SCHOOL OF HUMAN SCIENCES & TECHNOLOGY

INDIVIDUAL CASE REPORT

Counting and sorting the top Tweets by country using MapReduce

Hadoop

SCHOOL OF HUMAN SCIENCES & TECHNOLOGY

Case Solution

The case presented can be solved using a single MapReduce job, with a Combiner and (potentially) a Partitioner. The steps to be carried out to get the desired output are as follows:

Step 1, MAP: Carry out a map function that returns only words starting with a # (in a similar manner to the Game of Thrones capital letter extraction carried out during the MapReduce Labs), using the country abbreviations as a key and the hashtags as a value. To avoid issues with using upper-case and lower-case hashtags, the map function should also be calibrated such that all the words (and especially the hashtags) are changed to lower case before the outputs are sent to the next stage. An example of the map job would be:

Tweet 1, Mapper 1: map(US, "Top cloud providers #aws#azure#google") = [("#aws", US), ("#azure", US), ("#google", US)]

Tweet 2, Mapper 2: map(US, "Big data companies #hortonworks#teradata#cloudera#aws") = [("#hortonworks", US), ("#teradata", US), ("#cloudera", US), ("#aws", US)]

Tweet 3, Mapper 3: map(ES, " Los componentes de #hadoop son #hdfs #yarn y #mapreduce") = [("#hadoop", ES), ("#hdfs", ES), ("#yarn", ES), ("#mapreduce", ES)]

Step 2, COMBINERS: Since the job is cumulative (counting the hashtags), the results of the map job can be sent into a Combiner that will carry out a partial count of the tags using the country key. It should be noted that since the Combiner is run on same nodes that run the map task, they will only count the results of a single Mapper. The output from the Combiners can be calibrated to look as follows:

Combiner 1: [("#aws", US), ("#azure", US), ("#google", US)] =
(US, [("#aws",1), ("#azure",1), ("#google",1)])

Combiner 1: [("#hortonworks", US), ("#teradata", US), ("#cloudera", US), ("#aws", US)] =
(US,[("#hortonworks",1), ("#teradata",1), ("#cloudera",1), ("#aws", 1)])

Combiner 3: [("#hadoop", ES), ("#hdfs", ES), ("#yarn", ES), ("#mapreduce", ES)] =
(ES, [("#hadoop",1), (#"hdfs", 1)("#yarn",1), ("#mapreduce",1)])

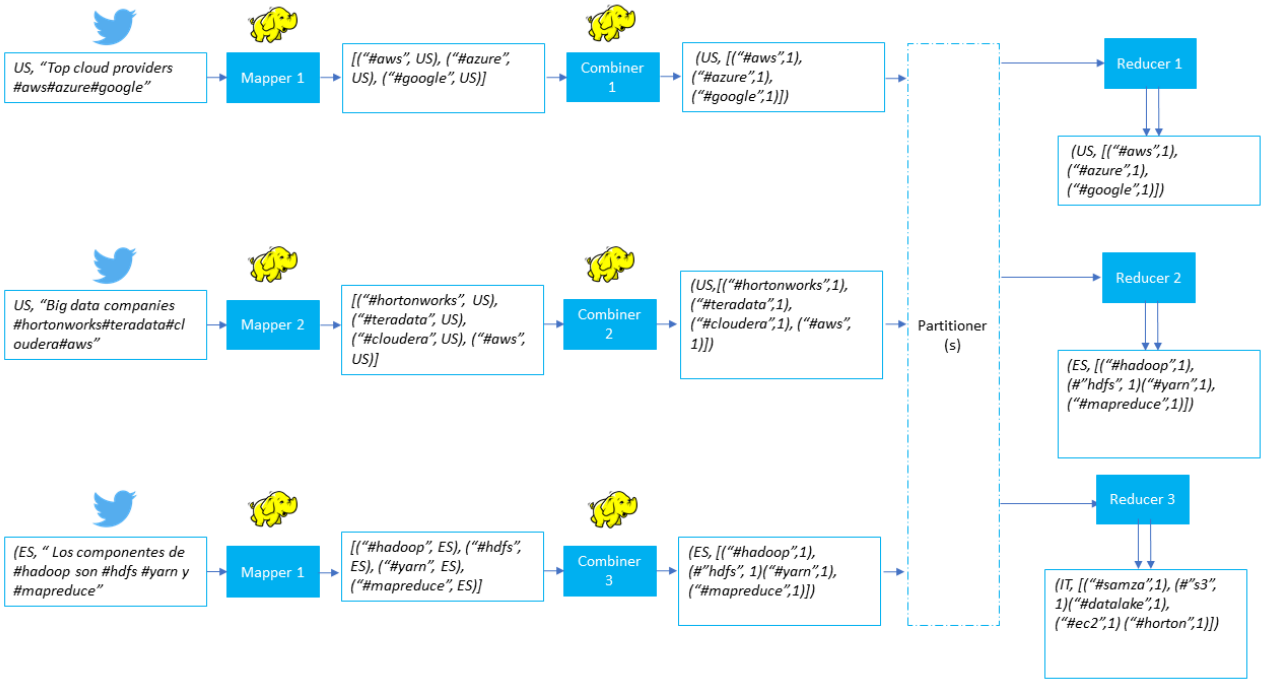
SORT -SHUFFLE: Prior to the reduction process, Hadoop should automatically carry out a sort-shuffle to sort all the results from the Combiner using the country as a key and send the all the hashtags with the same country key into the same Reducer.

