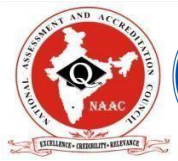
QMP 7.1 D/F



Channabasaveshwara Institute of Technology

(Affiliated to VTU, Belgavi & Approved by AICTE, New Delhi)
(NAAC Accredited & ISO 9001:2015 Certified Institution)
NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka



DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

BIG DATA ANALYTICS

&

BIS701

B.E - VII Semester Lab Manual

Academic Year: _____

Name : _____

USN : _____

Batch :_____Section : _____

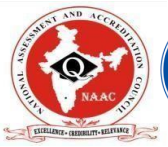
Mail -ID :_____Contact Number:

QMP 7.1 D/F



Channabasaveshwara Institute of Technology

(Affiliated to VTU, Belgavi & Approved by AICTE, New Delhi)
(NAAC Accredited & ISO 9001:2015 Certified Institution)
NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka



DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

BIG DATA ANALYTICS (BDA)

BIS701

B.E – VII Semester Lab Manual

Prepared By
Mr. Anil Kumar
Asst. Prof., Dept. of ISE

Reviewed by:
Dr. Thara D K
Prof & HOD., Dept. of ISE

Approved by:

Dr. Thara D K
Professor & HOD,
Dept. of ISE, CIT Gubbi

Channabasaveshwara Institute of Technology
(Affiliated to VTU, Belgavi & Approved by AICTE, New Delhi)
(NAAC Accredited & ISO 9001:2015 Certified Institution)
NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka

INSTITUTE VISION

To create centers of excellence in education and to serve the society by enhancing the quality of life through value based professional leadership

INSTITUTE MISSION

- To provide high quality technical and professionally relevant education in a diverse learning environment
- To provide the values that prepare students to lead their lives with personal integrity, professional ethics and civic responsibility in a global society
- To prepare the next generation of skilled professionals to successfully compete in the diverse global market
- To promote a campus environment that welcomes and honors women and men of all races, creeds and cultures, values and intellectual curiosity, pursuit of knowledge and academic integrity and freedom
- To offer a wide variety of off-campus education and training programmes to individuals and groups
- To stimulate collaborative efforts with industry, universities, government and professional societies
- To facilitate public understanding of technical issues and achieve excellence in the operations of the institute

QUALITY POLICY

Our organization delights customers (students, parents and society) by providing value added quality education to meet the national and international requirements. We also provide necessary steps to train the students for placement and continue to improve our methods of education to the students through effective quality management system, quality policy and quality objectives.

Channabasaveshwara Institute of Technology

(Affiliated to VTU, Belgavi & Approved by AICTE, New Delhi)

(NAAC Accredited & ISO 9001:2015 Certified Institution)

NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

Vision

To educate and train students by imparting strong foundational knowledge in Information Science and Engineering, while creating centres of excellence in computing education

Mission

- By providing qualified and dedicated human resources, we ensure students receive the best education in Information Science and Engineering
- By providing the state-of-the-art infrastructure, we facilitate a conducive learning environment for students pursuing excellence in the field
- By fostering collaboration with industry and research institutions, we enhance students' exposure to practical applications and cutting-edge developments
- By instilling a sense of social responsibility and promoting professional ethics, we prepare students to be ethical and socially conscious professionals in their careers.

Program Outcomes (POs)

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Engineering Tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and the world**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
8. **Individual and Collaborative team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
9. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
10. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
11. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Educational Objectives (PEOs)

After four Years of Graduation, our graduates can:

PEO1: Develop the next generation of highly skilled graduates equipped with a strong knowledge in Information Science and Engineering for creating innovative solutions to society's pressing challenges.

PEO2: To motivate students to pursue higher studies in various disciplines in Information Science and Engineering with the aim of cherishing careers in research and development, academia, industry and entrepreneurship.

PEO3: To Produce engineers who are professionals' entrepreneurs and capable of self-learning to excel in their career.

PEO4: To prepare graduates who excel in diverse learns upholding professional ethics and societal responsibilities.

Program Specific Outcomes (PSOs)

At the time of graduation, our graduates can:

PSO1: Students will possess the ability to analyze, design, develop, test and apply mathematical foundations, management principles, and adapt emerging technologies effectively in the creation of computational solutions.

