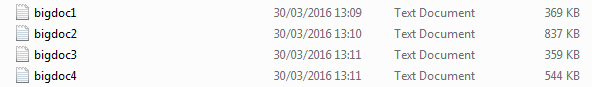
CT414 Assignment 2 MapReduce

Authors: Ultan Griffin (12450558) & Brendan Connolly (12370346)

The following text files were used for testing the map reduce program:



These text files were just big enough so that the program could have its efficiency tested but not so large as to take an unreasonable amount of time for the program to run. They consist of sections of text from norvig.com/**big**.**txt.**

# Part 1

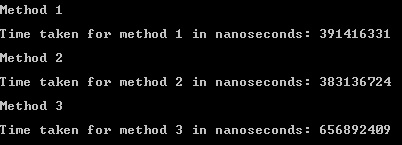
This part was relatively simple. Please refer to code supplied with submission.

# Part 2

The program was modified so that method 3 used a fixed thread pool to perform its operations rather than just 1 thread per input item. Command Used:

CommandP2.png

These are some sample results that were obtained (1-Brute Force, 2-Single Thread and 3-Distributed with fixedThreadPool and callbacks):



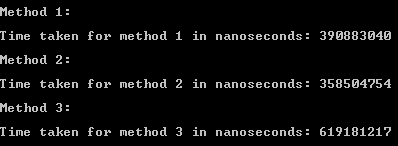
That’s a total of 0.39 seconds for method 1, 0.38 seconds for method 2 and 0.66 seconds for method 3. These results were obtained using 8 threads.

# Part 3

A new version of the program was made. This one used the thread safe CopyOnWriteArrayList<E> and ConcurrentHashMap<K,V> instead of the callback classes. Command Used:

CommandP3.png

Here are some sample results obtained (method 3 is now thread safe and not using callbacks, 1 and 2 are unchanged form part 2):



That’s a total of 0.39 seconds for method 1, 0.36 seconds for method 2 and 0.62 seconds for method 3. These results were obtained using 8 threads.

# Part 4

Which version was the most efficient?

The results from running the two method 3 codes (with and without callback) were inconsistent so we ran the programs 5 times and averaged the resultant times. Here is a table of timings:

|  |  |  |
| --- | --- | --- |
|  | With CallBacks | Without Callbacks |
| Run 1 | 656892409 | 619181217 |
| Run 2 | 1011024077 | 391202809 |
| Run 3 | 379394451 | 548589190 |
| Run 4 | 365287656 | 391658595 |
| Run 5 | 366666820 | 374709809 |
| Average | 555851082 | 465067924 |

As you can see, on average, the version that uses the thread safe data structures rather than callbacks is slightly quicker. This is to be expected as the callbacks add unnecessary complications and objects to the method.

On another note I also see that method 3 is often not the best method in terms of time. This is a surprising result because you would expect the concurrent method to be the fastest. This hints at the unpredictable nature of multi-threaded programming.

# Splitting The Work

Brendan handled parts 1 and 2 whilst Ultan did part 3, put the timing code in place and worked on the testing. We both worked on our respective parts of the write up.