TABLE OF CONTENTS

Introduction:

A dataset containing information about flight arrival and departure details for all commercial flights in the year 2008 is provided. The objective is to get the required information using Map-Reduce paradigm.

Some of the problems solved in this case are:

- How many flights flew on a weekday and what is the number of flights on each day
- The number of flights who flew long distances for more than 70 minutes.
- The reason for delay of a particular flight and delay time in minutes.

1.1 Literature Survey

Big data is a broad term for work with data sets so large or complex that traditional data processing applications are inadequate and distributed databases are needed. Challenges include sensor design, capture, data curation, search, sharing, storage, transfer, analysis, fusion, visualization, and information privacy.

MapReduce is a programming model and an associated implementation for processing and generating large data sets with a parallel, distributed algorithm on a cluster. Conceptually similar approaches have been very well known since 1995 with the Message Passing Interface standard having reduce and scatter operations.

A MapReduce program is composed of a Map() procedure (method) that performs filtering and sorting (such as sorting students by first name into queues, one queue for each name) and a Reduce() method that performs a summary operation (such as counting the number of students in each queue, yielding name frequencies). The "MapReduce System" (also called "infrastructure" or "framework") orchestrates the processing by marshalling the distributed servers, running the various tasks in parallel, managing all communications and data transfers between the various parts of the system, and providing for redundancy and fault tolerance.

The model is inspired by the map and reduce functions commonly used in functional programming although their purpose in the MapReduce framework is not the same as in their original forms. The key contributions of the MapReduce framework are not the actual map and reduce functions, but the scalability and fault-tolerance achieved for a variety of applications by optimizing the execution engine once. As such, a single-threaded implementation of MapReduce will usually not be faster than a traditional (non-MapReduce) implementation, any gains are usually only seen with multi-threaded implementations.

MapReduce libraries have been written in many programming languages, with different levels of optimization. A popular open-source implementation that has support for distributed shuffles is part of Apache Hadoop.

"Map" step: Each worker node applies the "map()" function to the local data, and writes the output to a temporary storage. A master node orchestrates that for redundant copies of input data, only one is processed.

"Shuffle" step: Worker nodes redistribute data based on the output keys (produced by the "map()" function), such that all data belonging to one key is located on the same worker node.

"Reduce" step: Worker nodes now process each group of output data, per key, in parallel.

MapReduce allows for distributed processing of the map and reduction operations. Provided that each mapping operation is independent of the others, all maps can be performed in parallel – though in practice this is limited by the number of independent data sources and/or the number of CPUs near each source. Similarly, a set of 'reducers' can perform the reduction phase, provided that all outputs of the map operation that share the same key are presented to the same reducer at the same time, or that the reduction function is associative. While this process can often appear inefficient compared to algorithms that are more sequential, MapReduce can be applied to significantly larger datasets than "commodity" servers can handle – a large server farm can use MapReduce to sort a petabyte of data in only a few hours.

The parallelism also offers some possibility of recovering from partial failure of servers or storage during the operation: if one mapper or reducer fails, the work can be rescheduled – assuming the input data is still available.

Another way to look at MapReduce is as a 5-step parallel and distributed computation:

Prepare the Map() input – the "MapReduce system" designates Map processors, assigns the input key value K1 that each processor would work on, and provides that processor with all the input data associated with that key value.

Run the user-provided Map() code – Map() is run exactly once for each K1 key value, generating output organized by key values K2.

"Shuffle" the Map output to the Reduce processors – the MapReduce system designates Reduce processors, assigns the K2 key value each processor should work on, and provides that processor with all the Map-generated data associated with that key value.

Run the user-provided Reduce() code – Reduce() is run exactly once for each K2key value produced by the Map step.

Produce the final output – the MapReduce system collects all the Reduce output, and sorts it by K2 to produce the final outcome.

These five steps can be logically thought of as running in sequence – each step starts only after the previous step is completed – although in practice they can be interleaved as long as the final result is not affected.

In many situations, the input data might already be distributed ("sharded") among many different servers, in which case step 1 could sometimes be greatly simplified by assigning Map servers that would process the locally present input data. Similarly, step 3 could sometimes be sped up by assigning Reduce processors that are as close as possible to the Map-generated data they need to process.

Logical View:

The Map and Reduce functions of MapReduce are both defined with respect to data structured in (key, value) pairs. Map takes one pair of data with a type in one data domain, and returns a list of pairs in a different domain:

Map(k1,v1) → list(k2,v2)

The Map function is applied in parallel to every pair in the input dataset. This produces a list of pairs for each call. After that, the MapReduce framework collects all pairs with the same key from all lists and groups them together, creating one group for each key.

