

UNIVERSITY OF MUMBAI
DEPARTMENT OF COMPUTER SCIENCE



M.Sc. Computer Science – Semester III

Big Data Analytics

JOURNAL

2023-2024

Seat No. _____



मुंबई विद्यापीठ
University of Mumbai
Re-accredited with A++ Grade
(CGPA 3.65) by NAAC (3rd Cycle 2021)



UNIVERSITY OF MUMBAI
DEPARTMENT OF COMPUTER SCIENCE

CERTIFICATE

This is to certify that the work entered in this journal was done in the University Department of Computer Science laboratory by Mr./Ms. _____

Seat No. _____ for the course of M.Sc. Computer Science - Semester III (CBCS) (Revised) during the academic year 2023- 2024 in a satisfactory manner.

Subject In-charge

Head of Department

External Examiner

Index

Sr. no.	Name of the practical	Page No.	Date	Sign
1	Installing and setting environment variables for Working with Apache Hadoop.	1		
2	Implementing Map-Reduce Program for Word Count problem	9		
3	Download and install Spark. Create Graphical data and access the graphical data using Spark.	16		
4	Write a Spark code for the given application and handle error and recovery of data.	20		
5	Write a Spark code to Handle the Streaming of data.	23		
6	Install Hive and use Hive Create and store structured databases	26		
7	Install HBase and use the HBase Data model Store and retrieve data.	28		
8	Write a Pig Script for solving counting problems	31		

PRACTICAL 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. In this article, you will learn how to install and configure Apache Hadoop on Ubuntu

Implementation:

Prerequisites

- Deploy a fully updated Version of Ubuntu

1. Install Java

Install the latest version of Java.

```
$ sudo apt install default-jdk default-jre -y
```

Verify the installed version of Java.

```
$ java -version
```

2. Create Hadoop User and Configure Password-less SSH

Add a new user hadoop.

```
$ sudo adduser hadoop
```

Add the hadoop user to the sudo group.

```
$ sudo usermod -aG sudo hadoop
```

Switch to the created user.

```
$ sudo su - hadoop
```

Install the OpenSSH server and client.

```
$ sudo apt install openssh-server openssh-client -y
```

When you get a prompt, respond with:

keep the local version currently installed

Switch to the created user.

```
$ sudo su - hadoop
```

Generate public and private key pairs.

```
$ ssh-keygen -t rsa
```

(don't add password just press enter as many times to reach normal prompt)

Add the generated public key from id_rsa.pub to authorized_keys.

```
$ sudo cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

Change the permissions of the authorized_keys file.

```
$ sudo chmod 640 ~/.ssh/authorized_keys
```

Verify if the password-less SSH is functional.

```
$ ssh localhost
```

3. Install Apache Hadoop

Log in with hadoop user.

```
$ sudo su - hadoop
```

Download the latest stable version of Hadoop. To get the latest version, go to Apache Hadoop

official download page.

```
$ wget https://downloads.apache.org/hadoop/common/hadoop-3.3.1/hadoop-3.3.1.tar.gz
```

Extract the downloaded file.

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

Move the extracted directory to the /usr/local/ directory.

```
$ sudo mv hadoop-3.3.1 /usr/local/hadoop
```

Create directory to store system logs.

```
$ sudo mkdir /usr/local/hadoop/logs
```

Change the ownership of the hadoop directory.

```
$ sudo chown -R hadoop:hadoop /usr/local/hadoop
```

4. Configure Hadoop

Edit file ~/.bashrc to configure the Hadoop environment variables.

```
$ sudo nano ~/.bashrc
```

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"
```

Activate the environment variables.

```
$ source ~/.bashrc
```

5. Configure Java Environment Variables

Hadoop has a lot of components that enable it to perform its core functions. To configure these

components such as YARN, HDFS, MapReduce, and Hadoop-related project settings, you need to

define Java environment variables in `hadoop-env.sh` configuration file.

Find the Java path.

```
$ which javac
```

Find the OpenJDK directory.

```
$ readlink -f /usr/bin/javac
```

Edit the `hadoop-env.sh` file.

```
$ sudo nano $HADOOP_HOME/etc/hadoop/hadoop-env.sh
```

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"
```

Browse to the hadoop lib directory.

```
$ cd /usr/local/hadoop/lib
```

Download the Javac activation file.

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

Verify the Hadoop version.

```
$ hadoop version
```

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

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

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>
```

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
```

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

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

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

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

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>
```

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

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


Add the following lines. Save and close the file.

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

Log in with hadoop user.

```
$ sudo su - hadoop
```

Validate the Hadoop configuration and format the HDFS NameNode.

```
$ hdfs namenode -format
```

6. Start the Apache Hadoop Cluster

Start the NameNode and DataNode.

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

Start the YARN resource and node managers.

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

Verify all the running components.

```
$ jps
```

7. Access Apache Hadoop Web Interface

You can access the Hadoop NameNode and DataNode on your browser via <http://localhost:9870>.

For example:

<http://localhost:9870>

<http://localhost:8088>

OUTPUT :

Overview '0.0.0.0:9000' (✓active)

Started:	Sun Dec 03 18:07:50 +0530 2023
Version:	3.3.1, ra3b9c37a397ad4188041dd80621bdeefc46885f2
Compiled:	Tue Jun 15 10:43:00 +0530 2021 by ubuntu from (HEAD detached at release-3.3.1-RC3)
Cluster ID:	CID-bb78bfac-0600-4797-936f-143262cb661b
Block Pool ID:	BP-1883256055-127.0.1.1-1693208411484

Summary

Security is off.
Safemode is off.

32 files and directories, 15 blocks (15 replicated blocks, 0 erasure coded block groups) = 47 total filesystem object(s).

Heap Memory used 48.55 MB of 144 MB Heap Memory. Max Heap Memory is 490 MB.

Non Heap Memory used 63.13 MB of 66.06 MB Committed Non Heap Memory. Max Non Heap Memory is <unbounded>.

