Big Data Analytics Lab (PMC112)

LAB MANUAL (PART- I)

Coding

1. Installation of Hadoop Framework.

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

Solution:

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

public class WordFrequency {

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

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

        private String targetWord;

        @Override
        protected void setup(Context context) {
            Configuration conf = context.getConfiguration();
            targetWord = conf.get("targetWord").toLowerCase(); // Get target word from
configuration
        }

        @Override
        public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
            String line = value.toString().toLowerCase(); // Convert the line to lowercase
            String[] tokens = line.split("\\s+"); // Split by whitespace

            for (String token : tokens) {
                if (token.equals(targetWord)) {
                    word.set(targetWord);
                    context.write(word, one);
                }
            }
        }
    }
```

```
    // Reducer Class
    public static class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable>
{
        private IntWritable result = new IntWritable();

        @Override
        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);
        }
    }

    // Main Method
    public static void main(String[] args) throws Exception {
        if (args.length < 3) {
            System.err.println("Usage: WordFrequency <input path> <output path> <word>");
            System.exit(-1);
        }

        Configuration conf = new Configuration();
        conf.set("targetWord", args[2]); // Set the target word

        Job job = Job.getInstance(conf, "Word Frequency");
        job.setJarByClass(WordFrequency.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

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

3. Write a program to implement a word count program using MapReduce.

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

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 {
            String[] words = value.toString().split("\\s+");
            for (String str : words) {
                word.set(str);
                context.write(word, one);
            }
        }
    }

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

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

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

4. Develop a MapReduce program to find the maximum temperature in each year.

Solution:

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

import java.io.IOException;

public class MaxTemperature {

    // Mapper Class
    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[] parts = value.toString().split("\\s+");
            if (parts.length == 2) {
                year.set(parts[0]);  // Extract year
                temperature.set(Integer.parseInt(parts[1])); // Extract temperature
                context.write(year, temperature);
            }
        }
    }

    // Reducer Class
    public static class MaxTemperatureReducer extends Reducer<Text, IntWritable, Text,
IntWritable> {
        public void reduce(Text key, Iterable<IntWritable> values, Context context) throws
IOException, InterruptedException {
            int maxTemp = Integer.MIN_VALUE;
            for (IntWritable val : values) {
                maxTemp = Math.max(maxTemp, val.get());
            }
            context.write(key, new IntWritable(maxTemp));
        }
    }

    // Driver Code
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Maximum Temperature");

        job.setJarByClass(MaxTemperature.class);
        job.setMapperClass(MaxTemperatureMapper.class);
        job.setReducerClass(MaxTemperatureReducer.class);
```

```
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

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

5. Develop a MapReduce program to find the grades of students.

Solution:

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

// Main Class containing Mapper, Reducer, and Driver
public class StudentGradeMapReduce {

    // Mapper Class
    public static class GradeMapper extends Mapper<Object, Text, Text, Text> {
        public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
            String[] parts = value.toString().split("\\s+");
            if (parts.length == 3) {
                String studentId = parts[0];
                String name = parts[1];
                int marks = Integer.parseInt(parts[2]);

                String grade;
                if (marks >= 90) grade = "A";
                else if (marks >= 80) grade = "B";
                else if (marks >= 70) grade = "C";
                else if (marks >= 60) grade = "D";
                else grade = "F";

                context.write(new Text(studentId), new Text(name + "\t" + grade));
            }
        }
    }

    // Reducer Class
    public static class GradeReducer extends Reducer<Text, Text, Text, Text> {
        public void reduce(Text key, Iterable<Text> values, Context context) throws
IOException, InterruptedException {
            for (Text val : values) {
                context.write(key, val);
            }
```

```
        }
    }

    // Driver Code
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Student Grades");

        job.setJarByClass(StudentGradeMapReduce.class);
        job.setMapperClass(GradeMapper.class);
        job.setReducerClass(GradeReducer.class);

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(Text.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

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

6. Develop a MapReduce to find the maximum electrical consumption in each year given electrical consumption for each month in each year. (using Python)

electricity.txt

```
Year Month Consumption
2020 Jan 320
2020 Feb 400
2020 Mar 280
2020 Apr 500
2020 May 450
2020 Jun 380
2021 Jan 300
2021 Feb 340
2021 Mar 480
2021 Apr 420
2021 May 410
2021 Jun 390
```

(Sample Dataset)

Solution:

Mapper.py

```
import sys

for line in sys.stdin:
    parts = line.strip().split()
    if len(parts) == 3:
        year, month, consumption = parts
        try:
            consumption = int(consumption)
            print(f"{year}\t{consumption}")
        except ValueError:
            pass  # Ignore lines with invalid numbers
```

Reducer.py

```
import sys

current_year = None
max_consumption = 0

for line in sys.stdin:
    year, consumption = line.strip().split("\t")
    consumption = int(consumption)

    if current_year == year:
        max_consumption = max(max_consumption, consumption)
    else:
        if current_year:
            print(f"{current_year}\t{max_consumption}")  # Print the previous year's max
        current_year = year
        max_consumption = consumption  # Reset for new year

if current_year:
    print(f"{current_year}\t{max_consumption}")  # Print the last year's max
```