PSO2: Students will possess advanced skills in artificial intelligence, cybersecurity, cloud computing, project management, mobile application development, embedded systems, and big data analytics, making them highly employable, well prepared for higher education, and capable of contributing to cutting-edge research and innovation in Information Science and Engineering.

Instructions to the Candidates

- Student should come with thorough preparation for the experiment to be conducted.
- Student should take prior permission from the concerned faculty before availing the leave.
- Student should come with proper dress code and to be present on time in the laboratory.
- Student will not be permitted to attend the laboratory unless they bring the practical record fully completed in all respects pertaining to the experiment conducted in the previous class.
- Student will not be permitted to attend the laboratory unless they bring the observation book fully completed in all respects pertaining to the experiment to be conducted in present class.
- Practical record and observation book should be neatly maintained.
- Student should obtain the signature of the staff-in-charge in the observation book after completing each experiment.
- Theory related to each experiment should be written in the practical record before procedure in your own words with appropriate references.

CONTENTS

PRACTICAL COMPONENT OF IPCC

1. Install Hadoop and Implement the following file management tasks in Hadoop:
   Adding files and directories
   Retrieving files
   Deleting files and directories.
   Hint: A typical Hadoop workflow creates data files (such as log files) elsewhere and copies them into HDFS using one of the above command line utilities.
2. Develop a MapReduce program to implement Matrix Multiplication
3. Develop a Map Reduce program that mines weather data and displays appropriate messages indicating the weather conditions of the day.
4. Develop a MapReduce program to find the tags associated with each movie by analyzing movie lens data.
5. Implement Functions: Count – Sort – Limit – Skip – Aggregate using MongoDB
6. Write Pig Latin scripts to sort, group, join, project, and filter the data.
7. Use Hive to create, alter, and drop databases, tables, views, functions, and indexes.
8. Implement a word count program in Hadoop and Spark.
9. Use CDH (Cloudera Distribution for Hadoop) and HUE (Hadoop User Interface) to analyze data and generate reports for sample datasets

COURSE OUTCOMES	
CO1:	Identify and list various Big Data concepts, tools and applications.
CO2:	Develop programs using HADOOP framework.
CO3:	Use Hadoop Cluster to deploy Map Reduce jobs, PIG,HIVE and Spark programs.
CO4:	Analyze the given data set and identify deep insights from the data set.

RBT LEVELS						
COS	**Remember (L1)**	**Understand (L2)**	**Apply (L3)**	**Analyze (L4)**	**Evaluate (L5)**	**Create (L6)**
CO1	√					
CO2			√			
CO3			√			
CO4				√		

CO – PO MAPPING														
CO-PO	**PO1**	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**	**PSO1**	**PSO2**
CO1	2	2	1	1	2				1				1	1
CO2	3	2	3	1	3				1				2	2
CO3	3	2	3	2	3				1	1			2	2
CO4	3	2	3	3	3				1	1			2	2
Average	**2.75**	**2**	**2.5**	**1.75**	**2.75**				**1**	**0.5**			**1.75**	**1.75**

BIG DATA ANALYTICS LAB

Programs, Commands and Solutions

Hadoop • MapReduce • MongoDB • Pig • Hive • Spark • CDH/HUE

1. Hadoop Installation and HDFS File Management

Aim

Install Hadoop and implement adding, retrieving, and deleting files and directories in HDFS.

Requirements

- Linux/Ubuntu environment or a Hadoop-compatible VM
- Java JDK
- Hadoop 3.x (or the version prescribed by the laboratory)
- Terminal/Command Prompt

Basic Hadoop Installation (Pseudo-Distributed Mode)

1. Install Java and verify it.
2. Download and extract Hadoop to a suitable directory.
3. Set HADOOP_HOME and add Hadoop binary directories to PATH.
4. Configure core-site.xml, hdfs-site.xml, mapred-site.xml and yarn-site.xml.
5. Format the NameNode once for a new installation.
6. Start HDFS and YARN services.
7. Verify running daemons with jps.

```
java -version
hadoop version
hdfs namenode -format
start-dfs.sh
start-yarn.sh
jps
```

Note: Exact Hadoop configuration files and service commands can vary by Hadoop distribution and installation method.

