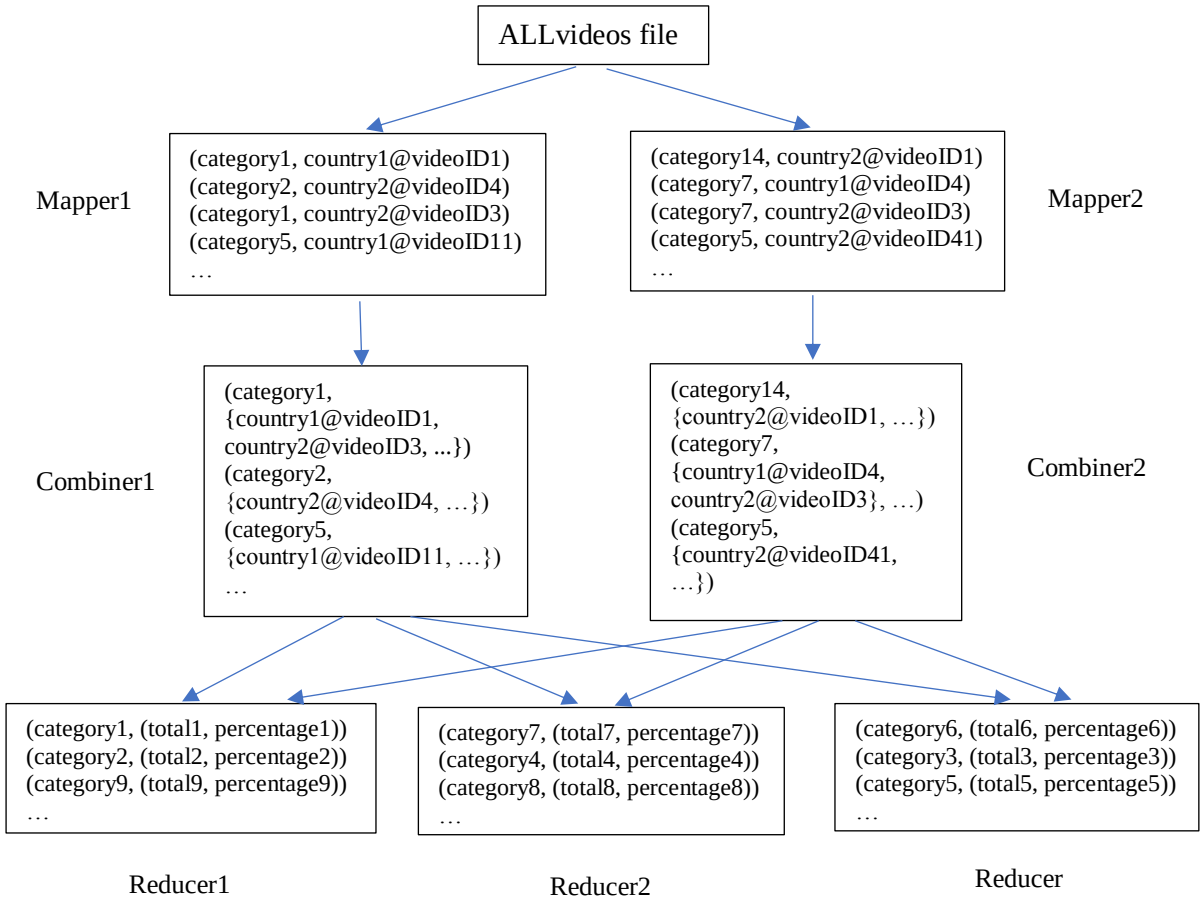
The data set I worked on is the **Trending Youtube Video Statistics** data from Kaggle (https://www.kaggle.com/datasnaek/ youtube-new).

Workload	Implementation	Programming Language
Category and Trending Correlation	Map Reduce	Java
Impact of Trending on View Number	Spark	Java

Workload: Category and Trending Correlation



One MapReduce job is used for this workload. It consists of only three phases: Map and Reduce.

