BDA Codes

Steps to execute MapReduce Programs

1. Create one folder on Desktop-”WordCountTutorial”
    - Paste WordCount.java file
    - Create a folder named”Input_Data”-&gt;Create Input.txt file (enter some words)
    - Create a folder name “tutorial_classes

2. export HADOOP_CLASSPATH=$(hadoop classpath)
3. echo $HADOOP_CLASPATH
4. hadoop fs -mkdir /WordCountTutorial
5. hadoop fs -mkdir /WordCountTutorial/input
6. hadoop fs -put /home/vignan/WordCountTutorial/Input_Data/Input.txt /WordCountTutorial/input
7. Change the current directory to the tutorial directory
cd ‘/home/vignan/WordCountTutorial’;
vignan@vignan-HP-Compaq-8200-Elite-SFF-PC:~/vignan/WordCountTutorial$
8. Compile the java code
javac -classpath ${HADOOP_CLASSPATH} -d /home/vignan/WordCountTutorial/tutorial_classes /home/vignan/WordCountTutorial/WordCount.java
9. Put the output files in one JAR file
jar -cvf firstTutorial.jar -C tutorial_classes/ .
10. Run JAR file
hadoop jar /home/vignan/WordCountTutorial/firstTutorial.jar WordCount /WordCountTutorial/input /WordCountTutorial/output
11. See the output
hadoop dfs -cat /WordCountTutorial/output/*

Question-1

```
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 StudentGradeClassification {

  // Mapper Class
  public static class GradeMapper extends Mapper<Object, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text grade = new Text();

    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
      // Input format: StudentID, Marks
```

```
            String[] tokens = value.toString().split(",");
            if (tokens.length == 2) {
                try {
                    int marks = Integer.parseInt(tokens[1].trim());
                    if (marks >= 85) {
                        grade.set("A");
                    } else if (marks >= 70) {
                        grade.set("B");
                    } else if (marks >= 50) {
                        grade.set("C");
                    } else {
                        grade.set("D");
                    }
                    context.write(grade, one);
                } catch (NumberFormatException e) {
                    System.err.println("Invalid marks entry: " + tokens[1]);
                }
            }
        }
    }

    // Reducer Class
    public static class GradeReducer 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);
        }
    }

    // Driver Class
    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.println("Usage: StudentGradeClassification <input path> <output path>");
            System.exit(-1);
        }

        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Student Grade Classification");

        job.setJarByClass(StudentGradeClassification.class);
        job.setMapperClass(GradeMapper.class);
        job.setCombinerClass(GradeReducer.class);
        job.setReducerClass(GradeReducer.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);
    }
}
```

Question-2

```
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;

public class HighestSalaryEmployee {

    // Mapper Class
    public static class SalaryMapper extends Mapper<Object, Text, Text, Text> {
        private Text salary = new Text();
        private Text employeeDetails = new Text();

        public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
            // Input format: eid, name, age, salary
            String[] tokens = value.toString().split(",");
            if (tokens.length == 4) {
                String eid = tokens[0].trim();
                String name = tokens[1].trim();
                String age = tokens[2].trim();
                String salaryValue = tokens[3].trim();

                // Emit salary as the key and employee details as the value
                salary.set(salaryValue);
                employeeDetails.set(eid + "," + name + "," + salaryValue);
                context.write(new Text("HighestSalary"), employeeDetails);
            }
        }
    }

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

        public void reduce(Text key, Iterable<Text> values, Context context)
                throws IOException, InterruptedException {
```

```
            String highestSalaryEmployee = "";
            double highestSalary = 0.0;

            // Compare salaries to find the highest
            for (Text val : values) {
                String[] tokens = val.toString().split(",");
                if (tokens.length == 3) {
                    double salary = Double.parseDouble(tokens[2].trim());
                    if (salary > highestSalary) {
                        highestSalary = salary;
                        highestSalaryEmployee = val.toString();
                    }
                }
            }

            result.set(highestSalaryEmployee);
            context.write(new Text("Employee with Highest Salary:"), result);
        }
    }

    // Driver Class
    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.println("Usage: HighestSalaryEmployee <input path> <output path>");
            System.exit(-1);
        }

        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Find Highest Salary Employee");

        job.setJarByClass(HighestSalaryEmployee.class);
        job.setMapperClass(SalaryMapper.class);
        job.setReducerClass(SalaryReducer.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);
    }
}
```

