1. Write a program in Map Reduce for Word Count operation.

**Step 1: Start Cloudera QuickStart VM**

* Open VirtualBox and start the Cloudera QuickStart VM.

**Step 2: Open Eclipse on Cloudera Desktop**

* Click on the Eclipse icon on the Cloudera desktop to open the Eclipse IDE.

**Step 3: Create a Java Project**

* In Eclipse, go to File -> New -> Java Project.
* Give the project a name, for example, "WordCount."
* Click Finish.

**Step 4: Add Hadoop Libraries to Project**

* Right-click on the project, select Build Path -> Configure Build Path.
* In the Libraries tab, click Add External JARs.
* Navigate to /usr/lib/hadoop and add all the JAR files.
* Repeat the process for /usr/lib/hadoop-client.

**Step 5: Write Java Code for Word Count**

* Right-click on the src folder of the project, go to New -> Class.
* Write the class names: WordCountDriver, WordMapper, and SumReducer.
* Write the code for each class. Ensure you have the necessary import statements.

**Step 6: Export the Project as JAR**

* Right-click on the project, select Export -> Java -> JAR file.
* Choose the export destination, provide a file name (e.g., wordcount.jar), and click Finish.

**Step 7: Move the JAR File to Hadoop File System**

* Open a terminal on the Cloudera desktop.
* Use the following command to move the JAR file to HDFS:
* hdfs dfs -put /path/to/local/wordcount.jar /user/cloudera

**Step 8: Create the Input File for the MapReduce Program**

* Prepare a text file containing the input data. You can create it using a text editor.

**Step 9: Move the Input File to Hadoop File System**

* Use the following command to move the input file to HDFS:
* hdfs dfs -put /path/to/local/myInputFile.txt /user/cloudera

**Step 10: Run MapReduce Program on Hadoop**

* Execute the following command to run the Word Count program:
* hadoop jar /path/to/local/wordcount.jar WordCountDriver /user/cloudera/myInputFile.txt /user/cloudera/myOutput

**Step 11: View Output Directory**

* To list the contents of the output directory, use:
* hdfs dfs -ls /user/cloudera/myOutput

**Step 12: View the Output File**

* Use the following command to view the output file:

hdfs dfs -cat /user/cloudera/myOutput/part-r-00000

code –

### 1. Mapper:

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.mapreduce.Mapper;

public class WordCountMapper extends 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, Context context) throws IOException, InterruptedException {

StringTokenizer tokenizer = new StringTokenizer(value.toString());

while (tokenizer.hasMoreTokens()) {

word.set(tokenizer.nextToken());

context.write(word, one);

}

}

}

### 2. Reducer:

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class WordCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

private IntWritable result = new IntWritable();

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

throws IOException, InterruptedException {

int sum = 0;

for (IntWritable value : values) {

sum += value.get();

}

result.set(sum);

context.write(key, result);

}

}

### 3. Driver Program:

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCountDriver {

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

if (args.length != 2) {

System.err.println("Usage: WordCountDriver <input path> <output path>");

System.exit(-1);

}

Job job = new Job();

job.setJarByClass(WordCountDriver.class);

job.setJobName("WordCount");

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

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

job.setMapperClass(WordCountMapper.class);

job.setReducerClass(WordCountReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}

**Q2 Write a program in Map Reduce to find average word count**

### Step 1: Start Cloudera QuickStart VM

* Open VirtualBox and start the Cloudera QuickStart VM.

### Step 2: Open Eclipse on Cloudera Desktop

* Click on the Eclipse icon on the Cloudera desktop to open the Eclipse IDE.

### Step 3: Create a Java Project

* In Eclipse, go to File -> New -> Java Project.
* Give the project a name, for example, "AvgWordCount."
* Click Finish.

### Step 4: Add Hadoop Libraries to Project

* Right-click on the project, select Build Path -> Configure Build Path.
* In the Libraries tab, click Add External JARs.
* Navigate to /usr/lib/hadoop and add all the JAR files.
* Repeat the process for /usr/lib/hadoop-client.