The Reduce function is then applied in parallel to each group, which in turn produces a collection of values in the same domain:

Reduce(k2, list (v2)) → list(v3)

Each Reduce call typically produces either one value v3 or an empty return, though one call is allowed to return more than one value. The returns of all calls are collected as the desired result list.

Thus the MapReduce framework transforms a list of (key, value) pairs into a list of values. This behavior is different from the typical functional programming map and reduce combination, which accepts a list of arbitrary values and returns one single value that combines all the values returned by map.

It is necessary but not sufficient to have implementations of the map and reduce abstractions in order to implement MapReduce. Distributed implementations of MapReduce require a means of connecting the processes performing the Map and Reduce phases. This may be a distributed file system. Other options are possible, such as direct streaming from mappers to reducers, or for the mapping processors to serve up their results to reducers that query them.

SOFTWARE REQUIREMENTS:

Software's Required

1) VMware Player

2) Cloudera QuickStart VM

1) VMware Player:

VMware Player is virtualization software packages supplied free of charge by VMware, Inc. A company which was formerly a division of, and whose majority shareholder remains EMC Corporation . VMware Player can run existing virtual applications and create its own virtual machines. It uses same virtualization core as VMware Workstation, a similar program with more features, but not free of charge. VMware Player is available for personal non-commercial use, or for distribution or other use by written agreement.VMware, Inc.does not formally support Player, but there is an active community website for discussing and resolving issues, and a knowledge base.

2) Cloudera QuickStart VM:

ClouderaInc. Is an American-based software company that provides Apache Hadoop-based software, support and services, and training to business customers.

Cloudera's open-source Apache Hadoop distribution,CHD (Cloudera Distribution Including Apache Hadoop),targets enterprise-class deployments of that technology.Cloudera says that more than 50% of its engineering output is denoted upstream to the various Apache-licensed open source (Apache Hive, Apache Avro, Apache HBase, and so on) that combine to form Hadoop Platform. Cloudera is also a sponsor of the Apache Software Foundation.

The fastest way may be to just install a pre-configured virtual Hadoop environment. Two such environments are:

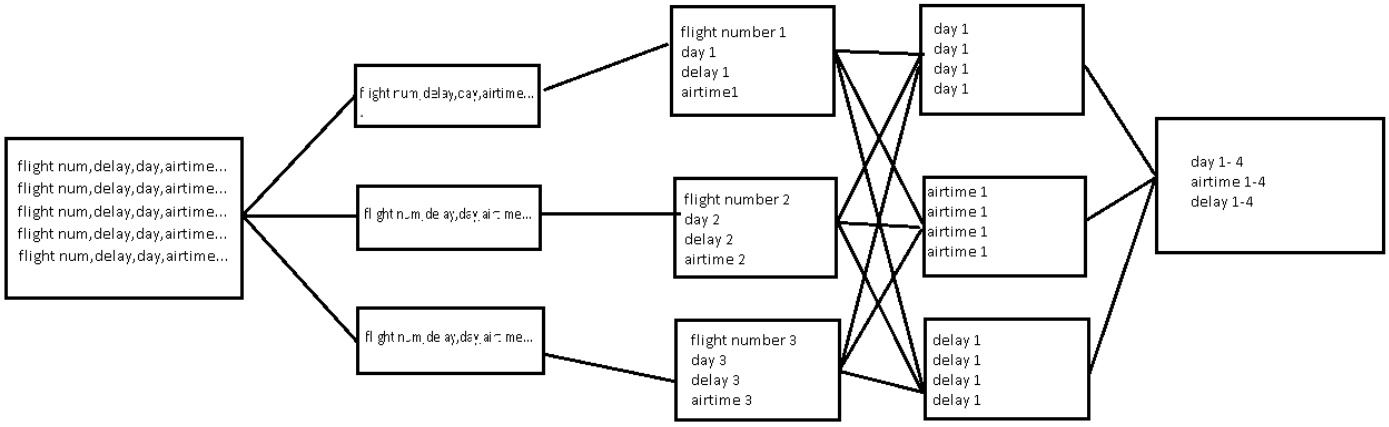
The Cloudera QuickStart Virtual machine. This image within the free VMware Player, VirtualBox, or KVM and has Hadoop, Hive, Pig and examples pre-loaded. Video lectures and screencasts walk you through everything.

The Hortonworks Sandbox. The Sandbox is a pre-configured virtual machine that comes with a dozen interactive Hadoop tutorials.