HDFS File Management

Create a local sample file:

```
mkdir -p ~/hadoop_lab/input
echo "Hadoop file management experiment" > ~/hadoop_lab/input/sample.txt
```

Create an HDFS directory:

```
hdfs dfs -mkdir -p /user/$USER/lab/input
```

Add/copy a file into HDFS:

```
hdfs dfs -put ~/hadoop_lab/input/sample.txt /user/$USER/lab/input/
# Alternative:
hdfs dfs -copyFromLocal ~/hadoop_lab/input/sample.txt /user/$USER/lab/input/
```

List files:

```
hdfs dfs -ls /user/$USER/lab/input
```

Retrieve/copy a file from HDFS to the local file system:

```
mkdir -p ~/hadoop_lab/output
hdfs dfs -get /user/$USER/lab/input/sample.txt ~/hadoop_lab/output/
# Alternative:
hdfs dfs -copyToLocal /user/$USER/lab/input/sample.txt ~/hadoop_lab/output/
```

Read a file directly from HDFS:

```
hdfs dfs -cat /user/$USER/lab/input/sample.txt
```

Delete a file:

```
hdfs dfs -rm /user/$USER/lab/input/sample.txt
```

Delete a directory and its contents:

```
hdfs dfs -rm -r /user/$USER/lab/input
```

Useful commands:

```
hdfs dfs -mkdir -p /path
hdfs dfs -ls /path
hdfs dfs -put localfile /path
hdfs dfs -get /path/file localdir
hdfs dfs -cat /path/file
hdfs dfs -rm /path/file
hdfs dfs -rm -r /path/directory
```

Result: Files and directories were successfully created, copied to HDFS, retrieved, viewed, and deleted using HDFS command-line utilities.

2. MapReduce Program for Matrix Multiplication

Aim

Develop a MapReduce program to multiply two matrices.

Logic

For A(m×n) and B(n×p), each output element C(i,j) is the dot product of row i of A and column j of B.

The mapper emits intermediate key-value pairs for every possible output position. The reducer groups values for each (i,j), multiplies matching A and B elements, and sums the products.

Java Program

```
// MatrixMultiply.java
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class MatrixMultiply {
  public static class MatrixMapper
      extends Mapper<LongWritable, Text, Text, Text> {
    public void map(LongWritable key, Text value, Context ctx)
        throws IOException, InterruptedException {
      String[] p = value.toString().trim().split(",");
      // Input format: Matrix,row,column,value
      String m=p[0]; int i=Integer.parseInt(p[1]);
      int j=Integer.parseInt(p[2]); double v=Double.parseDouble(p[3]);
      Configuration c=ctx.getConfiguration();
      int rowsA=c.getInt("rowsA",2), colsB=c.getInt("colsB",2);
      if (m.equals("A")) {
        for(int k=0;k<colsB;k++)
          ctx.write(new Text(i+","+k), new Text("A,"+j+","+v));
      } else {
        for(int k=0;k<rowsA;k++)
          ctx.write(new Text(k+","+j), new Text("B,"+i+","+v));
      }
    }
  }

  public static class MatrixReducer
      extends Reducer<Text,Text,Text,DoubleWritable> {
    public void reduce(Text key, Iterable<Text> vals, Context ctx)
        throws IOException, InterruptedException {
      java.util.Map<Integer,Double> a=new java.util.HashMap<>();
      java.util.Map<Integer,Double> b=new java.util.HashMap<>();
      for(Text t: vals){
        String[] p=t.toString().split(",");
        int k=Integer.parseInt(p[1]); double v=Double.parseDouble(p[2]);
        if(p[0].equals("A")) a.put(k,v); else b.put(k,v);
      }
      double sum=0;
      for(Integer k:a.keySet())
        if(b.containsKey(k)) sum += a.get(k)*b.get(k);
      ctx.write(key,new DoubleWritable(sum));
    }
```

```
  }

  public static void main(String[] args) throws Exception {
    Configuration conf=new Configuration();
    conf.setInt("rowsA",2); conf.setInt("colsB",2);
    Job job=Job.getInstance(conf,"Matrix Multiplication");
    job.setJarByClass(MatrixMultiply.class);
    job.setMapperClass(MatrixMapper.class);
    job.setReducerClass(MatrixReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(Text.class);
    FileInputFormat.addInputPath(job,new Path(args[0]));
    FileOutputFormat.setOutputPath(job,new Path(args[1]));
    System.exit(job.waitForCompletion(true)?0:1);
  }
}
```

