

**COMPUTER SCIENCE AND ENGINEERING DEPARTMENT**

**B.E. 7th SEM**

**BIG DATA ANALYTICS**

**NAME:. JAYSHREE ACHARYA. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .**

**ENROLLMENT NO.:. 181080131001. . . BATCH NO.:. . . . . . .**

**BRANCH: Computer Science. . . . . . . . . . . . . . . . . . YEAR:. 2021-22 . . .**

**AMIRAJ COLLEGE OF ENGINEERING AND TECHNOLOGY**

**Nr. Tata Nano Plant, Khoraj, Sanand, Ahmedabad.**



This is to certify that ~~Mr.~~ /Ms. Jayshree Acharya Enrollment No. 181080131001 Student of B.E. 7 Semester Computer Science branch has successfully completed the work in the subject of Big Data Analytics.

The work done is done satisfactorily within the four walls of the institute.

Date of Submission: 06/10/2021

Faculty in Charge Dept. Coordinator

**AMIRAJ COLLEGE OF ENGINEERING AND TECHNOLOGY**

**NR. TATA NANO PLANT, KHORAJ, SANAND, AHMEDABAD.**

**AMIRAJ COLLEGE OF ENGINEERING AND TECHNOLOGY**

**Nr. Tata Nano Plant, Khoraj, Sanand, Ahmedabad.**

|  |  |
| --- | --- |
|  | **AMIRAJ COLLEGE OF ENGINEERING AND TECHNOLOGY**  **DEPARTMENT : Computer Science and Engineering**  **SUBJECT : BDA SEM : 7th** |

**INDEX**

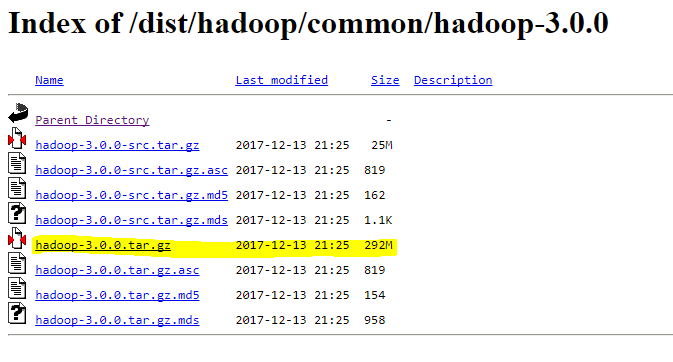
|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO.** | **Experiment** | **Submission Date** | **Signature** |
| 1. | To install Hadoop 3.X on Ubuntu. |  |  |
| 2. | To install MongoDB and perform various commands in MongoDB. |  |  |
| 3. | Implement following using Map- Reduce.   1. Matrix multiplication 2. Sorting 3. indexing |  |  |
| 4. | To load a local file on to HDFS from local file system. |  |  |
| 5. | To install Cassandra and perform CRUD operations on Cassandra. |  |  |
| 6. | Distributed Cache & Map Side Join, Reduce side Join Building and Running a Spark Application Word count in Hadoop and Spark Manipulating RDD. |  |  |
| 7. | Implementation of commands in Spark Sql programming. |  |  |
| 8. | Implementing K-Means Clustering algorithm using Map-Reduce. |  |  |
| 9. | Implementing any one Frequent Item set algorithm using Map-Reduce. |  |  |
| 10. | Create A Data Pipeline Based On Messaging Using PySpark And Hive. |  |  |

**PRACTICAL – 1**

**AIM: To install Hadoop 3.X on Ubuntu.**

**Step-1 Download Hadoop 3.0 and Install**

Visit Apache Hadoop 3.0 download site & download the tar file

following command will help to download and install (change the folder as needed for your execution)

cd /tmp

wget http://www-us.apache.org/dist/hadoop/common/hadoop-3.0.0/hadoop-3.0.0.tar.gz

tar -xzf /tmp/hadoop-3.0.0.tar.gz /home/toppertips/hadoop3.x

**Step – 2 Hadoop 3.0 Configuration**

Once we have extracted the hadoop 3.0 bundle, it is time to configure some of the path in **.bashrc** file, so open the **.barhrc** file in user’s home directory and add following parameters

export HADOOP\_PREFIX="/home/toppertips/hadoop3.x"

export PATH=$PATH:$HADOOP\_PREFIX/bin

export PATH=$PATH:$HADOOP\_PREFIX/sbin

export HADOOP\_MAPRED\_HOME=${HADOOP\_PREFIX}

export HADOOP\_COMMON\_HOME=${HADOOP\_PREFIX}

export HADOOP\_HDFS\_HOME=${HADOOP\_PREFIX}

export YARN\_HOME=${HADOOP\_PREFIX}

#after adding above line, run the following command

source .bashrc

**Step – 3 Configure hadoop-env.sh**

Update the configuration file called hadoop-env.sh (located in HADOOP\_HOME/etc/hadoop) and set JAVA\_HOME:

export JAVA\_HOME=/usr/lib/jvm/java-8/

**Step – 4 Configure core-site.xml, hdfs-site.xml & mapred-site.xml**

Edit configuration file ***core-site.xml,***located in ***HADOOP\_HOME/etc/hadoop*** and add following entries:

<configuration>

<property>

<name>fs.defaultFS</name>

<value>hdfs://localhost:9000</value>

</property>

<property>

<name>hadoop.tmp.dir</name>

<value>/home/toppertips/hdata</value>

</property>

</configuration>

Now open ***hdfs-site.xml*** file located in ***HADOOP\_HOME/etc/hadoop*** and add the following entries

<configuration>

<property>

<name>dfs.replication</name>

<value>3</value>

</property>

</configuration>

Now edit mapred-site.xml and if it is missing then copy it using the template available there

cp mapred-site.xml.template mapred-site.xml

Edit following configuration in the file

<configuration>

<property>

<name>mapreduce.framework.name</name>

<value>yarn</value>

</property>

</configuration>

**Step – 5 Configure Yarn xml file**

mapred-site.xml file is located in your HADOOP\_HOME/etc/hadoop location, open it and add/edit following part as shown below

<configuration>

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce\_shuffle</value>

</property>

<property>

<name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>

<value>org.apache.hadoop.mapred.ShuffleHandler</value>

</property>

</configuration>

Now you are all set to start your Hadoop 3.0 installation services.

**Step – 6 Start Hadoop 3.0 Installation Services**

***NOTE****: This command execution will only be done on your first hadoop installation. Do not perform this on existing hadoop installation, else it will permanently erase all your data from HDFS file system.*

The first step to starting up your Hadoop 3.0 installation is formatting the Hadoop filesystem followed by hdfs service as shown below

#only perform after fresh and first installation

bin/hdfs namenode -format

#Now start the hdfs service using dfs command

sbin/start-dfs.sh

#it may give an error at the time of startup and then use following command

echo "ssh" | sudo tee /etc/pdsh/rcmd\_default

#now start the yarn services

sbin/start-yarn.sh

#Once it is started successfully, check how many daemons are running

jps

2961 ResourceManager

2482 DataNode

3077 NodeManager

2366 NameNode

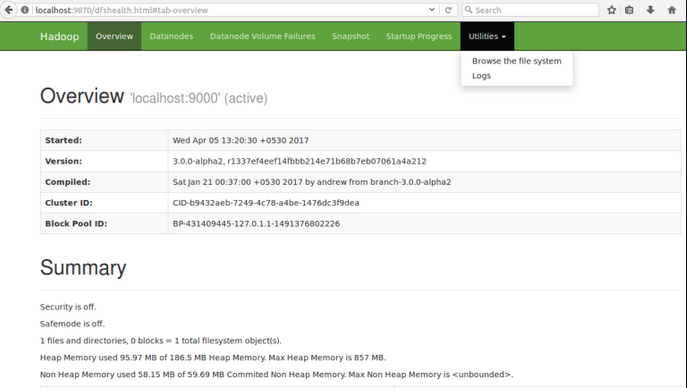
2686 SecondaryNameNode

3199 Jps

Congratulation, your installation is completed and you are ready to run MapReduce programs.