Cloudera also provides their distribution for Hadoop including support for Hive and Pig and configuration management for various operating systems.

We installed Cloudera QuickStart Virtual Machine for our project and worked Map-Reduce.

DESIGN:



Implementation:

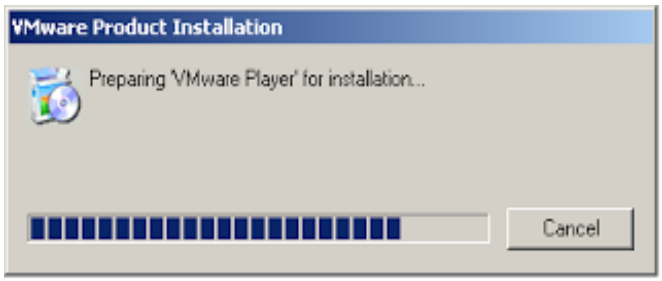
Map-Reduce must be Implemented in Linux based Environment for better results, as LINUX provides better performance for File-Based applications.The Implementation process is shown below.

Procedure for Installation:

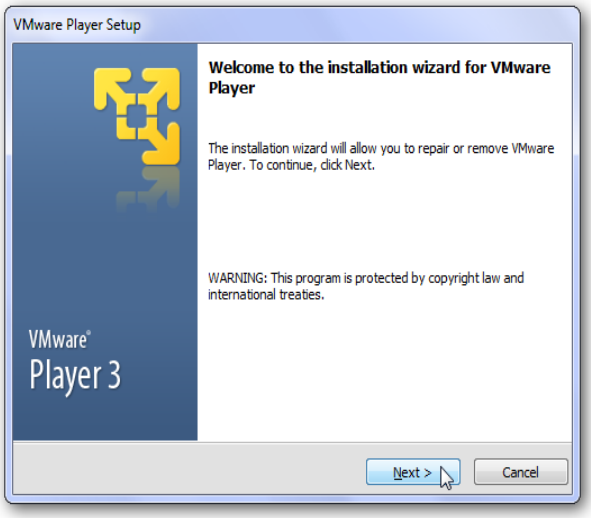
Download the latest version of VMware Player from the Official website. The link for Downloading process is https://my.vmware.com/web/vmware/details?downloadGroup=WKST-1112-_OSS&productId=407

Fig: VMware Application

Click on the installer and start installing the product. This gives the dialog box indicating the preparation of installation.



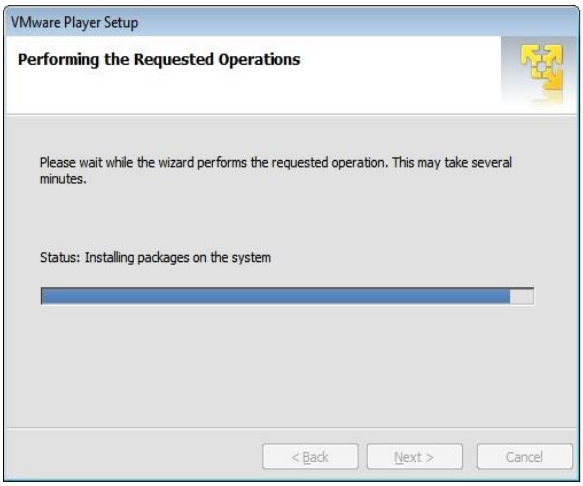
Installation of the product starts showing the wizard that the VMware player is started.



The VMware player starts configuring and the status of the progress of its installation is shown in this figure

Now the product is ready for installation. Thus the product can be installed by clicking the Install button.

After the Install button is started and the wizard showing the status of installation.

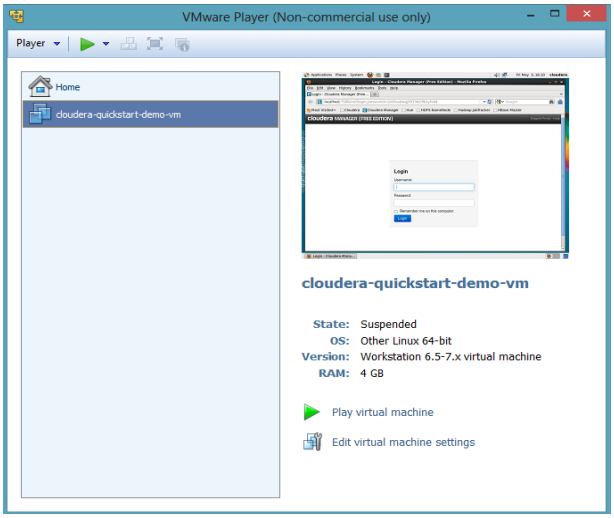


Finally if the installation is done, this wizard will show the success page and VMware is installed into the local machine.

