MAULANA AZAD NATIONAL INSTITUE OF TECHNOLOGY, BHOPAL

COMPUTER SCIENCE AND ENGINEERING (CSE)

SESSION 2015-2016

MAJOR PROJECT

ROAD ACCIDENT DATA ANALYSIS USING HADOOP

(A data mining approach to analyse large datasets having main factors associated with road accident using Hadoop).

SUBMITTED BY:

Snehal Shirgure	**- 121112069**
Sandesh Kumar	**- 121112045**
Venukant Sahu	**- 121112130**
Hitesh Sharma	**- 121112155**

UNDER GUIDENCE OF:

Dr. Nilay Khare

MAULANA AZAD NATIONAL INSTITUE OF TECHNOLOGY, BHOPAL

COMPUTER SCIENCE AND ENGINEERING (CSE)

SESSION 2015-2016

CERTIFICATE

This is to certify that **Snehal Shirgure** (121112069), **Sandesh Kumar** (121112045), **Venukant Sahu** (121112130) and **Hitesh Sharma** (121112155), students of final year Bachelor of COMPUTER SCIENCE ENGINEERING have successfully completed their major project "**ROAD ACCIDENT DATA ANALYSIS USING HADOOP**" in the partial fulfilment their Bachelor degree.

Dr. Nilay Khare

(Associate Professor, CSE)

DECLARATION

We hereby declare that the work is being presented in the project report entitled “ROAD ACCIDENT DATA ANALYSIS USING HADOOP " in the partial fulfillment of Bachelor degree in Computer Science Engineering is an authentic record of our own work carried out under the able guidance of Dr. Nilay Khare. The work has been carried out at MAULANA AZAD NATIONAL INSTITUTE OF TECHNOGLOGY, BHOPAL.

The matter embodied in this report has not been submitted for the award of any degree or diploma.

Name	**Scholar No.**	**Signature**
Snehal Shirgure	121112069	
Sandesh Kumar	121112045	
Venukant Sahu	121112130	
Hitesh Sharma	121112155	

ACKNOWLEDGEMENT

With due respect, we express our deep sense of gratitude to our respected guide **Dr. Nilay Khare** for his valuable help and guidance. We are thankful for the encouragement that he has given us in completing this project successfully. His rigorous evaluation and constructive criticism was of great assistance..

We are also grateful to our respected director **Dr. Appu Kuttan K.K.** for permitting us to utilize all the necessary facilities of the college.

Needless to mention is the additional help and support extended by our respected HOD, **Dr. R.K. Pateriya**, in allowing us to use the departmental laboratories and other services.

We are also thankful to all the other faculty, staff members and laboratory attendants of our department for their kind co-operation and help.

Last but certainly not the least, we would like to express our deep appreciation towards our family members and batch mates for providing the much needed support and encouragement.

ABSTRACT

"*Road accident data analysis for large datasets using data mining algorithms in Hadoop environment.*"

Road and traffic accidents are uncertain and unpredictable incidents and their analysis requires the knowledge of the factors affecting them. Road and traffic accidents are defined by a set of variables which are mostly of discrete nature.

The aim of this project is to identify the more frequently occurring patterns of factors of road accidents and analyze them using Data mining algorithms like Apriori with the help of Hadoop framework. The data collected from various resources in structured format is first pre-processed and uploaded to HDFS. It is a challenge to gather all such relevant data, detect and analyze it to give insights on previous accidents. For this purpose, we propose to harness the power of Big Data technologies like Hadoop Map Reduce and generate the output which is in the form of patterns of frequent factors leading to road accidents.

CONTENT

1. INTRODUCTION

Project Title: Road accident data analysis using Hadoop

Project Definition:
This is a data analysis project in which we try to analyze a large dataset not capable of being analyzed by typical database or data analysis software like Excel, Weka or RapidMiner. To overcome this, we try to implement distributed processing using Hadoop.

Project Scope
Traffic engineer can compare various road segments for optimization. Government agencies can run instructional recommendation system.

Tools Required:

- **Hadoop MapReduce:** Hadoop MapReduce is a programming model for processing large data sets with a parallel, distributed algorithm on a cluster

Why Hadoop?
Processing accident data in conventional data mining softwares like Weka or RapidMiner is sure possible, but these software take a long time and are restricted in terms of memory. Hadoop lets us harness the power of parallel and distributed processing of the data to generate the result in tangible amount of time

- **Eclipse (Lunar)**: IDE for developing the Java mapper and reducer code.

Big Data Analysis – An overview

Before the age of Big Data, the analysis of data for business processes was usually limited to hundreds of rows and 4-10 columns per organization. In the 21st Century, Data is the new oil for market. Big businesses are thriving by analyzing this data and taking out useful insights from the same. This results in great profits and cost savings for a lot of organization.
Previously the data analysis using software like MS Excel or RapidMiner was enough as datasets were not that big and can bit fit in the memory easily. This led to wide development in terms of machine learning libraries to easily analyze such data. This led to wide popularity of data mining software like Weka which are still used for small cases today.

But, as they say, nothing is permanent and the advent of Information has again proved that notion true. This is the era of information where hundreds of Gigabytes of data are generated within few hours across the globe. Managing and analyzing such large scale data is a humungous task. Not only the current In-memory software are incapable for this, we need a distributed framework for processing such data. This leads to the birth of the field called Big Data. Big Data is further divided into two child fields called Big data management and Big data analytics.

Not only the analysis of big data helpful for commercial and corporate purposes, Government and public service departments can also benefit from the wide horizons and depths of the insights that are produced by mining Big data. One such example is the ongoing research field of mining accident data to improve road traffic safety. While preliminary research in this field consisted of associative rule mining on small sample of data, we are here proposing to analyze this big data set using distributed framework.

Data Mining – Overview

There is a huge amount of data available in the Information Industry. This data is of no use until it is converted into useful information. It is necessary to analyze this huge amount of data and extract useful information from it.

Extraction of information is not the only process we need to perform; data mining also involves other processes such as Data Cleaning, Data Integration, Data Transformation, Data Mining, Pattern Evaluation and Data Presentation. Once all these processes are over, we would be able to use this information in many applications such as Fraud Detection, Market Analysis, Production Control, Science Exploration, etc.

What is Data Mining?

Data Mining is defined as extracting information from huge sets of data. In other words, we can say that data mining is the procedure of mining knowledge from data. The information or knowledge extracted so can be used for any of the following applications –

- Market Analysis
- Fraud Detection
- Customer Retention
- Production Control
- Science Exploration

Data preprocessing

Data preprocessing is one of the important tasks in data mining. Data preprocessing mainly deals with removing noise, handle missing values, removing irrelevant attributes in order to make the data ready for the analysis. In this step, our aim is to preprocess the accident data in order to make it appropriate for the analysis.

Association rules

Association rule mining is a very popular data mining technique that extracts interesting and hidden relations between various attributes in a large data set. Association rule mining produces a set of rules that define the underlying patterns in the data set. The associativity of two characteristics of accident is determined by the frequency of their occurrence together in the data set. A rule $A \rightarrow B$ indicates that if A occurs then B will also occur. Given a data set D of n transactions where each transaction $T \in D$. Let $I = \{I1, I2, \ldots In\}$ is a set of items. An item set A will occur in T if and only if $A \subseteq T$. $A \rightarrow B$ is and association rule, provided that $A \subset I$, $B \subset I$ and $A \cap B = \emptyset$.

The Apriori Algorithm:

Basics The Apriori Algorithm is an influential algorithm for mining frequent itemsets.

Key Concepts :

• Frequent Itemsets: The sets of item which has minimum support (denoted by L_i for ith-Itemset).

• Apriori Property: Any subset of frequent itemset must be frequent.

• Join Operation: To find L_k , a set of candidate k-itemsets is generated by joining L_{k-1} with itself.

Apriori Algorithm in a Nutshell

• Find the frequent itemsets: the sets of items that have minimum support

– A subset of a frequent itemset must also be a frequent itemset

• i.e., if {AB} is a frequent itemset, both { A} and { B} should be a frequent itemset

– Iteratively find frequent itemsets with cardinality from 1 to k (k-itemset)

Pseudo code

• Join Step: C k is generated by joining Lk-1with itself

• Prune Step: Any (k-1)-itemset that is not frequent cannot be a subset of a frequent k-itemset

• Pseudo-code:

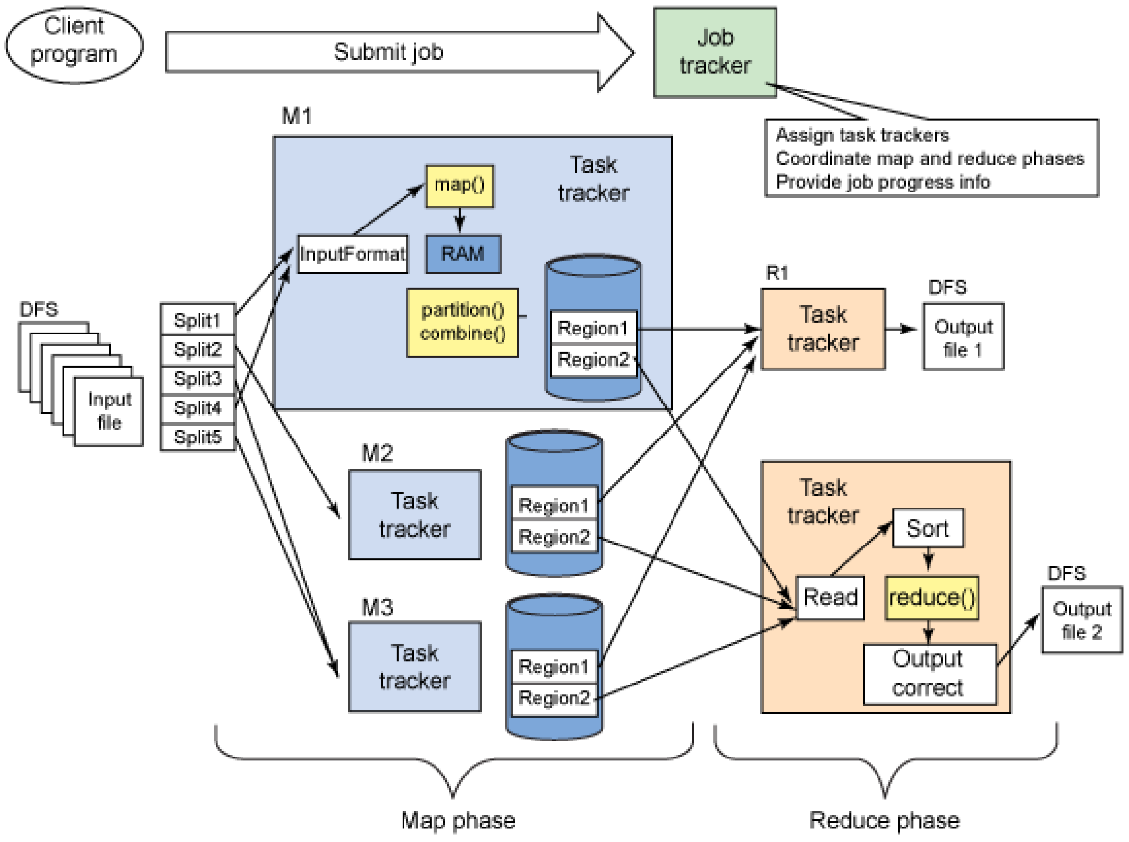
```
  $C_k$: Candidate itemset of size k
  $L_k$ : frequent itemset of size k
  $L_1$ = {frequent items};
for ( k = 1; $L_k$ != ∅; k++) do begin
    $C_{k+1}$ = candidates generated from $L_k$;
    for each transaction t in database
        do increment the count of all candidates in $C_{k+1}$
         that are contained in t
        $L_{k+1}$ = candidates in $C_{k+1}$ with min_support
end
return $\cup_k L_k$;
```

2. SYSTEM DESIGN

The Hadoop Context Flow Diagram shown below sums up the steps involved in breaking down the job into multiple tasks. The mapper and reducer phase are described.

