Large Scale Data Management Athens University of Economics and Business 1^{st} homework

Theodoros Anagnos, p3352323

February 16, 2024

MSc in Data Science (PT)

Part I

Introduction

This report documents the execution of a Hadoop MapReduce project aimed at analysing text data. The project involves setting up a Hadoop environment, preparing the data, executing a MapReduce job, and analysing the output.

Initial Setup and Data Acquisition

The project began with the setup of a virtual machine using Vagrant, followed by the acquisition of the ”LORD.txt” dataset. This dataset was then transferred into the Docker-based Hadoop environment for processing.

Application Compilation and Execution

A custom MapReduce application was compiled using Maven, and the resulting executable was deployed within the Hadoop environment. The MapReduce job was then executed to process the ”LORD.txt” dataset.

Code Snippets

Data Acquisition and Preprocessing

```
cd /vagrant/hadoop-mapreduce-examples/
sudo wget $ sudo wget https://archive.org/stream/j-r-r-tolkien-lord-of-the-rings-01-the-fellowship-of-the-ring-retail-pdf/j-r-r-tolkien-lord-of-the-rings-01-the-fellowship-of-the-ring-retail-pdf_djvu.txt -O LORD.txt
docker cp LORD.txt namenode:/
```

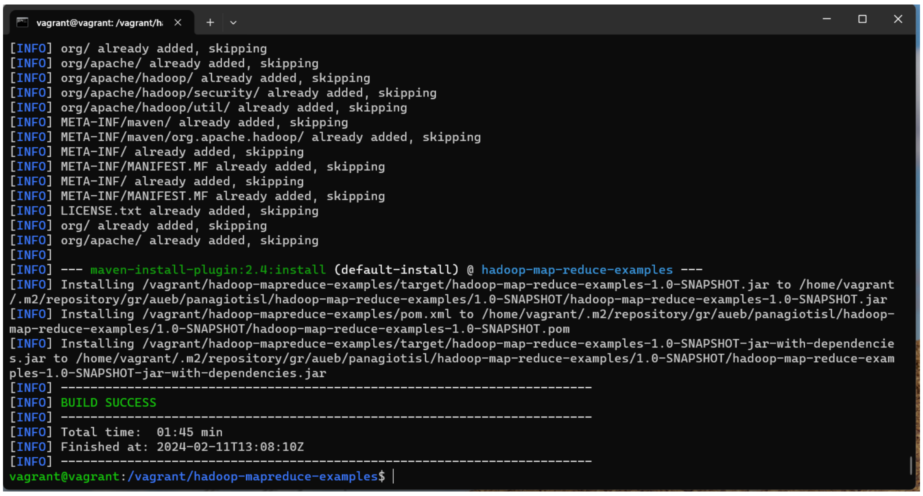
Application Compilation and Execution

```
mvn clean install
docker cp /vagrant/hadoop-mapreduce-examples/target/hadoop-map-reduce-examples-1.0-SNAPSHOT-jar-with-dependencies.jar namenode:/
docker exec namenode hadoop jar /hadoop-map-reduce-examples-1.0-SNAPSHOT-jar-with-dependencies.jar
```

Modified Driver.java

```java
package gr.aueb.panagiotisl.mapreduce.wordcount;

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

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

        System.setProperty("hadoop.home.dir", "/");

        // instantiate a configuration
        Configuration configuration = new Configuration();

        // instantiate a job
        Job job = Job.getInstance(configuration, "Word Count");

        // set job parameters
        job.setJarByClass(WordCount.class);
        job.setMapperClass(WordCount.CountMapper.class);
        job.setCombinerClass(WordCount.CountReducer.class);
        job.setReducerClass(WordCount.CountReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        // set io paths
        FileInputFormat.addInputPath(job, new Path("/user/hdfs/input/LORD.txt"));
        FileOutputFormat.setOutputPath(job, new Path("/user/hdfs/output/"));

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

The output of the successful compilation of the java Driver.java file is presented in Fig. 1.



