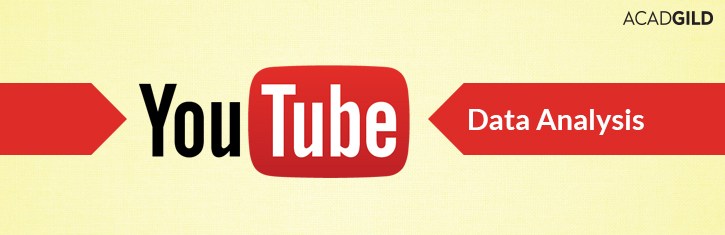
**MapReduce Use Case: YouTube Data Analysis**



**YouTube Data Analysis**

This blog is about, how to perform YouTube data analysis in [Hadoop](https://acadgild.com/big-data/big-data-development-training-certification) MapReduce.  
This YouTube data is publicly available and the YouTube data set is described below under the heading Data Set Description.  
Using that dataset we will perform some Analysis and will draw out some insights like what are the top 10 rated videos on YouTube, who uploaded the most number of videos.  
By reading this blog you will understand how to handle data sets that do not have proper structure and how to sort the output of reducer.

**DATA SET DESCRIPTION**

**Column 1:** Video id of 11 characters.  
**Column 2:** uploader of the video  
  
**Column 3:** Interval between the day of establishment of Youtube and the date of uploading of the video.  
  
**Column 4:** Category of the video.  
  
**Column 5:** Length of the video.  
  
**Column 6:** Number of views for the video.  
  
**Column 7:** Rating on the video.  
  
**Column 8:** Number of ratings given for the video  
**Column 9:** Number of comments done on the videos.  
  
**Column 10:** Related video ids with the uploaded video.  
You can download the data set from the below link.

[Get Skilled in Big Data](https://acadgild.com/big-data/big-data-development-training-certification)

**DATA SET LINK**

[Youtube Data set](https://drive.google.com/open?id=0ByJLBTmJojjzR2x0MzVpc2Z6enM)

**PROBLEM STATEMENT 1**

Here we will find out what are the top 5 categories with maximum number of videos uploaded.

**SOURCE CODE**

Now from the mapper, we want to get the *video category as key* and final int value *‘1’ as values* which will be passed to the *shuffle* and *sort* phase and are further sent to the reducer phase where the aggregation of the values is performed.

**MAPPER CODE**

public class Top5\_categories {

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

private Text category = new Text();

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

public void map(LongWritable key, Text value, Context context )

throws IOException, InterruptedException {

String line = value.toString();

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

if(str.length > 5){

category.set(str[3]);

}

context.write(category, one);

}

}

Explanation of the above Mapper code:  
In **line 1** we are taking a class by name Top5\_categories,  
In **line 2** we are extending the Mapper default class having the arguments *keyIn* as *LongWritable* and *ValueIn* as *Text* and *KeyOut* as *Text* and *ValueOut* as *IntWritable.*

In **line 3** we are declaring a private Text variable *‘category’* which will store the category of videos in youtube

In **line 4** we are declaring a private final static *IntWritable* variable *‘one’* which will be constant for every value. MapReduce deals with Key and Value pairs.Here we can set the *key as gender* and *value as age*.

In **line 5** we are overriding the map method which will run one time for every line.

In **line 7** we are storing the line in a string variable *‘line’*

In **line 8** we are splitting the line by using tab “\t” delimiter and storing the values in a String Array so that all the columns in a row are stored in the string array.

In **line 9** we are taking a condition if we have the string array of length greater than 6 which means if the line or row has at least 7 columns then it will enter into the if condition and execute the code to eliminate the ***ArrayIndexOutOfBoundsException.***

In **line 10** we are storing the category which is in the 4th column.

In **line 12** we are writing the key and value into the context which will be the output of the map method.

**REDUCER CODE**

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));

}

}

**While coming to the Reducer code**

**line 1** extends the default Reducer class with arguments *KeyIn* as *Text* and *ValueIn* as *IntWritable* which are same as the outputs of the mapper class and *KeyOut* as *Text* and *ValueOut* as *IntWritbale* which will be final outputs of our MapReduce program.

In **line 2** we are overriding the Reduce method which will run each time for every key.

In **line 3** we are declaring an integer variable *sum* which will store the sum of all the values for each key.

In **line 4** a for each loop is taken which will run each time for the values inside the *“Iterable values”* which are coming from the *shuffle* and *sort* phase after the mapper phase.

In **line 5** we are storing and calculating the sum of the values.

In **line 7** writes the respected key and the obtained sum as value to the context.

**CONF CODE**

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(IntWritable.class);

This two configuration classes are included in the main class whereas to clarify the Output key type of mapper and the output value type of the Mapper.  
You can download the whole source code from the below link.

**SOURCE CODE LINK**

[GitHub link for Problem statement 1](https://github.com/kiran0541/Map-Reduce/blob/master/top5%20categories%20of%20videos%20in%20youtube%20data)

**HOW TO EXECUTE**

**hadoop jar top5.jar /youtubedata.txt /top5\_out**

Here *‘hadoop’* specifies we are running a Hadoop command and *jar* specifies which type of application we are running and *top5.jar* is the jar file which we have created consisting of the above source code.

