Assignment 8 - Music Track Program

Problem Statement

XYZ.com is an online music website where users listen to various tracks, the data gets collected like shown below. Write a map reduce program to get following stats

- Number of unique listeners
- Number of times the track was shared with others
- Number of times the track was listened to on the radio
- Number of times the track was listened to in total
- Number of times the track was skipped on the radio

The data is coming in log files and looks like as shown below.

```
UserId |TrackId|Shared|Radio|Skip
111115|222      |0       |1       |0
111113|225      |1       |0       |0
111117|223      |0       |1       |1
111115|225      |1       |0       |0
```

Solution

First we are going to solve the first problem that is finding out unique listeners per track.

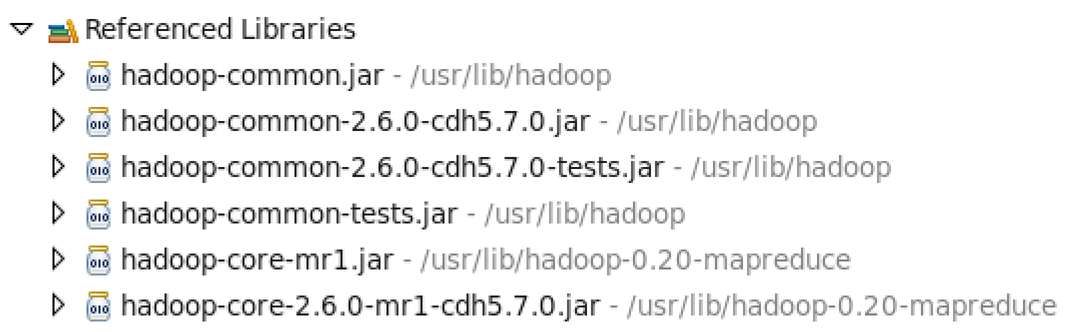
First of all we need to understand the data, here the first column is UserId and the second one is Track Id. So we need to write a mapper class which would emit trackId and userIds and intermediate key value pairs. so make it simple to remember the data sequence, let's create a constants class as shown below

```
package com.mr;
public class LastFMConstants {

   public static final int USER_ID = 0;
   public static final int TRACK_ID = 1;
   public static final int IS_SHARED = 2;
   public static final int RADIO = 3;
   public static final int IS_SKIPPED = 4;

}
```

Now, let’s create the mapper class which would emit intermediate key value pairs as (TrackId, UserId) as shown below

Step 1. Class Creation
Right click com package->new class-> give class name as "**UniqueListener**” and then
Click **Finish** button

Step 2. Add External Jars (Added Already)
Add JARS file:
Right click “**src**”->**build path->configure build path-> click Libraries pane->add external jars->file system->**



```
Referenced Libraries
    hadoop-common.jar - /usr/lib/hadoop
    hadoop-common-2.6.0-cdh5.7.0.jar - /usr/lib/hadoop
    hadoop-common-2.6.0-cdh5.7.0-tests.jar - /usr/lib/hadoop
    hadoop-common-tests.jar - /usr/lib/hadoop
    hadoop-core-mr1.jar - /usr/lib/hadoop-0.20-mapreduce
    hadoop-core-2.6.0-mr1-cdh5.7.0.jar - /usr/lib/hadoop-0.20-mapreduce
```

Step 3. Type the following MapReduce Program "UniqueListener"

```
public static class UniqueListenersMapper extends Mapper< Object , Text, IntWritable, IntWritable > {
    IntWritable trackId = new IntWritable();
    IntWritable userId = new IntWritable();

public void map(Object key, Text value, Mapper< Object , Text, IntWritable, IntWritable > .Context context)
    throws IOException, InterruptedException {

  String[] parts = value.toString().split("[|]");
  trackId.set(Integer.parseInt(parts[LastFMConstants.TRACK_ID]));
  userId.set(Integer.parseInt(parts[LastFMConstants.USER_ID]));
    if (parts.length == 5) {
    context.write(trackId, userId);
  } else {
    // add counter for invalid records
    context.getCounter(COUNTERS.INVALID_RECORD_COUNT).increment(1L);
  }
  }
}
```

You would have also noticed that we are using a counter here named INVALID_RECORD_COUNT , to count if there are any invalid records which are not coming the expected format. Remember, if we don't do this then in case of invalid records, our program might fail.