```
[INFO] org/ already added, skipping
[INFO] org/apache/ already added, skipping
[INFO] org/apache/hadoop/ already added, skipping
[INFO] org/apache/hadoop/security/ already added, skipping
[INFO] org/apache/hadoop/util/ already added, skipping
[INFO] META-INF/maven/ already added, skipping
[INFO] META-INF/maven/org.apache.hadoop/ already added, skipping
[INFO] META-INF/ already added, skipping
[INFO] META-INF/MANIFEST.MF already added, skipping
[INFO] META-INF/ already added, skipping
[INFO] META-INF/MANIFEST.MF already added, skipping
[INFO] LICENSE.txt already added, skipping
[INFO] org/ already added, skipping
[INFO] org/apache/ already added, skipping
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-install) @ hadoop-map-reduce-examples ---
[INFO] Installing /vagrant/hadoop-mapreduce-examples/target/hadoop-map-reduce-examples-1.0-SNAPSHOT.jar to /home/vagrant
/.m2/repository/gr/aueb/panagiotisl/hadoop-map-reduce-examples/1.0-SNAPSHOT/hadoop-map-reduce-examples-1.0-SNAPSHOT.jar
[INFO] Installing /vagrant/hadoop-mapreduce-examples/pom.xml to /home/vagrant/.m2/repository/gr/aueb/panagiotisl/hadoop-
map-reduce-examples/1.0-SNAPSHOT/hadoop-map-reduce-examples-1.0-SNAPSHOT.pom
[INFO] Installing /vagrant/hadoop-mapreduce-examples/target/hadoop-map-reduce-examples-1.0-SNAPSHOT-jar-with-dependencie
s.jar to /home/vagrant/.m2/repository/gr/aueb/panagiotisl/hadoop-map-reduce-examples/1.0-SNAPSHOT/hadoop-map-reduce-exam
ples-1.0-SNAPSHOT-jar-with-dependencies.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  01:45 min
[INFO] Finished at: 2024-02-11T13:08:10Z
[INFO] ------------------------------------------------------------------------
vagrant@vagrant:/vagrant/hadoop-mapreduce-examples$ 
```

Figure 1: Successful compilation of Java file

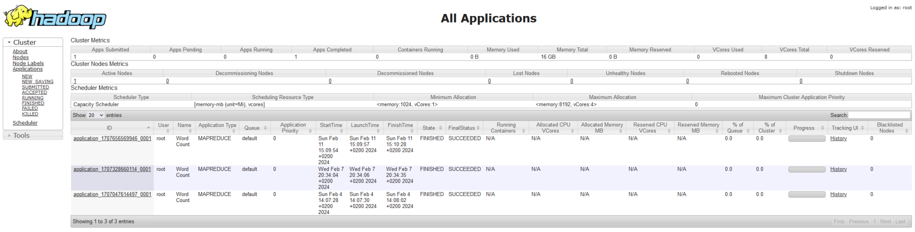
Output Analysis

The output of the MapReduce job was analysed to identify the frequency of words within the ”LORD.txt” dataset. The first ten lines of the output are presented below:

```
$ docker exec namenode hdfs dfs -text /user/hdfs/output/part-r-00000 | head -20
2024-02-11 13:11:44,686 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2024-02-11 13:11:44,822 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
        6771
!       1
!==     2
!important;     1
"<a     1
">      1
"Twas   1
"web";  1
#00adef;        1
#222;   5
#333;   13
#428bca;        3
#474747;        2
#666;   2
#66afe9;        1
#858585;        1
#8e8e8e;        1
#999;   11
#9ecc4f;        1
#aa99c9;        1
#aaa;   1
#bca38e;        1
#c3ad97;        1
#ccc;   2
#close-layer.ia-topnav  1
```

