ENCE360 Report

Name: Daniel mcinnes

ID: 11933946

16/10/2020

Algorithm analysis

Describe

The algorithm on lines 226-256 in the given downloader.c file starts by creating all the threads and the queue by calling spawn_worker(). Next it goes through all the link/lines in the given input file and replaces the end character with a null terminator. this line along with the number of workers is then passed into the function get_num_tasks(), this function calculates the number of tasks and the byte size of each task. This only returns the number of tasks hence the next line that calls the get_max_chunk_size() to get the chunk size of each task. Next is a for loop that creates the number of wanted tasks, each task has the download URL and the start and end its bytes to download. The tasks are also added to the queue. The strut holding the queue with all the tasks is added run through a while loop, this runs through all the tasks and waits for all of them do be done before moving on. Lastly the chucks are merged into the final file and the chunks are removed. This algorithm is similar to the Worker threads algorithm showed in the second lecture notes

Improvements

Having to loop through each chunk to merge and then loop through again is inefficient. A better choice would be to delete the chunks as they are merged or by using a temp folder to store the chunks and removing the folder if this can be done in a single remove call. Another option is to add a thread safe method of merging a chunk into the final file in the same thread that was used to download it, this would remove the need for separate merge loop.

Another improvement could be to add a way to resume a download after the likes a internet crash, this could be done by storing what full files and chunk have been downloaded so the download can be resumed

Performance

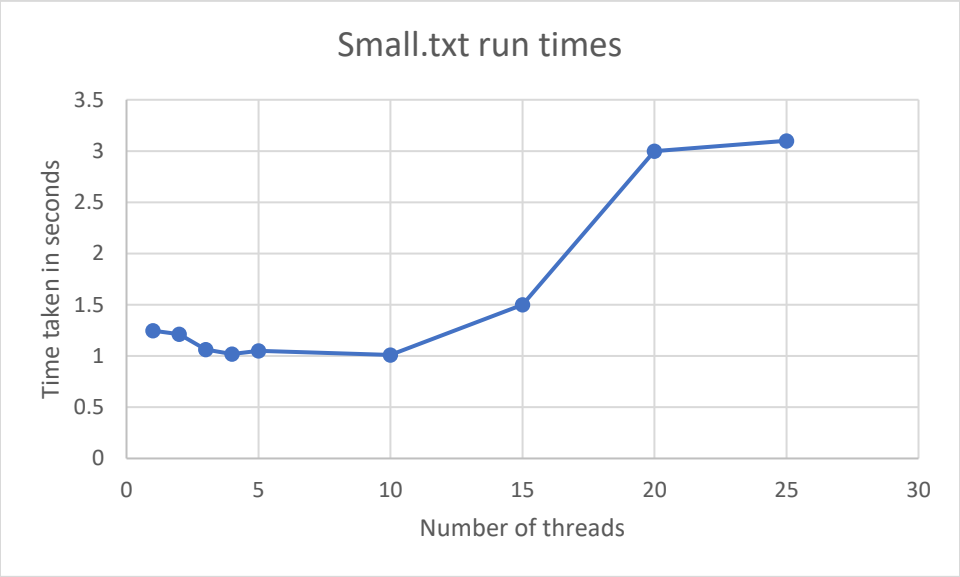

Figure 1: the run time of Small.txt with my download program

Table 1: table of threads to run time

threads	time
1	1.248
2	1.214
3	1.063
4	1.019
5	1.05
10	1.01
15	1.5
20	3
25	3.1

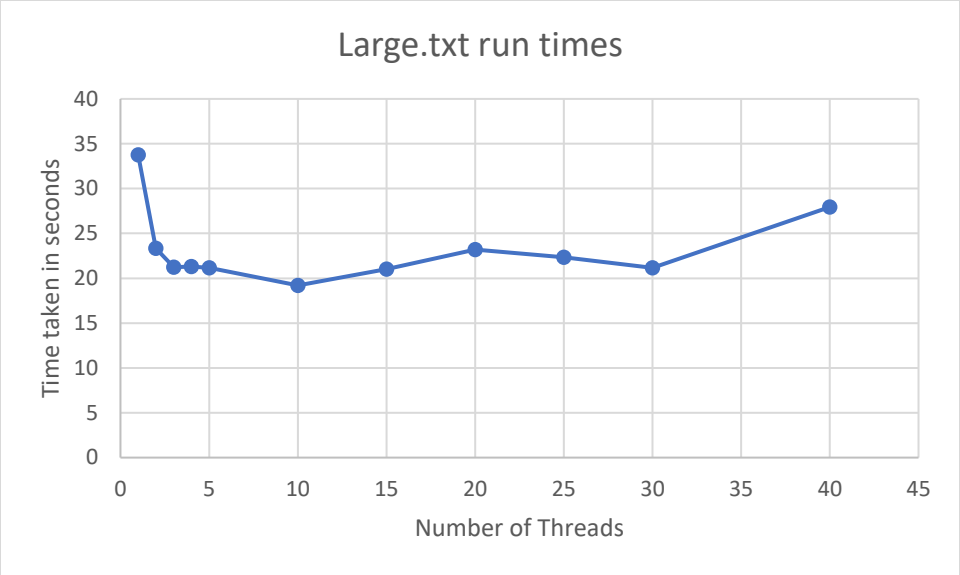

Figure 2: the run time of large.txt with my download program

Table 2: table of threads to run times

Threads	time
1	33.76
2	23.35
3	21.25
4	21.32
5	21.15
10	19.2
15	21.02
20	23.2
25	22.34
30	21.16
40	27.93

This Data was gathered using the time call in the Linux terminal (EG "time ./download.txt 15 download"). A ubuntu VM using 12 cores 12 threads was used to run this test

For the small.txt it appears that the best worker size is 10 but this not much better than 5. For the Large.txt it looks like 10 is the better worker count but from my testing 15 workers would sometimes be faster but I think this depended on how fast the server I was requesting from could respond. But I have no way of testing this. When testing large amounts of workers, I ran into errors where the program would hang and then time out with "ERROR opening socket" or just not say anything at all. This might be a problem in the queue or the http function but as time was running out, I chose not to investigate this.

I found that when downloading small files that using high threads cause an increase in download time, my guess is that the chunk file read and write times had a big negative Impact, larger files had less of a problem with this but there was still a point were more workers had to much overhead. There are a few ways to improve this, the first on is what was said early and that is to merge files as downloaded but this could cause locking problems. But other options could be to download whole files but have the workers download several files at once or even a mix of whole files and chunks.

My code had similar times compared to the given download program even at high worker number, but the big difference is that the given download program is more stable than mine with higher worker numbers. But as the size of workers increased the range of results, I would get for each worker count would get larger and larger. Making it harder and harder to get an actual number. Again, I think this is because of their server's response times

As said when it comes to the file merge and delete it would be faster to delete the chunks as soon as they are merged compared to the current implantation of looping through once to merge then again to remove.