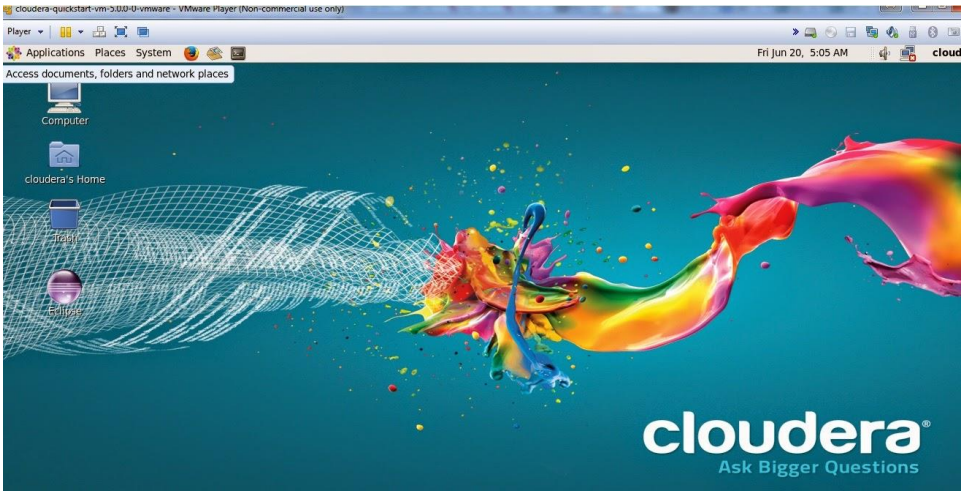
The installation of VMware Player is successful, now the Cloudera must be downloaded. Download the cloudera QuickStart VM from the site http://www.cloudera.com/content/www/en-us/downloads/quickstart_vms/5-4.html

The bundle must not be unzipped but it must be opened from the VMware Player using the browser option in the player. So, VMware must be started first using the installed icon and shortcut created.

Clicking on the VMware Player it opens a wizard as follows indicating that VMware Player is successfully installed.



The Cloudera is loaded is loaded into the home of VMware Player. Thus CentOS 6.2 is loaded displaying the virtual machine. Once the guest OS is loaded , and indicates that the installation is successful, Displaying the startpage of Cloudera.



Loading the dataset into File System:

Once the Cloudera is ready with the installation now we can proceed to execution, some basic commands in Hadoop are tested in the first phase.

We need to load the data set into the hdfs, this can be done using command

Hadoop fs -put '/home/cloudera/Downloads/2008.csv' doc.csv

After typing this command, if the prompt appears then it indicates that the data set is loaded into the hdfs. This can be checked by going into the hdfs name node in the browser and navigating through the file system.

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
hbase	dir				2014-06-02 09:23	rwxr-xr-x	hbase	hbase
solr	dir				2014-06-02 09:22	rwxr-xr-x	solr	solr
tmp	dir				2014-06-02 09:22	rwxrwxrwx	hdfs	supergroup
user	dir				2014-06-02 09:24	rwxr-xr-x	hdfs	supergroup
var	dir				2014-06-02 09:21	rwxr-xr-x	hdfs	supergroup

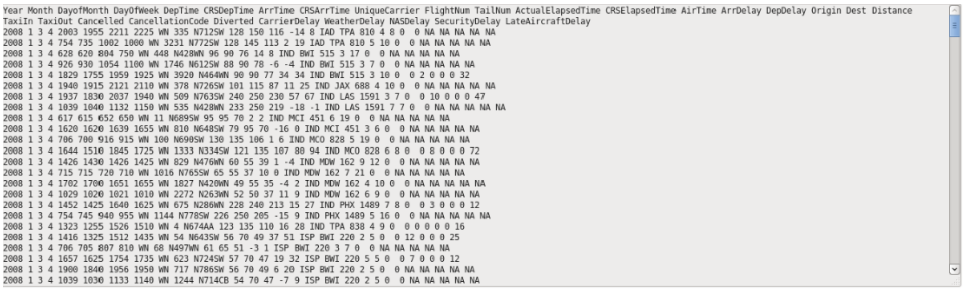
While browsing the file system we need to look at the location user/cloudera/doc.csv. So, we click on the user and navigate to the next screen in the browser.