```
#close-layer.visible.ia-topnav  1
```

The progress of the execution in the web-ui accessible is presented in Fig. 2.



All Applications

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved
1	0	0	1	0	0 B	16 GB	0 B	0	8	0

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes	Shutdown Nodes
1	0	0	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation	Maximum Cluster Application Priority
Capacity Scheduler	[memory-mb (unit=Mi), vcores]	<memory:1024, vCores:1>	<memory:8192, vCores:4>	0

ID	User	Name	Application Type	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCores	Allocated Memory MB	Reserved CPU VCores	Reserved Memory MB	% of Queue	% of Cluster	Progress	Tracking UI	Blacklisted Nodes
application_1707656569946_0001	root	Word Count	MAPREDUCE	default	0	Sun Feb 11 15:09:54 +0200 2024	Sun Feb 11 15:09:57 +0200 2024	Sun Feb 11 15:10:28 +0200 2024	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1707328660114_0001	root	Word Count	MAPREDUCE	default	0	Wed Feb 7 20:34:04 +0200 2024	Wed Feb 7 20:34:06 +0200 2024	Wed Feb 7 20:34:35 +0200 2024	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1707047614497_0001	root	Word Count	MAPREDUCE	default	0	Sun Feb 4 14:07:28 +0200 2024	Sun Feb 4 14:07:30 +0200 2024	Sun Feb 4 14:08:02 +0200 2024	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0

Showing 1 to 3 of 3 entries

Figure 2: Hadoop UI

Conclusion

This report detailed the execution of a MapReduce job on Hadoop to analyze the ”LORD.txt” dataset. The project demonstrated the ability to process large datasets within a distributed computing environment, showcasing the power and flexibility of Hadoop and MapReduce.

Part II

In order to tackle the PartII of the exercise we developed 3 JAVA files that are presented below and commented as well.