Question-6

```
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.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;

public class CustomerTransactionCount {

    // Mapper Class
    public static class TransactionMapper extends Mapper<Object, Text, Text, IntWritable> {
        private final static IntWritable one = new IntWritable(1);
        private Text customerID = new Text();

        @Override
        public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
            // Split the input line by spaces
            String[] fields = value.toString().split("\\s+");

            // Extract Customer ID (assumes it's the first field in the record)
            if (fields.length >= 4) { // Ensure there are enough fields in the record
                customerID.set(fields[0]); // Customer ID is in the first field
                context.write(customerID, one); // Emit <Customer ID, 1> for each transaction
            }
        }
    }

    // Reducer Class
    public static class TransactionReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
        private IntWritable totalTransactions = new IntWritable();

        @Override
        public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,
InterruptedException {
            int sum = 0;

            // Sum the values (which are all 1's)
            for (IntWritable val : values) {
                sum += val.get();
            }

            totalTransactions.set(sum); // Set the total transaction count
            context.write(key, totalTransactions); // Emit <Customer ID, Total Transactions>
        }
    }

    // Driver Code
    public static void main(String[] args) throws Exception {
        // Check if proper arguments are passed
        if (args.length != 2) {
            System.err.println("Usage: CustomerTransactionCount <input path> <output path>");
            System.exit(-1);
        }
        // Set up the configuration
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Customer Transaction Count");

        // Set the jar file for the job
        job.setJarByClass(CustomerTransactionCount.class);

        // Set Mapper and Reducer classes
        job.setMapperClass(TransactionMapper.class);
        job.setReducerClass(TransactionReducer.class);

        // Set the output key and value types
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        // Set input and output paths
        FileInputFormat.addInputPath(job, new Path(args[0])); // Input file path
        FileOutputFormat.setOutputPath(job, new Path(args[1])); // Output directory path

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

Question-9

```
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.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;

public class RestaurantRating {

    // Mapper Class
    public static class RatingMapper extends Mapper<Object, Text, Text, IntWritable> {
        private final static IntWritable one = new IntWritable(1);
        private Text ratingCategory = new Text();

        @Override
        public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
            // Split the input line by space or tab
            String[] fields = value.toString().split("\\s+");

            // Ensure the input has the rating field
            if (fields.length >= 2) {
                int rating = Integer.parseInt(fields[1]);
                // Classify the rating into categories
                if (rating == 1) {
                    ratingCategory.set("Very Poor");
                } else if (rating == 2) {
                    ratingCategory.set("Poor");
                } else if (rating == 3) {
                    ratingCategory.set("Average");
                } else if (rating == 4) {
                    ratingCategory.set("Good");
                } else if (rating == 5) {
                    ratingCategory.set("Excellent");
                } else {
                    return; // If rating is out of range, we ignore this record
                }

                // Emit the rating category and a count of 1 for each rating occurrence
                context.write(ratingCategory, one);
            }
        }
    }

    // Reducer Class
    public static class RatingReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
        private IntWritable totalRatings = new IntWritable();

        @Override
        public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,
InterruptedException {
            int sum = 0;

            // Sum the values (each value is 1, so we are counting occurrences)
            for (IntWritable val : values) {
                sum += val.get();
            }

            totalRatings.set(sum); // Set the total count for the rating category
            context.write(key, totalRatings); // Emit the category and its total count
        }
    }

