SRM Institute of Science and Technology
Delhi – Meerut Road, Sikri Kalan, Ghaziabad, Uttar Pradesh – 201204
Department of Computer Applications
Circular – 2023-24
BCA DS 5th Sem

Big Data Analytics with Applications (UDS23502J)

Lab Manual

Here's the lab manual structure for the programs listed:

Lab 1: Install Apache Hadoop

Title: Installation of Apache Hadoop

Aim: To install and configure Apache Hadoop on a suitable operating system.

Procedure:

Download the Apache Hadoop distribution.

Extract the downloaded archive to a desired location.

Configure the necessary environment variables (JAVA_HOME, HADOOP_HOME, PATH).

Edit the Hadoop configuration files (core-site.xml, hdfs-site.xml, mapred-site.xml, yarn-site.xml) to set up the Hadoop cluster.

Format the NameNode.

Start the Hadoop cluster (start-dfs.sh, start-yarn.sh).

Verify the installation by accessing the Hadoop web interfaces (NameNode, ResourceManager).

Source Code: (Configuration files - too lengthy to include directly, but will specify key settings)

core-site.xml:

```
<configuration>
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://localhost:9000</value>
    </property>
</configuration>
```

hdfs-site.xml:

```
<configuration>
```

```
    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>
    <property>
       <name>dfs.name.dir</name>
       <value>/path/to/hadoop/data/namenode</value>
    </property>
    <property>
       <name>dfs.data.dir</name>
       <value>/path/to/hadoop/data/datanode</value>
    </property>
</configuration>
```

mapred-site.xml:

```
<configuration>
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
</configuration>
```

yarn-site.xml

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

Input: Hadoop distribution file, system configuration details (disk space, memory).

Expected Output: A running Hadoop cluster, accessible via the command line (HDFS, YARN commands) and web interfaces.

Lab 2: Develop a MapReduce program to calculate the frequency of a given word in a given file.

Title: Word Frequency Calculation using MapReduce

Aim: To write a MapReduce program that counts the occurrences of each word in a text file.

Procedure:

1. Write a Mapper class that reads the input file, tokenizes each line into words, and emits key-value pairs where the key is the word and the value is 1.
2. Write a Reducer class that receives the key-value pairs from the Mapper, sums the values for each unique word, and outputs the word and its total count.
3. Configure and run the MapReduce job on the Hadoop cluster.

Source Code:

```
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;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.conf.Configuration;

public class WordCount {

    public static class TokenizerMapper
            extends Mapper<Object, Text, Text, IntWritable> {

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

        public void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
            StringTokenizer itr = new StringTokenizer(value.toString());
            while (itr.hasMoreTokens()) {
                word.set(itr.nextToken());
                context.write(word, one);
            }
        }
    }

    public static class IntSumReducer
            extends Reducer<Text, IntWritable, Text, IntWritable> {
```

```
        private IntWritable result = new IntWritable();

        public void reduce(Text key, Iterable<IntWritable> values, Context
context) throws IOException, InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            result.set(sum);
            context.write(key, result);
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class); //optional, can improve
performance
        job.setReducerClass(IntSumReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new
org.apache.hadoop.fs.Path(args[0]));
        FileOutputFormat.setOutputPath(job, new
org.apache.hadoop.fs.Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Input: A text file containing the words to be counted. For example:

```
The quick brown fox jumps over the lazy dog.  The dog is lazy.
```

Expected Output: A list of words and their corresponding counts. For example:

```
The     2
quick   1
brown   1
fox     1
jumps   1
over    1
lazy    2
dog     2
is      1
```

Lab 3: Develop a MapReduce program to find the maximum temperature in each year.

Title: Maximum Temperature per Year using MapReduce

Aim: To develop a MapReduce program that processes weather data to determine the highest temperature recorded for each year.

Procedure:

1. Write a Mapper that reads weather records (year, temperature) and emits the year as the key and the temperature as the value.
2. Write a Reducer that receives the year and a list of temperatures for that year, and outputs the year and the maximum temperature.
3. Configure and run the MapReduce job.

Source Code:

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.conf.Configuration;

public class MaxTemperature {

    public static class MaxTemperatureMapper
            extends Mapper<Object, Text, Text, IntWritable> {

        private Text year = new Text();
        private IntWritable temperature = new IntWritable();

        public void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
            String line = value.toString();
            String[] parts = line.split(","); // Assuming comma-separated
data
            if (parts.length > 1) {
                try {
                    year.set(parts[0].trim());  // Year is the first field
                    int temp = Integer.parseInt(parts[1].trim()); //
Temperature is the second field
                    temperature.set(temp);
                    context.write(year, temperature);
                } catch (NumberFormatException e) {
```

```
                    // Handle cases where temperature is not a valid number
                    System.err.println("Invalid temperature value: " +
parts[1] + " in line: " + line);
                }
            }
        }
    }

    public static class MaxTemperatureReducer
            extends Reducer<Text, IntWritable, Text, IntWritable> {

        private IntWritable maxTemp = new IntWritable();

        public void reduce(Text key, Iterable<IntWritable> values, Context
context) throws IOException, InterruptedException {
            int max = Integer.MIN_VALUE;
            for (IntWritable val : values) {
                max = Math.max(max, val.get());
            }
            maxTemp.set(max);
            context.write(key, maxTemp);
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "max temperature");
        job.setJarByClass(MaxTemperature.class);
        job.setMapperClass(MaxTemperatureMapper.class);
        job.setReducerClass(MaxTemperatureReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new
org.apache.hadoop.fs.Path(args[0]));
        FileOutputFormat.setOutputPath(job, new
org.apache.hadoop.fs.Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Input: A dataset containing year and temperature information (e.g., CSV file). Example:

```
2020,25
2020,30
2021,28
2021,32
2022,27
2022,35
```

Expected Output: The maximum temperature for each year. Example:

```
2020    30
2021    32
2022    35
```

Lab 4: Develop a MapReduce program to find the grades of students.

Title: Student Grade Calculation using MapReduce

Aim: To develop a MapReduce program that calculates student grades based on their scores.

Procedure:

1. Write a Mapper that reads student records (student ID, score) and emits the student ID as the key and the score as the value.
2. Write a Reducer that receives the student ID and their scores, calculates the grade based on a predefined grading scale, and outputs the student ID and their grade.
3. Configure and run the MapReduce job.

Source Code:

```
import java.io.IOException;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.conf.Configuration;

public class StudentGrades {

    public static class GradeMapper extends Mapper<Object, Text, Text,
IntWritable> {
        private Text studentId = new Text();
        private IntWritable score = new IntWritable();

        public void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
            String line = value.toString();
            String[] parts = line.split(","); // Assuming comma-separated
data: studentId, score
            if (parts.length == 2) {
                studentId.set(parts[0].trim());
                try {
                    int s = Integer.parseInt(parts[1].trim());
                    score.set(s);
                    context.write(studentId, score);
                } catch (NumberFormatException e) {
                    System.err.println("Invalid score: " + parts[1] + " for
student: " + parts[0]);
                }
            }
```

```
        }
    }

    public static class GradeReducer extends Reducer<Text, IntWritable, Text,
Text> {
        private Text grade = new Text();

        public void reduce(Text key, Iterable<IntWritable> values, Context
context) throws IOException, InterruptedException {
            int totalScore = 0;
            int count = 0;
            for (IntWritable value : values) {
                totalScore += value.get();
                count++;
            }
            double averageScore = (double) totalScore / count; // Calculate
average if multiple scores
            String gradeStr;
            if (averageScore >= 90) {
                gradeStr = "A";
            } else if (averageScore >= 80) {
                gradeStr = "B";
            } else if (averageScore >= 70) {
                gradeStr = "C";
            } else if (averageScore >= 60) {
                gradeStr = "D";
            } else {
                gradeStr = "F";
            }
            grade.set(gradeStr);
            context.write(key, new Text("Grade: " + gradeStr + ", Avg Score:
" + averageScore));
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "student grades");
        job.setJarByClass(StudentGrades.class);
        job.setMapperClass(GradeMapper.class);
        job.setReducerClass(GradeReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

job.setOutputFormatClass(org.apache.hadoop.mapreduce.lib.output.TextOutputFor
mat.class);
        FileInputFormat.addInputPath(job, new
org.apache.hadoop.fs.Path(args[0]));
        FileOutputFormat.setOutputPath(job, new
org.apache.hadoop.fs.Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Input: A dataset containing student IDs and scores. Example:

```
1001,85
1002,92
1001,88
1003,75
1004,60
1005,55
```

Expected Output: Student IDs and their calculated grades. Example:

```
1001    Grade: B, Avg Score: 86.5
1002    Grade: A, Avg Score: 92.0
1003    Grade: C, Avg Score: 75.0
1004    Grade: D, Avg Score: 60.0
1005    Grade: F, Avg Score: 55.0
```

Lab 5: Develop a MapReduce program to implement Matrix Multiplication

Title: Matrix Multiplication using MapReduce

Aim: To implement matrix multiplication using the MapReduce paradigm.

