Practical No. 1

Aim: Installing and setting environment variables for Working with Apache Hadoop.

Theory: Apache Hadoop is an open-source software framework used to store, manage and process large datasets for various big data computing applications running under clustered systems. It is Java-based and uses Hadoop Distributed File System (HDFS) to store its data and process data using MapReduce.

Implementation:

# Installing Java

1. In Ubuntu, open the terminal and enter the following command:
   ```
   $ sudo apt install default-jdk default-jre -y
   ```
2. Verify the installation via
   ```
   $ java -version
   ```

# Create new user Hadoop and configure password-less SSH

1. Create the new user
   ```
   $ sudo adduser hadoop
   ```
2. Add the hadoop user to the sudo group.
   ```
   $ sudo usermod -aG sudo hadoop
   ```
3. Log out of the current user and switch to the newly created Hadoop user
4. In terminal, install openssh client and server $ apt install openssh-server openssh-client -y
5. Generate public and private key pairs.
   ```
   $ ssh-keygen -t rsa
   ```
6. Add the generated public key from id_rsa.pub to authorized_keys.
   ```
   $ sudo cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
   ```
7. Change the permissions of the authorized_keys file.
   ```
   $ sudo chmod 640 ~/.ssh/authorized_keys
   ```
8. Verify if the password-less SSH is functional.
   ```
   $ ssh localhost
   ```

# Install Apache Hadoop

1. Download the latest stable version of Hadoop. To get the latest version, go to Apache Hadoop official download page.
2. Extract the downloaded file.

```
$ tar -xvzf hadoop-3.3.1.tar.gz
```

3. Move the extracted directory to the /usr/local/ directory.
   ```
   $ sudo mv hadoop-3.3.1 /usr/local/hadoop
   ```
4. Create directory to store system logs.
   ```
   $ sudo mkdir /usr/local/hadoop/logs
   ```
5. Change the ownership of the hadoop directory.
   ```
   $ sudo chown -R hadoop:hadoop /usr/local/Hadoop
   ```

# Configure Hadoop

1. Edit file ~/.bashrc to configure the Hadoop environment variables.
   ```
   $ sudo nano ~/.bashrc
   ```
2. Add the following lines to the file. Save and close the file. export HADOOP_HOME=/usr/local/hadoop export HADOOP_INSTALL=$HADOOP_HOME export HADOOP_MAPRED_HOME=$HADOOP_HOME export HADOOP_COMMON_HOME=$HADOOP_HOME
   ```
   export HADOOP_HDFS_HOME=$HADOOP_HOME

   export YARN_HOME=$HADOOP_HOME export
   HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
   export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
   export HADOOP_OPTS="-
   Djava.library.path=$HADOOP_HOME/lib/native"
   ```
3. Add the changes to the memory
   ```
   $ source ~/.bashrc
   ```

# Configure Java Environment Variables

1. Find the OpenJDK directory and copy the output
   ```
   $ readlink -f /usr/bin/javac
   ```
2. Edit the hadoop-env.sh file.
   ```
   $ sudo nano $HADOOP_HOME/etc/hadoop/hadoop-env.sh
   ```
3. Add the following lines to the file. Then, close and save the file.
   ```
   export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64 export
   HADOOP_CLASSPATH+=" $HADOOP_HOME/lib/*.jar"
   ```
4. Browse to the hadoop lib directory.
   ```
   $ cd /usr/local/hadoop/lib
   ```
5. Download the Javax activation file.

```
$ sudo wget
https://jcenter.bintray.com/javax/activation/javax.activationapi/1.2.0/javax.activation-api-1.2.0.jar
```

6. Verify the Hadoop version.

```
$ hadoop version
```

7. Edit the core-site.xml configuration file to specify the URL for your NameNode.

```
$ sudo nano $HADOOP_HOME/etc/hadoop/core-site.xml
```

8. Add the following lines. Save and close the file.

```
<configuration>
        <property>
                <name>fs.default.name</name>
                <value>hdfs://0.0.0.0:9000</value>
                <description>The default file system URI</description>
        </property>
</configuration>
```

9. Create a directory for storing node metadata and change the ownership to hadoop.

```
$ sudo mkdir -p /home/hadoop/hdfs/{namenode,datanode}
$ sudo chown -R hadoop:hadoop /home/hadoop/hdfs
```

10. Edit hdfs-site.xml configuration file to define the location for storing node metadata, fsimage file.

```
$ sudo nano $HADOOP_HOME/etc/hadoop/hdfs-site.xml
```

11. Add the following lines. Close and save the file.

```
<configuration>
        <property>
                <name>dfs.replication</name>
                <value>1</value>
        </property>
        <property>
                <name>dfs.name.dir</name>
                <value>file:///home/hadoop/hdfs/namenode</value>
        </property>
        <property>
                <name>dfs.data.dir</name>
                <value>file:///home/hadoop/hdfs/datanode</value>
        </property>
        <property>
                <name>dfs.permissions.enabled</name>
                <value>false</value>
        </property>
</configuration>
```

12. Edit mapred-site.xml configuration file to define MapReduce values.

```
$ sudo nano $HADOOP_HOME/etc/hadoop/mapred-site.xml
```

13. Add the following lines. Save and close the file.

```
<configuration>
        <property>
                <name>mapreduce.framework.name</name>
                <value>yarn</value>
        </property>
        <property>
                <name>yarn.app.mapreduce.am.env</name>
                <value>HADOOP_MAPRED_HOME=/usr/local/hadoop</value>
                <description>Change this to your hadoop location.</description>
        </property>
        <property>
                <name>mapreduce.map.env</name>
                <value>HADOOP_MAPRED_HOME=/usr/local/hadoop</value>
                <description>Change this to your hadoop location.</description>
        </property>
        <property>
                <name>mapreduce.reduce.env</name>
                <value>HADOOP_MAPRED_HOME=/usr/local/hadoop/<value>
                <description>Change this to your hadoop location.</description>

        </property>
</configuration>
```

14. Edit the yarn-site.xml configuration file and define YARN-related settings.

```
$ sudo nano $HADOOP_HOME/etc/hadoop/yarn-site.xml
```

15. Add the following lines. Save and close the file.

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

16. Validate the Hadoop configuration and format the HDFS NameNode.

```
$ hdfs namenode -format
```

# Start the Apache Hadoop Cluster

1. Start the NameNode and DataNode.

```
$ start-dfs.sh
```

2. Start the YARN resource and node managers.

```
$ start-yarn.sh
```

3. Verify all the running components. $ jps

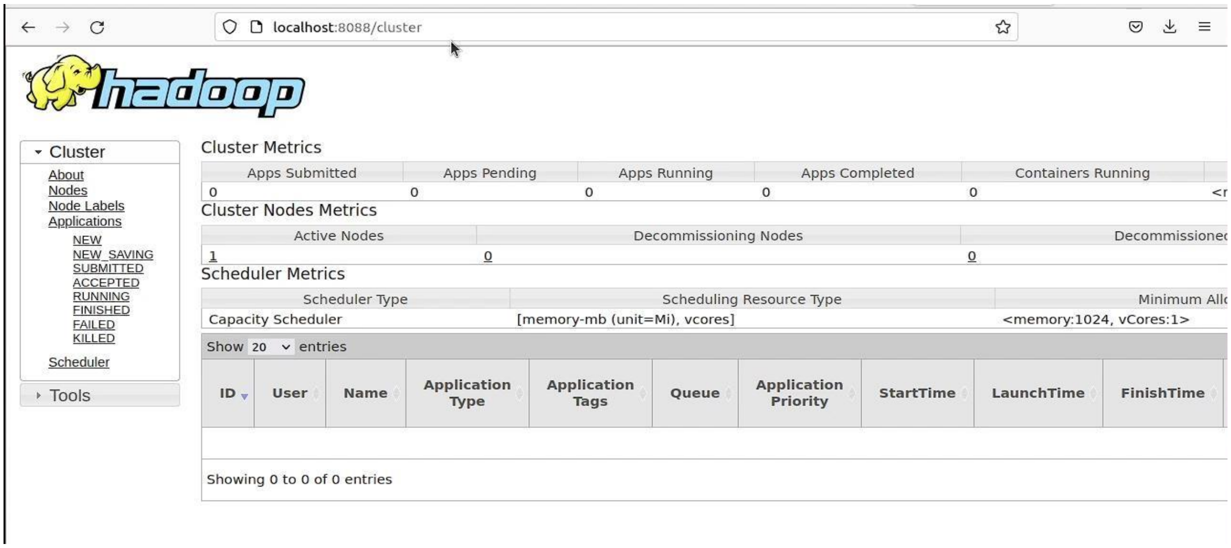
# Access Apache Hadoop Web Interface

In your web browser enter the URLs given below to access the interface

http://localhost:9870

http://localhost:8088

Output:



Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	
0	0	0	0	0	<r

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissionec
1	0	0

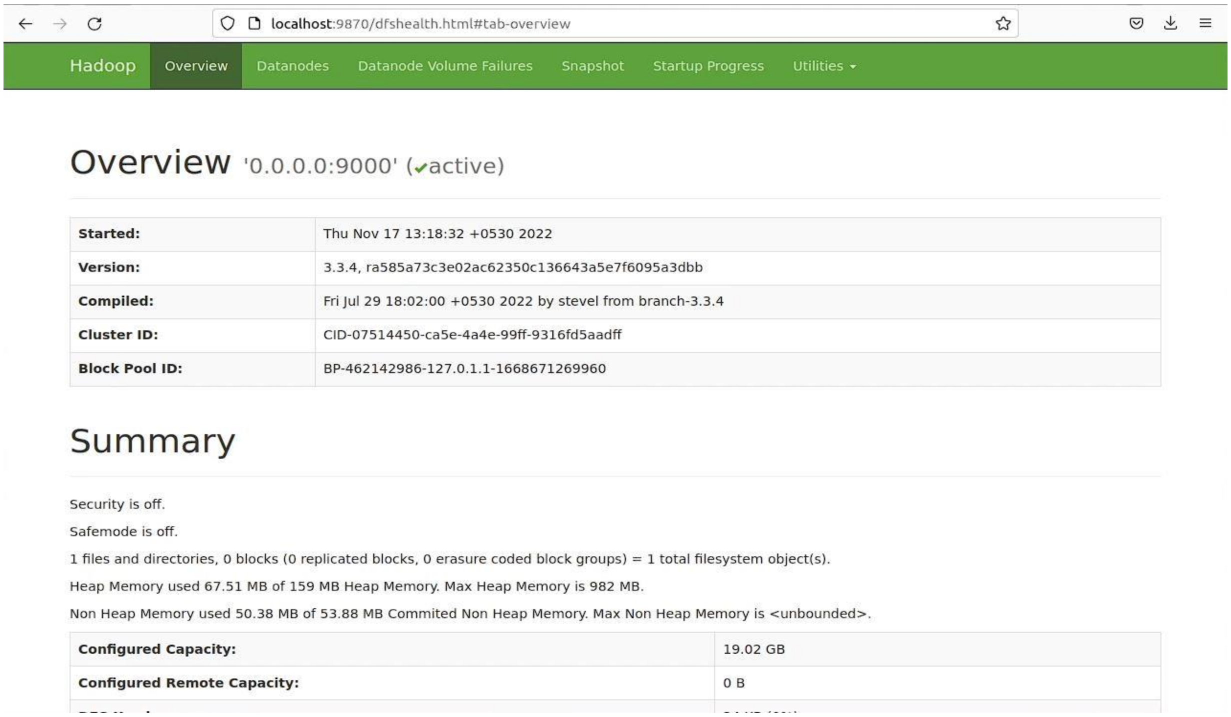
Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allo
Capacity Scheduler	[memory-mb (unit=Mi), vcores]	<memory:1024, vCores:1>

Show 20 entries

ID	User	Name	Application Type	Application Tags	Queue	Application Priority	StartTime	LaunchTime	FinishTime

Showing 0 to 0 of 0 entries



Overview '0.0.0.0:9000' (✔active)

Started:	Thu Nov 17 13:18:32 +0530 2022
Version:	3.3.4, ra585a73c3e02ac62350c136643a5e7f6095a3dbb
Compiled:	Fri Jul 29 18:02:00 +0530 2022 by stevel from branch-3.3.4
Cluster ID:	CID-07514450-ca5e-4a4e-99ff-9316fd5aadff
Block Pool ID:	BP-462142986-127.0.1.1-1668671269960

Summary

Security is off.

Safemode is off.

1 files and directories, 0 blocks (0 replicated blocks, 0 erasure coded block groups) = 1 total filesystem object(s).

Heap Memory used 67.51 MB of 159 MB Heap Memory. Max Heap Memory is 982 MB.

Non Heap Memory used 50.38 MB of 53.88 MB Commited Non Heap Memory. Max Non Heap Memory is <unbounded>.

Configured Capacity:	19.02 GB
Configured Remote Capacity:	0 B

Practical No. 2

Aim: Implementing Map-Reduce Program for Word count problem

Theory: MapReduce is a programming model and an associated implementation for processing and generating big data sets with a parallel, distributed algorithm on a cluster.

Implementation:

```
# create the 3 Java files in a folder ‘prac2’ while logged in as Hadoop user
# WC_Mapper.java package
com.wordcountproblem; import
java.io.IOException; import
java.util.StringTokenizer; 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 WC_Mapper extends MapReduceBase implements
Mapper & lt;
LongWritable, Text, Text, IntWritable & gt; {     private
final static IntWritable one = new IntWritable(1);     private
Text word = new Text();
    public void map(LongWritable key, Text value, OutputCollector & lt; Text, IntWritable & gt;
output,
        Reporter reporter) throws IOException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
           word.set(tokenizer.nextToken());
output.collect(word, one);
        }
    }

}

# WC_Reducer.java package
com.wordcountproblem; 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 WC_Reducer extends MapReduceBase implements
Reducer & lt;
Text, IntWritable, Text, IntWritable & gt; {
    public void reduce(Text key, Iterator & lt; IntWritable & gt; values, OutputCollector & lt; Text,
IntWritable & gt; output,
        Reporter reporter) throws IOException {
int sum = 0;
        while (values.hasNext()) {
sum += values.next().get();
        }
        output.collect(key, new IntWritable(sum));
    }
}

#WC_Runner package
com.wordcountproblem;

import java.io.IOException; 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.mapred.TextInputFormat; import
org.apache.hadoop.mapred.TextOutputFormat; public
class WC_Runner {
    public static void main(String[] args) throws IOException {
JobConf conf = new JobConf(WC_Runner.class);
conf.setJobName( & quot; WordCount & quot;);
conf.setOutputKeyClass(Text.class);
conf.setOutputValueClass(IntWritable.class);
conf.setMapperClass(WC_Mapper.class);
conf.setCombinerClass(WC_Reducer.class);
conf.setReducerClass(WC_Reducer.class);
conf.setInputFormat(TextInputFormat.class);
conf.setOutputFormat(TextOutputFormat.class);
FileInputFormat.setInputPaths(conf, new Path(args[0]));

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

# Compile the java files

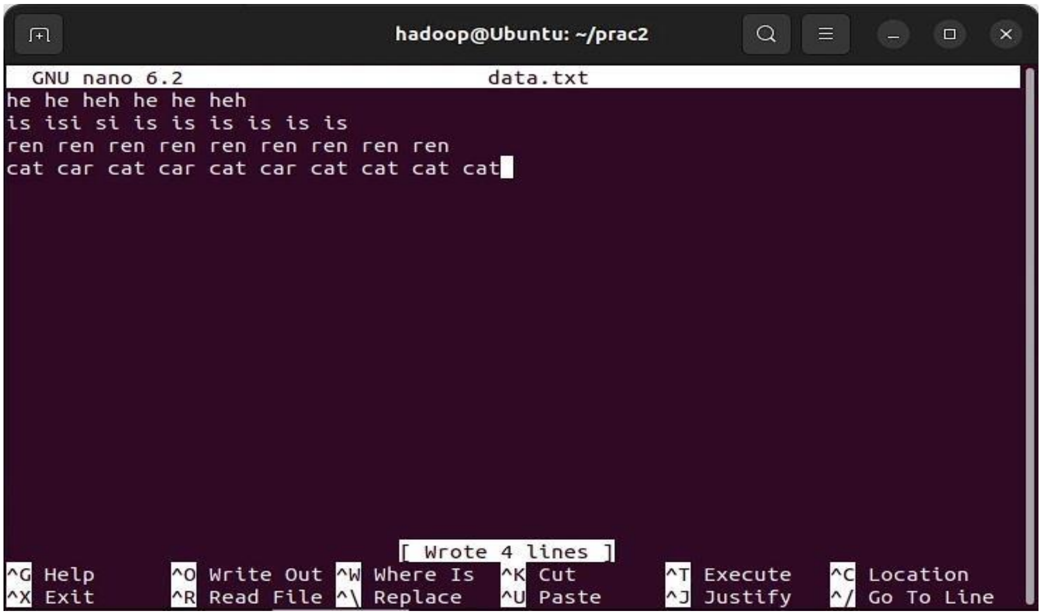
```
            javac -classpath hadoop-core-1.2.1.jar -d wordcountproblem WC_Mapper.java
WC.Reducer.java WC_Runner.java
```

# Create the JAR file

```
            $ jar -cvf wordcountproblem.jar -C wordcountproblem /
```

# Create a text file in the same folder with some content

```
            $ sudo nano data.txt
```



```
  GNU nano 6.2                      data.txt
he he heh he he heh
is isi si is is is is is is
ren ren ren ren ren ren ren ren ren
cat car cat car cat car cat cat cat cat

                         [ Wrote 4 lines ]
^G Help       ^O Write Out ^W Where Is  ^K Cut        ^T Execute    ^C Location
^X Exit       ^R Read File ^\ Replace   ^U Paste      ^J Justify    ^/ Go To Line
```

# start the Hadoop server

$ start-dfs.sh

$ start-yarn.sh

# Uploading file to Hadoop

1. Go to http://localhost:9870/ > click on utilities > Browse the file system
2. Click on create a new folder and enter name as ‘test’
3. Enter the created folder > click on upload button beside the create new folder button > Locate the created text file and open it

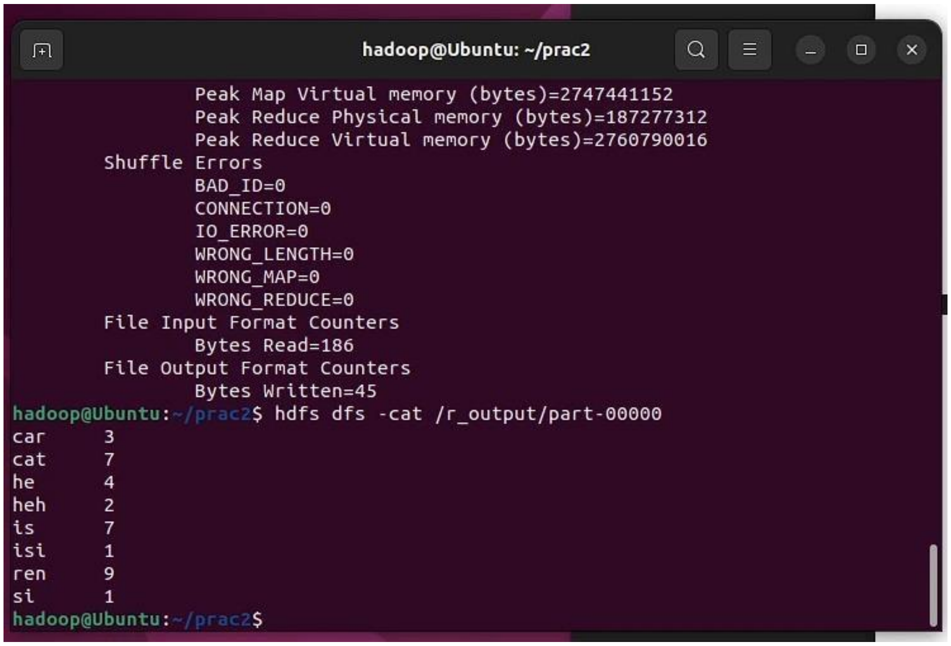
# Run the MapReduce program with the JAR file

$ hadoop jar wordcountproblem.jar com.wordcountproblem.WC_Runner /test/data.txt /r_output

# print the generated output to the console

$ hdfs dfs -cat /r_output/part-00000

Output:



```
                Peak Map Virtual memory (bytes)=2747441152
                Peak Reduce Physical memory (bytes)=187277312
                Peak Reduce Virtual memory (bytes)=2760790016
        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=186
        File Output Format Counters
                Bytes Written=45
hadoop@Ubuntu:~/prac2$ hdfs dfs -cat /r_output/part-00000
car     3
cat     7
he      4
heh     2
is      7
isi     1
ren     9
si      1
hadoop@Ubuntu:~/prac2$
```

Practical No. 3

Aim: Download and install Spark. Create Graphical data and access the graphical data using Spark.

Theory: Apache Spark is an open-source unified analytics engine for large-scale data processing. Spark provides an interface for programming clusters with implicit data parallelism and fault tolerance.

Implementation:

# Scala installation

```
$ sudo apt install scala
```

# check scala installation

```
$ scala –version
```

# Spark installation

1. Extract the Spark (make sure to be in the directory the spark is downloaded)
   ```
   $ sudo tar -xvf spark-3.2.1-bin-hadoop3.2.tgz
   ```
2. Create an installation directory /opt/spark.
   ```
   $ sudo mkdir /opt/spark
   ```
3. Move the extracted files to the installation directory.
   ```
   $ sudo mv spark-3.2.0-bin-hadoop3.2/* /opt/spark
   ```
4. Change the permission of the directory.
   ```
   $ sudo chmod -R 777 /opt/spark
   ```
5. Edit the bashrc configuration file to add Apache Spark installation directory to the system path. $ sudo nano ~/.bashrc
6. Add the code below at the end of the file, save and exit the file: export SPARK_HOME=/opt/spark
   ```
   export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin
   ```
7. Save the changes to take effect.
   ```
   $ source ~/.bashrc
   ```
8. Start the standalone master server.
9. $ start-master.sh
10. Find your server hostname from the dashboard by visiting http://Localhost:8080. Under the URL value. It might look like this:
    ```
    Spark://Ubuntu.myguest.virtualbox.org:7077
    ```

11. Start the Apache Spark worker process. Change spark://ubuntu:7077 with your server hostname.

    ```
    $ start-worker.sh Spark://Ubuntu.myguest.virtualbox.org:7077
    ```

12. Use jps to confirm the status
13. Type spark-shell to access the shell

# Entering graphical data in spark shell

#creating graphical data in graphx

```
import org.apache.spark.graphx._
```

# creating own data type case class

```
User(name: String, age: Int)

val users = List((1L, User("Alex", 26)), (2L, User("Bill", 42)), (3L, User("Carol", 18)), (4L,
User("Dave", 16)), (5L, User("Eve", 45)), (6L, User("Farell", 30)), (7L, User ("Garry", 32)), (8L,
User("Harry", 36)), (9L,
User("Ivan", 28)), (10L, User("Jill",

48))) val usersRDD = sc.parallelize

(users)

val follows = List(Edge(1L, 2L, 1), Edge(2L, 3L, 1), Edge(3L, 1L, 1), Edge(3L, 4L, 1), Edge(3L,
5L, 1), Edge(4L,
5L, 1), Edge(6L, 5L, 1), Edge(7L, 6L, 1), Edge(6L, 8L, 1), Edge(7L, 8L, 1), Edge(7L, 9L,
1), Edge(9L, 8L, 1), Edge(8L, 10L, 1), Edge(10L, 9L, 1), Edge(1L, 1L, 1)) val followsRDD
= sc.parallelize(follows)
```

# creating user to access data 19 val defaultUser =

```
User("Icarus", 22) val socialgraph = Graph (usersRDD,

followsRDD, defaultUser)
```

#Access data of the graph

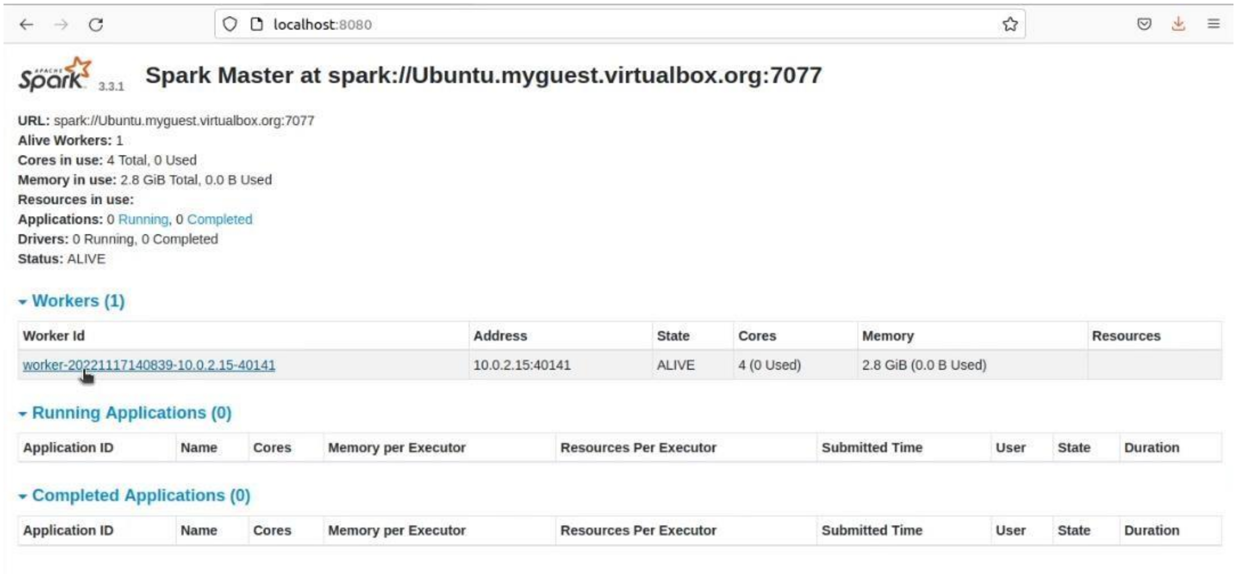
```
socialgraph.numEdges
```

socialgraph.numVertices

socialgraph.inDegrees.collect

socialgraph.outDegrees.collect

Output:



Spark Master at spark://Ubuntu.myguest.virtualbox.org:7077

URL: spark://Ubuntu.myguest.virtualbox.org:7077
Alive Workers: 1
Cores in use: 4 Total, 0 Used
Memory in use: 2.8 GiB Total, 0.0 B Used
Resources in use:
Applications: 0 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (1)

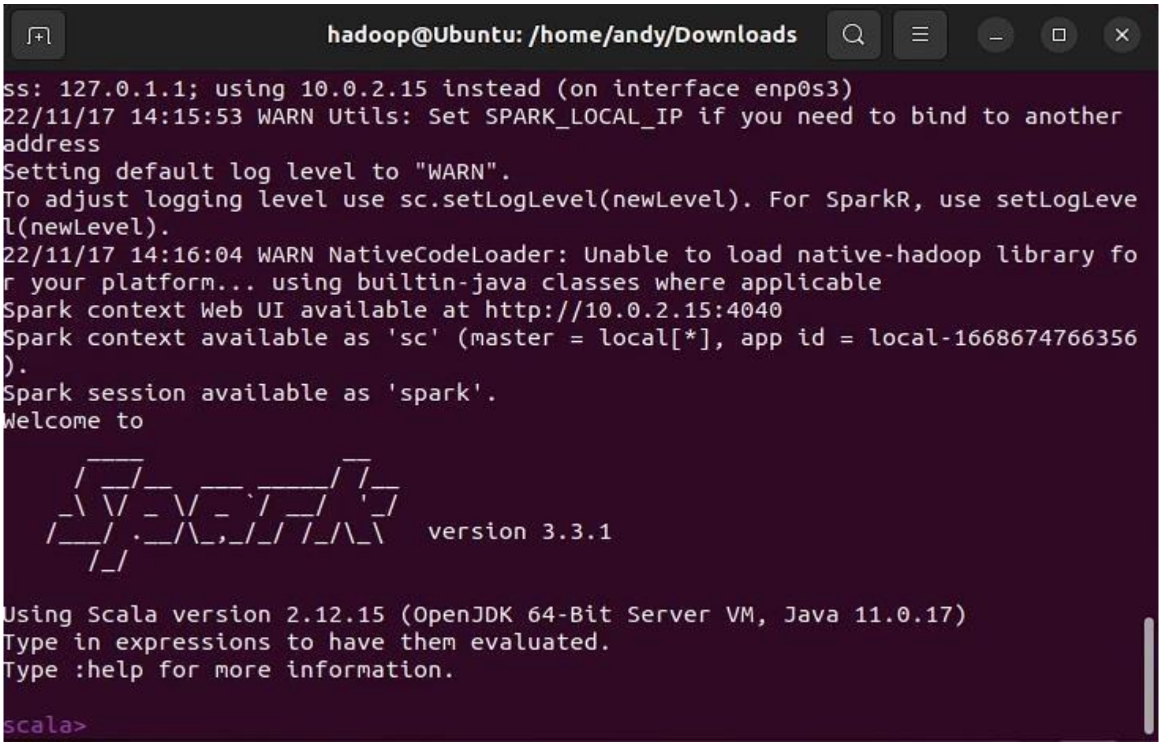
Worker Id	Address	State	Cores	Memory	Resources
worker-20221117140839-10.0.2.15-40141	10.0.2.15:40141	ALIVE	4 (0 Used)	2.8 GiB (0.0 B Used)	

Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration

Completed Applications (0)

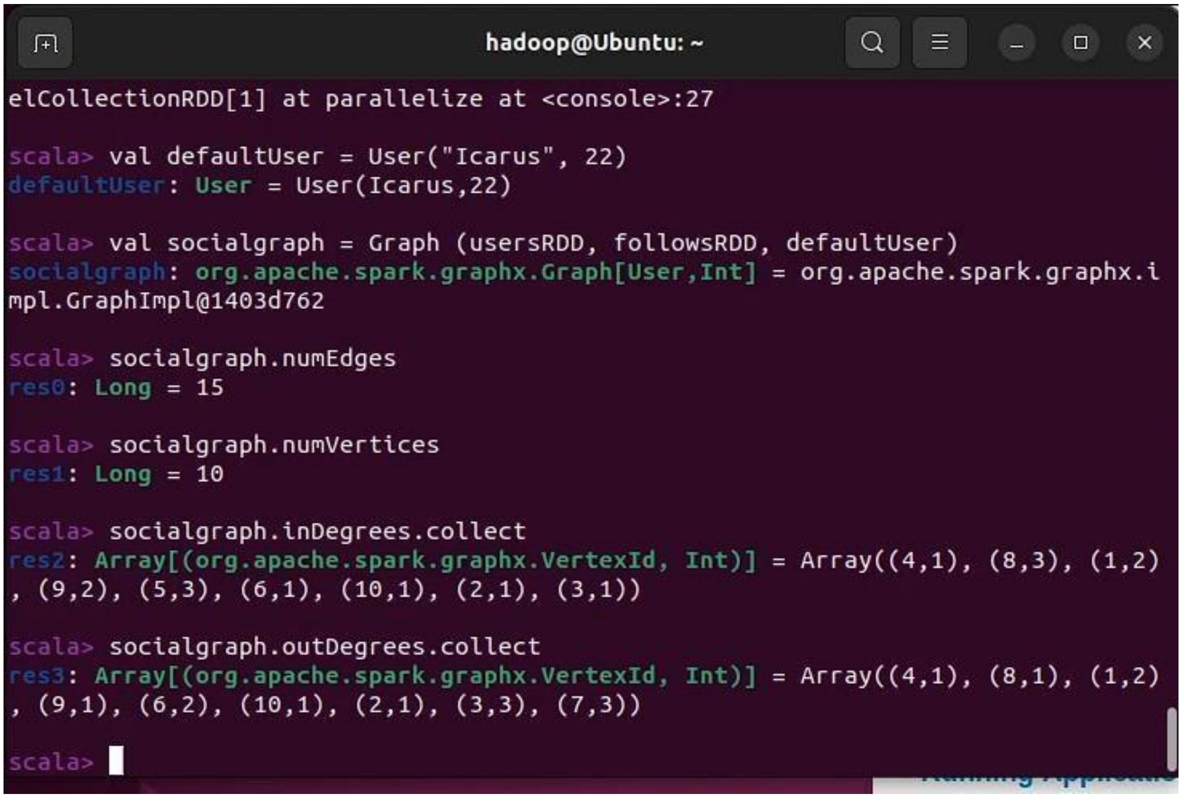
Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration



```
ss: 127.0.1.1; using 10.0.2.15 instead (on interface enp0s3)
22/11/17 14:15:53 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another
address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLeve
l(newLevel).
22/11/17 14:16:04 WARN NativeCodeLoader: Unable to load native-hadoop library fo
r your platform... using builtin-java classes where applicable
Spark context Web UI available at http://10.0.2.15:4040
Spark context available as 'sc' (master = local[*], app id = local-1668674766356
).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.3.1
      /_/

Using Scala version 2.12.15 (OpenJDK 64-Bit Server VM, Java 11.0.17)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```



```
elCollectionRDD[1] at parallelize at <console>:27

scala> val defaultUser = User("Icarus", 22)
defaultUser: User = User(Icarus,22)

scala> val socialgraph = Graph (usersRDD, followsRDD, defaultUser)
socialgraph: org.apache.spark.graphx.Graph[User,Int] = org.apache.spark.graphx.i
mpl.GraphImpl@1403d762

scala> socialgraph.numEdges
res0: Long = 15

scala> socialgraph.numVertices
res1: Long = 10

scala> socialgraph.inDegrees.collect
res2: Array[(org.apache.spark.graphx.VertexId, Int)] = Array((4,1), (8,3), (1,2)
, (9,2), (5,3), (6,1), (10,1), (2,1), (3,1))

scala> socialgraph.outDegrees.collect
res3: Array[(org.apache.spark.graphx.VertexId, Int)] = Array((4,1), (8,1), (1,2)
, (9,1), (6,2), (10,1), (2,1), (3,3), (7,3))

scala> 
```

Practical No. 4

Aim: Write a Spark code for the given application and handle error

Theory: Scala offers different classes for functional error handling in Spark. These classes include but are not limited to Try/Success/Failure, Option/Some/None, Either/Left/Right. Depending on what you are trying to achieve you may want to choose a trio class based on the unique expected outcome of your code.

Implementation:

# switch to Hadoop user

# Install scala-sbt via the latest command available in the terminal

```
echo "deb https://repo.scala-sbt.org/scalasbt/debian all main" | sudo tee /etc/apt/so
urces.list.d/sbt.list

echo "deb https://repo.scala-sbt.org/scalasbt/debian /" | sudo tee /etc/apt/sources.l ist.d/sbt_old.list

curl -sL "https://keyserver.ubuntu.com/pks/lookup?op=get&search=0x2EE0EA64E40A89B84B2
DF73499E82A75642AC823" | sudo apt-key
add sudo apt-get update sudo apt-get install
sbt
```

# enter sbt in the terminal to verify installation

# create the scala program for exception handling

```
$ nano ExceptionHandlingTest.scala
```

```
import org.apache.spark.sql.SparkSession

object ExceptionHandlingTest {   def main(args:
Array[String]): Unit = {     val spark = SparkSession
      .builder
      .appName("ExceptionHandlingTest")
      .getOrCreate()

    spark.sparkContext.parallelize(0 until spark.sparkContext.defaultParallelism).foreach { i =>       if
(math.random > 0.75) {
        throw new Exception("Testing exception handling")
      }
    }
    spark.stop()
  }
}
```

# create the sbt dependency file ‘exceptionhandlingtest.sbt’ name

```
:= "exceptionhandlingtest"


version := "1.0"

scalaVersion := "2.12.15"

val sparkVersion = "3.3.1"

libraryDependencies ++= Seq(
  "org.apache.spark" %% "spark-core" % sparkVersion,
  "org.apache.spark" %% "spark-sql" % sparkVersion
  )
```

# start spark master & worker

```
 $ spark-master.sh
 $ spark-worker.sh spark://Ubuntu.myguest.virtualbox.org:7077/
```

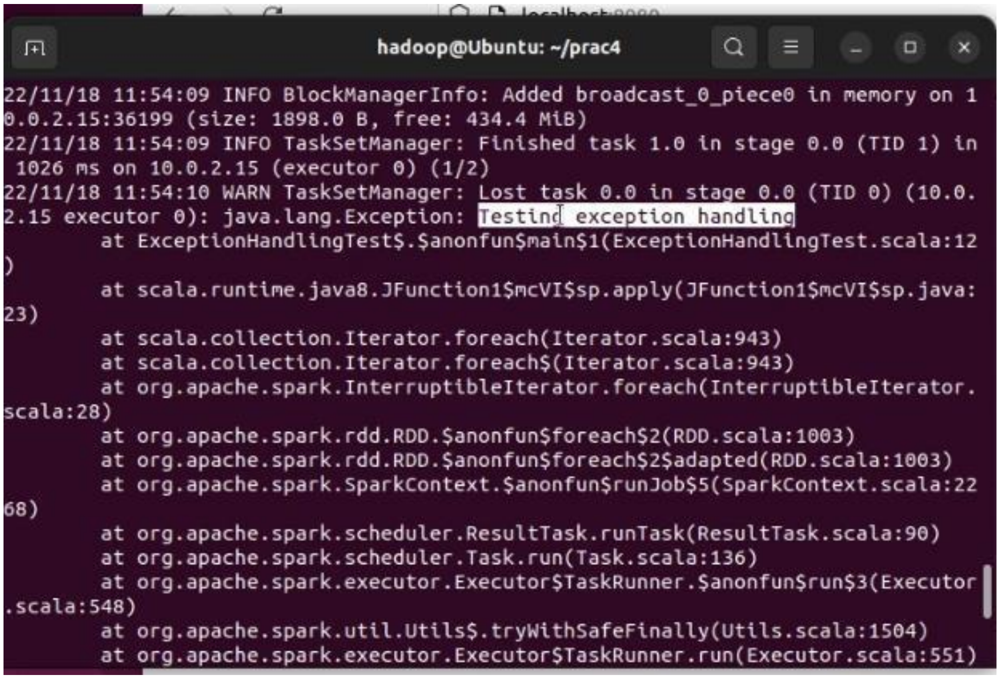
# go back to the terminal pointing to the directory with scala and sbt file and create the sbt package

```
        $ sbt package
```

# submit the program to spark

```
        $ spark-submit –class ‘ExceptionHandlingTest’ –master
‘spark://Ubuntu.myguest.virtualbox.org:7077/’ \target/scala-2.12/exceptionhandling-2.12-1.0.jar
```