Example input (matrix entries):

```
A,0,0,1
A,0,1,2
A,1,0,3
A,1,1,4
B,0,0,5
B,0,1,6
B,1,0,7
B,1,1,8
```

For A=[[1,2],[3,4]] and B=[[5,6],[7,8]], the result is C=[[19,22],[43,50]].

```
hadoop jar MatrixMultiply.jar MatrixMultiply /matrix/input /matrix/output
hdfs dfs -cat /matrix/output/part-r-00000
```

3. MapReduce Program for Weather Data

Aim

Mine weather data using MapReduce and display a message indicating the weather condition of each day.

Sample Input

```
2026-09-01,32,60
2026-09-02,24,82
2026-09-03,38,35
2026-09-04,28,70
```

Format: date,temperature_in_C,humidity_percent

Java Program

```
// WeatherMapper.java
import java.io.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;

public class WeatherMapper extends Mapper<LongWritable,Text,Text,Text>{
  public void map(LongWritable k, Text v, Context c)
      throws IOException,InterruptedException{
    String[] p=v.toString().split(",");
    double t=Double.parseDouble(p[1]);
    double h=Double.parseDouble(p[2]);
    String condition;
    if(t>=35) condition="Very Hot";
    else if(t>=30 && h<50) condition="Hot and Dry";
    else if(t<25 && h>=80) condition="Cool and Humid";
    else if(h>=80) condition="Humid";
    else if(t<25) condition="Cool";
    else condition="Pleasant";
    c.write(new Text(p[0]),new Text(condition));
  }
}

// WeatherDriver.java
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WeatherDriver{
  public static void main(String[] a)throws Exception{
    Job j=Job.getInstance(new Configuration(),"Weather Analysis");
    j.setJarByClass(WeatherDriver.class);
    j.setMapperClass(WeatherMapper.class);
    j.setNumReduceTasks(0);
    j.setOutputKeyClass(Text.class);
    j.setOutputValueClass(Text.class);
    FileInputFormat.addInputPath(j,new Path(a[0]));
    FileOutputFormat.setOutputPath(j,new Path(a[1]));
    System.exit(j.waitForCompletion(true)?0:1);
  }
}
```

Result: Each date is classified as Very Hot, Hot and Dry, Cool and Humid, Humid, Cool, or Pleasant according to the defined rules.

4. MapReduce Program to Find Tags Associated with Each Movie

Aim

Analyze MovieLens data to associate tags with each movie.

Typical MovieLens files include movies.csv, ratings.csv and tags.csv. This experiment uses tags.csv, whose fields include userId, movieId, tag and timestamp.

Mapper and Reducer

```
// MovieTagMapper.java
import java.io.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;

public class MovieTagMapper
 extends Mapper<LongWritable,Text,Text,Text>{
 public void map(LongWritable k,Text v,Context c)
  throws IOException,InterruptedException{
  String line=v.toString();
  if(line.startsWith("userId")) return;
  String[] p=line.split(",",4);
  if(p.length>=3) c.write(new Text(p[1]),new Text(p[2]));
 }
}

// MovieTagReducer.java
public class MovieTagReducer
 extends Reducer<Text,Text,Text,Text>{
 public void reduce(Text movieId,Iterable<Text> tags,Context c)
  throws IOException,InterruptedException{
  java.util.LinkedHashSet<String> set=new java.util.LinkedHashSet<>();
  for(Text t:tags) set.add(t.toString().trim());
  c.write(movieId,new Text(String.join(", ",set)));
 }
}
```

To display movie titles as well, join the resulting movieId-to-tags data with movies.csv using a second MapReduce job or an appropriate distributed join.

```
hadoop jar MovieTags.jar MovieTagsDriver /movielens/tags /movielens/movie_tags
hdfs dfs -cat /movielens/movie_tags/part-r-00000
```