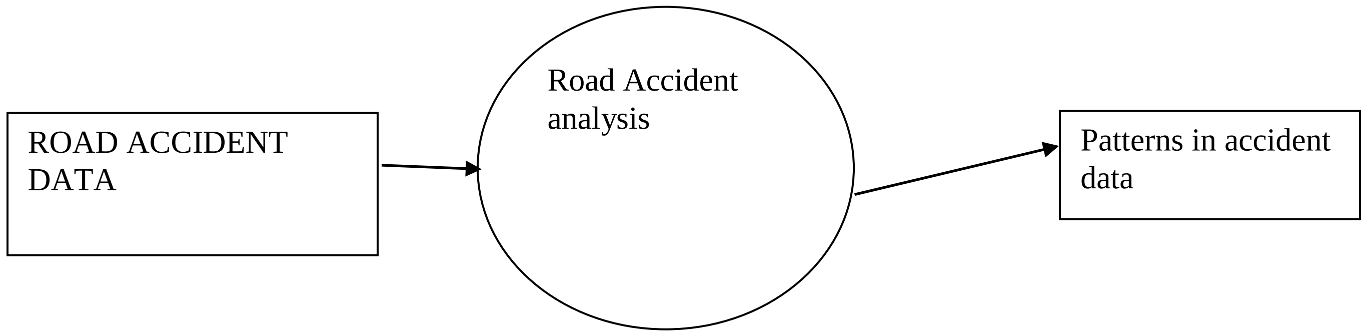
In map phase job tracker splits the input file and mapper stores these in form of key-value pair in the common region.

In reduce phase reducer shuffles and sorts according to key and provides output.



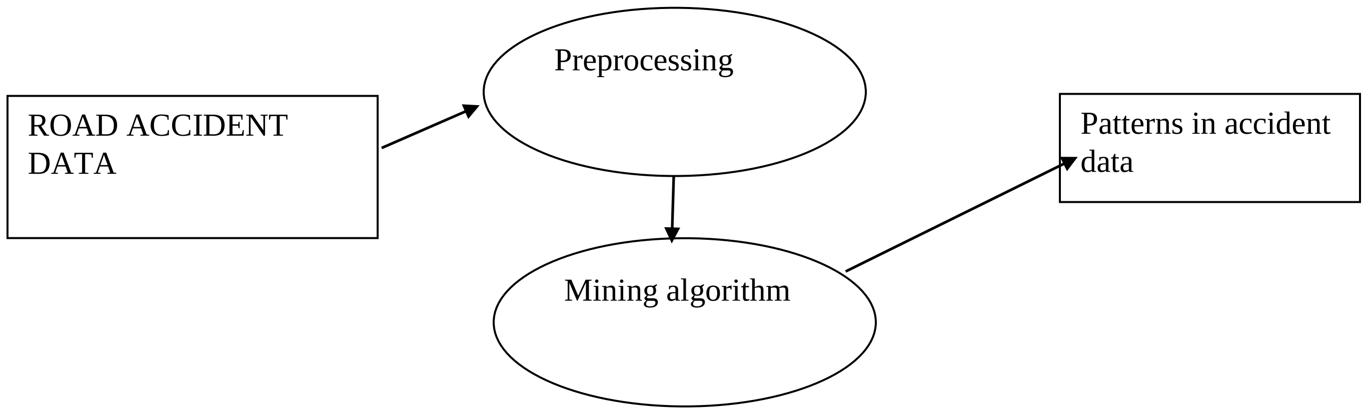
The following Data flow diagrams describe how the data flows in the project.

Context level :



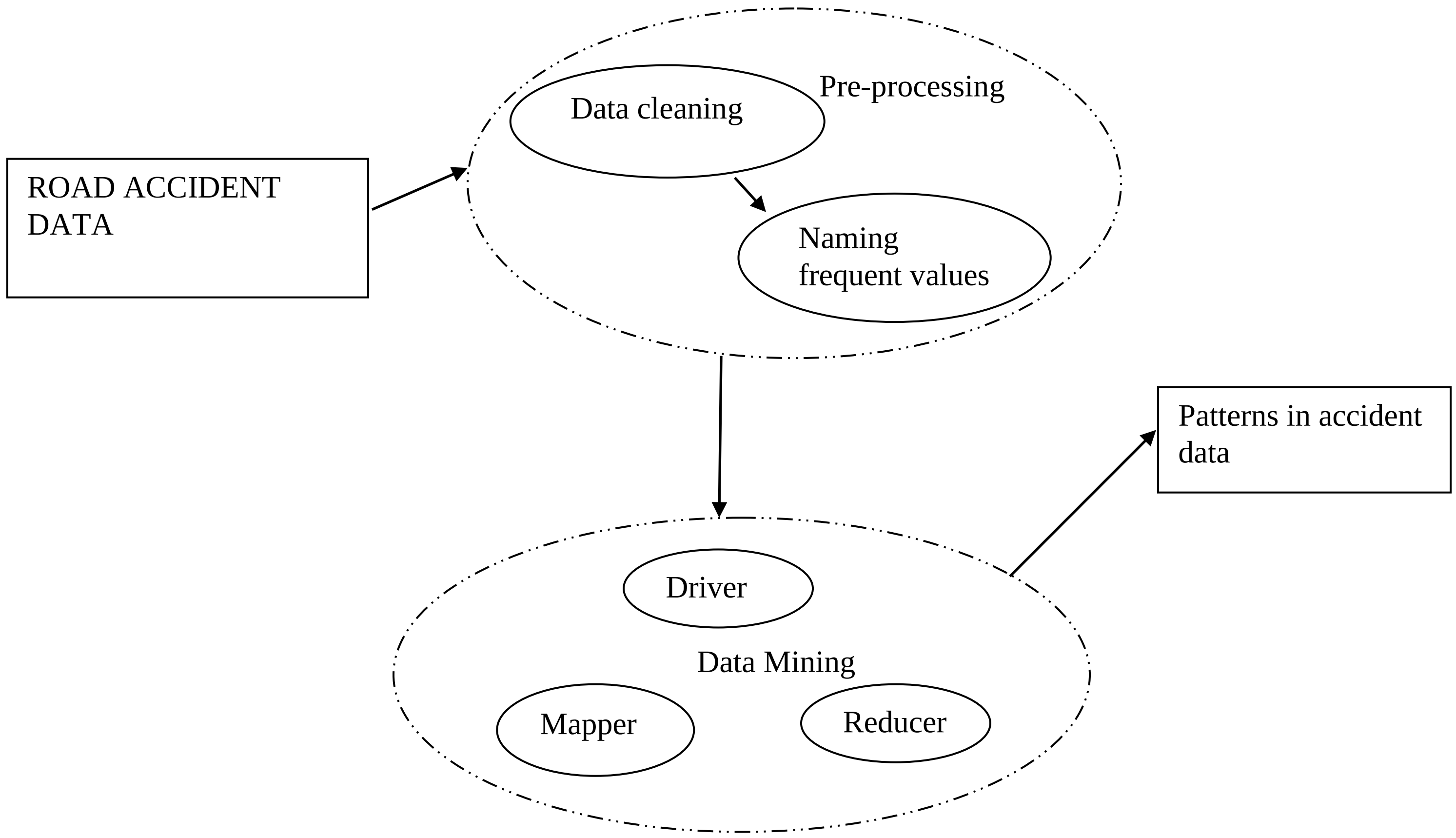
The input given is the raw road accident data that needs to be preprocessed using appropriate techniques like Aliasing, Default values, Averaging, etc to make the data uniform. Next the mining algorithm is applied to finally get required patterns of factors affecting road safety and analyze it.

Dfd level -1:



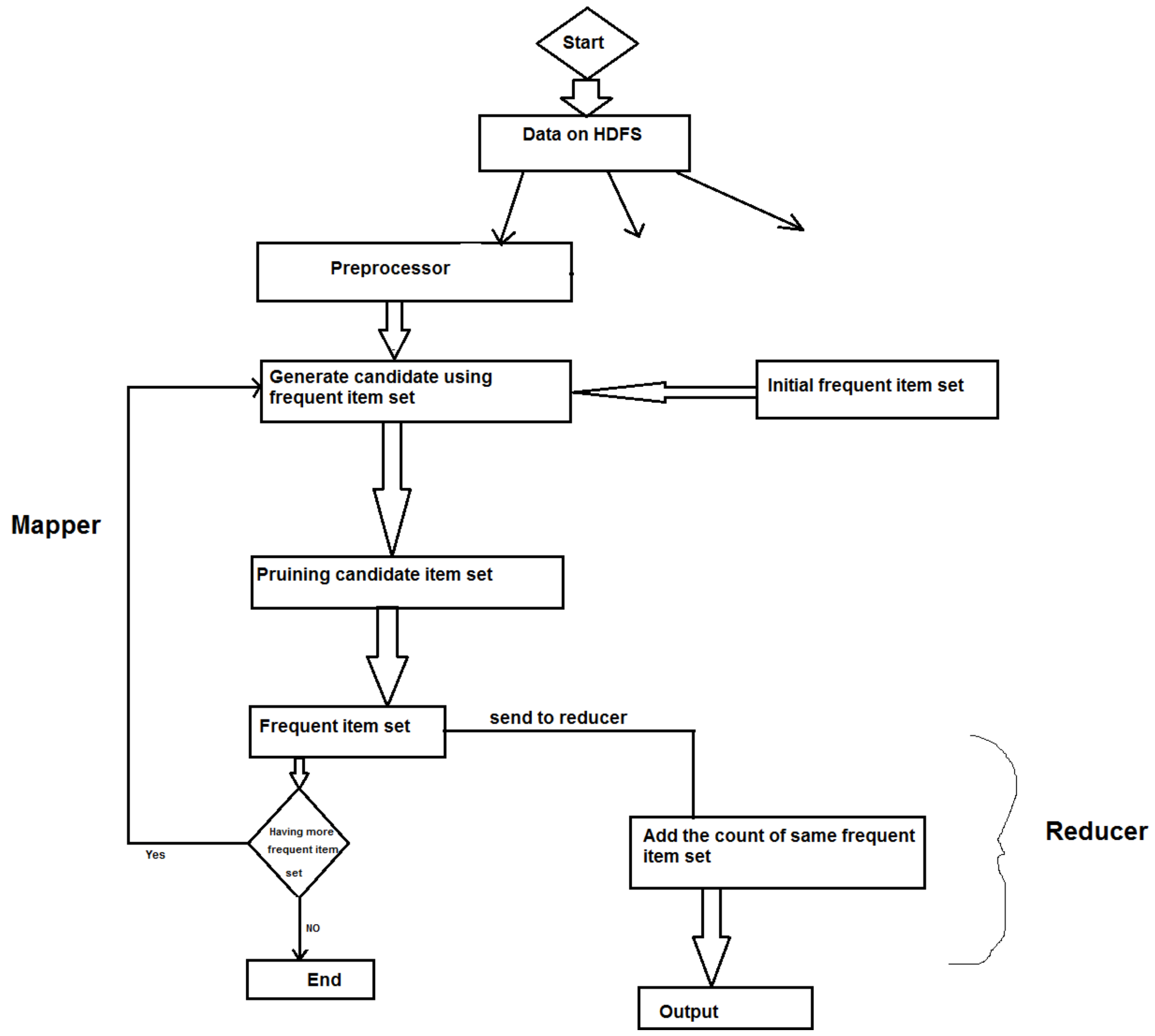
The data preprocessing usually involves data cleaning tasks as well as filling and naming empty fields in order to make data free from outliers. The algorithm phase further divides into distinct Hadoop processes of the Driver, Mapper and Reducer functions.

