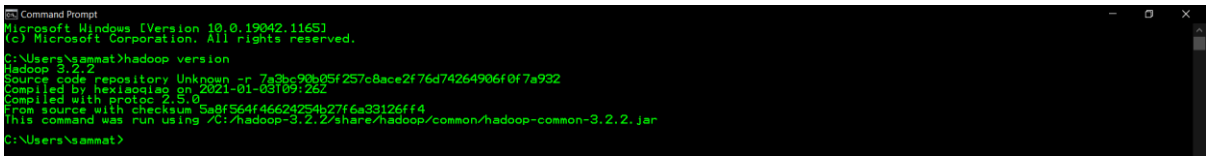
CS6220 – Assignment 1 – Report

Sambhav Mattoo

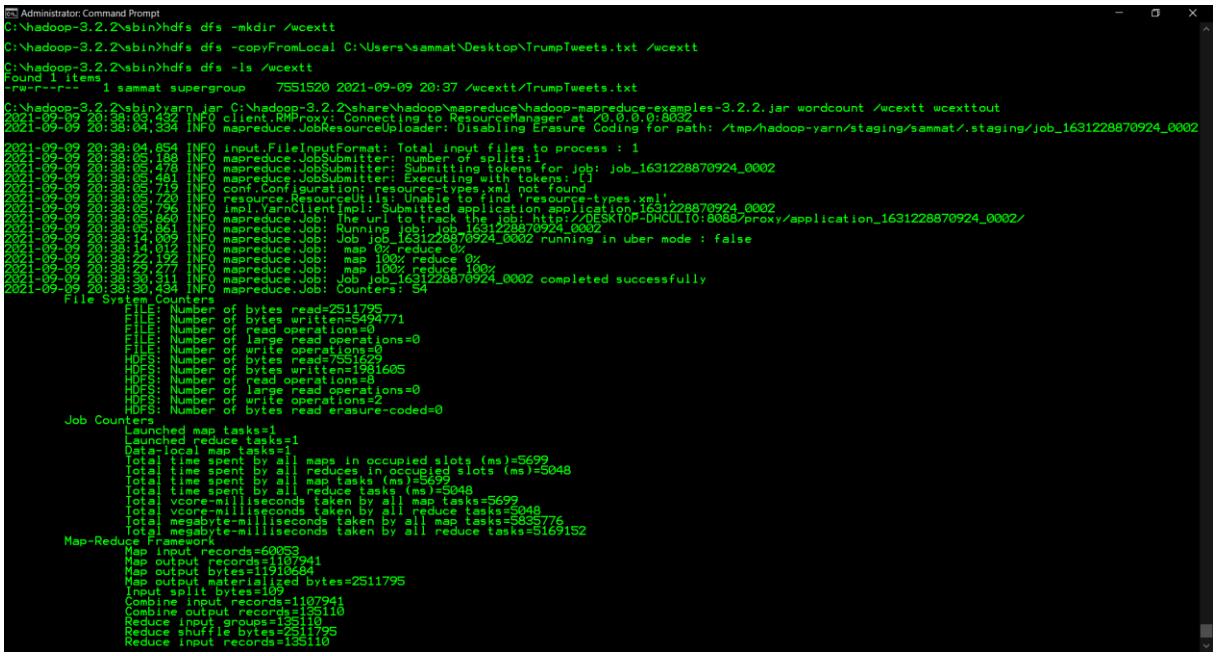
My attempt at the Assignment 1 for CS6220, I did the first option for the Programming Assignment.