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
cloudera	dir				2015-11-03 09:41	rwxr-xr-x	cloudera	cloudera
history	dir				2014-06-02 09:21	rwxr-xr-x	mapred	yarn
hive	dir				2014-06-02 09:23	rwxrwxrwx	hive	hive
oozie	dir				2014-06-02 09:23	rwxrwxr-x	oozie	oozie
sqoop2	dir				2014-06-02 09:24	rwxrwxr-x	sqoop2	sqoop

Here we select the Cloudera options and check for the doc.csv dataset which is loaded onto the hdfs.

delay2output	dir				2015-10-02 22:40	rwxr-xr-x	cloudera	cloudera
delay5.csv	file	211.07 MB	3	128 MB	2015-10-03 00:36	rw-r--r--	cloudera	cloudera
delay5output	dir				2015-10-03 00:41	rwxr-xr-x	cloudera	cloudera
doc.csv	file	211.07 MB	3	128 MB	2015-11-03 09:41	rw-r--r--	cloudera	cloudera
examplecounter.txt	file	169 B	3	128 MB	2015-09-29 09:03	rw-r--r--	cloudera	cloudera
exampleoutput	dir				2015-09-29 09:16	rwxr-xr-x	cloudera	cloudera
file.txt	file	152 B	3	128 MB	2015-09-25 04:01	rw-r--r--	cloudera	cloudera
file1.txt	file	152 B	3	128 MB	2015-09-25 04:02	rw-r--r--	cloudera	cloudera

Thus the dataset doc.csv is available, from the local machine the dataset is mounted onto the hdfs file system. Any analysis can be applied on the dataset that is available on the hdfs file system.



```
Year Month DayofMonth DayOfWeek DepTime CRSDepTime ArrTime CRSArrTime UniqueCarrier FlightNum TailNum ActualElapsedTime CRSElapsedTime AirTime ArrDelay DepDelay Origin Dest Distance
TaxiIn TaxiOut Cancelled CancellationCode Diverted CarrierDelay WeatherDelay NASDelay SecurityDelay LateAircraftDelay
2008 1 3 4 2003 1955 2211 2225 WN 335 N712SW 128 150 116 -14 8 IAD TPA 810 4 8 0  0 NA NA NA NA NA
2008 1 3 4 754 735 1002 1000 WN 3231 N772SW 128 145 113 2 19 IAD TPA 810 5 10 0  0 NA NA NA NA NA
2008 1 3 4 628 620 804 750 WN 448 N428WN 96 90 76 14 8 IND BWI 515 3 17 0  0 NA NA NA NA NA
2008 1 3 4 926 930 1054 1100 WN 1746 N612SW 88 90 78 -6 -4 IND BWI 515 3 7 0  0 NA NA NA NA NA
2008 1 3 4 1829 1755 1959 1925 WN 3920 N464WN 90 90 77 34 34 IND BWI 515 3 10 0  0 2 0 0 0 32
2008 1 3 4 1940 1915 2121 2110 WN 378 N726SW 101 115 87 11 25 IND JAX 688 4 10 0  0 NA NA NA NA NA
2008 1 3 4 1937 1830 2037 1940 WN 509 N763SW 240 250 230 57 67 IND LAS 1591 3 7 0  0 10 0 0 0 47
2008 1 3 4 1039 1040 1132 1150 WN 535 N428WN 233 250 219 -18 -1 IND LAS 1591 7 7 0  0 NA NA NA NA NA
2008 1 3 4 617 615 652 650 WN 11 N689SW 95 95 70 2 2 IND MCI 451 6 19 0  0 NA NA NA NA NA
2008 1 3 4 1620 1620 1639 1655 WN 810 N648SW 79 95 70 -16 0 IND MCI 451 3 6 0  0 NA NA NA NA NA
2008 1 3 4 706 700 916 915 WN 100 N690SW 130 135 106 1 6 IND MCO 828 5 19 0  0 NA NA NA NA NA
2008 1 3 4 1644 1510 1845 1725 WN 1333 N334SW 121 135 107 80 94 IND MCO 828 6 8 0  0 8 0 0 0 72
2008 1 3 4 1426 1430 1426 1425 WN 829 N476WN 60 55 39 1 -4 IND MDW 162 9 12 0  0 NA NA NA NA NA
2008 1 3 4 715 715 720 710 WN 1016 N765SW 65 55 37 10 0 IND MDW 162 7 21 0  0 NA NA NA NA NA
2008 1 3 4 1702 1700 1651 1655 WN 1827 N420WN 49 55 35 -4 2 IND MDW 162 4 10 0  0 NA NA NA NA NA
2008 1 3 4 1029 1020 1021 1010 WN 2272 N263WN 52 50 37 11 9 IND MDW 162 6 9 0  0 NA NA NA NA NA
2008 1 3 4 1452 1425 1640 1625 WN 675 N286WN 228 240 213 15 27 IND PHX 1489 7 8 0  0 3 0 0 0 12
2008 1 3 4 754 745 940 955 WN 1144 N778SW 226 250 205 -15 9 IND PHX 1489 5 16 0  0 NA NA NA NA NA
2008 1 3 4 1323 1255 1526 1510 WN 4 N674AA 123 135 110 16 28 IND TPA 838 4 9 0  0 0 0 0 0 16
2008 1 3 4 1416 1325 1512 1435 WN 54 N643SW 56 70 49 37 51 ISP BWI 220 2 5 0  0 12 0 0 0 25
2008 1 3 4 706 705 807 810 WN 68 N497WN 61 65 51 -3 1 ISP BWI 220 3 7 0  0 NA NA NA NA NA
2008 1 3 4 1657 1625 1754 1735 WN 623 N724SW 57 70 47 19 32 ISP BWI 220 5 5 0  0 7 0 0 0 12
2008 1 3 4 1900 1840 1956 1950 WN 717 N786SW 56 70 49 6 20 ISP BWI 220 2 5 0  0 NA NA NA NA NA
2008 1 3 4 1039 1030 1133 1140 WN 1244 N714CB 54 70 47 -7 9 ISP BWI 220 2 5 0  0 NA NA NA NA NA
```