DFD level-2



Control Flow Diagram:

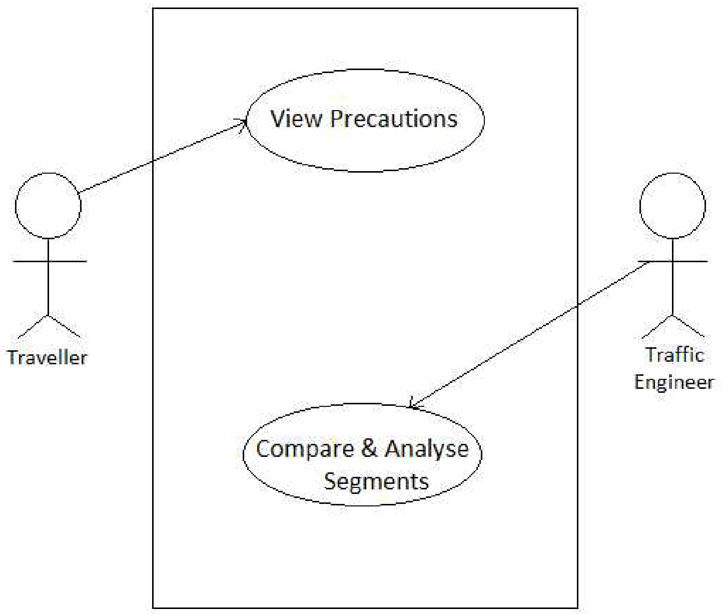
The control flow diagram shows how control passes in the program. First data is loaded onto Hadoop and when the Java code runs, the Job tracker splits the input file assigned to each split input to a single cluster. On each cluster a mapper runs a map function for each line of input. Within map function we have applied preprocessing technique and Apriori algorithm for frequent pattern mining. In reducer we count the frequency for each frequent item.


Start
Data on HDFS
Preprocessor
Generate candidate using frequent item set
Initial frequent item set
Mapper
Pruining candidate item set
Frequent item set
send to reducer
Having more frequent item set
Yes
NO
End
Add the count of same frequent item set
Reducer
Output

Use Case:

Use case diagram given below shows users of the results of the project. A traveler can easily conclude precautions by analyzing the factors affecting majority of road accidents and thereby employ measures to curb the same.

On the other hand, a traffic engineer can use results obtained to compare and analyze segments of road safety and improve methods to curb road and traffic accidents.



Use Case Diagram

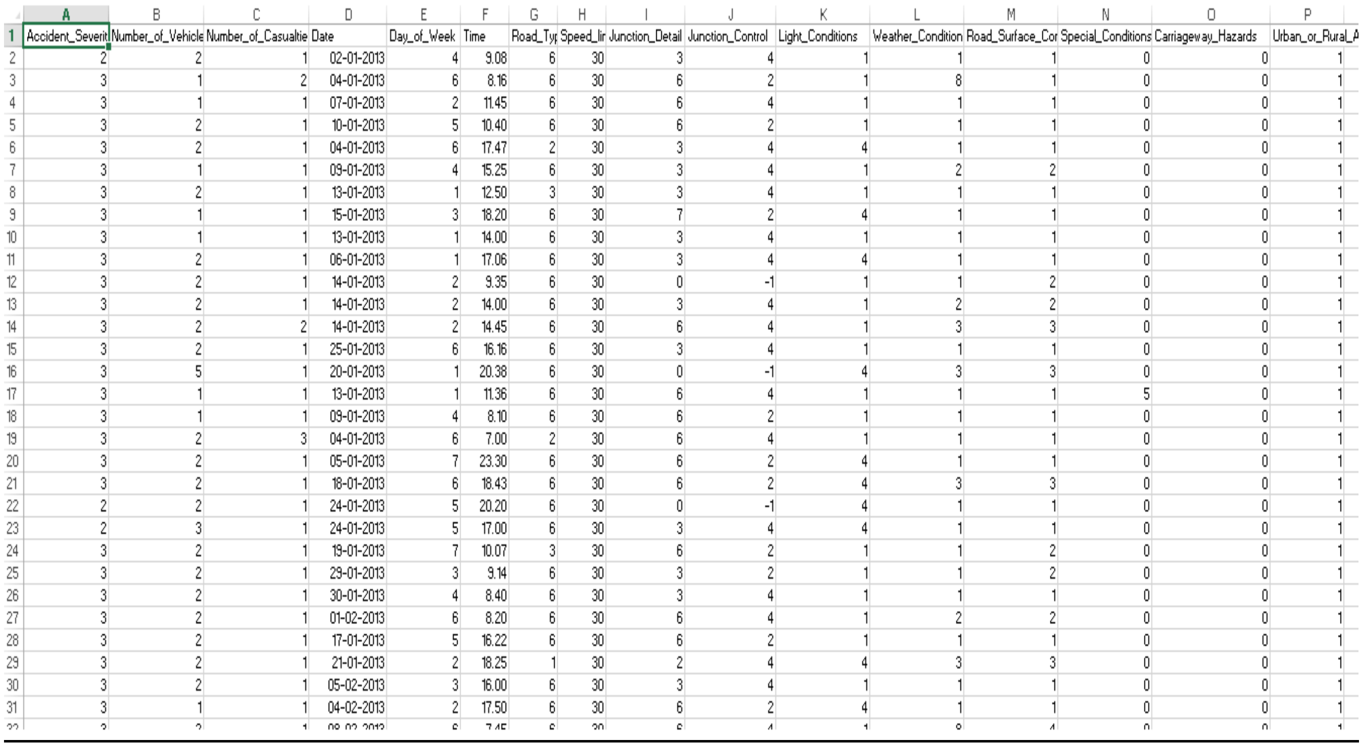
3. DATASET DESCRIPTION

Road Safety Data

These files provide detailed road safety data about the circumstances of personal injury road accidents in GB from 1979, the types (including Make and Model) of vehicles involved and the consequential casualties. The statistics relate only to personal injury accidents on public roads that are reported to the police, and subsequently recorded, using the STATS19 accident reporting form.

All the data variables are coded rather than containing textual strings. The lookup tables are available in the "Additional resources" section towards the bottom of the table. Accident, Vehicle and Casualty data for individual years 2005 - 2009 have now been removed but are available on request from roadacc.stats@dft.gsi.gov.uk. Data for these years is also available within the 2005 - Latest Year datasets.

Also includes: Results of breath-test screening data from recently introduced digital breath testing devices, as provided by Police Authorities in England and Wales.

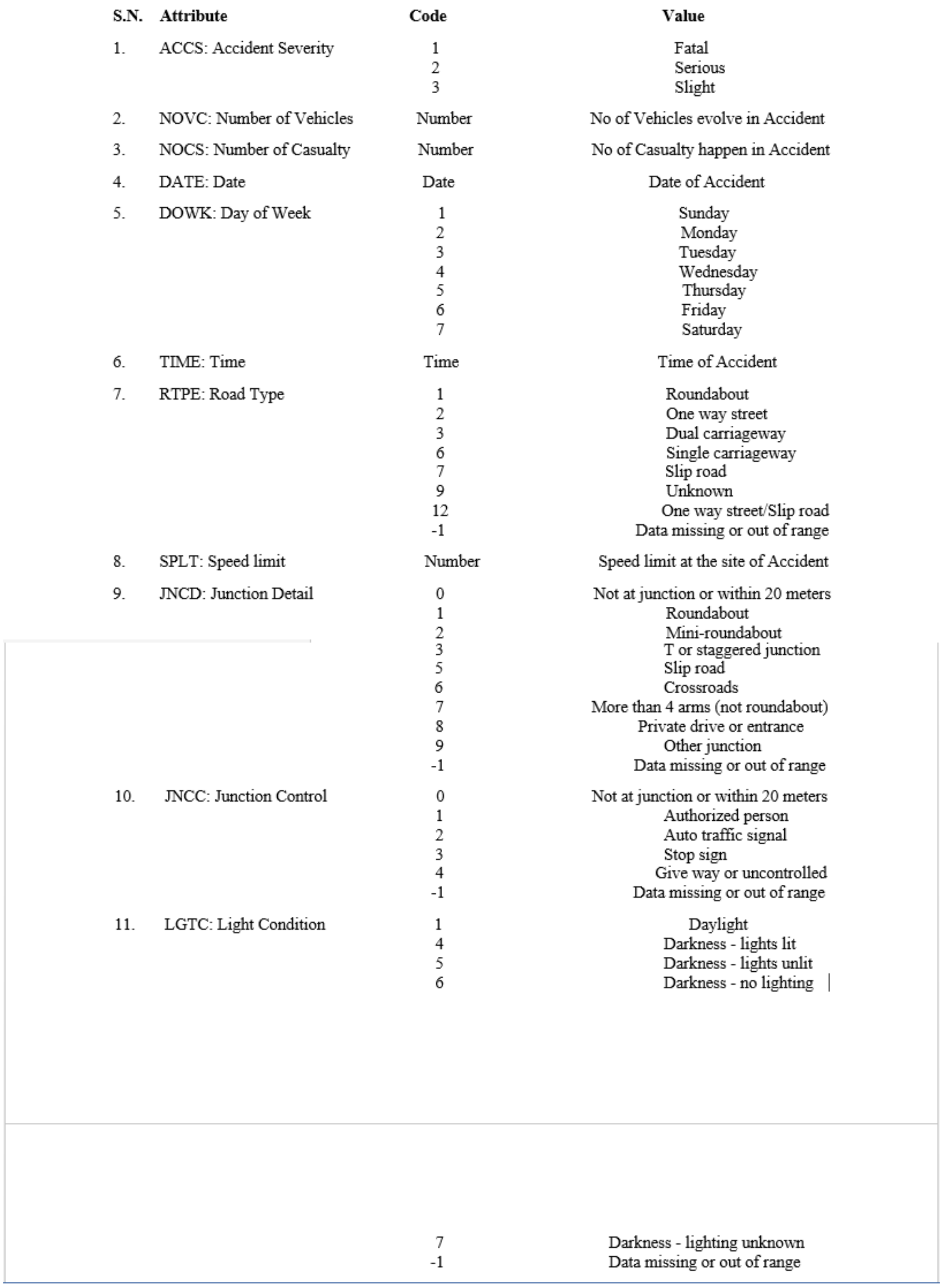


	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	Accident_Severit	Number_of_Vehicle	Number_of_Casualtie	Date	Day_of_Week	Time	Road_Ty	Speed_lir	Junction_Detail	Junction_Control	Light_Conditions	Weather_Condition	Road_Surface_Cor	Special_Conditions	Carriageway_Hazards	Urban_or_Rural_A
2	2	2	1	02-01-2013	4	9.08	6	30	3	4	1	1	1	0	0	1
3	3	1	2	04-01-2013	6	8.16	6	30	6	2	1	8	1	0	0	1
4	3	1	1	07-01-2013	2	11.45	6	30	6	4	1	1	1	0	0	1
5	3	2	1	10-01-2013	5	10.40	6	30	6	2	1	1	1	0	0	1
6	3	2	1	04-01-2013	6	17.47	2	30	3	4	4	1	1	0	0	1
7	3	1	1	09-01-2013	4	15.25	6	30	3	4	1	2	2	0	0	1
8	3	2	1	13-01-2013	1	12.50	3	30	3	4	1	1	1	0	0	1
9	3	1	1	15-01-2013	3	18.20	6	30	7	2	4	1	1	0	0	1
10	3	1	1	13-01-2013	1	14.00	6	30	3	4	1	1	1	0	0	1
11	3	2	1	06-01-2013	1	17.06	6	30	3	4	4	1	1	0	0	1
12	3	2	1	14-01-2013	2	9.35	6	30	0	-1	1	1	2	0	0	1
13	3	2	1	14-01-2013	2	14.00	6	30	3	4	1	2	2	0	0	1
14	3	2	2	14-01-2013	2	14.45	6	30	6	4	1	3	3	0	0	1
15	3	2	1	25-01-2013	6	16.16	6	30	3	4	1	1	1	0	0	1
16	3	5	1	20-01-2013	1	20.38	6	30	0	-1	4	3	3	0	0	1
17	3	1	1	13-01-2013	1	11.36	6	30	6	4	1	1	1	5	0	1
18	3	1	1	09-01-2013	4	8.10	6	30	6	2	1	1	1	0	0	1
19	3	2	3	04-01-2013	6	7.00	2	30	6	4	1	1	1	0	0	1
20	3	2	1	05-01-2013	7	23.30	6	30	6	2	4	1	1	0	0	1
21	3	2	1	18-01-2013	6	18.43	6	30	6	2	4	3	3	0	0	1
22	2	2	1	24-01-2013	5	20.20	6	30	0	-1	4	1	1	0	0	1
23	2	3	1	24-01-2013	5	17.00	6	30	3	4	4	1	1	0	0	1
24	3	2	1	19-01-2013	7	10.07	3	30	6	2	1	1	2	0	0	1
25	3	2	1	29-01-2013	3	9.14	6	30	3	2	1	1	2	0	0	1
26	3	2	1	30-01-2013	4	8.40	6	30	3	4	1	1	1	0	0	1
27	3	2	1	01-02-2013	6	8.20	6	30	6	4	1	2	2	0	0	1
28	3	2	1	17-01-2013	5	16.22	6	30	6	2	1	1	1	0	0	1
29	3	2	1	21-01-2013	2	18.25	1	30	2	4	4	3	3	0	0	1
30	3	2	1	05-02-2013	3	16.00	6	30	3	4	1	1	1	0	0	1
31	3	1	1	04-02-2013	2	17.50	6	30	6	2	4	1	1	0	0	1