**Step – 7 Hadoop 3.0 Installation Health Check**

Previous version of Hadoop 2.x, web UI port is 50070 and it has been moved to 9870 in Hadoop 3.0. It can be accessed via web UI from **localhost:9870**



**PRACTICAL-2**

**AIM: To install MongoDB and perform various commands in MongoDB.**

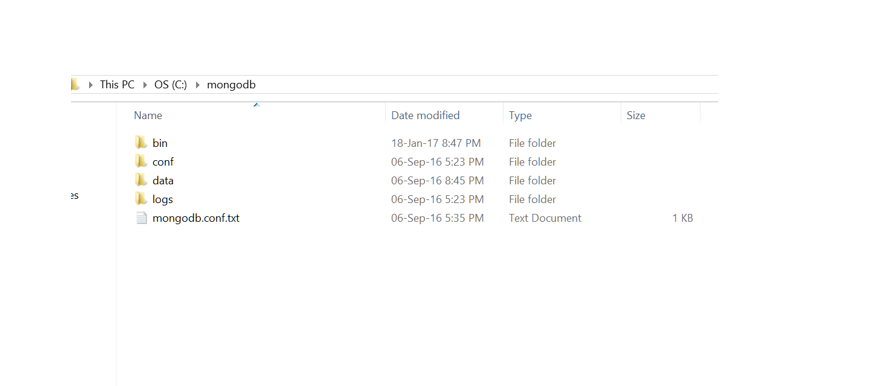
The following steps are required to install MongoDB on Windows:

**Step 1:** Download the latest version of MongoDB

**Step 2**: Extract the zip file.

**Step 3:** Set up the MongoDB environment.

* Create a folder with the name MongoDB within this path: C:\mongodb
* Have a list of four folders along with the one for the property file as shown in the following screenshot.



* bin folder contains all your Mongo executable files
* conf file is just the configuration file
* We use the data folder for the purpose of storing data
* Create one more folder called **db** inside it
* The logs folder is used for checking mongo related errors

**Paste the following link in mongodb.conf.txt**

# mongodb.conf

# Data

dbpath=c:\mongodb\data\db

# Log

logpath=c:\mongodb\logs\mongodb.log

logappend=true

# Only run on localhost for development

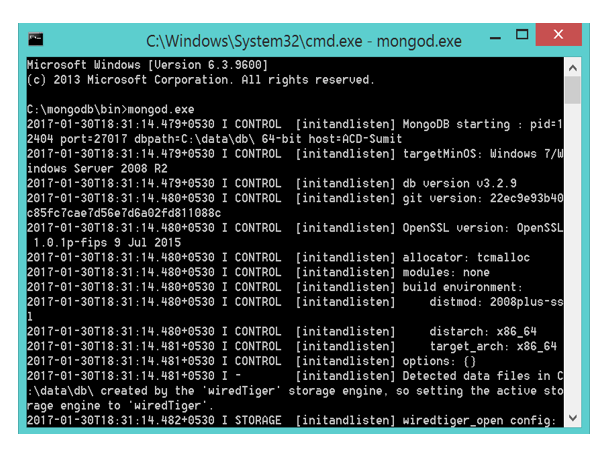
bind\_ip=127.0.0.1

# Default MongoDB port

port=27017

Save the file after copy pasting it. With this, we are done with the installation.

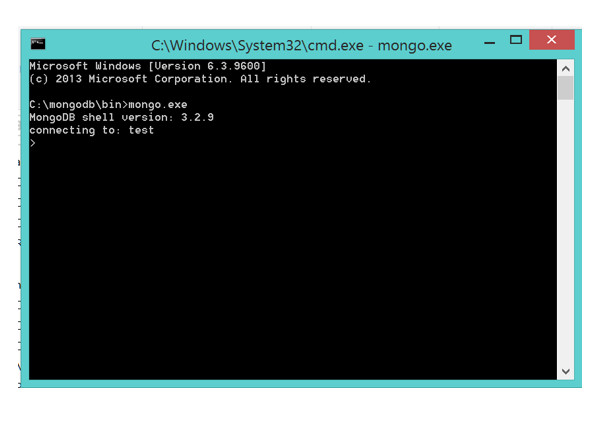
Let us test the Mongo environment next.

* Open the command prompt from C:\mongodb\bin
* Type mongod.exe

**Check the following command to ensure successful installation:**

* Open one more command prompt from the same path:

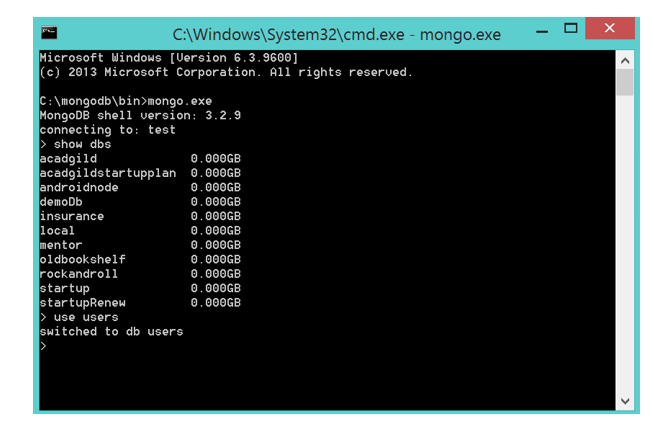
“C:\mongodb\bin”

* Type: mongo.exe
* As we type this command, it shows the following screenshot:

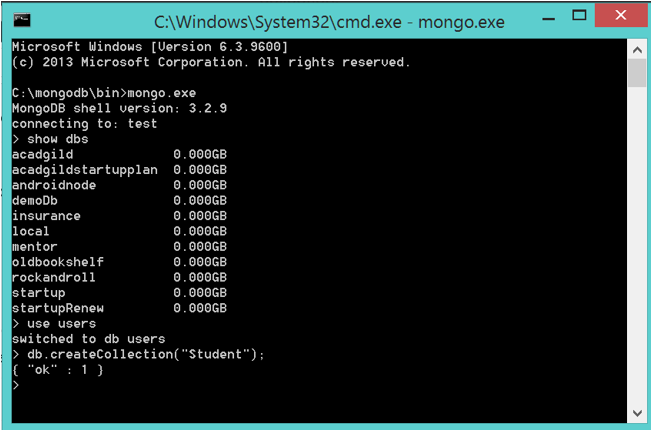
### CRUD Operations Using MongoDB

Follow the following steps one at a time:

**Step 1**: Create the database by typing the following commands (refer to the screenshot below):



**Step 2**: Create the collections (i.e., Tables in SQL)

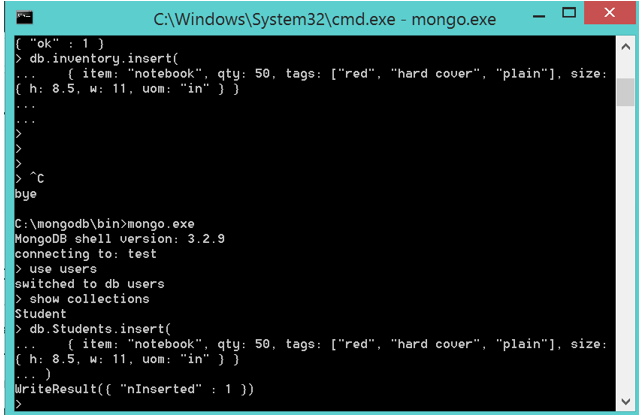


**Step 3**: Insert some records using the following commands:

db.Students.insert(

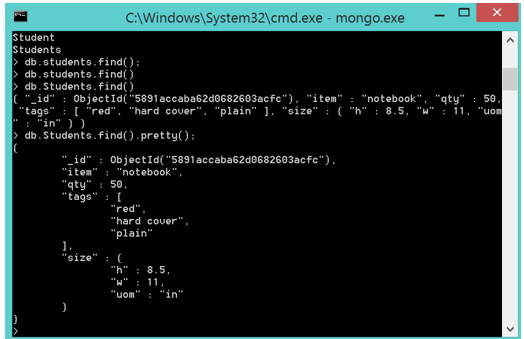
{ item: “notebook”, qty: 50, tags: [“red”, “hard cover”, “plain”], size: { h: 8.5, w: 11, uom: “in” } }

)



**Step 4**: Fetch records for the inserted values in the table (**Step 3**)

 db.Students.find().pretty();



**PRACTICAL-3**

**AIM: Implement following using Map- Reduce.**

1. **Matrix multiplication**
2. **Sorting**
3. **indexing**

**PROGRAM:**

**Mapper.py**

#!/usr/bin/env python

import sys

for line in sys.stdin:

matrix\_index, row, col, value = line.rstrip().split(",")

if matrix\_index == "A":

for i in range(5):

key = row + "," + str(i)

print "%s\t%s\t%s"%(key,col,value)

else:

for j in range(5):

key = str(j) + "," + col

print "%s\t%s\t%s"%(key,row,value)