Configured Capacity:	19.02 GB
-----------------------------	----------

Datanode Information

✓ In service ⚠ Down ⚙ Decommissioning ⚠ Decommissioned ⚠ Decommissioned & dead
 ⚙ Entering Maintenance ⚠ In Maintenance ⚠ In Maintenance & dead

Datanode usage histogram

Disk usage of each DataNode (%)


In operation

DataNode State: All Show: 25 entries Search:

Activities Firefox Web Browser Dec 3 18:12

All Applications

localhost:8088/cluster



Cluster

- About
- Nodes
- Node Labels
- Applications
- NEW
- NEW SAVING
- SUBMITTED
- ACCEPTED
- RUNNING
- FINISHED
- FAILED
- KILLED
- Scheduler

Tools

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running
0	0	0	0	0

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decom
1	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Mi
Capacity Scheduler	[memory-mb (unit=Mi), vcores]	<memory:1024, vCore

Show 20 entries

ID	User	Name	Application Type	Application Tags	Queue	Application Priority	StartTime	LaunchTime	Finis
Showing 0 to 0 of 0 entries									

Practical no-2

Aim: Implementing Map-Reduce Program for Word Count Problem.

Theory: MapReduce in Hadoop is a software framework for ease in writing applications of software, processing huge amounts of data. MapReduce provides the facility to distribute the workload (computations) among various nodes(analogous to commodity hardware). Hence, reducing the processing time as data on which the computation needs to be done is now divided into small chunks and individually processed. Through MapReduce you can achieve parallel processing resulting in faster execution of the job

MapReduce Word Count is a framework which splits the chunk of data, sorts the map outputs and input to reduce tasks. A File-system stores the output and input of jobs. Re- execution of failed tasks, scheduling them and monitoring them is the task of the framework.

Implementation:

Word count MapReduce problem

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<LongWritable,Text,Text,IntWritable>{

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

    private Text word = new Text();

    public void map(LongWritable key, Text value,OutputCollector<Text,IntWritable>
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_Reduce.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<Text,IntWritable,Text,IntWritable> {

    public void reduce(Text key, Iterator<IntWritable>
values,OutputCollector<Text,IntWritable> output,

    Reporter reporter) throws IOException {

        int sum=0;

        while (values.hasNext()) {

```

```

sum+=values.next().get();

}

output.collect(key,new IntWritable(sum));

}

}

```

WC_Runner.java

```
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("WordCount");

        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);