5. MongoDB Functions: Count, Sort, Limit, Skip and Aggregate

Aim

Implement common MongoDB query and aggregation functions.

```
use bigdata

db.movies.insertMany([
 {title:"Inception",year:2010,rating:8.8,genre:"Sci-Fi"},
 {title:"Titanic",year:1997,rating:7.9,genre:"Drama"},
 {title:"Avatar",year:2009,rating:7.8,genre:"Sci-Fi"},
 {title:"The Matrix",year:1999,rating:8.7,genre:"Sci-Fi"}
])

// Count
db.movies.countDocuments({})
db.movies.countDocuments({genre:"Sci-Fi"})

// Sort: descending rating
db.movies.find().sort({rating:-1})

// Limit: top 2
db.movies.find().sort({rating:-1}).limit(2)

// Skip: skip first 2
db.movies.find().sort({rating:-1}).skip(2)

// Aggregate: average rating by genre
db.movies.aggregate([
  {$group:{_id:"$genre", averageRating:{$avg:"$rating"}, count:{$sum:1}}},
  {$sort:{averageRating:-1}}
])
```

Result: MongoDB successfully performs counting, sorting, pagination using skip/limit, and aggregation.

6. Pig Latin Scripts: Sort, Group, Join, Project and Filter

Aim

Use Pig Latin to perform common data transformation operations.

```
-- Load
A = LOAD 'students.csv' USING PigStorage(',')
    AS (id:int, name:chararray, dept:chararray, marks:int);

-- Filter
F = FILTER A BY marks >= 60;

-- Project
P = FOREACH F GENERATE id, name, marks;

-- Sort
S = ORDER P BY marks DESC;

-- Group
G = GROUP A BY dept;
C = FOREACH G GENERATE group AS dept, COUNT(A) AS student_count;

-- Join
B = LOAD 'departments.csv' USING PigStorage(',')
    AS (dept:chararray, hod:chararray);
J = JOIN A BY dept, B BY dept;
R = FOREACH J GENERATE A::name, A::dept, B::hod;

DUMP F;
DUMP P;
DUMP S;
DUMP C;
DUMP R;
```

Result: Pig scripts successfully filter, project, sort, group and join the sample datasets.

7. Hive: Databases, Tables, Views, Functions and Indexes

Aim

Use Hive DDL and related commands to create, alter and drop databases, tables, views, functions and indexes where supported by the selected Hive version.

```
-- Database
CREATE DATABASE IF NOT EXISTS bigdata;
SHOW DATABASES;
USE bigdata;
ALTER DATABASE bigdata SET DBPROPERTIES ('purpose'='lab');
DROP DATABASE IF EXISTS olddb CASCADE;

-- Table
CREATE TABLE students(
  id INT,
  name STRING,
  dept STRING,
  marks INT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';

ALTER TABLE students ADD COLUMNS (email STRING);
ALTER TABLE students RENAME TO student_details;

-- View
CREATE VIEW high_scorers AS
SELECT id,name,marks FROM student_details WHERE marks >= 80;
SHOW VIEWS;
DROP VIEW high_scorers;

-- Function
SHOW FUNCTIONS;
DESCRIBE FUNCTION upper;
SELECT upper(name) FROM student_details;
-- Custom temporary function syntax (requires a compatible JAR):
-- ADD JAR /path/to/myudf.jar;
-- CREATE TEMPORARY FUNCTION myfunc AS 'com.example.MyUDF';
-- DROP TEMPORARY FUNCTION myfunc;

-- Index
-- Legacy Hive versions support:
-- CREATE INDEX student_idx ON TABLE student_details(marks)
-- AS 'COMPACT' WITH DEFERRED REBUILD;
-- DROP INDEX student_idx ON student_details;
```

Important: Hive index support is version-dependent and has been removed/deprecated in newer Hive releases. For current Hive versions, use partitioning, bucketing and appropriate storage formats instead when indexing is unavailable.

8. Word Count Using Hadoop MapReduce and Apache Spark

Aim

Implement the classic Word Count program using Hadoop MapReduce and Spark.