We can view the dataset in the hdfs available in this form. If there are any null values then the data need to be refined removing all the null values.

BASIC CODE OF THE PROJECT:

//Main.java

```
import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.io.LongWritable;

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 DayMain extends Configured implements Tool {
@Override
public int run(String[] args) throws Exception {
if(args.length<2){
System.out.println("error argument length");
 return -1;
}
JobConf conf=new JobConf(DayMain.class);
FileInputFormat.setInputPaths(conf,new Path(args[0]));
FileOutputFormat.setOutputPath(conf,new Path(args[1]));
conf.setMapperClass(DayMapper.class);
conf.setReducerClass(Reducer.class);
conf.setMapOutputKeyClass(Text.class);
conf.setMapOutputValueClass(LongWritable.class);
conf.setOutputKeyClass(Text.class);
conf.setOutputValueClass(LongWritable.class);
JobClient.runJob(conf);
return 0;
}
public static void main(String args[]) throws Exception{
int exitCode=ToolRunner.run(new DayMain(), args);
System.exit(exitCode);
}
```

```
}
```

//Mapper.java

```
import java.io.IOException;
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 DayMapper extends MapReduceBase implements Mapper<LongWritable, Text, Text, LongWritable> {
@Override
public void map(LongWritable key, Text value,
OutputCollector<Text, LongWritable> output, Reporter r)
throws IOException {
long ac=0,num=0;
String s=value.toString();
String Sunday=new String("sunday");
String Monday=new String("monday");
String Tuesday=new String("tuesday");
String Wednesday=new String("wednesday");
String Thursday=new String("thursday");
```

```
String Friday=new String("friday");

String Saturday=new String("saturday");

for(String word:s.split(" ")){

                ac=ac+1;

                if(ac==4) //retrieving the weekday field

                {

               try

               {


               num=Long.parseLong(word);
                 }
                catch(Exception e)
               {}
if(num==1)//first day of the week
               {
output.collect(new Text(Sunday),new LongWritable(1));
}
    if(num==2)//second day of the week
              {
output.collect(new Text(Monday),new LongWritable(1));
}

                if(num==3)// third day of the week
                {
             output.collect(new Text(Tuesday),new LongWritable(1));
}


                if(num==4) //fourth day of the week
                {
    output.collect(new Text(Wednesday),new LongWritable(1));
}
                if(num==5)// fifth day of the week
                {
```

```
            output.collect(new Text(Thursday),new LongWritable(1));
}
              if(num==6)// sixth day of the week
              {
            output.collect(new Text(Friday),new LongWritable(1));
}
              if(num==7)// seventh day of the week
              {
            output.collect(new Text(Saturday),new LongWritable(1));
              }
              }
 if(ac==14)//airtime greater than 70min
          {
 try
              {
                  num=Integer.parseInt(word);
               }
              catch(Exception e)
              {}
              if(num>70)
              {
             System.out.println(ac);
output.collect(new Text(airtime),new IntWritable(1));
}


 if(ac==25)// calculating the different delays
              {
             try
             {
                if(!word.equals("NA"))
             num=Integer.parseInt(word);
              }
              catch(Exception e)
              {}


output.collect(new Text(delay1),new IntWritable(num));//delay due to engine failure
```

```
}


                if(ac==26)// delay due to bad weather
                {
               try
               {
                  if(!word.equals("NA"))
               num=Integer.parseInt(word);
                }
                catch(Exception e)
                {}


output.collect(new Text(delay2),new IntWritable(num));
}
            if(ac==26)//delay due to maintenance
                {
               try
               {
                  if(!word.equals("NA"))
               num=Integer.parseInt(word);
                }
                catch(Exception e)
                {}
 output.collect(new Text(delay3),new IntWritable(num))
}
               if(ac==28)//delay 4
                {
               try
               {
                  if(!word.equals("NA"))
               num=Integer.parseInt(word);
                }
                catch(Exception e)
                {}


output.collect(new Text(delay4),new IntWritable(num));
```

```
}
                if(ac==29)delay 5
                {
               try
               {
                  if(!word.equals("NA"))
               num=Integer.parseInt(word);
                 }
                catch(Exception e)
                {}


output.collect(new Text(delay5),new IntWritable(num));
}


}
}
}
```