**reducer.py**

#! / usr / bin / env python

import sys

from operator import itemgetter

prev\_index = None

value\_list = []

for line in sys.stdin:

curr\_index, index, value = line.rstrip (). split ("\ t")

index, value = map (int, [index, value])

if curr\_index == prev\_index:

value\_list.append ((index, value))

else:

if prev\_index:

value\_list = sorted (value\_list, key = itemgetter (0))

i = 0

result = 0

while i <len (value\_list) - 1:

if value\_list [i] [0] == value\_list [i + 1] [0]:

result + = value\_list [i] [1] \* value\_list [i + 1] [1]

i + = 2

else:

i + = 1

print "% s,% s"% (prev\_index, str (result))

prev\_index = curr\_index

value\_list = [(index, value)]

if curr\_index == prev\_index:

value\_list = sorted (value\_list, key = itemgetter (0))

i = 0

result = 0

while i <len (value\_list) - 1:

if value\_list [i] [0] == value\_list [i + 1] [0]:

result + = value\_list [i] [1] \* value\_list [i + 1] [1]

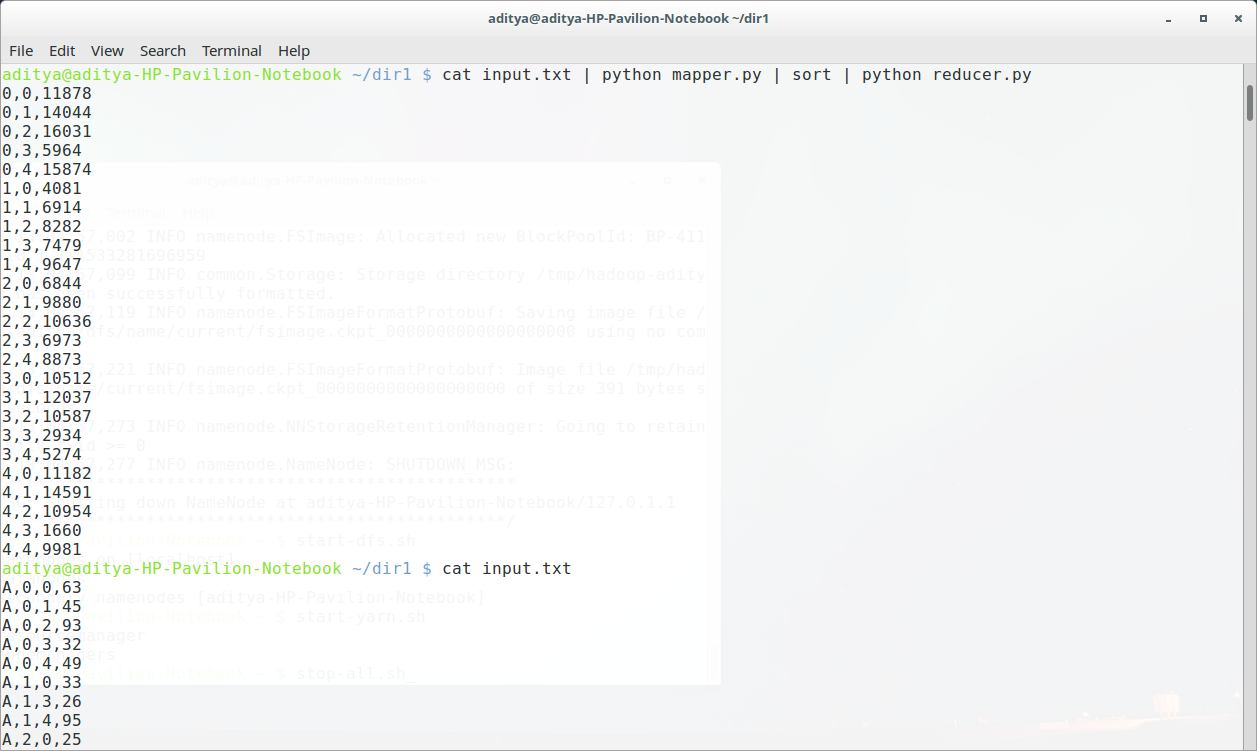
i + = 2

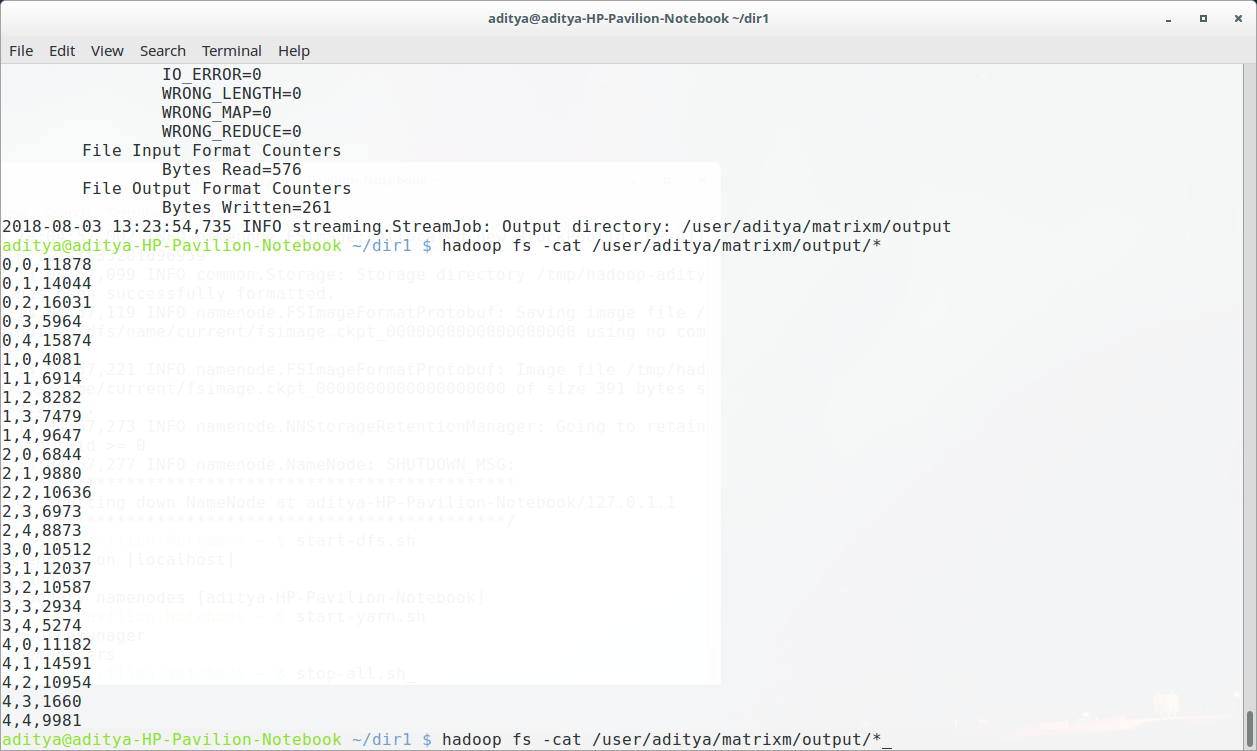
else:

i + = 1

print "% s,% s"% (prev\_index, str (result))

**OUTPUT:**





**PRACTICAL-4**

**AIM:** **To load a local file on to HDFS from local file system.**

**Download the Drivers Related Datasets**

1.Open a terminal on your local machine, SSH into the sandbox:

ssh root@sandbox-hdp.hortonworks.com -p 2222

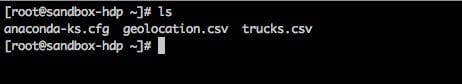
2.Copy and paste the commands to download the **geolocation.csv** and **trucks.csv** files. We will use them while we learn file management operations.

