Map Reduce Assignment

John Cummins

16373731

Brief explanation of program:

**Reading in file**

Firstly I created a list of all the files passed in as command line arguments. I then used a buffer reader to read in each file from the file list. Each file in the list was concatenated together to make one string. I then used replaceAll to get rid of any characters in the string. These included character like full stops, commas and quotations, which would interfere with grouping the words. This string was then put into the hash map.

**Thread Pool**

The first argument entered represented the required number of threads needed. This was entered as a string and then parsed to an int. A new executer was created with a fixed number of threads in the pool. The executer ensures there are no more than the specified threads running at the same time. The specified the number of threads was declared by passing in the “threadsNum” variable talked about previously. The new thread from the original code was taken out and executor.execute(() was put in instead followed by a to the map(file, contents, mapCallback)); This allows the map to run in a thread pool.

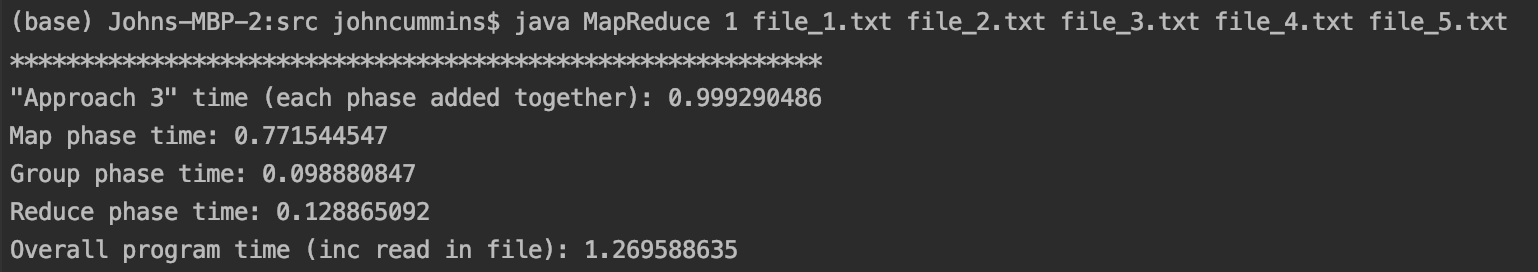
**Time**

I used system.nanoTime to keep track of the run time for each phase of the map reduce. This was done by placing a long system.nanoTime variable at the start and the end of each phase. I took the start time away from the end time in order to get the duration. This was in nanoseconds I then needed to divide by 1 billion to display the time in seconds. I also recorded the time from the start to the end of the program. The main time which I used in testing was the overall time. This consisted of adding up each individual time from each phase to give me an overall run time.

Below is a screenshot of the files and file sizes used in the first three set of tests:

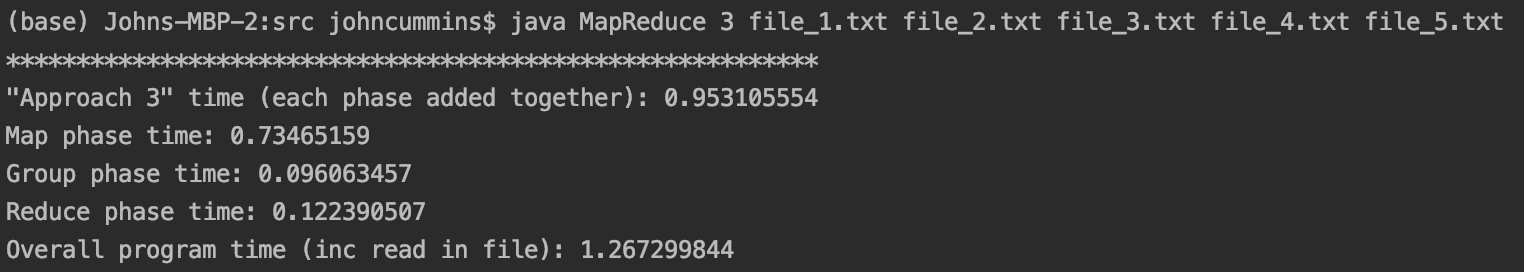
**Testing approach 3 with thread pool:**

