Code:

**Logfile:**

**package Process;**

**import java.io.IOException;**

**import java.util.StringTokenizer;**

**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 Process {**

**public static class IPMapper**

**extends Mapper<Object, Text, Text, IntWritable>{**

**private final static IntWritable one = new IntWritable(1);**

**private Text ip = new Text();**

**public void map(Object key, Text value, Context context**

**) throws IOException, InterruptedException {**

**// Assuming the IP address is the first token in each line**

**StringTokenizer itr = new StringTokenizer(value.toString());**

**if (itr.hasMoreTokens()) {**

**ip.set(itr.nextToken());**

**context.write(ip, one);**

**}**

**}**

**}**

**public static class IntSumReducer**

**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);**

**}**

**}**

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

**Configuration conf = new Configuration();**

**Job job = Job.getInstance(conf, "IP address count");**

**job.setJarByClass(Process.class);**

**job.setMapperClass(IPMapper.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);**

**}**

**}**

**Unique listener:**

**1.UniqueListenersMapper.java**

**package mapreduce;**

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

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

**import java.io.IOException;**

**public class UniqueListenersMapper extends Mapper<LongWritable, Text, Text, Text> {**

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

**String[] fields = value.toString().split("\\|");**

**String trackId = fields[1];**

**String userId = fields[0];**

**context.write(new Text(trackId), new Text(userId));**

**}**

**}**

**2. UniqueListenersReducer.java**

**package mapreduce;**

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

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

**import java.io.IOException;**

**public class UniqueListenersReducer extends Reducer<Text, Text, Text, IntWritable> {**

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

**int uniqueListeners = 0;**

**for (Text value : values) {**

**uniqueListeners++;**

**}**

**context.write(key, new IntWritable(uniqueListeners));**

**}**

**}**

**3. class MusicAnalyticsDriver**

**package mapreduce;**

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

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

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

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

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

**import org.apache.hadoop.fs.Path;**

**public class MusicAnalyticsDriver {**

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

**if (args.length != 2) {**

**System.err.println("Usage: MusicAnalyticsDriver <inputPath> <outputPath1>");**

**System.exit(1);**

**}**

**Configuration conf = new Configuration();**

**Job job1 = Job.getInstance(conf, "Unique Listeners Count");**

**job1.setJarByClass(MusicAnalyticsDriver.class);**

**job1.setMapperClass(UniqueListenersMapper.class);**

**job1.setReducerClass(UniqueListenersReducer.class);**

**job1.setOutputKeyClass(Text.class);**

**job1.setOutputValueClass(Text.class);**

**FileInputFormat.addInputPath(job1, new Path(args[0]));**

**FileOutputFormat.setOutputPath(job1, new Path(args[1]));**

**job1.waitForCompletion(true);**

**Job job2 = Job.getInstance(conf, "Shares Count");**

**job2.setJarByClass(MusicAnalyticsDriver.class);**

**job2.setMapperClass(SharesMapper.class);**

**job2.setReducerClass(SharesReducer.class);**

**job2.setOutputKeyClass(Text.class);**

**job2.setOutputValueClass(IntWritable.class);**

**FileInputFormat.addInputPath(job2, new Path(args[0]));**

**FileOutputFormat.setOutputPath(job2, new Path(args[2]));**

**System.exit(job2.waitForCompletion(true) ? 0 : 1);**

**}**

**}**

**Sharecount:**

**package yes;**

**import java.io.IOException;**

**import org.apache.hadoop.fs.Path;**

**import org.apache.hadoop.conf.Configuration;**

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

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

**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 MusicTrackShareCount {**

**public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {**

**private Text trackId = new Text();**

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

