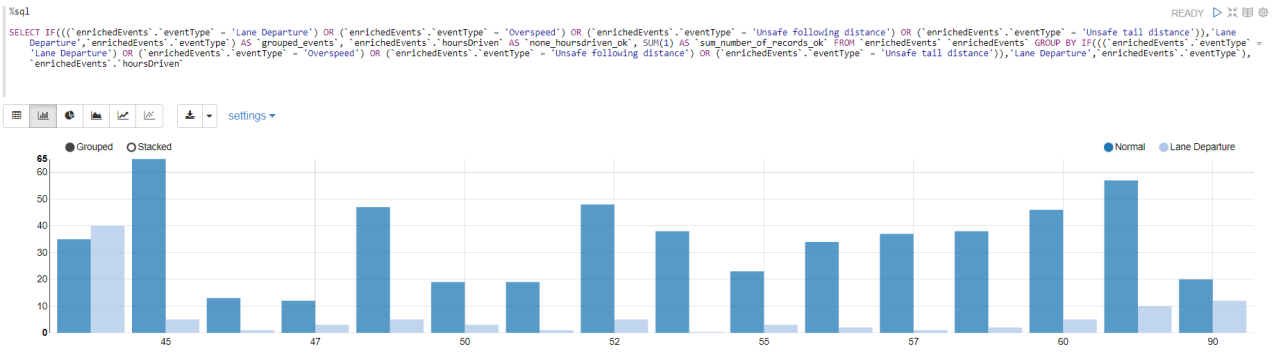

Figure 7.7: Hours driven Vs Lane Departure

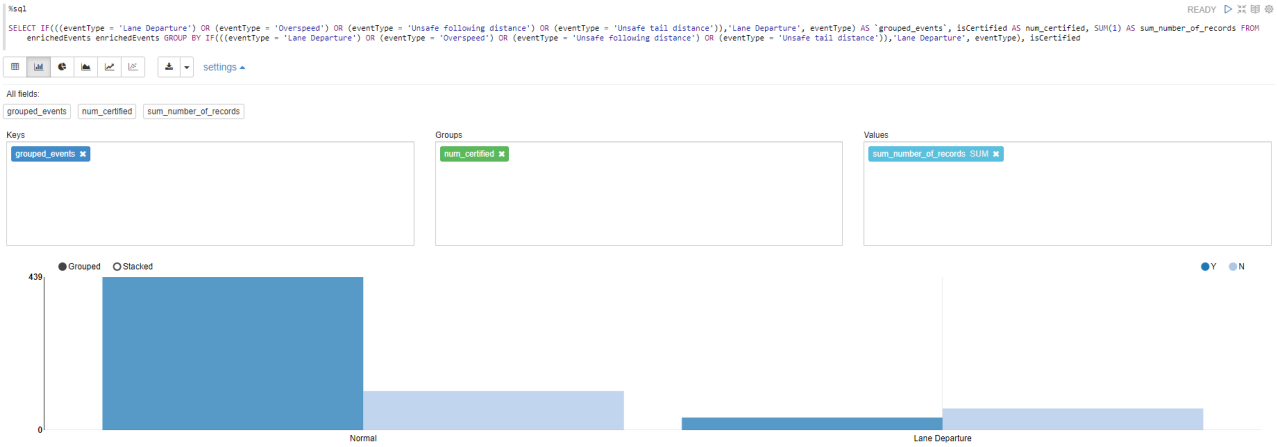

Figure 7.8: Seems Certified Drivers have less Lane Departure

What's the impact of fog on driving?

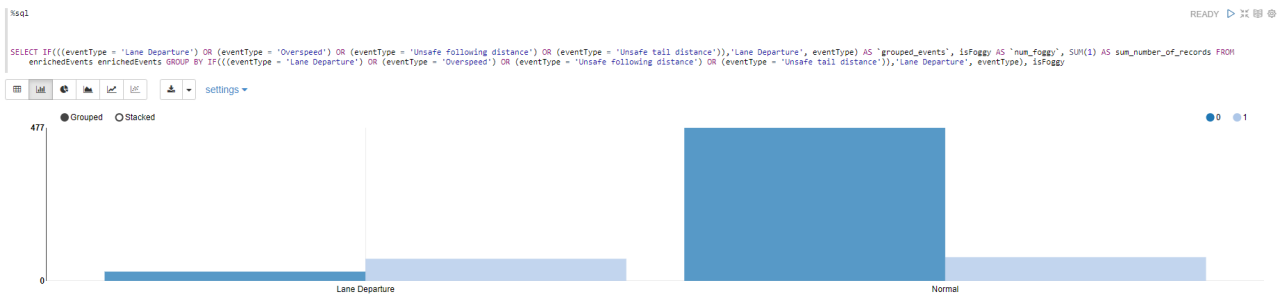

Figure 7.9: Fog weather Vs Lane Departure

Figure 7.10: Rain weather Vs Lane Normal driving

Figure 7.11: Impact of Wind on driving

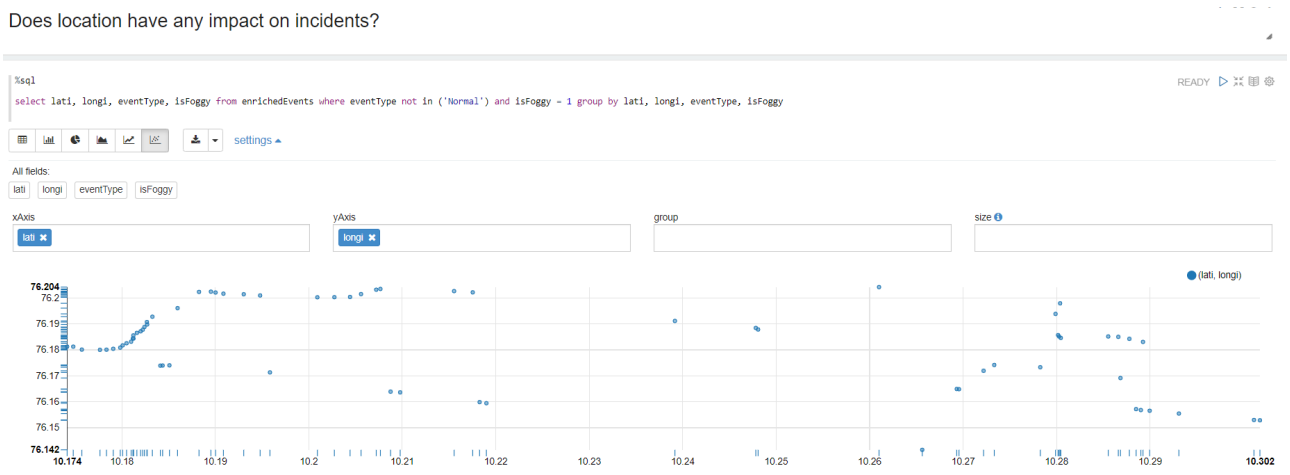

Figure 7.12: Location and Incident

Let's build a regression model to predict violations

Took 0 sec. Last updated by anonymous at November 19 2020, 4:57:37 PM.

```
import org.apache.spark.mllib.util.MLUtils;
import org.apache.spark.mllib.optimization;
import org.apache.spark.mllib.classification.LogisticRegressionWithSGD;
import org.apache.spark.mllib.optimization.SquaredL2Updater;
import org.apache.spark.mllib.evaluation.BinaryClassificationMetrics;

    val examples = MLUtils.loadLabeledPoints(sc,"hdfs:///SurajPB/MidTermReview/SampleSensorDataFromEdgeDevices/trainingData").cache()

    val splits = examples.randomSplit(Array(0.8, 0.2))
    val training = splits(0).cache()
    val test = splits(1).cache()

    val numTraining = training.count()
    val numTest = test.count()
    println(s"Training: $numTraining, test: $numTest.")

    val updater = new SquaredL2Updater()

    val model = {
       val algorithm = new LogisticRegressionWithSGD()
       algorithm.optimizer.setNumIterations(200).setStepSize(1.0).setUpdater(updater).setRegParam(0.1)
       algorithm.run(training).clearThreshold()
    }

    val rprediction = model.predict(test.map(_.features))
    val rpredictionAndLabel = rprediction.zip(test.map(_.label))
    val rmetrics = new BinaryClassificationMetrics(rpredictionAndLabel)

    println("\n");
    println("Certification Weight " + model.weights(0))
    println("Wage Plan Weight " + model.weights(1))
    println("Fatigue by Hours Weight " + model.weights(2))
    println("Fatigue by Miles Weight " + model.weights(3))
    println("Foggy weather Weight " + model.weights(4))
    println("Rainy weather Weight " + model.weights(5))
    println("Windy weather Weight " + model.weights(6))
    println("\n")
    println("Intercept " + model.intercept)
```

Figure 7.13: Regression model

We are building a logistic regression model here, by calculating the weights. In the output we can see the weights. More the weight means it has more impact on 'predicting the label'. The label will be violation occurs or not. So we can identify which weather condition is most weighted feature and which may cause more violations. So in such a weather we will alert the driver to take more precautions.

```
println("\n");
println("Certification Weight " + model.weights(0))
println("Wage Plan Weight " + model.weights(1))
println("Fatigue by Hours Weight " + model.weights(2))
println("Fatigue by Miles Weight " + model.weights(3))
println("Foggy weather Weight " + model.weights(4))
println("Rainy weather Weight " + model.weights(5))
println("Windy weather Weight " + model.weights(6))
println("\n")
println("Intercept " + model.intercept)
println(s"Test areaUnderPR = ${rmetrics.areaUnderPR()}.")
println(s"Test areaUnderROC = ${rmetrics.areaUnderROC()}.")
```

```
examples: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint] = MapPartitionsRDD[1744] at map at MLUtils.scala:216
splits: Array[org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint]] = Array(MapPartitionsRDD[1745] at randomSplit at <console>:109, MapPartitionsRDD[17
training: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint] = MapPartitionsRDD[1745] at randomSplit at <console>:109
test: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint] = MapPartitionsRDD[1746] at randomSplit at <console>:109
numTraining: Long = 1080
numTest: Long = 279
Training: 1080, test: 279.
updater: org.apache.spark.mllib.optimization.SquaredL2Updater = org.apache.spark.mllib.optimization.SquaredL2Updater@2848d772
warning: there was one deprecation warning; re-run with -deprecation for details
model: org.apache.spark.mllib.classification.LogisticRegressionModel = org.apache.spark.mllib.classification.LogisticRegressionModel: intercept = 0.0, numFeatures = 7, num
rprediction: org.apache.spark.rdd.RDD[Double] = MapPartitionsRDD[1925] at mapPartitions at GeneralizedLinearAlgorithm.scala:70
rpredictionAndLabel: org.apache.spark.rdd.RDD[(Double, Double)] = ZippedPartitionsRDD2[1927] at zip at <console>:110
rmetrics: org.apache.spark.mllib.evaluation.BinaryClassificationMetrics = org.apache.spark.mllib.evaluation.BinaryClassificationMetrics@37dce6cc
Certification Weight 0.0
Wage Plan Weight 0.0
Fatigue by Hours Weight -1.0138246179847925
Fatigue by Miles Weight -0.5042026909720859
Foggy weather Weight 0.07928402584672738
```