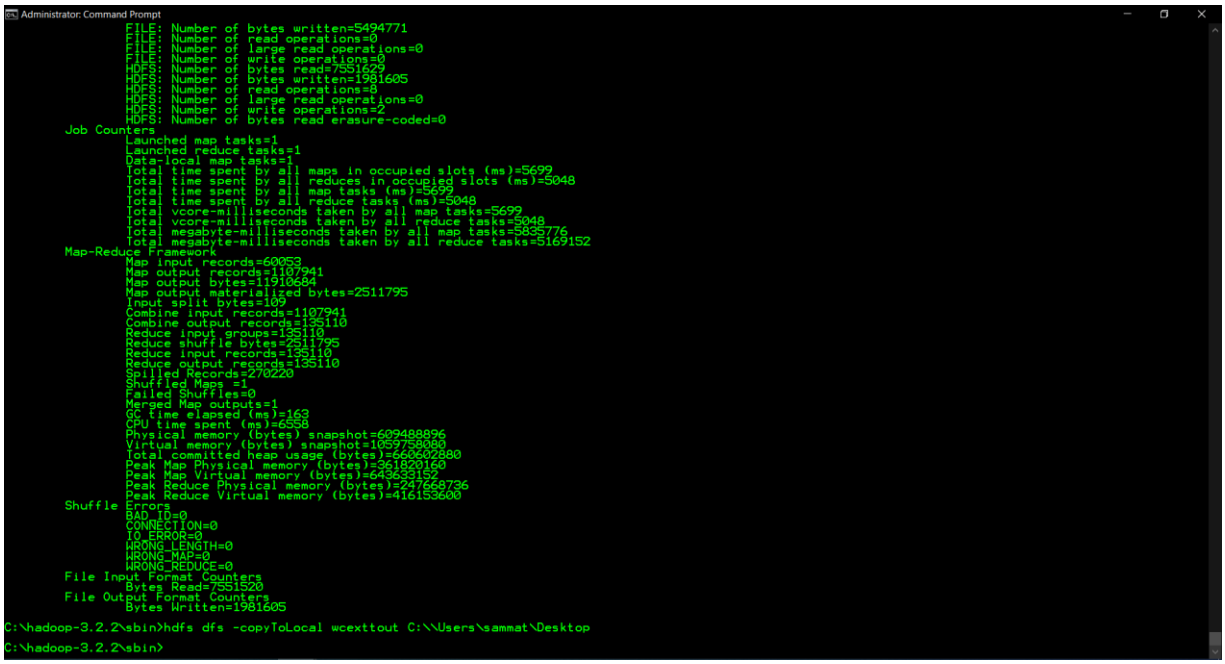
Firstly, let me start with system specifications. I am using Hadoop Version 3.2.2 (HDFS + MapReduce) single node cluster on my HP Pavilion 15-cc1xx laptop, which runs on an Intel Core i7 CPU (1.80GHz, 1992 MHz, 4 Physical, 8 Logical cores, x64 Architecture), using 16GB of RAM and over 1.8TB of disk space. The operating system is Windows 10, Home version.

The first task of this programming assignment was "Install HDFS and Hadoop MapReduce on your laptop." I used [1] as a rough guide to do so, as advised on Piazza as well. With Windows, a lot of problems relevant to Java versions, folder permissions and environmental variables occurred; I used a lot of suggestions from stackexchange.com and from CHD (Cloudera Hadoop Distribution) forums to solve them. Here is a small screenshot of the Hadoop version command for my system.

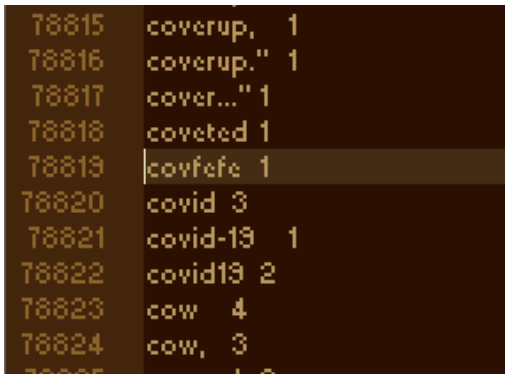


The next task of this programming assignment was "Run the word count map-reduce program, and report the runtime for two different sizes of datasets." During the installation, I came to know of an example .jar file implementing this very program already present as an example, prepackaged with the distribution: it can be found at "~\hadoop-3.2.2\share\hadoop\mapreduce\hadoop-mapreduce-examples-3.2.2.jar", where ~ denotes whichever directory Hadoop was installed in. Further [2] gave me a quick guide on how to actually use the HDFS to operate on a real dataset, using this example file. Here are some screenshots from my first trials at using Hadoop.