REDUCE JOB: Using the country abbreviation as the key, a reduce job that counts the sorted tweets per country should be carried out. The Reducer must also be configured using a chained job such that it sorts by the descending order of counted values and limits the output to only 5 values as follows:

Reducer 1: reduce(ES, [("#hadoop",1), (#"hdfs", 1)("#yarn",1), ("#mapreduce",1)]) = (ES, [("#hadoop",1), (#"hdfs", 1)("#yarn",1), ("#mapreduce",1)])

Reducer 2: reduce(US, [("#aws", [1,1]), ("#azure", [1]), ("#google",[1]), ("#hortonworks", [1]), ("#teradata", [1]), ("#cloudera", [1])) = (US,[("#aws",[2]), ("#azure", [1]), ("#google",[1]), ("#hortonworks", [1]), ("#teradata", [1]), ~~("#cloudera", [1]))~~(not in top 5)~~.~~

Basic Process Flow



Problems Analysis & Solutions

In a "normal" MapReduce job, the default Reducers are configured in a way such that they carry out a full job for each key value pair before moving onto another key value pair. In the case presented above for example, if we assume that only one Reducer was configured, it will first complete the job for all hashtags for Italy before moving onto Spain and the US (alphabetical order). This system works well for jobs in which most of the mapping outputs are more or less of the same size. In certain situations, however, and particularly in this one, even with a Combiner, the normal configuration cannot be fully relied on for the following reasons:

a) A Combiner is optional in MapReduce. As such, Hadoop can decide whether to execute a Combiner function. Without a way of ensuring that the Combiner will execute every single time, a job that relies on it decreasing the amount of data sent into the Reducers can lead to problems in the sense that a failed Combiner execution can increase in input volumes in the Reducers, which can severely delay the job. To allow for flexibility in the reduction process in the case that a Combiner execution fails, the number of Reducers can be increased more than the default such that even if there is an unexpected failure of the Combiner, there are Reducers with excess capability standing by.

b) A second issue is that, for example, while Spain has a relatively lower Twitter usage compared to the US, at times (like elections or the FIFA World Cup) it may experience higher than average volumes. This will mean that the Mappers (Combiners) for the tweets from Spain will propagate unusually

large volumes of data, leading to much higher key value inputs for the Reducers, something that may be difficult to manage even with excess Reducers. To manage the job during these unusually high-volume periods, Partitioners can be used. Placed right after the Mappers (Combiners in this case), Partitioners use the keys (the country abbreviations) and send all the Mapper outputs with a certain key (e.g ES) into one Reducer. By having a dedicated Reducer for a certain key (country), a queue is avoided, and the job can run more smoothly.

c) There is also a relatively more permanent problem of certain countries, like the US, having a higher percentage of the overall population using Twitter. The twitter users in the US are also more active than in other countries. Thus, tweets from the US will normally have a larger volume than most other countries. Consequently, the Mappers (or Combiners) will propagate a lot more values into the Reducers for the US than it will for Spain. Since by default each Partitioner only sends the Combiner output into one Reducer, having a dedicated Partitioner for US keys may not fully solve the issue. The US may have enough volume to overwhelm even a single dedicated Reducer. Combining this with a worst-case scenario of a failed Combiner execution and a time when Spain is also experiencing unusually high volumes, it can lead to a major delay (or even a termination), of the MapReduce job. For this problem, a custom Partitioner, which allows the storage of one key value pair into more than one Reducer, can be configured.

   It should be noted that the number of Partitioners used must be equal to the number of Reducers so the configuration of b) and c) should always be done keeping a) in mind.

Another important consideration is that of duplicate values. What if the same tweet repeated the hashtags? The accepted method of solving this problem is to use a separate MapReduce job to transform the data. The deduplicating is carried out as a MapReduce job where the mappers take the input HDFS file, read line by line and put the entire text (in this case Tweet) as key and a NullWritable.get() as the value i.e it would output the Tweet as the key as NULL as the value. The reducer in turn would group the NULLs by key and output only the keys (NULLs have no relevance), assuring uniqueness of the data.

In the presented case however, since our purpose is to count and sort the hashtags and not the Tweets, deduplication should be done after the mapping phase, which will not only ensure that duplicate hashtags get removed, it will also mean that the deduplication is done only on the relevant data rather than the entire input dataset. Ideally, an additional functionality to the Combiner (also known as the mini-reducer) should be added, which takes the hashtags as keys and outputs NULLS as values prior to carrying out the partial count described in Step 2 above.

In summary, the potential solutions outlined above should help with the most pertinent data and scalability issues with regards to this MapReduce problem. Any technical issues with regards to HDFS and YARN are assumed to be not be required as a part of this case report and therefore have not been considered.