Now let's write a Reducer class to aggregate the results. Here we simply can not use sum reducer as the records we are getting are not unique and we have to count only unique users. Here is how the code would look like

```
public static class UniqueListenersReducer extends Reducer< IntWritable , IntWritable, IntWritable,
IntWritable> {

public void reduce(IntWritable trackId,Iterable< IntWritable > userIds,Reducer< IntWritable , IntWritable,
IntWritable,IntWritable>.Context context)  throws IOException, InterruptedException {

    Set< Integer > userIdSet = new HashSet< Integer >();
    for (IntWritable userId : userIds) {
    userIdSet.add(userId.get());
    }

    IntWritable size = new IntWritable(userIdSet.size());
    context.write(trackId, size);
  }
}
```

Here we are using Set to eliminate duplicate userIds. Now we can take look at the Driver class

```
public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        if (args.length != 2) {
            System.err.println("Usage: uniquelisteners < in >< out >");
            System.exit(2);
        }
        Job job = new Job(conf, "Unique listeners per track");
        job.setJarByClass(UniqueListeners.class);
        job.setMapperClass(UniqueListenersMapper.class);
        job.setReducerClass(UniqueListenersReducer.class);
        job.setOutputKeyClass(IntWritable.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);
        org.apache.hadoop.mapreduce.Counters counters = job.getCounters();
        System.out.println("No. of Invalid Records :"
                + counters.findCounter(COUNTERS.INVALID_RECORD_COUNT).getValue());
    }
```

Step 4. Export JAR file creation:
Right click src->Export->Java->JAR File->click Next button

Step Music Track Execution:

```
[cloudera@quickstart ~]$ hadoop fs -put /home/cloudera/Desktop/UniqueListener.txt /user/cloudera/sriMR

[cloudera@quickstart ~]$ hadoop jar
/home/cloudera/Desktop/srihadoop/MR/UniqueListener/UniqueListener.jar com.mr.
UniqueListener/user/cloudera/sriMR/UniqueListener.txt /user/cloudera/sriMR/UniqueListenerresult
```

Assignment 1 -Wordcount Program

Apply your MapReduce programming knowledge and write a MapReduce program to process a text file. You need to print the count of number of occurrences of each word in the text file.
The dataset for this problem is the text file **‘wordcount’** available in your Lab

Problem statement
Let’s understand the problem through a sample text file content:
“Hello everyone this is a sample dataset. You need to print the word count of particular words in this dataset.”
Your MapReduce program should process this text file and should provide output as follows:
Output

Word	**Word Count**
a	**1 (As the word ‘a’ occurred only once)**
this	**2 (As the word ‘this’ occurred twice)**

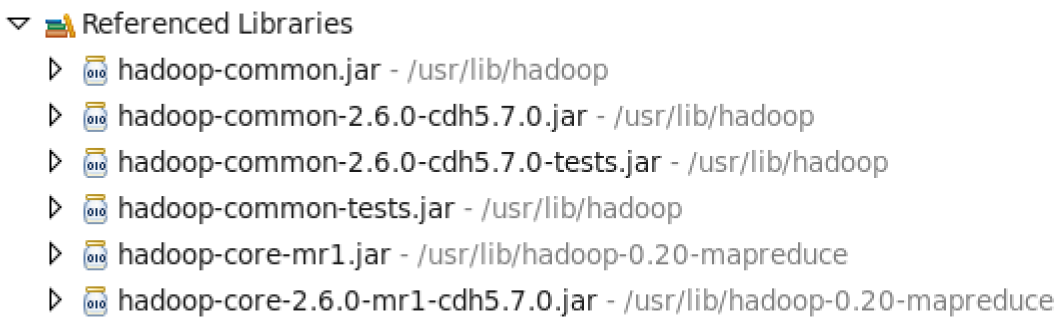
Solution

Step 1. Project Creation
File->New->Java Project->project name: “**MR”** and then
Click **Finish** button