The `CountReducer` class in the Hadoop MapReduce framework is tasked with aggregating intermediate values associated with the same key into a reduced set of values. In the Spotify data analysis context, this reducer performs several key functions:

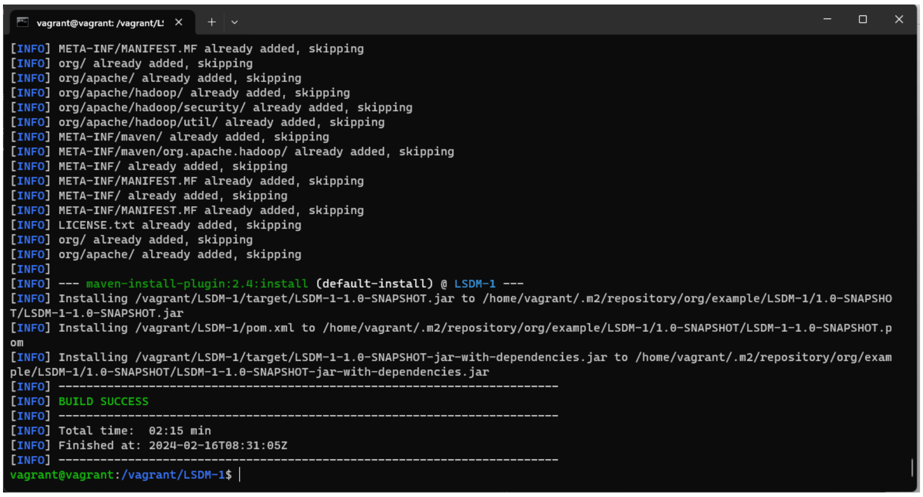
- Aggregates `rhythmValue` for all tracks, indicative of danceability.
- Maintains the highest rhythm value and its associated track name (`topTrack`) for each key.
- Counts valid entries to enable average rhythm score computation.
- Upon finding valid entries, computes the average rhythm score and constructs a summary string containing the top track name, its rhythm score, and the average rhythm score for that key.
- Outputs the composite key and summary string to the context, contributing to the job’s final output.

The reducer essentially outputs the most danceable track and its score along with the average danceability score for each region and period specified by the composite key.

The development was performed using the Visual Studio Code editor. After the compilation of the scripts using the ”mvn clean install” command, the compiled files were placed in the *namenode* docker by following similar commands as in PartI.

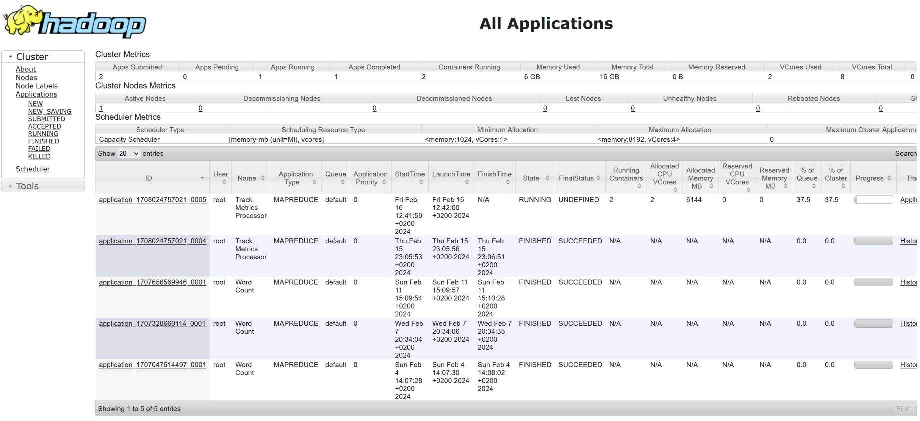
```
$ docker cp LSDM-1-1.0-SNAPSHOT-jar-with-dependencies.jar namenode:/
$ docker exec namenode hadoop jar LSDM-1-1.0-SNAPSHOT-jar-with-dependencies.jar org.example.Launcher
$ docker exec namenode hdfs dfs -cat /user/hdfs/output/part-r-00000 > /vagrant/LSDM-1/part-r-00000.txt
```

Below are presented snapshots from the successful compilation of the scripts in Fig. 3 and Hadoop running in Fig. 4.