### Step 5: Write Java Code for Average Word Count

* Right-click on the src folder of the project, go to New -> Class.
* Write the class names: AvgDriver, AvgMapper, and AvgReducer.
* Write the code for each class. Ensure you have the necessary import statements.

### Step 6: Export the Project as JAR

* Right-click on the project, select Export -> Java -> JAR file.
* Choose the export destination, provide a file name (e.g., AvgWordCount.jar), and click Finish.

### Step 7: Verify the JAR File

* Open a terminal and use the following command to list the contents of the JAR:
* jar -tf /path/to/local/AvgWordCount.jar

### Step 8: Move the JAR File to Hadoop File System

* Use the following command to move the JAR file to HDFS:
* hdfs dfs -put /path/to/local/AvgWordCount.jar /user/cloudera

### Step 9: Create an Input File for the MapReduce Program

* Prepare a text file containing the input data. You can create it using a text editor. Each line represents a separate document.

### Step 10: Move the Input File to Hadoop File System

* Use the following command to move the input file to HDFS:
* hdfs dfs -put /path/to/local/myInputFile.txt /user/cloudera

### Step 11: Run the MapReduce Program on Hadoop

* Execute the following command to run the program:
* hadoop jar /path/to/local/AvgWordCount.jar AvgDriver /user/cloudera/myInputFile.txt /user/cloudera/ResultAvg

### Step 12: View Output Directory

* To list the contents of the output directory, use:
* hdfs dfs -ls /user/cloudera/ResultAvg

### Step 13: View the Output File

* Use the following command to view the output file:

hdfs dfs -cat /user/cloudera/ResultAvg/part-r-00000

**Code –**

### Mapper:

java

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.mapreduce.Mapper;

public class AverageWordCountMapper extends 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, Context context) throws IOException, InterruptedException {

StringTokenizer tokenizer = new StringTokenizer(value.toString());

while (tokenizer.hasMoreTokens()) {

word.set(tokenizer.nextToken());

context.write(word, one);

}

}

}

### Reducer:

java

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class AverageWordCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

private IntWritable result = new IntWritable();

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

throws IOException, InterruptedException {

int sum = 0;

for (IntWritable value : values) {

sum += value.get();

}

result.set(sum);

context.write(key, result);

}

}

### AverageWordCountDriver:

java

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class AverageWordCountDriver {

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

if (args.length != 2) {

System.err.println("Usage: AverageWordCountDriver <input path> <output path>");

System.exit(-1);

}

Job job = new Job();

job.setJarByClass(AverageWordCountDriver.class);

job.setJobName("AverageWordCount");

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

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

job.setMapperClass(AverageWordCountMapper.class);

job.setReducerClass(AverageWordCountReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}

**Q3. Write a program in Map Reduce to demonstrate sorting operation**.

### Step 1: Start Cloudera QuickStart VM

* Open VirtualBox and start the Cloudera QuickStart VM.

### Step 2: Open Eclipse on Cloudera Desktop

* Click on the Eclipse icon on the Cloudera desktop to open the Eclipse IDE.

### Step 3: Create a Java Project

* In Eclipse, go to File -> New -> Java Project.
* Give the project a name, for example, "MatrixMultiplication."
* Click Finish.

### Step 4: Add Hadoop Libraries to Project

* Right-click on the project, select Build Path -> Configure Build Path.
* In the Libraries tab, click Add External JARs.
* Navigate to /usr/lib/hadoop and add all the JAR files.
* Repeat the process for /usr/lib/hadoop-client.

### Step 5: Write Java Code for Matrix Multiplication

* Right-click on the src folder of the project, go to New -> Class.
* Write the class names: MatrixDriver, MatrixMapper, and MatrixReducer.
* Write the code for each class. Ensure you have the necessary import statements.

### Step 6: Export the Project as JAR

* Right-click on the project, select Export -> Java -> JAR file.
* Choose the export destination, provide a file name (e.g., MatrixMultiplication.jar), and click Finish.