Figure 7.14: Model Train and Tests

```
println("\n");
println("Certification Weight " + model.weights(0))
println("Wage Plan Weight " + model.weights(1))
println("Fatigue by Hours Weight " + model.weights(2))
println("Fatigue by Miles Weight " + model.weights(3))
println("Foggy weather Weight " + model.weights(4))
println("Rainy weather Weight " + model.weights(5))
println("Windy weather Weight " + model.weights(6))
println("\n")
println("Intercept " + model.intercept)
println(s"Test areaUnderPR = ${rmetrics.areaUnderPR()}.")
println(s"Test areaUnderROC = ${rmetrics.areaUnderROC()}.")
```

```
numTest: Long = 279
Training: 1080, test: 279.
updater: org.apache.spark.mllib.optimization.SquaredL2Updater = org.apache.spark.mllib.optimization.SquaredL2Updater@2848d772
warning: there was one deprecation warning; re-run with -deprecation for details
model: org.apache.spark.mllib.classification.LogisticRegressionModel = org.apache.spark.mllib.classification.LogisticRegressionModel: intercept = 0.0, numFeatures = 7, num
rprediction: org.apache.spark.rdd.RDD[Double] = MapPartitionsRDD[1925] at mapPartitions at GeneralizedLinearAlgorithm.scala:70
rpredictionAndLabel: org.apache.spark.rdd.RDD[(Double, Double)] = ZippedPartitionsRDD2[1927] at zip at <console>:110
rmetrics: org.apache.spark.mllib.evaluation.BinaryClassificationMetrics = org.apache.spark.mllib.evaluation.BinaryClassificationMetrics@37dce6cc
Certification Weight 0.0
Wage Plan Weight 0.0
Fatigue by Hours Weight -1.0138246179847925
Fatigue by Miles Weight -0.5042026909720859
Foggy weather Weight 0.07928402584672738
Rainy weather Weight 0.037369653570587906
Windy weather Weight -0.3960081383034084
Intercept 0.0
Test areaUnderPR = 0.35399931616529823.
Test areaUnderROC = 0.614470688673876.
```

Figure 7.15: Classification metric

Chapter 8

Stream Processing Analytics on Vehicle Sensor data

A stream processing Architecture is needed:

- To bring real time data (Sensor data and Weather API and traffic congestion data) - Data Simulator should run in the local environment as of now.
- To Combine Real time data with Static data – History and Real time
- To create a machine learning model to use the past data to detect the incident pattern - Done in Last Chapter
- After building the model, for each incoming real time event to apply the model and check if it matches to predict possibilities of Accidents and Possible maintenance to the Vehicle.

Hence for the above requirements we needed following tools to process the data from data simulator

Technology	Use
Apache Nifi	To bring the events from Sensors of Vehicle using MQTT protocol
Kafka	As a message Queue between Producer and Consumer
Storm	Process real time events
Hadoop	Storage layer and Analysis
Hbase	Database
Spark	To build Machine Learning Model

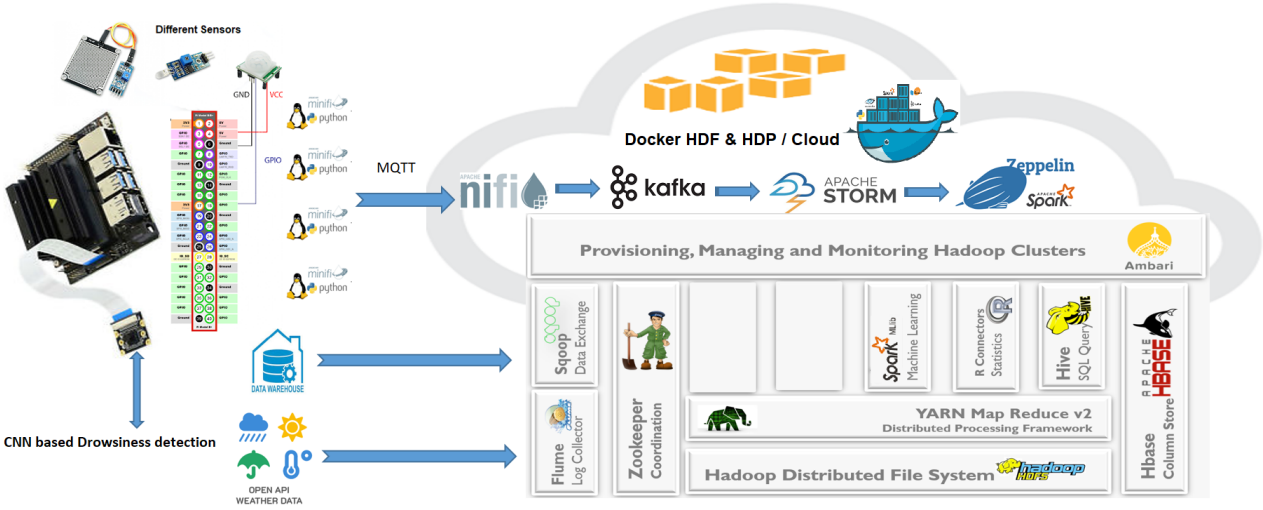

Figure 8.1: Big data / Stream Processing Architecture

8.1 Data Ingestion Layer

At the edge hardware level, sensors collect information on the digital world and send them to a gateway through a variety of wired and wireless protocols (Serial, RS-232, USB, WiFi, and so on). In our project, we will use various sensors (optical sensors for speed calculation, temperature sensors, cameras, water sensors, GPS, Left and Right indicator switches, horn switches, brake switches, buzzers, Piezo sensors and so on) that send data to the gateway via WiFi and internet. The gateway is a Raspberry Pi running a Mosquitto Broker and a MiNiFi agent. Mosquitto is an Open Source, lightweight messaging broker that we use to expose sensor data through the MQTT protocol. MQTT has a minimal footprint which makes it suitable for IoT apps and resource constrained hardware, such as raspberry pi or microcontrollers.

Apache MiNiFi — a subproject of Apache NiFi — is a light-weight agent that implements the core features of Apache NiFi, focusing on data collection at the edge.MiNiFi design goals are: small size and low resource consumption, central management of agents, and edge intelligence. MiNiFi can be easily integrated with NiFi through Site-to-Site protocol (S2S) to build an end-to-end flow management solution which is scalable, secure. In our system, MiNiFi will subscribe to all the topics of the Mosquitto broker, and forward every new message to NiFi at the Regional level

MiNiFi C2 Server (MiNiFi Commande and Control) is another subproject of Apache NiFi currently under development. Its role is to provide a central point of configuration to hundreds or thousands of MiNiFi agents in the wild. The C2 server manages versioned classes of applications (MiNiFi flow configurations) and exposes them through

a Rest API. MiNiFi agents can connect to this API at a defined frequency to update their configuration.

8.1.1 Preparing the Raspberry Pi : MQTT and MiNiFi

To install Mosquitto MQTT broker and MiNiFi agent, run the following commands on Raspberry Pi.
sudo apt-get update : install and run Mosquitto broker on default port 1883
sudo apt-get install mosquittomosquitto : install and prepare MiNiFi agent
wget http://apache.crihan.fr/dist/nifi/minifi/0.4.0/minifi-0.4.0-bin.tar.gztar -xvf minifi-0.4.0-bin.tar.gzcd minifi-0.4.0 : add mqtt processor

To have a small size, MiNiFi is packaged with a minimal set of default processors. It's possible to add any NiFi processor by deploying the NAR (NiFi Archive) in the lib directory.By default, configuring a MiNiFi agent requires editing the file ./conf/config.yml to include the list of used processors and their configurations. The configuration can be written manually, or designed using the NiFi UI and exporting the flow as a template. The template is an XML file that we need to convert to a YML file with the MiNiFi toolkit.

MiNiFi uses a "Change Ingestor" by which the agent is notified of a potential new configuration. Change ingestors are pluggable modules, and currently three ingestors are:

1. FileChangeIngestor
2. RestChangeIngestor
3. PullHttpChangeIngestor

We will use a PullHttpChangeIngestor to query a C2 server every period of time and download any new configuration available. To configure this ingestor, edit the file ./conf/bootstrap.conf, uncomment the corresponding lines, and set the ingestor properties as follows:

nifi.minifi.notifier.ingestors=org.apache.nifi.minifi.bootstrap.configuration.ingestors.PullHttpChangeIngestor
nifi.minifi.notifier.ingestors.pull.http.hostname=c2-server
nifi.minifi.notifier.ingestors.pull.http.port=10080
nifi.minifi.notifier.ingestors.pull.http.path=/c2/config
nifi.minifi.notifier.ingestors.pull.http.query=class=iot-minifi-raspberry-agent
nifi.minifi.notifier.ingestors.pull.http.period.ms=60000

With this configuration, each MiNiFi agent will query the C2 Server REST API at http://c2-server:10080/c2/config every 1 minute and ask for the latest configuration for the "iot-minifi-raspberry-agent" class.

8.1.2 Installing and configuring the MiNiFi C2 Server

Install MiNiFi C2 server on company data center that's reachable from the MiNiFi agents. Use hierarchical C2 deployment for network constrained applications as described a few lines below. Run the following command to install the C2 server:

wget http://apache.crihan.fr/dist/nifi/minifi/0.4.0/minifi-c2-0.4.0-bin.tar.gztar -xvf minifi-c2-0.4.0-bin.tar.gzcd minifi-c2-0.4.0

Add a consumeMQTT processor to subscribe to the Mosquitto broker and subscribe to all topics under iot/sensors. Note that the tcp://raspberrypi:1883 here is equivalent to tcp://localhost:1883, since this flow will be running on the Raspberry Pi.

