Ian Walsh

Prof. Bassem

Distributed Computing

4 February 2024

Assignment 1: Local WordCount

Objective: Create localized single and multi-threaded wordcount implementations in Go.

Synchronization Points:

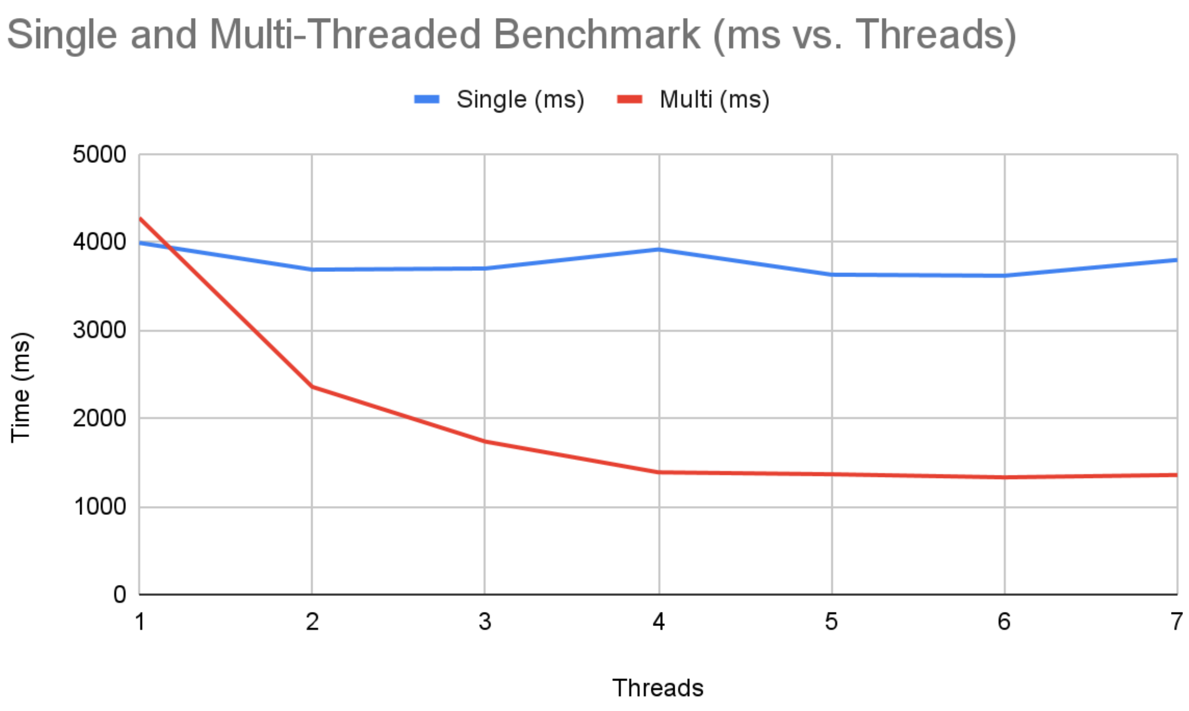
1. How many threads will you create? Does the number of threads need to be equal to the number of files?
   a. The number of threads is equal to the number of files, though a smarter implementation might break the file into chunks so that all processor cores are always running.
2. How to distribute the files over the different threads?
   a. In my implementation, threads are passed a single file path and return a hashmap of type string:int (word:count).
3. How will you compile the output into a single file at the end of the computation?
   a. The hashmaps returned from each thread are iterated through and compiled into one large hashmap. This may be the best implementation because it runs in O(n) time. The final large hashmap is again iterated over and the key/value pairs are written to disk.
4. Do you need to use any data structure to ease the computation? If yes, was it a shared resource? How can you guarantee mutual exclusion?
   a. Yes, a struct that kept track of the large global hashmap, a WaitGroup object, and the mutex used to guarantee mutual exclusion. The mutex was used only in the transfer of a thread-scoped hashmap that contained the wordcounts of a single file to allow for minimum blocking time.

Challenges/Solutions: Some challenges I faced were:

- Removing punctuation (I used Go’s regex engine, and had to re-learn those nutsy codes)

- Indexing function outputs when there were more than one variable. I had been adding keys instead of values when I did my multi-threaded recombination step (just took a bit of debug time)

Running times: I copied the big file 13 times and ran some timing tests (results shown below). I expected the multithreaded runtime to bottleneck at 8 threads (number of cores on my M1 Mac), but it did it at 4 - this leads me to believe that my code is severely limited by IO - that is, it takes roughly 25% of the time to copy data from a thread-scoped hash table to a global one. In that time, the main hashmap is unavailable for reads/writes from the other threads, and they are blocked. Yikes! How would we fix this? I'm not really sure. I'm also not really sure if this is actually what's going on. I've heard people use profilers, would this be a good place to use one?



Max Threads:	1	2	3	4	5	6	7
Single-Threaded (ms)	3990	3686	3699	3815	3629	3617	3796
Multi-Threaded (ms)	4273	2356	1735	1386	1364	1329	1356