The datasets I wanted to use for a first-pass analysis were the Trump Tweets Archive [3] and the King James Bible on Project Gutenberg [4]. The latter of these has a file-size of 4.97MB (0.01MB more on disk, possibly because of metadata), and the former has a file-size of 7.20MB, a difference at which I hoped to observe some sort of scaling effect for comparison. In order to compare runtimes, I used the "Elapsed" field as available on the Resource Manager hosted at http://localhost:8088/cluster, which I had configured in my settings. However, I also have enclosed more detailed log files of both these tasks in my submission, for further perusal. I found that the smaller dataset took a longer time (~26 secs) than the Trump Tweets archive (~23 secs), in total time for operation, GC time (37ms vs 36ms) and CPU time (1920ms vs 1811ms).



I did observe some interesting features of the data, however. Some of the reasons behind this scaling could be the several other concurrently running applications on my system, simple experimental error, the datasets being too small to see any scaling effect as well as the vocabulary (average word size), and therefore the number of string-matching operations occurring under the hood of the larger database being fewer. However, I had not compiled this word count Map-Reduce program myself, and referred myself to [5], wherein I found that the example .jar file was compiled based on the first version of the given tutorial problem.

I then tried compiling the second, and improved version of the word count implementation specified in that tutorial. This required some minor changes to the code in order to better fit the way environment variables are set on Windows vs. Linux. I'd like to note at this point that a.) the object of the exercise specified was to "run" the word count program and b.) using the code from the tutorial, with due referencing, was permitted to be used, as expressed on Piazza.

I used this code to better benchmark the runtime for the two testing datasets that I had chosen, as illustrated in the following table, for 5 trials, cataloguing total, map and reduce times by D1 (the Bible) and D2 (the Trump Tweets Archive).

	N = 1	N = 2	N = 3	N = 4	N = 5
D1 tot.	22s	22s	21s	22s	22s
D2 tot.	23s	27s	31s	30s	30s
D1 map.	5165ms	5118ms	4989ms	4998ms	5106ms
D2 map.	5543ms	7000ms	8013ms	8057ms	7707ms
D1 red.	4417ms	4690ms	4330ms	4334ms	4774ms
D2 red.	4557ms	6564ms	7533ms	6504ms	7211ms

This is accompanied by output files, and also, output files with the processing possible through Hadoop i.e., removing case, leaving out punctuation marks and common words etc., which is the primary advantage of WordCount2 over WordCount1; in terms of the algorithm, both are almost the same. Now in this table, we can see the scaling effect far more accurately than the normal precision, with ~7.2 secs total time being taken more on average by the larger dataset, which is approximately linear with respect to data-size.

The final task of this assignment was "Using MapReduce to solve another problem. You may choose one of the following or create a problem yourself: Consider a dataset of 20 files, print the top 100 words occurring in the most files." My approach to solving this was as follows: most of the work was already done by the second version of word count map reduce, which could take in 20 files, and create one output list containing the frequency of every word in all of those 20 files. I then implemented a second map reduce that would nest on top of the first, that started the map step by reversing the data in the key and value pairs and sorting it, and a reduce step that collected the top 100 of the words with the largest frequency. For exact details on the functioning, please look at my comments in the Problem.java file for details. My implementation was based more or less off the Piazza discussions and hints left by the instructor, and in my opinion is not the best way to solve this problem. Rather, a priority queue-based method might be better, where the key value pairs are stored in a tree during reduce steps such that only 100 nodes are in the tree at any time. However, in my opinion, this too isn't the right way of doing it; I couldn't implement that yet because my skill with Hadoop has some catching up to do.

