Data Intensive Computing – Lab3

911 Emergency Helpline Number Data Analysis using Apache Spark

Sonali Yeshwant Naidu (UB ID #50299652)
Pratibhaa Reddy Kandimalla (UB ID #50299669)

Abstract:

In many countries, the public telephone network has universal emergency telephone number that allows a caller to contact local emergency services for assistance. In this Use Case 911 Emergency Helpline Number Data Analysis, we will perform analysis on the data provided the callers who had called the emergency helpline number in North America. We analyse the real-time emergency calls recorded and propose an algorithm to predict the highly affected areas/unsafe areas. The output of this predictive analysis will be visualized using visualizing tools. Here the data set for the 911 data includes zip, city, state, latitude, longitude, time zone and description.

Problem Statement:

Accident is an event, occurring suddenly, unexpectedly and inadvertently under unforeseen circumstances. All living humans continually bear a certain degree of risk of injury/accident. Developing countries bear a significant portion of the worldwide burden, accounting from accidents such as Traffic Vehicle accident, Fire, Emergency Medical Services.
Statisticians and data scientists specialize in determining whether a particular activity or product poses an unreasonable risk. Such risks are predictable and preventable, but good data are important to understand the ways in which safety interventions and technology can be successfully transferred from developed countries where they have proven effective. Risk estimation involves establishing a reference period and then collecting information about the number of accidents, injuries (or other adverse events) suffered, and the amount of exposure during this period. In Use Case 911 Emergency Helpline Number Data Analysis, we will be performing analysis on the data provided by the callers who had called the emergency helpline number in US.
In our use case the real time 9-1-1 data can be attributed primarily to the recognition of characteristics of modern society, i.e., increased incidences of crimes, accidents, and medical emergencies, inadequacy of existing emergency reporting methods, and the continued growth and mobility of the population.

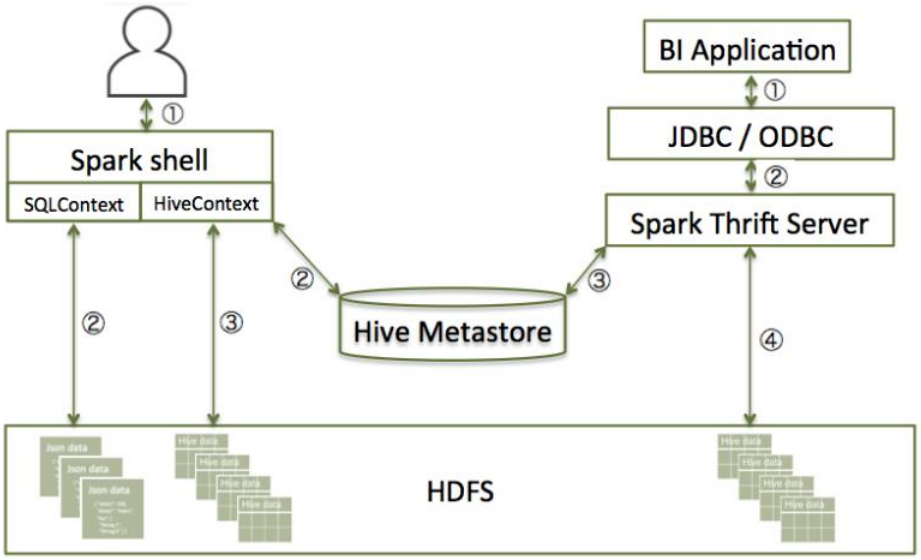
This includes collecting, collating, cleaning, aggregating, adding and removing parts of the data. It also includes deciding how to analyse the data. We have 2 problem statement in this use case

a. What kind of problems are prevalent and in which State
b. What kind of problems are prevalent and in which area of the State