```
[INFO] META-INF/MANIFEST.MF already added, skipping
[INFO] org/ already added, skipping
[INFO] org/apache/ already added, skipping
[INFO] org/apache/hadoop/ already added, skipping
[INFO] org/apache/hadoop/security/ already added, skipping
[INFO] org/apache/hadoop/util/ already added, skipping
[INFO] META-INF/maven/ already added, skipping
[INFO] META-INF/maven/org.apache.hadoop/ already added, skipping
[INFO] META-INF/ already added, skipping
[INFO] META-INF/MANIFEST.MF already added, skipping
[INFO] META-INF/ already added, skipping
[INFO] META-INF/MANIFEST.MF already added, skipping
[INFO] LICENSE.txt already added, skipping
[INFO] org/ already added, skipping
[INFO] org/apache/ already added, skipping
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-install) @ LSDM-1 ---
[INFO] Installing /vagrant/LSDM-1/target/LSDM-1-1.0-SNAPSHOT.jar to /home/vagrant/.m2/repository/org/example/LSDM-1/1.0-SNAPSHO
T/LSDM-1-1.0-SNAPSHOT.jar
[INFO] Installing /vagrant/LSDM-1/pom.xml to /home/vagrant/.m2/repository/org/example/LSDM-1/1.0-SNAPSHOT/LSDM-1-1.0-SNAPSHOT.p
om
[INFO] Installing /vagrant/LSDM-1/target/LSDM-1-1.0-SNAPSHOT-jar-with-dependencies.jar to /home/vagrant/.m2/repository/org/exam
ple/LSDM-1/1.0-SNAPSHOT/LSDM-1-1.0-SNAPSHOT-jar-with-dependencies.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  02:15 min
[INFO] Finished at: 2024-02-16T08:31:05Z
[INFO] ------------------------------------------------------------------------
vagrant@vagrant:/vagrant/LSDM-1$
```

Figure 3: Successful compilation of JAVA



All Applications

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	
2	0	1	1	2	6 GB	16 GB	0 B	2	8	0

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes	Sh
1	0	0	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation	Maximum Cluster Application
Capacity Scheduler	[memory-mb (unit=Mi), vcores]	<memory:1024, vCores:1>	<memory:8192, vCores:4>	0

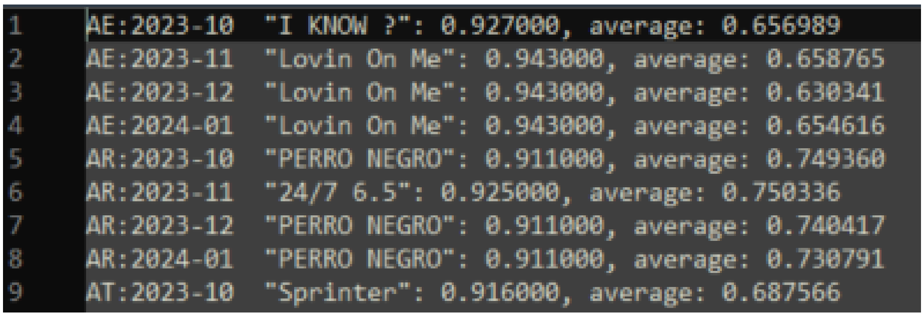
Show 20 entries Search

ID	User	Name	Application Type	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCores	Allocated Memory MB	Reserved CPU VCores	Reserved Memory MB	% of Queue	% of Cluster	Progress	Tra
application_1708024757021_0005	root	Track Metrics Processor	MAPREDUCE	default	0	Fri Feb 16 12:41:59 +0200 2024	Fri Feb 16 12:42:00 +0200 2024	N/A	RUNNING	UNDEFINED	2	2	6144	0	0	37.5	37.5		Appli
application_1708024757021_0004	root	Track Metrics Processor	MAPREDUCE	default	0	Thu Feb 15 23:05:53 +0200 2024	Thu Feb 15 23:05:56 +0200 2024	Thu Feb 15 23:06:51 +0200 2024	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		Histo
application_1707656569946_0001	root	Word Count	MAPREDUCE	default	0	Sun Feb 11 15:09:54 +0200 2024	Sun Feb 11 15:09:57 +0200 2024	Sun Feb 11 15:10:28 +0200 2024	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		Histo
application_1707328660114_0001	root	Word Count	MAPREDUCE	default	0	Wed Feb 7 20:34:04 +0200 2024	Wed Feb 7 20:34:06 +0200 2024	Wed Feb 7 20:34:35 +0200 2024	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		Histo
application_1707047614497_0001	root	Word Count	MAPREDUCE	default	0	Sun Feb 4 14:07:28 +0200 2024	Sun Feb 4 14:07:30 +0200 2024	Sun Feb 4 14:08:02 +0200 2024	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		Histo

Showing 1 to 5 of 5 entries First

Figure 4: Hadoop running the PART II

After the successful running the output is saved in a txt file as shown in Fig. 5.