//Reducer.java

```
import java.io.IOException;

import java.util.Iterator;

import org.apache.hadoop.io.LongWritable;

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 Reducer extends MapReduceBase implements
Reducer<Text,LongWritable,Text,LongWritable> {

@Override
```

```
public void reduce(Text key, Iterator<LongWritable> values,

OutputCollector<Text, LongWritable> output, Reporter r)

throws IOException {

long count=0;

while(values.hasNext())

{

LongWritable i=values.next();

count+=i.get();

}

output.collect(key, new LongWritable(count));

}

}
```

Description of the classes:

There are totally three classes to our code they are:

1.Main class

2.Mapper class

3.Reducer class

1.Main class:

```
JobConf conf=new JobConf(DayMain.class);
```

The above line will create configuration object “conf” and it will provide access to configuration parameters. Configuration parameters are defined in the form of xml data and by default Hadoop specifies two configuration files.

core-default.xml : Read-only defaults for hadoop.

core-site.xml : Site-specific configuration for a given hadoop installation.

All the initialization are made in this class,we have to set the Mapper class and Reducer class using the command conf.setMapperClass(mapper.class) and conf.setReducerClass(reducer.class).

To run the project this is the main class which has to be executed as it contains all the links to the mapper class and reducer class

2.Mapper Class:

This is the class where mapping takes place,here we segregate the data and the output is sent to the reducer for further processing. Here the class takes four parameters the first parameter is LongWritable which is similar to the java long data type this LongWritable indicate the line number,the second parameter is of type class Text which is similar to the string type in java but provides more efficient functionality and this represents the entire line because this class takes each record as input and this record can be caught using this Text class ,now the third parameter is regarded to the output of the mapper class output of the mapper is given as a word and its count,so the third parameter is a Text class object because it is usually a word and the fourth and the final parameter in of LongWritable class as it is mostly count of the word that is sent into the reducer class

Now in the mapper class we have to catch the field and the field we require for getting the count of the weekday is field number four so in the mapper we use the split() command and maintain the count of the split taken place,whenever there is a count of 4 recorded in the counter we go into the if clause where the number in integer format is converted and it is sent into the output class using the statement output.collect(new Text(x),new LongWritable(1));

Here the x represents the string if the number caught in the field number 4 is one 1 then it is Sunday if the number is 2 then it is Monday if it is 3 then Wednesday and so on up to Saturday.

The other functioning is about finding the occurrence of delays there are five types of delays and they are located in the field 25,26,27,28 and 29 respectively.Here to compute this data we have to make use of the counter,whenever we enter into the split() command the counter is incremented and when the counter value is 25,26,27,28,29 respectively the delays are catched and the count is sent in the output.collect() method which is later processed by the reducer.

The third functioning is to know the number of flights whose airtime is greater than 70 minutes, in order to do so we have to catch the fourteenth field as the data in that field is the airtime of the respective flight so we maintain a counter again and when the value of the counter is 14 we enter into the loop where we check the condition, whether the airtime is greater than 70 minutes or not if the condition is satisfied then we send the Text class object as airtime and we send the count into the output.collect() method which sends this information to the reducer class and the reducer class will do the further processing.

