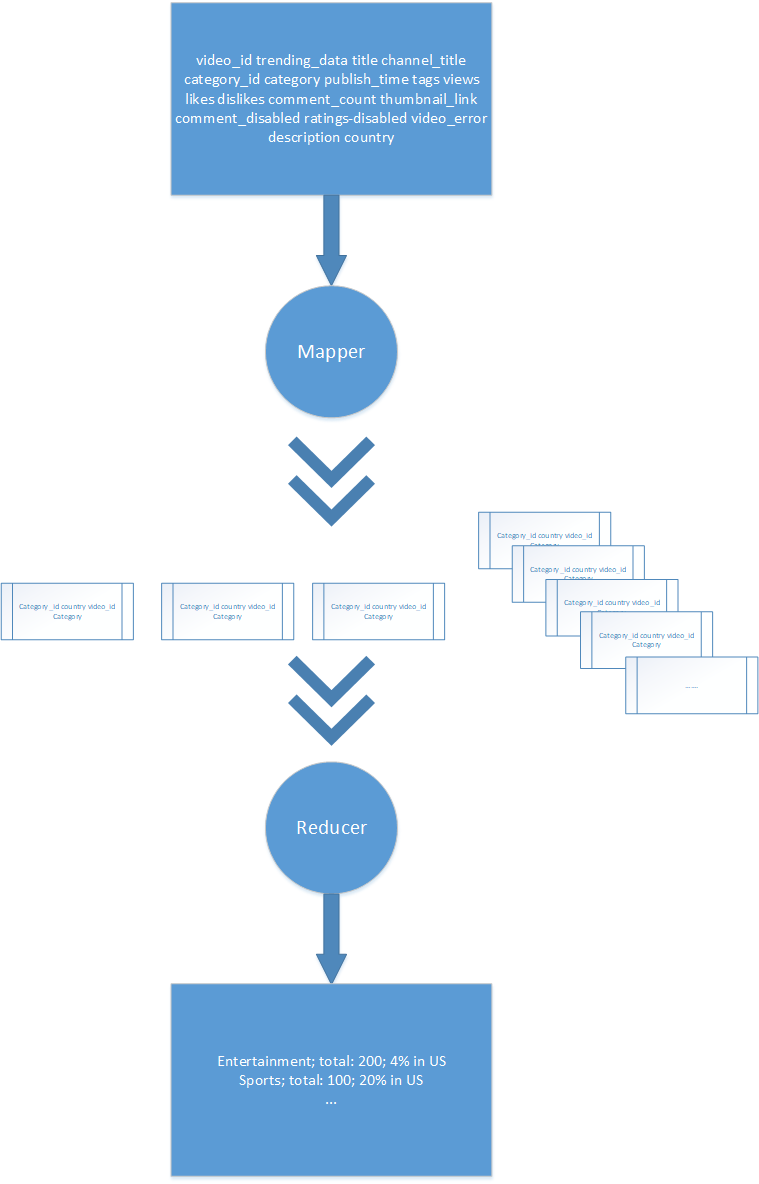
|  |  |  |
| --- | --- | --- |
| Workload | Implementation | Programming Language |
| Category and Trending Correlation | MapReduce | Python |
| Impact of Trending on View number | Spark | Python |

# Workload: Category and Trending Correlation

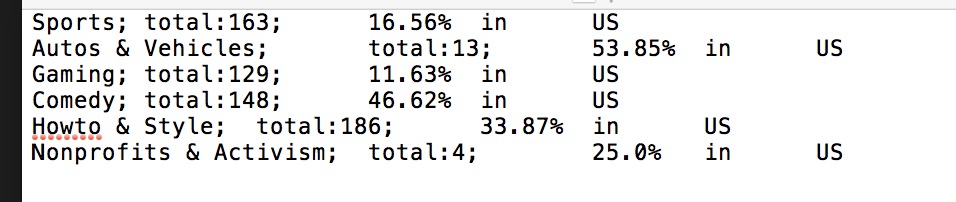
This is task is done with one MapReduce Job and it is consists of only two phases: Map then reduce, which is illustrated in Figure 1.