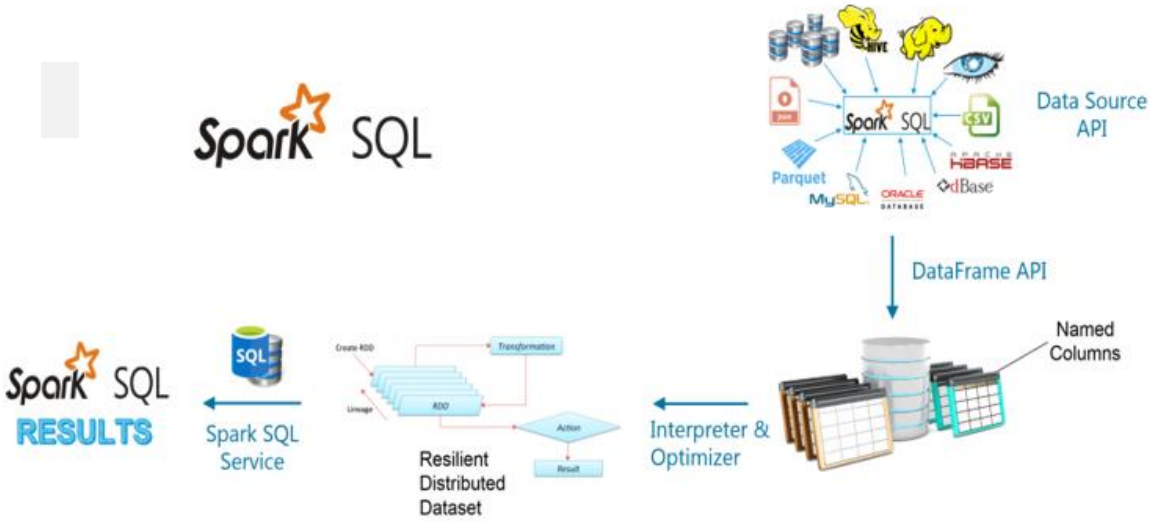
This data analysed to understand the highly affected/risk prone areas can further be used to find the main reason for the accidents and work towards it.

Solution(model) and Design Document:

Architectural Block diagram of Spark

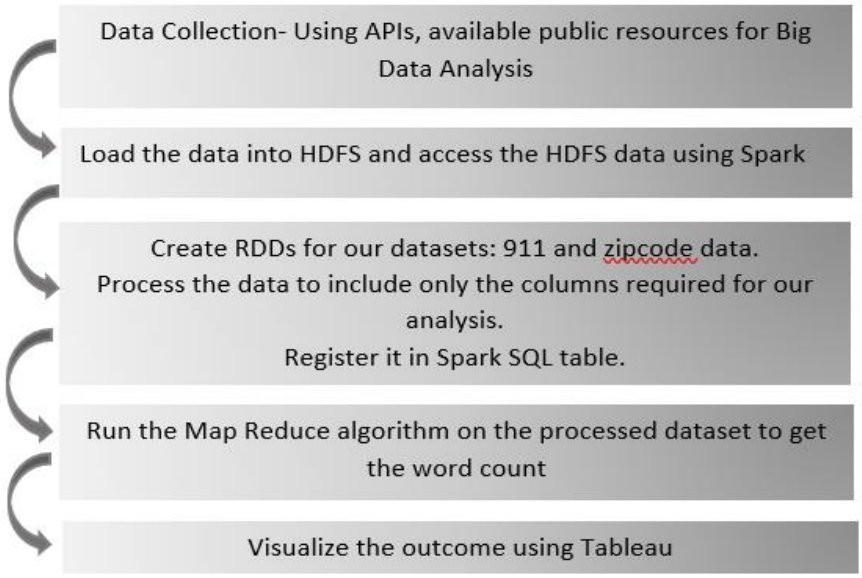


Block Diagram of Spark



Flow diagram representing SparkSQL process

Work Flow of 911 Emergency Helpline Use Case:



Solution:

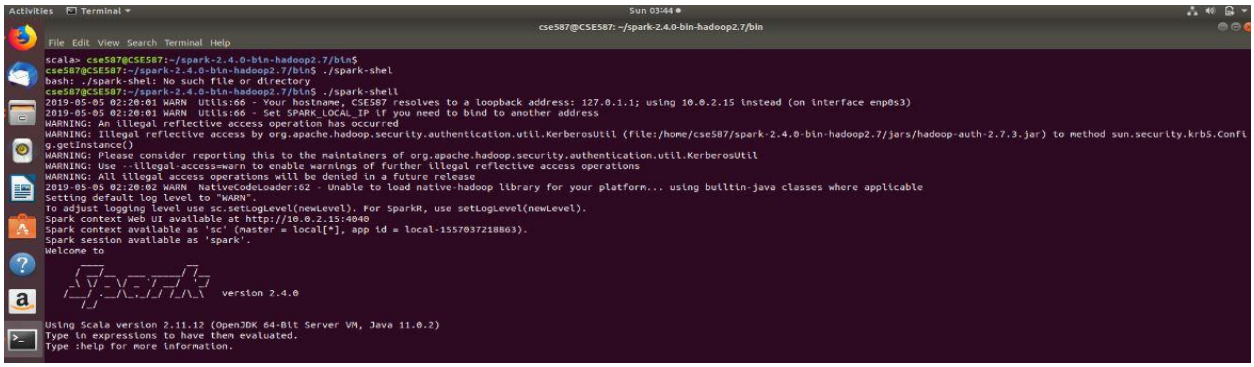
In this Use Case 911 Emergency Helpline Number Data Analysis, we perform analysis on the data provided the callers who had called the emergency helpline number in US. We collected 911 emergency data from a publicly available resource for our analysis.

We have two data sets here; one is the data that the callers have given when they called the emergency helpline number. The other data set contains the details about the zip code.

The data set description for the above two data sets are as follows:
Emergency Details File: Latitude, Longitude, Description, Zip Code, Title, Timestamp, Township, Address.
Zip Code File: Zip Code, city, state, latitude, longitude, time zone. district

The above data can alternatively be collected from different resources such as the NY Times using an API. However, for our analysis we have used the data that is available publicly for the users for performing the data analysis.

Once the data is collected, we will copy it into the HDFS and start the spark shell using the below command.
./spark-shell – This starts the spark shell.



We will start with parsing the data into a dataframe.

```
scala> val data = sc.textFile("/home/cse587/exampleSpark/911.csv")
data: org.apache.spark.rdd.RDD[String] = /home/cse587/exampleSpark/911.csv MapPartitionsRDD[1] at textFile at <console>:24

scala> val header = sc.textFile("/home/cse587/exampleSpark/header.csv")
header: org.apache.spark.rdd.RDD[String] = /home/cse587/exampleSpark/header.csv MapPartitionsRDD[3] at textFile at <console>:24

scala> val data1 = sc.textFile("/home/cse587/exampleSpark/data1.csv")
data1: org.apache.spark.rdd.RDD[String] = /home/cse587/exampleSpark/data1.csv MapPartitionsRDD[5] at textFile at <console>:24

scala> case class emergency(lat:String,lng:String,desc:String,zip:String,title:String,timeStamp:String,twp:String,addr:String,e:String)
defined class emergency

scala> val emergency_data = data1.map(x=>x.split(",")).filter(x => x.length>=9).map(x => emergency(x(0),x(1),x(2),x(3),x(4).substring(0 , x(4).indexOf(":")
     | ),x(5),x(6),x(7),x(8))).toDF
emergency_data: org.apache.spark.sql.DataFrame = [lat: string, lng: string ... 7 more fields]

scala> emergency_data.registerTempTable("emergency_911")
warning: there was one deprecation warning; re-run with -deprecation for details
```

Steps to register the Emergency dataset into a table using Spark SQL:

1. We start with creating an RDD file and store it in immutable variable data.
2. Similarly we create RDDs called header containing only the header and val data1 containing all rows except the header.
3. We create a class called emergency to define the attributes in the dataset.
4. Now we parse the dataset into the case class, that was declared earlier and in the column 5 we trim out the string up to the character ":" as we only need the cause of calling the emergency number.
5. Save this data into a table and name it as emergency_911 as shown in the image below:

```
scala> :power
Power mode enabled. :phase is at typer.
import scala.tools.nsc._, intp.global._, definitions._
Try :help or completions for vals._ and power._

scala> settings.deprecation.value = true
settings.deprecation.value: Boolean = true

scala> emergency_data.registerTempTable("emergency_911")
<console>:51: warning: method registerTempTable in class Dataset is deprecated: Use createOrReplaceTempView(viewName) instead.
       emergency_data.registerTempTable("emergency_911")
                      ^

scala> emergency_data.createOrReplaceTempView("emergency_911")
```