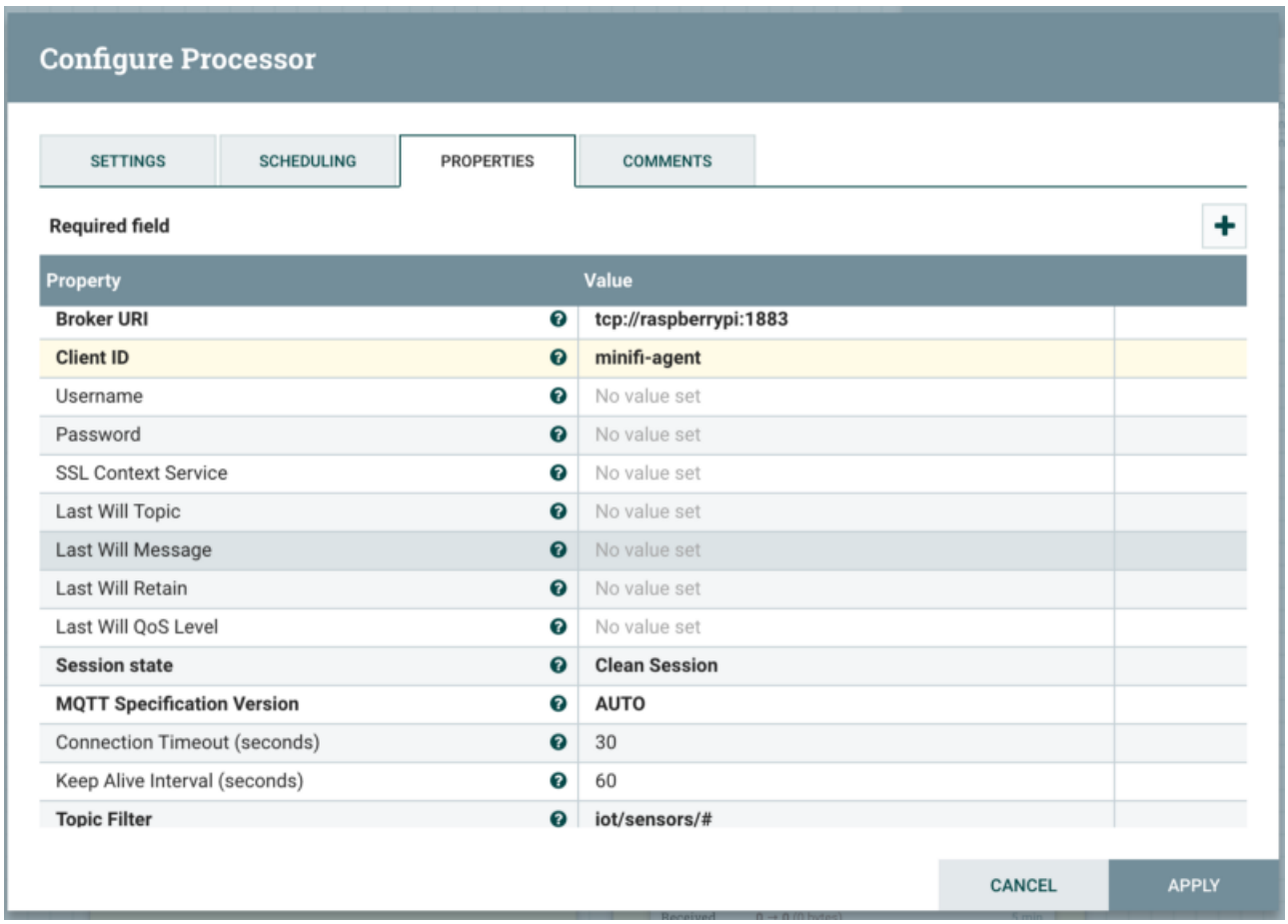

Figure 8.2: Raspberry Pi Nifi UI Processor configuration

Use an UpdateAttribute processor to add a "version" attribute that we will use to show the re-configuration feature. You can add any attribute you want: timestamp,

agent name, location, and so on. And finally, add a Remote Process Group (RPG) to send the consumed events to NiFi.

The Nifi flow now looks like as below. The left flow will be running in NiFi to receive data from MiNiFi. The right flow here is only for design and will effectively run on each Raspberry Pi.

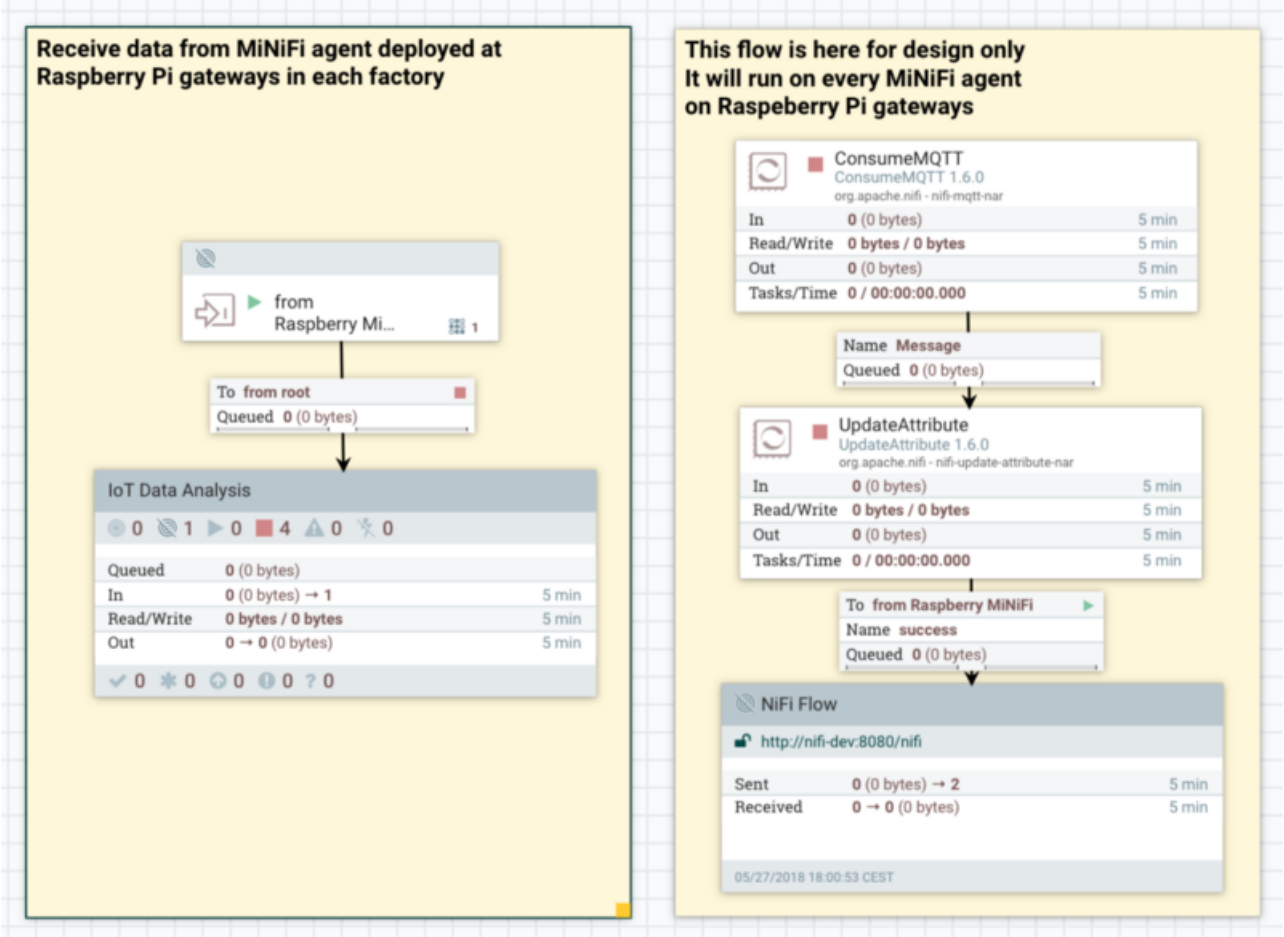

Figure 8.3: Nifi Work flow for Raspberry Pi Minifi Agent

Deploy and Start the application and verify Rest API calls. Also, the MQTT shows that the MiNiFi agent connected to the broker and subscribed to the topics iot/sensors.

8.1.3 Data Simulator to generate Big Data events

Minify in the vehicle edge hardware system gathering data from the sensors and sends the data to Nifi cluster (As long as we use local machine docker container in order to avoid the cloud bill, we can go ahead with Data simulator without using Minifi, which could be done inside the Nifi cluster itself, to generate the events to process by Stream processing engines). Once we commercialize the product we can add cloud app

end-point to connect with edge hardware node.

In Nifi cluster, we parse, filter and enrich the data then we publish the data to Kafka topic. Nifi processing flow chart : We get vehicle data and traffic data as one stream We separate Vehicle data and Traffic data based on the field attributes, while processing the vehicle data say Vehicle Truck, we enrich each data events (enrich includes: adding more value to the data for each event received , we add weather data gathered at that time of event), for e.g we have received event from Suraj's vehicle, event contain the speed at which vehicle moves at 7:30am, braking system, engine oil level, how close to other vehicles etc. We also got weather data for Suraj's location. Now we will enrich Vehicle event data by joining weather data received. Now after enriching we have an event like Mr. Suraj is driving at 50km per hour, maintaining good distance with other vehicles, braking system is good, and now driving in Heavy rain. On the other hand we process the traffic data, real time traffic data will receive from Google servers. We use this traffic data in future to direct the driver the best route. We read comma separated traffic data and we convert into Avro format. Nifi will publish the enrich event data and enriched weather data to separate Kafka topic.. Now we have to build and implement this flow in Nifi. Nifi supports flow management , assures guaranteed delivery, assures data buffering with prioritized queuing.

There is a data simulator that replicates MiNiFi's place in the data flow on IoT edge hardware module, MiNiFi is embedded on the vehicles edge node, so the simulator generates truck and traffic data here for the project demo purpose. NiFi ingests this sensor data. NiFi's flow performs pre-processing on the data to prepare it and to sent to Kafka.

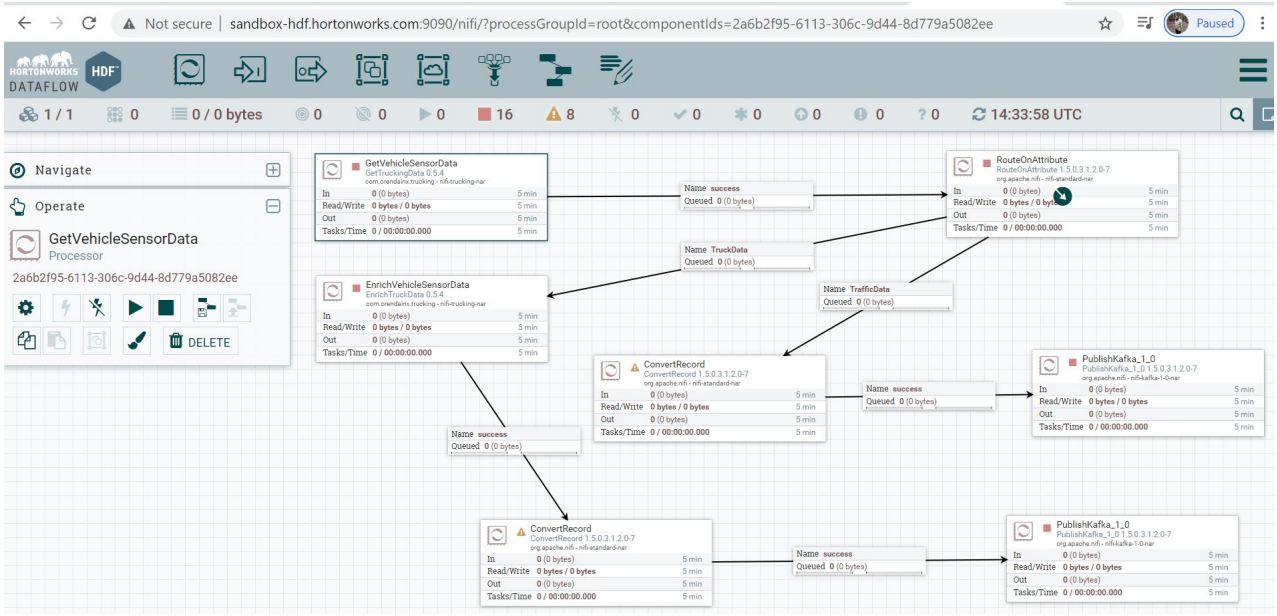

Figure 8.4: Data Pipeline inside NiFi

To run the Nifi Flow : Let's select the entire data flow. In the Operate Pallete, click on the start button start-button and let it run for 1 minute. The red stop symbols red-symbol at the corner of each component in the dataflow will turn to a green play symbol green-symbol. We can see the numbers in the connection queues change from 0 to a higher number indicating that the data is being processed.