**1 thread**



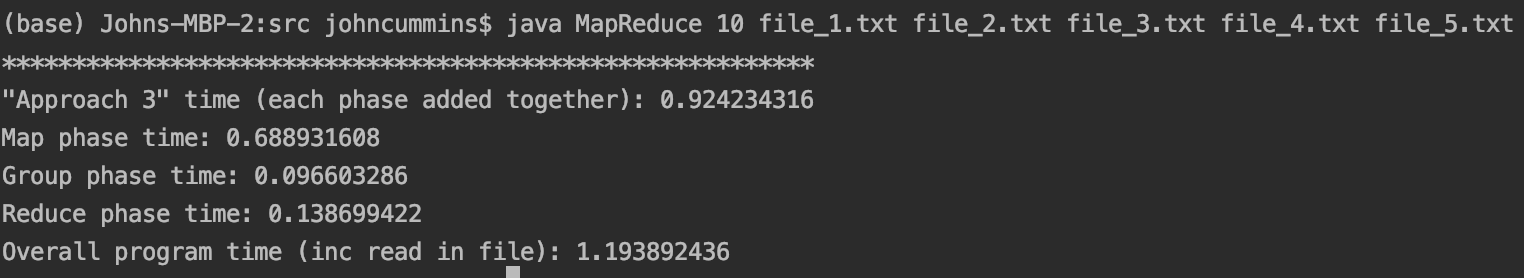
Above are the results from running the program with the Map and Reduce functions (in approach 3) assigned to a Thread pool with the number of threads set to 1. 5 text files were passed in, ranging in sizes from 213 KB to 1.2 MB. As you can see the map phase took 0.771 seconds to run. The group phase took 0.098 seconds to run. The reduce phase took 0.128 seconds to run. The run time for each of the three phases added together was 0.999. The total run time for the program (including reading in the files) was 1.269

**3 threads**

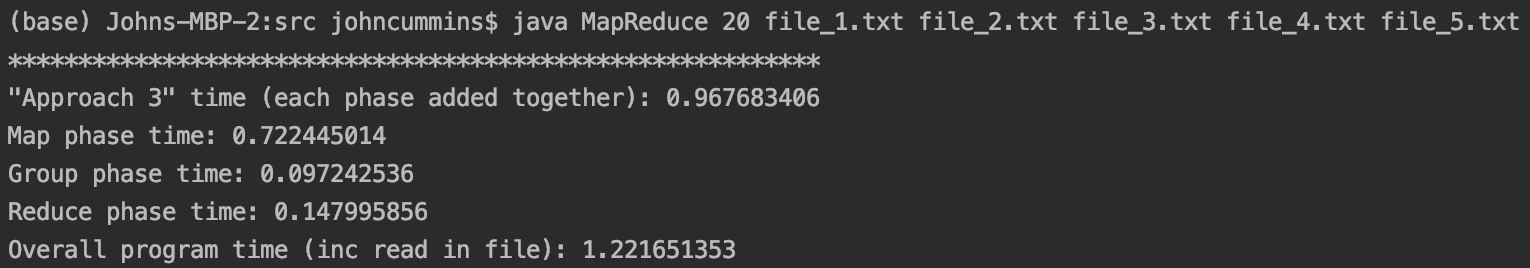


Above are the results from running the program with the Map and Reduce functions (in approach 3) assigned to a Thread pool with the number of threads set to 3. The same 5 text files as above were passed into the program. As you can see the map phase took 0.734 seconds to run, this was 0.037 seconds faster than using one thread. The group phase took 0.096 seconds to run, this was 0.002 seconds faster than using 1 thread. The reduce phase took 0.122 seconds to run, this was 0.006 seconds faster than using 1 thread. The run time for each of the three phases added together was 0.953, this was 0.46 seconds faster than using 1 thread. The total run time for the program (including reading in the files) was 1.267. which was 0.002 seconds faster compared to using just 1 thread. Overall using 3 threads leads to a faster run time from every aspect of the program.