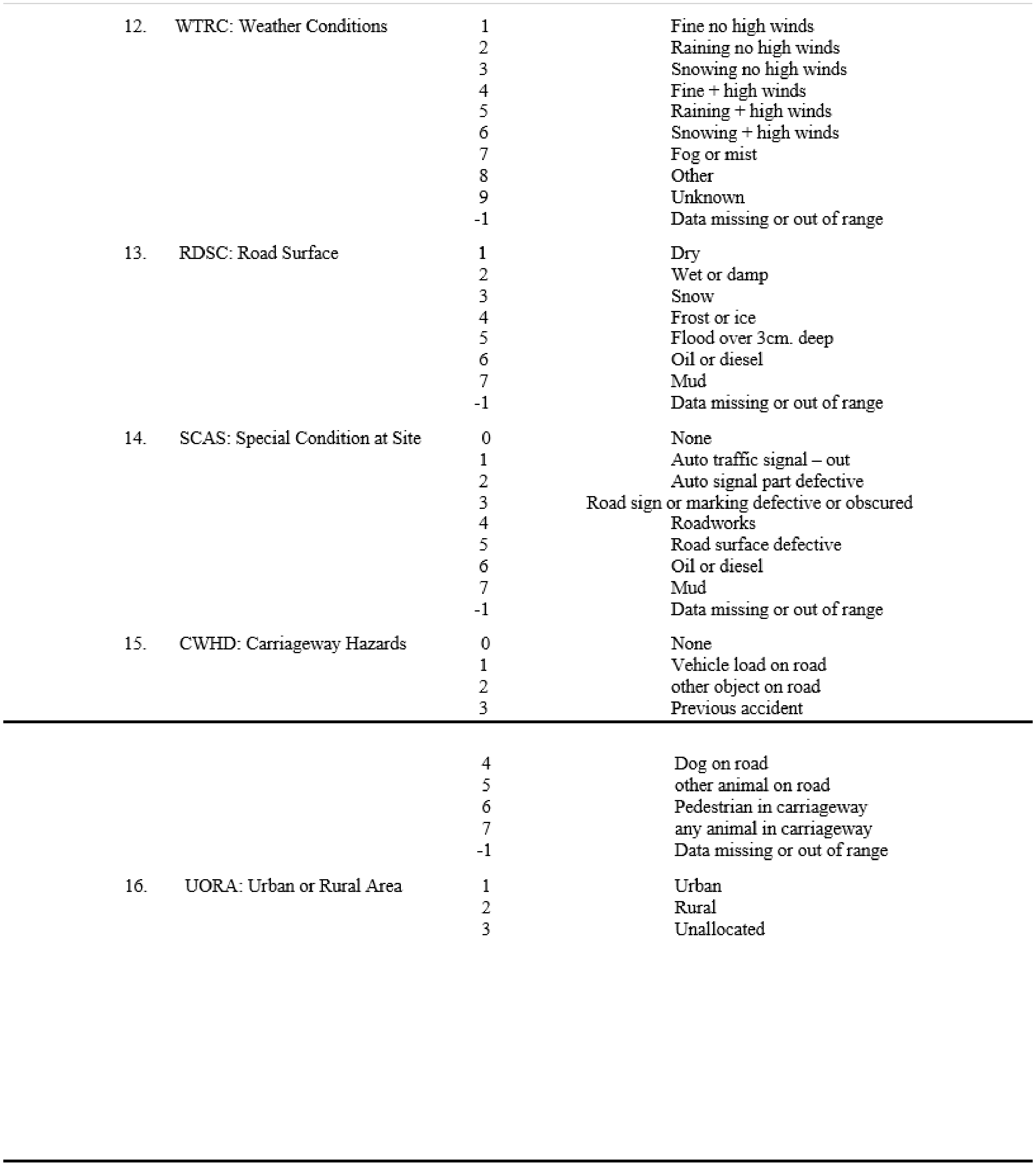
The above table shows the dataset used for analysis in the given project. The attributes relating to various road accidents are shown with their values.

Results of blood alcohol levels (milligrams / 100 millilitres of blood) provided by matching coroners' data (provided by Coroners in England and Wales and by Procurators Fiscal in Scotland) with fatality data from the STATS19 police data of road accidents in Great Britain. For cases when the Blood Alcohol Levels for a fatality are "unknown" are a consequence of an unsuccessful match between the two data sets.

ATTRIBUTE CODE AND VALUES:



S.N.	Attribute	Code	Value
1.	ACCS: Accident Severity	1	Fatal
		2	Serious
		3	Slight
2.	NOVC: Number of Vehicles	Number	No of Vehicles evolve in Accident
3.	NOCS: Number of Casualty	Number	No of Casualty happen in Accident
4.	DATE: Date	Date	Date of Accident
5.	DOWK: Day of Week	1	Sunday
		2	Monday
		3	Tuesday
		4	Wednesday
		5	Thursday
		6	Friday
		7	Saturday
6.	TIME: Time	Time	Time of Accident
7.	RTPE: Road Type	1	Roundabout
		2	One way street
		3	Dual carriageway
		6	Single carriageway
		7	Slip road
		9	Unknown
		12	One way street/Slip road
		-1	Data missing or out of range
8.	SPLT: Speed limit	Number	Speed limit at the site of Accident
9.	JNCD: Junction Detail	0	Not at junction or within 20 meters
		1	Roundabout
		2	Mini-roundabout
		3	T or staggered junction
		5	Slip road
		6	Crossroads
		7	More than 4 arms (not roundabout)
		8	Private drive or entrance
		9	Other junction
		-1	Data missing or out of range
10.	JNCC: Junction Control	0	Not at junction or within 20 meters
		1	Authorized person
		2	Auto traffic signal
		3	Stop sign
		4	Give way or uncontrolled
		-1	Data missing or out of range
11.	LGTC: Light Condition	1	Daylight
		4	Darkness - lights lit
		5	Darkness - lights unlit
		6	Darkness - no lighting
		7	Darkness - lighting unknown
		-1	Data missing or out of range



12.	WTRC: Weather Conditions	1	Fine no high winds
		2	Raining no high winds
		3	Snowing no high winds
		4	Fine + high winds
		5	Raining + high winds
		6	Snowing + high winds
		7	Fog or mist
		8	Other
		9	Unknown
		-1	Data missing or out of range
13.	RDSC: Road Surface	1	Dry
		2	Wet or damp
		3	Snow
		4	Frost or ice
		5	Flood over 3cm. deep
		6	Oil or diesel
		7	Mud
		-1	Data missing or out of range
14.	SCAS: Special Condition at Site	0	None
		1	Auto traffic signal – out
		2	Auto signal part defective
		3	Road sign or marking defective or obscured
		4	Roadworks
		5	Road surface defective
		6	Oil or diesel
		7	Mud
		-1	Data missing or out of range
15.	CWHD: Carriageway Hazards	0	None
		1	Vehicle load on road
		2	other object on road
		3	Previous accident
		4	Dog on road
		5	other animal on road
		6	Pedestrian in carriageway
		7	any animal in carriageway
		-1	Data missing or out of range
16.	UORA: Urban or Rural Area	1	Urban
		2	Rural
		3	Unallocated

The above attribute codes and values are used to classify different data in an easier manner.

4. SYSTEM REQUIREMENTS

A. HARDWARE REQUIREMENTS

Processor Type	:	Intel Xeon® CPU E31245 @ 3.30Ghz x 4.
System RAM	:	3.8 GiB
Graphics	:	Intel® Sandybridge Server
Input Device	:	Basic Keyboard and Scroll Mouse
Output Device	:	Standard Color Monitor
Storage Device	:	55.7 GB

B. SOFTWARE REQUIREMTNTS

Operating System	:	Ubuntu 14.04 LTS
Development Kit	:	Hadoop 2.6.0, Eclipse

5. CONFIGURATION:

Installing Software

If your cluster doesn't have the requisite software you will need to install it.

For example on Ubuntu Linux:

```
$ sudo apt-get install ssh
$ sudo apt-get install rsync
```

Download

To get a Hadoop distribution, download a recent stable release from one of the Apache Download Mirrors.

Prepare to Start the Hadoop Cluster

Unpack the downloaded Hadoop distribution. In the distribution, edit the file etc/hadoop/hadoop-env.sh to define some parameters as follows:

```
# set to the root of your Java installation
 export JAVA_HOME=/usr/java/latest
```

Try the following command:

```
$  bin/hadoop
```

This will display the usage documentation for the hadoop script.

Now you are ready to start your Hadoop cluster in one of the three supported modes:

- Local (Standalone) Mode
- Pseudo-Distributed Mode
- Fully-Distributed Mode

Standalone Operation

By default, Hadoop is configured to run in a non-distributed mode, as a single Java process. This is useful for debugging.

The following example copies the unpacked conf directory to use as input and then finds and displays every match of the given regular expression. Output is written to the given output directory.

```
$ mkdir input
```

```
  $ cp etc/hadoop/*.xml input
  $ bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-
examples-2.7.2.jar grep input output 'dfs[a-z.]+'
  $ cat output/*
```

Pseudo-Distributed Operation

Hadoop can also be run on a single-node in a pseudo-distributed mode where each Hadoop daemon runs in a separate Java process.

Configuration

Use the following:

etc/hadoop/core-site.xml:

```
<configuration>
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://localhost:9000</value>
    </property>
</configuration>
```

etc/hadoop/hdfs-site.xml:

```
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>
</configuration>
```

Setup passphraseless ssh

Now check that you can ssh to the localhost without a passphrase:

```
  $ ssh localhost
```

If you cannot ssh to localhost without a passphrase, execute the following commands:

```
  $ ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa
  $ cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys
  $ chmod 0600 ~/.ssh/authorized_keys
```

<u>Execution</u>

The following instructions are to run a MapReduce job locally. If you want to execute a job on YARN, see YARN on Single Node.

1. Format the filesystem:
2. `$ bin/hdfs namenode -format`
3. Start NameNode daemon and DataNode daemon:
4. `$ sbin/start-dfs.sh`

   The hadoop daemon log output is written to the $HADOOP_LOG_DIR directory (defaults to $HADOOP_HOME/logs).