    }

}

```

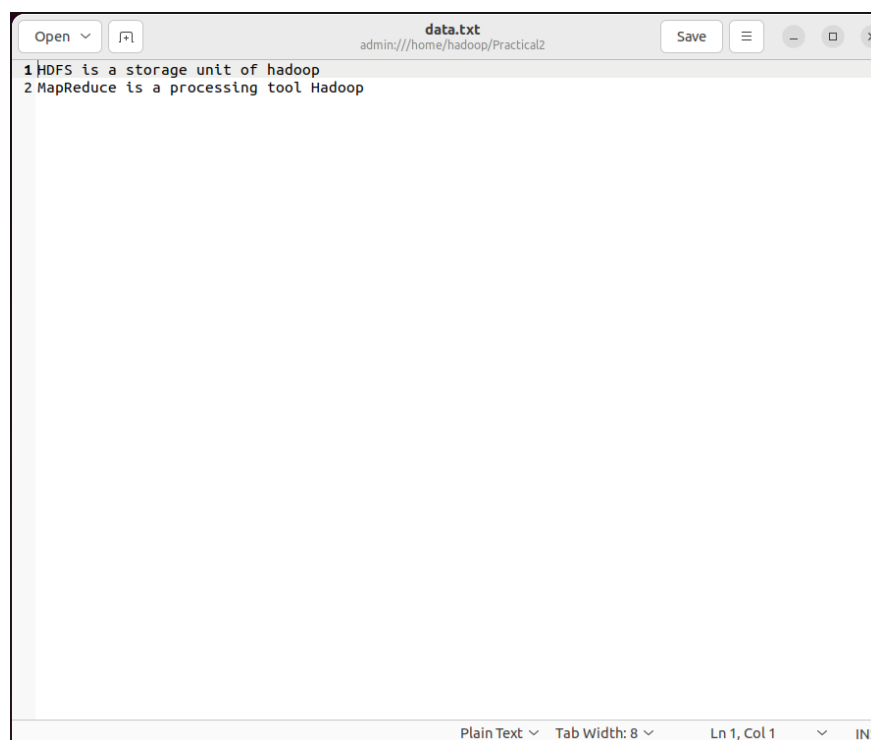
Download Hadoop-core-1.2.1.jar and put it in same folder in which we have this 3 files.

Compiling and making jar files (make sure to login as root user)

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

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

Create data.txt file using command (**nano data.txt**) and put your desired data.



Start Hadoop server and yarn server (start-dfs.sh and start-yarn.sh)

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

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

Create a folder in Hadoop and upload the data.txt file

Ubuntu [Running] - Oracle VM VirtualBox

File Machine View Input Help

browser Dec 3 18:20

localhost:9870/dfshealth.html#tab-overview

ally sends some data to Mozilla so that we can improve your experience. Choose What I Share

Overview Datanodes Datanode Volume Failures Snapshot Startup Progress Utilities

Utilities

- Browse the file system
- Logs
- Log Level
- Metrics
- Configuration
- Process Thread Dump
- Network Topology

New '0.0.0.0:9000' (active)

Sun Dec 03 18:19:25 +0530 2023
3.3.1, ra3b9c37a397ad4188041dd80621bdeefc46885f2
Tue Jun 15 10:43:00 +0530 2021 by ubuntu from (HEAD detached at release-3.3.1-RC3)
CID-bb78bfac-0600-4797-936f-143262cb661b
BP-1883256055-127.0.1.1-1693208411484

ary

ories, 15 blocks (15 replicated blocks, 0 erasure coded block groups) = 47 total filesystem object(s).

ed 57.3 MB of 99 MB Heap Memory. Max Heap Memory is 490 MB.

er.html 49.54 MB of 52.81 MB Committed Non Heap Memory. Max Non Heap Memory is <unbounded>.

Dec 3 18:22

localhost:9870/explorer.html#/

me data to Mozilla so that we can improve your experience. Choose What I Share

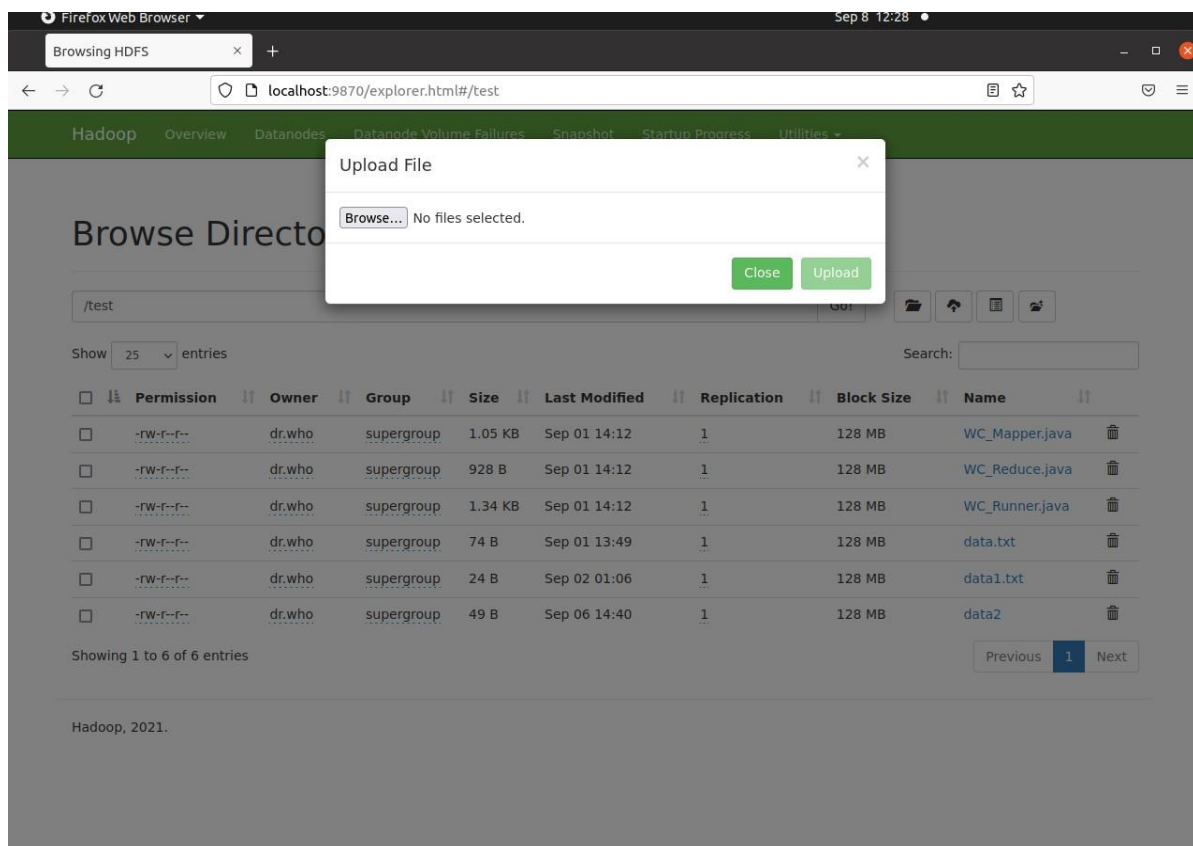
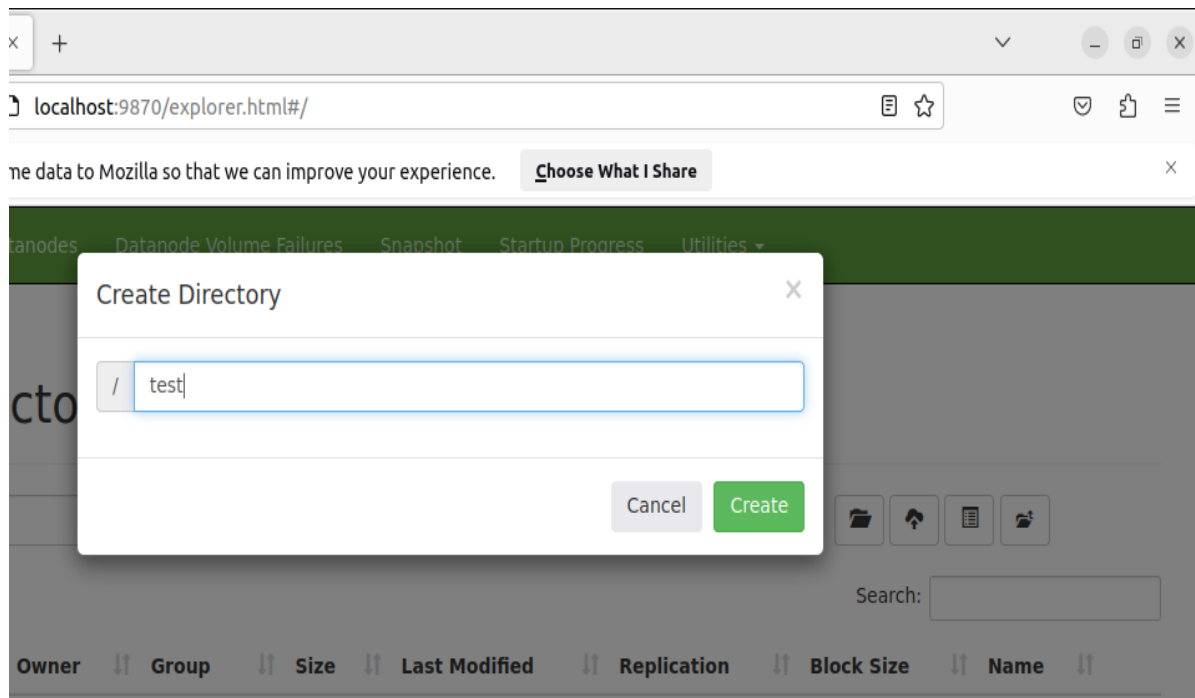
Datanodes Datanode Volume Failures Snapshot Startup Progress Utilities

Directory

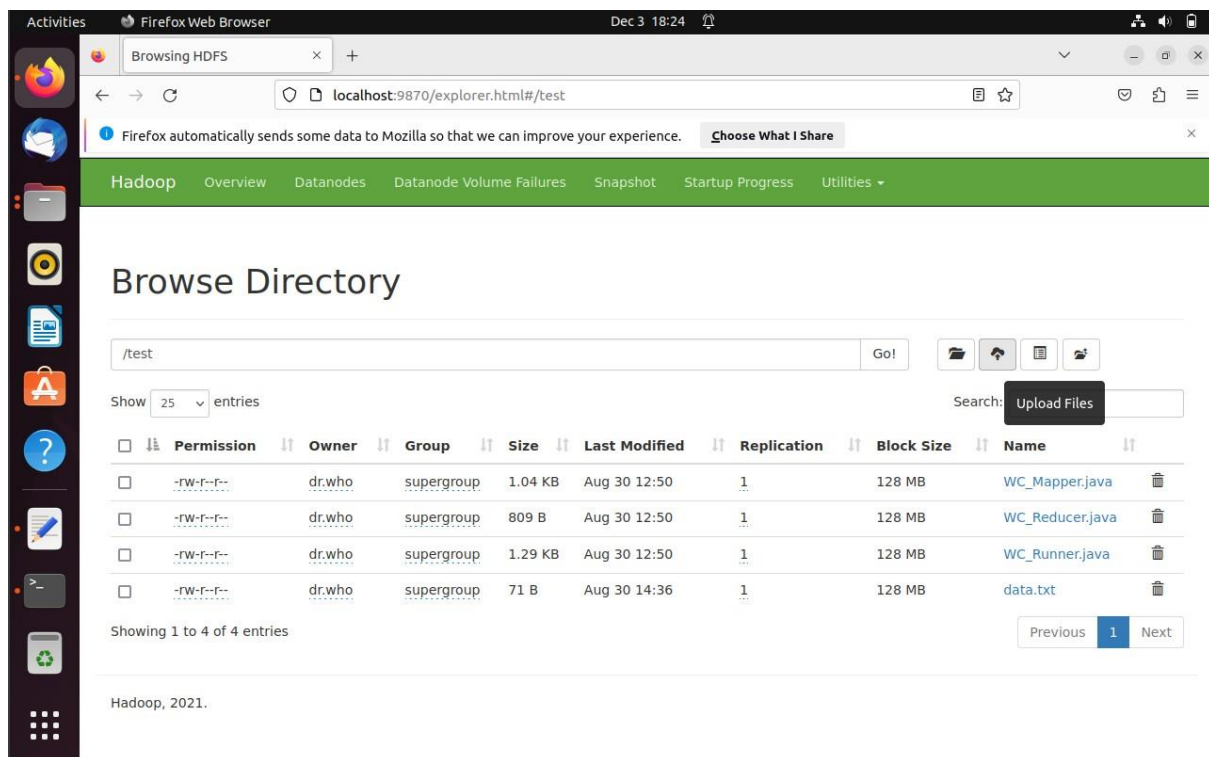
Go!

Create Directory

Owner	Group	Size	Last Modified	Replication	Block Size	Name
-------	-------	------	---------------	-------------	------------	------



Select data.txt and upload



Compiling (make sure to login as Hadoop user)

Use cd command to go to directory where wordcountproblem.jar is saved

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

Running