**// Skip the header line**

**if (value.toString().equals("UserId|TrackId|Shared|Radio|Skip")) {**

**return;**

**}**

**String[] tokens = value.toString().split("\\|");if (tokens.length >= 3) {**

**try {**

**trackId.set(tokens[1]);**

**int shared = Integer.parseInt(tokens[2]);**

**context.write(trackId, new IntWritable(shared));**

**} catch (NumberFormatException e) {**

**System.err.println("Error parsing line: " + value.toString());**

**e.printStackTrace();**

**}**

**}else {**

**System.err.println("Invalid line format: " + value.toString());**

**}**

**}**

**}**

**public static class Reduce 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, "Music Track Share Count");**

**job.setJarByClass(MusicTrackShareCount.class);**

**job.setMapperClass(Map.class);**

**job.setCombinerClass(Reduce.class);**

**job.setReducerClass(Reduce.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);**

**}**

**}**

**TrackRadio:**

**package yes;**

**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.LongWritable;**

**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 TrackRadioListenerCount {**

**public static class RadioTrackMapper extends Mapper<LongWritable, Text, Text, IntWritable> {**

**private final static IntWritable one = new IntWritable(1);**

**private Text trackId = new Text();**

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

**String[] tokens = value.toString().split("\\|"); if (tokens.length >= 4 && tokens[3].equals("1")) { // Checking if the track was listened to on the radio**

**trackId.set(tokens[1]);**

**context.write(trackId, one);**

**}**

**}**

**}**

**public static class RadioTrackReducer 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);**

**}**

**}**

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

**Configuration conf = new Configuration();**

**Job job = Job.getInstance(conf, "Radio Track Counter");**

**job.setJarByClass(TrackRadioListenerCount.class);**

**job.setMapperClass(RadioTrackMapper.class);**

**job.setCombinerClass(RadioTrackReducer.class);**

**job.setReducerClass(RadioTrackReducer.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);**

**}**

**}**

**SkipTrack:**

**package yes;**

**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.LongWritable;**

**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 SkipTrackCounter {**

**public static class SkipTrackMapper extends Mapper<LongWritable, Text, Text, IntWritable> {**

**private final static IntWritable one = new IntWritable(1);**

**private Text trackId = new Text();**

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

**String[] tokens = value.toString().split("\\|");**

**if (tokens.length >= 5 && tokens[4].equals("1")) { // Checking if the track was skipped**

**trackId.set(tokens[1]);**

**context.write(trackId, one);**

**}**

**}**

**}**

**public static class SkipTrackReducer 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);**

**}**

**}**

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

**Configuration conf = new Configuration();**

**Job job = Job.getInstance(conf, "Skip Track Counter");**

**job.setJarByClass(SkipTrackCounter.class);**

**job.setMapperClass(SkipTrackMapper.class);**

**job.setCombinerClass(SkipTrackReducer.class);**

**job.setReducerClass(SkipTrackReducer.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);**

**}**

**}**

**Movie:**

**package yes;**

**import java.io.IOException;**

**import org.apache.hadoop.conf.Configuration;**

**import org.apache.hadoop.fs.Path;**

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

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

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

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

**public class MovieRecommendation {**

**public static class MovieMapper extends Mapper<LongWritable, Text, Text, DoubleWritable> {**

**@Override**

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

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

**if (parts.length == 2) {**

**String movieName = parts[0].trim();**

**double rating = Double.parseDouble(parts[1].trim());**

**context.write(new Text(movieName), new DoubleWritable(rating));**

**}**

**}**

**}**

**public static class MovieReducer extends Reducer<Text, DoubleWritable, Text, DoubleWritable> {**

**@Override**

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

**double sum = 0;**

**int count = 0;**

**for (DoubleWritable value : values) {**

**sum += value.get();**

**count++;**

**}double averageRating = sum / count;**

**context.write(key, new DoubleWritable(averageRating));**

**}**

**}**

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

**Configuration conf = new Configuration();**

**Job job = Job.getInstance(conf, "Movie Recommendation");**

**job.setJarByClass(MovieRecommendation.class);**

**job.setMapperClass(MovieMapper.class);**

**job.setReducerClass(MovieReducer.class);job.setOutputKeyClass(Text.class);**

**job.setOutputValueClass(DoubleWritable.class);**

**FileInputFormat.addInputPath(job, new Path(args[0]));**

**FileOutputFormat.setOutputPath(job, new Path(args[1]));**

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

**}**

**}**