Procedure:

1. Represent matrices A and B as input data. Each line will contain the matrix identifier (A or B), row/column index, and value.
2. Write a Mapper that processes each matrix entry. For an element in matrix A, it emits key-value pairs where the key is the column index of A and the value contains the row index of A, the value, and a marker 'A'. For an element in matrix B, it emits key-value pairs where the key is the row index of B and the value contains the column index of B, the value, and a marker 'B'.
3. Write a Reducer that receives key-value pairs for a specific column of A and a row of B. It joins the corresponding elements, multiplies them, and sums the results to compute one element of the output matrix C.
4. Configure and run the MapReduce job.

Source Code:

```
import java.io.IOException;
import java.util.ArrayList;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.conf.Configuration;

public class MatrixMultiplication {

    public static class MatrixMapper extends Mapper<LongWritable, Text, Text,
Text> {
        private Text keyOut = new Text();
        private Text valueOut = new Text();

        public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
            String line = value.toString();
            String[] parts = line.split(","); // Matrix, Row, Column, Value
            if (parts.length == 4) {
```

```
                String matrix = parts[0].trim();
                int row = Integer.parseInt(parts[1].trim());
                int col = Integer.parseInt(parts[2].trim());
                double val = Double.parseDouble(parts[3].trim());

                if (matrix.equals("A")) {
                    for (int i = 0; i < 3; i++) { // Assuming 3x3 matrices.
Use a parameter for size.
                        keyOut.set(col + "");  // Use column of A as key
                        valueOut.set(row + "," + val + ",A"); // Row of A,
Value, Matrix Indicator
                        context.write(keyOut, valueOut);
                    }
                } else if (matrix.equals("B")) {
                    for (int i = 0; i < 3; i++) {  // Assuming 3x3 matrices.
Use a parameter for size.
                        keyOut.set(row + "");      // Use row of B as key
                        valueOut.set(col + "," + val + ",B");  // Column of
B, Value, Matrix Indicator
                        context.write(keyOut, valueOut);
                    }
                }
            }
        }
    }

    public static class MatrixReducer extends Reducer<Text, Text, Text, Text>
{
        private Text outputKey = new Text();
        private Text outputValue = new Text();

        public void reduce(Text key, Iterable<Text> values, Context context)
throws IOException, InterruptedException {
            ArrayList<String> aValues = new ArrayList<String>();
            ArrayList<String> bValues = new ArrayList<String>();

            for (Text value : values) {
                String[] parts = value.toString().split(",");
                if (parts[2].equals("A")) {
                    aValues.add(parts[0] + "," + parts[1]); // row, value
                } else if (parts[2].equals("B")) {
                    bValues.add(parts[0] + "," + parts[1]); // col, value
                }
            }

            for (String aVal : aValues) {
                String[] aParts = aVal.split(",");
                int aRow = Integer.parseInt(aParts[0]);
                double aValue = Double.parseDouble(aParts[1]);

                for (String bVal : bValues) {
                    String[] bParts = bVal.split(",");
                    int bCol = Integer.parseInt(bParts[0]);
                    double bValue = Double.parseDouble(bParts[1]);

                    double product = aValue * bValue;
                    outputKey.set(aRow + "," + bCol);
                    outputValue.set(product + "");
                    context.write(outputKey, outputValue);
                }
            }
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "matrix multiplication");
```

```
        job.setJarByClass(MatrixMultiplication.class);
        job.setMapperClass(MatrixMapper.class);
        job.setReducerClass(MatrixReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);
        // Multiple text output format to handle the intermediate results

job.setOutputFormatClass(org.apache.hadoop.mapreduce.lib.output.TextOutputFor
mat.class);
        FileInputFormat.addInputPath(job, new
org.apache.hadoop.fs.Path(args[0]));
        FileOutputFormat.setOutputPath(job, new
org.apache.hadoop.fs.Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Input: Input files representing matrices A and B. Example (for 2x2 matrices):

File A:

```
A,0,0,1
A,0,1,2
A,1,0,3
A,1,1,4
```

File B:

```
B,0,0,5
B,0,1,6
B,1,0,7
B,1,1,8
```

Expected Output: The resulting matrix C. Example:

```
0,0    17.0
0,1    23.0
1,0    39.0
1,1    53.0
```

Lab 6: Develop **a MapReduce to find the maximum electrical consumption in each year given electrical consumption for each month in each year.**

Title: Maximum Electrical Consumption per Year

Aim: To determine the highest electrical consumption for each year from monthly consumption data.

Procedure:

1. Write a Mapper that reads records containing year, month, and electrical consumption. It emits the year as the key and the consumption as the value.
2. Write a Reducer that receives the year and a list of monthly consumption values for that year. It finds the maximum consumption value and outputs the year and the maximum consumption.
3. Configure and run the MapReduce job.

Source Code:

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.conf.Configuration;

public class MaxElectricalConsumption {

    public static class ConsumptionMapper extends Mapper<Object, Text, Text,
IntWritable> {
        private Text year = new Text();
        private IntWritable consumption = new IntWritable();

        public void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
            String line = value.toString();
            String[] parts = line.split(","); // Assuming comma-separated:
year, month, consumption
            if (parts.length == 3) {
                year.set(parts[0].trim());
                try {
                    int cons = Integer.parseInt(parts[2].trim());
                    consumption.set(cons);
                    context.write(year, consumption);
                } catch (NumberFormatException e) {
```

```
                    System.err.println("Invalid consumption value: " +
parts[2] + " in line: " + line);
                }
            }
        }
    }

    public static class ConsumptionReducer extends Reducer<Text, IntWritable,
Text, IntWritable> {
        private IntWritable maxConsumption = new IntWritable();

        public void reduce(Text key, Iterable<IntWritable> values, Context
context) throws IOException, InterruptedException {
            int max = Integer.MIN_VALUE;
            for (IntWritable value : values) {
                max = Math.max(max, value.get());
            }
            maxConsumption.set(max);
            context.write(key, maxConsumption);
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "max electrical consumption");
        job.setJarByClass(MaxElectricalConsumption.class);
        job.setMapperClass(ConsumptionMapper.class);
        job.setReducerClass(ConsumptionReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new
org.apache.hadoop.fs.Path(args[0]));
        FileOutputFormat.setOutputPath(job, new
org.apache.hadoop.fs.Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Input: A dataset with year, month, and electrical consumption data. Example:

```
2020,1,120
2020,2,150
2020,3,130
2021,1,140
2021,2,160
2021,3,155
```

Expected Output: The maximum electrical consumption for each year. Example:

```
2020    150
2021    160
```

Lab 7: Develop a MapReduce to analyze weather data set and print **whether the day is shiny or cool day.**

Title: Weather Analysis using MapReduce

Aim: To analyze weather data and classify days as "shiny" or "cool" based on temperature.

Procedure:

1. Write a Mapper that reads weather records (date, temperature). It emits the date as the key and the temperature as the value.
2. Write a Reducer that receives the date and temperature. It determines if the day is "shiny" (e.g., temperature above 25 degrees) or "cool" (e.g., temperature below 25 degrees) and outputs the date and classification.
3. Configure and run the MapReduce job.

Source Code:

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.conf.Configuration;

public class WeatherAnalysis {

    public static class WeatherMapper extends Mapper<Object, Text, Text,
IntWritable> {
        private Text date = new Text();
        private IntWritable temperature = new IntWritable();

        public void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
            String line = value.toString();
            String[] parts = line.split(","); // Assuming comma-separated:
date, temperature
            if (parts.length == 2) {
                date.set(parts[0].trim());
                try {
                    int temp = Integer.parseInt(parts[1].trim());
                    temperature.set(temp);
                    context.write(date, temperature);
                } catch (NumberFormatException e) {
```

```
                    System.err.println("Invalid temperature: " + parts[1] + "
in line: " + line);
                }
            }
        }
    }

    public static class WeatherReducer extends Reducer<Text, IntWritable,
Text, Text> {
        private Text result = new Text();

        public void reduce(Text key, Iterable<IntWritable> values, Context
context) throws IOException, InterruptedException {
            int temp = 0;
            for(IntWritable value: values){
                temp = value.get();
            }
            String weatherType = (temp > 25) ? "Shiny" : "Cool"; // Example
threshold: 25 degrees
            result.set(weatherType);
            context.write(key, result);
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "weather analysis");
        job.setJarByClass(WeatherAnalysis.class);
        job.setMapperClass(WeatherMapper.class);
        job.setReducerClass(WeatherReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

job.setOutputFormatClass(org.apache.hadoop.mapreduce.lib.output.TextOutputFor
mat.class);
        FileInputFormat.addInputPath(job, new
org.apache.hadoop.fs.Path(args[0]));
        FileOutputFormat.setOutputPath(job, new
org.apache.hadoop.fs.Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Input: A dataset with date and temperature information. Example:

```
2023-01-15,28
2023-01-16,22
2023-01-17,30
2023-01-18,18
```

Expected Output: Each date classified as "Shiny" or "Cool". Example:

```
2023-01-15    Shiny
2023-01-16    Cool
2023-01-17    Shiny
2023-01-18    Cool
```

Lab 8: Develop a MapReduce program to find the number of products sold in each country by considering sales data containing fields.

Title: Product Sales by Country using MapReduce

Aim: To calculate the total number of products sold in each country from sales data.

Procedure:

1. Write a Mapper that reads sales records (country, product, quantity). It emits the country as the key and the quantity as the value.
2. Write a Reducer that receives the country and a list of quantities sold. It sums the quantities and outputs the country and the total number of products sold.
3. Configure and run the MapReduce job.

Source Code:

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.conf.Configuration;

public class ProductSalesByCountry {

    public static class SalesMapper extends Mapper<Object, Text, Text,
IntWritable> {
        private Text country = new Text();
        private IntWritable quantity = new IntWritable();

        public void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
            String line = value.toString();
            String[] parts = line.split(","); // Assuming comma-separated:
country, product, quantity
            if (parts.length == 3) {
                country.set(parts[0].trim());
                try {
                    int qty = Integer.parseInt(parts[2].trim());
                    quantity.set(qty);
                    context.write(country, quantity);
                } catch (NumberFormatException e) {
```

```
                    System.err.println("Invalid quantity: " + parts[2] + " in
line: " + line);
                }
            }
        }
    }

    public static class SalesReducer extends Reducer<Text, IntWritable, Text,
IntWritable> {
        private IntWritable totalQuantity = new IntWritable();

        public void reduce(Text key, Iterable<IntWritable> values, Context
context) throws IOException, InterruptedException {
            int total = 0;
            for (IntWritable value : values) {
                total += value.get();
            }
            totalQuantity.set(total);
            context.write(key, totalQuantity);
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "product sales by country");
        job.setJarByClass(ProductSalesByCountry.class);
        job.setMapperClass(SalesMapper.class);
        job.setReducerClass(SalesReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new
org.apache.hadoop.fs.Path(args[0]));
        FileOutputFormat.setOutputPath(job, new
org.apache.hadoop.fs.Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Input: A dataset containing sales information. Example:

```
USA,ProductA,10
USA,ProductB,20
Canada,ProductA,15
USA,ProductC,5
Canada,ProductB,25
```

Expected Output: The total number of products sold in each country. Example:

```
USA     35
Canada  40
```

Lab 9: Develop a MapReduce program to find the tags associated with each movie by analyzing movie lens data.

Title: Movie Tags Analysis using MapReduce

Aim: To identify the tags associated with each movie using the MovieLens dataset.

Procedure:

1. Obtain the MovieLens dataset.
2. Write a Mapper that reads the movie and tag data. It emits the movie ID as the key and the tag as the value.
3. Write a Reducer that receives the movie ID and a list of tags. It outputs the movie ID and the list of unique tags associated with that movie.
4. Configure and run the MapReduce job.

Source Code:

```
import java.io.IOException;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.conf.Configuration;
import java.util.HashSet;

public class MovieTagsAnalysis {

    public static class TagMapper extends Mapper<LongWritable, Text, Text,
Text> {
        private Text movieID = new Text();
        private Text tag = new Text();

        public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
            String line = value.toString();
            String[] parts = line.split(","); // Assuming comma-separated:
movieID, tag
            if (parts.length == 2) {
                movieID.set(parts[0].trim());
                tag.set(parts[1].trim());
                context.write(movieID, tag);
            }
```

```
        }
    }

    public static class TagReducer extends Reducer<Text, Text, Text, Text> {
        private Text result = new Text();

        public void reduce(Text key, Iterable<Text> values, Context context)
throws IOException, InterruptedException {
            HashSet<String> uniqueTags = new HashSet<String>();
            for (Text value : values) {
                uniqueTags.add(value.toString());
            }
            StringBuilder tagsString = new StringBuilder();
            for (String tag : uniqueTags) {
                tagsString.append(tag).append(",");
            }
            if (tagsString.length() > 0) {
                tagsString.deleteCharAt(tagsString.length() - 1); // Remove
trailing comma
            }
            result.set(tagsString.toString());
            context.write(key, result);
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "movie tags analysis");
        job.setJarByClass(MovieTagsAnalysis.class);
        job.setMapperClass(TagMapper.class);
        job.setReducerClass(TagReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);
        FileInputFormat.addInputPath(job, new
org.apache.hadoop.fs.Path(args[0]));
        FileOutputFormat.setOutputPath(job, new
org.apache.hadoop.fs.Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Input: MovieLens dataset (specifically, the file containing movie IDs and tags). Example:

```
1,comedy
1,funny
2,drama
2,serious
1,action
3,thriller
3,suspense
```

Expected Output: A list of movies and their associated tags. Example:

```
1    comedy,funny,action
2    drama,serious
3    thriller,suspense
```

Lab 10: XYZ.com is an online music website where users listen to various tracks, the data gets collected which is given... (Provide a MapReduce program)

Title: Music Track Analysis using MapReduce

Aim: To analyze user listening data from an online music website (XYZ.com) using MapReduce. (The specific aim will depend on the exact data provided, so I'll create a general example. Please provide the specific data fields for a more tailored solution.) Let's assume the goal is to find the most popular tracks.