3.Reducer class:

This is the class which take the input from the Mapper class and processes the information and provides the output accordingly. It has four parameter the first two parameters represent the input and the second two parameters represent the output . The first parameter is the Text class which is the object returned from the mapper class, second parameter is to the iterator class which is taken from the mapper class as the mapper class produces a series of number,third parameter is related to the output which is of the class Text and the fourth parameter is of the class LongWritable as it is the count so the output we get is in the form of a String and its count

In the first case where we calculate the frequency of each weekday,reducer receive the name of the weekday and the iterator class contains 1's as many number of times that weekday as repeated so he add all the ones present in the iterator and the count of each weekday is displayed by output.collect()

The second function where we find the occurrence of the delays, reducer receives the same kind of the input as the above one but here the repetition is as per the number of the delays and using the iterator we add all the 1's present and get the occurrence of every delay.

The third function we display the number of flights whose airtime if greater than 70 min, here the reducer receives the text airtime and the iterator consisting of all the 1's as the input so we add all the 1's present in the iterator and retrieve the count of the flights whose airtime is greater than 70 minutes.

Execution:

For executing project in Cloudera, sequence of steps are to be followed.Open the telnet and type the following commands:

1) Load the comma separated value file onto the Hadoop distributed file system by executing the following command:

hadoop fs -put 2008.csv doc.csv

2) Press the long listing command to check whether the dataset is loaded onto hdfs. This can be done by executing the following com vmand.

hadoop fs -ls

3) Now go to the workspace directory where further execution takes place.

cd workspace

4) Now enter the following command to start the mapper and reducer classes.Here we need to specify both the input and output files.

hadoop jar today.jar DayMain doc.csv docoutput

5) Now open browser and traverse to "Jobtracker" tab to check the percentage of job completed. After successful completion the percentage turns to 100%.

6) Now go to "name node" tab and press "browse the file system" tab. On clicking the tab it displays the directory content. Now go to user/Cloudera/ where the output file is present.

user/cloudera/docoutput/part-00000

Testing:

Testing is performed first on wordcount problem by taking a file as input which contains some text like

"apple mango banana chocolate pine

cocoa apple mango pine pine

chocolate cocoa apple mango

mango mango apple apple mango …"

then this file is processed using mapper class, shuffle and sort classes and finally through reducer class and the output is stored in another file 'wordoutput.txt'. The output we get is the number of apples,bananas,chocos etc. in the text file.

delay2output	dir				2015-10-02 22:40	rwxr-xr-x	cloudera	cloudera
delay5.csv	file	211.07 MB	3	128 MB	2015-10-03 00:36	rw-r--r--	cloudera	cloudera
delay5output	dir				2015-10-03 00:41	rwxr-xr-x	cloudera	cloudera
doc.csv	file	211.07 MB	3	128 MB	2015-11-03 09:41	rw-r--r--	cloudera	cloudera
examplecounter.txt	file	169 B	3	128 MB	2015-09-29 09:03	rw-r--r--	cloudera	cloudera
exampleoutput	dir				2015-09-29 09:16	rwxr-xr-x	cloudera	cloudera
file.txt	file	152 B	3	128 MB	2015-09-25 04:01	rw-r--r--	cloudera	cloudera
file1.txt	file	152 B	3	128 MB	2015-09-25 04:02	rw-r--r--	cloudera	cloudera

In this way the output files are stored in the directory which is in hadoop namenode (which can be accessed through localhost in the brower) with all the access permissions ,size of the file,type of the file etc.

Further Enhancements:

The project of ours if executed on a single node environment but the real use of Hadoop is seen only when the Hadoop is used in a distributed system as the data present in all the nodes which are working in the distributed system is loaded into the hdfs(Hadoop distributed file system).

As the amount of data generated by businesses is increasing at a vast rate, the opportunities around it are exploding in terms of both scale and variety, so It is quite evident that Hadoop and other big data technologies have lot of scope in the coming times

Bibliography:

1. Hadoop: The Definitive Guide, 3rd Edition: http://shop.oreilly.com/product/0636920021773.do
2. http://static.googleusercontent.com/media/research.google.com/es/us/archive/mapreduce-osdi04.pdf
3. MapReduce functions and classes:https://en.wikipedia.org/wiki/MapReduce
4. BigData: https://en.wikipedia.org/wiki/Big_data
5. BigData :http://www.forbes.com/sites/lisaarthur/2013/08/15/what-is-big-data/
6. BigData: http://www.ibm.com/big-data/us/en/

7. www/en-us/documentation/other/tutorial/CDH5/Hadoop-Tutorial/CDH5/Hadoop-Tutorial/ht_overview/

8. Basics of Hadoop: http://www.hadooplessons.info/2013/06/data-sets-for-practicing-hadoop.html

9. Airline dataset details: http://www.stat.purdue.edu/~sguha/rhipe/doc/html/airline.html