The best and most accurate way to solve this problem would be a single key as a word and a vector of integers as a value in the key value pair, where the vector stores the occurrences of the word in each of the datasets, with the priority queue system being based on the sum of the integers in the vectors, and that a vector would be out of consideration if any element in it were to be = 0 (implemented in the reduce step). This method would be the closest to the answer, because unlike the other implementations it would contain words that are truly in common across all datasets; if one dataset has a word that occurs in a very high frequency but the other datasets do not contain said word, then it would not appear in the final list.

The datasets I chose to demonstrate my 2-step Map Reduce solution were the top 20 most popular Charles Dickens books available free on Project Gutenberg [6]. Their sizes in KB, in decreasing order of popularity (see file naming scheme) were "788, 177, 1000, 1930, 933, 1900, 611, 1760, 1830, 1880, 1860, 182, 1220, 184, 1890, 2000, 137, 501, 609, 1430".

I started with a full solution; taking all these files, and first doing WordCount2 on the lot of them, followed by using my Problem class on it. The former took ~1 min total time to compute, all map tasks taking 246663ms, and all reduce tasks taking 22940ms, and the second "Problem" map reduce task took but 21s in total, with all map tasks taking 4356ms, and all reduce tasks taking 4968ms. This can be corroborated with terminal screenshots I took instead of syslog.txt files, as several of them generated, each with disparate times, presumably because of the number of input files processed one after the other. Again, it was interesting to see how this total score measured up with increasing size of data, so I devised another series of runtime tests, that is, with the first dataset, with the first five, with the first ten, the first fifteen and finally the full 20 datasets. Here are the results:

	Sets = 1	Sets = 5	Sets = 10	Sets = 15	Sets = 20
M/R 1 tot.	21s	27s	39s	52s	60s
M/R 1 map.	4827ms	55888ms	133128ms	204719ms	246663ms
M/R 1 red.	4531ms	4789ms	11897ms	11864ms	22940ms
M/R 2 tot.	21s	21s	22s	22s	21s
M/R 2 map.	4809ms	4493ms	4027ms	4175ms	4356ms
M/R 2 red.	4393ms	4622ms	4819ms	4655ms	4968ms

This illustrated several interesting points. As expected, the "Problem" part of the solution, or the sorter, really had more or less consistent performance on datasets because the file it was working with had more or less the same size. On the other hand, the given Word Count Map Reduce that really drives the top 100 words sorter had a total runtime that correlated almost linearly with dataset size, with some given positive y-intercept implying that some part of the runtime is not necessarily dedicated to problem computations. On the other hand, we see a very interesting result; as dataset size grows, the amount of time spent on map tasks increases at a rate much higher than the increase in time spent on reduce tasks. I believe this is simply because while the mappers have a larger data size to map, the actual reduction does not deviate from what is approximately the same lexicon/ dictionary across datasets; the only thing that changes are the "values". That is, map tasks seem to grow almost quadratically with data size (in the sense, not linearly) because they "map" the whole dataset, while as reduce tasks map the dictionary that constitutes the dataset, which is why it doesn't really scale to dataset size, but rather, one can assume, the number of "new words" that are added with each Charles Dickens Novel I add to the pool. The interesting corollary is that the reduce task should scale in size with the map task given that each one of these 20 books is in a different language (or comes from a different lexicon).

A note to the grader: output_a, captureA etc... corresponds to the Word Count part of my 2-step 100 words process, and those files ending with "b" correspond to the "Problem.java" part of my code.

References:

[1]: https://towardsdatascience.com/installing-hadoop-3-2-1-single-node-cluster-on-windows-10-ac258dd48aef

[2]: https://www.srccodes.com/run-hadoop-wordcount-mapreduce-example-windows/

[3]: https://www.thetrumparchive.com/

[4]: https://gutenberg.org/ebooks/30

[5]: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

[6]: https://www.gutenberg.org/ebooks/search/?query=charles+dickens&submit_search=Go%21