#Download geolocation.csv

wget https://github.com/hortonworks/data-tutorials/raw/master/tutorials/hdp/manage-files-on-hdfs-via-cli-ambari-files-view/assets/drivers-datasets/geolocation.csv

#Download trucks.csv

wget https://github.com/hortonworks/data-tutorials/raw/master/tutorials/hdp/manage-files-on-hdfs-via-cli-ambari-files-view/assets/drivers-datasets/trucks.csv



**Create a Directory in HDFS, Upload a File and List Contents**

**1 Login under hdfs user, so we can give root user permission to perform file operations:**

#Login under hdfs user

su hdfs

cd

**2. We will use the following command to run filesystem commands on the file system of Hadoop:**

hdfs dfs [command\_operation]

hdfs dfs -chmod 777 /user

**The command mkdir takes the path URI’s as an argument and creates a directory or multiple directories.**

#Syntax to create directory in HDFS

hdfs dfs -mkdir <paths>

**1. Let’s create the directory for the driver dataset by entering the following commands into your terminal:**

#Creates a directory called hadoop under users

hdfs dfs -mkdir /user/hadoop

#Creates two directories geolocation.csv and trucks.csv under the directory hadoop

hdfs dfs -mkdir /user/hadoop/geolocation /user/hadoop/trucks

hdfs dfs -put:

**The command put copies single src file or multiple src files from local file system to the Hadoop Distributed File System.**

#Syntax to copy file(s) from local to HDFS

hdfs dfs -put <local-src> ... <HDFS\_dest\_path>

**1 Copy both source files from your local file system to the Hadoop Distributed File System by entering the following commands into your terminal:**

#Copy the geolocation.csv file to HDFS

hdfs dfs -put geolocation.csv /user/hadoop/geolocation

#Copy the trucks.csv file to HDFS

hdfs dfs -put trucks.csv /user/hadoop/trucks

hdfs dfs -ls:

**The command ls lists the contents of a directory. For a file, it returns stats of a file. The full syntax is below:**

#Syntax for listing content on HDFS

hdfs dfs -ls <args>

**1. Enter the commands below to list the content of the directories we just created:**

#List the content of the hadoop directory

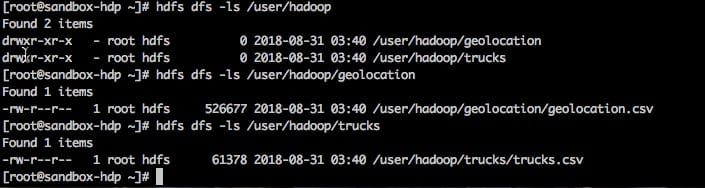
hdfs dfs -ls /user/hadoop

#List the content of the geolocation directory

hdfs dfs -ls /user/hadoop/geolocation

##List the content of the trucks directory

hdfs dfs -ls /user/hadoop/trucks



**Find Out Space Utilization in a HDFS Directory**

hdfs dfs -du:

**The command du displays the size of files and directories contained in the given directory or the size of a file if its just a file.**

#Syntax for displaying the size of a file and directory in HDFS

hdfs dfs -du URI

**1. Continuing with our example, enter the commands below in your terminal to show the size of contents of the hadoop directory and the geolocation.csv file:**

#Displays the size of the directories in the hadoop directory including the geolocation.csv file

hdfs dfs -du /user/hadoop/ /user/hadoop/geolocation/geolocation.csv



**Download Files From HDFS to Local File System**

hdfs dfs -get:

The command **get** Copies/Downloads files from HDFS to the local file system:

//Syntax to copy/download files from HDFS your local file system

hdfs dfs -get <hdfs\_src> <localdst>

**1. Enter the command below to copy the geolocation.csv file into your home directory:**

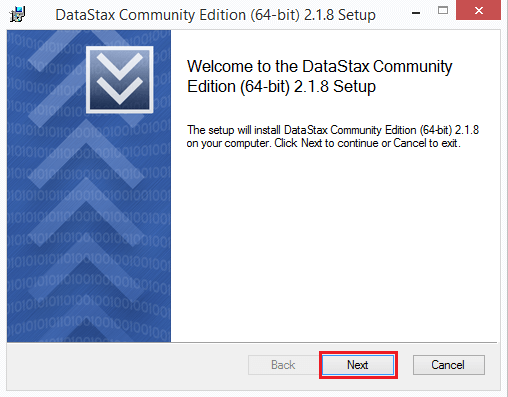
#Copying geolocation.csv into your local file system directory

hdfs dfs -get /user/hadoop/geolocation/geolocation.csv /home/

**PRACTICAL-5**

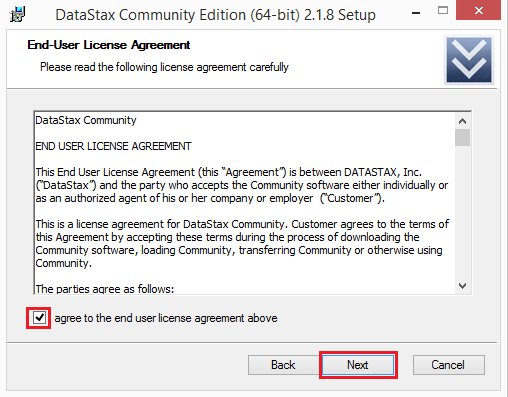
**AIM: To install Cassandra and perform CRUD operations on Cassandra.**

**Step 1)** Run the Datastax community edition setup. After running the Setup, following page will be displayed. Here in the screenshot 64 bit version is being installed. You can download 32 bit version as well according to your requirements. But I recommend 64 bit version to use.



This page gives you information about the Cassandra version you are going to install. Press the 'next' button.

**Step 2)** After pressing the 'next' button, following page will be displayed.



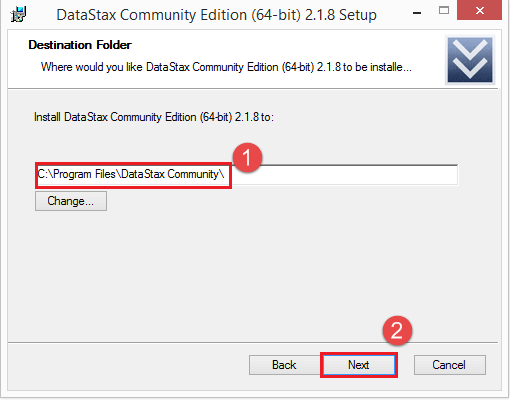
This page is about the license agreement. Mark the checkbox and press the next button.

**Step 3)** After pressing the 'next' button, the following page will be displayed.

This page asks about the installation location.

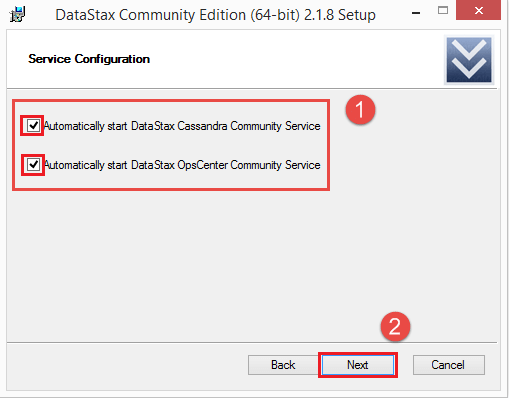
1. Default location is C:\Program Files. You can change installation location if you want to change. It is recommended not to change installation location.

2. After setting installation location, press the 'next' button

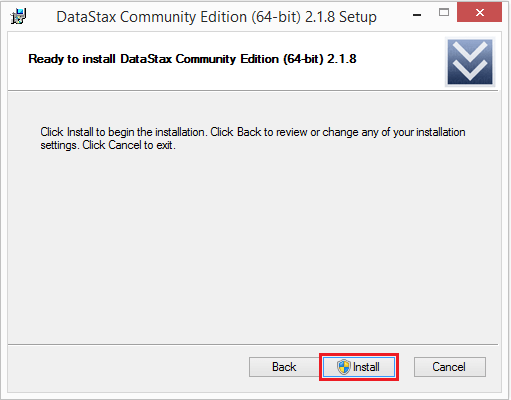


**Step 4)** After pressing 'next' button in above step, the following page will be displayed. This page asks about whether you want to automatically start Cassandra and OpsCenter.

1. Mark the checkboxes if you to want to automatically start Cassandra and opsCenter.
2. After providing this information, press the 'next' button.