*Figure 1: MapReduce phase for the workload*

For process described in the Figure 1, the mapper breaks the row into 18 segments and then just assigns the *country, category\_id* and *category\_name* tags for each *video\_id* tag. To avoid splitting the information inside the double quotes, there are two methods can be used. One is reading the file with the *csv.reader*() and the other is implement the following code: *re.split(r',(?=(?:[^\"]\*\"[^\"]\*\")\*[^\"]\*$)', record)* . In my assignment, the *csv.reader()* is chosen. Hence, each row yield the key, which is *(category\_id, category\_name, country, video\_id*).

The reducer receives the output from the mapper sorted by the *category\_id* (the key), and calculate the category and trending correlation between these two countries. For all the same category, two lists are generated to store those video\_id for each country separately, while the different category is detected, the program calculate the intersection of the two lists and output the percentage for the previous category. The partial results are shown in the Figure 2.



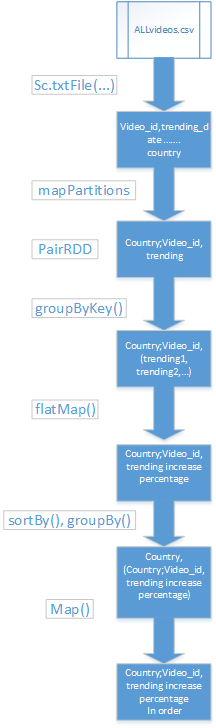
*Figure 2: Partial result of the category and trending correlation*

## Parallelization

Both mapper and reducer phases can run in parallel. Mappers run in parallel on different partition of the input dat. Three reducers are set and run in parallel on different partition of the intermediate results.

# Workload: Impact of Trending on View Number

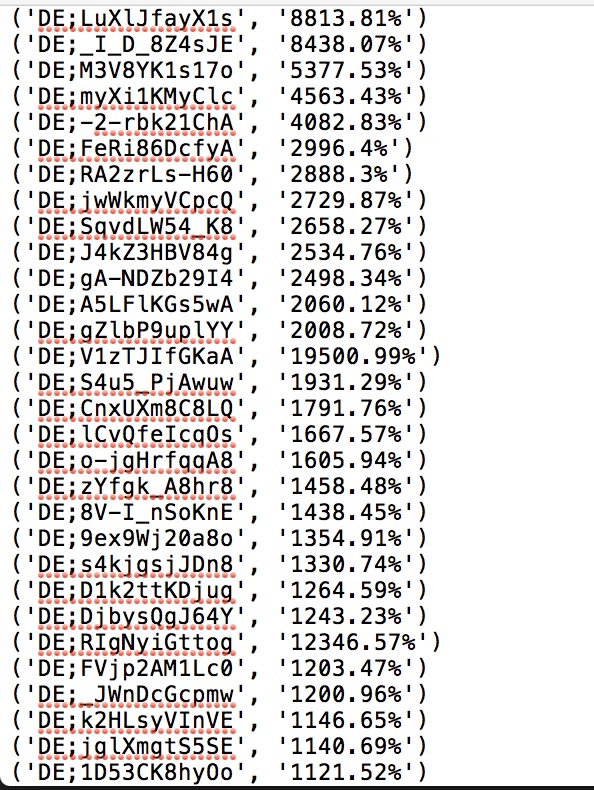
The sequence of transformations and cations are illustrated in the Figure 3.

****

*Figure 3: Spark phase for the workload*

The input file is read in and map to create RDD pair *(country;video\_id, trending)*. In order to implement the csv reader, the *mapPartitions()* is being used as it accepts an iterable object.

Then the *groupByKey()* is applied to group all the same *video\_id* with the same *country* tag and the viewing numbers are in a list. After this step, a *flatMap()* involved a function called *extratViews* is used to calculate the increase in viewing numbers between the first and the second trending appearance. Finally, a *sortBy()* and a *groupBy()* function are applied to sort the results, which will be grouped by country and displayed in descending order. The partial results are shown in the Figure 4.



*Figure 4: Partial results of impact of trending on view number*

## Parallelization

The input file is read in then parallelize using the built-in Spark Content parallelize. The *mapPartitons()* operations can run in parallel on different partitions of the videos file. The *`* can run in parallel and pipeline with the next map operation. The Sort process can run in parallel and pipeline with the next map operation.