```
hdfs dfs -cat /r_output/part-00000
```

OUTPUT:

```
hadoop@varun:~$ hdfs dfs -cat /r_output2/part-00000
HDFS      1
Hadoop    1
MapReduce      1
a          2
hadoop      1
is          2
of          1
processing    1
storage      1
tool         1
unit         1
hadoop@varun:~$
```

Practical no 3

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

Theory: Apache Spark is a multi-language engine for executing data engineering, data science, and machine learning on single-node machines or clusters. Spark comes packaged with higher-level libraries, including support for SQL queries, streaming data, machine learning and graph processing. These standard libraries increase developer productivity and can be seamlessly combined to create complex workflows.

GraphX is Apache Spark's API for graphs and graph-parallel computation. It extends the Spark RDD by introducing a new Graph abstraction: a directed multigraph with properties attached to each vertex and edge.

GraphX library provides graph operators like subgraph, joinVertices, and aggregateMessages to transform the graph data. It provides several ways of building a graph from a collection of vertices and edges in an RDD or on disk. GraphX also includes a number of graph algorithms and builders to perform graph analytics tasks.

Implementation:

(Make sure java is installed)

Install Apache Spark

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

<https://www.scala-sbt.org/1.x/docs/Installing-sbt-on-Linux.html>

```
sudo apt-get update
sudo apt-get install apt-transport-https curl gnupg -yqq
echo "deb https://repo.scala-sbt.org/scalasbt/debian all main" | sudo tee
/etc/apt/sources.list.d/sbt.list
echo "deb https://repo.scala-sbt.org/scalasbt/debian /" | sudo tee
/etc/apt/sources.list.d/sbt_old.list
curl -sL
"https://keyserver.ubuntu.com/pks/lookup?op=get&search=0x2EE0EA64E40A89B84B2DF
73499E82A75642AC823" | sudo -H gpg --no-default-keyring --keyring gnupg-
ring:/etc/apt/trusted.gpg.d/scalasbt-release.gpg --import
sudo chmod 644 /etc/apt/trusted.gpg.d/scalasbt-release.gpg
sudo apt-get update
sudo apt-get install sbt
```

Download Apache Spark. Find the latest release from the [downloads page](#).
Extract the Spark (make sure to be in the directory the spark is downloaded)

\$ sudo tar xvf spark-3.2.0-bin-hadoop3.2.tgz (Make sure to be in the folder where spark is downloaded)

Create an installation directory /opt/spark.