8.1.4 Build NiFi DataFlow for Data Simulator

Lets see the process behind configuring controller services and configuring processors to build this NiFi DataFlow

As the first step in building the DataFlow, we needed to setup NiFi Controller Service called HortonworksSchemaRegistry. Set Properties Tab of this Controller Service

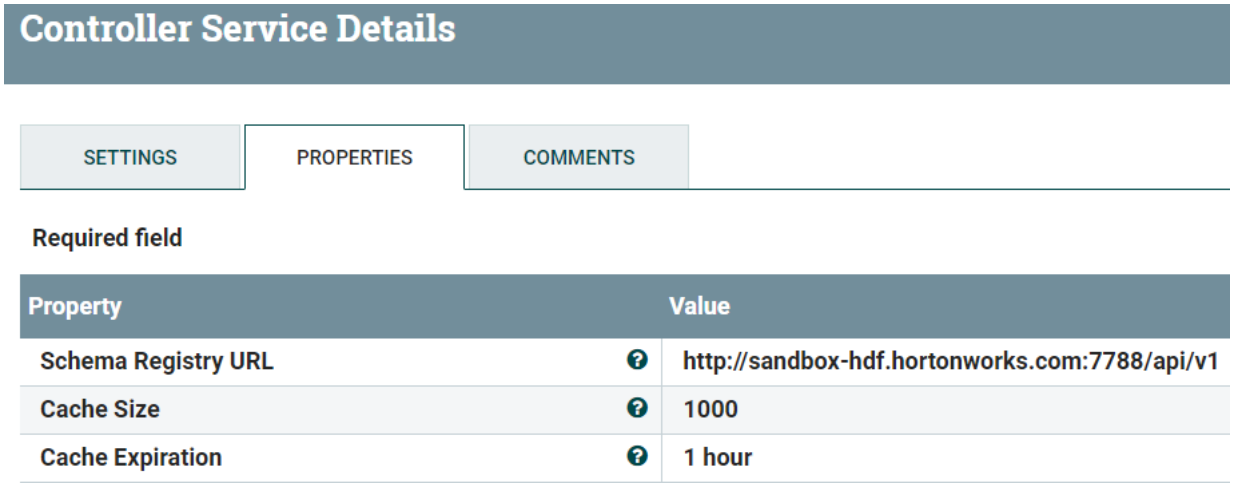

Figure 8.5: Properties Tab of HortonworksSchemaRegistry Controller Service

A schema is used for categorizing the data into separate categories: TruckData and TrafficData will be applied on the data during the use of the ConvertRecord processor.

From the configuration in the table above, we can see the URL that allows NiFi to interact with Schema Registry, the amount of cache that can be sized from the schemas and the amount of time required until the schema cache expires and NiFi has to communicate with Schema Registry again.

NiFi Data Simulator -
Generates data of two types: TruckData and TrafficData as a CSV string. We generate this simulator data with NAR file. NAR stands for NiFi Archive. The reason for creating a custom packaging for a NiFi processor is to provide a bit of Java ClassLoader isolation. The processors and controller services all come from different companies and contributors, utilizing differing versions of libraries (such as apache-commons, etc. . .). A NAR file provides isolation from the potential issue of NoClassDefFoundError exceptions being thrown for having the wrong version of a dependency already loaded in the ClassLoader from a different processor.

A NAR file essentially mirrors the Java Web application ARchive (WAR) or Java Application ARchive (JAR) with a few differences. Instead of a WEB-INF folder that would be present in a WAR file, the root directory is META-INF (as would be in a JAR). Under the META-INF root directory, are descriptive files such as LICENSE, DEPENDENCIES (which list the license information of the bundled dependencies), and NOTICE (which contains the license of the processor itself). Additionally, there are 3 key files/directories under META-INF as well. First is the MANIFEST.MF file. This is the same as the manifest file that is generated for any jar, however, this one includes a Nar-id that is used to identify the nar to the NiFi Nar Unpacker. The later

versions of NiFi also contain Nar-Version and dependency manifest entries that will be included to support workflows such as processor versioning. It also includes helpful metadata around the version of Java and Maven used to build the NAR and where it came from.

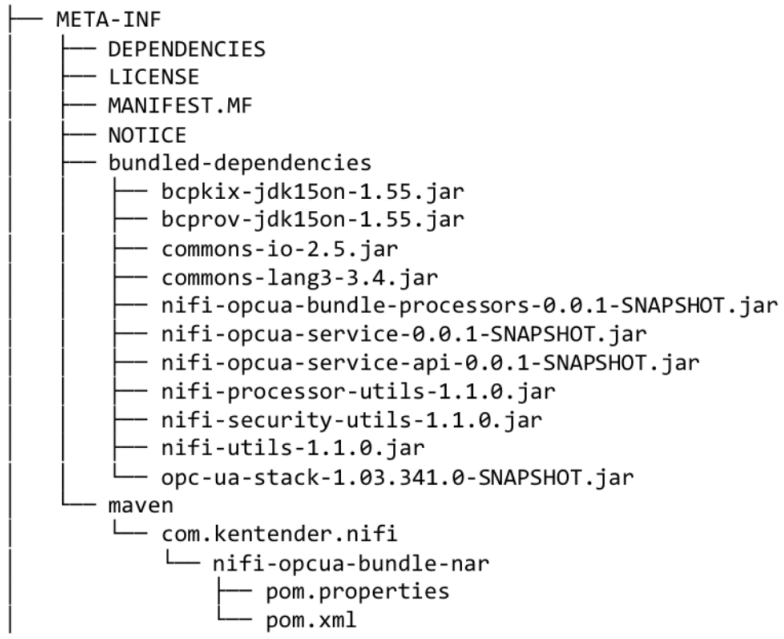

Figure 8.6: NAR File Structure

Bundled-dependencies :
The bundled-dependencies contains the actual jar files that will be used by the processor and accompanying controller services (if the NAR contains a controller service). These jar files will also be loaded in the ClassLoader that is dedicated to that processor.

To Build A Nar :
The easiest way to build a NAR is to start development using the Maven Archtype for processors or controller services, this will create a maven project the NAR plugin as well as any dependencies needed from the framework in to the controller service or property.The effective POM snippet from the Archtype shows the NAR plugin.NAR is just a modified WAR packaging (which in-itself is a lightly-modified packaging structure from JAR). Therefore the easiest way to unpack a NAR file is to use the jar command line utility. Assuming you have the JRE bin directory in your system path the instruction would be:

jar –xvf <nar-file-name.nar>

```
<plugin>
   <groupId>org.apache.nifi</groupId>
   <artifactId>nifi-nar-maven-
plugin</artifactId>
   <version>1.1.0</version>
   <extensions>true</extensions>
   <executions>
     <execution>
       <id>default-nar</id>
       <phase>package</phase>
       <goals>
         <goal>nar</goal>
       </goals>
     </execution>
   </executions>
 </plugin>
```

Figure 8.7: NAR plugin POM snippet

RouteOnAttribute - Filters TruckData and TrafficData types into two separate flows from GetTruckingData. The properties should be defined as below

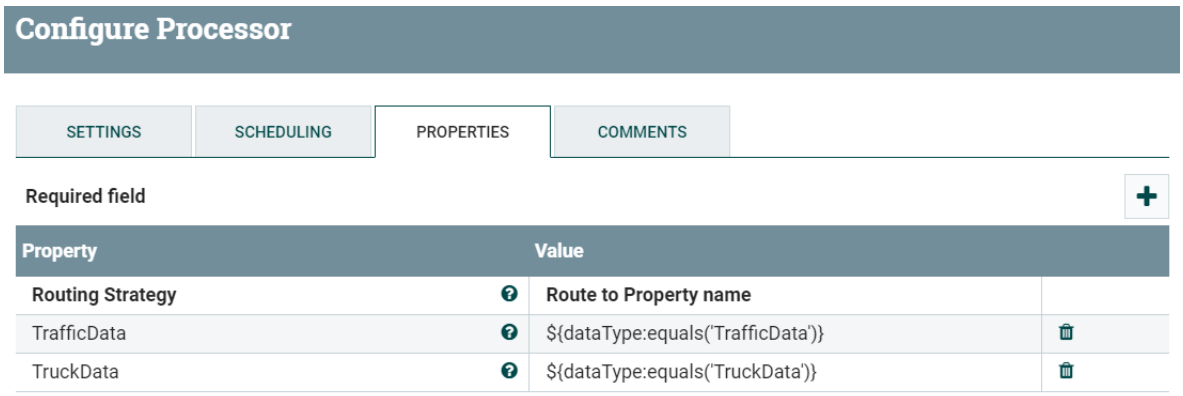

Figure 8.8: RouteonAttribute

EnrichTruckData -
Adds weather data (fog, wind, rain) to the content of each flowfile incoming from RouteOnAttribute's TruckData queue.
ConvertRecord -
Uses Controller Service to read in incoming CSV Truck/TrafficData FlowFiles from the EnrichTruckData processor and uses another Controller Service to transform CSV to Avro TruckData FlowFiles.

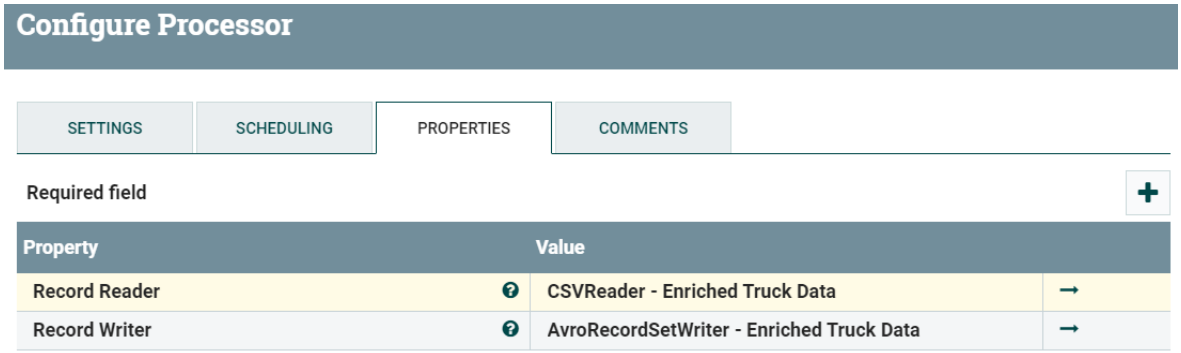

Figure 8.9: Convert CSV to Avro

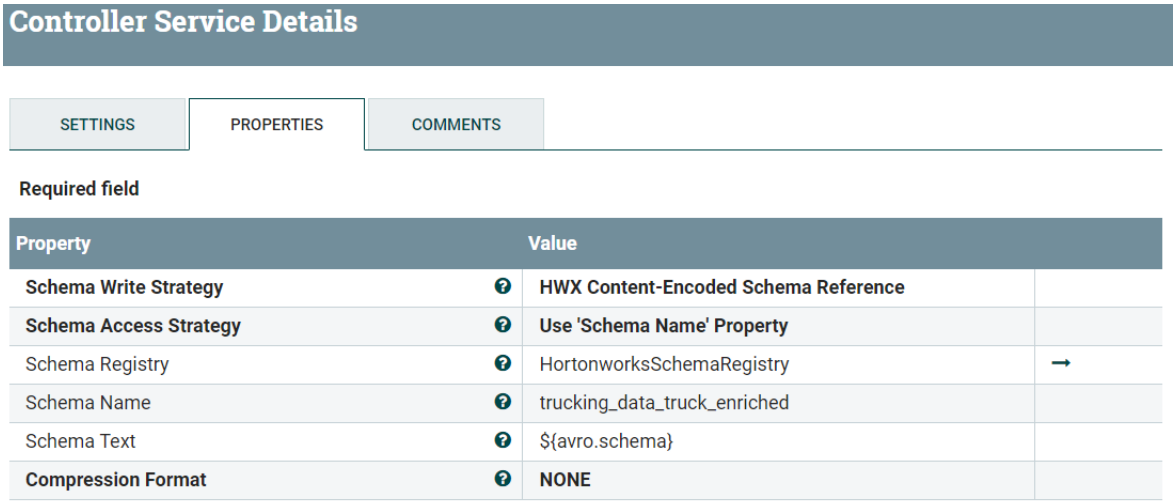

Figure 8.10: Properties Tab of Controller Service

PublishKafka_1_0: Truck/TrafficData:

PublishKafka_1_0 - Receives flowfiles from ConvertRecord - Truck/TrafficData processor and sends each flowfile's content as a message to Kafka Topic: trucking_data_truck and trucking_data_traffic respectively using the Kafka Producer API.
End point should be : sandbox-hdf.hortonworks.com:6667 as put in the configuration

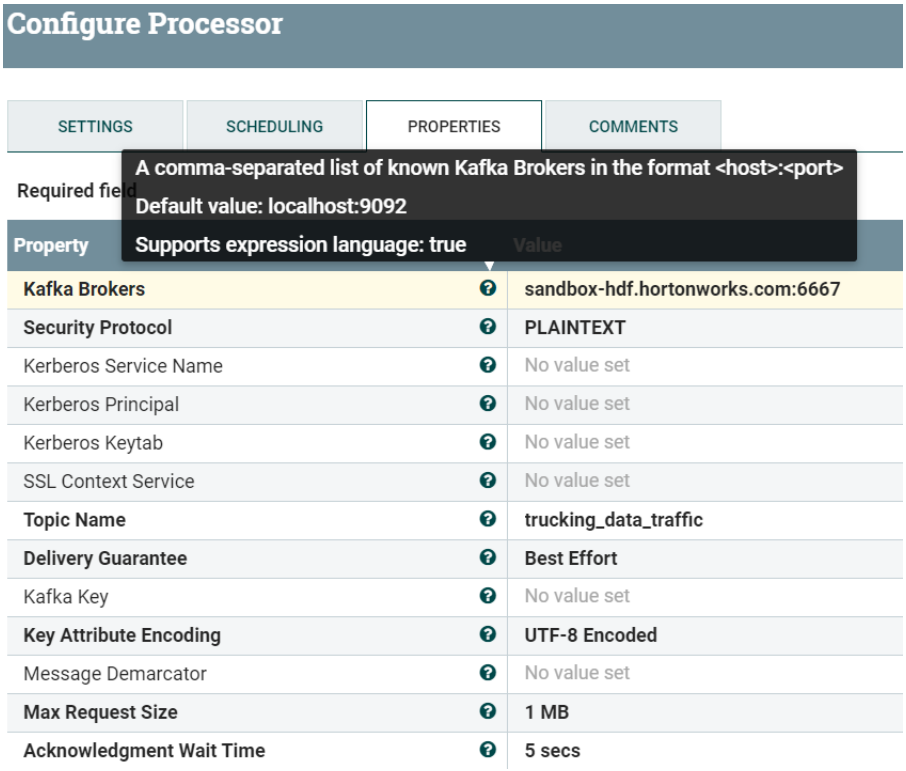

Figure 8.11: Properties Tab of Publish Kafka

8.1.5 Kafka in our System

Messaging systems transfer data between client applications. One application produces the data, such as reading from sensors embedded on vehicles and the other application receives the data, processes it to be ready to be visualized to show the characteristics about the drivers driving behavior who drive those vehicles. As you can see each application developer can focus on writing code to analyze the data and not worry about how to share the data. There are two messaging systems used in this scenario, point to point and publish subscribe. The system most often used is publish and subscribe. Hence Pub-Sub messaging system work as

- publisher sends messages into 1 or more topics
- subscribers can arrange to receive 1 or more topics, then consume all the messages

Architectural view :

- NiFi Producer : Produces a continuous real-time data feed from truck sensors and traffic information that are separately published into two Kafka topics using a NiFi Processor implemented as a Kafka Producer.

- Kafka Cluster : Has 1 or more topics for supporting 1 or multiple categories of messages that are managed by Kafka brokers, which create replicas of each topic (category queue) for durability.
- Storm Consumer : Reads messages from Kafka Cluster and emits them into the Apache Storm Topology to be processed.

Before we can perform Kafka operations on the data, we must first have data in Kafka, so let's run the NiFi DataFlow Application.

1. Persist Data Into Kafka Topics : A NiFi simulator generates data of two types: TruckData and TrafficData as a CSV string. There is some preprocessing that happens on the data to prepare it to be split and sent by NiFi's Kafka producers to two separate Kafka Topics: trucking_data_truck and trucking_data_traffic.
2. List Kafka Topics : From the terminal, we can see the two Kafka Topics that have been created:

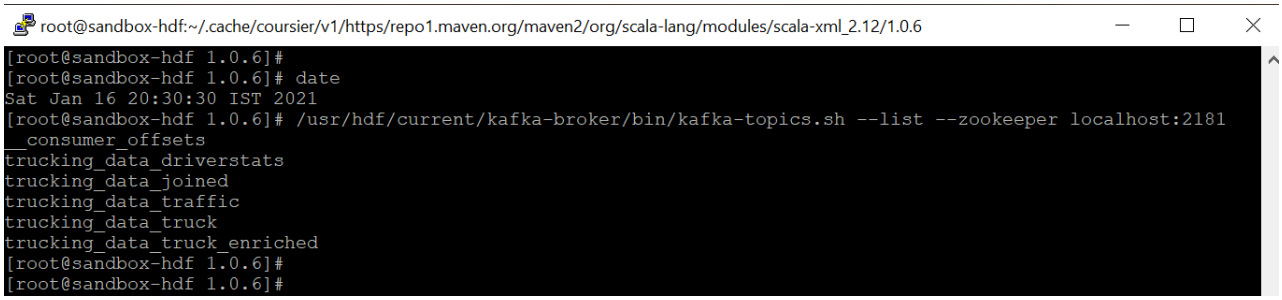

Figure 8.12: List Kafka Topics

3. View data in Kafka topics : As messages are persisted into the Kafka Topics from the producer, you can see them appear in each topic by writing the following commands: View Data for Kafka Topic: trucking_data_truck_enriched:

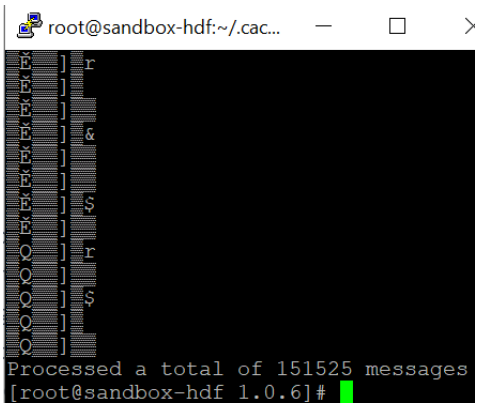

Figure 8.13: Data for Kafka Topic - Traffic Data

Figure 8.14: Data for Kafka Topic - Trucking data

We can notice that the is data encoded in a format we cannot read, this format is necessary for Schema Registry. The reason we are using Schema Registry is because we need it for Stream Analytics Manager to pull data from Kafka.

Starting the Producer to Send Messages:

In our project, we utilize a dataflow framework known as Apache NiFi to generate our sensor truck data and online traffic data, process it and integrate Kafka's Producer API, so NiFi can transform the content of its flowfiles into messages that can be sent to Kafka. Start all the processors in the NiFi flow including the Kafka one and data will be persisted into the two Kafka Topics.

Starting the Consumer to Receive Messages:

We utilize a stream processing framework known as Apache Storm to consume the messages from Kafka. Storm integrates Kafka's Consumer API to pull in messages from the Kafka brokers and then perform complex processing and send the data to destinations to be stored or visualized.

Submit the Storm topology and messages from the Kafka Topics will be pulled into Storm.

8.1.6 Storm in Our Use-case

Imagine our Organization VodafoneIdea (Vi) have a plan to develop an end to end IoT product and service for reducing no. of accident cases in India. As a first phase, we have identified that a particular vehicle category, say Truck has more impact to Vehicle accidents due to continuous night driving across the country. So trucks are outfitted with sensors that collect data - data like the name of the driver, the route the truck is bound for, the speed of the truck, and even what event recently occurred (speeding, the truck weaving out of its lane, following too closely, etc). Data like this is generated very often, say once per second and is streamed back to our organization servers that supports big data stream analytics.

Additionally, the company is also polling an internet service for information about traffic congestion on the different trucking routes.

The company needs a way to process both of these streams of data and combine them in such a way that data from the truck is combined with the most up-to-date congestion data. Additionally, they want to run some analysis on the data so that it can make sure trucks are traveling on time but also keeping cargo safe. Oh, and this also needs to be done in real-time! The trucking company benefits by having a system in place that ingests data from multiple sources, correlates these independent sources of data, runs analysis and intelligently reports on important events going on and even actions that the company can do to immediately improve the situation. This even includes catching imminent truck breakdowns before they occur and analyzing driving habits to predict accidents before the driver gets into one!. Sounds like an important task - this is where Storm comes in.

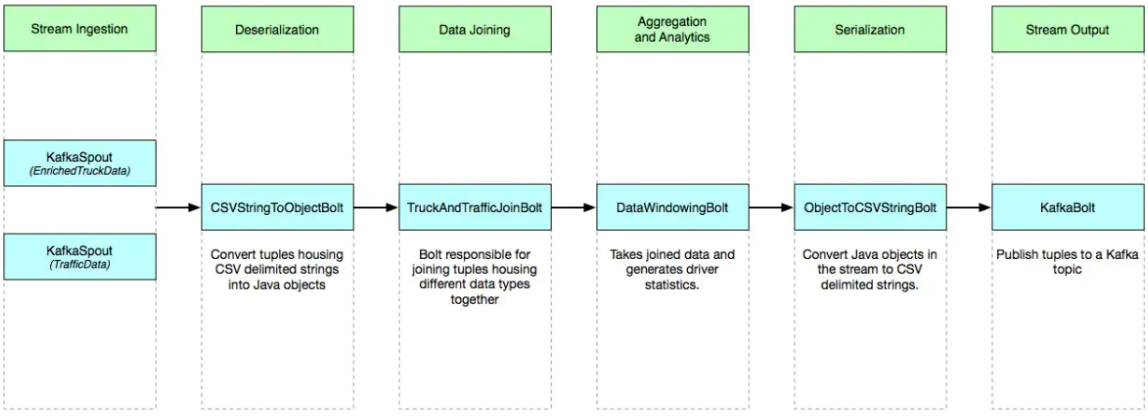

Figure 8.15: Data for Kafka Topic - Trucking data

In the first section, continuous and real-time data from sensors onboard each truck is streamed to the system in real-time by NiFi and published to Kafka topics. A

separate, second, stream carrying traffic congestion information about trucking routes is also streamed into the system and stored in a Kafka topic.