A. Hadoop MapReduce Word Count

```
// WordCount.java
import java.io.*;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {
 public static class WCMapper
  extends Mapper<LongWritable,Text,Text,IntWritable>{
  private static final IntWritable one=new IntWritable(1);
  private Text word=new Text();
  public void map(LongWritable k,Text v,Context c)
   throws IOException,InterruptedException{
   StringTokenizer st=new StringTokenizer(v.toString());
   while(st.hasMoreTokens()){
    word.set(st.nextToken().toLowerCase().replaceAll("[^a-z0-9]",""));
    if(word.getLength()>0)c.write(word,one);
   }
  }
 }
 public static class WCReducer
  extends Reducer<Text,IntWritable,Text,IntWritable>{
  public void reduce(Text k,Iterable<IntWritable> v,Context c)
   throws IOException,InterruptedException{
   int sum=0; for(IntWritable x:v) sum+=x.get();
   c.write(k,new IntWritable(sum));
  }
 }
 public static void main(String[] a)throws Exception{
  Job j=Job.getInstance(new Configuration(),"Word Count");
  j.setJarByClass(WordCount.class);
  j.setMapperClass(WCMapper.class);
  j.setCombinerClass(WCReducer.class);
  j.setReducerClass(WCReducer.class);
  j.setOutputKeyClass(Text.class);
  j.setOutputValueClass(IntWritable.class);
  FileInputFormat.addInputPath(j,new Path(a[0]));
  FileOutputFormat.setOutputPath(j,new Path(a[1]));
  System.exit(j.waitForCompletion(true)?0:1);
 }
}

hadoop jar WordCount.jar WordCount /wordcount/input /wordcount/output
hdfs dfs -cat /wordcount/output/part-r-00000
```

B. Spark Word Count

```
# PySpark
from pyspark.sql import SparkSession
import re

spark = SparkSession.builder.appName("WordCount").getOrCreate()
text = spark.sparkContext.textFile("hdfs:///wordcount/input")
counts = (text.flatMap(lambda line: re.findall(r"[A-Za-z0-9]+", line.lower()))
              .map(lambda word: (word, 1))
              .reduceByKey(lambda a,b: a+b)
              .sortByKey())

for word,count in counts.collect():
    print(word, count)

spark.stop()
```

Run with a suitable Spark installation:

```
spark-submit wordcount.py
```

Result: Word frequencies are produced using both Hadoop MapReduce and Spark.

9. CDH and HUE: Data Analysis and Report Generation

Aim

Use Cloudera Distribution for Hadoop (CDH/Cloudera platform) and HUE to upload, query, analyze and report on sample datasets.

Procedure

8. Start the Hadoop/Cloudera services in the lab environment.
9. Open HUE in a web browser using the URL/port configured by the administrator.
10. Upload a CSV or text sample dataset into HDFS using the HUE file browser.
11. Create a Hive table over the uploaded data or load the data into an existing table.
12. Open the SQL editor in HUE and execute queries.
13. Use HUE's available visualizations/dashboards to display query results.
14. Export or save the results according to the lab environment's reporting options.

Sample Hive Analysis

```
CREATE DATABASE IF NOT EXISTS reportdb;
USE reportdb;

CREATE TABLE sales(
  order_id INT,
  product STRING,
  category STRING,
  quantity INT,
  amount DOUBLE
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';

SELECT category, SUM(amount) AS total_sales
FROM sales
GROUP BY category
ORDER BY total_sales DESC;

SELECT product, SUM(quantity) AS units_sold
FROM sales
GROUP BY product
ORDER BY units_sold DESC
LIMIT 10;
```

Suggested Report

- Dataset name and source
- Number of records
- Top categories/products
- Total and average values
- Query results in tabular form
- Charts/visualizations available in the installed HUE version
- Short observations and conclusion

Result: The sample dataset is stored in the Hadoop ecosystem, analyzed using Hive through HUE, and the resulting tables/charts are used to prepare a report.

Overall Conclusion

The above experiments demonstrate the major components of a Big Data Analytics laboratory: HDFS file management, MapReduce programming, MongoDB operations, Pig Latin transformations, Hive data management, Hadoop and Spark Word Count, and data analysis through the Cloudera/HUE environment.