5. Browse the web interface for the NameNode; by default it is available at:
   - NameNode - http://localhost:50070/
6. Make the HDFS directories required to execute MapReduce jobs:
7. `$ bin/hdfs dfs -mkdir /user`
8. `$ bin/hdfs dfs -mkdir /user/<username>`
9. Copy the input files into the distributed filesystem:
10. `$ bin/hdfs dfs -put etc/hadoop input`
11. Run some of the examples provided:
12. `$ bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.2.jar grep input output 'dfs[a-z.]+'`
13. Examine the output files: Copy the output files from the distributed filesystem to the local filesystem and examine them:
14. `$ bin/hdfs dfs -get output output`
15. `$ cat output/*`

    or

    View the output files on the distributed filesystem:

    ```
    $ bin/hdfs dfs -cat output/*
    ```

16. When you’re done, stop the daemons with:
17. `$ sbin/stop-dfs.sh`

YARN on a Single Node

You can run a MapReduce job on YARN in a pseudo-distributed mode by setting a few parameters and running ResourceManager daemon and NodeManager daemon in addition.

The following instructions assume that 1. ~ 4. steps of the above instructions are already executed.

1. Configure parameters as follows:etc/hadoop/mapred-site.xml:
2. `<configuration>`
3. `    <property>`
4. `        <name>mapreduce.framework.name</name>`
5. `        <value>yarn</value>`
6. `    </property>`
7. `</configuration>`

   etc/hadoop/yarn-site.xml:

   ```
   <configuration>
       <property>
           <name>yarn.nodemanager.aux-services</name>
           <value>mapreduce_shuffle</value>
       </property>
   </configuration>
   ```

8. Start ResourceManager daemon and NodeManager daemon:
9. `$ sbin/start-yarn.sh`
10. Browse the web interface for the ResourceManager; by default it is available at:
    - ResourceManager - http://localhost:8088/
11. Run a MapReduce job.
12. When you’re done, stop the daemons with:
13. `$ sbin/stop-yarn.sh`

6. SYSTEM IMPLEMENTATION

Starting Hadoop and running Java Program

For starting Hadoop there are two command for starting hdfs configuration:

1. start-dfs.sh
2. start-yarn.sh

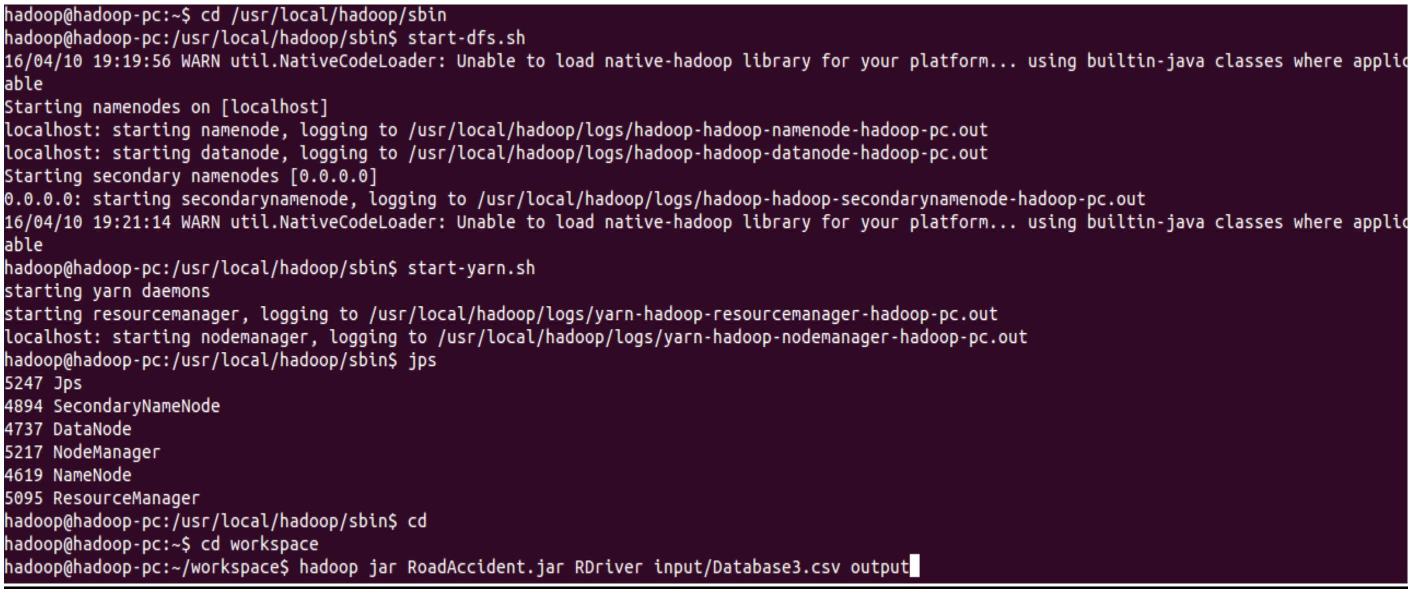
For checking running status of Hadoop there is a command

1. jps

It will show all running nodes on Hadoop.

For running Hadoop program first we have to create a jar file for java program then run using following command:

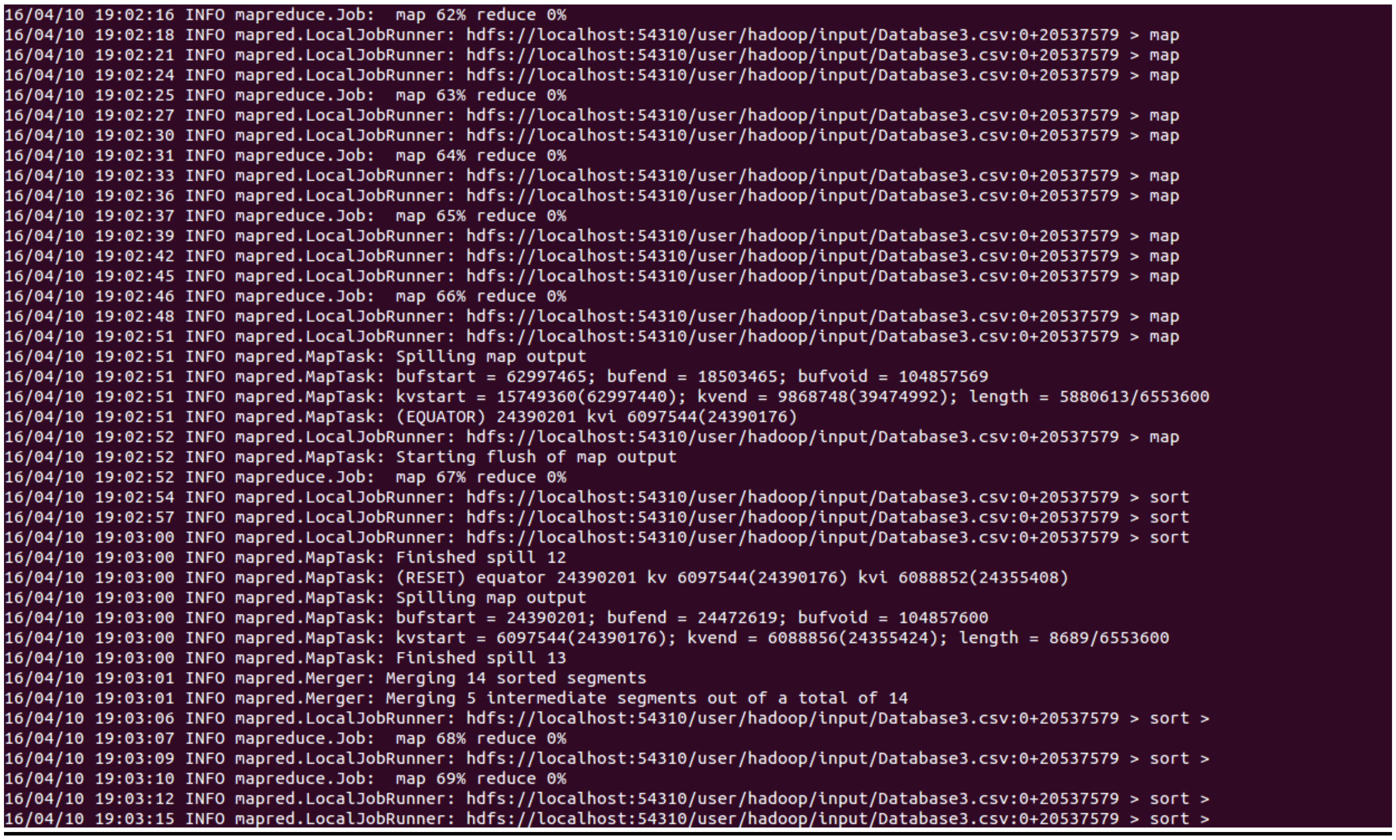
1. Hadoop jar path of jar file driver name input-path and output-path



```
hadoop@hadoop-pc:~$ cd /usr/local/hadoop/sbin
hadoop@hadoop-pc:/usr/local/hadoop/sbin$ start-dfs.sh
16/04/10 19:19:56 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applic
able
Starting namenodes on [localhost]
localhost: starting namenode, logging to /usr/local/hadoop/logs/hadoop-hadoop-namenode-hadoop-pc.out
localhost: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hadoop-datanode-hadoop-pc.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/logs/hadoop-hadoop-secondarynamenode-hadoop-pc.out
16/04/10 19:21:14 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applic
able
hadoop@hadoop-pc:/usr/local/hadoop/sbin$ start-yarn.sh
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/logs/yarn-hadoop-resourcemanager-hadoop-pc.out
localhost: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hadoop-nodemanager-hadoop-pc.out
hadoop@hadoop-pc:/usr/local/hadoop/sbin$ jps
5247 Jps
4894 SecondaryNameNode
4737 DataNode
5217 NodeManager
4619 NameNode
5095 ResourceManager
hadoop@hadoop-pc:/usr/local/hadoop/sbin$ cd
hadoop@hadoop-pc:~$ cd workspace
hadoop@hadoop-pc:~/workspace$ hadoop jar RoadAccident.jar RDriver input/Database3.csv output
```

