How to run with the standalone jar

1. Make a folder "inputdata" in /user/s3754699, upload the 3 text files into inputdata
2. Upload simpleWordCount-0.0.1-SNAPSHOT.jar to HDFS /user/s3754699
3. Cpoy simpleWordCount-0.0.1-SNAPSHOT.jar from HDFS to EMR
$ hadoop fs -copyToLocal /user/s3754699/simpleWordCount-0.0.1-SNAPSHOT.jar ~/
4. Run the Jar
- Task 1: hadoop jar simpleWordCount-0.0.1-SNAPSHOT.jar BigData.Assignment1.simpleWordCount.task1 /user/s3754699/inputdata /user/s3754699/output1
- Task 2: hadoop jar simpleWordCount-0.0.1-SNAPSHOT.jar BigData.Assignment1.simpleWordCount.task2 /user/s3754699/inputdata /user/s3754699/output2
- Task 3: hadoop jar simpleWordCount-0.0.1-SNAPSHOT.jar BigData.Assignment1.simpleWordCount.task3 /user/s3754699/inputdata /user/s3754699/output3
- Task 4: hadoop jar simpleWordCount-0.0.1-SNAPSHOT.jar BigData.Assignment1.simpleWordCount.task4 /user/s3754699/inputdata /user/s3754699/output4

Analysis on different numbers of nodes

Below is the CPU_MILLISECONDS for each MAP task and each REDUCE task for different numbers of nodes

Number of nodes	Mapper		Reducer A	Reducer B	Reducer C
3	71340		790	670	510
5	72500		820	700	540
7	69860		750	660	460

We can see that, in general, the CPU_MILLISECONDS for each task doesn't change much as the number of nodes change, so there's not much to be discussed in terms of the relationship between number of nodes and job efficiency at this point, unless further research is done with more different numbers of nodes and a larger input data.

However, there are 2 things we can do to improve the efficiency here. By default, there are 3 reducers launched, but only 2 are actually doing the work, because the third reducer output is empty. Therefore, we can take advantage of the 3 reducers we have by assigning, for example, short words to reducer A, medium words to reducer B, long words and extra-long words to Reducer 3. With the workload evenly spread out, it can reduce the CPU_MILLISECONDS spent on each reducer, as 3 reducers are working in parallel.

Similarly, for the Mapper task, the total file to be processed is one, so there's only one mapper doing the work. We can also split the input file into multiple smaller files, so that the mapper workload is evenly spread out and the mappers work in parallel. Thus the time spent on Mapper task as a whole would be reduced, and it also improves the efficiency for the whole job, as the reducers won't need to wait for so long until the mappers finish working.