Database Management Systems

CS-236

Project Report

Team Members:
Ankit Mane [862393644] [amane011@ucr.edu]
Daiyaan Ahmed Shaik [862393960] [dshai005@ucr.edu]

1 Project Description

1.1 Experimental Setup

Installed Hadoop 3.2.1 and the compatible version of Java to create the MapReduce jobs.

1.2 Combining the datasets / Logic for join(s)

The first dataset, 'hotel-bookings' contains data on reservations made between 2015 and 2016. The second dataset 'customer-reservations' offers details on reservations made between 2017 and 2018.

1.2.1 Mapper

Mapper handles the merging and initial data processing for both the hotel booking datasets. It transforms input records into a set of intermediate key-value pairs. We used a seperate set of variables to store the year, month and price for each of the datasets. The Mapper works as follows:

- The map function initially checks if the current line contains **avg_price_per_room** which is in the header row in both the datasets to ensure only data rows are processed.
- We then check if the current line is from dataset 1 or 2 by confirming the number of fields of the current line (**13 for dataset 1, 10 for dataset 2**). Either way, the price per room is parsed from the correct field and multiplied by the number of weekend and weekday nights (**2nd and 3rd fields in dataset 1, 8th and 9th field in dataset 2**).
- The calculated total price is then written to the context with a key combining the year and month and the value being the total price.

After all the lines are processed, the context will contain key-value pairs where the key is a combination of year and month and the value is the total revenue for that booking. This data is then passed to the reducer for further

processing, where revenues for each month will be summed up across all years and then sorted to find the most profitable month.

1.2.2 Reducer

The Reducer takes the intermediate key-value pairs from the Mapper and reduces the values for each key to a single output. The reducer functions as follows:

- The reduce function takes in a key and an iterable of values. The key represents the combination of year and month from the Mapper output, and the values are a set of numbers that represent the total revenues for each booking corresponding to that key.
- We then iterate over each value from the values, which contains all the total revenues for bookings of that month across all years. It adds up these revenues to a total revenue variable.
- Lastly, it writes the key (year and month combination) and the total revenue for that key to the context. Therefore, the output will be a list of year and month combinations and the corresponding total revenue for that month across all years.

The output from the Reducer is then written to the final output file on HDFS. The output key-value pairs are sorted by key, so the revenues for each month are in ascending order of year and month.

1.3 Computing the most profitable month

1.3.1 Mapper

The output from the previous reducer is taken as input for the Mapper function. Each line from the input is split into fields based on tab (\t). The first field (**fields[0]**) contains the **month and year combination** and the second field (**fields[1]**) contains the **average revenue for that month**. The Mapper is just preparing the data for the next Reducer.

1.3.2 Reducer

This reducer takes the output from the previous Mapper and determines which month in each year had the highest revenue. It works as follows:

- The Reducer has two HashMaps - where one keeps track of the maximum revenue encountered for each year(**yearMaxRevenue**) and the other keeps track of which month that maximum revenue occurred in (**yearMaxMonth**).
- It takes in a key (the month and year) and a set of values (the revenues for that month). For each key-value group, it calculates the total revenue

for that month and year. If this is the first time it has seen this year or if the total revenue is greater than the current maximum for that year, it updates **yearMaxRevenue** and **yearMaxMonth**

- The cleanup method is called at the end of the reduce phase. In this method, the reducer writes the most profitable month and corresponding revenue for each year to the context.

The reducer will finally output one record for each year in the dataset, with the record containing the most profitable month and the corresponding revenue for that year.

2 Code Snippets

2.1 Mapper for combining datasets

Mapper.java 1

Mapper.java > Mapper > map(LongWritable, Text, Context)

```java
package org.example;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;

import java.io.IOException;

public class Mapper extends org.apache.hadoop.mapreduce.Mapper<LongWritable, Text, Text, LongWritable> {
    public static int d1Year = 4;
    public static int d1Month = 5;
    public static int d1Price = 8;

    public static int d2Year = 3;
    public static int d2Month = 4;
    public static int d2Price = 11;
    public static int d1 = 10;
    public static int d2 = 13;

    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String line = value.toString();
        String[] fields = line.split(regex:",");
        if (line.contains(s:"avg_price_per_room")) {
            return;
        }
        if (fields.length == d1) {
            double price = Double.parseDouble(fields[d1Price]);
            long total = (long) (price * (Integer.parseInt(fields[1]) + Integer.parseInt(fields[2])));
            Text outputKey = new Text(fields[d1Year] + "-" + fields[d1Month]);
            context.write(outputKey, new LongWritable(total));
        } else if (fields.length == d2) {
            double price = Double.parseDouble(fields[d2Price]);
            long total = (long) (price * (Integer.parseInt(fields[7]) + Integer.parseInt(fields[8])));
            Text outputKey = new Text(fields[d2Year] + "-" + fields[d2Month]);
            context.write(outputKey, new LongWritable(total));
        }
    }
}