Running status of mapper

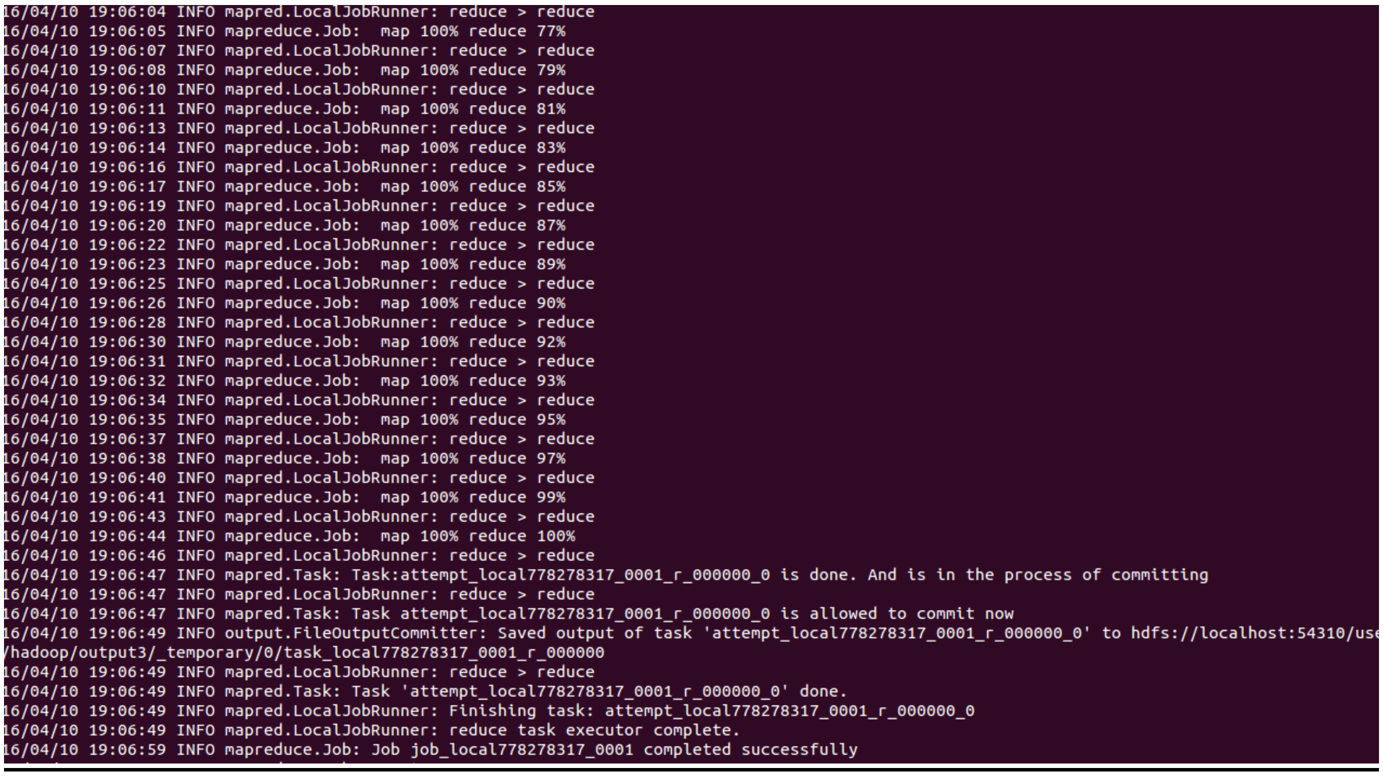
Firstly, mapper executes for all the split input file on Hadoop clusters.
A single map function runs for each input line.



```
16/04/10 19:02:16 INFO mapreduce.Job:  map 62% reduce 0%
16/04/10 19:02:18 INFO mapred.LocalJobRunner: hdfs://localhost:54310/user/hadoop/input/Database3.csv:0+20537579 > map
16/04/10 19:02:21 INFO mapred.LocalJobRunner: hdfs://localhost:54310/user/hadoop/input/Database3.csv:0+20537579 > map
16/04/10 19:02:24 INFO mapred.LocalJobRunner: hdfs://localhost:54310/user/hadoop/input/Database3.csv:0+20537579 > map
16/04/10 19:02:25 INFO mapreduce.Job:  map 63% reduce 0%
16/04/10 19:02:27 INFO mapred.LocalJobRunner: hdfs://localhost:54310/user/hadoop/input/Database3.csv:0+20537579 > map
16/04/10 19:02:30 INFO mapred.LocalJobRunner: hdfs://localhost:54310/user/hadoop/input/Database3.csv:0+20537579 > map
16/04/10 19:02:31 INFO mapreduce.Job:  map 64% reduce 0%
16/04/10 19:02:33 INFO mapred.LocalJobRunner: hdfs://localhost:54310/user/hadoop/input/Database3.csv:0+20537579 > map
16/04/10 19:02:36 INFO mapred.LocalJobRunner: hdfs://localhost:54310/user/hadoop/input/Database3.csv:0+20537579 > map
16/04/10 19:02:37 INFO mapreduce.Job:  map 65% reduce 0%
16/04/10 19:02:39 INFO mapred.LocalJobRunner: hdfs://localhost:54310/user/hadoop/input/Database3.csv:0+20537579 > map
16/04/10 19:02:42 INFO mapred.LocalJobRunner: hdfs://localhost:54310/user/hadoop/input/Database3.csv:0+20537579 > map
16/04/10 19:02:45 INFO mapred.LocalJobRunner: hdfs://localhost:54310/user/hadoop/input/Database3.csv:0+20537579 > map
16/04/10 19:02:46 INFO mapreduce.Job:  map 66% reduce 0%
16/04/10 19:02:48 INFO mapred.LocalJobRunner: hdfs://localhost:54310/user/hadoop/input/Database3.csv:0+20537579 > map
16/04/10 19:02:51 INFO mapred.LocalJobRunner: hdfs://localhost:54310/user/hadoop/input/Database3.csv:0+20537579 > map
16/04/10 19:02:51 INFO mapred.MapTask: Spilling map output
16/04/10 19:02:51 INFO mapred.MapTask: bufstart = 62997465; bufend = 18503465; bufvoid = 104857569
16/04/10 19:02:51 INFO mapred.MapTask: kvstart = 15749360(62997440); kvend = 9868748(39474992); length = 5880613/6553600
16/04/10 19:02:51 INFO mapred.MapTask: (EQUATOR) 24390201 kvi 6097544(24390176)
16/04/10 19:02:52 INFO mapred.LocalJobRunner: hdfs://localhost:54310/user/hadoop/input/Database3.csv:0+20537579 > map
16/04/10 19:02:52 INFO mapred.MapTask: Starting flush of map output
16/04/10 19:02:52 INFO mapreduce.Job:  map 67% reduce 0%
16/04/10 19:02:54 INFO mapred.LocalJobRunner: hdfs://localhost:54310/user/hadoop/input/Database3.csv:0+20537579 > sort
16/04/10 19:02:57 INFO mapred.LocalJobRunner: hdfs://localhost:54310/user/hadoop/input/Database3.csv:0+20537579 > sort
16/04/10 19:03:00 INFO mapred.LocalJobRunner: hdfs://localhost:54310/user/hadoop/input/Database3.csv:0+20537579 > sort
16/04/10 19:03:00 INFO mapred.MapTask: Finished spill 12
16/04/10 19:03:00 INFO mapred.MapTask: (RESET) equator 24390201 kv 6097544(24390176) kvi 6088852(24355408)
16/04/10 19:03:00 INFO mapred.MapTask: Spilling map output
16/04/10 19:03:00 INFO mapred.MapTask: bufstart = 24390201; bufend = 24472619; bufvoid = 104857600
16/04/10 19:03:00 INFO mapred.MapTask: kvstart = 6097544(24390176); kvend = 6088856(24355424); length = 8689/6553600
16/04/10 19:03:00 INFO mapred.MapTask: Finished spill 13
16/04/10 19:03:01 INFO mapred.Merger: Merging 14 sorted segments
16/04/10 19:03:01 INFO mapred.Merger: Merging 5 intermediate segments out of a total of 14
16/04/10 19:03:06 INFO mapred.LocalJobRunner: hdfs://localhost:54310/user/hadoop/input/Database3.csv:0+20537579 > sort >
16/04/10 19:03:07 INFO mapreduce.Job:  map 68% reduce 0%
16/04/10 19:03:09 INFO mapred.LocalJobRunner: hdfs://localhost:54310/user/hadoop/input/Database3.csv:0+20537579 > sort >
16/04/10 19:03:10 INFO mapreduce.Job:  map 69% reduce 0%
16/04/10 19:03:12 INFO mapred.LocalJobRunner: hdfs://localhost:54310/user/hadoop/input/Database3.csv:0+20537579 > sort >
16/04/10 19:03:15 INFO mapred.LocalJobRunner: hdfs://localhost:54310/user/hadoop/input/Database3.csv:0+20537579 > sort >
```