Steps to register the Zip code dataset into a table using Spark SQL:

1. We start with creating an RDD file and store it in immutable variable data2.
2. Similarly we create RDDs called header1 containing only the header and val data3 containing all rows except the header.
3. We create a class called zipcode to define the attributes in the zipcode dataset.
4. Now we parse the dataset into the case class, that was declared earlier and we trim out the double quotes in the string.
5. Save this data into a table and name it as emergency_911 as shown in the image below: Zipcodes.createOrReplaceTempView("zipcode_table")

Look at the below commands for executing these steps:

```
scala> val data2 = sc.textFile("/home/cse587/exampleSpark/zipcode.csv")
data2: org.apache.spark.rdd.RDD[String] = /home/cse587/exampleSpark/zipcode.csv MapPartitionsRDD[12] at textFile at <console>:49

scala> val header1 = sc.textFile("/home/cse587/exampleSpark/zheader.csv")
header1: org.apache.spark.rdd.RDD[String] = /home/cse587/exampleSpark/zheader.csv MapPartitionsRDD[14] at textFile at <console>:49

scala> val data3 = sc.textFile("/home/cse587/exampleSpark/zwithdata.csv")
data3: org.apache.spark.rdd.RDD[String] = /home/cse587/exampleSpark/zwithdata.csv MapPartitionsRDD[16] at textFile at <console>:49

scala> case class zipcode(zip:String,city:String,state:String,latitude:String,longitude:String,timezone:String,dst:String)
defined class zipcode

scala> val zipcodes = data3.map(x => x.split(",")).map(x=> zipcode(x(0).replace("\"", "")
     | ,x(1).replace("\"", ""),x(2).replace("\"", ""),x(3),x(4),x(5),x(6))).toDF
zipcodes: org.apache.spark.sql.DataFrame = [zip: string, city: string ... 5 more fields]
```

Now, join both the datasets using the common columns in the datasets for our analysis. That is, join the datasets by using Zip code as a key to find the state and city from which the person has called. The command for this is as follows

val build1 = sqlContext.sql("Select e.title, z.city, z.state from emergency_911 e join zipcode_table z on e.zip=z.zip")

Data Pipeline:

In this Use Case study, we used Apache Spark for analysing the data as the dataset is huge. We chose Spark as it can process the data faster than Hadoop since it does the processing in the main memory of the worker nodes and prevents the unnecessary I/O operations with the disks. Apache Spark is a fast, in-memory data processing engine with elegant and expressive development APIs to allow data workers to efficiently execute streaming, machine learning or SQL workloads that require fast iterative access to datasets.

Language used:

Spark can be written in Scala, Python and Java. But we chose Scala in this use case because Scala is a functional paradigm, it can collaborate within MapReduce Big Data Model for counting the words in the dataset.

Data source:

In this use case, we extract the data recorded from the 911 emergency service in US. For our analysis we have used the data from a resource that is available publicly for the users for performing the data analysis [1]. We analyse the real-time emergency calls recorded and propose an algorithm to predict the highly affected areas/unsafe areas.

We have two datasets here: 911 and zipcode

The data set description for the 911 data is as follows:
Latitude, Longitude, Description, Zip Code, Title, Timestamp, Township, Address.
The data set description for the zip code file is as follows:
Zip Code, city, state, latitude, longitude, time zone, District