Important Lab Notes

- Use a new output directory for each MapReduce job because Hadoop normally does not allow a job to write into an existing output directory.
- Replace paths such as /user/$USER/... with the HDFS path configured in your laboratory.
- Matrix multiplication requires the matrix dimensions to match; the sample program assumes a 2×2 example.
- MovieLens CSV files can contain commas inside quoted fields, so a production implementation should use a CSV parser rather than a simple split(',').
- Hive index commands depend strongly on the Hive version. Check the version installed in your lab before executing index DDL.
- CDH/HUE menu names and ports differ by Cloudera distribution and deployment.

BIG DATA ANALYTICS LAB
VIVA-VOCE QUESTIONS

5 Viva Questions for Each of the 9 Experiments

Experiment 1: Hadoop Installation and HDFS File Management

1. What is Hadoop, and what are its main components?
2. What is HDFS, and why is it used?
3. What is the difference between the `-put` and `-get` commands in HDFS?
4. How do you create a directory and list files in HDFS?
5. What is the difference between `hdfs dfs -rm` and `hdfs dfs -rm -r`?

Experiment 2: MapReduce Matrix Multiplication

1. What is MapReduce, and what are the roles of the Mapper and Reducer?
2. How can matrix multiplication be expressed as a row-by-column operation?
3. What key-value pairs can the Mapper emit for matrix multiplication?
4. Why must the number of columns in the first matrix equal the number of rows in the second matrix?
5. What does the Reducer calculate to produce each element of the result matrix?

Experiment 3: MapReduce Weather Data Analysis

1. What is the purpose of analyzing weather data using MapReduce?
2. What fields are present in the sample weather dataset?
3. What is the role of the Mapper in the weather analysis program?
4. How are weather conditions classified in the program?
5. Why might a real weather-analysis system use a Reducer to calculate daily minimum, maximum, or average temperature?

Experiment 4: MovieLens Tag Analysis using MapReduce

1. What is the MovieLens dataset, and what information does `tags.csv` contain?
2. What is a movie ID, and why is it useful when analyzing tags?
3. What key and value does the Mapper emit in the movie-tag program?
4. What is the purpose of the Reducer in grouping tags by movie?
5. Why can a simple comma-based split be unreliable when reading CSV files?

Experiment 5: MongoDB Count, Sort, Limit, Skip and Aggregate

1. What is MongoDB, and how does it store data?
2. What is the purpose of `countDocuments()`?
3. How does `sort()` arrange documents in ascending or descending order?
4. What is the difference between `limit()` and `skip()`?
5. What is an aggregation pipeline, and what do `$group` and `$sum` do?

Experiment 6: Pig Latin Sort, Group, Join, Project and Filter

1. What is Apache Pig, and what is Pig Latin?
2. What does the `FILTER` operator do in Pig?
3. What is the purpose of `FOREACH ... GENERATE`?
4. How does `ORDER` differ from `GROUP`?
5. What is a `JOIN` operation in Pig, and when would you use it?

Experiment 7: Hive Databases, Tables, Views, Functions and Indexes

1. What is Apache Hive, and what is HiveQL?
2. What is the difference between a managed table and an external table in Hive?
3. What is a view, and why is it used?
4. What is the purpose of `ALTER TABLE` and `DROP TABLE`?
5. Are Hive indexes supported in every Hive version? What alternatives can be used in newer versions?

Experiment 8: Word Count using Hadoop MapReduce and Spark

1. What is the Word Count problem in Big Data processing?
2. What are the Mapper and Reducer responsibilities in Hadoop Word Count?
3. What is the purpose of a Combiner in a MapReduce job?
4. What is an RDD in Apache Spark?
5. How does Spark's `reduceByKey()` help calculate word frequencies?

Experiment 9: CDH and HUE Data Analysis and Reporting

1. What is CDH, and what role does it play in a Hadoop ecosystem?
2. What is HUE, and how does it help users work with Hadoop?
3. How can a dataset be uploaded to HDFS using HUE?
4. How can Hive queries be executed through HUE?
5. What information should be included in a data-analysis report?