**10 threads**



Above are the results from running the program with the Map and Reduce functions (in approach 3) assigned to a Thread pool with the number of threads set to 10. The same 5 text files as above were passed into the program. The map phase took 0.688 seconds to run, this was 0.083 seconds faster than using 1 thread. The group phase took 0.096 seconds to run, this was the similar time to using 3 threads. The reduce phase took 0.138 seconds to run, this was 0.01 seconds **slower** than using 1 thread. The run time for each of the three phases added together was 0.924, 0.075 seconds faster than using 1 thread and 0.029 seconds faster than 3 threads. The total run time for the program (including reading in the files) was 1.193. Both the map was slightly faster, the group was pretty much the same and the reduce was actually slower than using 1 thread.

**20 threads**

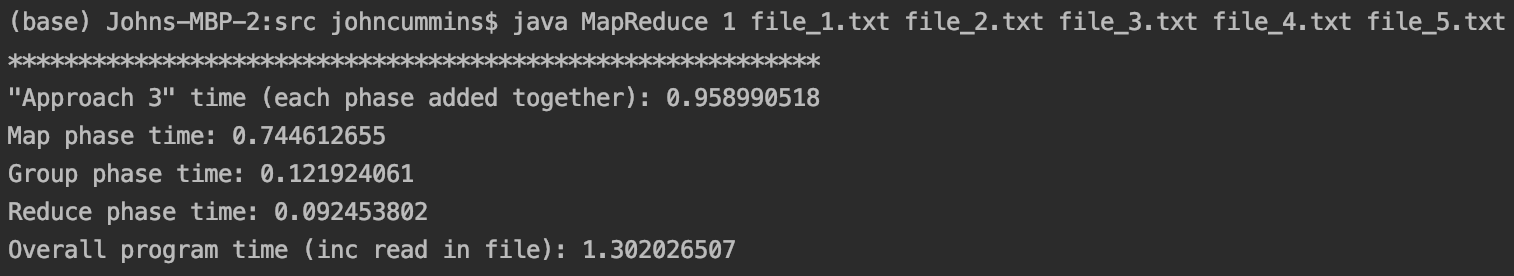
Above are the results from running the program with the Map and Reduce functions (in approach 3) assigned to a Thread pool with the number of threads set to 20. The same 5 text files as above were passed into the program. The map phase took 0.722 seconds to run, this was 0.049 seconds faster than using 1 thread. The group phase took 0.097 seconds to run, this was the similar time to using 3 threads (0.001 secs faster). The reduce phase took 0.147 seconds to run, this was 0.019 seconds **slower** than using 1 thread. The run time for each of the three phases added together was 0.967, 0.032 seconds faster than using 1 thread and 0.014 seconds slower than using 3 threads, and 0.043 seconds slower than using 10 threads. The total run time for the program (including reading in the files) was 1.221.

Overall the fastest run time for the combined three phases was with 10 threads, 3 threads was 0.029 seconds slower, 20 threads 0.014 seconds behind that and 1 thread was 0.032 seconds slower again.