    // Driver Code
    public static void main(String[] args) throws Exception {
        // Check if proper arguments are passed
        if (args.length != 2) {
            System.err.println("Usage: RestaurantRating <input path> <output path>");
            System.exit(-1);
        }

        // Set up the configuration
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Restaurant Rating Classification");
```

```
        // Set the jar file for the job
        job.setJarByClass(RestaurantRating.class);

        // Set Mapper and Reducer classes
        job.setMapperClass(RatingMapper.class);
        job.setReducerClass(RatingReducer.class);

        // Set the output key and value types
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        // Set input and output paths
        FileInputFormat.addInputPath(job, new Path(args[0])); // Input file path
        FileOutputFormat.setOutputPath(job, new Path(args[1])); // Output directory path

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

Steps to enter hive shell:

start-all.sh
hive

Question-3

Tables creation:

- create table if not exists Doctors (DoctorName string, Specialization string, ExperienceYears int, ContactNumber string) row format delimited fields terminated by ',' stored as a textfile;
- create table if not exists Patients (PatientOpNumber string, Name string, Age int, Gender string, Contact string) row format delimited fields terminated by ',' stored as a textfile;

Data Insertion: Insert some sample data in local textfiles with comma separated values.

- load data local inpath 'path_to_doctors.txt' into table Doctors;
- load data local inpath 'path_to_patients.txt' into table Patients;

Verify data :

- select * from Doctors;
- select * from Patients;

Question-4

Table creation:

- create table if not exists Instagram(PostId int, Likes int, Comments int) row format delimited fields terminated by ',' stored as textfile;

Data insertion: Insert some data in a local txtfile with comma separated values.

- Load data local inpath'path_to_instagram.txt' into table Instagram;

Queries:

i. select count(PostId) as TotalPosts from Instagram;
ii. select avg(Likes) as AverageLikes, avg(Comments) as AverageComments from Instagram;

Question-11

Table creation

- create table if not exists Books(BookId int, Title string, Author string, Genre string, Rating float) row format delimited fields terminated by ',' stored as textfile;
- describe Books;

Data insertion: Insert some data in a local txtfile with comma separated values.

- Load data local inpath’path_to_books.txt’ into table Books;

Queries:

- select * from Books where Rating > 4.0;
- select * from Books where Genre != ‘Fiction’;

Steps to enter grunt shell:

```
start-all.sh
pig
```

Question-5

Question-13:

```
# Step 1: Install PySpark and set up environment
!apt-get install openjdk-17-jdk-headless -qq > /dev/null
!wget -q https://dlcdn.apache.org/spark/spark-3.5.3/spark-3.5.3-bin-hadoop3.tgz
!tar xf spark-3.5.3-bin-hadoop3.tgz
!pip install -q findspark

import os
import findspark

findspark.init()

# Step 2: Set environment variables for Spark and Hadoop
os.environ["SPARK_HOME"] = "/content/spark-3.5.3-bin-hadoop3"  # Corrected path for Spark 3.5.3
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-17-openjdk-amd64"  # Corrected to Java 17

# Step 3: Initialize SparkContext
from pyspark.sql import SparkSession

# Ensure SPARK_HOME is set correctly and without '/bin'
spark = SparkSession.builder.master("local[*]").appName("TCS Selection").getOrCreate()
sc = spark.sparkContext

# Step 4: Sample data for selected students (student_id, name)
data = [
    (1, "John"),
    (2, "Alice"),
    (3, "Bob"),
    (4, "Charlie"),
    (5, "David")
]

# Create RDD
students_rdd = sc.parallelize(data)

# Step 5: Count total number of students selected for TCS
total_students = students_rdd.count()
print(f"Total number of students selected for TCS: {total_students}")

# Step 6: Sort the selected students by name in ascending order
sorted_students_rdd = students_rdd.sortBy(lambda x: x[1])  # Sorting by name (index 1)
```

```
sorted_students = sorted_students_rdd.collect()

print("Sorted students by name (ascending):")
for student in sorted_students:
    print(student)
```