Step 2. Package Creation
Expand the project click **“src”**->right click->new package->give package name as "com.mr" and then
Click **Finish** button

Step 3. Class Creation
Right click com package->new class-> give class name as "**WordCount**” and then
Click **Finish** button

Step 4. Add External Jars
Add JARS file:
Right click “**src**”->**build path->configure build path-> click Libraries pane->add external jars->file system->**


```
Referenced Libraries
    hadoop-common.jar - /usr/lib/hadoop
    hadoop-common-2.6.0-cdh5.7.0.jar - /usr/lib/hadoop
    hadoop-common-2.6.0-cdh5.7.0-tests.jar - /usr/lib/hadoop
    hadoop-common-tests.jar - /usr/lib/hadoop
    hadoop-core-mr1.jar - /usr/lib/hadoop-0.20-mapreduce
    hadoop-core-2.6.0-mr1-cdh5.7.0.jar - /usr/lib/hadoop-0.20-mapreduce
```

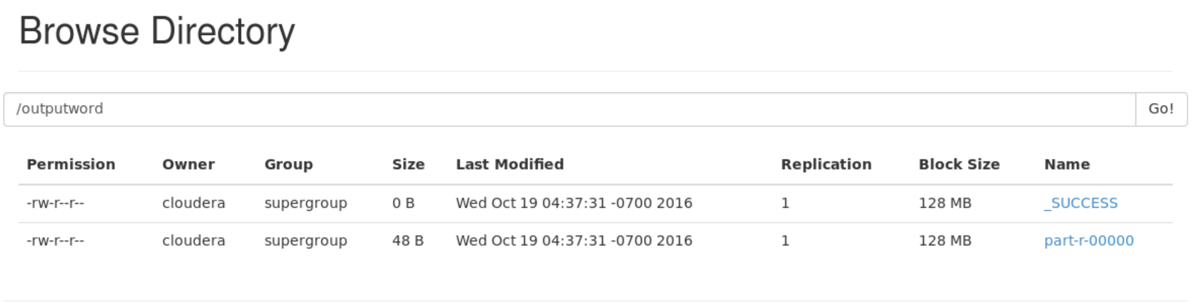
Step 5. Type the following MapReduce Program “WordCount”

```
package com.mr;
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.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.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordCount {
public static class Map extends Mapper<LongWritable,Text,Text,IntWritable>
        {
                private final static IntWritable one=new IntWritable(1);
                private Text word=new Text();

                public void map(LongWritable key,Text value,Context context) throws
IOException,InterruptedException
                {
                        String line=value.toString();
                        StringTokenizer tokenizer=new StringTokenizer(line);
                        while(tokenizer.hasMoreTokens())
                        {
                                word.set(tokenizer.nextToken());
                                context.write(word,one);
                        }
                }
        }

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=new Job(conf,"WordCount");
        job.setJarByClass(WordCount.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        job.setMapperClass(Map.class);
        job.setCombinerClass(Reduce.class);
        job.setReducerClass(Reduce.class);

        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);

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

        }

        }
```

Step 6. Export JAR file creation:
Right click src->Export->Java->JAR File->click Next button

Step 7. WordCount Execution:

```
[cloudera@quickstart ~]$ Hadoop fs –mkdir /sriMR

[cloudera@quickstart ~]$ hadoop fs -put /home/cloudera/Desktop/srihadoop/MR/WordCount/WordCount.txt
/user/cloudera/sriMR
[cloudera@quickstart ~]$ hadoop jar /home/cloudera/Desktop/srihadoop/MR/WordCount/WordCount.jar
com.mr.WordCount /user/cloudera/sriMR/inputword.txt /user/cloudera/sriMR/WordCountresult
```


Browse Directory

/outputword

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	cloudera	supergroup	0 B	Wed Oct 19 04:37:31 -0700 2016	1	128 MB	_SUCCESS
-rw-r--r--	cloudera	supergroup	48 B	Wed Oct 19 04:37:31 -0700 2016	1	128 MB	part-r-00000