**Comparison to approach 2 and 3 being assigned to a thread pool**

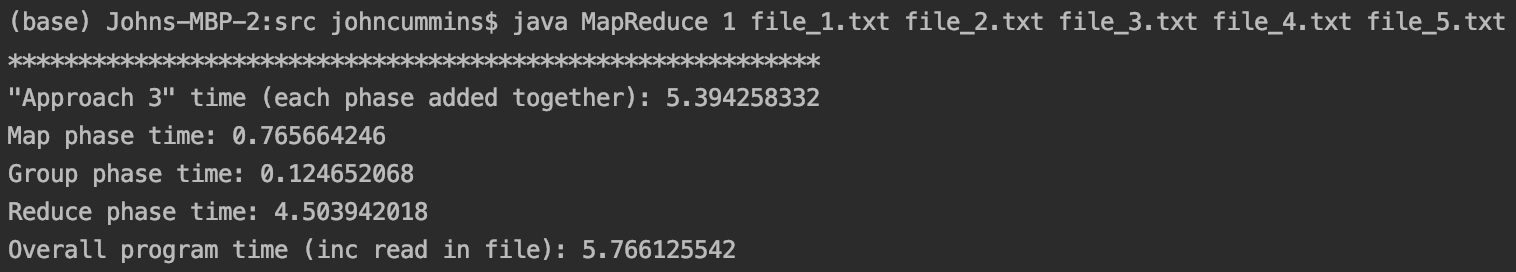
For the purpose of comparing both the approach 2 and 3 with thread pooling, I will compare the results to the best set of results I got from the thread pooling approach. This was the case when the number of threads was set to 10.

**Testing approach 2 without pool threading:**

****

The screenshot above is the results from running the map reduce program using approach 2 and without the use of thread pooling. As seen above the map phase time was slower by 0.56 seconds, the group phase was significant slower (by 0.39). the reduce phase was 0. 046 seconds faster. Overall this approach was 0.34 seconds slower than approach 3 using the thread pooling.

**Testing Approach 3 without pool threading:**

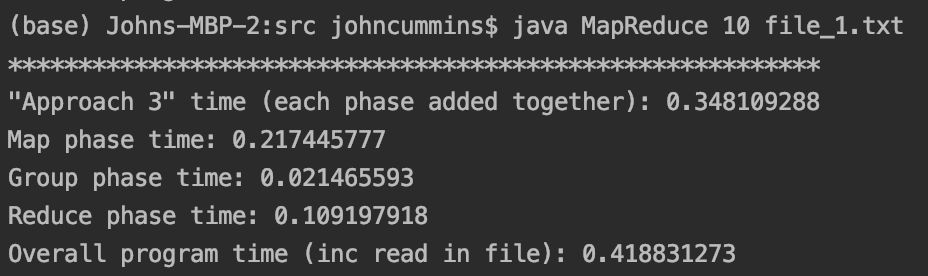
****

The screenshot above shows the results from running approach 3 without thread pooling. The overall result was significantly different, mainly due to the reduce phase which took 4.411 seconds slower to run. This resulted in the overall run time for this approach to be 4.436 seconds slower than using this approach **with** thread pooling.

**Testing varying quantities of text files**

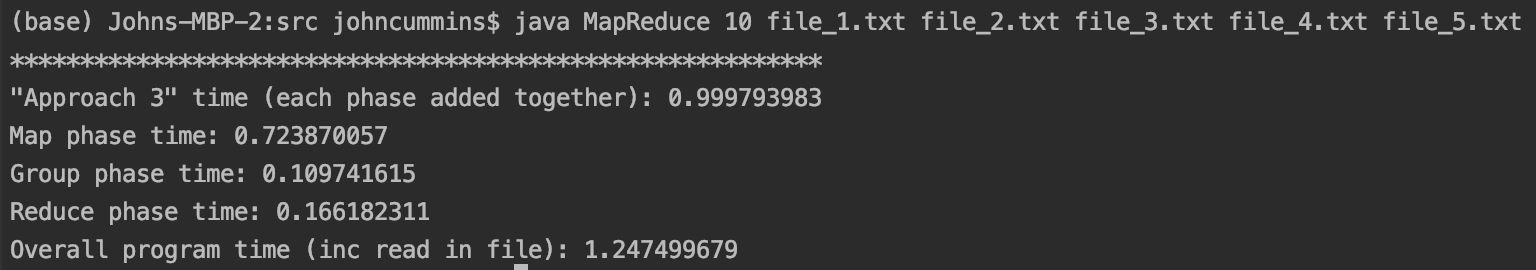
For this test I will conduct 3 tests, each with different number of text files passed in. The first test will have one text file of medium size. The second test will have 5 text files and the last test will have 10 text files. Testing conducted using approach 3 with thread pooling (number of threads set to 10 for all tests).