1. Upload Input File to HDFS

```
hdfs dfs -mkdir -p /user/hadoop/input
hdfs dfs -put electricity.txt /user/hadoop/input/
```

2. Run the MapReduce Job

```
hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-*.jar \
   -input /user/hadoop/input/electricity.txt \
   -output /user/hadoop/output \
   -mapper "mapper.py" \
   -reducer "reducer.py" \
   -file mapper.py \
   -file reducer.py
```

3. View the Output

```
hdfs dfs -cat /user/hadoop/output/part-00000
```

Expected Output

```
2020    500
2021    480
```

7. Experiment on Hadoop Map-Reduce and PySpark: -Implementing simple algorithms in Map-Reduce: Matrix multiplication.

Solution:

```
from pyspark.sql import SparkSession

# Initialize Spark Session
spark = SparkSession.builder.appName("MatrixMultiplication").getOrCreate()

# Read input data from HDFS
lines = spark.sparkContext.textFile("hdfs:///user/hadoop/input/matrix.txt")

# Parse input lines
def parse_line(line):
    parts = line.split()
    return parts[0], int(parts[1]), int(parts[2]), int(parts[3])

matrix_rdd = lines.map(parse_line)

# Separate Matrix A and B
A = matrix_rdd.filter(lambda x: x[0] == "A").map(lambda x: (x[1], x[2], x[3]))
B = matrix_rdd.filter(lambda x: x[0] == "B").map(lambda x: (x[1], x[2], x[3]))

# Transform into key-value pairs
A_transformed = A.flatMap(lambda x: [((x[0], k), ('A', x[1], x[2])) for k in range(2)])
B_transformed = B.flatMap(lambda x: [((k, x[1]), ('B', x[0], x[2])) for k in range(2)])

# Join Matrices on (i, k)
joined = A_transformed.union(B_transformed).groupByKey()

# Matrix Multiplication Logic
def multiply(values):
    A_values = [v for v in values if v[0] == 'A']
    B_values = [v for v in values if v[0] == 'B']
    return sum(a[2] * b[2] for a in A_values for b in B_values if a[1] == b[1])

result = joined.mapValues(multiply)

# Save the result to HDFS
result.saveAsTextFile("hdfs:///user/hadoop/output")

# Stop Spark
spark.stop()
```

```
from pyspark.sql import SparkSession

# Initialize Spark Session
spark = SparkSession.builder.appName("MatrixMultiplication").getOrCreate()

# Define Matrices as RDDs
A = spark.sparkContext.parallelize([
    (0, 0, 1), (0, 1, 2),
    (1, 0, 3), (1, 1, 4)
])

B = spark.sparkContext.parallelize([
    (0, 0, 5), (0, 1, 6),
    (1, 0, 7), (1, 1, 8)
])

# Convert (i, j, value) into key-value pairs ((i, k), (matrix, j, value))
A_transformed = A.flatMap(lambda x: [((x[0], k), ('A', x[1], x[2])) for k in range(2)])
B_transformed = B.flatMap(lambda x: [((k, x[1]), ('B', x[0], x[2])) for k in range(2)])

# Join A and B on (i, k)
joined = A_transformed.union(B_transformed).groupByKey()

# Compute Matrix Multiplication
def multiply(values):
    A_values = [v for v in values if v[0] == 'A']
    B_values = [v for v in values if v[0] == 'B']
    return sum(a[2] * b[2] for a in A_values for b in B_values if a[1] == b[1])

result = joined.mapValues(multiply)

# Display the Output
for row in result.collect():
    print(row)
```

8. Write queries to sort and aggregate the data in a table using HiveQL.

Step 1: Create a Sample Table using (CREATE TABLE)

Step 2: Load Data into the Table using LOAD (considering you already have a csv file)

Step 3: Sort by field name in Descending Order (ORDER BY)

Step 4: Aggregation Queries in Hive (GROUP BY)

Step 5: Sorting Aggregated Data (GROUP BY, ORDER BY)

9. Develop a Java application to find the maximum temperature using Spark.

Spark JAVA Code:

```
import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.SparkConf;
import scala.Tuple2;

public class MaxTemperature {
    public static void main(String[] args) {
        // Step 1: Create Spark Configuration
        SparkConf conf = new SparkConf().setAppName("MaxTemperature").setMaster("local");
        JavaSparkContext sc = new JavaSparkContext(conf);

        // Step 2: Load Input Data (Assuming args[0] is the input file)
        JavaRDD<String> data = sc.textFile(args[0]);

        // Step 3: Convert Data into Key-Value Pairs (Year, Temperature)
        JavaPairRDD<String, Integer> yearTempPairs = data.mapToPair(line -> {
            String[] parts = line.split("\\s+");
            return new Tuple2<>(parts[0], Integer.parseInt(parts[1]));
        });

        // Step 4: Find Maximum Temperature for Each Year
        JavaPairRDD<String, Integer> maxTemperatures =
yearTempPairs.reduceByKey(Integer::max);

        // Step 5: Save the Output (args[1] is the output path)
        maxTemperatures.saveAsTextFile(args[1]);

        // Step 6: Stop Spark Context
        sc.close();
    }
```

****