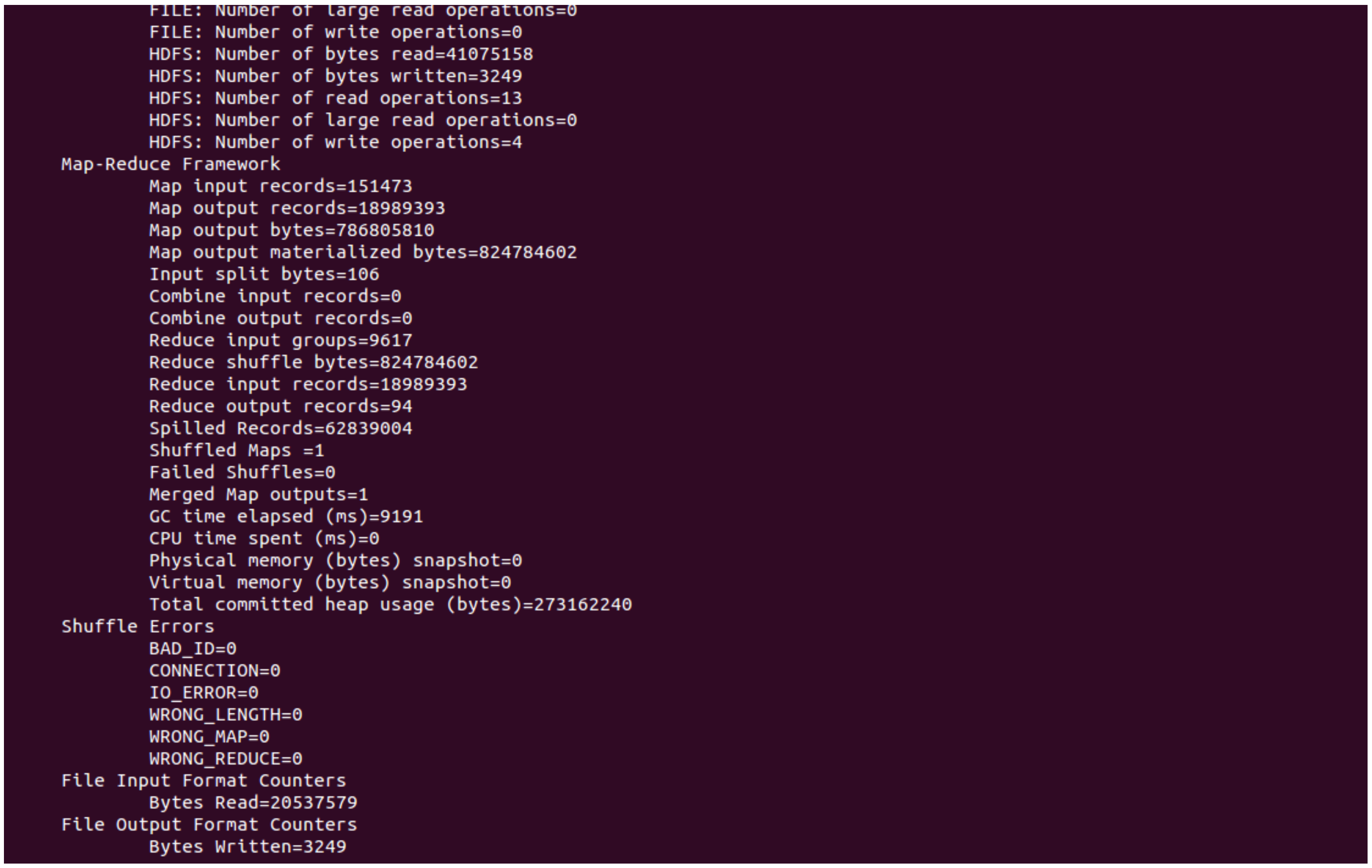
Running status of Reducer

After mapper function executes, the reducer starts running to generate final output. Number of reducers depends upon number of output values we want. Reducer counts frequency of same frequent item and yields the final output.


```
16/04/10 19:06:04 INFO mapred.LocalJobRunner: reduce > reduce
16/04/10 19:06:05 INFO mapreduce.Job:  map 100% reduce 77%
16/04/10 19:06:07 INFO mapred.LocalJobRunner: reduce > reduce
16/04/10 19:06:08 INFO mapreduce.Job:  map 100% reduce 79%
16/04/10 19:06:10 INFO mapred.LocalJobRunner: reduce > reduce
16/04/10 19:06:11 INFO mapreduce.Job:  map 100% reduce 81%
16/04/10 19:06:13 INFO mapred.LocalJobRunner: reduce > reduce
16/04/10 19:06:14 INFO mapreduce.Job:  map 100% reduce 83%
16/04/10 19:06:16 INFO mapred.LocalJobRunner: reduce > reduce
16/04/10 19:06:17 INFO mapreduce.Job:  map 100% reduce 85%
16/04/10 19:06:19 INFO mapred.LocalJobRunner: reduce > reduce
16/04/10 19:06:20 INFO mapreduce.Job:  map 100% reduce 87%
16/04/10 19:06:22 INFO mapred.LocalJobRunner: reduce > reduce
16/04/10 19:06:23 INFO mapreduce.Job:  map 100% reduce 89%
16/04/10 19:06:25 INFO mapred.LocalJobRunner: reduce > reduce
16/04/10 19:06:26 INFO mapreduce.Job:  map 100% reduce 90%
16/04/10 19:06:28 INFO mapred.LocalJobRunner: reduce > reduce
16/04/10 19:06:30 INFO mapreduce.Job:  map 100% reduce 92%
16/04/10 19:06:31 INFO mapred.LocalJobRunner: reduce > reduce
16/04/10 19:06:32 INFO mapreduce.Job:  map 100% reduce 93%
16/04/10 19:06:34 INFO mapred.LocalJobRunner: reduce > reduce
16/04/10 19:06:35 INFO mapreduce.Job:  map 100% reduce 95%
16/04/10 19:06:37 INFO mapred.LocalJobRunner: reduce > reduce
16/04/10 19:06:38 INFO mapreduce.Job:  map 100% reduce 97%
16/04/10 19:06:40 INFO mapred.LocalJobRunner: reduce > reduce
16/04/10 19:06:41 INFO mapreduce.Job:  map 100% reduce 99%
16/04/10 19:06:43 INFO mapred.LocalJobRunner: reduce > reduce
16/04/10 19:06:44 INFO mapreduce.Job:  map 100% reduce 100%
16/04/10 19:06:46 INFO mapred.LocalJobRunner: reduce > reduce
16/04/10 19:06:47 INFO mapred.Task: Task:attempt_local778278317_0001_r_000000_0 is done. And is in the process of committing
16/04/10 19:06:47 INFO mapred.LocalJobRunner: reduce > reduce
16/04/10 19:06:47 INFO mapred.Task: Task attempt_local778278317_0001_r_000000_0 is allowed to commit now
16/04/10 19:06:49 INFO output.FileOutputCommitter: Saved output of task 'attempt_local778278317_0001_r_000000_0' to hdfs://localhost:54310/use
/hadoop/output3/_temporary/0/task_local778278317_0001_r_000000
16/04/10 19:06:49 INFO mapred.LocalJobRunner: reduce > reduce
16/04/10 19:06:49 INFO mapred.Task: Task 'attempt_local778278317_0001_r_000000_0' done.
16/04/10 19:06:49 INFO mapred.LocalJobRunner: Finishing task: attempt_local778278317_0001_r_000000_0
16/04/10 19:06:49 INFO mapred.LocalJobRunner: reduce task executor complete.
16/04/10 19:06:59 INFO mapreduce.Job: Job job_local778278317_0001 completed successfully
```

<u>Final Status of Program</u>

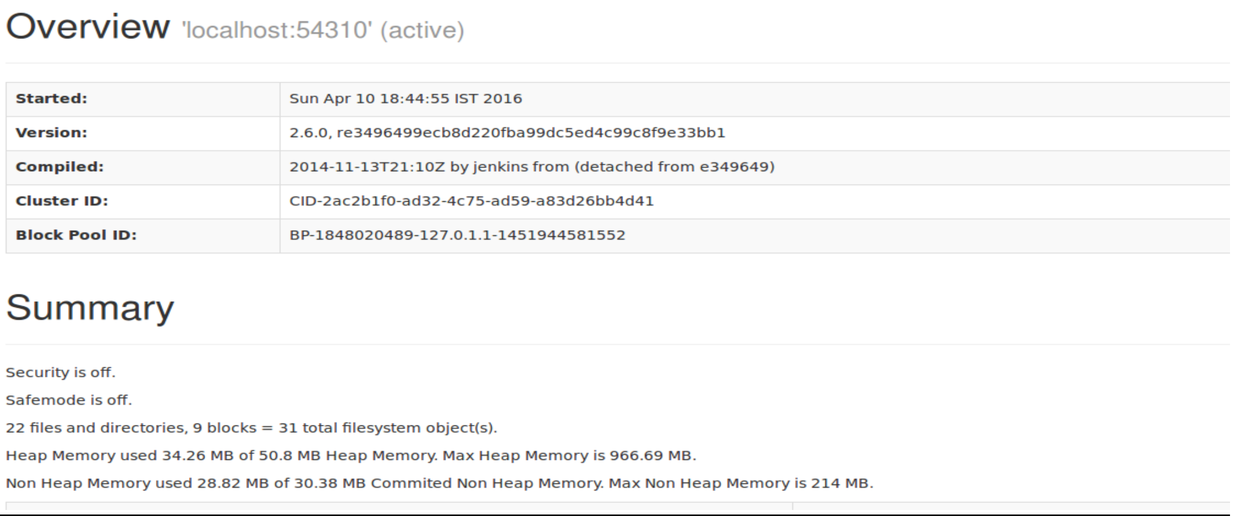
Final status of program running command shows how many byte are read from input file and how many byte are to be written to the output file.



```
		FILE: Number of large read operations=0
		FILE: Number of write operations=0
		HDFS: Number of bytes read=41075158
		HDFS: Number of bytes written=3249
		HDFS: Number of read operations=13
		HDFS: Number of large read operations=0
		HDFS: Number of write operations=4
	Map-Reduce Framework
		Map input records=151473
		Map output records=18989393
		Map output bytes=786805810
		Map output materialized bytes=824784602
		Input split bytes=106
		Combine input records=0
		Combine output records=0
		Reduce input groups=9617
		Reduce shuffle bytes=824784602
		Reduce input records=18989393
		Reduce output records=94
		Spilled Records=62839004
		Shuffled Maps =1
		Failed Shuffles=0
		Merged Map outputs=1
		GC time elapsed (ms)=9191
		CPU time spent (ms)=0
		Physical memory (bytes) snapshot=0
		Virtual memory (bytes) snapshot=0
		Total committed heap usage (bytes)=273162240
	Shuffle Errors
		BAD_ID=0
		CONNECTION=0
		IO_ERROR=0
		WRONG_LENGTH=0
		WRONG_MAP=0
		WRONG_REDUCE=0
	File Input Format Counters
		Bytes Read=20537579
	File Output Format Counters
		Bytes Written=3249
```

Hadoop Configuration Status

This is the Hadoop configuration status accessed by browser using localhost//:50070



Overview 'localhost:54310' (active)

Started:	Sun Apr 10 18:44:55 IST 2016
Version:	2.6.0, re3496499ecb8d220fba99dc5ed4c99c8f9e33bb1
Compiled:	2014-11-13T21:10Z by jenkins from (detached from e349649)
Cluster ID:	CID-2ac2b1f0-ad32-4c75-ad59-a83d26bb4d41
Block Pool ID:	BP-1848020489-127.0.1.1-1451944581552

Summary

Security is off.

Safemode is off.

22 files and directories, 9 blocks = 31 total filesystem object(s).

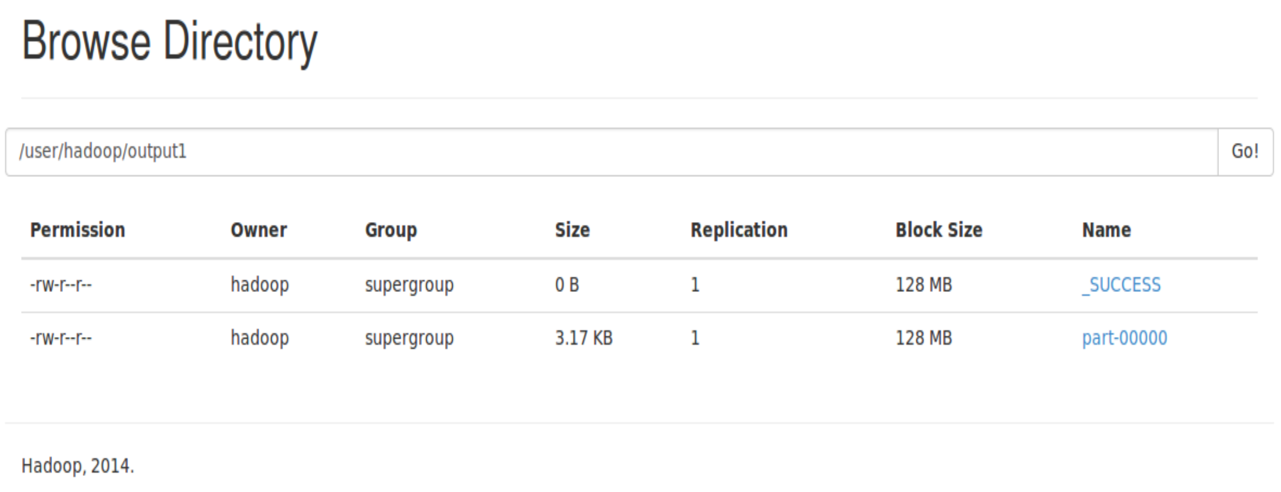
Heap Memory used 34.26 MB of 50.8 MB Heap Memory. Max Heap Memory is 966.69 MB.

Non Heap Memory used 28.82 MB of 30.38 MB Commited Non Heap Memory. Max Non Heap Memory is 214 MB.

<u>Hadoop output directory</u>

This is the Hadoop output directory accessed by browser using localhost//:50070

From there we have to download the output file



Browse Directory

/user/hadoop/output1

Permission	Owner	Group	Size	Replication	Block Size	Name
-rw-r--r--	hadoop	supergroup	0 B	1	128 MB	_SUCCESS
-rw-r--r--	hadoop	supergroup	3.17 KB	1	128 MB	part-00000

Hadoop, 2014.

7. Code:

Mapper function :

```
import java.io.IOException;
import java.util.Vector;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;


public class RMapper extends MapReduceBase implements
Mapper<LongWritable,Text,Text,IntWritable>{
	//Function genrate Candidate itemset using Frequent itemset
	Vector< Vector<String> > Gen_Candidate(Vector< Vector<String> > Frequent){
		Vector< Vector<String> >Candidate=new Vector< Vector<String> >();
		//check (size -1) element of ith and jth freqent itemset match or not and
last one different on not
		for(int i=0;i<Frequent.size();i++){
			for(int j=i+1;j<Frequent.size();j++){
				boolean ismatch=true;
				int size=Frequent.get(i).size();
				for(int k=0;k<size-
1;k++)if(!Frequent.get(i).get(k).equals(Frequent.get(j).get(k)))ismatch=false;
				if(Frequent.get(i).get(size-1).equals(Frequent.get(j).get(size-
1)))ismatch=false;
				//if match is found then then create a new candidate item by
taking union of ith and jth frequent itemsets
				if(ismatch){
					Vector<String>item=new Vector<String>();
					for(int k=0;k<size-
1;k++)item.add(Frequent.get(i).get(k));
					item.add(Frequent.get(i).get(size-1));
					item.add(Frequent.get(j).get(size-1));
					Candidate.add(item);
				}
			}
		}
		return Candidate;
	}
```

```
        //check element of item are present in database or not
        boolean match(Vector<String>item,Vector<String>Database){
                for(int i=0;i<item.size();i++){
                        boolean ismatch=false;
                        for(int
j=0;j<Database.size();j++)if(item.get(i).equals(Database.get(j)))ismatch=true;
                        if(ismatch==false)return false;
                }
                return true;
        }

        //preprocces database items
        Vector<String> Preprocessor(String str){
                Vector<String>Database=new Vector<String>();
                Vector<String>Prefix=new Vector<String>();
                //this short name of used attribute for appending with attribute vakue
                String
pre="DATA,LOEO,LONO,DATA,DATA,DATA,ACCS,NOVC,NOCS,DATE,DOWK,TIME,
LADT,LAHW,DATA,DATA,RTPE,SPLT,JNCD,JNCC,DATA,DATA,DATA,DATA,LGTC,
WTRC,RDSC,SCAS,CWHD,UORA,DATA,DATA";
      for(String word:pre.split(","))if(word.length()>0)Prefix.add(new String(word));
                int k=0;
                for(String word:str.split(",")){
                        if(word.length()>0&&k<Prefix.size()){
                                String atr=Prefix.get(k++)+"("+word+")";
                                Database.add(atr);
                        }
                }
                return Database;
        }

        @Override
        public void map(LongWritable key, Text value,OutputCollector<Text,
IntWritable> output, Reporter r)throws IOException {
                //Datastructure for storing frequent item sets of aproiri algorithm
                Vector<Vector<String>>Frequent=new Vector<Vector<String>>();
                //Datastructure for storing candidate item sets of aproiri algorithm
                Vector<Vector<String>>Candidate=new Vector<Vector<String>>();
                //Datastructure for storing database which is passed to that map function
                Vector<String>Database=new Vector<String>();

                String str=value.toString();
                Database=Preprocessor(str);
                //Initial frequent item which appear major time in attribute coloumn
                String
Freq="LOEO(198460),LONO(894000),ACCS(3),NOVC(2),NOCS(1),DATE(18/01/1979)
,DOWK(5),TIME(8:00),LADT(11),LAHW(9999),RTPE(1),SPLT(30),JNCD(1),JNCC(4),
LGTC(1),WTRC(8),RDSC(1),SCAS(-1),CWHD(0),UORA(-1)";
                //split intial frequent item and store in Frequent Item datastructure
                for(String word:Freq.split(",")){
                        Vector<String>item=new Vector<String>();
                item.add(new String(word));
                Frequent.add(item);
                }
                //approiri algorithm
            while(!Frequent.isEmpty()){
                Candidate.clear();
                //Generate candidate item using Frequent Itemsets passed to it
                Candidate=Gen_Candidate(Frequent);
                Frequent.clear();
                //check if candidate item present in datastructure or not if present then
this is the frequent item
                for(int i=0;i<Candidate.size();i++){

        if(match(Candidate.get(i),Database))Frequent.add(Candidate.get(i));
                }
                //convert every frequent itemsets into string and send it to reducer
                for(int i=0;i<Frequent.size();i++){
                        String pattern="{";
                        for(int          j=0;j<Frequent.get(i).size();j++)pattern=pattern+"
"+Frequent.get(i).get(j);
                        pattern=pattern+" }";
                        output.collect(new Text(pattern), new IntWritable(1));
                }
            }
        }
}
```