Input file:

inputword.txt

```
hello welcome
welcome to big data
data is good
```

Debugging

Ctrl+Shift+O

Org.apache.hadoop.io.Text
Org.apache.hadoop.mapreduce.Job
Org.apache.hadoop.mapreduce.lib.output.FileOutputFormat
Org.apache.hadoop.mapreduce.lib.input.FileInputFormat
Org.apache.hadoop.mapreduce.lib.output.TextOutputFormat
Org.apache.hadoop.mapreduce.lib.input.TextInputFormat

Assignment 2 -WordSizeWordCount Program

Apply your MapReduce programming knowledge and write a MapReduce program to process two text files.
You need to calculate the size of each word and count the number of words of that size in the text file.
The dataset for this problem is the text file **'alphabets'** available in your LMS.

Problem statement

Let's understand the problem through a sample text file content:

"Hello everyone this is a sample dataset. Calculate the word size and count the number of words of that size in this text file."

Your MapReduce program should process this text file and should provide output as follows:

Sample Output

Word Size	**Word Count**
1	**1 (As the word of size 1 is: a)**
2	**4 (As the words of size 2 are: is, of, of, in)**
3	**3 (As the words of size 3 are: the, and, the)**
4	**6 (As the words of size 4 are: this, word, size, that, size)**

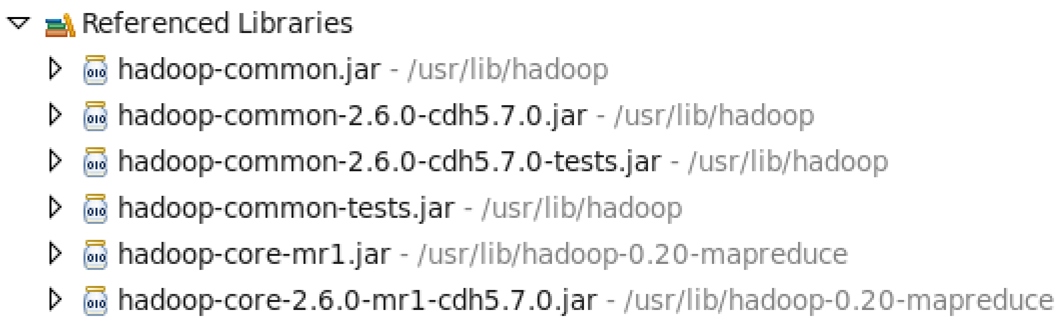
Solution

Step 1. Class Creation

Right click com package->new class-> give class name as "**WordSizeWordCount**" and then
Click **Finish** button

Step 2. Add External Jars (Added Already)

Add JARS file:
Right click "**src**"->**build path->configure build path-> click Libraries pane->add external jars->file system->**



```
Referenced Libraries
    hadoop-common.jar - /usr/lib/hadoop
    hadoop-common-2.6.0-cdh5.7.0.jar - /usr/lib/hadoop
    hadoop-common-2.6.0-cdh5.7.0-tests.jar - /usr/lib/hadoop
    hadoop-common-tests.jar - /usr/lib/hadoop
    hadoop-core-mr1.jar - /usr/lib/hadoop-0.20-mapreduce
    hadoop-core-2.6.0-mr1-cdh5.7.0.jar - /usr/lib/hadoop-0.20-mapreduce
```

Step 3. Type the following MapReduce Program "WordSizeWordCount"

```
package com.mr;
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.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
```

```java
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.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordSizeWordCount {

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

   //Defining a local variable count of type IntWritable
           private static IntWritable count ;

   //Defining a local variable word of type Text
       private Text word = new Text();


//Mapper

           /**
            * @method map
            * <p>This method takes the input as text data type and splits the input into words.
            * Now the length of each word in the input is determined and key value pair is made.
            * This key value pair is passed to reducer.
            * @method_arguments key, value, output, reporter
            * @return void
            */

       /*
       * (non-Javadoc)
       * @see org.apache.hadoop.mapred.Mapper#map(java.lang.Object, java.lang.Object,
org.apache.hadoop.mapred.OutputCollector, org.apache.hadoop.mapred.Reporter)
       */

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

           //Converting the record (single line) to String and storing it in a String variable line
         String line = value.toString();

         //StringTokenizer is breaking the record (line) into words
         StringTokenizer tokenizer = new StringTokenizer(line);

         //iterating through all the words available in that line and forming the key value pair
         while (tokenizer.hasMoreTokens()) {

           String thisH = tokenizer.nextToken();

           //finding the length of each token(word)
           count= new IntWritable(thisH.length());
           word.set(thisH);

           //Sending to output collector which in turn passes the same to reducer
           //So in this case the output from mapper will be the length of a word and that word
           context.write(count,word);
```

```
        }
      }
    }
```