In the Map phase, the input of the map function is a key-value pair. The key is the byte offset of the line. The value is the actual line content as a string. It outputs a list of (category, country@videoID) pair with respect to the two given countries.

There is a combiner function in the map phase. It does local aggregation after each map. The input key is the category. The input value is a list of the id of the video belongs to this category following with the country of the video. The output key is the category and the output value is a summary of the nonredundant country@videoID.

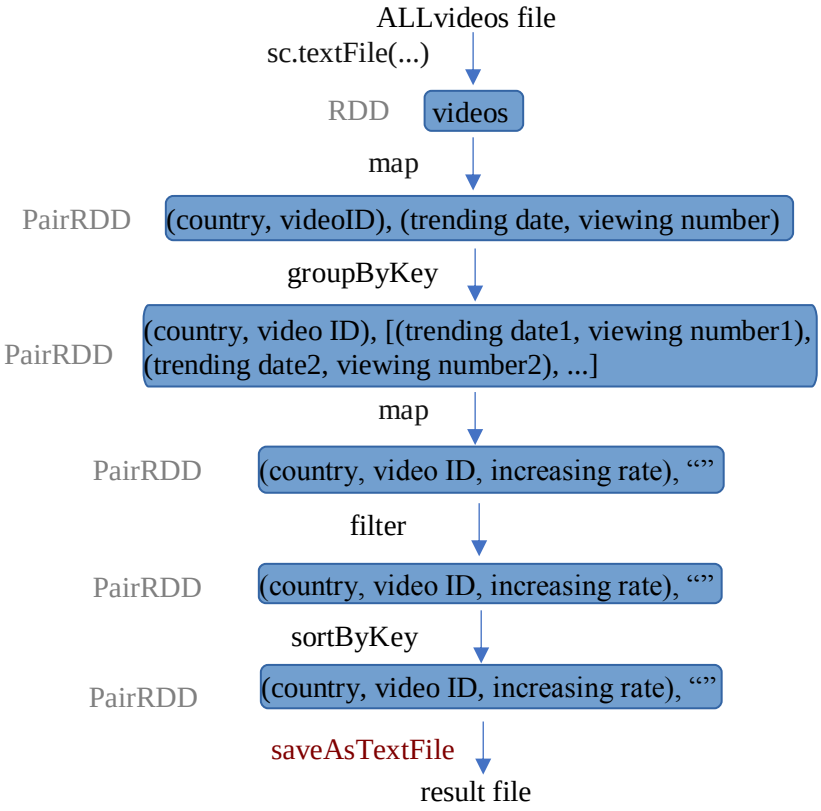
The Reduce phase only contain one reduce function. The input key is the category. The input value is a list of the summary of the country@videoID. The output key is still the category. It outputs the total number of the videos trending in the first given country followed by the percentage of them that are also trending in the second given country.

Parallelization

Both mapper and reducer phases can run in parallel. Mappers run in parallel on different partition of the input data. We have set to use 3 reducers, they run in parallel on different partition of the intermediate results. Each partition handles data related with a subset of tags.

The data set I worked on is the **Trending Youtube Video Statistics** data from Kaggle (https://www.kaggle.com/datasnaek/ youtube-new).

Workload: Impact of Trending on View Number



ALLvideos file is read in and mapped to create ((country, videoID), (trending date, viewing number)) RDD pair. Then the groupByKey transformation is applied to do the aggregation to create a summary. The transformed RDD pair is then mapped to create (((country, video ID, increasing rate), “”)) RDD pair. Then filter transformation is applied to filter the redundant values and sortByKey transformation are applied to get the final sorted result.

Parallelization

The ALLvideos files is read in then parallelize using the built-in SparkContent parallelize method. The map and mapToPair operations can run in parallel on different partitions of the videos contents. The groupByKey operations can run in parallel and pipeline with the next map, filter and sortByKey operation. Shuffling is required during the groupByKey operation. The filter and sortByKey operations can both run in parallel as well.