### Step 7: Move the JAR File to Hadoop File System

* Open a terminal and use the following command to move the JAR file to HDFS:
* hdfs dfs -put /path/to/local/MatrixMultiplication.jar /user/cloudera

### Step 8: Create Input Files for the MapReduce Program

* Prepare two text files (matrixA.txt and matrixB.txt) containing the input matrices. Ensure that the matrices are in the correct format for multiplication.

### Step 9: Move Input Files to Hadoop File System

* Use the following commands to move the input files to HDFS:
* hdfs dfs -put /path/to/local/matrixA.txt /user/cloudera/matrixInput/
* hdfs dfs -put /path/to/local/matrixB.txt /user/cloudera/matrixInput/

### Step 10: Run the MapReduce Program on Hadoop

* Execute the following command to run the program:
* hadoop jar /path/to/local/MatrixMultiplication.jar MatrixDriver matrixInput ResultMatrixMultiplication

### Step 11: View Output Directory

* To list the contents of the output directory, use:
* hdfs dfs -ls /user/cloudera/ResultMatrixMultiplication

### Step 12: View the Output File

* Use the following command to view the output file:

hdfs dfs -cat /user/cloudera/ResultMatrixMultiplication/part-r-00000

### Mapper:

import java.io.IOException;

import java.util.StringTokenizer;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class SortingMapper extends Mapper<Object, Text, IntWritable, Text> {

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

private Text word = new Text();

public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

StringTokenizer tokenizer = new StringTokenizer(value.toString());

while (tokenizer.hasMoreTokens()) {

word.set(tokenizer.nextToken());

context.write(one, word);

}

}

}

### Reducer:

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class SortingReducer extends Reducer<IntWritable, Text, IntWritable, Text> {

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

throws IOException, InterruptedException {

for (Text value : values) {

context.write(key, value);

}

}

}

### SortingDriver:

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class SortingDriver {

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

if (args.length != 2) {

System.err.println("Usage: SortingDriver <input path> <output path>");

System.exit(-1);

}

Job job = new Job();

job.setJarByClass(SortingDriver.class);

job.setJobName("SortingJob");

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

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

job.setMapperClass(SortingMapper.class);

job.setReducerClass(SortingReducer.class);

job.setOutputKeyClass(IntWritable.class);

job.setOutputValueClass(Text.class);

System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}

**Q4 Write a program in Map Reduce for Matrix Multiplication**

### Step 1: Start Cloudera QuickStart VM

* Open VirtualBox and start the Cloudera QuickStart VM.

### Step 2: Open Eclipse on Cloudera Desktop

* Click on the Eclipse icon on the Cloudera desktop to open the Eclipse IDE.

### Step 3: Create a Java Project

* In Eclipse, go to File -> New -> Java Project.
* Give the project a name, for example, "MatrixMultiplication."
* Click Finish.

### Step 4: Add Hadoop Libraries to Project

* Right-click on the project, select Build Path -> Configure Build Path.
* In the Libraries tab, click Add External JARs.
* Navigate to /usr/lib/hadoop and add all the JAR files.
* Repeat the process for /usr/lib/hadoop-client.

### Step 5: Write Java Code for Matrix Multiplication

* Right-click on the src folder of the project, go to New -> Class.
* Write the class names: MatrixMultiplicationDriver, MatrixMultiplicationMapper, and MatrixMultiplicationReducer.
* Write the code for each class. Ensure you have the necessary import statements.

### Step 6: Export the Project as JAR

* Right-click on the project, select Export -> Java -> JAR file.
* Choose the export destination, provide a file name (e.g., matrixmultiplication.jar), and click Finish.