The second section represents the biggest requirement. We need something capable of: unpacking the compressed and serialized sensor data, merging independent streams together, performing aggregation and analytics, reserializing the transformed data, and sending streams back out for persistence and visualization. All this should be done in real-time and in distributed fashion while guaranteeing message processing and low-latency.For these critical tasks we use Apache Storm.

8.1.7 Storm Topology Build and Submit

Inside the KafkaToKafka.scala class and there is a companion object with our standard entry point, main, and a KafkaToKafka class with a method named buildTopology which handles the building of our Storm topology. The primary purpose of our main method is to configure and build our topology and then submit it for deployment onto our cluster. Let's take a closer look at what's inside:

// Set up configuration for the Storm Topology val stormConfig = new Config()
stormConfig.setDebug(config.getBoolean(Config.TOPOLOGY_DEBUG))
stormConfig.setMessageTimeoutSecs(config.getInt(Config.TOPOLOGY_MESSAGE))
stormConfig.setNumWorkers(config.getInt(Config.TOPOLOGY_WORKERS))
The org.apache.storm.Config class provides a convenient way to create a topology config by providing setter methods for all the configs that can be set. It also makes it easier to do things like add serializations.

Following the creation of a Config instance, our main method continues with:
// Build and submit the Storm config and topology
val topology = new KafkaToKafka(config).buildTopology()
StormSubmitter.submitTopologyWithProgressBar("KafkaToKafka",
stormConfig, topology)

Here, we invoke the buildTopology method of our class, which is responsible for building a StormTopology. With the topology that is returned, we use the StormSubmitter class to submit topologies to run on the Storm cluster.

Let's dive into the buildTopology method to see exactly how to build a topology from the ground up.
// Builder to perform the construction of the topology.
implicit val builder = new TopologyBuilder()

We start by creating an instance of TopologyBuilder, which exposes an easy-to-use Java API for putting together a topology. Next, we pull in some values from our configuration file (application.conf).

Building a Kafka Spout:

- Configure a KafkaSpout to connect to a particular set of bootstrap servers, naming the spout "trucking_data_traffic"
- Set the record translator to the one defined above. The two values that the record translator outputs we are naming "dataType" and "data", respectively.
- Configure the spout to poll the Kafka topic starting from the earliest (i.e. oldest) possible data available. In other words, ingests the entire Kafka topic.
- Set the groupID to the character "g"
- Build the KafkaSpoutConfig object following these configurations.

In order to build a KafkaSpout, we first need to decide what Kafka topic will be read from, where it exists, and exactly how we want to ingest that data in our topology. This is where the above KafkaSpoutConfig comes in handy.

Now that we have a KafkaSpoutConfig, we use it to build a KafkaSpout and place it in the topology. Remember that builder refers to the TopologyBuilder. We're creating a new KafkaSpout with a parallelism_hint of 1 (how many tasks, or instances, of the component to run on the cluster). We place the spout in the topology blueprint with the name "enrichedTruckData".

Building a Custom Bolt: We now have a way to ingest our CSV-delimited strings from Kafka topics and into our Storm topology. We now need a way to unpackage these strings into Java objects so we can more easily interact with them.

Let's go ahead and build a custom Storm Bolt for this purpose. We'll call it CSVStringToObjectBolt. But first, let's see how this new custom bolt will fit into our topology blueprint.

- Build a bolt for creating JVM objects from the ingested strings
- Our custom bolt, CSVStringToObjectBolt, is given the bolt id of "unpackaged-Data". Storm is told to assign only
- a single task for this bolt (i.e. create only 1 instance of this bolt in the cluster).

- ShuffleGrouping shuffles data flowing in from the specified spouts evenly across all instances of the newly
- created bolt (which is only 1 in this example)

We create a new CSVStringToObjectBolt bolt, and tell Storm to assign only a single task for this bolt (i.e. create only 1 instance of this bolt in the cluster). We name it "unpackagedData".

ShuffleGrouping shuffles data flowing in from the specified spouts evenly across all instances of the newly created bolt.

Let's dig in and see how we create this bolt from scratch: check out the CSVString-ToObjectBolt.java file.
class CSVStringToObjectBolt extends BaseRichBolt {
Rather than creating a Storm bolt entirely from scratch, we leverage one of Storm's base classes and simply extend BaseRichBolt. BaseRichBolt takes care of a lot of the lower-level implementation for us.

The prepare method provides the bolt with an OutputCollector that is used for emitting tuples from this bolt. Tuples can be emitted at anytime from the bolt – in the prepare, execute, or cleanup methods, or even asynchronously in another thread. This prepare implementation simply saves the OutputCollector as an instance variable to be used later on in the execute method.

The execute method receives a tuple from one of the bolt's inputs. For each tuple that this bolt processes, the execute method is called.

We start by extracting the value of the tuple stored under the name "dataType", which we know is either "EnrichedTruckData" or "TrafficData". Depending on which it is, we call the fromCSV method of the appropriate object, which returns a JVM object based on this CSV string.

Next, we use the outputCollector to emit a Tuple onto this bolt's outbound stream. Finally, we ack (acknowledge) that the bolt has processed this tuple. This is part of Storm's reliability API for guaranteeing no data loss.

The last method in this bolt is a short one:

The declareOutputFields method declares that this bolt emits 2-tuples with fields called "dataType" and "data".

Building a Tumbling Windowed Bolt:

Let's get back to our KafkaToKafka class and look at what other components we're adding downstream of the CSVStringToObjectBolt.

We now have KafkaSpouts ingesting in CSV strings from Kafka topics and a bolt that creating Java objects from these CSV strings. The next step in our process is to join these two types of Java objects into one.

- Create a tumbling windowed bolt using our custom TruckAndTrafficJoinBolt, which houses the logic for how to merge the different Tuples.
- A tumbling window with a duration means the stream of incoming Tuples are partitioned based on the time they were processed (think of a traffic light, allowing all vehicles to pass but only the ones that get there by the time the light turns red). All tuples that made it within the window are then processed all at once in the TruckAndTrafficJoinBolt.

Here, we create a tumbling windowed bolt using our custom TruckAndTrafficJoinBolt, which houses the logic for how to merge the different Tuples. This bolt processes both EnrichedTruckData and TrafficData and joins them to emit instances of EnrichedTruckAndTrafficData.

A tumbling window with a duration means the stream of incoming Tuples are partitioned based on the time they were processed. Think of a traffic light, allowing all vehicles to pass but only the ones that get there by the time the light turns red. All tuples that made it within the window are then processed all at once in the TruckAndTrafficJoinBolt.

Building a Sliding Windowed Bolt

Now that we have successfully joined data coming in from two streams, let's perform some windowed analytics on this data

- Creates a sliding windowed bolt using our custom DataWindowindBolt, which is responsible for reducing a list of recent Tuples(data) for a particular driver into a single datatype. This data is used for machine learning.
- This sliding windowed bolt with a tuple count of 10 means we always process the last 10 tuples in the specified bolt.
- The window slides over by one, dropping the oldest, every time a new tuple is processed.

Creates a sliding windowed bolt using our custom DataWindowingBolt, which is responsible for reducing a list of recent Tuples(data) for a particular driver into a single datatype. This data is used for machine learning.

This sliding windowed bolt with a tuple count as a length means we always process the last 'N' tuples in the specified bolt. The window slides over by one, dropping the oldest, each time a new tuple is processed.

The next step is to build a bolt and then place in the topology blueprint connected to the "joinedData" stream.

Building Another Custom Bolt:

Before we push our Storm-processed data back out to Kafka, we want to serialize the Java objects we've been working with into string form.

These bolts, ObjectToCSVStringBolt are inverse to our previous custom bolt, CSVStringToObjectBolt. They expect tuples with Java objects and emit a CSV string representation of them. Check out the source code if you're interested in their inner-workings.

Now, we have two streams emitting string data: "serializedJoinedData" which is the result of the two joined streams, and "serializedDriverStats", which is the result of windowed analytics we performed.

Building a Kafka Bolt: We now build KafkaBolts to push data from these streams into Kafka topics. withTopicSelector specifies the Kafka topic to drop entries into. withTupleToKafkaMapper is passed an instance of FieldNameBasedTupleToKafkaMapper, which tells the bolt which fields of a Tuple the data to pass in is stored as. withProducerProperties takes in properties to set itself up with.

Finally, we drop this bolt into the rest of the topology.

Creating the Topology: Now that we have specified the entire Storm topology by adding components into our TopologyBuilder, we create an actual topology using the builder's blueprint and return it. Now package and deploy to storm. It will create a topology called KafkaToKafka as below

Figure 8.16: Kafka Topology summary in Sandbox

For entire code and implementation steps : Visit my GitHub URL

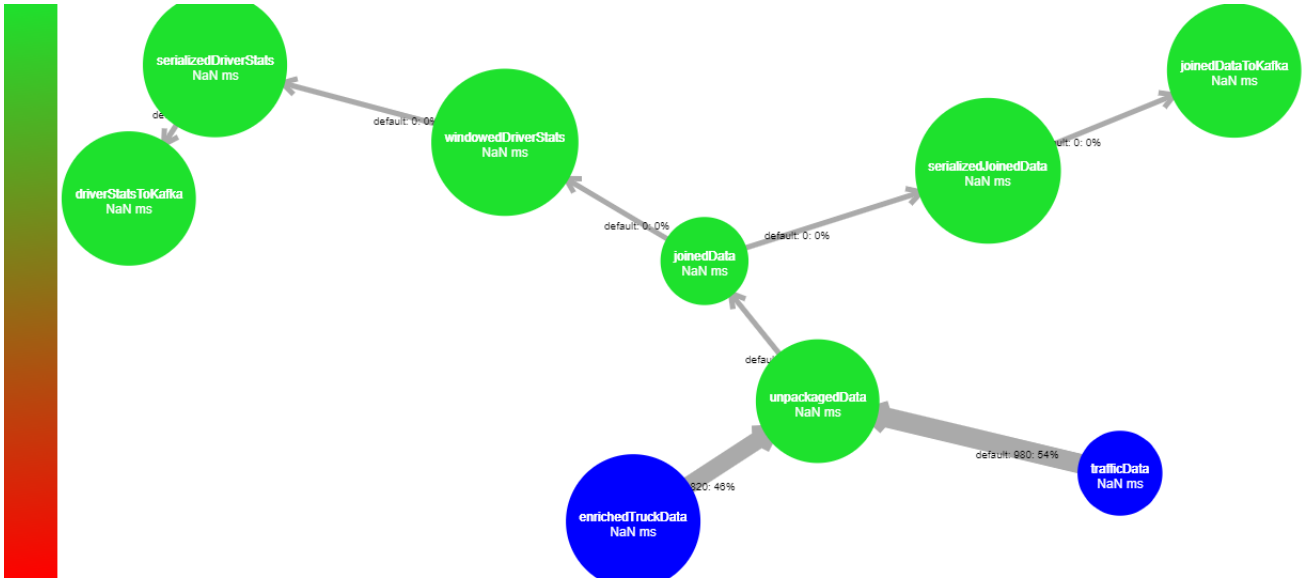

Figure 8.17: Storm topology visualization

8.2 Future scope

1. This is a prototype and can be implemented in real life scenario with the feature of self driving feature
    - Over speed detection and control the vehicle speed from remote if the driver is snoozy
2. Develop a system which can be used in any road vehicle to perform these functions
3. Full benefits of this project can only be assured after the full-fledged implementation of Cloud to run Big data / Stream processing engines after Implementing the edge devices in all road vehicles like Truck / Cars / Bus etc
4. Integrate with Map to identify Accident Prone area and alert driver
5. Integrate to Web application for analysis of results and better visualization

