Kate Chamberlin 19/10/2018
54384616

ENCE360 Assignment

Algorithm Analysis

The algorithm shown in lines 200-230 functions as follows:

```
For each line from the file:
        -Replace the newline character with an EOF character
        -Create a new task using that line, and append it to the queue
        -If there are enough tasks to keep each worker thread busy, process a single task
Once all lines are processed, continue with tasks until there are none left.
Once all tasks are complete, free all resources, and finish.
```

An improvement might be forking the process so that the parent processes the lines and stores them in a buffer while the child spawns the threads, so that the threads can immediately start processing the tasks rather than waiting for there to be enough tasks.

Another improvement might be closing the file as soon as it is no longer being used. This would free up memory but would possibly make line-processing smaller as it would introduce a check to the while loop.

Performance Analysis

For the performance analysis I increased the number of worker threads from one to 500 for each test file. This analysis was performed with my personal computer, which is a Ryzen 1500X and has four cores with eight threads. The results for this are shown in Table 1 and Figure 1. These results are dependent on several factors such as connection quality and processor availability.

From these results we can see that the file size does affect performance, as the download time increases as the file size increases. This also shows that download times were low around 5 threads, but then increased drastically between 10 and 20 threads before going down again. This might suggest that the optimal number of worker threads is 5 or 50 before rising again.

Kate Chamberlin
54384616

Table 1: Average results for time taken to download

	AVERAGE					
NUMBER OF THREADS	**Small**	**Test**	**Large**	**Small (Sample)**	**Test (Sample)**	**Large (Sample)**
1	29.80	0.41	34.56	26.23	0.57	33.95
5	12.06	0.25	22.65	14.93	0.38	19.89
10	13.33	0.25	35.23	17.77	0.31	29.31
20	12.44	0.27	37.48	13.67	0.45	26.90
50	12.17	0.27	22.36	15.04	0.91	21.60
100	11.12	0.27	23.86	13.78	0.56	24.17
150	12.96	0.29	23.98	15.35	0.37	26.51
500	11.83	0.30	24.61	17.51	0.33	24.59

As threads are increased past approximately 50, the download times rise again. This is due to the queuing (commonly known as bottlenecking). Furthermore, the size of the queue is dependent on the number of worker threads spawned, which means that more dynamic memory allocation is required. This can also lead so slower processing and thus longer download times.

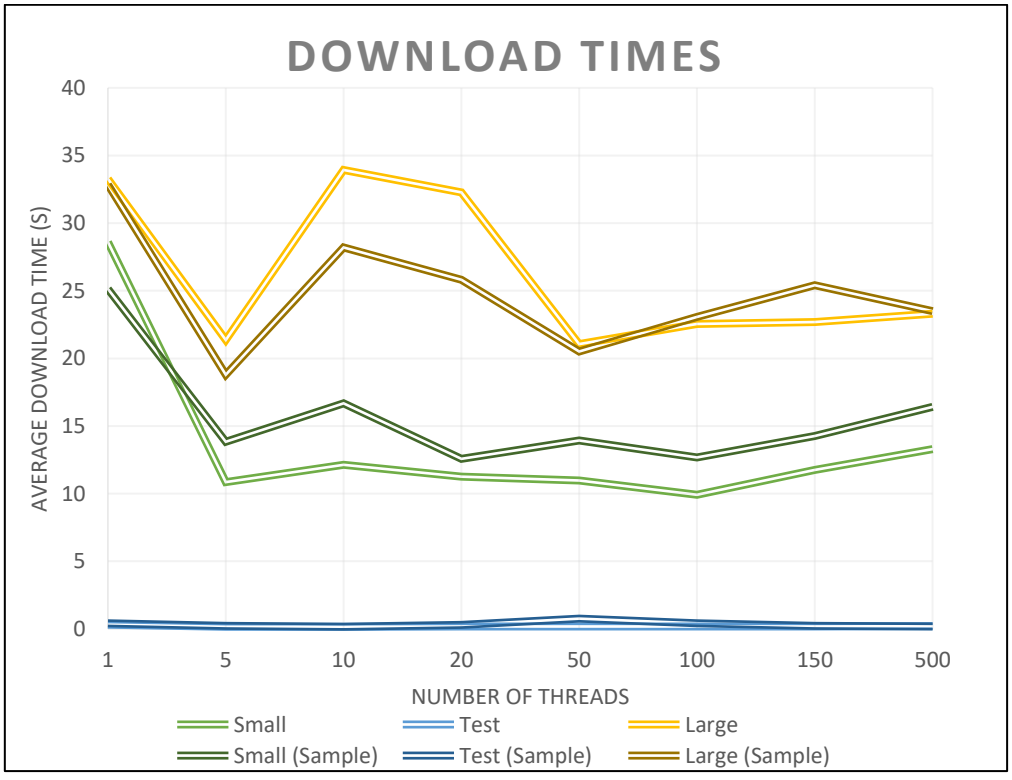

Figure 1: Graph of average download time for range of files and worker threads