**1 text file:**



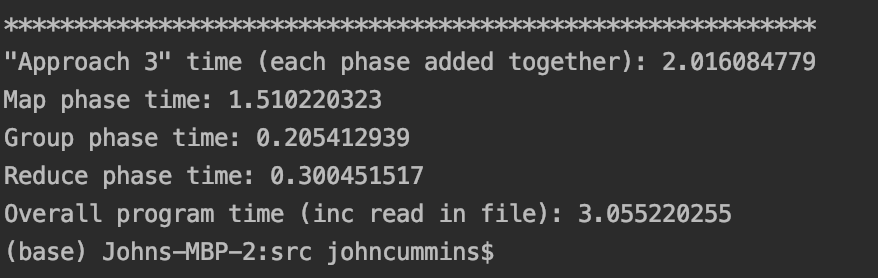
The screenshot above is the result from the test with only one text file passed in as an argument. This resulted in an overall run time of 0.348.

**5 text files:**

****

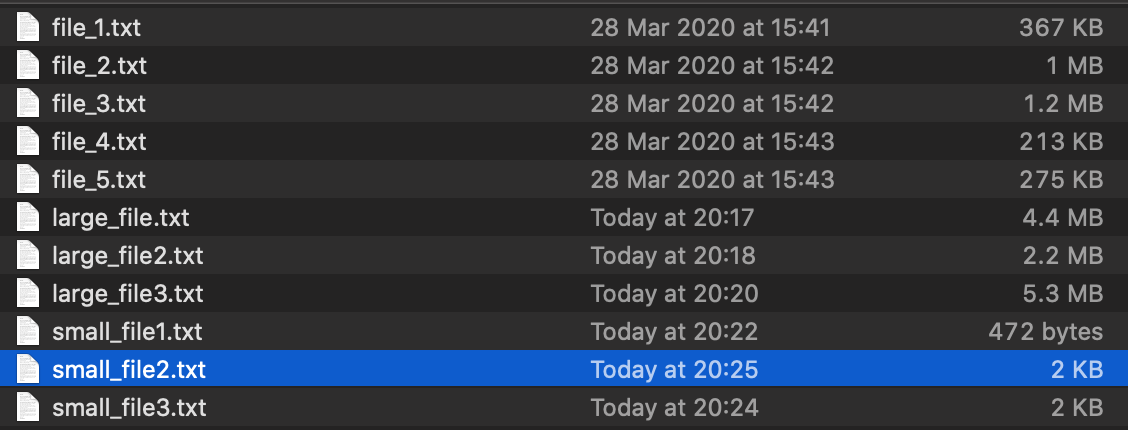
The above screenshot shows the results from passing 5 text files into the program. This resulted in an overall run time of 0.999. This was an increase of 0.651 seconds compared to the test with only one text file.

**10 text files:**

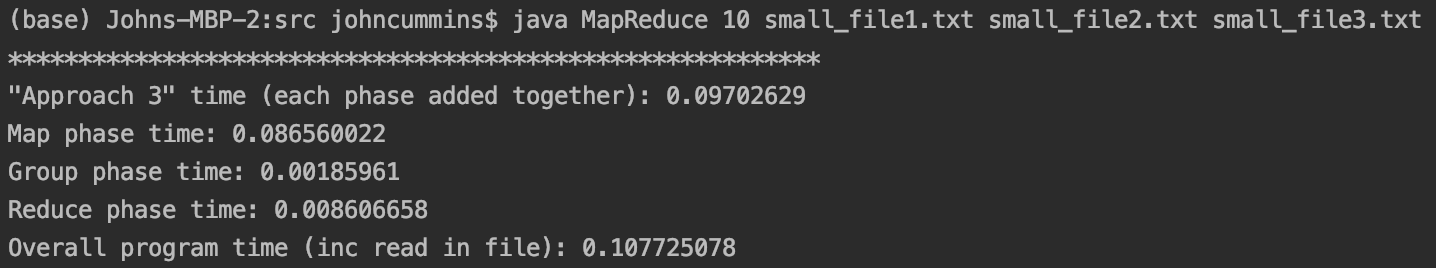
****

Above are the results from the test with the programming running with 10 text files passed in. This resulted in a overall run time of 2.0160. This was 1.668 seconds slower compared to the test with 1 text file and 1.017 seconds slower than the test involving 5 text files.