The above data can alternatively be collected from different resources such as the NY Times using an API. Here is the code snippet for data collection from New York Times.

```
#import sys
import sys
!{sys.executable} -m pip install nytimesarticle
!{sys.executable} -m pip install requests
```

```
import os

os.getcwd( )

os.chdir('C:/Users/Sonali/Desktop/')

os.getcwd( )
```

```
from nytimesarticle import articleAPI
api = articleAPI('nXUvx96o1kOa0U9a5YQY6J7aGgp2cb75')
```

```
#subTopics = ['NBA', 'NCAA', 'golf','pga tour', 'cricket', 'ipl' , 'tennis','ATP','WTA','ufc']
#subTopics = ['basketball','golf','cricket','football','badminton','tennis']
subTopics = ['traffic','accident','emergency','911']
```

```
articles = []
for subtopic in subTopics:
    print(subtopic)
    for i in range(1,101):
        articles.append(api.search( q = subtopic,page = i, begin_date = 20190401))
```

```
with open("url.txt", "a") as myfile:
    for x in articles:
        if 'response' in x.keys():
            for j in x['response']['docs']:
                myfile.write('\n'+j['web_url'])
```

```
import json
from bs4 import BeautifulSoup
import requests
```

```
records=[]
```

```
with open('url.txt') as file:
    for line in file:
        content = ""
        r = requests.get(line)
        soup = BeautifulSoup(r.text, 'html.parser')
        results = soup.findAll('p',attrs={'class':'css-1ygdjhk evys1bk0'})
        if len(results)<15:
            ulimit=len(results)
        else:
            ulimit=15
        for i in range(0,ulimit):
            content+=results[i].text
             if 'traffic' in content or 'emergency' in content or '911' in content or 'accident' in content:
            topic='emergency'
        records.append((topic,content))
```

```
with open("nydata.txt", "w", encoding="utf-8") as myfile:
       for i in range(len(records)):
            myfile.write('\n'+records[i][1])
```

```
import pandas as pd
df = pd.DataFrame(records, columns=['topic','content'])
```

```
df.to_csv('output.csv',index = False)
```

```
ascii_list = []
```

```
with open('nydata.txt',encoding="utf-8") as file:
    for line in file:
        ascii_list.append(ascii(line))
```

```
with open("nydata_enc.txt", "w") as myfile:
       for i in ascii_list:
            myfile.write('\n'+i)
```

Expected Outcome:

We will run the below commands to find the prevalent problem based on the State. Also, we can find the prevalent problem based on City.

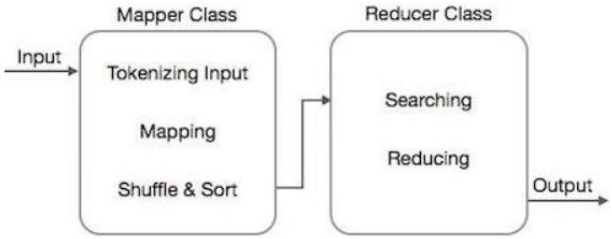
```
val ps = build1.map(x => (x(0)+" -->"+x(2).toString))
val ps1 = ps.map(x=> (x,1)).reduceByKey(_+_).map(item => item.swap).sortByKey(false).foreach(println)
```

Algorithm:

We have used the MapReduce Algorithm to find the state/area with the highest number of accidents/crimes recorded. MapReduce implements various mathematical algorithms to divide a task into small parts and assign them to multiple systems. In technical terms, MapReduce algorithm helps in sending the Map & Reduce tasks to appropriate servers in a cluster.

These mathematical algorithms may include the following −

- Sorting
- Searching
- Indexing



The output of the Map Function is a set of key and value pairs as <Key, Value>.
Reducer combines all these values together and provide single output value for the specific key.
The MapReduce algorithm on our dataset returns the output with type of problem and its count state-wise. (count, Problem ---> State)
Below is the sample output from the run.