Output:



```
22/11/18 11:54:09 INFO BlockManagerInfo: Added broadcast_0_piece0 in memory on 1
0.0.2.15:36199 (size: 1898.0 B, free: 434.4 MiB)
22/11/18 11:54:09 INFO TaskSetManager: Finished task 1.0 in stage 0.0 (TID 1) in
 1026 ms on 10.0.2.15 (executor 0) (1/2)
22/11/18 11:54:10 WARN TaskSetManager: Lost task 0.0 in stage 0.0 (TID 0) (10.0.
2.15 executor 0): java.lang.Exception: Testing exception handling
        at ExceptionHandlingTest$.$anonfun$main$1(ExceptionHandlingTest.scala:12
)
        at scala.runtime.java8.JFunction1$mcVI$sp.apply(JFunction1$mcVI$sp.java:
23)
        at scala.collection.Iterator.foreach(Iterator.scala:943)
        at scala.collection.Iterator.foreach$(Iterator.scala:943)
        at org.apache.spark.InterruptibleIterator.foreach(InterruptibleIterator.
scala:28)
        at org.apache.spark.rdd.RDD.$anonfun$foreach$2(RDD.scala:1003)
        at org.apache.spark.rdd.RDD.$anonfun$foreach$2$adapted(RDD.scala:1003)
        at org.apache.spark.SparkContext.$anonfun$runJob$5(SparkContext.scala:22
68)
        at org.apache.spark.scheduler.ResultTask.runTask(ResultTask.scala:90)
        at org.apache.spark.scheduler.Task.run(Task.scala:136)
        at org.apache.spark.executor.Executor$TaskRunner.$anonfun$run$3(Executor
.scala:548)
        at org.apache.spark.util.Utils$.tryWithSafeFinally(Utils.scala:1504)
        at org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:551)
```

Practical No. 5

Aim: Write a Spark code to handle the Streaming of data

Theory: Spark streaming uses Spark Core's fast scheduling capability to perform streaming analytics. It ingests data in mini-batches and performs RDD transformations on those mini-batches of data. This design enables the same set of application code written for batch analytics to be used in streaming analytics, thus facilitating easy implementation of lambda architecture.

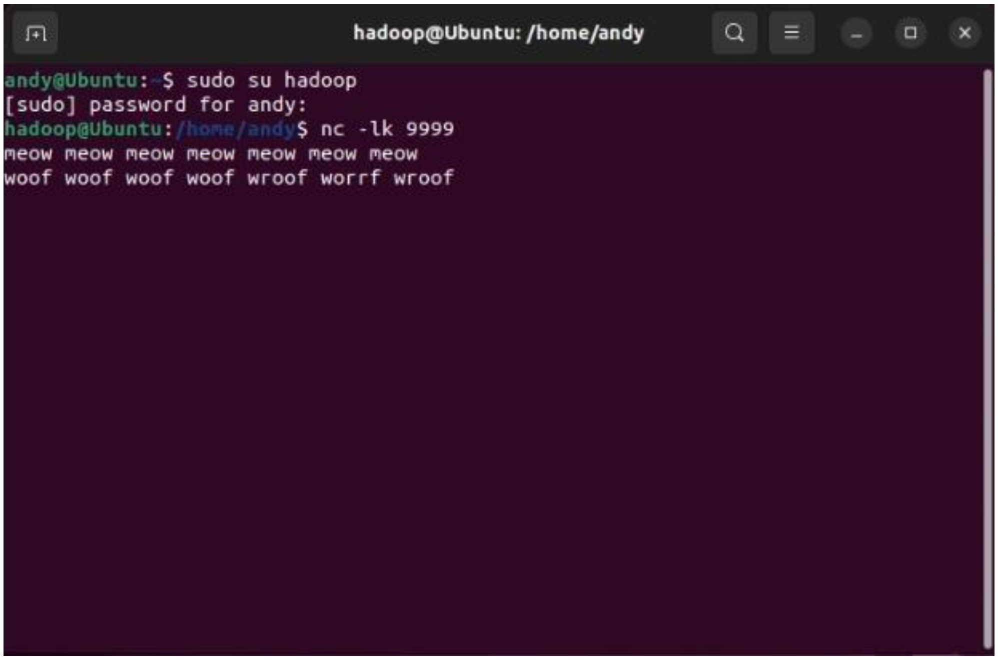
Implementation:

# sbt package creation

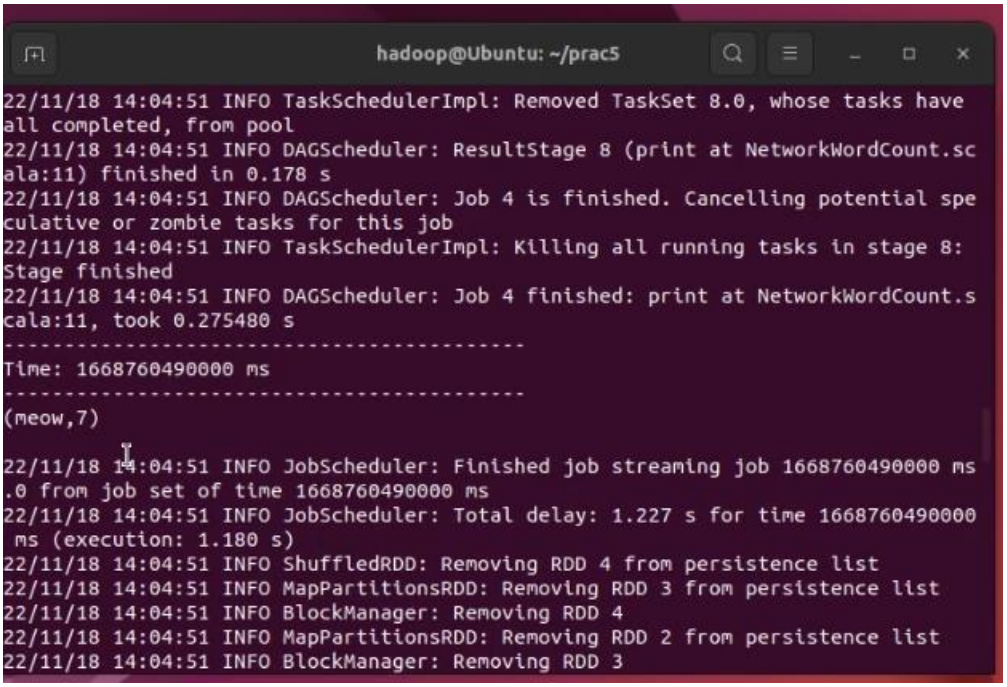
1. Create a new folder logged in as Hadoop user
2. Create file ‘NetworkWordCount.scala’ in that folder import
   ```
   org.apache.spark.SparkConf import org.apache.spark.streaming._
   object NetworkWordCount {
                 def main(args: Array[String]): Unit = {
                   val sparkConf = new
               SparkConf().setMaster("local[2]")setAppName("NetworkWordCount")
                   val ssc = new StreamingContext(sparkConf,
               Seconds(10))     val lines =
               ssc.socketTextStream("localhost",9999)
                   val words = lines.flatMap(_.split(" "))     val
               tuples = words.map(word => (word ,1))     val
               wordCounts = tuples.reduceByKey((t, v) => t +
               v)     wordCounts.print()     ssc.start()
                   ssc.awaitTermination()
                 }
               }
   ```
3. Create a ‘networkwordcount.sbt’ file in the same folder name :=
   ```
   "networkwordcount" version := "1.0.0" scalaVersion := "2.12.15"
               libraryDependencies += "org.apache.spark" % "spark-streaming_2.12" % "3.3.1"
               % "provided"
   ```
4. Create the sbt package
   ```
   $ sbt package
   ```
5. Start the spark server $ start-master.sh
   ```
   $ start-worker.sh spark://Ubuntu.myguest.virtualbox.org:7077/
   ```

6. On a separate terminal, start the netscape server
   $ nc –lk 9999
7. On the original terminal, submit the scala program to spark $ spark-submit –class ‘NetworkWordCount –master
   ‘spark://Ubuntu.myguest.virtualbox.org:7077/’ \target/scala-2.12/networkwordcount_2.12-1.0.0.jar
8. On the netscape terminal provide textual input for the scala program to count words of the live stream every 10 seconds

Output:

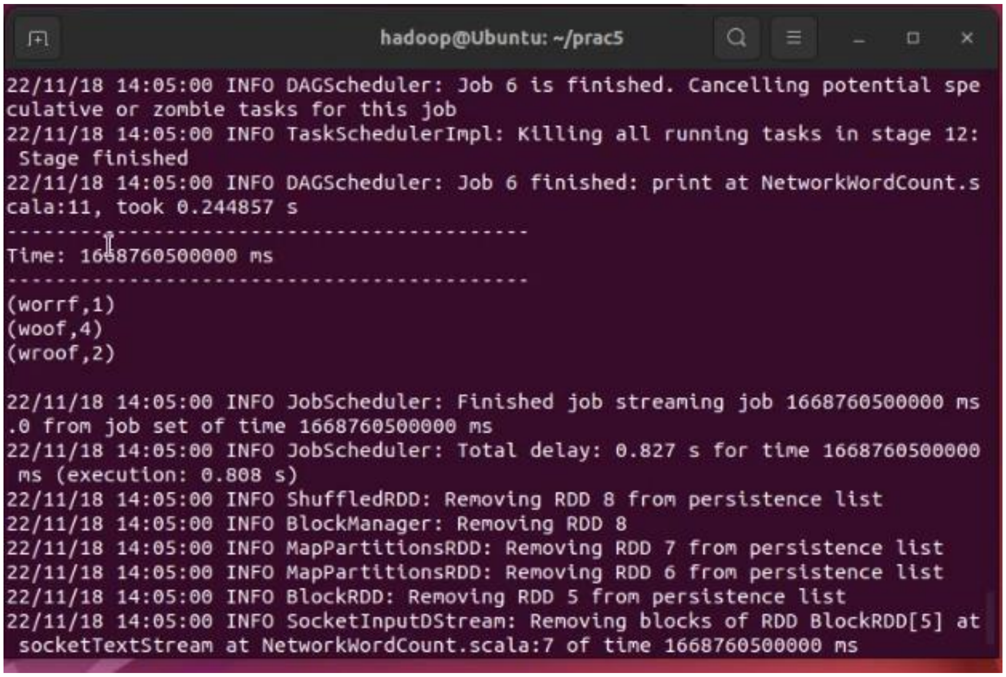


```
andy@Ubuntu:~$ sudo su hadoop
[sudo] password for andy:
hadoop@Ubuntu:/home/andy$ nc -lk 9999
meow meow meow meow meow meow meow
woof woof woof woof wroof worrf wroof
```



```
22/11/18 14:04:51 INFO TaskSchedulerImpl: Removed TaskSet 8.0, whose tasks have
all completed, from pool
22/11/18 14:04:51 INFO DAGScheduler: ResultStage 8 (print at NetworkWordCount.sc
ala:11) finished in 0.178 s
22/11/18 14:04:51 INFO DAGScheduler: Job 4 is finished. Cancelling potential spe
culative or zombie tasks for this job
22/11/18 14:04:51 INFO TaskSchedulerImpl: Killing all running tasks in stage 8:
Stage finished
22/11/18 14:04:51 INFO DAGScheduler: Job 4 finished: print at NetworkWordCount.s
cala:11, took 0.275480 s
-------------------------------------------
Time: 1668760490000 ms
-------------------------------------------
(meow,7)

22/11/18 14:04:51 INFO JobScheduler: Finished job streaming job 1668760490000 ms
.0 from job set of time 1668760490000 ms
22/11/18 14:04:51 INFO JobScheduler: Total delay: 1.227 s for time 1668760490000
 ms (execution: 1.180 s)
22/11/18 14:04:51 INFO ShuffledRDD: Removing RDD 4 from persistence list
22/11/18 14:04:51 INFO MapPartitionsRDD: Removing RDD 3 from persistence list
22/11/18 14:04:51 INFO BlockManager: Removing RDD 4
22/11/18 14:04:51 INFO MapPartitionsRDD: Removing RDD 2 from persistence list
22/11/18 14:04:51 INFO BlockManager: Removing RDD 3
```



```
22/11/18 14:05:00 INFO DAGScheduler: Job 6 is finished. Cancelling potential spe
culative or zombie tasks for this job
22/11/18 14:05:00 INFO TaskSchedulerImpl: Killing all running tasks in stage 12:
 Stage finished
22/11/18 14:05:00 INFO DAGScheduler: Job 6 finished: print at NetworkWordCount.s
cala:11, took 0.244857 s
-------------------------------------------
Time: 1668760500000 ms
-------------------------------------------
(worrf,1)
(woof,4)
(wroof,2)

22/11/18 14:05:00 INFO JobScheduler: Finished job streaming job 1668760500000 ms
.0 from job set of time 1668760500000 ms
22/11/18 14:05:00 INFO JobScheduler: Total delay: 0.827 s for time 1668760500000
 ms (execution: 0.808 s)
22/11/18 14:05:00 INFO ShuffledRDD: Removing RDD 8 from persistence list
22/11/18 14:05:00 INFO BlockManager: Removing RDD 8
22/11/18 14:05:00 INFO MapPartitionsRDD: Removing RDD 7 from persistence list
22/11/18 14:05:00 INFO MapPartitionsRDD: Removing RDD 6 from persistence list
22/11/18 14:05:00 INFO BlockRDD: Removing RDD 5 from persistence list
22/11/18 14:05:00 INFO SocketInputDStream: Removing blocks of RDD BlockRDD[5] at
 socketTextStream at NetworkWordCount.scala:7 of time 1668760500000 ms
```

Practical No. 6

Aim: Install Hive and use Hive Create and store structured databases.

Theory: Apache Hive is a data warehouse software project built on top of Apache Hadoop for providing data query and analysis. Hive gives an SQL-like interface to query data stored in various databases and file systems that integrate with Hadoop. Traditional SQL queries must be implemented in the MapReduce Java API to execute SQL applications and queries over distributed data. Hive provides the necessary SQL abstraction to integrate SQL-like queries (HiveQL) into the underlying Java without the need to implement queries in the low-level Java API. Since most data warehousing applications work with SQL-based querying languages, Hive aids portability of SQL-based applications to Hadoop.

Implementation:

# Hive installation

1. Download hive-2.3.9 from the given link https://downloads.apache.org/hive/
2. Extract, rename and move the downloaded zip file to the appropriate folder
3. Edit the .bashrc file $ nano ~/.bashrc
   ```
   export HIVE_HOME=/home/hadoop/hive
   export PATH=$PATH:$HIVE_HOME/bin
   ```
4. Edit the core-site.xml and add the following properties within the existing Hadoop configuration
   ```
   $ nano $HADOOP_HOME/etc/Hadoop/core-site.xml
   <property>
   <name>hadoop.proxyuser.hadoop.groups</name>
   <value>*</value>
   </property>
   <property>
   <name>hadoop.proxyuser.hadoop.hosts</name>
   <value>*</value>
   </property>
   <property>
   <name>hadoop.proxyuser.server.hosts</name>
   <value>*</value>
   </property>
   <property>
   <name>hadoop.proxyuser.server.groups</name>
   <value>*</value>
   </property>
   ```
5. Make the hdfs directory

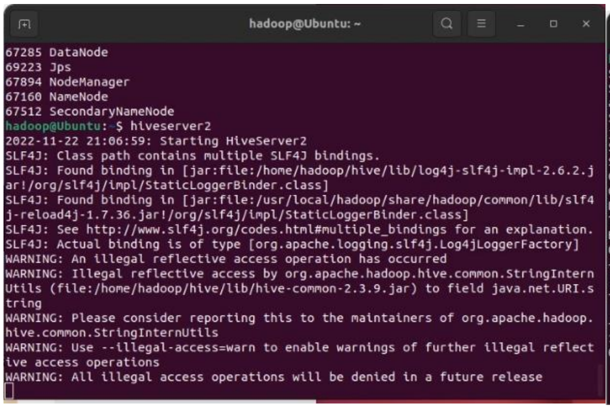
```
$ hadoop fs –mkdir /tmp
$ hadoop fs –mkdir /tmp/user
$ hadoop fs –mkdir /tmp/user/hive
$ hadoop fs –mkdir /tmp/user/hive/warehouse
```

6. Give the permissions
   ```
   $ hadoop fs -chmod g+w /tmp
   $ hadoop fs -chmod g+w tmp/user/hive/warehouse
   ```
7. Initialize the derby database
   ```
   $ schematool -dbType derby -initSchema
   ```
8. Start the hiveserver2
   ```
   $ hiveserver2
   ```
9. Open a new terminal and connect beeline with hive server
   ```
   $ beeline -n hadoop -u jdbc:hive2://localhost:10000
   ```

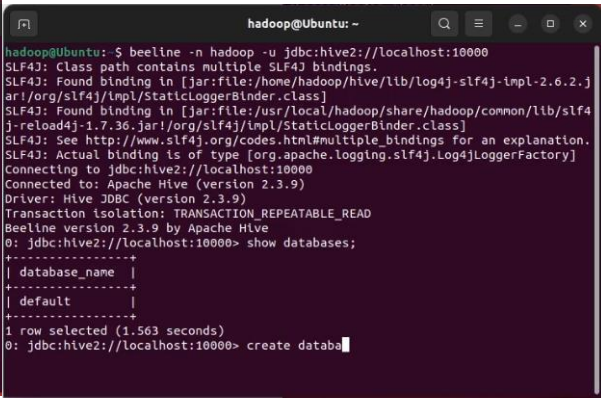
# Create and store data

1. Create a new database
   ```
   $ create database test;
   # verify with
   $ show databases;
   ```
2. Create a new table
   ```
   $ create table test.emp (id int, name string);
   ```
3. Insert a few tuples/records in the created table
   ```
   $ insert into test.emp VALUES(1, ‘Andy’);
   $ insert into test.emp VALUES(2, ‘Ramy’);
   $ insert into test.emp VALUES(3, ‘Youka’);
   ```
4. Display the table data
   ```
   $ select * from test.emp
   ```

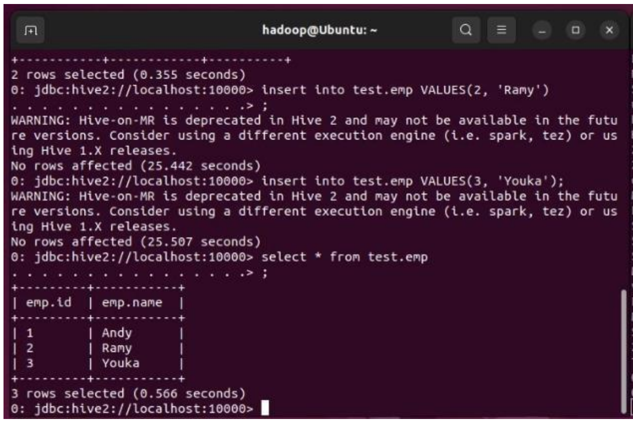
Output:



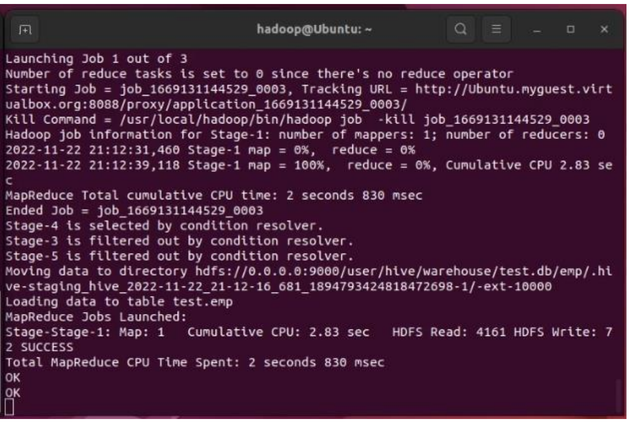
```
67285 DataNode
69223 Jps
67894 NodeManager
67160 NameNode
67512 SecondaryNameNode
hadoop@Ubuntu:~$ hiveserver2
2022-11-22 21:06:59: Starting HiveServer2
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/hadoop/hive/lib/log4j-slf4j-impl-2.6.2.j
ar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/hadoop/share/hadoop/common/lib/slf4
j-reload4j-1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.hadoop.hive.common.StringIntern
Utils (file:/home/hadoop/hive/lib/hive-common-2.3.9.jar) to field java.net.URI.s
tring
WARNING: Please consider reporting this to the maintainers of org.apache.hadoop.
hive.common.StringInternUtils
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflect
ive access operations
WARNING: All illegal access operations will be denied in a future release
```



```
hadoop@Ubuntu:~$ beeline -n hadoop -u jdbc:hive2://localhost:10000
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/hadoop/hive/lib/log4j-slf4j-impl-2.6.2.j
ar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/hadoop/share/hadoop/common/lib/slf4
j-reload4j-1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Connecting to jdbc:hive2://localhost:10000
Connected to: Apache Hive (version 2.3.9)
Driver: Hive JDBC (version 2.3.9)
Transaction isolation: TRANSACTION_REPEATABLE_READ
Beeline version 2.3.9 by Apache Hive
0: jdbc:hive2://localhost:10000> show databases;
+----------------+
| database_name  |
+----------------+
| default        |
+----------------+
1 row selected (1.563 seconds)
0: jdbc:hive2://localhost:10000> create databa
```



```
+-----------+-----------+----------+
2 rows selected (0.355 seconds)
0: jdbc:hive2://localhost:10000> insert into test.emp VALUES(2, 'Ramy')
. . . . . . . . . . . . . . . . .> ;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the futu
re versions. Consider using a different execution engine (i.e. spark, tez) or us
ing Hive 1.X releases.
No rows affected (25.442 seconds)
0: jdbc:hive2://localhost:10000> insert into test.emp VALUES(3, 'Youka');
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the futu
re versions. Consider using a different execution engine (i.e. spark, tez) or us
ing Hive 1.X releases.
No rows affected (25.507 seconds)
0: jdbc:hive2://localhost:10000> select * from test.emp
. . . . . . . . . . . . . . . . .> ;
+---------+----------+
| emp.id  | emp.name  |
+---------+----------+
| 1       | Andy      |
| 2       | Ramy      |
| 3       | Youka     |
+---------+----------+
3 rows selected (0.566 seconds)
0: jdbc:hive2://localhost:10000>
```



```
Launching Job 1 out of 3
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1669131144529_0003, Tracking URL = http://Ubuntu.myguest.virt
ualbox.org:8088/proxy/application_1669131144529_0003/
Kill Command = /usr/local/hadoop/bin/hadoop job  -kill job_1669131144529_0003
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2022-11-22 21:12:31,460 Stage-1 map = 0%,  reduce = 0%
2022-11-22 21:12:39,118 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.83 se
c
MapReduce Total cumulative CPU time: 2 seconds 830 msec
Ended Job = job_1669131144529_0003
Stage-4 is selected by condition resolver.
Stage-3 is filtered out by condition resolver.
Stage-5 is filtered out by condition resolver.
Moving data to directory hdfs://0.0.0.0:9000/user/hive/warehouse/test.db/emp/.hi
ve-staging_hive_2022-11-22_21-12-16_681_1894793424818472698-1/-ext-10000
Loading data to table test.emp
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1   Cumulative CPU: 2.83 sec   HDFS Read: 4161 HDFS Write: 7
2 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 830 msec
OK
OK
```

Practical No.7

Aim: Install HBase and use the HBase Data model Store and retrieve data.

Theory: HBase is an open-source non-relational distributed database modeled after Google's Bigtable and written in Java. It is developed as part of Apache Software Foundation's Apache Hadoop project and runs on top of HDFS (Hadoop Distributed File System) or Alluxio, providing Bigtable-like capabilities for Hadoop. That is, it provides a fault-tolerant way of storing large quantities of sparse data (small amounts of information caught within a large collection of empty or unimportant data, such as finding the 50 largest items in a group of 2 billion records, or finding the non-zero items representing less than 0.1% of a huge collection).

Implementation:

#HBase installation

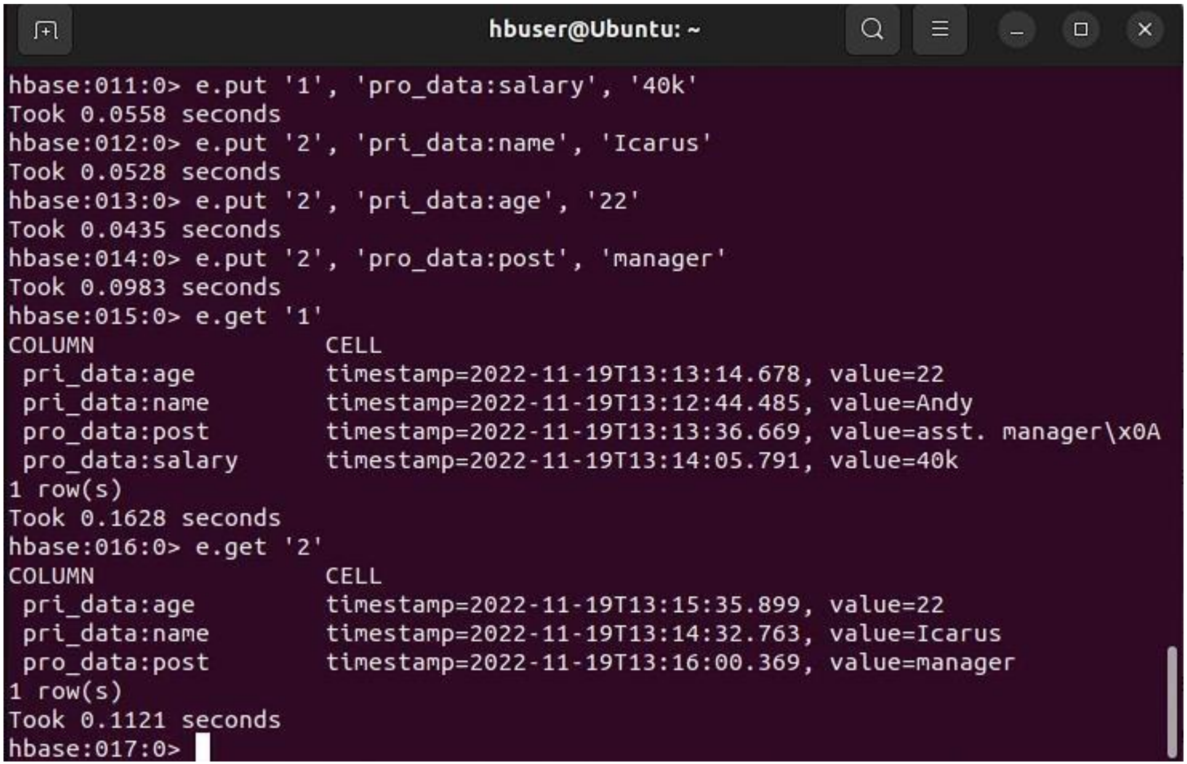
1. Go to the official site and download the most stable version of hbase
2. Extract the zip file and place it in Hadoop home directory
3. Open hbase-env.sh in hbase/conf and assign the JAVA_HOME path $ sudo nano hbase/conf/hbase-env.sh
   ```
   export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
   ```
4. Edit the .bashrc file $ nano ~/.basrhc
   ```
   export HBASE_HOME=/home/hbuser/hbase
   export PATH= $PATH:$HBASE_HOME/bin
   ```
5. Read the edited bashrc file to the running memory
   ```
   $ source ~/.bashrc
   ```
6. Add the following properties below the existing ones in the hbase/conf/hbase-site.xml file
   ```
   <property>
           <name>hbase.rootdir</name>
           <value>file:///home/hbuser/hbase</value>
   </property>
   <property>
           <name>hbase.zookeeper.property.dataDir</name>
           <value>/home/hbuser/hbase</value>
   </property>
   ```
7. Run hbase by typing
   ```
   $ start-hbase.sh
   $ hbase shell
   ```