```
$ sudo mkdir /opt/spark
```

Move the extracted files to the installation directory.

```
$ sudo mv spark-3.2.0-bin-hadoop3.2/* /opt/spark
```

Change the permission of the directory.

```
$ sudo chmod -R 777 /opt/spark
```

Edit the bashrc configuration file to add Apache Spark installation directory to the system path.

```
$ sudo nano ~/.bashrc
```

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
```

Save the changes to take effect.

```
$ source ~/.bashrc
```

Start the standalone master server.

```
$ start-master.sh
```

Find your server hostname from the dashboard by visiting <http://localhost:8080>. Under the URL value. It might look like this:

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

```
$ start-worker.sh spark://suraj-linux:7077
```

3. Access Apache Spark Web Interface

Go to your browser address bar to access the web interface and type in <http://localhost:8080> to access the web install wizard. For example: <http://localhost:8080>

Spark Master at spark://varun.myquest.virtualbox.org:7077

URL: spark://varun.myquest.virtualbox.org:7077
 Alive Workers: 0
 Cores in use: 0 Total, 0 Used
 Memory in use: 0.0 B Total, 0.0 B Used
 Resources in use:
 Applications: 0 Running, 0 Completed
 Drivers: 0 Running, 0 Completed
 Status: ALIVE

Workers (0)

Worker Id	Address	State	Cores	Memory	Resources
-----------	---------	-------	-------	--------	-----------

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
----------------	------	-------	---------------------	------------------------	----------------	------	-------	----------

Spark Master at spark://varun.myquest.virtualbox.org:7077

URL: spark://varun.myquest.virtualbox.org:7077
 Alive Workers: 1
 Cores in use: 2 Total, 0 Used
 Memory in use: 1024.0 MiB 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-20231203225535-10.0.2.15-37043	10.0.2.15:37043	ALIVE	2 (0 Used)	1024.0 MiB (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
----------------	------	-------	---------------------	------------------------	----------------	------	-------	----------

```
$ 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
```

```
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:

```
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), (6,1), (8,3), (10,1), (2,1), (1,2), (3,1), (9,2), (5,3))

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

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

<https://www.scala-sbt.org/1.x/docs/Installing-sbt-on-Linux.html>

```
sudo apt-get update
sudo apt-get install apt-transport-https curl gnupg -yqq
echo "deb https://repo.scala-sbt.org/scalasbt/debian all main" | sudo tee
/etc/apt/sources.list.d/sbt.list
echo "deb https://repo.scala-sbt.org/scalasbt/debian /" | sudo tee
/etc/apt/sources.list.d/sbt_old.list
curl -sL
"https://keyserver.ubuntu.com/pks/lookup?op=get&search=0x2EE0EA64E40A89B84B2DF
73499E82A75642AC823" | sudo -H gpg --no-default-keyring --keyring gnupg-
ring:/etc/apt/trusted.gpg.d/scalasbt-release.gpg --import
sudo chmod 644 /etc/apt/trusted.gpg.d/scalasbt-release.gpg
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
```

```
)
```

One thing to remember is start this file from first line and after every line leave one space next line even after last line

start spark master & worker

```
$ spark-master.sh
```

Open <http://localhost:8080>

```
$ 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:

```

hadoop@varun: ~/WorkCount/WordCount
23/12/04 00:26:31 INFO TaskSetManager: Starting task 1.0 in stage 0.0 (TID 1) (1
0.0.2.15, executor 0, partition 1, PROCESS_LOCAL, 4582 bytes) taskResourceAssign
ments Map()
23/12/04 00:26:31 INFO BlockManagerInfo: Added broadcast_0_piece0 in memory on 1
0.0.2.15:43793 (size: 1898.0 B, free: 434.4 MiB)
23/12/04 00:26:31 INFO TaskSetManager: Finished task 0.0 in stage 0.0 (TID 0) in
831 ms on 10.0.2.15 (executor 0) (1/2)
23/12/04 00:26:31 WARN TaskSetManager: Lost task 1.0 in stage 0.0 (TID 1) (10.0.
2.15 executor 0): java.lang.Exception: Testing exception handling
    at ExceptionHandlingTest$.anonfun$main$1(ExceptionHandlingTest.scala:11
)
    at scala.runtime.java8.JFunction1$mcV$sp.apply(JFunction1$mcV$sp.java:
23)
    at scala.collection.Iterator.foreach(Iterator.scala:943)
    at scala.collection.Iterator.foreach$(Iterator.scala:943)
    at org.apache.spark.InterruptibleIterator.foreach(InterruptibleIterator.
scala:28)
    at org.apache.spark.rdd.RDD$.anonfun$foreach$2(RDD.scala:1012)
    at org.apache.spark.rdd.RDD$.anonfun$foreach$2$adapted(RDD.scala:1012)
    at org.apache.spark.SparkContext$.anonfun$runJob$5(SparkContext.scala:22
64)
    at org.apache.spark.scheduler.ResultTask.runTask(ResultTask.scala:90)
    at org.apache.spark.scheduler.Task.run(Task.scala:131)
    at org.apache.spark.executor.Executor$TaskRunner$.anonfun$run$3(Executor

```

Practical 5

Aim: Write a Spark code to Handle the Streaming of data.

Theory: Spark Streaming is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams. Data can be ingested from many sources like Kafka, Kinesis, or TCP sockets, and can be processed using complex algorithms expressed with high-level functions like map, reduce, join and window. Finally, processed data can be pushed out to filesystems, databases, and live dashboards. In fact, you can apply Spark's machine learning and graph processing algorithms on data streams.

Implementation:

1. Create a new folder logged in

As Hadoop user or where you have installed spark

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()

  }
}
```

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"
```

3. Create the sbt package

```
$ sbt package
```

4. Start the spark server

```
$ start-master.sh
```

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

5. On a separate terminal, start The netscape server

```
$ nc -lk 9999
```

6. On the original terminal, Submit the scala program to spark

Inside the folder where you have scala and sbt package