The path of the Input file in our case is root directory of hdfs denoted by */youtubedata.txt*  and the output file location to store the output has been given as *top5\_out.*

**How to view output**

**hadoop fs -cat /top5\_out/part-r-00000 | sort –n –k2 –r | head  –n5**  
Here *‘hadoop’* specifies that we are running a Hadoop command and *dfs* specifies that we are performing an operation related to Hadoop Distributed File System and *‘- cat’* is used to view the contents of a file and *top5\_out/part-r-00000* is the file where output is stored.  
Part file containing the actual output is created by default by the *TextInputFormat* class of Hadoop.  
Here ***sort –n –k2 –r | head –n5*** brings you the top 5 categories with maximum number of videos uploaded.  
Instead of writing a *secondary sort* after reducer we can simply use this command to get the required output.  
*Sort* will sort the data, *–n* means sorting numerically, *–k2* means second column,  *–r* is for recursive operation and *head –n5* means to bring the first 5 values after sorting.

**Output**

[acadgild@localhost -]hadoop jar top5.jar /youtubedata.txt /top5\_out

15/10/22 11:06:45 WARN util.NativeCodeLoader: Unable to load native-hadoop libra ry for your platform... using builtin-java classes where applicable

15/10/22 11:06:48 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0 :8032

15/10/22 11:06:49 WARN mapreduce.JobSubmitter: Hadoop command-line option parsin g not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.

15/10/22 11:06:50 INFO input.FileInputFormat: Total input paths to process : 1

15/10/22 11:06:50 INFO mapreduce.JobSubmitter: number of splits:1

15/10/22 11:06:51 INFO mapreduce.JobSubmitter: Submitting tokens for job: job\_14 45504384269 0002

15/10/22 11:06:51 INFO impl.YarnClientlmpl: Submitted application application 14 45504384269\_0002

15/10/22 11:06:52 INFO mapreduce.Job: The url to track the job: http://localhost .localdomain:8088/proxy/application 1445504384269\_0002/

15/10/22 11:06:52 INFO mapreduce.Job: Running job: job 1445504384269\_0002

15/10/22 11:07:05 INFO mapreduce.Job: Job job\_1445504384269\_0002 running in uber mode : false

15/10/22 11:07:05 INFO mapreduce.Job: map 0% reduce 0%

15/10/22 11:07:15 INFO mapreduce.Job: map 100% reduce 0%

15/10/22 11:07:27 INFO mapreduce.Job: map 100% reduce 100%

[acadgild@localhost —]$ hadoop fs -cat /top5\_out/part-r-88000 1 sort -n -k2 -r 1 head -n5

15/18/22 13:22:06 WARN util.NativeCodeLoader: Unable to load native-hadoop Libra ry for your platform... using builtin-java classes where applicable

Entertainment 911

Music 870

Comedy 420

Sports 253

Education 65

**PROBLEM STATEMENT 2**

In this problem statement, we will find the top 10 rated videos on youtube.

**SOURCE CODE**

Now from the mapper, we want to get the *video id* as *key* and *rating* as *a value* which will be passed to the *shuffle* and *sort* phase and is further sent to the reducer phase where the aggregation of the values is performed.

**MAPPER CODE**

1.    public class Video\_rating {

2.     public static class Map extends Mapper<LongWritable, Text, Text,

3. FloatWritable> {

4.        private Text video\_name = new Text();

5.        private  FloatWritable rating = new FloatWritable();

6.        public void map(LongWritable key, Text value, Context context )

7. throws IOException, InterruptedException {

8.            String line = value.toString();

9. If(line.length()>0) {

10.            String str[]=line.split("\t");

11.                 video\_name.set(str[0]);

12.                 if(str[6].matches("\\d+.+")){

13.                 float f=Float.parseFloat(str[6]);

14.                 rating.set(f);

15. }

16. }

17.       context.write(video\_name, rating);

18. }

19. }

20. }

Explanation of the above Mapper code  
In **line 1** we are taking a class by name Video\_rating  
In **line 2** we are extending the Mapper default class having the arguments *keyIn* as *LongWritable* and *ValueIn* as *Text* and *KeyOut* as Text and *ValueOut* as *FloatWritable.*  
In **line 4** we are declaring a private Text variable ‘*video\_name’* which will store the video name which is in an encrypted format.  
In **line 5** we are declaring a private *FloatWritable* variable *‘rating’* which will store the rating of the video. MapReduce deals with *Key* and *Value* pairs.Here we can set the *key* as *gender* and *value* as *age*.  
In **line 6** we are overriding the map method which will run one time for every line.  
In **line 8** we are storing the line in a string variable *‘line’*  
In **line 9** we are taking a condition if we have the string array length greater than 7 which means if the line or row has at least 7 columns then it will enter into the if condition and execute the code to eliminate the ***ArrayIndexOutOfBoundsException.***  
In **line 10** we are splitting the line by using tab *“\t”* delimiter and storing the values in a *String Array* so that all the columns in a row are stored in the string array.  
In **line 11** we are storing the *video name* which is in the 1st column.  
In **line 12** we are checking whether the data in that index is numeric data or not by using a regular expression which can be achieved by “matches function in java”, if it is numeric data then it will proceed and it should be a floating value as well.  
In **line 13**we are converting that numeric data into Float data by type casting.  
In **line 14** we are storing the rating of the video in *‘rating’* variable.  
In **line 17** we are writing the key and value into the *context* which will be the output of the map method.