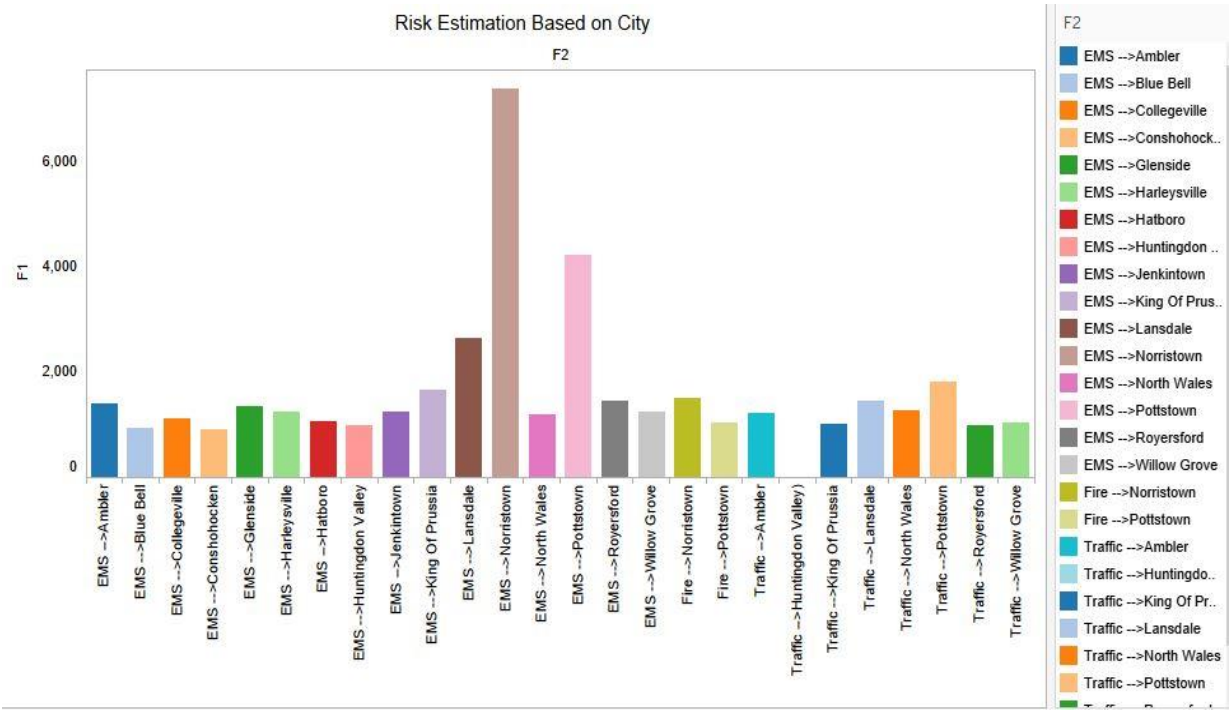
```
(13012,Fire -->PA)
(44326,EMS -->PA)
(29297,Traffic -->PA)
(1,Traffic -->AL)
(1,EMS -->TX)
```

The same algorithm can be used to find the type of problem and its count city wise.
(count, Problem ---> City)
Below is the sample output from the run.

```
(7409,EMS -->Norristown)
(4237,EMS -->Pottstown)
(2665,EMS -->Lansdale)
(1822,Traffic -->Pottstown)
(1677,EMS -->King Of Prussia)
(1505,Fire -->Norristown)
(1466,Traffic -->Lansdale)
 (1214,Traffic -->Ambler)
 (1257,EMS -->Jenkintown)
 (1257,EMS -->Willow Grove)
 (1253,EMS -->Harleysville)
 (1205,EMS -->North Wales)
 (1071,EMS -->Hatboro)
 (1128,EMS -->Collegeville)
 (1051,Traffic -->Willow Grove)
 (1042,Fire -->Pottstown)
```

Visualization:

We can visualize the output of MapReduce Algorithm using Tableau or Spark GraphX. But we chose to visualize the output using Tableau. Below is the image shown for visualizing the output:



Here, X-axis represents the problem in the corresponding city and Y-axis represents the estimation of risk for each city in North America.

Apparently, the Emergency Medical Services (EMS) calls are more received from Norristown since the risk estimation is more (approximately 7500).

Summary:

- As spark can process huge volumes of data faster than Hadoop since it does processing in main memory of the worker nodes and avoids the unnecessary operations with the disk. Hence, Apache Spark is the ideal way for analysing large volumes of data.
- Spark is a general-purpose distributed data processing engine that is suitable for a range of Use cases such as Stream Processing, Machine Learning, Interactive analytics, Data Integration. Spark supports many libraries for SQL, machine learning, graph computation, and stream processing, which can be used together in an application.
- Although this Use Case helps to predict the risk estimation based on city, it can be further extended to find the root cause of the problem and work towards it.

References:

[1] 911 Emergency Data: https://drive.google.com/file/d/0ByJLBTmJojjzNjI5ZWdYa2FYbGs/view911
zipcode: https://drive.google.com/file/d/0ByJLBTmJojjzemowRE9oMUphY2M/view

- https://acadgild.com/blog/spark-sql-use-case-911-emergency-helpline-number-data-analysis
- https://mapr.com/blog/spark-101-what-it-what-it-does-and-why-it-matters/
- https://www.cloudera.com/documentation/enterprise/5-12-x/PDF/cloudera-spark.pdf