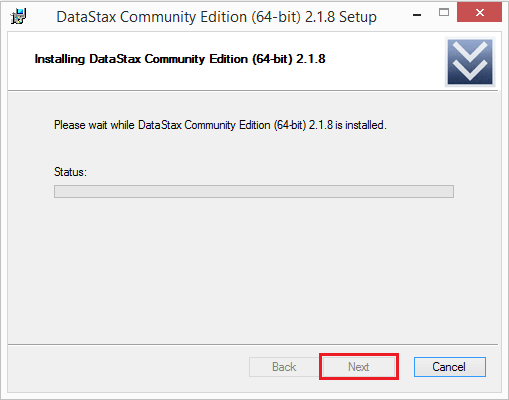


**Step 5)** After pressing the next button, following page will be displayed.

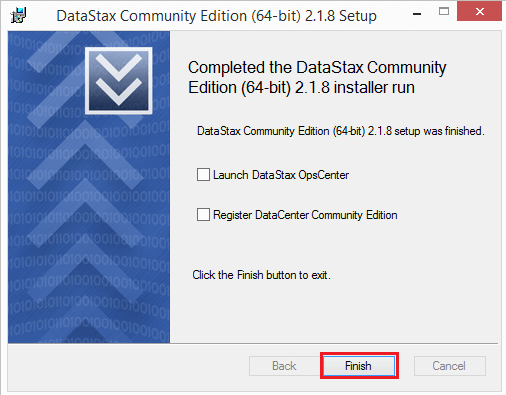


Setup has collected all the necessary information and now the setup is ready to install. Press install button.

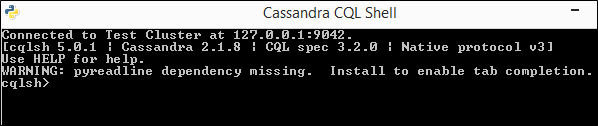
**Step 6)** After pressing 'install' button, following page will be displayed.



Datastax community edition is being installed. After installation is completed, click on next button. When setup is installed successfully, press the 'Finish' button.



Go to windows start programs, search Cassandra CQL Shell and run the Cassandra Shell. After running Cassandra shell, you will see the following command line

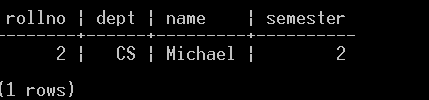


Now you can create a keyspace, tables, and write queries.

**CRUD Operations :-**

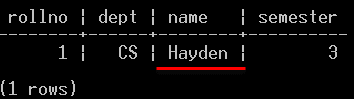
## Insert Data

cqlsh> insert into university.student(RollNo,Name,Dept,Semester) values(2,’Michale’,’CS’,2);



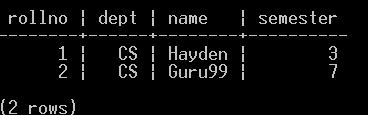
**Update Data**

cqlsh> udate university.student set name = ‘Hayden’ where RollNo = 2;

****

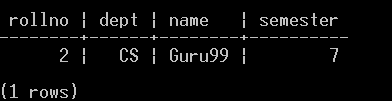
**Read Data**

cqlsh> Select \* from university.student 2;



**Delete Data**

cqlsh> Delete from university.student where RollNo = 2;



**PRACTICAL-6**

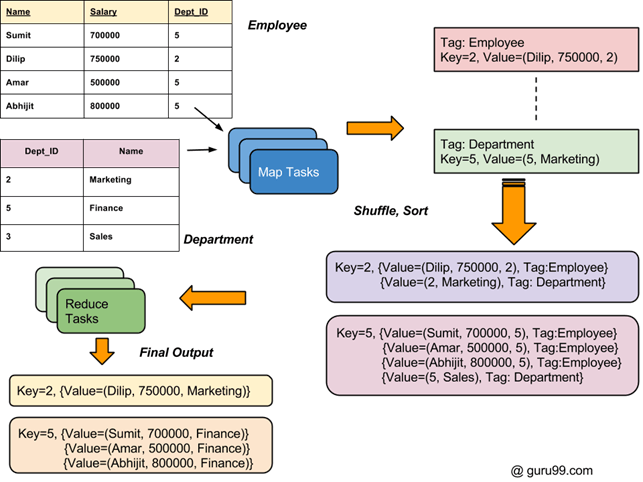
**AIM: Distributed Cache & Map Side Join, Reduce side Join Building and Running a Spark Application Word count in Hadoop and Spark Manipulating RDD.**

**1. Map-side join** - When the join is performed by the mapper, it is called as map-side join. In this type, the join is performed before data is actually consumed by the map function. It is mandatory that the input to each map is in the form of a partition and is in sorted order. Also, there must be an equal number of partitions and it must be sorted by the join key.

**2. Reduce-side join** - When the join is performed by the reducer, it is called as reduce-side join. There is no necessity in this join to have a dataset in a structured form (or partitioned).

Here, map side processing emits join key and corresponding tuples of both the tables. As an effect of this processing, all the tuples with same join key fall into the same reducer which then joins the records with same join key.

An overall process flow of joins in Hadoop is depicted in below diagram.

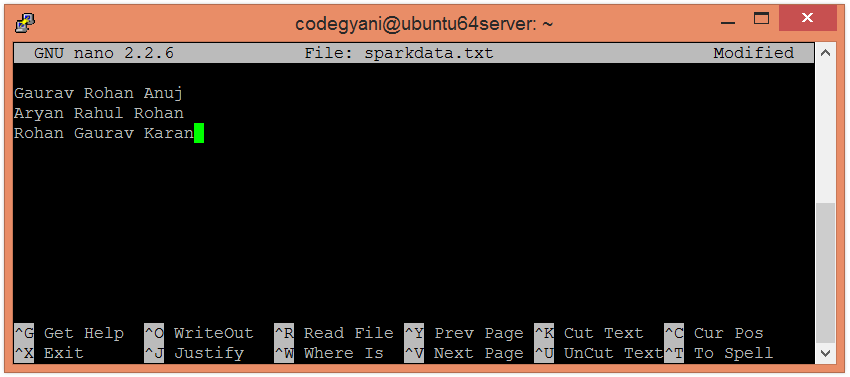
[](https://cdn.guru99.com/images/Big_Data/061114_1003_Introductio1.png)

## Steps to execute Spark word count

In this, we find and display the number of occurrences of each word.

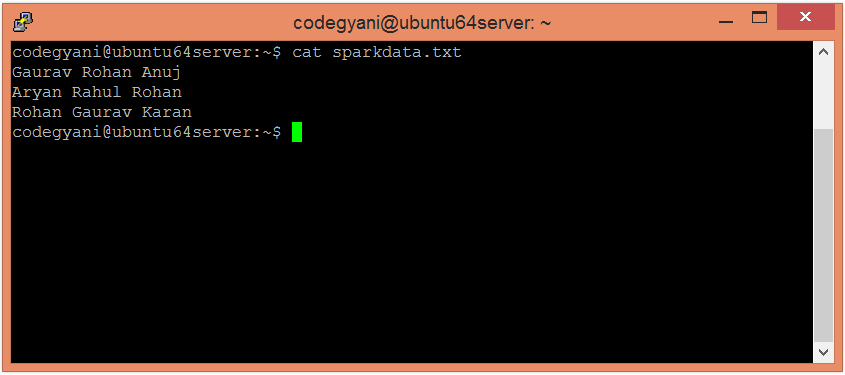
* Create a text file in your local machine and write some text into it.

1. $ nano sparkdata.txt



* Check the text written in the sparkdata.txt file.

1. $ cat sparkdata.txt

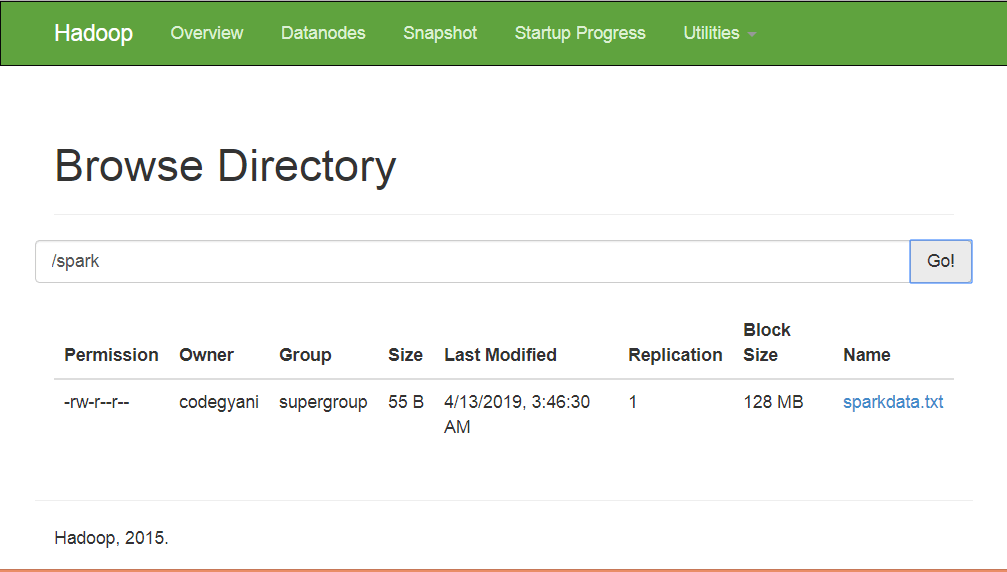


* Create a directory in HDFS, where to kept text file.