public static class Reduce extends Reducer<Text, FloatWritable,Text, FloatWritable> {

public void reduce(Text key, Iterable<FloatWritable> values,Context context)

throws IOException, InterruptedException {

float sum = 0;

Int l=0;

for (FloatWritable val : values) {

l+=1;

sum += val.get();

}

sum=sum/l;

context.write(key, new FloatWritable(sum));

}

}

**REDUCER CODE**

While coming to the Reducer code

**line 1** extends the default Reducer class with arguments *KeyIn* as *Text* and *ValueIn* as *IntWritable* which are same as the outputs of the mapper class and *KeyOut* as *Text* and *ValueOut* as *IntWritbale* which will be final outputs of our MapReduce program.

In **line 2** we are overriding the Reduce method which will run each time for every key.

In **line 4** we are declaring an integer sum which will store the sum of all the ages of people in it.

In **line 5** we are taking another variable as *“l”* which will be incremented every time as many values are there for that key.

In **line 6** a for each loop is taken which will run each time for the values inside the *“Iterable values”* which are coming from the *shuffle* and *sort* phase after the mapper phase.

In **line 8** we are storing and calculating the sum of the values.

In **line 10**we are performing the average of the obtained sum and writes the respected key and the obtained sum as *value* to the context.

**CONF CODE**

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(FloatWritable.class);

This two configuration classes are included in the main class whereas to clarify the Output key type of mapper and the output value type of the Mapper.  
You can download the whole source code from the below link

**SOURCE CODE LINK**

[GitHub link for Problem statement 2](https://github.com/kiran0541/Map-Reduce/blob/master/Finding%20rating%20of%20each%20video%20in%20youtube%20data)

**HOW TO EXECUTE**

**hadoop jar video\_rating.jar /youtubedata.txt /videorating\_out**

The explanation for the above command will be as same as given in problem statement 1.

**How to view output**

**hadoop fs -cat /videorating\_out/part-r-00000 | sort –n –k2 –r | head –n10**  
Explanation for the above command will be as same as given in problem statement 1.

**Output:**

[acadgild@localhost -]$ hadoop jar video\_rating.jar /youtubedata.txt /videoratin g\_out

15/10/22 12:52:55 WARN util.NativeCodeLoader: Unable to load native-hadoop libra ry for your platform... using builtin-java classes where applicable

15/10/22 12:52:59 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0 :8032

15/10/22 12:53:00 WARN mapreduce.JobSubmitter: Hadoop command-line option parsin g not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this. 15/10/22 12:53:01 INFO input.FilelnputFormat: Total input paths to process : 1

15/10/22 12:53:02 INFO mapreduce.JobSubmitter: number of splits:1

15/10/22 12:53:02 INFO mapreduce.JobSubmitter: Submitting tokens for job: job\_14 45504384269 0006

15/10/22 12:53:03 INFO impl.YarnClientlmpl: Submitted application application 14 45504384269\_0006 15/10/22 12:53:03 INFO mapreduce.Job: The url to track the job: http://localhost .localdomain:8088/proxy/application 1445504384269\_0006/

15/10/22 12:53:03 INFO mapreduce.Job: Running job: job 1445504384269\_0006

15/10/22 12:53:21 INFO mapreduce.Job: Job job\_1445504384269\_0006 running in uber mode : false

15/10/22 12:53:21 INFO mapreduce.Job: map 0% reduce 0%

15/10/22 12:53:36 INFO mapreduce.Job: map 100% reduce 0%

15/10/22 12:53:46 INFO mapreduce.Job: map 100% reduce 100%

15/10/22 12:53:46 INFO mapreduce.Job: Job job\_1445504384269\_0006 completed successfully

[acadgild@localhost -]$ hadoop fs -cat /videorating\_out/part-r-00000 1 sort -n -k2 -r 1 head -n10 15/10/22 12:54:28 WARN util.NativeCodeLoader: Unable to load native-hadoop libra ry for your platform... using builtin-java classes where applicable

r30-2Q3V1jc 4.99

KOweSiiviVO 4.99

jIuCA4RRtXE 4.99

h\_8gsd8IT7Y 4.99

cYbVkXai6Ec 4.99

aoDBacpCX34 4.99

3v1oRJYR6A 4.99

xe-f-zg\_KIU 4.98

U4yJB1ynN-Y 4.98

sWIOyZnnChk 4.98

We hope this blog will help you to get a grip on MapReduce programming. Refer the below blog to understand the analysis done on Titanic data set.