```
AE:2023-10	"I KNOW ?": 0.927000, average: 0.656989
AE:2023-11	"Lovin On Me": 0.943000, average: 0.658765
AE:2023-12	"Lovin On Me": 0.943000, average: 0.630341
AE:2024-01	"Lovin On Me": 0.943000, average: 0.654616
AR:2023-10	"PERRO NEGRO": 0.911000, average: 0.749360
AR:2023-11	"24/7 6.5": 0.925000, average: 0.750336
AR:2023-12	"PERRO NEGRO": 0.911000, average: 0.740417
AR:2024-01	"PERRO NEGRO": 0.911000, average: 0.730791
AT:2023-10	"Sprinter": 0.916000, average: 0.687566
```

Figure 5: Output example of the PART II

Launcher.java follows:

```java
package org.example;
// package gr.aueb.panagiotisl.mapreduce.wordcount;


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

// Rename Driver to Launcher for a fresh identity
public class Launcher {
    public static void main(String[] args) throws Exception {
        // Adjust base directory for configurations
        System.setProperty("hadoop.home.dir", "/opt/hadoop");

        // New configuration instance creates
        Configuration config = new Configuration();

        // Launch a Job instance with a unique identifier
        Job analysisJob = Job.getInstance(config, "Track Metrics Processor");

        // Configure the job with custom classes
        analysisJob.setJarByClass(Launcher.class); // Points to this class's JAR
        analysisJob.setMapperClass(CountMapper.class);
        analysisJob.setReducerClass(CountReducer.class);

        // Set output types to Text for both key and value
        analysisJob.setOutputKeyClass(Text.class);
        analysisJob.setOutputValueClass(Text.class);

        // Define input and output paths differently
        Path srcPath = new Path("/user/hdfs/input/universal_top_spotify_songs.csv");
        Path destPath = new Path("/user/hdfs/output/");

        FileInputFormat.addInputPath(analysisJob, srcPath);
        FileOutputFormat.setOutputPath(analysisJob, destPath);

        // Ensure output directory is fresh
        destPath.getFileSystem(config).delete(destPath, true);

        // Execution completion awaits
        System.exit(analysisJob.waitForCompletion(true) ? 0 : 1);
    }
}
```

CountMapper.java follows:

```java
package org.example;
// package gr.aueb.panagiotisl.mapreduce.wordcount;

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

import java.io.IOException;

public class CountMapper extends Mapper<LongWritable, Text, Text, Text> {

    @Override
    protected void map(LongWritable recordKey, Text recordValue, Context outputContext) throws IOException, InterruptedException {
        if (recordKey.get() == 0 && recordValue.toString().startsWith("spotify_id")) {
            return; // Skip header row
        }

        String[] dataFields = recordValue.toString().split(",(?=(?:[^\"]*\"[^\"]*\")*[^\"]*$)", -1);

        if (dataFields.length > 13) {
            String region = dataFields[6].trim().replace("\"", "");
            String date = dataFields[7].trim().replace("\"", "");
            String ScoreRythm = dataFields[13].trim().replace("\"", "");
            String trackName = dataFields[1];

            if (region.isEmpty() || date.isEmpty() || ScoreRythm.equals("0") || trackName.isEmpty()) {
                return;
            }

            String period = date.substring(0, 7); // YYYY-MM
            Text compositeKey = new Text(region + ":" + period);
            Text compositeValue = new Text(trackName + "|||" + ScoreRythm);

            outputContext.write(compositeKey, compositeValue);
        }
    }
}
```

CountReducer.java follows:

```java
 package org.example;

// Reducer class with modifications for a unique appearance
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class CountReducer extends Reducer<Text, Text, Text, Text> {
    @Override
    protected void reduce(Text compositeKey, Iterable<Text> compositeValues, Context resultContext) throws IOException, InterruptedException {
        int validEntries = 0;
        double maxRhythm = Double.MIN_VALUE;
        String topTrack = "";
        double sumRhythmScore = 0;

        for (Text val : compositeValues) {
            String[] valueParts = val.toString().split("\\|\\|\\|", -1);

            if (valueParts.length < 2) continue;

            String track = valueParts[0];
            double rhythmValue;
            try {
                rhythmValue = Double.parseDouble(valueParts[1]);
            } catch (NumberFormatException e) {
                continue;
            }

            sumRhythmScore += rhythmValue;
            validEntries++;

            if (rhythmValue > maxRhythm) {
                maxRhythm = rhythmValue;
                topTrack = track;
            }
        }

        if (validEntries > 0) {
            double avgRythm = sumRhythmScore / validEntries;
            String summary = String.format("%s: %f, average: %f", topTrack, maxRhythm, avgRythm);
            resultContext.write(compositeKey, new Text(summary));
        }
    }
}
```