

Task 1 consists of 3 main steps: step 1 is to delete all the stopwords from both input files and sum the frequencies of each word, step 2 is to get the number of common words and step 3 is to sort the common words in descending order and only take top 15 words out of the result.

Step 1:

In the setup function of TokenizerWCMapper class, I am following the template given for the task and just changed the path for stopwords file, which is `new Path("hdfs://172.17.0.2:9000/user/root/commonwords/data/input/stopwords.txt")` where the test is done by using docker clusters (172.17.0.2 refers to the master IP address and 9000 is the port of HDFS namenode). After that, for each words read from the opened stopwords file, there will be a conversion of the words to be lower-cased. Similarly, in the map function, which is going to map words from task1-input1 and task1-input2 file, I implemented a lower-case conversion of the words only during the if condition so that it can match with the stored stopwords (also in lower-case) and the condition checking between file words and stopwords will proceed smoothly.

Since the mapper will produce key values in form of (word, 1), the values grouping will be done according to the key word and for the length of the values (each value is 1) do a summation and emit the key word and the word count. I leave the function just like in the original template. This set of map reduce functions is done for each of the input files.

Step 2:

Then we proceed to the step 2 which is getting the number of common words. To achieve this, each mapper will process each file in which stopwords have been filtered out earlier. In the mapper function the key is still the key word and the value is the frequency that was summed and originating file appended to the value along with the frequency. In the end, the key value emitted by the mapper will be in the form of (word, frequency_s1) or (word, frequency_s2).

Lastly, in the reducer first I define 2 variables which is `counter` and `min`. `counter` is added during every iteration of the values loop (will be explained later) and the purpose is to check if each key word that was grouped has 1 or 2 values. If a particular key only has 1 value, that means the key word is not common because one of the files does not have that word. On the other hand, if the key has 2 values, that means the word is common. Hence, only emit a key value pair if `counter` is more than 1. The next problem is to decide which frequency among those 2 values to choose from. First, a loop is done through all the values and for each value (which is in form of word frequency) compare it with the `min` variable. Since we want the smaller value, there is a need to compare each value with `min`. If the value is smaller than `min`, `min` value is set to be the same as the looped value. After the loop is done, the `min` value will have the smaller frequency.

Step 3:

To sort the common words by frequency in descending order, we need to set the frequency as the key in the sorting mapper since key is the element that gets sorted unlike value. The problem to tackle next is how to order in descending way since the default order format is ascending. To achieve that, I multiply each of the frequency by -1 (e.g. 54 becomes -54) before emitting the key value pair from the mapper function. In the shuffling phase, words that have the same key (in this case frequency) will be grouped together and in the sort phase the key will still get sorted in ascending order (e.g. -54, -49, -20, -12, ...) but we can convert the numbers in the reducer function so it looks as if they were in descending order by multiplying each key by -1 again before emitting the key value pair. The final result will come in the following format: 54 word1, 49 word2, 20 word3, 12 word4, ... and sorting is finished.

Along with the reverting process in the reducer function, a `counter` is also incremented during each iteration of looping through grouped values. That means every time a key value pair is emitted, the `counter` will increase by 1. If `counter` equals to 15, break the loop and the rest of the key value pairs that have not been processed by reducer will not be processed. Since the reducer function is called for every key values pair, we need to set a condition to check the `counter` whether it has reached 15 or not. If it has, there is no need to reduce the pair.