```
$ spark-submit --class „NetworkWordCount --master  
„spark://Ubuntu.myguest.virtualbox.org:7077/“ \target/scala2.12/networkwordcount_2.12-  
1.0.0.jar
```

Sometime try remove forward slash from ...7077/ to -> ... 7077

7. On the netscape terminal

Provide textual input for the scala program to count words of the live stream

Every 10 seconds

OUTPUT:

```

hadoop@varun: ~/WordCount
tes result sent to driver
23/12/04 00:40:20 INFO TaskSetManager: Finished task 0.0 in stage 8.0 (TID 4) in
 112 ms on 10.0.2.15 (executor driver) (1/1)
23/12/04 00:40:20 INFO TaskSchedulerImpl: Removed TaskSet 8.0, whose tasks have
all completed, from pool
23/12/04 00:40:20 INFO DAGScheduler: ResultStage 8 (print at NetworkWordCount.sc
ala:41) finished in 0.244 s
23/12/04 00:40:20 INFO DAGScheduler: Job 4 is finished. Cancelling potential spe
culative or zombie tasks for this job
23/12/04 00:40:20 INFO TaskSchedulerImpl: Killing all running tasks in stage 8:
Stage finished
23/12/04 00:40:20 INFO DAGScheduler: Job 4 finished: print at NetworkWordCount.s
cala:41, took 0.303786 s
-----
Time: 1701630620000 ms
-----

23/12/04 00:40:20 INFO JobScheduler: Finished job streaming job 1701630620000 ms
.0 from job set of time 1701630620000 ms
23/12/04 00:40:20 INFO JobScheduler: Total delay: 0.601 s for time 1701630620000
ms (execution: 0.586 s)
23/12/04 00:40:20 INFO ShuffledRDD: Removing RDD 4 from persistence list
23/12/04 00:40:20 INFO MapPartitionsRDD: Removing RDD 3 from persistence list
23/12/04 00:40:20 INFO MapPartitionsRDD: Removing RDD 2 from persistence list

It looks like you haven't started Firefox in a while. Do you want to open it?

Spark Master at spark://varun.myguest.virtualbox.org:7077
Live Workers: 0

hadoop@varun: ~/WordCount
Stage finished
23/12/04 00:40:30 INFO DAGScheduler: Job 6 finished: print at NetworkWordCount.s
cala:41, took 0.185610 s
-----
Time: 1701630630000 ms
-----

(my,1)
(name,1)
(hiii,1)

23/12/04 00:40:30 INFO JobScheduler: Finished job streaming job 1701630630000 ms
.0 from job set of time 1701630630000 ms
23/12/04 00:40:30 INFO BlockManagerInfo: Removed block 1701630630000 from BlockM
anager:45609 in memory (size: 2.9 KiB, free: 434.4 KiB)
23/12/04 00:40:30 INFO ShuffledRDD: Removing RDD 7 from persistence list
23/12/04 00:40:30 INFO JobScheduler: Total delay: 0.800 s for time 1701630630000
ms (execution: 0.838 s)
23/12/04 00:40:31 INFO MapPartitionsRDD: Removing RDD 7 from persistence list
23/12/04 00:40:31 INFO MapPartitionsRDD: Removing RDD 6 from persistence list
23/12/04 00:40:31 INFO BlockRDD: Removing RDD 5 from persistence list
23/12/04 00:40:31 INFO SocketInputDStream: Removing blocks of RDD BlockRDD[5] at
socketTextStream at NetworkWordCount.scala:25 of time 1701630630000 ms
23/12/04 00:40:31 INFO ReceivedBlockTracker: Deleting batches: 1701630610000 ms
23/12/04 00:40:31 INFO InputInfoTracker: remove old batch metadata: 1701630610000
ms

hadoop@varun: ~/WordCount
t/spark/jars/spark-unsafe_2.12-3.2.4.jar) to constructor java.nio.DirectByteBuffer(long,int)
WARNING: Please consider reporting this to the maintainers of org.apache.spark.un
safe.Platform
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
Welcome to

Spark logo
version 3.2.4

Using Scala version 2.12.15, OpenJDK 64-Bit Server VM, 11.0.20
Branch HEAD
Compiled by user centos on 2023-04-09T20:59:10Z
Revision 0ae10ac18298d1792828f1d59b652ef17462d76e
Url https://github.com/apache/spark
Type --help for more information.
hadoop@varun:~/WordCount$ cd
hadoop@varun:~$ nc -lk 9999
hiii my name

```

Submitted Time	User	State	Duration
----------------	------	-------	----------

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;
```

```
$ use test;
```

2. Create a new table

```
$ CREATE TABLE IF NOT EXISTS emp(id INT,name STRING);
```

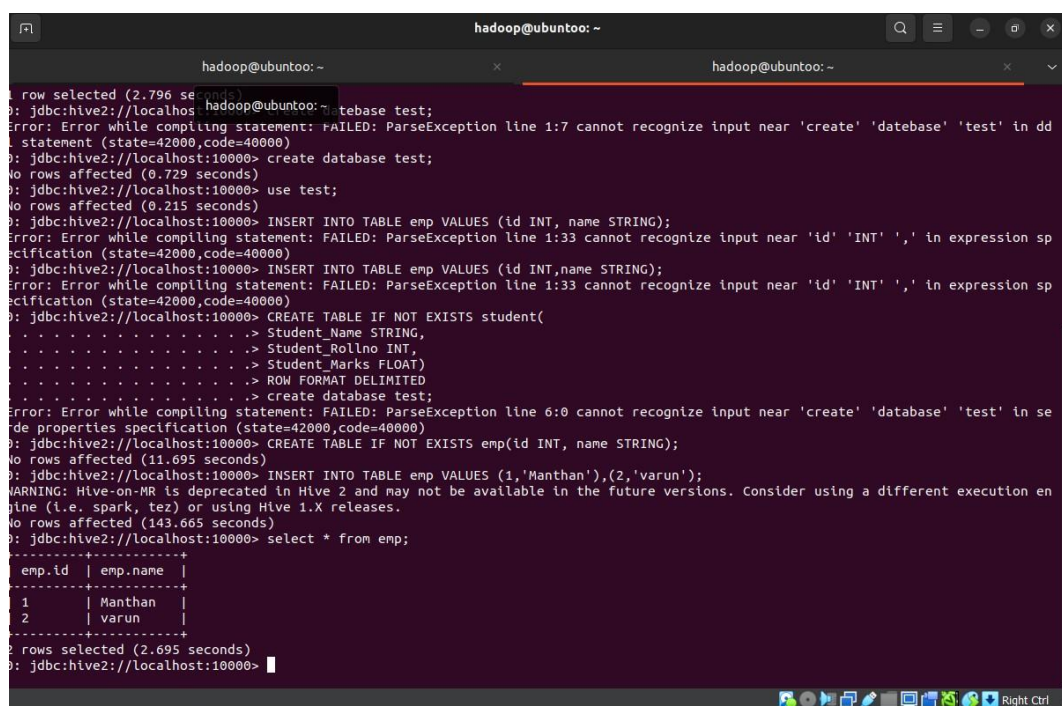
3. Insert a few tuples/records in the created table

```
$ INSERT INTO TABLE emp VALUES (1,"Manthan"),(2,"Varun");
```