1. $ hdfs dfs -mkdir /spark

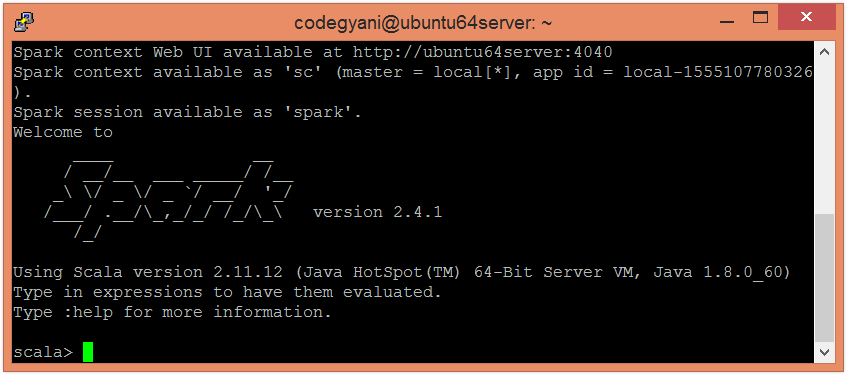
* Upload the sparkdata.txt file on HDFS in the specific directory.

1. $ hdfs dfs -put /home/codegyani/sparkdata.txt /spark



* Now, follow the below command to open the spark in Scala mode.

1. $ spark-shell



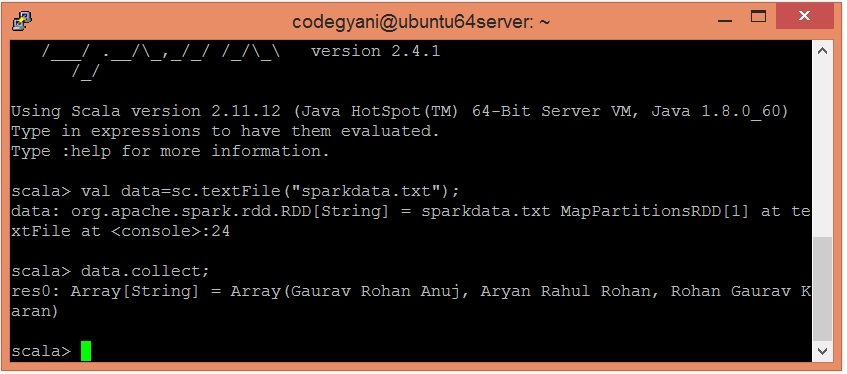
* Let's create an RDD by using the following command.

1. scala**>** val data=sc.textFile("sparkdata.txt")

Here, pass any file name that contains the data.

* Now, we can read the generated result by using the following command.

1. scala**>** data.collect;

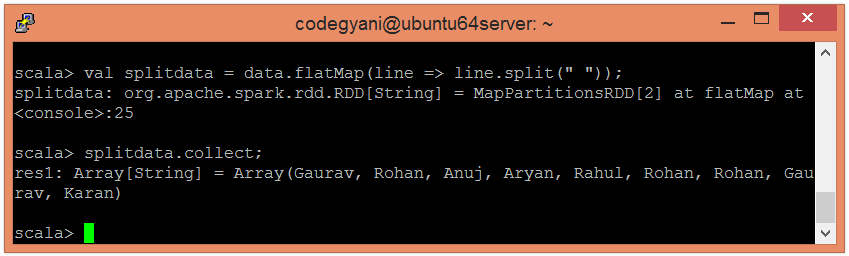


* Here, we split the existing data in the form of individual words by using the following command.

1. scala**>** val splitdata = data.flatMap(line =**>** line.split(" "));

* Now, we can read the generated result by using the following command.

1. scala**>** splitdata.collect;



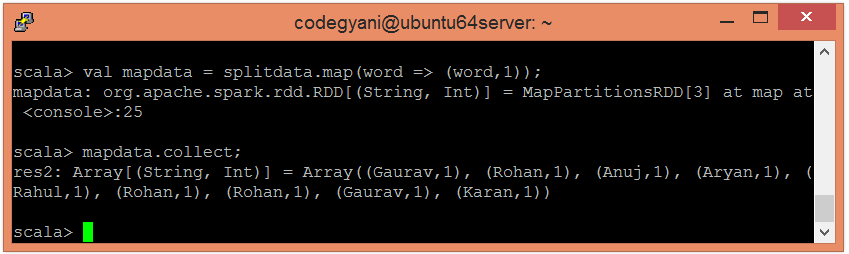
* Now, perform the map operation.

1. scala**>** val mapdata = splitdata.map(word =**>** (word,1));

Here, we are assigning a value 1 to each word.

* Now, we can read the generated result by using the following command.

1. scala**>** mapdata.collect;



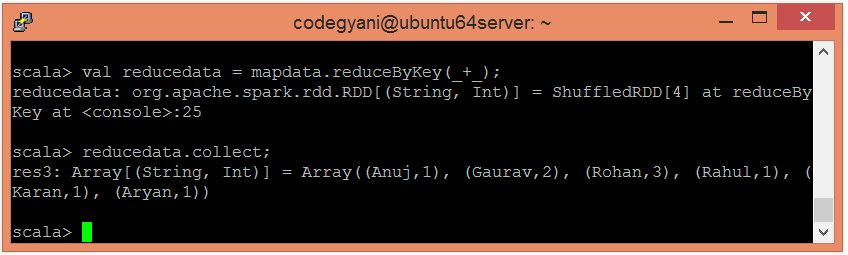
* Now, perform the reduce operation

1. scala**>** val reducedata = mapdata.reduceByKey(\_+\_);

Here, we are summarizing the generated data.

* Now, we can read the generated result by using the following command.

1. scala**>** reducedata.collect;



**PRACTICAL-7**

**AIM: Implementation of commands in Spark Sql programming.**

**PROGRAM:**

# importing required libraries

from pyspark.sql import SQLContext

from pyspark.sql import Row

# read the text data

raw\_data = sc.textFile('sample\_data\_final\_wh.txt').cache()

# get number of partitions

raw\_data.getNumPartitions()

## >> 19

# create spark sql context

sql\_context = SQLContext(sc)

# split the data

csv\_rdd = raw\_data.map(lambda row: row.split(','))

# top 2 rows

csv\_rdd.take(2)

# map the datatypes of each column

parsed = csv\_rdd.map(lambda r : Row( age = int(r[0]),

blood\_group = r[1],

city = r[2],

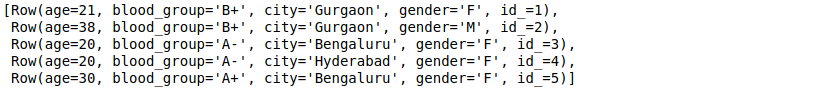
gender = r[3],

id\_ = int(r[4])))

# top 5 rows

parsed.take(5)

spark sql



# create dataframe

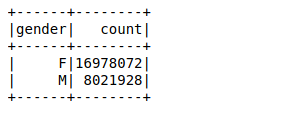
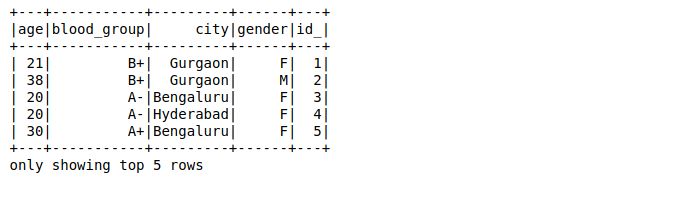
data\_frame = sql\_context.createDataFrame(parsed)

# view the dataframe

data\_frame.show(5)

# value counts of gender

data\_frame.groupby('gender').count().show()



spark sql results

# register temporary table

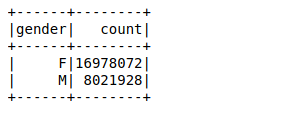
data\_frame.registerTempTable('sample')

