**HPCI Code for Map-Reduce Chaining to answer the following question:**

***What are the top 10 genre console combination which are popular among the buyers based on total sales?***

**Create First Mapper Class: GenresConsoleMapper**

package org.myorg;

//Importing the libraries that contain the classes and methods needed in the Mapper class

import java.io.IOException;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapreduce.\*;

*{Melissa K.and the Heart of Gold Collectors Edition,PC,Adventure,SDP Games,9,10,2,0.18,0.95,4.45,1.13,6.66,13.19,1971}*

//Creating a subclass (child class) called GenresConsoleMapper from a parent class called Mapper using the keyword extends

public class GenresConsoleMapper extends Mapper<LongWritable, Text, Text, DoubleWritable> {

//<Input Key & Value datatype, Output Key & Value datatype>

/\*@override is used to change the original behavior of the method “map” in the parent Class “Mapper” \*/

@Override

// the map method takes three parameters that are a key, a value, and a context as input parameters

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

/\*create a variable line of data type String, convert value into string and store it in line and Split the input text line using a comma as separator and store the result in the 'line' array.\*/

*{Melissa K.and the Heart of Gold Collectors Edition,PC,Adventure,SDP Games,9,10,2,0.18,0.95,4.45,1.13,6.66,13.19,1971}*

String[] line = value.toString().split(",");

/\*Concatenate the 2nd(genre) and 3rd element(console) of the 'line' array using a hyphen (-) as a separator by and store the result in the 'genreConsole' variable.\*/

String genreConsole = line[2] + "-" + line[1]; //Adventure-PC

// Parse the 12th element(global\_sales) of the 'line' array as a double and store the result in the 'sales' variable.

double sales = Double.parseDouble(line[12]); //13.19

/\*Use context that was defined as a parameter in the map method above to emit new key-value pair as output that is 'genreConsole' variable as key and 'sales' as value. \*/

context.write(new Text(genreConsole), new DoubleWritable(sales)); //Adventure-PC 13.19

}

}

**Create First Reducer Class: GenresConsoleReducer**

package org.myorg;

//Importing the libraries that contain the classes and methods needed in the Reducer class

import java.io.IOException;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapreduce.\*;

//Creating a subclass (child class) called GenresConsoleReducer from a parent class called Reducer using the keyword extends

public class GenresConsoleReducer extends Reducer<Text, DoubleWritable, Text, DoubleWritable> {

//<Input Key & Value datatype, Output Key & Value datatype>

/\*@override is used to change the original behavior of the method “reduce” in the parent Class “Reducer” \*/

@Override

// the reduce method takes three parameters that are a key, a value, and a context as input parameters

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

//initialize the variable totalSales with 0 value.

double totalSales = 0;

/\* Using for loop on input values, such that for each value in values, value is added to totalSales variable. \*/

for (DoubleWritable value: values) {

totalSales += value.get();

}

/\*Use context that was defined as a parameter in the map method above to emit new key-value pair to output using input key and 'totalSales' as the value.\*/

context.write(key, new DoubleWritable(totalSales));

}

}

**Create Second Mapper Class: TopNMapper**

package org.myorg;

//Importing the libraries that contain the classes and methods needed in the Mapper class

import java.io.IOException;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapreduce.\*;

/\*Creating a subclass (child class) called TopNMapper from a parent class called Mapper using the keyword extends \*/

public class TopNMapper extends Mapper<LongWritable, Text, Text, NullWritable> {

//<Input Key & Value datatype, Output Key & Value datatype>

/\*@override is used to change the original behavior of the method “map” in the parent Class “Mapper” \*/

@Override

// the map method takes three parameters that are a key, a value, and a context as input parameters

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

// Use context that was defined as a parameter in the map method above to emit the input value as key and NullWritable as value

context.write(value, NullWritable.get());

}

}

**Create Second Reducer Class: TopNReducer**

package org.myorg;

//Importing the libraries that contain the classes and methods needed in the Reducer class

import java.io.IOException;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapreduce.\*;

/\*Creating a subclass (child class) called TopNReducer from a parent class called Reducer using the keyword extends \*/

public class TopNReducer extends Reducer<Text, NullWritable, Text, DoubleWritable> {

/\*Initialize integer variable TOP with value 10 which is used to keep track of the number of top genre-console combinations to be output. \*/

private static final int TOP= 10;

/\*Initialize private instance variable top10 which is a TreeMap object that maps sales to their corresponding genre-console combinations.\*/

private java.util.TreeMap<Double, String> top10;

/\*@override is used to setup method from the parent class.\*/

@Override

/\*initialize the topN instance variable with a new TreeMap object. This method is called once at the start of the reduce task.\*/

public void setup(Context context) throws IOException, InterruptedException {

top10 = new java.util.TreeMap<>();

}

/\*@override is used to change the original behavior of the method “reduce” in the parent Class “Reducer” \*/

@Override

// the reduce method takes three parameters that are a key, a value, and a context as input parameters

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

/\*create a variable line of data type String, convert value into string and store it in line and split the input text line using a tab("\t") as separator and store the result in the 'line' array. \*/

String[] line = key.toString().split("\t");

//Store the key that is genre-console combination into genreConsole variable

String genreConsole = line[0];

//Store the value that is total sales variable totalSales.

double totalSales = Double.parseDouble(line[1]);

// Add the current genre-console combination to the TreeMap

top10.put(totalSales, genreConsole);

// Keep only the top 10 entries in the TreeMap

if (top10.size() > TOP) {

top10.remove(top10.firstKey());

}

}

//@Override is used for the cleanup method from the parent class.