```

2.2 Reducer for combining datasets

Reducer.java > ...

```java
package org.example;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;

import java.io.IOException;

public class Reducer extends org.apache.hadoop.mapreduce.Reducer<Text, LongWritable, Text, LongWritable> {

    @Override
    public void reduce(Text key, Iterable<LongWritable> values, Context context) throws IOException, InterruptedException {
        long totalRevenue = 0;
        for (LongWritable value : values) {
            totalRevenue += value.get();
        }
        context.write(key, new LongWritable(totalRevenue));
    }
}
```

2.3 Driver code for combining datasets

Driver.java 1

Driver.java > {} org.example

```java
package org.example;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
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 Driver {
    Run | Debug
    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.println(x:"Usage: App <input path> <output path>");
            System.exit(-1);
        }

        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Hotel Revenue");
        job.setJarByClass(Driver.class);
        job.setMapperClass(Mapper.class);
        job.setReducerClass(Reducer.class);

        // Set the types of output key and value for Mapper & Reducer
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(LongWritable.class);  // change here
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(LongWritable.class);  // and here

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

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

```

2.4 Mapper for finding most profitable month

src > main > java > org > example > J TopMonthMapper.java > ...

```java
package org.example;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import java.io.IOException;
import org.apache.hadoop.io.LongWritable;

public class TopMonthMapper extends Mapper<LongWritable, Text, Text, LongWritable> {
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String[] fields = value.toString().split(regex:"\t");
        String monthYear = fields[0];
        long avgRevenue = Long.parseLong(fields[1]);

        context.write(new Text(monthYear), new LongWritable(avgRevenue));
    }
}
```

2.5 Reducer for finding most profitable month

```java
package org.example;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

public class TopMonthReducer extends Reducer<Text, LongWritable, Text, LongWritable> {
    private Map<String, Long> yearMaxRevenue = new HashMap<>();
    private Map<String, String> yearMaxMonth = new HashMap<>();

    @Override
    public void reduce(Text key, Iterable<LongWritable> values, Context context) throws IOException, InterruptedException {
        String keyStr = key.toString();
        String year = keyStr.split(regex:"-")[0];

        long revenueSum = 0;
        for (LongWritable val : values) {
            revenueSum += val.get();
        }

        if (!yearMaxRevenue.containsKey(year) || revenueSum > yearMaxRevenue.get(year)) {
            yearMaxRevenue.put(year, revenueSum);
            yearMaxMonth.put(year, keyStr);
        }
    }

    @Override
    protected void cleanup(Context context) throws IOException, InterruptedException {
        for (Map.Entry<String, String> entry : yearMaxMonth.entrySet()) {
            context.write(new Text(entry.getValue()), new LongWritable(yearMaxRevenue.get(entry.getKey())));
        }
    }
}
```

2.6 Driver code for finding most profitable month

Driver.java M

src > main > java > org > example > Driver.java > ...

```java
package org.example;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
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 Driver {
    Run | Debug
    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.println(x:"Usage: App <input path> <output path>");
            System.exit(-1);
        }

        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Top Month");
        job.setJarByClass(Driver.class);
        job.setMapperClass(TopMonthMapper.class);
        job.setReducerClass(TopMonthReducer.class);

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


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

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

3 Results

3.1 Estimated time

3.1.1 Time taken for combining the datasets

```
Total time spent by all map tasks (ms)=6383
Total time spent by all reduce tasks (ms)=2660
```

3.1.2 Time taken for computing the most profitable months

```
Total time spent by all map tasks (ms)=2341
Total time spent by all reduce tasks (ms)=2443
```

3.2 Most profitable month

The most profitable months for each year calcualted by our MapReduce logic are mentioned below:

- 2015 - August (1698057)
- 2016 - August (2896166)
- 2017 - October (477451)
- 2018 - October (1126826)

4 Team member contribution

4.1 Ankit Mane:

Loaded the datasets into HDFS. Implemented the logic for MapReduce to find the most profitable month of the year for the reservations made by the customers based on revenue from both the given datasets.

4.2 Daiyaan Ahmed Shaik:

Developed the logic for combining both the datasets, modified the dataset for faster execution and made the report.