//Reducer

```
            public static class Reduce extends Reducer<IntWritable, Text, IntWritable, IntWritable> {

            /**
             * @method reduce
             * <p>This method takes the input as key and list of values pair from mapper, it does aggregation
             * based on keys and produces the final output.
             * @method_arguments key, values, output, reporter
             * @return void
             */

                    /*
                    * (non-Javadoc)
                    * @see org.apache.hadoop.mapred.Reducer#reduce(java.lang.Object, java.util.Iterator,
org.apache.hadoop.mapred.OutputCollector, org.apache.hadoop.mapred.Reporter)
                    */

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

            //Defining a local variable sum of type int
          int sum = 0;

          /*
           * Iterates through all the values available with a key and add them together and give the final
           * result as the key and sum of its values.
           */

          for(Text x : values)
          {
            sum++;
          }

          //Dumping the output
          context.write(key, new IntWritable(sum));
        }

    }
```

//Driver

```
    /**
     * @method main
     * <p>This method is used for setting all the configuration properties.
     * It acts as a driver for map reduce code.
     * @return void
     * @method_arguments args
     * @throws Exception
     */

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

            //reads the default configuration of cluster from the configuration xml files

            Configuration conf = new Configuration();

            //Initializing the job with the default configuration of the cluster

                    Job = new Job(conf, "Wordsize");

                    //Assigning the driver class name

                    job.setJarByClass(WordSizeWordCount.class);

                    //Defining the mapper class name

                    job.setMapperClass(Map.class);

                    //Defining the reducer class name

                    job.setReducerClass(Reduce.class);

                    //Defining the output key class for the mapper

                    job.setMapOutputKeyClass(IntWritable.class);

                    //Defining the output value class for the mapper

                    job.setMapOutputValueClass(Text.class);

                    //Defining the output key class for the final output i.e. from reducer

                    job.setOutputKeyClass(IntWritable.class);

                    //Defining the output value class for the final output i.e. from reduce

                    job.setOutputValueClass(IntWritable.class);

                    //Defining input Format class which is responsible to parse the dataset into a key value pair

                    job.setInputFormatClass(TextInputFormat.class);

                    //Defining output Format class which is responsible to parse the final key-value output from
MR framework to a text file into the hard disk

                    job.setOutputFormatClass(TextOutputFormat.class);

                     //setting the second argument as a path in a path variable

                    Path outputPath = new Path(args[1]);

                     //Configuring the input/output path from the filesystem into the job

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

                    //deleting the output path automatically from hdfs so that we don't have delete it explicitly

                     outputPath.getFileSystem(conf).delete(outputPath);

                     //exiting the job only if the flag value becomes false

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

}
```

Step 4. Export JAR file creation:
Right click src->Export->Java->JAR File->click Next button

Step 5 WordSizeWordCount Execution:

```
[cloudera@quickstart ~]$ hadoop fs -put
/home/cloudera/Desktop/srihadoop/MR/WordSizeWordCount/WordSizeWordCount.txt /user/cloudera/sriMR

[cloudera@quickstart ~]$ hadoop jar
/home/cloudera/Desktop/srihadoop/MR/WordSizeWordCount/WordSizeWordCount.jar
com.mr.WordSizeWordCount /user/cloudera/sriMR/WordSizeWordCount.txt
/user/cloudera/sriMR/WordSizeWordCountresult
```