# get the value count using the sql query

gender = sql\_context.sql(" SELECT gender, count(\*) as freq from sample GROUP BY gender ")

# view the results

gender.show()

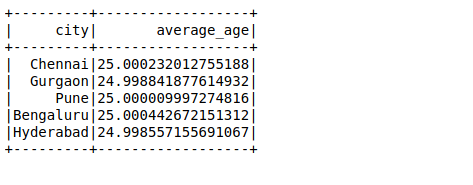
 

# average age city wise

average\_age = sql\_context.sql(" SELECT city, AVG(age) as average\_age from sample GROUP BY city ")

# view the results

average\_age.show()



**PRACTICAL-8**

**AIM: Implementing K-Means Clustering algorithm using Map-Reduce.**

**PROGRAM:**

!rm -r spark\*

!apt-get install openjdk-8-jdk-headless -qq

!wget -q http://mirrors.koehn.com/apache/spark/spark-3.0.0-preview2/spark-3.0.0-preview2-bin-hadoop3.2.tgz

!tar xf spark-3.0.0-preview2-bin-hadoop3.2.tgz

!pip install -q findspark

import os

os.environ["JAVA\_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"

os.environ["SPARK\_HOME"] = "/content/spark-3.0.0-preview2-bin-hadoop3.2"

import findspark

findspark.init()

from pyspark.sql import SparkSession

spark = SparkSession.builder.master("local[\*]").getOrCreate()

spark

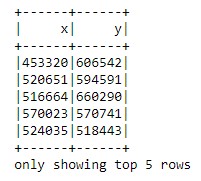
sc= spark.sparkContext.getOrCreate()

dataSet = spark.read.csv("s3.csv", inferSchema=True)

dataSet\_rename = dataSet.withColumnRenamed("\_c0","x") \

.withColumnRenamed("\_c1","y")

dataSet\_rename.show(5,True)



from scipy.spatial import distance

import numpy as np

data\_array = np.array(dataSet\_rename.select("x", "y").collect())

print (len(data\_array))

flag = True ;

#Calculates the Straight line distance between two points

from scipy.spatial import distance

def calculate\_distance(x,y):

return distance.euclidean(x,y);

#MapReduce function for K-means clustering

def mapperReducer(data, centroidPoints):

mappedData = data.rdd.map(lambda x: ([calculate\_distance(x, centroidPoints[i]) for i in range(k)], x)).map(lambda x : (x[0].index(min(x[0])), (x[1],1)))

reducedData= mappedData.reduceByKey(lambda x, y: ((x[0][0]+y[0][0],x[0][1]+y[0][1] ), x[1]+y[1]));

meanXY = reducedData.map(lambda x: (x[0], (x[1][0][0]/x[1][1], x[1][0][1]/x[1][1])));

sort\_meanXY = meanXY.sortByKey(ascending=True).values().collect();

return (sort\_meanXY);

#Finds the new centroids

def findNewCentroids(data):

newCentroidPoints= [];

for i in data:

newCentroidPoints.append([i[0],i[1]]);

return newCentroidPoints;

#To form random centroids for the first iteration.

vectors = [np.array(point) for point in data\_array]

parallelVectors = sc.parallelize(vectors)

parallelVectors.cache()

k =15 #Number of clusters

# You can set the iteration

iteration = 30;

centroidPoints =parallelVectors.takeSample(False, k)

print (centroidPoints)



# Iteration for K-means CLustering

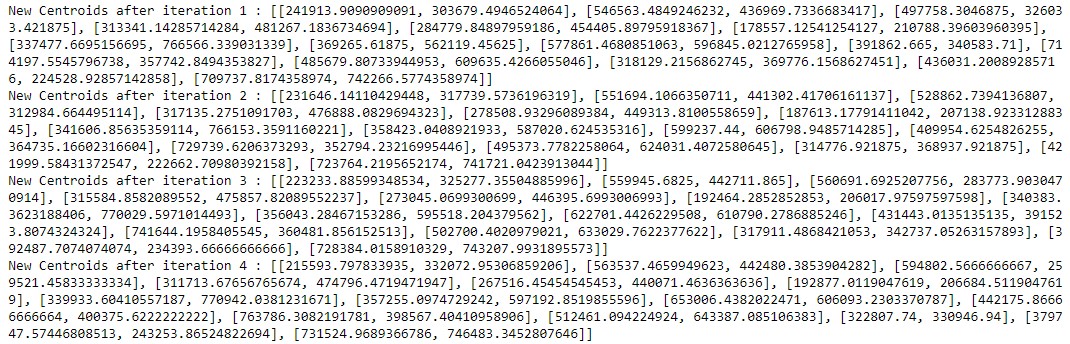
for i in range(iteration):

mappedData = mapperReducer(dataSet\_rename, centroidPoints);

centroidPoints = [];

centroidPoints = findNewCentroids(mappedData);

print ("New Centroids after iteration",i+1,":" ,centroidPoints)



#Cluster assignment to each data point after "n" number of iteration

clusterAssigment = clusterAssigment1= dataSet\_rename.rdd.map(lambda x: ([calculate\_distance(x, centroidPoints[i]) for i in range(k)], x)).map(lambda x : (x[0].index(min(x[0])), x[1]));

clusterAssigment.take(5)

clusterAssigment.saveAsTextFile("resultsClusterAssignment.csv")

#Now refresh. You will see the csv file in the files section in the left side of the window.

#Scatter plot - Run this to generate scatter plot

data = np.array(clusterAssigment.map(lambda x: x[1]).collect());

clusters = np.array(clusterAssigment1.map(lambda x: x[0]).collect());

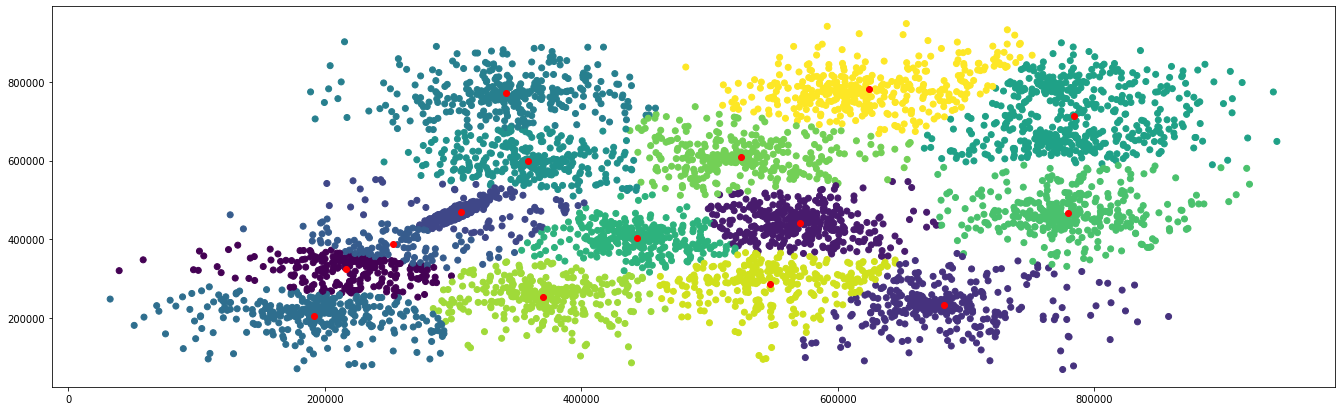
import matplotlib.pyplot as plt

plt.figure(figsize=(23,7))

plt.scatter(data[:,0],data[:,1], c=clusters)

for i in range(k):

plt.scatter(centroidPoints[i][0] ,centroidPoints[i][1], color='red')

****

**PRACTICAL-9**

**AIM: Implementing any one Frequent Item set algorithm using Map-Reduce.**

**PROGRAM:**

import java.util.\*;

import java.io.\*;

import java.util.Scanner;

import java.io.IOException;

import org.apache.hadoop.\*;

public class Project {

static int ntrans=0;

public static class MapperClass

extends Mapper<Object, Text, Text, IntWritable>{

private final static IntWritable one = new IntWritable(1);

private Text word = new Text();

public void map(Object key, Text value, Context context

) throws IOException, InterruptedException {

StringTokenizer items = new StringTokenizer(value.toString()," \t\n\r\f,.:;?![]'");

LinkedList <String>list = new LinkedList<String>();