**Testing varying text files sizes**

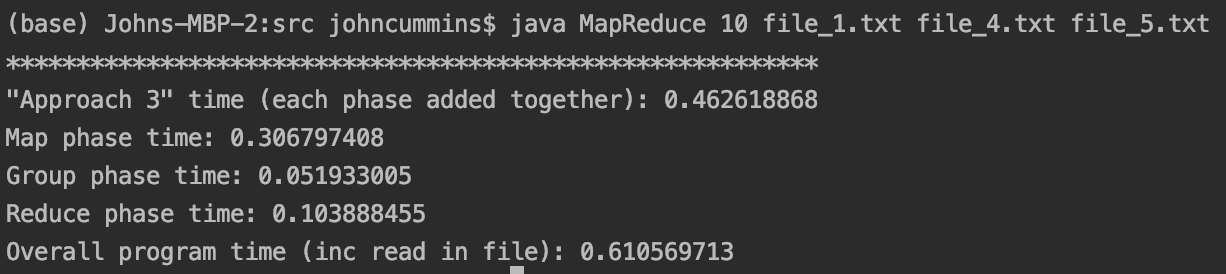
For the purpose of this test, I will compare the overall run time of each set of file sizes. I will use three small text files as shown in the screenshot below, rangin from 472 bytes to 2 KB. I will then run a test with 3 medium text files ranging from 213 KB to 367 KB. I will then run a final test on large text files, using three files ranging in sizes from 2.2 MB to 5.3 MB as shown below. The following tests were run with the Map and Reduce functions from approach 3 assigned to a Thread pool with the number of threads set to 10 (as this was the thread with the fastest time).

**3 Small text files:**



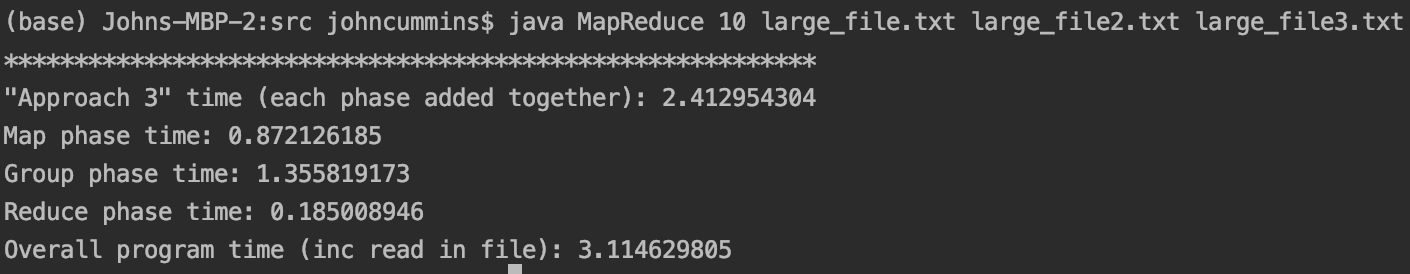
The overall run time was 0.097 with for an input of three small text files, as shown in the above screenshot.

**3 Medium text files:**



The overall run time was 0.462 with for an input of three medium text files, as shown in the above screenshot. This was slightly slower than the small file size run time as expected.

**3 Large text files:**

****

The overall run time was 2.41 with for an input of three large text files, as shown in the above screenshot. This was a good bit slower than both the small and medium size text files. This large text file test was 2.315 seconds slower compared to the small text files test and 1.95 seconds slower compared to the medium size test.