Reducer function:

```
import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;


public class RReducer extends MapReduceBase implements
Reducer<Text,IntWritable,Text,IntWritable>{
        //overwitten reduce funtion of Reducer
        @Override
        public void reduce(Text key, Iterator<IntWritable> values,OutputCollector<Text,
IntWritable> output, Reporter r)throws IOException {
                int count=0;
                //set minimum support count for taking as frequent item
                int min_sup=50000;
                //couting the frequency of particular pattern
                while(values.hasNext()){
                        IntWritable i=values.next();
                        count+=i.get();
                }
                //frequency of frequent item is greater than minimum support then put it
in output
                if(count>=min_sup)output.collect(key, new IntWritable(count));
        }
}
```

<u>Driver function:</u>

```
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;


public class RDriver extends Configured implements Tool {

        @Override
        public int run(String[] args) throws Exception {
                if(args.length<2){
                        System.out.println("Give proper input and output directory");
                        return -1;
                }
                JobConf conf=new JobConf(RDriver.class);

                //Set input and output path from taking input and providing out on hdfs
                FileInputFormat.setInputPaths(conf, new Path(args[0]));
```

```
		FileOutputFormat.setOutputPath(conf, new Path(args[1]));

		//set mapper and reducer class for execution in map phase and reduce
phase
		conf.setMapperClass(RMapper.class);
		conf.setReducerClass(RReducer.class);

		//set mapper output key and value type
		conf.setMapOutputKeyClass(Text.class);
		conf.setMapOutputValueClass(IntWritable.class);

		//set final output key and value type
		conf.setOutputKeyClass(Text.class);
		conf.setOutputValueClass(IntWritable.class);

		//submit the job
		JobClient.runJob(conf);
		return 0;
	}

	public static void main(String[] args) throws Exception{
		int exitcode=ToolRunner.run(new RDriver(), args);
		System.out.println(exitcode);
	}

}
```

8. OUTPUT

Following are the frequent patterns or combination of factors which appear more frequently in road accident data as analyzed in this project.

```
{ ACCS(3) CWHD(0) }          126521
{ ACCS(3) JNCC(4) CWHD(0) }          64786
{ ACCS(3) JNCC(4) LGTC(1) } 50124
{ ACCS(3) JNCC(4) }          65377
{ ACCS(3) LGTC(1) CWHD(0) }          94781
{ ACCS(3) LGTC(1) RDSC(1) CWHD(0) }          73743
{ ACCS(3) LGTC(1) RDSC(1) } 74743
{ ACCS(3) LGTC(1) }          96164
{ ACCS(3) NOCS(1) CWHD(0) }          98219
{ ACCS(3) NOCS(1) JNCC(4) CWHD(0) }          50822
{ ACCS(3) NOCS(1) JNCC(4) }51272
{ ACCS(3) NOCS(1) LGTC(1) CWHD(0) }          73903
{ ACCS(3) NOCS(1) LGTC(1) RDSC(1) CWHD(0) } 58068
{ ACCS(3) NOCS(1) LGTC(1) RDSC(1) }  58809
{ ACCS(3) NOCS(1) LGTC(1) } 74942
{ ACCS(3) NOCS(1) RDSC(1) CWHD(0) }          72494
{ ACCS(3) NOCS(1) RDSC(1) }73568
{ ACCS(3) NOCS(1) SPLT(30) CWHD(0) }          69299
{ ACCS(3) NOCS(1) SPLT(30) LGTC(1) CWHD(0) }          52406
{ ACCS(3) NOCS(1) SPLT(30) LGTC(1) }52940
{ ACCS(3) NOCS(1) SPLT(30) RDSC(1) CWHD(0) }          53511
{ ACCS(3) NOCS(1) SPLT(30) RDSC(1) }54059
{ ACCS(3) NOCS(1) SPLT(30) }          70067
{ ACCS(3) NOCS(1) }          99851
{ ACCS(3) NOVC(2) CWHD(0) }          78829
{ ACCS(3) NOVC(2) LGTC(1) CWHD(0) }          60645
{ ACCS(3) NOVC(2) LGTC(1) }          61185
```

{ ACCS(3) NOVC(2) NOCS(1) CWHD(0) }	59933
{ ACCS(3) NOVC(2) NOCS(1) }	60460
{ ACCS(3) NOVC(2) RDSC(1) CWHD(0) }	59269
{ ACCS(3) NOVC(2) RDSC(1) }	59768
{ ACCS(3) NOVC(2) SPLT(30) CWHD(0) }	54691
{ ACCS(3) NOVC(2) SPLT(30) }	55057
{ ACCS(3) NOVC(2) }	79582
{ ACCS(3) RDSC(1) CWHD(0) }	92348
{ ACCS(3) RDSC(1) }	93800
{ ACCS(3) SPLT(30) CWHD(0) }	85055
{ ACCS(3) SPLT(30) LGTC(1) CWHD(0) }	64005
{ ACCS(3) SPLT(30) LGTC(1) RDSC(1) CWHD(0) }	51972
{ ACCS(3) SPLT(30) LGTC(1) RDSC(1) }	52491
{ ACCS(3) SPLT(30) LGTC(1) }	64676
{ ACCS(3) SPLT(30) RDSC(1) CWHD(0) }	65126
{ ACCS(3) SPLT(30) RDSC(1) }	65827
{ ACCS(3) SPLT(30) }	86019
{ JNCC(4) CWHD(0) }	74889
{ JNCC(4) LGTC(1) CWHD(0) }	56980
{ JNCC(4) LGTC(1) }	57459
{ JNCC(4) RDSC(1) CWHD(0) }	56156
{ JNCC(4) RDSC(1) }	56654
{ LGTC(1) CWHD(0) }	110409
{ LGTC(1) RDSC(1) CWHD(0) }	86199
{ LGTC(1) RDSC(1) }	87379
{ NOCS(1) CWHD(0) }	114942
{ NOCS(1) JNCC(4) CWHD(0) }	58700
{ NOCS(1) JNCC(4) }	59228
{ NOCS(1) LGTC(1) CWHD(0) }	85734
{ NOCS(1) LGTC(1) RDSC(1) CWHD(0) }	67717
{ NOCS(1) LGTC(1) RDSC(1) }	68605
{ NOCS(1) LGTC(1) }	86953

{ NOCS(1) RDSC(1) CWHD(0) }	85142
{ NOCS(1) RDSC(1) }	86432
{ NOCS(1) SPLT(30) CWHD(0) }	80110
{ NOCS(1) SPLT(30) LGTC(1) CWHD(0) }	60017
{ NOCS(1) SPLT(30) LGTC(1) }	60621
{ NOCS(1) SPLT(30) RDSC(1) CWHD(0) }	61883
{ NOCS(1) SPLT(30) RDSC(1) }	62528
{ NOCS(1) SPLT(30) }	80999
{ NOVC(2) CWHD(0) }	89450
{ NOVC(2) JNCC(4) CWHD(0) }	52072
{ NOVC(2) JNCC(4) }	52344
{ NOVC(2) LGTC(1) CWHD(0) }	68546
{ NOVC(2) LGTC(1) RDSC(1) CWHD(0) }	54595
{ NOVC(2) LGTC(1) RDSC(1) }	55036
{ NOVC(2) LGTC(1) }	69149
{ NOVC(2) NOCS(1) CWHD(0) }	67426
{ NOVC(2) NOCS(1) LGTC(1) CWHD(0) }	52136
{ NOVC(2) NOCS(1) LGTC(1) }	52551
{ NOVC(2) NOCS(1) RDSC(1) CWHD(0) }	51663
{ NOVC(2) NOCS(1) RDSC(1) }	52048
{ NOVC(2) NOCS(1) }	68019
{ NOVC(2) RDSC(1) CWHD(0) }	67323
{ NOVC(2) RDSC(1) }	67896
{ NOVC(2) SPLT(30) CWHD(0) }	60810
{ NOVC(2) SPLT(30) }	61212
{ RDSC(1) CWHD(0) }	108907
{ SPLT(30) CWHD(0) }	98082
{ SPLT(30) JNCC(4) CWHD(0) }	55608
{ SPLT(30) JNCC(4) }	56091
{ SPLT(30) LGTC(1) CWHD(0) }	73041
{ SPLT(30) LGTC(1) RDSC(1) CWHD(0) }	59463
{ SPLT(30) LGTC(1) RDSC(1) }	60054

{ SPLT(30) LGTC(1) }	73791
{ SPLT(30) RDSC(1) CWHD(0) }	75135
{ SPLT(30) RDSC(1) }	75947

9. RESULTS AND CONCLUSION

At the end it has been found that while the most of the accidents have occurred due to one of the few primary reasons for crashes in most of the situations, but some of the findings are really insightful and some areas need special care in terms of maintenance and infrastructure.
The critical findings like Road types and Light conditions are the areas which can be further explored using detailed data and there is a lot of scope for traffic safety improvement as well as saving government expenditure in terms of maintenance.

Conclusion:

The patterns generated after analysis are pretty accurate and insightful. Further room for improvement exists by adding more clusters to the distributed processing module, using software like Tableau and other commercial suits for more user friendly visualizations and more thorough data preprocessing to come up with more cost effective insights.

10. REFERENCES

1. http://hadoop.apache.org/docs/r2.6.4/hadoop-project-dist/hadoop-common/SingleCluster.html
2. http://www.tutorialspoint.com/data_mining/dm_overview.htm
3. https://en.wikipedia.org/wiki/Big_data
4. https://en.wikipedia.org/wiki/Apriori_algorithm
5. http://www.slideshare.net/INSOFE/apriori-algorithm-36054672
6. https://data.gov.uk/dataset/road-accidents-safety-data
7. https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html
8. http://hortonworks.com/hadoop/

11. BIBLIOGRAPHY

1. Data Communications and Networking By Behrouz A.Forouzan

2. Apoorv Mehta,Maitray Thaker,Shail Shah; Improving Road Traffic Safety by Minning Accident Data: Developing a Decision Tree to Classify Injury Severity Using R and Hadoop, May 2014

3. Kumar and Toshniwal *Journal of Big Data (2015) 2:26*
   DOI 10.1186/s40537-015-0035-y