Chapter 9

Architect Big data Solution

As an IOT big data architect or as an IOT developer here we needed to think about what is the total throughput of our system and what is the cumulative amount of events per second being generated by the whole vehicle edge system. So we have to go about sizing our cluster, so if there are 5000 vehicles where we planning to outfit blackbox sensor sets and then, if we look at the sizing of this cluster, then there is a certain equation of how you resize these distributed HDFS nodes because they not only does HDFS data but also generates intermediate data in the process. We always want to compress data so we can make use of the space more efficient as we have converted into avro format as we are processing row by row continuous stream.
So how big will be vehicle IoT data?
As explained above, Suppose if we have installed IoT edge device in 5000 vehicles, if each sensor events to company data center is 10, then per second 50000 events should get processed in real time (DB hits, Continuously train and fine-tune the ML model). Suppose size of each event is 128bytes then for

one Year = 5000*10*128*60*60*24*365 = 200TB and for

5 years it might become 1.5PB

For sizing cluster storage requirement:

$$\frac{\left[\text{Effective Capacity}\right] \times \text{Intermediate Size} \times \text{Replication Count} \times \text{Temp Space}}{\text{Compression Ratio}}$$

Figure 9.1: Cluster storage sizing common equation

Rule of thumb : Replication Count: 3, Temp Space: x1.2, Compression Ratio: 2-4
HBase is a fast lookup store, so we don't need to size HBase for storing 5 years of

data because it only need to store for the last week or two weeks at the most, because our application will only require that much because the rest of it is historical archive. so for HBase, we want to store 15 days of data then storage needed is only 8.2 TB.

Hbase storage needed is, 5000 * 10 * 60 * 60 * 24 * 15 * 128 = 8.2 TB. We have 1.5 PB of 5 years total data volume and we have 6.4 kilobytes per second of ingest rate as,

Ingest rate = 128 Bytes X 5000 trucks X 10 events/s = 6.4 KB/s. So, How many Hbase nodes are needed for 8.2TB storage? - No. of Worker Nodes = Total Cluster Storage / Storage Per Server = 1.5 PB / 48 TB = 32.

So we can architect the cluster with 32 nodes.

- NiFi can collect @ 50 MB/s/node
- Kafka can ingest @10MB/s/node or 100,000 events/s/node
- Storm can process @ 100,000 events/s/node
- Each HBase Region Server can store 1TB

1. Hence for 6.4 KB/s ingest rate: 1 NiFi , 1 Kafka, 1 Storm nodes are sufficient.
2. We will use 2 NiFi 3 Kafka for HA.
3. Hbase nodes needed = 1.5PB/1TB = 8 nodes
4. Co-locate Kafka and Storm.
5. Co-locate DataNode and Hbase.

For sizing cluster storage requirement:

DataNodes & Hbase	Nifi	Kafka & Storm Ingest Nodes	Client Nodes	Master Nodes	Total
32	2	3	2	5	44

Figure 9.2: Node count

Nodes connectivity architecture

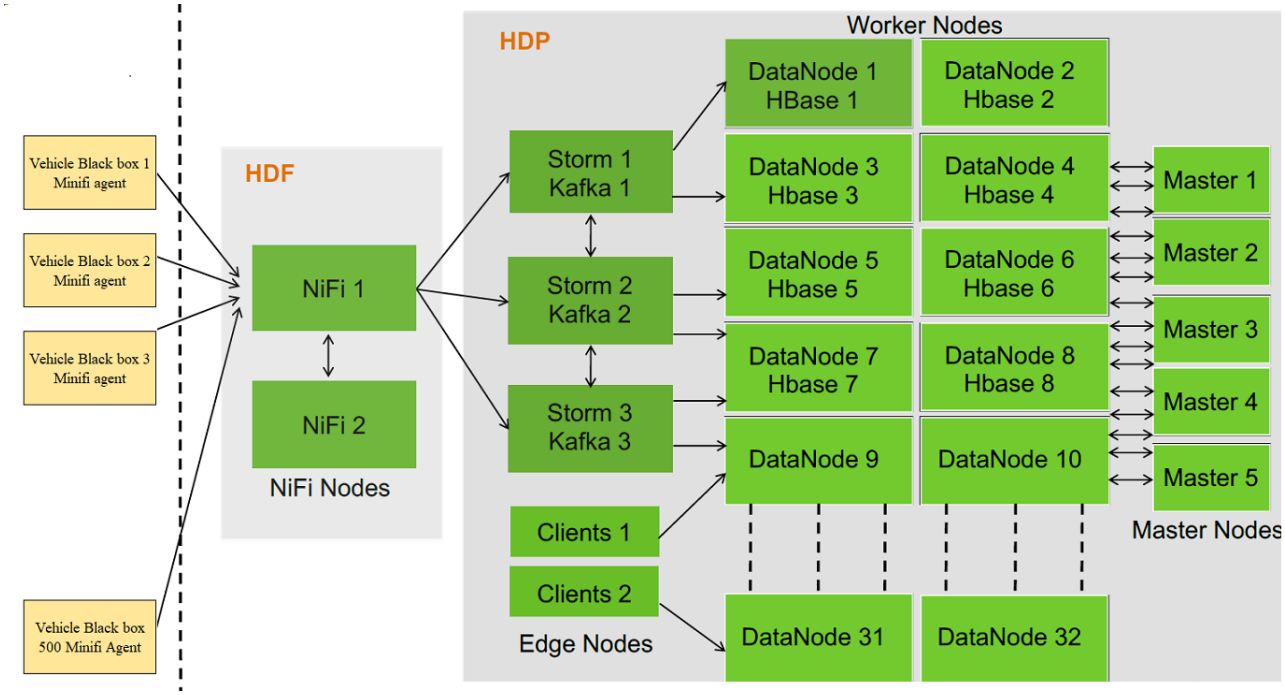

Figure 9.3: Nodes connectivity architecture

Chapter 10

Conclusion

Life is precious and we should be doing whatever possible to make roads safer. WHO has already predicted 2 million casualties by the year 2021. India is especially at risk as the country is being reformed by building more bridges, roads and better transportation networks and new areas emerges to develop. Gradually with assistance from both the vehicle owners and the Government assisting in deployment of such devices in vehicles, we can reduce the impact from the ever so concerning issue of road accidents. In addition it will help save lives, aid in better data collection and build an infrastructure solution using Emergency Crash Reporting Software to support the rescue services of the country vary the sizable detection, sensitivity, capacity and resolution of the captured image. The sensor data hereby captured and stored in memory for future references and analysis also sending into cloud server for Big data and stream analytics operations with Open source tools, for accessing the data from remote anytime anywhere.

Bibliography

[1] VANET Based Vehicle Tracking Module for Safe and Efficient Road Transportation System by P.A.Sumayya a, *, P.S.Shefeena, International Conference on Information and Communication Technologies (ICICT 2014)

[2] Lind, R. et al., The Network Vehicle-a glimpse into the future of mobile multimedia,IEEE Aerospace and Electronic Systems Magazine, Sept. (1999). , 14(9), 27-32.

[3] Agarwal, A, Little, T. D. C. Opportunistic Networking in Delay Tolerant Vehicular Ad Hoc Networks, In Advances in Vehicular Ad-Hoc Networks: Developments and Challenges (Ed. M. Watfa), (2010). , 282-300.

[4] Vegni, A. M, Cusani, R. Connectivity Support in Heterogeneous Wireless Networks,in Recent Advances in Wireless Communications and Networks, Edited by Jia-Chin Lin, 978-9-53307-274-6INTECH Pub, August (2011).

[5] Vegni, A. M, Inzerilli, T, Cusani, R. Seamless Connectivity Techniques in VehicularAd-hoc Networks, in Advances in Vehicular Networking Technologies, Edited by Miguel Almeida, 978-9-53307-241-8INTECH Pub, April (2011).

[6] Inzerilli, T, Vegni, A. M, Neri, A, Cusani, R, Cross-layer, A. Location-Based Approach for Mobile-Controlled Connectivity, Int. Journal of Digital Multimedia Broadcasting, Hindawi Publ. Corp., Article ID 597105, 13 pages, doi:(2010). , 2010

[7] Soldo, F, Casetti, C, Chiasserini, C. F, Chaparro, P. Streaming Media Distribution in VANETs, In Proc. of the IEEE GLOBECOM, November-December (2008). , 1-6.

[8] Tseng, Y, Ni, C, Chen, S. -Y, Sheu, Y. -S. J.-P., The Broadcast Storm Problem in a Mobile ad hoc Network. Wireless Networks, 8: (2002). , 153-167. 18 Vehicular Technologies - Deployment and Applications

[9] Kaul, S, Ramachandran, K, Shankar, P, Oh, S, Gruteser, M, Seskar, I, Nadeem, T. Effect of Antenna Placement and Diversity on Vehicular Network Communica-

tions,In 4th Annual IEEE Communications Society Conference on Sensor, Mesh and AdHoc Communications and Networks, 2007. SECON'07, (2007). , 112-121.

[10] Sengupta, R, Rezaei, S, Shladover, S. E, Cody, D, Dickey, S, Krishnan, H. Crosslayer- based Adaptive Vertical Handoff with Predictive RSS in Heterogeneous Wireless Networks, Vehicular Technology, IEEE Transactions on, 7(6): (2008). , 3679-3692.

[11] Moustafa, H, Zhang, Y. Vehicular Networks: Techniques, Standards, and Applications,Auerbach Publications, Taylor and Francis Group, 450 pages, Ch. 2, (2009).

[12] Varaiya, P. Smart cars on smart roads: problems of control," IEEE Transactions on Automatic Control, Feb (1993). , 38(2), 195-207.

[13] Leen, G, Heffernan, D. Expanding Automotive Electronic Systems, Computer, Jan. (2002). , 35(1), 88-93.

[14] Vegni, A. M, Little, T. D. C. Hybrid Vehicular Communications Based on V2V-V2I Protocol Switching, Int. Journal of Vehicle Information and Communication Systems (IJVICS), Nos. 3/4, , 2, 213-231.

[15] Di Felice MDoost-Mohammady R., Chowdhury K.R., Bononi L. Smart Radios for Smart Vehicles: Cognitive Vehicular Networks, IEEE Vehicular Technology Magazine,June (2012). , 7(2), 26-33.

[16] Drawil, N. Improving the VANET Vehicles' Localization Accuracy Using GPS Receiver in Multipath Environments, Master of Applied Science, available online http://libdspace.uwaterloo.ca/bitstream

[17] Sivaraj, R, Gopalakrishna, A. K, Chandra, M. G, Balamuralidhar, P. QoS-enabled group communication in integrated VANET-LTE heterogeneous wireless networks, Wireless and Mobile Computing, Networking and Communications (WiMob), 2011 IEEE 7th International Conference on, vol., no., Oct. (2011). , 17-24.