LinkedList <String>clist = new LinkedList<String>();

LinkedList <String>templist1 = new LinkedList<String>();

LinkedList <String>templist2 = new LinkedList<String>();

ntrans++;

String str="";

int f=0;

int count=0,i=0,j=0,nitem=0;

Iterator iterator;

while (items.hasMoreTokens()) {

word.set(items.nextToken());

nitem++;

count=0;

context.write(word, one);

}

StringTokenizer items2 = new StringTokenizer(value.toString()," \t\n\r\f,.:;?![]'");

while (items2.hasMoreTokens()) {

list.add(items2.nextToken());

}

count=0;

clist.clear();

count=1;

while (count < nitem)

{

count=count+1;

for (i=0;i<(list.size()-1);i++)

{

items2 = new StringTokenizer(list.get(i));

while (items2.hasMoreTokens())

{

templist1.add(items2.nextToken());

}

for (j=i+1;j<list.size();j++)

{

items2 = new StringTokenizer(list.get(j));

while (items2.hasMoreTokens())

{

templist2.add(items2.nextToken());

}

f=0;

for (int k=0;k < (templist1.size()-1);k++)

{

if(!(templist1.get(k).equals(templist2.get(k))))

{

f=1;

break;

}

}

if(f == 0)

{

str="";

str=list.get(i)+" "+templist2.get(templist2.size()-1);

clist.add(str);

items2 = new StringTokenizer(str,"\n");

if(items2.hasMoreTokens()){

word.set(items2.nextToken());

context.write(word, one);

}

}

templist2.clear();

}

templist1.clear();

}

list.clear();

for(int k=0;k < clist.size();k++)

list.add(clist.get(k));

clist.clear();

}

}

}

public static class ReducerClass

extends Reducer<Text,IntWritable,Text,IntWritable> {

private IntWritable result = new IntWritable();

public void reduce(Text key, Iterable<IntWritable> values,

Context context) throws IOException, InterruptedException {

int sum = 0;

for (IntWritable val : values) {

sum += val.get();

}

result.set(sum);

if(sum>7)

context.write(key, result);

}

}

public static void main(String[] args) throws Exception {

long start = System.currentTimeMillis( );

int m;

System.out.println("Enter one inteser");

Scanner in=new Scanner(System.in);

Configuration conf = new Configuration();

Job job = new Job(conf, "project");

job.setJarByClass(Project.class);

job.setMapperClass(MapperClass.class);

job.setReducerClass(ReducerClass.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

FileInputFormat.addInputPath(job, new Path(args[0]));

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

if(job.waitForCompletion(true))

{

}

else

System.exit(1);

}

}

**OUTPUT:**

{Item1 Item2 Item3}

{Item4 Item5 Item6}

{Item3 Item5 Item6}

**PRACTICAL-10**

**AIM: Create A Data Pipeline Based On Messaging Using PySpark And Hive.**

This example demonstrates how to use spark.sql to create and load two tables and select rows from the tables into two DataFrames. The next steps use the DataFrame API to filter the rows for salaries greater than 150,000 from one of the tables and shows the resulting DataFrame. Then the two DataFrames are joined to create a third DataFrame. Finally the new DataFrame is saved to a Hive table.

1. Copy the Hue sample\_07.csv and sample\_08.csv files to your object store in a location accessible by the Spark cluster:

hdfs dfs -put /opt/cloudera/parcels/CDH/lib/hue/apps/beeswax/data/sample\_0\* s3a://*<bucket\_name>*/

1. Launch spark-shell:

spark-shell --conf "spark.yarn.access.hadoopFileSystems=s3a://*<bucket\_name>*"

1. Create Hive tables sample\_07 and sample\_08:

scala> spark.sql("CREATE EXTERNAL TABLE sample\_07 (code string,description string,total\_emp int,salary int) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' STORED AS TextFile LOCATION 's3a://*<bucket\_name>*/s07/'")

scala> spark.sql("CREATE EXTERNAL TABLE sample\_08 (code string,description string,total\_emp int,salary int) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' STORED AS TextFile LOCATION 's3a://*<bucket\_name>*/s08/'")

1. In another session, launch Beeline:

beeline -u "jdbc:hive2://*<HiveServer2\_host>*:10001/default;principal=hive/\_HOST@CLOUDERA.SITE;transportMode=http;httpPath=cliservice"

1. In Beeline, show the Hive tables:

0: jdbc:hive2://hs2.cloudera.site:> show tables;

+------------+--+

| tab\_name |

+------------+--+

| sample\_07 |

| sample\_08 |

+------------+--+

1. In the Spark shell, load the data from the CSV files into the tables:

scala> spark.sql("LOAD DATA INPATH 's3a://*<bucket\_name>*/sample\_07.csv' OVERWRITE INTO TABLE sample\_07")

scala> spark.sql("LOAD DATA INPATH 's3a://*<bucket\_name>*/sample\_08.csv' OVERWRITE INTO TABLE sample\_08")

1. Create DataFrames containing the contents of the sample\_07 and sample\_08 tables:

scala> val df\_07 = spark.sql("SELECT \* from sample\_07")

scala> val df\_08 = spark.sql("SELECT \* from sample\_08")

1. Show all rows in df\_07 with salary greater than 150,000:

scala> df\_07.filter(df\_07("salary") > 150000).show()

The output should be:

+-------+--------------------+---------+------+

| code| description|total\_emp|salary|

+-------+--------------------+---------+------+

|11-1011| Chief executives| 299160|151370|

|29-1022|Oral and maxillof...| 5040|178440|

|29-1023| Orthodontists| 5350|185340|

|29-1024| Prosthodontists| 380|169360|

|29-1061| Anesthesiologists| 31030|192780|

|29-1062|Family and genera...| 113250|153640|

|29-1063| Internists, general| 46260|167270|

|29-1064|Obstetricians and...| 21340|183600|

|29-1067| Surgeons| 50260|191410|

|29-1069|Physicians and su...| 237400|155150|

+-------+--------------------+---------+------+

1. Create the DataFrame df\_09 by joining df\_07 and df\_08, retaining only the code and description columns.

scala> val df\_09 = df\_07.join(df\_08, df\_07("code") === df\_08("code")).select(df\_07.col("code"),df\_07.col("description"))

scala> df\_09.orderBy($"code".asc).show()

The new DataFrame looks like:

+-------+--------------------+

| code| description|

+-------+--------------------+

|00-0000| All Occupations|

|11-0000|Management occupa...|

|11-1011| Chief executives|

|11-1021|General and opera...|

|11-1031| Legislators|

|11-2011|Advertising and p...|

|11-2021| Marketing managers|

|11-2022| Sales managers|

|11-2031|Public relations ...|

|11-3011|Administrative se...|

|11-3021|Computer and info...|

|11-3031| Financial managers|

|11-3041|Compensation and ...|

|11-3042|Training and deve...|

|11-3049|Human resources m...|

|11-3051|Industrial produc...|

|11-3061| Purchasing managers|

|11-3071|Transportation, s...|

|11-9011|Farm, ranch, and ...|

|11-9012|Farmers and ranchers|

+-------+--------------------+

1. Save DataFrame df\_09 as the Hive table sample\_09:

scala> df\_09.write.option("path","s3a://*<bucket\_name>*/s09/").saveAsTable("sample\_09")

1. In Beeline, show the Hive tables:

0: jdbc:hive2://hs2.cloudera.site:> show tables;

+------------+--+

| tab\_name |

+------------+--+

| sample\_07 |

| sample\_08 |

| sample\_09 |

+------------+--+

Here is an equivalent program in Python, that you could submit using spark-submit:

from pyspark import SparkContext, SparkConf, HiveContext

if \_\_name\_\_ == "\_\_main\_\_":

# create Spark context with Spark configuration

conf = SparkConf().setAppName("Data Frame Join")

sc = SparkContext(conf=conf)

sqlContext = HiveContext(sc)

df\_07 = sqlContext.sql("SELECT \* from sample\_07")

df\_07.filter(df\_07.salary > 150000).show()

df\_08 = sqlContext.sql("SELECT \* from sample\_08")

tbls = sqlContext.sql("show tables")

tbls.show()

df\_09 = df\_07.join(df\_08, df\_07.code == df\_08.code).select(df\_07.code,df\_07.description)

df\_09.show()

df\_09.write.saveAsTable("sample\_09")

tbls = sqlContext.sql("show tables")

tbls.show()