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

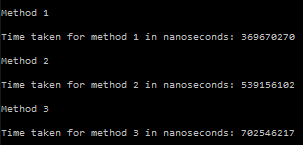
This part was relatively simple. Please refer to code supplied with submission.

# 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 the results that were obtained:



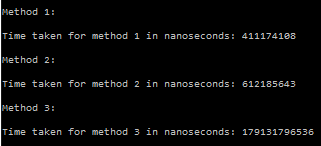
That’s a total of 0.37 seconds for part 1, 0.54 seconds for part 2 and 0.70 seconds for part 3. These results were obtained using 8 threads.

# 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 cal back classes. Command Used:

CommandP3.png

Here are the results obtained:



That’s a total of 0.41 seconds for part 1, 0.61 seconds for part 2 and a huge 179 seconds for part 3. These results were obtained using 8 threads.

# Part 4

Which version was the most efficient?

Clearly the version with the callbacks is far more efficient and provides much better performance. As to the reason for this, I assume that the CopyOnWriteArrayList is a very inefficient way of storing the mapped items. It means that there is a lot of copying to and from memory rather than simply writing once to memory when all threads have done their business. I think that these needless memory copies kill the performance to the extent of what we see in the results where the program slows down by a massive 178 seconds.

On another note I also see that method 3 is consistently the worst method in terms of time. This is a surprising result because you would expect the concurrent method to be the fastest. This leads me to speculate that the threads are not being run in parallel at all. As a result, the extra overhead in creating threads is being wasted because you are not getting any performance boost.