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

# Workload: Category and Trending Correlation

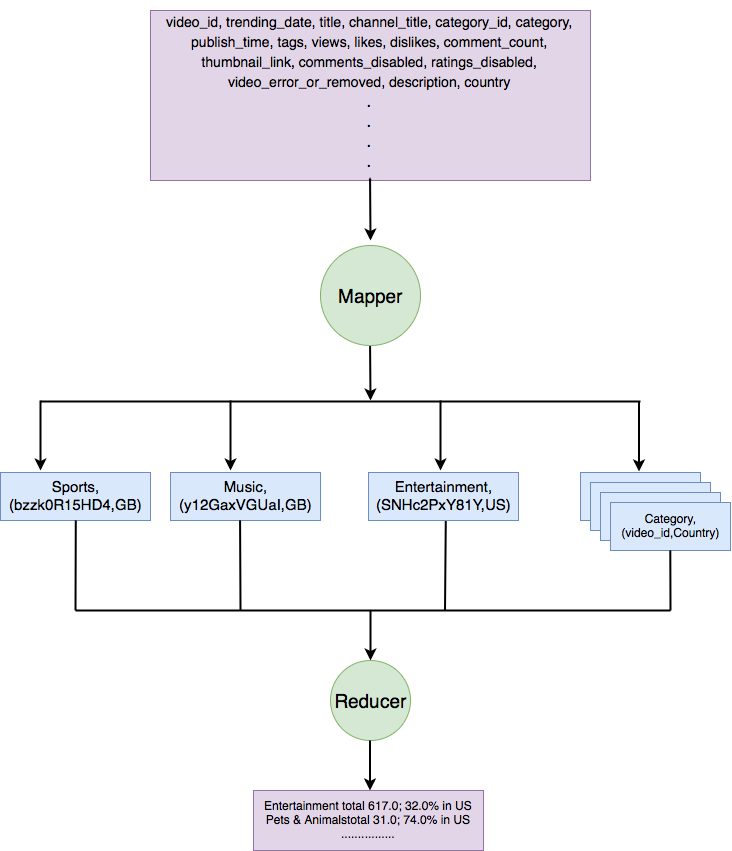
The sequence of transformations and actions are illustrated in Figure [1.](#page1)

Figure 1: MapReduce phase for the workload.

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

For each input row described above, the mapper use category as the key and delete the videos which are not belong to country1 and country2. Each row yields only one key and the value is the (video\_id,country).

The reducer received output from mapper sorted by Category, and count for the number of videos in the country1. Then count the number of videos in country1 also in country2.

I use a list to save all the video in country, then the size of the list is the number of videos in the counrty1. As for the number of videos in counrt2 also in country2, I use a Hashmap(video\_id, List<String>). The list is used to store the country which this video belongs to. Then for each one in the map, if the size of the list equals two, I add 1 to the count of overlap.

## 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 categories.

# Workload: Impact of Trending on View Number

Figure 2: Spark phase for the workload.

Allvideos file is read in and mapped to create {(Country,video\_id) ,(trendingdate,views)} RDD pair. Then I use group by key, the PairRDD becomes into {(Country,video\_id), [ (trendingdate1,views1) (trendingdate2,views2)…]} format.

Then a map transformation is applied to the RDD pair. In this phase, I use a list to store only two (trendingdate,views) whose trending date is the first one or second one. Then I extract the views to compute the change percent. If there is only one trending date in the list, I give it a percent of 0. What’s more, I create a new object, SecondaryKey, here. This object contains two keys in it, country and change percent. This object will be used to sort the pair RDD in the following process. The output pair RDD in this phase is {SecondaryKey,out\_put\_String}. The output string is like “US; O6K0ZPRqzFU, 5452.7%” and this string will be used in the final output phase.

Afterwards, a sort by key operation is applied to the RDD pair. I sort them according to two keys. The first one is country and the second one is the change percent. The I use the filter operation to delete the items whose change percent is lower than 1000%.

After sort phase and filter phase, a map operation is applies to save all the output\_string in to the RDD, then save the RDD as text file into the result file.

## Parallelization

The allvidoes 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. The groupByKey transformation can run in parallel and pipeline with the next map operation on different partition of the intermediate results. The sort by key operation and filter operations can also run in parallel. Each partition processes a few genres' data.