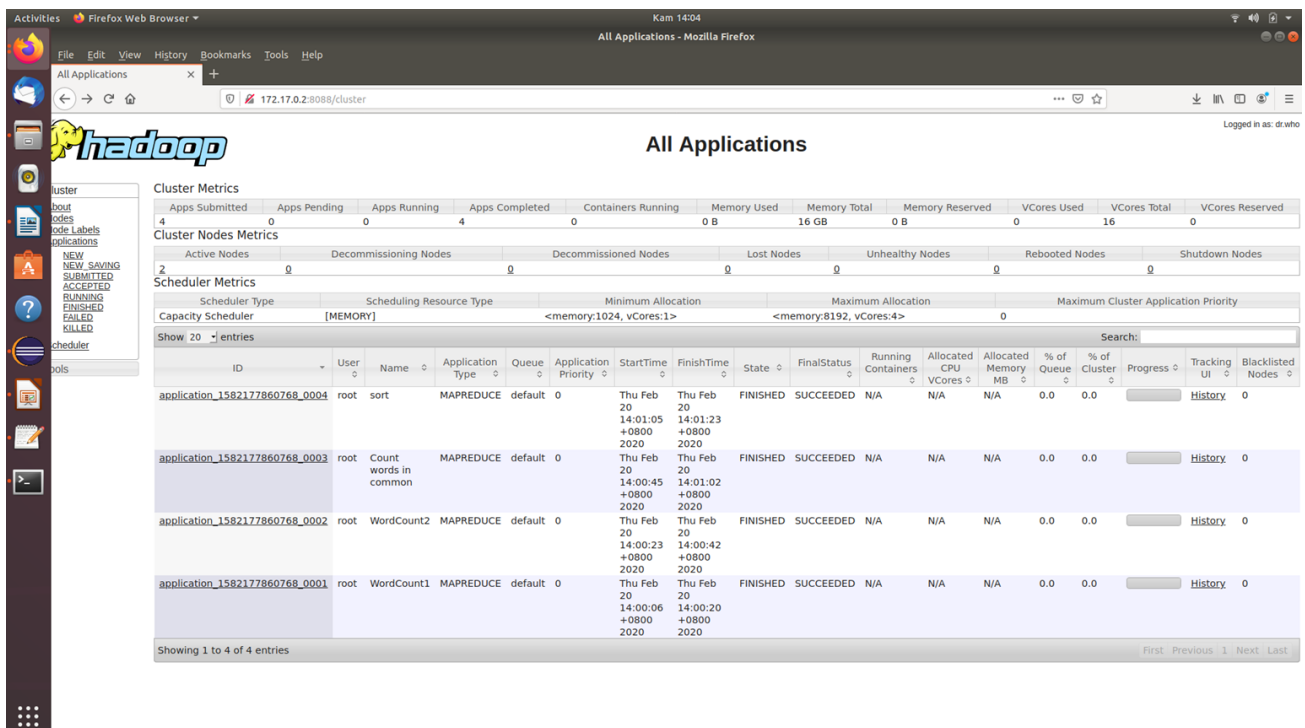
Result Snapshot:

Considering the inputs are stopwatch.txt, task1-input1.txt and task2-input1.txt which none are modified at all, here is the common words result:

```

191  a
77   Project
53   Gutenberg-tm
32   work
27   electronic
22   works
19   Gutenberg
19   terms
17   will
14   good
14   Foundation
13   Literary
13   Archive
12   it.
12   copyright

```



The screenshot shows the Hadoop All Applications web interface. The main table displays application metrics for four applications. The table has columns for ID, User, Name, Application Type, Queue, Application Priority, Start Time, Finish Time, State, Final Status, Running Containers, Allocated CPU Vcores, Allocated Memory MB, % of Queue, % of Cluster, Progress, Tracking UI, and Blacklisted Nodes.

ID	User	Name	Application Type	Queue	Application Priority	StartTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU Vcores	Allocated Memory MB	% of Queue	% of Cluster	Progress	Tracking UI	Blacklisted Nodes
application_1582177860768_0004	root	sort	MAPREDUCE	default	0	Thu Feb 20 14:01:05 +0800 2020	Thu Feb 20 14:01:23 +0800 2020	FINISHED	SUCCEEDED	N/A	N/A	N/A	0.0	0.0	<div></div>	History	0
application_1582177860768_0003	root	Count words in common	MAPREDUCE	default	0	Thu Feb 20 14:00:45 +0800 2020	Thu Feb 20 14:01:02 +0800 2020	FINISHED	SUCCEEDED	N/A	N/A	N/A	0.0	0.0	<div></div>	History	0
application_1582177860768_0002	root	WordCount2	MAPREDUCE	default	0	Thu Feb 20 14:00:23 +0800 2020	Thu Feb 20 14:00:42 +0800 2020	FINISHED	SUCCEEDED	N/A	N/A	N/A	0.0	0.0	<div></div>	History	0
application_1582177860768_0001	root	WordCount1	MAPREDUCE	default	0	Thu Feb 20 14:00:06 +0800 2020	Thu Feb 20 14:00:20 +0800 2020	FINISHED	SUCCEEDED	N/A	N/A	N/A	0.0	0.0	<div></div>	History	0

Showing 1 to 4 of 4 entries