@Override

// The cleanup method takes context object as parameter and return void.

public void cleanup(Context context) throws IOException, InterruptedException {

/\* The java.util.Set object stores Map.Entry objects with key of type Double and value of type String. The top 10 entries are retrieved from the TreeMap in descending order using the descendingMap() method. \*/

java.util.Set<java.util.Map.Entry<Double, String>> entries = top10.descendingMap().entrySet();

/\*For loop iterate on each entry, storing the key (total sales) to sales and value (genre-console) to variable gc\*/

for (java.util.Map.Entry<Double, String> entry : entries) {

double sales = entry.getKey();

String gc = entry.getValue();

// It returns the key-value pair, here key is genre-console combination(Text) and value is total sales(double).

context.write(new Text(gc), new DoubleWritable(sales));

}

}

}

**Main Driver Class: TopGenresConsoleSales**

package org.myorg;

//Importing the libraries that contain the classes and methods needed in the driver class

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapreduce.\*;

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

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

public class TopGenresConsoleSales {

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

if (args.length != 4) {

System.err.println("Usage: TopGenresConsoleSales <input path> <output path>");

System.exit(-1);

}

//create the “conf” object from the “Configuration” class, which provides access to configuration parameters necessary for Hadoop job.

Configuration conf = new Configuration();

// Create the first MapReduce job to get the total sales for each genre-console combination

Job job1 = Job.getInstance(conf, "GenresConsoleSales");

//specifying the driver class in the JAR file

job1.setJarByClass(TopGenresConsoleSales.class);

//Setting the mapper and reducer for the job

job1.setMapperClass(GenresConsoleMapper.class);

job1.setReducerClass(GenresConsoleReducer.class);

//Setting the data type for the output key class to be Hadoop’s data type Text

//Setting the data type for the output Value class to be Hadoop’s data type DoubleWritable

job1.setOutputKeyClass(Text.class);

job1.setOutputValueClass(DoubleWritable.class);

//Set the path of the input and output directories

FileInputFormat.addInputPath(job1, new Path(args[1]));

FileOutputFormat.setOutputPath(job1, new Path(args[2]));

//Submit the job to the cluster and wait for it to complete

job1.waitForCompletion(true);

// Create the second MapReduce job to get the top 10 genre-console combinations by total sales

Job job2 = Job.getInstance(conf, "TopGenresConsoleSales");

//specifying the driver class in the JAR file

job2.setJarByClass(TopGenresConsoleSales.class);

//Setting the mapper and reducer for the job

job2.setMapperClass(TopNMapper.class);

job2.setReducerClass(TopNReducer.class);

//Set the data type for the output key class to be Text and

//set the data type for the output Value class to be NullWritable

job2.setMapOutputKeyClass(Text.class);

job2.setMapOutputValueClass(NullWritable.class);

//Setting the data type for the output key class to be Hadoop’s data type Text and

//Setting the data type for the output Value class to be Hadoop’s data type DoubleWritable

job2.setOutputKeyClass(Text.class);

job2.setOutputValueClass(DoubleWritable.class);

//Set the path of the input and output directories

FileInputFormat.addInputPath(job2, new Path(args[2]));

FileOutputFormat.setOutputPath(job2, new Path(args[3]));

//Submit the job to the cluster and wait for it to complete

job2.waitForCompletion(true);

}

}

**Commands to Run This program.**

1. Format the filesystem (if you have already done first task, then no need for this):

**hadoop@hadoop-VirtualBox:~$** hdfs namenode –format

1. Start the HDFS and Yarn daemons:

**hadoop@hadoop-VirtualBox:~$** start-dfs.sh

**hadoop@hadoop-VirtualBox:~$** start-yarn.sh

**jps**

1. Make the HDFS directories required to execute MapReduce jobs:

**hadoop@hadoop-VirtualBox:~$** hdfs dfs -mkdir /input

1. Copy the input data files into the distributed filesystem:

**hadoop@hadoop-VirtualBox:~$** hdfs dfs -put Downloads/ Final\_dataset.txt /input

1. Delete if there exist any output files

**hdfs dfs -rm -r /user/Hadoop/output1**

**hdfs dfs -rm -r /user/Hadoop/output**

1. Now, execute the program on Hadoop using the following command:

**hadoop@hadoop-VirtualBox:~$** Hadoop jar Downloads/final.jar TopGenresConsoleSales /input/Final\_dataset.txt output output1

1. Then, copy the output files from the distributed filesystem to the local filesystem: **hadoop@hadoop-VirtualBox:~$** hdfs dfs -get output1 output1