[18] Remy, G, Senouci, S, Jan, F, Gourhant, Y, Lte, V. X: LTE for a Centralized VANET Organization, Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE, vol., no., Dec. (2011). , 1-6.

[19] Nadeem, T, Dashtinezhad, S, Liao, C, Iftode, L. TrafficView: traffic data dissemination using car-to-car communication, ACM SIGMOBILE Mobile Comput. Commun. Rev. 8 (3) ((2004). , 6-19.

Appendix A

EMBEDDED C PROGRAM CODE FOR PIC

```
#include<pic.h>
#include<ADC.h>
#include<stdlib.h>
#include<lcd.h>
#include<uart.h>
#define _XTAL_FREQ 16000000
__CONFIG(0X3F7A);
void main()
{
char c[5],c1[5];
int temp,n;
T0SE =1;
uart_init(9600);
lcd_init();
lcd_clear();
adc_init();
while(1)
{
lcd_goto(1,1);
lcd_puts("Temp= ");
lcd_goto(2,1);
lcd_puts("Speed= ");
lcd_goto(2,8);
itoa(c1,n,10);
lcd_puts(c1);
lcd_goto(2,12);
lcd_puts("RPM");
temp =(adc_read(0)/2)-1;
lcd_goto(1,7);
itoa(c,temp,10);
lcd_puts(c);
uart_write('#');
uart_puts(c);
uart_write('$');
```

```
uart_puts(c1);
TMR0 =0X00;
__delay_ms(2000);
n =TMR0;
n =n*30;
lcd_clear();
}
}
```

Appendix B

PYTHON PROGRAM CODE FOR RASPBERRY PI3

```
import RPi.GPIO as GPIO #Enables internal pull_up/pull_down
import time
import serial
import thread
import urllib2
import os
#------- B.2 ------------------------#
from twilio import TwilioRestException
from twilio.rest import TwilioRestClient
#------- B.2 ------------------------#
#------- B.4 ------------------------------------#
import email
import email.utils
import smtplib
from email.MIMEMultipart import MIMEMultipart
from email.MIMEText import MIMEText
from email.MIMEBase import MIMEBase
from email import encoders
#------- B.4 ------------------------------------#
#------- B.1 ------------------------------------------------------#
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)
GPIO.setup(38,1,pull_up_down =GPIO.PUD_UP) #For alcohol
GPIO.setup(36,1,pull_up_down =GPIO.PUD_UP) #water interface
GPIO.setup(31,1,pull_up_down =GPIO.PUD_UP) #Horn
GPIO.setup(33,1,pull_up_down =GPIO.PUD_UP) #Brake
GPIO.setup(35,1,pull_up_down =GPIO.PUD_UP)#right indicator
GPIO.setup(37,1,pull_up_down =GPIO.PUD_UP)#left indicator
GPIO.setup(32,1,pull_up_down =GPIO.PUD_UP)#piezo sensor
GPIO.setup(8,0) #right indicator output
GPIO.setup(10,0) # left indicator output
GPIO.setup(40,0) # buzzer output
#------- B.1 ------------------------------------------------------#
```

```
picser =serial.Serial('/dev/ttyUSB1',baudrate =9600,timeout =1) #pic

#------- B.6 ----------#
gpsser =serial.Serial('/dev/ttyUSB0',baudrate =9600,timeout =1) #GPS
#------- B.6 ----------#

#---------- B.16 ----------#
data ='#0$0' # variables initialized
lat ='0'
lon ='0'
temp ='0'
rpm ='0'
tim ='0'
RI ='0'
LI ='0'
al ='0'
b ='0'
h ='0'
m ='0'
TS ='0'
ST ='0'
file =open("/home/pi/Desktop/bb.txt","w") # to write in SD card
#---------- B.16 ----------#

#---------- B.11 ----------#
def txn(idn,val): # upload to server. Replace with Cloud App end point
try:
request = urllib2.Request("http://nodeiot.online/site/up.php?id=%d&value=%s"%(idn,val),
headers ={"Accept" : "text/html",
"User-Agent":"Mozilla/5.0 (Windows NT 10.0; WOW64; rv:40.0)
Gecko/20100101 Firefox/40.0",
"Accept": "text/html,application/xhtml+xml,application
/xml;q=0.9,*/*;q=0.8",
"Referer": "http://thewebsite.com",
"Connection": "keep-alive"})
contents = urllib2.urlopen(request).read()
except:
```

```
print 'Sending Failed'
#———————————— B.11 ————————————————————————-#


#————————- B.5 ——————————————————————————-#
def mail(to,sub):# function for mailing -mail( TO mail-id, subject)
try: # try and except method adopted for mailing
fromaddr = "2018ab04032wilpbits@gmail.com"
toaddr = to
msg = MIMEMultipart()
msg['From'] = fromaddr
msg['To'] = toaddr
msg['Subject'] = sub
body = "HELP\n" +'Latitude=' +lat +', Longitude=' +lon +'\nDate
& Time=' +tim
msg.attach(MIMEText(body, 'plain'))
filename = "Image"
attachment = open("/home/pi/Desktop/image.jpg", "rb")
part = MIMEBase('application', 'octet-stream')
part.set_payload((attachment).read())
encoders.encode_base64(part)
part.add_header('Content-Disposition', "attachment;
filename= %s" % filename)
msg.attach(part)
server = smtplib.SMTP('smtp.gmail.com', 587)
server.starttls()
server.login(fromaddr, "goodjobdone123")
text = msg.as_string()
server.sendmail(fromaddr, toaddr, text)
server.quit()
print 'Mail Sent'
except:
print 'Mailing Failed'
#————————- B.5 ——————————————————————————-#

#———— B.3 ———————————————————————————#
def mob(msg):
```

```
account_sid = "ACb31e6a98d56b4b98a6cf0264a7802ae7" #Your Account
SID from www.twilio.com/console
auth_token = "43a2acfde3aa44c42853962b34643cf8" #Your Auth Token
from www.twilio.com/console
client = TwilioRestClient(account_sid, auth_token)
try:
message = client.messages.create(body =msg,
to ="+919947099911", #Your phone number
from_="+18329814729") #Your Twilio number
print 'sent'
except:
print 'msg failed'
#------- B.3 ----------#

#---------- B.7 ----------#
def gps(threadName, delay):
global lat
global lon
while(1):
msg =gpsser.readline() #To receive datas
if 'GPRMC' in msg: # Lat and long datas will be after "GPRMC"
lat =str(msg[20:28]) # string position from 22 to 28 will be latitude
lon =str(msg[32:41]) # longitude
#print 'lat='+lat, #print 'lon='+lon
time.sleep(delay)
#---------- B.7 ----------#

#---------- B.10 ----------#
def pic(threadName, delay):
global data
global rpm
global temp
while(1):
try:
data =picser.readline()
data =str(data)
l =len(data)
```

```
temp =str(data[data.index('#')+1:data.index('$')]) #RPM data from PIC
print 'rpm=' +rpm
print 'temp=' +temp
except:
print 'Waiting'
time.sleep(delay)
#———————————— B.10 ————————————————————————#

#———————————— B.14 ————————————————————————-#
def status(threadName, delay):
global tim
global RI
global LI
global al
global h
global b
global m
global TS
global ST
while(1):
tim =str(time.ctime()) #system date and time
if(GPIO.input(35)==  =0):
GPIO.output(8,1)
print 'Right Indicator ON'
RI ='\nRight Indicator ON'
RIS ='RION,'
else:
GPIO.output(8,0)
RI =''
RIS ='RIOFF,'
if(GPIO.input(37)==  =0):
GPIO.output(10,1)
print 'Left Indicator ON'
LI ='\nLeft Indicator ON'
LIS ='LION,'
else:
GPIO.output(10,0)
```

```
LI =''
LIS ='LIOFF,'
if(GPIO.input(38)==  =0):
print 'Alcohol Detected'
al ='\nAlcohol Detected'
als ='AD'
else:
als ='AnD'
al =''
if(GPIO.input(31)==  =0):
print 'Horn Pressed'
h ='\nHorn Pressed'
HS ='HP,'
GPIO.output(40,1)
else:
GPIO.output(40,0)
h =''
HS ='HnP,'
if(GPIO.input(33)==  =0):
print 'Brake Pressed'
b ='\nBrake Pressed'
BS ='BP,'
else:
b =''
BS ='BnP,'
#---------- B.14 ----------#

#---------- B.15 ----------#
file =open("/home/pi/Desktop/bb.txt","w")
file.write(" ") # write with blank data
file.close()
m = \n HERE IS THE BLACKBOX DATA SAVED DURING
ACCIDENT\n' +tim +'\nLatitude=' +lat +'\nLongitude=' +lon
+h +b +RI +LI +al +'\nSpeed=' +rpm +'\nTemperature=' +temp
file =open("/home/pi/Desktop/bb.txt","w")
file.write(m)
file.close()
```

```
TS =tim[0:3]+tim[4:7]+tim[11:19]+',2021,' #collected date and time
ST =RIS +LIS +HS +BS +als +',Temp=' +temp +',RPM=' +rpm #status
#———————————— B.15 ————————————————————————#

#———————————— B.9 ————————————————————————-#
if(GPIO.input(36)=  =0):
print 'accident(Plunged into water)'
acc ='Accident\n' +'Plunged into water\n' +'Latitude='
+lat +'\nLongitude=' +lon
mob(acc) #calling function mob(msg) for sending message to mobile
os.system("sudo fswebcam -r 320x240 /home/pi/Desktop/image.jpg")
mail("2018ab04032wilpbits@gmail.com","Accident Detected") #calls mail
time.sleep(delay)
#———————————— B.9 ————————————————————————-#
def upload(threadName, delay): #continuous upload to server
while(1):
yy =TS +ST
txn(5622692,yy)
time.sleep(delay)

#———————————— B.8 ————————————————————————#
def accident(channel):
if(GPIO.input(32)=  =0):
print 'accident'
os.system("sudo fswebcam -r 320x240 /home/pi/Desktop/image.jpg")
acc ='Accident\n' +'Latitude=' +lat +'\nLongitude=' +lon
mob(acc)
mail("2018ab04032wilpbits@gmail.com","Accident Detected")
GPIO.output(40,0)
time.sleep(1)
#———————————— B.8 ————————————————————————#

#———————————— B.13 ———————————————————————#
GPIO.add_event_detect(32,GPIO.FALLING,callback =accident,
bouncetime =300) # interrupt for sensing piezo, will call function accident
#———————————— B.13 ———————————————————————#
mob('Accident detection Edge Hardware is ON')
```

```
#———————————— B.12 ————————————————————————————-#
try:
thread.start_new_thread(gps, ("Thread-1", 1, ) )
thread.start_new_thread(upload, ("Thread-1", 1, ) )
thread.start_new_thread(pic, ("Thread-1", 1, ) )
thread.start_new_thread(status, ("Thread-1", .5, ) )
except:
print "Error: unable to start thread"
while 1:
pass
#———————————— B.12 ————————————————————————————-#
```

Appendix C

Circuit Diagram of Edge Hardware System