Procedure:

1. Write a Mapper that reads user listening records (user ID, track ID, timestamp). It emits the track ID as the key and 1 as the value.
2. Write a Reducer that receives the track ID and a list of 1s (representing listens). It sums the listens for each track and outputs the track ID and its total plays.
3. Write a second MapReduce job (or include in the first reducer and use sorting) to find the track with the maximum plays.
4. Configure and run the MapReduce job(s).

Source Code:

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.conf.Configuration;

public class MusicTrackAnalysis {

    public static class ListenMapper extends Mapper<LongWritable, Text, Text,
IntWritable> {
        private Text trackId = new Text();
        private IntWritable one = new IntWritable(1);

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

```
            String line = value.toString();
            String[] parts = line.split(","); // Assuming comma-separated:
userID, trackID, timestamp
            if (parts.length == 3) {
                trackId.set(parts[1].trim()); // Track ID
                context.write(trackId, one);
            }
        }
    }

    public static class ListenReducer extends Reducer<Text, IntWritable,
Text, IntWritable> {
        private IntWritable totalPlays = 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();
            }
            totalPlays.set(sum);
            context.write(key, totalPlays);
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "music track analysis");
        job.setJarByClass(MusicTrackAnalysis.class);
        job.setMapperClass(ListenMapper.class);
        job.setReducerClass(ListenReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new
org.apache.hadoop.fs.Path(args[0]));
        FileOutputFormat.setOutputPath(job, new
org.apache.hadoop.fs.Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Input: User listening data. Example:

```
user1,trackA,2023-01-01 10:00:00
user2,trackB,2023-01-01 10:05:00
user1,trackA,2023-01-01 10:10:00
user3,trackC,2023-01-01 10:15:00
user2,trackB,2023-01-01 10:20:00
user1,trackA,2023-01-01 10:25:00
```

Expected Output:

```
trackA  3
trackB  2
trackC  1
```

Lab 11: Develop a MapReduce program to find the frequency of books published each year and find in which year maximum number of books were published using the given data.

Title: Book Publication Analysis using MapReduce

Aim: To determine the frequency of book publications per year and identify the year with the highest number of publications.

Procedure:

1. Write a Mapper that reads book data (book ID, title, author, year). It emits the year as the key and 1 as the value.
2. Write a Reducer that receives the year and a list of 1s. It sums the 1s to get the publication count for each year and emits the year and the count.
3. (Optional) A second MapReduce job can be used to find the year with the maximum count, or this logic can be included in the reducer with a global variable.
4. Configure and run the MapReduce job(s).

Source Code:

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.conf.Configuration;

public class BookPublicationAnalysis {

    public static class PublicationMapper extends Mapper<LongWritable, Text,
Text, IntWritable> {
        private Text year = new Text();
        private IntWritable one = new IntWritable(1);

        public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
            String line = value.toString();
            String[] parts = line.split(","); // Assuming comma-separated:
bookID, title, author, year
            if (parts.length == 4) {
                year.set(parts[3].trim()); // Year of publication
```

```
                context.write(year, one);
            }
        }
    }

    public static class PublicationReducer extends Reducer<Text, IntWritable,
Text, IntWritable> {
        private IntWritable bookCount = new IntWritable();

        public void reduce(Text key, Iterable<IntWritable> values, Context
context) throws IOException, InterruptedException {
            int count = 0;
            for (IntWritable value : values) {
                count += value.get();
            }
            bookCount.set(count);
            context.write(key, bookCount);
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "book publication analysis");
        job.setJarByClass(BookPublicationAnalysis.class);
        job.setMapperClass(PublicationMapper.class);
        job.setReducerClass(PublicationReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new
org.apache.hadoop.fs.Path(args[0]));
        FileOutputFormat.setOutputPath(job, new
org.apache.hadoop.fs.Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Input: A dataset containing book information. Example:

```
1,BookA,Author1,2020
2,BookB,Author2,2021
3,BookC,Author1,2020
4,BookD,Author3,2022
5,BookE,Author2,2021
6,BookF,Author3,2020
```

Expected Output:

```
2020    3
2021    2
2022    1
```

Lab 12: Develop a MapReduce program to analyze Titanic ship data and to find the average age of the people (both male and female) who died in the tragedy. How many persons are survived in each class.

Title: Titanic Data Analysis using MapReduce

Aim: To analyze the Titanic dataset to calculate the average age of male and female passengers who died and the number of survivors in each passenger class.

Procedure:

1. Obtain the Titanic dataset.
2. Write a Mapper that reads passenger records. It emits key-value pairs for two separate analyses:
   - For average age of deceased: Key = "died_" + gender, Value = age.
   - For survival by class: Key = "class_" + passengerClass, Value = 1 if survived, 0 if not.
3. Write a Reducer that performs two calculations:
   - For each "died_gender" key, calculate the average age.
   - For each "class_passengerClass" key, sum the values to get the number of survivors.
4. Configure and run the MapReduce job.

Source Code:

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.conf.Configuration;

public class TitanicAnalysis {

    public static class TitanicMapper extends Mapper<LongWritable, Text,
Text, IntWritable> {
        private Text keyOut = new Text();
        private IntWritable valueOut = new IntWritable();

        public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
            String line = value.toString();
```

```
            String[] parts = line.split(","); // Assuming comma-separated:
PassengerId,Survived,Pclass,Name,Sex,Age,...
            if (parts.length > 5) { // Ensure enough fields exist
                try {
                    int survived = Integer.parseInt(parts[1].trim());
                    int pclass = Integer.parseInt(parts[2].trim());
                    String sex = parts[4].trim();
                    String ageStr = parts[5].trim();
                    double age = ageStr.isEmpty() ? -1 :
Double.parseDouble(ageStr); // Handle missing age

                    // Analyze average age of deceased
                    if (survived == 0 && age != -1) { // Only consider known
ages of deceased
                        keyOut.set("died_" + sex);
                        valueOut.set((int) age);
                        context.write(keyOut, valueOut);
                    }

                    // Analyze survival by class
                    keyOut.set("class_" + pclass);
                    valueOut.set(survived);
                    context.write(keyOut, valueOut);

                } catch (NumberFormatException e) {
                    System.err.println("Error parsing data in line: " + line
+ " : " + e.getMessage());
                }
            }
        }
    }

    public static class TitanicReducer extends Reducer<Text, IntWritable,
Text, Text> {
        private Text resultValue = new Text();

        public void reduce(Text key, Iterable<IntWritable> values, Context
context) throws IOException, InterruptedException {
            if (key.toString().startsWith("died_")) {
                // Calculate average age of deceased
                int sum = 0;
                int count = 0;
                for (IntWritable value : values) {
                    sum += value.get();
                    count++;
                }
                double averageAge = (count > 0) ? (double) sum / count : 0;
                resultValue.set("Average Age: " + averageAge);
                context.write(key, resultValue);
            } else if (key.toString().startsWith("class_")) {
                // Calculate number of survivors in each class
                int survivors = 0;
                for (IntWritable value : values) {
                    survivors += value.get();
                }
                resultValue.set("Survivors: " + survivors);
                context.write(key, resultValue);
            }
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "titanic analysis");
        job.setJarByClass(TitanicAnalysis.class);
        job.setMapperClass(TitanicMapper.class);
        job.setReducerClass(TitanicReducer.class);
```

```
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new
org.apache.hadoop.fs.Path(args[0]));
        FileOutputFormat.setOutputPath(job, new
org.apache.hadoop.fs.Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Input: Titanic dataset (e.g., CSV file).

Expected Output:

```
died_male    Average Age: 30.0  (Example)
died_female  Average Age: 25.0  (Example)
class_1      Survivors: 200     (Example)
class_2      Survivors: 150     (Example)
class_3      Survivors: 100     (Example)
```

Lab 13: Develop a MapReduce program to analyze Uber data set to find the days on which each basement has more trips using the given dataset.

Title: Uber Trip Analysis using MapReduce

Aim: To analyze Uber trip data to determine the days on which each basement location has the most trips.

Procedure:

1. Obtain the Uber dataset.
2. Write a Mapper that reads Uber trip records (date, basement, trips). It emits the basement as the key and the date and number of trips as the value.
3. Write a Reducer that receives the basement and a list of (date, trips) pairs. It finds the date with the maximum number of trips for that basement and outputs the basement and that date.
4. Configure and run the MapReduce job.

Source Code:

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.conf.Configuration;

public class UberTripAnalysis {

    public static class UberMapper extends Mapper<LongWritable, Text, Text,
Text> {
        private Text basement = new Text();
        private Text dateAndTrips = new Text();

        public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
            String line = value.toString();
            String[] parts = line.split(","); // Assuming comma-separated:
date, basement, trips
            if (parts.length == 3) {
                basement.set(parts[1].trim());
                dateAndTrips.set(parts[0].trim() + "," + parts[2].trim()); //
date,trips
                context.write(basement, dateAndTrips);
```

```
            }
        }
    }

    public static class UberReducer extends Reducer<Text, Text, Text, Text> {
        private Text maxTripDate = new Text();

        public void reduce(Text key, Iterable<Text> values, Context context)
throws IOException, InterruptedException {
            String maxDate = "";
            int maxTrips = -1;
            for (Text value : values) {
                String[] parts = value.toString().split(",");
                String date = parts[0];
                int trips = Integer.parseInt(parts[1]);
                if (trips > maxTrips) {
                    maxTrips = trips;
                    maxDate = date;
                }
            }
            maxTripDate.set(maxDate);
            context.write(key, maxTripDate);
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "uber trip analysis");
        job.setJarByClass(UberTripAnalysis.class);
        job.setMapperClass(UberMapper.class);
        job.setReducerClass(UberReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);
        FileInputFormat.addInputPath(job, new
org.apache.hadoop.fs.Path(args[0]));
        FileOutputFormat.setOutputPath(job, new
org.apache.hadoop.fs.Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Input: Uber trip data. Example:

```
2023-01-01,BasementA,100
2023-01-02,BasementA,120
2023-01-01,BasementB,80
2023-01-03,BasementA,130
2023-01-02,BasementB,90
```

Expected Output: The date with the most trips for each basement. Example:

```
BasementA    2023-01-03
BasementB    2023-01-02
```

Lab 14: Develop a program to calculate the maximum recorded temperature by yearwise for the weather dataset in Pig Latin

Title: Maximum Temperature per Year using Pig Latin

Aim: To calculate the maximum recorded temperature for each year from a weather dataset using Apache Pig.

Procedure:

1. Write a Pig Latin script.
2. Load the weather data into a relation.
3. Group the data by year.
4. For each group, find the maximum temperature.
5. Display (dump) or store the results.

Source Code:

```
-- Load the weather data (assuming comma-separated: year, temperature)
weather_data = LOAD 'weather.txt' USING PigStorage(',') AS (year:chararray,
temperature:int);

-- Group the data by year
grouped_data = GROUP weather_data BY year;

-- Calculate the maximum temperature for each year
max_temp = FOREACH grouped_data GENERATE group,
MAX(weather_data.temperature);

-- Dump the results to the console
DUMP max_temp;

-- Store the results (optional)
-- STORE max_temp INTO 'max_temperatures' USING PigStorage(',');
```

Input: A weather dataset (e.g., CSV file). Example:

```
2020,25
2020,30
2021,28
2021,32
2022,27
2022,35
```

Expected Output: The maximum temperature for each year (displayed on the console). Example:

```
(2020,30)
(2021,32)
(2022,35)
```

Lab 15: Write queries to sort and aggregate the data in a table using HiveQL.

Title: Data Sorting and Aggregation using HiveQL

Aim: To write HiveQL queries to sort and aggregate data in a table. (Provide a specific table schema and data for more precise queries.) I will provide general examples.

Procedure:

1. Start the Hive CLI or Beeline.
2. Create a table or use an existing table.
3. Write HiveQL queries to perform sorting and aggregation.
4. Execute the queries and view the results.

Source Code: (HiveQL Queries)

Example 1: Sorting

```
-- Create a sample table (if it doesn't exist)
CREATE TABLE IF NOT EXISTS employees (
    id INT,
    name STRING,
    age INT,
    department STRING,
    salary DECIMAL(10, 2)
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';

-- Load sample data (if the table is empty)
LOAD DATA INPATH 'employees.txt' INTO TABLE employees;

-- Query to sort employees by salary in descending order
SELECT id, name, salary
FROM employees
ORDER BY salary DESC;

-- Query to sort employees by department and then by age
SELECT id, name, department, age
FROM employees
ORDER BY department ASC, age DESC;
```

Example 2: Aggregation

```
-- Query to find the average salary of all employees
SELECT AVG(salary) AS average_salary
FROM employees;

-- Query to find the number of employees in each department
```

```
SELECT department, COUNT(*) AS employee_count
FROM employees
GROUP BY department;

-- Query to find the maximum and minimum salary in each department
SELECT department, MAX(salary) AS max_salary, MIN(salary) AS min_salary
FROM employees
GROUP BY department;

-- Query to find the total salary for each department, only for
departments with more than 2 employees
SELECT department, SUM(salary) AS total_salary
FROM employees
GROUP BY department
HAVING COUNT(*) > 2;
```

Input: A table in Hive. For the examples above, the input file 'employees.txt' would contain data like:

```
1,Alice,30,Sales,50000.00
2,Bob,25,Marketing,45000.00
3,Charlie,35,Sales,60000.00
4,David,28,Marketing,52000.00
5,Eve,40,HR,55000.00
6,Frank,32,Sales,58000.00
7,Grace,27,Marketing,48000.00
```

Expected Output: The output depends on the specific query. The examples above would produce results like:

Example 1 Output:

```
3       Charlie 60000.00
6       Frank   58000.00
1       Alice   50000.00
4       David   52000.00
2       Bob     45000.00
7       Grace   48000.00
5       Eve     55000.00
```

Example 2 Output:

```
48571.4286
```

Sales 3 Marketing 3 HR 1

```
Sales    60000.00  50000.00
Marketing 52000.00  45000.00
HR        5500
```