# Storing and retrieval of data

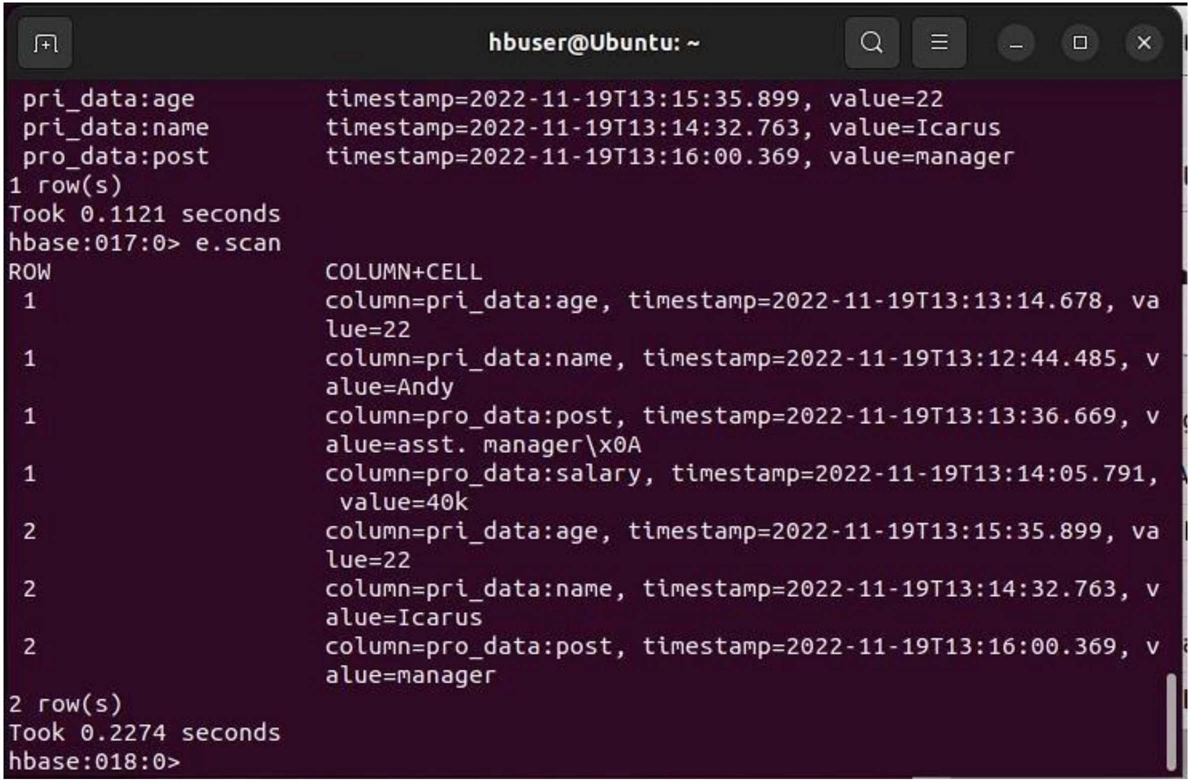
# Enter the following command in the hbase shell

- Create ‘emp’, ‘pri_data’, ‘pro_data’
- Put ‘emp’, ‘1’, ‘pri_data:name’, ‘Andy’
- e =get_table
- e.put ‘1’, ‘pri_data:age’, ‘22’
- e.put ‘1’, ‘pro_data:post’, ‘asst. manager’
- e.put ‘1’, ‘pro_data:salary’, ‘40k’
- e.put ‘2’, ‘pri_data:name’, ‘Icarus’
- e.put ‘2’, ‘pri_data:age’, ‘22’
- e.put ‘2’, ‘pro_data:post’, ‘manager’
- e.get ‘1’
- e.get ‘2’

Output:



```
hbase:011:0> e.put '1', 'pro_data:salary', '40k'
Took 0.0558 seconds
hbase:012:0> e.put '2', 'pri_data:name', 'Icarus'
Took 0.0528 seconds
hbase:013:0> e.put '2', 'pri_data:age', '22'
Took 0.0435 seconds
hbase:014:0> e.put '2', 'pro_data:post', 'manager'
Took 0.0983 seconds
hbase:015:0> e.get '1'
COLUMN                 CELL
 pri_data:age          timestamp=2022-11-19T13:13:14.678, value=22
 pri_data:name         timestamp=2022-11-19T13:12:44.485, value=Andy
 pro_data:post         timestamp=2022-11-19T13:13:36.669, value=asst. manager\x0A
 pro_data:salary       timestamp=2022-11-19T13:14:05.791, value=40k
1 row(s)
Took 0.1628 seconds
hbase:016:0> e.get '2'
COLUMN                 CELL
 pri_data:age          timestamp=2022-11-19T13:15:35.899, value=22
 pri_data:name         timestamp=2022-11-19T13:14:32.763, value=Icarus
 pro_data:post         timestamp=2022-11-19T13:16:00.369, value=manager
1 row(s)
Took 0.1121 seconds
hbase:017:0>
```



```
 pri_data:age          timestamp=2022-11-19T13:15:35.899, value=22
 pri_data:name         timestamp=2022-11-19T13:14:32.763, value=Icarus
 pro_data:post         timestamp=2022-11-19T13:16:00.369, value=manager
1 row(s)
Took 0.1121 seconds
hbase:017:0> e.scan
ROW                    COLUMN+CELL
 1                     column=pri_data:age, timestamp=2022-11-19T13:13:14.678, va
                       lue=22
 1                     column=pri_data:name, timestamp=2022-11-19T13:12:44.485, v
                       alue=Andy
 1                     column=pro_data:post, timestamp=2022-11-19T13:13:36.669, v
                       alue=asst. manager\x0A
 1                     column=pro_data:salary, timestamp=2022-11-19T13:14:05.791,
                        value=40k
 2                     column=pri_data:age, timestamp=2022-11-19T13:15:35.899, va
                       lue=22
 2                     column=pri_data:name, timestamp=2022-11-19T13:14:32.763, v
                       alue=Icarus
 2                     column=pro_data:post, timestamp=2022-11-19T13:16:00.369, v
                       alue=manager
2 row(s)
Took 0.2274 seconds
hbase:018:0>
```

Practical No. 8

Aim: Write a Pig Script for solving counting problems

Theory: Apache Pig is a high-level platform for creating programs that run on Apache Hadoop. The language for this platform is called Pig Latin. Pig can execute its Hadoop jobs in MapReduce, Apache Tez, or Apache Spark. Pig Latin abstracts the programming from the Java MapReduce idiom into a notation which makes MapReduce programming high level, similar to that of SQL for relational database management systems.

Implementation:

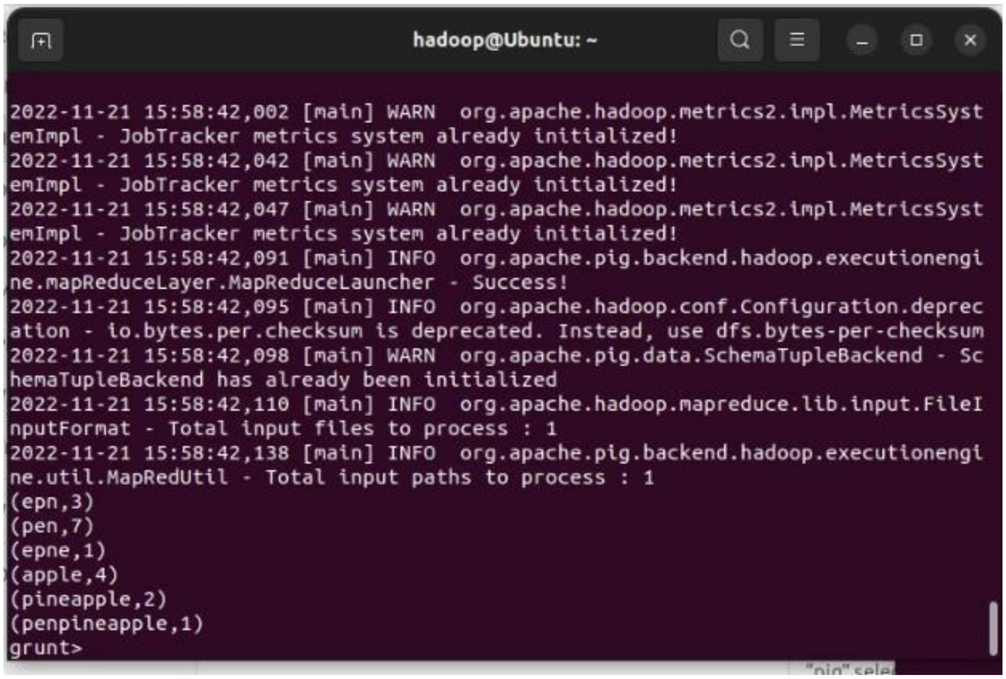
# Pig installation

1. Download the zip file from the official Pig release site
2. Extract the pig zip
   ```
   $ sudo tar –xf pig-0-17.0.tar.gx
   ```
3. Moving the file
   ```
   $ mv pig-0-17-0 /home/Hadoop/pig
   ```
4. Set the .bashrc file export PIG_HOME=/home/hadoop/pig export PATH =$PATH:/home/hadoop/pig/bin
   ```
   export PIG_CLASSPATH=$HADOOP_HOME/conf
   ```
5. Verify installation
   ```
   $ pig –version
   ```
6. Run pig locally
   ```
   $ pig –x local
   ```

# Pig script for word count problem

1. Create a text file in /home/Hadoop/textfile.txt path and provide content
   ```
   $ nano /home/hadoop/textfile.txt
   ```
2. Enter the following command in the terminal lines = LOAD '/home/hadoop/textfile.txt' AS (line:chararray); words = FOREACH lines GENERATE FLATTEN(TOKENIZE(line)) as word; grouped = GROUP words BY word;
   ```
   wordcount = FOREACH grouped GENERATE group,
   COUNT(words); DUMP wordcount;
   ```

Output:



```
2022-11-21 15:58:42,002 [main] WARN  org.apache.hadoop.metrics2.impl.MetricsSyst
emImpl - JobTracker metrics system already initialized!
2022-11-21 15:58:42,042 [main] WARN  org.apache.hadoop.metrics2.impl.MetricsSyst
emImpl - JobTracker metrics system already initialized!
2022-11-21 15:58:42,047 [main] WARN  org.apache.hadoop.metrics2.impl.MetricsSyst
emImpl - JobTracker metrics system already initialized!
2022-11-21 15:58:42,091 [main] INFO  org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MapReduceLauncher - Success!
2022-11-21 15:58:42,095 [main] INFO  org.apache.hadoop.conf.Configuration.deprec
ation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2022-11-21 15:58:42,098 [main] WARN  org.apache.pig.data.SchemaTupleBackend - Sc
hemaTupleBackend has already been initialized
2022-11-21 15:58:42,110 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileI
nputFormat - Total input files to process : 1
2022-11-21 15:58:42,138 [main] INFO  org.apache.pig.backend.hadoop.executionengi
ne.util.MapRedUtil - Total input paths to process : 1
(epn,3)
(pen,7)
(epne,1)
(apple,4)
(pineapple,2)
(penpineapple,1)
grunt>
```