4. Display the table data

```
$ select * from emp;
```

OUTPUT:



```
hadoop@ubuntu: ~
hadoop@ubuntu: ~
hadoop@ubuntu: ~
1 row selected (2.796 seconds)
hadoop@ubuntu: ~
j: jdbc:hive2://localhost:10000> create database test;
Error: Error while compiling statement: FAILED: ParseException line 1:7 cannot recognize input near 'create' 'database' 'test' in dd
statement (state=42000,code=40000)
j: jdbc:hive2://localhost:10000> create database test;
No rows affected (0.729 seconds)
j: jdbc:hive2://localhost:10000> use test;
No rows affected (0.215 seconds)
j: jdbc:hive2://localhost:10000> INSERT INTO TABLE emp VALUES (id INT, name STRING);
Error: Error while compiling statement: FAILED: ParseException line 1:33 cannot recognize input near 'id' 'INT' ',' in expression sp
ecification (state=42000,code=40000)
j: jdbc:hive2://localhost:10000> INSERT INTO TABLE emp VALUES (id INT,name STRING);
Error: Error while compiling statement: FAILED: ParseException line 1:33 cannot recognize input near 'id' 'INT' ',' in expression sp
ecification (state=42000,code=40000)
j: jdbc:hive2://localhost:10000> CREATE TABLE IF NOT EXISTS student(
. . . . .> Student_Name STRING,
. . . . .> Student_Rollno INT,
. . . . .> Student_Marks FLOAT)
. . . . .> ROW FORMAT DELIMITED
. . . . .> create database test;
Error: Error while compiling statement: FAILED: ParseException line 6:0 cannot recognize input near 'create' 'database' 'test' in se
rde properties specification (state=42000,code=40000)
j: jdbc:hive2://localhost:10000> CREATE TABLE IF NOT EXISTS emp(id INT, name STRING);
No rows affected (11.695 seconds)
j: jdbc:hive2://localhost:10000> INSERT INTO TABLE emp VALUES (1,'Manthan'),(2,'varun');
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution en
gine (i.e. spark, tez) or using Hive 1.X releases.
No rows affected (143.665 seconds)
j: jdbc:hive2://localhost:10000> select * from emp;
+----+-----+
| emp.id | emp.name |
+----+-----+
| 1      | Manthan  |
| 2      | varun    |
+----+-----+
2 rows selected (2.695 seconds)
j: jdbc:hive2://localhost:10000>
```

Practical 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. Install Java

Install the latest version of Java.

```
$ sudo apt install default-jdk default-jre -y
```

Verify the installed version of Java.

```
$ java -version
```

Create a Hbase user

```
$ sudo adduser hbuser
```

```
$ sudo usermod -aG sudo hbuser
```

```
$ sudo su - hbuser
```

```
$ sudo mkdir /home/hbuser/hbase
```

```
$ sudo chown -R hbuser:hbuser /home/hbuser/hbase
```

If you have downloaded and unzipped the hbase-2.5.5-bin.tar.gz inside "hbuser" home and extract it in

home, you will see a folder "hbase-2.5.5-bin" inside it you will see a folder "hbase-2.5.5"

Inside this folder you will find the required files so do one thing select all file inside "hbase-2.5.5" copy inside "hbase-

2.5.5bin" and after pasting everything inside "hbase-2.5.5-bin" delete that "hbase-2.5.5" folder now we have only "hbase-2.5.5-bin" and files inside it so just change the name of "hbase-2.5.5-bin" to "hbase".

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
```

Edit the .bashrc file

```
$ nano ~/.bashrc
```

```
export
```

```
HBASE_HOME=/home/hbuser/hbase
```

```
export
```

```
PATH=$PATH:$HBASE_HOME/bin
Read the edited bashrc file to the running memory
$ source ~/.bashrc
```

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>
```

while editing hbase-site.xml you will have to delete first two properties after editing hbase-site.xml should

```
<property>
  <name>hbase.rootdir</name>
  <value>file:///home/hbuser/hbase</value>
</property>
<property>
  <name>hbase.zookeeper.property.dataDir</name>
<value>/home/hbuser/hbase/zookeeper</value>
</property>
<property>
  <name>hbase.unsafe.stream.capability.enforce</name>
  <value>false</value>
</property>
```

```
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', 'Varun'
>e = get_table 'emp'
>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:002:0> create 'emp', 'pri_data', 'pro_data'
Created table emp
Took 1.6240 seconds
=> Hbase::Table - emp
hbase:003:0> put 'emp', '1', 'pri_data:name','Varun'
Took 0.6364 seconds
hbase:004:0> e = get_table 'emp'
Took 0.0010 seconds
=> Hbase::Table - emp
hbase:005:0> e.put '1', 'pri_data:age','23'
Took 0.0512 seconds
hbase:006:0> e.put '1', 'pro_data:post','Ass.Manager'
Took 0.0890 seconds
hbase:007:0> e.put '1', 'pro_data:salary','40k'
Took 0.0479 seconds
hbase:008:0> e.get '1'
COLUMN                                CELL
pri_data:age                          timestamp=2023-12-03T20:06:09.311, value=23
pri_data:name                         timestamp=2023-12-03T20:05:24.840, value=Varun
pro_data:post                         timestamp=2023-12-03T20:07:01.277, value=Ass.Manager
pro_data:salary                      timestamp=2023-12-03T20:07:25.730, value=40k
1 row(s)
Took 0.1276 seconds
hbase:009:0>
```

Practical 8

Aim: Write a Pig Script for solving counting problems.