### Step 7: Move the JAR File to Hadoop File System

* Open a terminal and use the following command to move the JAR file to HDFS:
* hdfs dfs -put /path/to/local/matrixmultiplication.jar /user/cloudera

### Step 8: Create Input Files for the MapReduce Program

#### First Input File (myMMmatrix.txt):

cat > myMMmatrix.txt

M,0,0,1

M,0,1,2

M,1,0,3

M,1,1,4

Ctrl+D

#### Second Input File (myNMatrix.txt):

cat > myNMatrix.txt

N,0,0,3

N,0,1,6

N,1,0,4

N,1,1,2

Ctrl+D

### Step 9: Create the Input Directory and Move Input Files

bash

hdfs dfs -mkdir /user/cloudera/matrixInput

hdfs dfs -put myMMmatrix.txt /user/cloudera/matrixInput/

hdfs dfs -put myNMatrix.txt /user/cloudera/matrixInput/

hdfs dfs -ls /user/cloudera/matrixInput

### Step 10: Run the MapReduce Program on Hadoop

* Execute the following command to run the program:
* hadoop jar matrixmultiplication.jar MatrixMultiplicationDriver matrixInput matrixOutput

### Step 11: View Output Directory

* To list the contents of the output directory, use:
* hdfs dfs -ls /user/cloudera/matrixOutput

### Step 12: View the Output File

* Use the following command to view the output file:

hdfs dfs -cat /user/cloudera/matrixOutput/part-r-00000

### MatrixMapper:

java

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class MatrixMapper extends Mapper<Object, Text, Text, Text> {

private Text outputKey = new Text();

private Text outputValue = new Text();

public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

String[] tokens = value.toString().split(",");

// Check if the input line is for matrix A or matrix B

if (tokens[0].equals("A")) {

// Matrix A entry

String i = tokens[1];

String j = tokens[2];

String val = tokens[3];

for (int k = 0; k < MatrixMultiplication.NUM\_COLUMNS\_A; k++) {

outputKey.set(i + "," + k);

outputValue.set("A," + j + "," + val);

context.write(outputKey, outputValue);

}

} else if (tokens[0].equals("B")) {

// Matrix B entry

String j = tokens[1];

String k = tokens[2];

String val = tokens[3];

for (int i = 0; i < MatrixMultiplication.NUM\_ROWS\_A; i++) {

outputKey.set(i + "," + k);

outputValue.set("B," + j + "," + val);

context.write(outputKey, outputValue);

}

}

}

}

### MatrixReducer:

java

import java.io.IOException;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class MatrixReducer extends Reducer<Text, Text, Text, Text> {

private Text outputValue = new Text();

public void reduce(Text key, Iterable<Text> values, Context context)

throws IOException, InterruptedException {

int[] vectorA = new int[MatrixMultiplication.NUM\_COLUMNS\_A];

int[] vectorB = new int[MatrixMultiplication.NUM\_COLUMNS\_A];

// Initialize vectors

for (int i = 0; i < MatrixMultiplication.NUM\_COLUMNS\_A; i++) {

vectorA[i] = 0;

vectorB[i] = 0;

}

for (Text value : values) {

String[] tokens = value.toString().split(",");

if (tokens[0].equals("A")) {

vectorA[Integer.parseInt(tokens[1])] = Integer.parseInt(tokens[2]);

} else if (tokens[0].equals("B")) {

vectorB[Integer.parseInt(tokens[1])] = Integer.parseInt(tokens[2]);

}

}

// Compute the dot product

int result = 0;

for (int i = 0; i < MatrixMultiplication.NUM\_COLUMNS\_A; i++) {

result += vectorA[i] \* vectorB[i];

}

outputValue.set(Integer.toString(result));

context.write(key, outputValue);

}

}

### MatrixMultiplicationDriver:

java

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class MatrixMultiplicationDriver {

public static final int NUM\_ROWS\_A = 2;

public static final int NUM\_COLUMNS\_A = 3;

public static final int NUM\_COLUMNS\_B = 2;

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

if (args.length != 2) {

System.err.println("Usage: MatrixMultiplicationDriver <input path> <output path>");

System.exit(-1);

}

Job job = new Job();

job.setJarByClass(MatrixMultiplicationDriver.class);

job.setJobName("MatrixMultiplicationJob");

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

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

job.setMapperClass(MatrixMapper.class);

job.setReducerClass(MatrixReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

System.exit(job.waitForCompletion(true) ? 0 : 1);

}