Theory:

Pig is a high-level scripting language designed for processing and analyzing large datasets in Apache Hadoop. It was developed by Yahoo! and later contributed to the Apache Software Foundation. Pig simplifies the process of writing complex MapReduce programs by providing a higher-level abstraction over the Hadoop framework.

Implementation:

Download

<https://dlcdn.apache.org/pig/pig-0.17.0/>

extract the files

rename the extracted folder to pig and copy it to home of the Hadoop user

Configure Apache Pig

After installing Apache Pig, we have to configure it. To configure, we need to edit two files – **bashrc** and **pig.properties**.

.bashrc file

In the **.bashrc** file, set the following variables –

- **PIG_HOME** folder to the Apache Pig's installation folder,
- **PATH** environment variable to the bin folder, and
- **PIG_CLASSPATH** environment variable to the etc (configuration) folder of your Hadoop installations (the directory that contains the core-site.xml, hdfs-site.xml and mapred-site.xml files).

```
export PIG_HOME=/home/hadoop/pig
export PATH=$PATH:/home/hadoop/pig/bin export
PIG_CLASSPATH=$HADOOP_HOME/conf
```

pig.properties file

In the **conf** folder of Pig, we have a file named **pig.properties**. In the pig.properties file, you can set various parameters as given below.

pig -h properties

The following properties are supported –

Logging: verbose = true|false; default is false. This property is the same as -v switch
brief=true|false; default is false. This property is the same

as -b switch debug=OFF|ERROR|WARN|INFO|DEBUG; default is INFO.

This property is the same as -d switch aggregate.warning = true|false; default is true. If true, prints count of warnings of each type rather than logging each warning.

Performance tuning: pig.cachedbag.memusage=<mem fraction>; default is 0.2 (20% of all memory).

Note that this memory is shared across all large bags used by the application.

pig.skewedjoin.reduce.memusage=<mem fraction>; default is 0.3 (30% of all memory).

Specifies the fraction of heap available for the reducer to perform the join.

pig.exec.nocombiner = true|false; default is false.

Only disable combiner as a temporary workaround for problems. opt.multiquery = true|false; multiquery is on by default.

Only disable multiquery as a temporary workaround for problems. opt.fetch=true|false; fetch is on by default.

Scripts containing Filter, Foreach, Limit, Stream, and Union can be dumped without MR jobs.

pig.tmpfilecompression = true|false; compression is off by default. Determines whether output of intermediate jobs is compressed.

pig.tmpfilecompression.codec = lzo|gzip; default is gzip.

Used in conjunction with pig.tmpfilecompression. Defines compression type.

pig.noSplitCombination = true|false. Split combination is on by default.

Determines if multiple small files are combined into a single map.

pig.exec.mapPartAgg = true|false. Default is false.

Determines if partial aggregation is done within map phase, before records are sent to combiner.

pig.exec.mapPartAgg.minReduction=<min aggregation factor>. Default is 10.

If the in-map partial aggregation does not reduce the output num records by this factor, it gets disabled.

Miscellaneous: `exectype = mapreduce|tez|local`; default is `mapreduce`. This property is the same as `-x` switch

`pig.additional.jars.uris=<comma seperated list of jars>`. Used in place of `register` command.

`udf.import.list=<comma seperated list of imports>`. Used to avoid package names in UDF.

`stop.on.failure = true|false`; default is `false`. Set to `true` to terminate on the first error.

`pig.datetime.default.tz=<UTC time offset>`. e.g. `+08:00`. Default is the default timezone of the host.

Determines the timezone used to handle `datetime` datatype and UDFs.

Additionally, any Hadoop property can be specified.

Verifying the Installation

Verify the installation of Apache Pig by typing the version command. If the installation is successful, you will get the version of Apache Pig as shown below.

```
$ pig -version
```

```
Apache Pig version 0.15.0 (r1682971) compiled Jun
01 2015, 11:44:35
```

Create a `textfile.txt` in `/home/Hadoop/textfile.txt`

```
$ nano textfile.txt
```

run pig using following command in local mode without using `hadoop pig -x local`

Write a Pig Script for solving counting problems.

```
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:

```

hadoop@ubuntuoo: ~
hadoop@ubuntuoo: ~
hadoop@ubuntuoo: ~

Output(s):
Successfully stored 8 records in: "file:/tmp/temp439904900/tmp1008351232"

Counters:
Total records written : 8
Total bytes written : 0
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_local612133419_0001

2023-12-06 23:37:32,902 [main] WARN org.apache.hadoop.metrics2.impl.MetricsSystemImpl - JobTracker metrics system already initialized!
2023-12-06 23:37:32,907 [main] WARN org.apache.hadoop.metrics2.impl.MetricsSystemImpl - JobTracker metrics system already initialized!
2023-12-06 23:37:32,916 [main] WARN org.apache.hadoop.metrics2.impl.MetricsSystemImpl - JobTracker metrics system already initialized!
2023-12-06 23:37:32,936 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2023-12-06 23:37:32,948 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2023-12-06 23:37:32,949 [main] WARN org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend has already been initialized
2023-12-06 23:37:32,978 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input files to process : 1
2023-12-06 23:37:32,979 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(hi,1)
(im,1)
(good,1)
(this,1)
(doing,1)
(varun,1)
(sessions,1)
(practical,2)
grunt>

```