Luke Grammer

CSCE-313

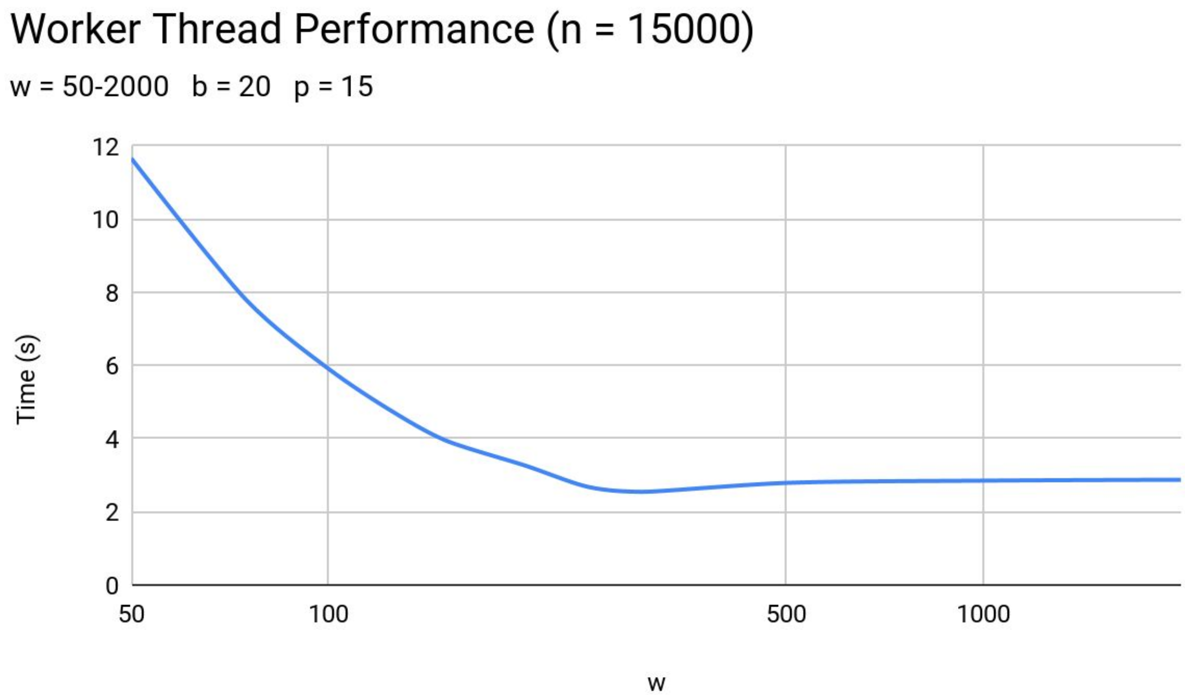
3/31/19

Programming Assignment 4 Analysis - Threading and Synchronization

Graph Analysis:

For runtime analysis of the multithreaded client I have included four graphs. Two of the graphs show client performance for data requests and two show performance for file requests. Of the graphs for data requests, one shows the performance of the client with a varying number of worker threads w in the range [50-2000] and fixed request buffer size b = 20 bytes for n = 15000 data requests. The other graph for data requests shows varying request buffer size b in the range [1-1000] with fixed number of workers w = 500 for n = 15000 data requests. The first graph for file requests shows the performance of the client with a varying number of worker threads w in the range [50-2000] and fixed buffer capacity m = 256 bytes for a 500MB binary file. The other graph for file requests shows performance with varying buffer capacity m in the range [1-200] with fixed number of workers w = 500 for a 5MB binary file.

Data requests graph 1: Varying number of worker threads [50-2000]

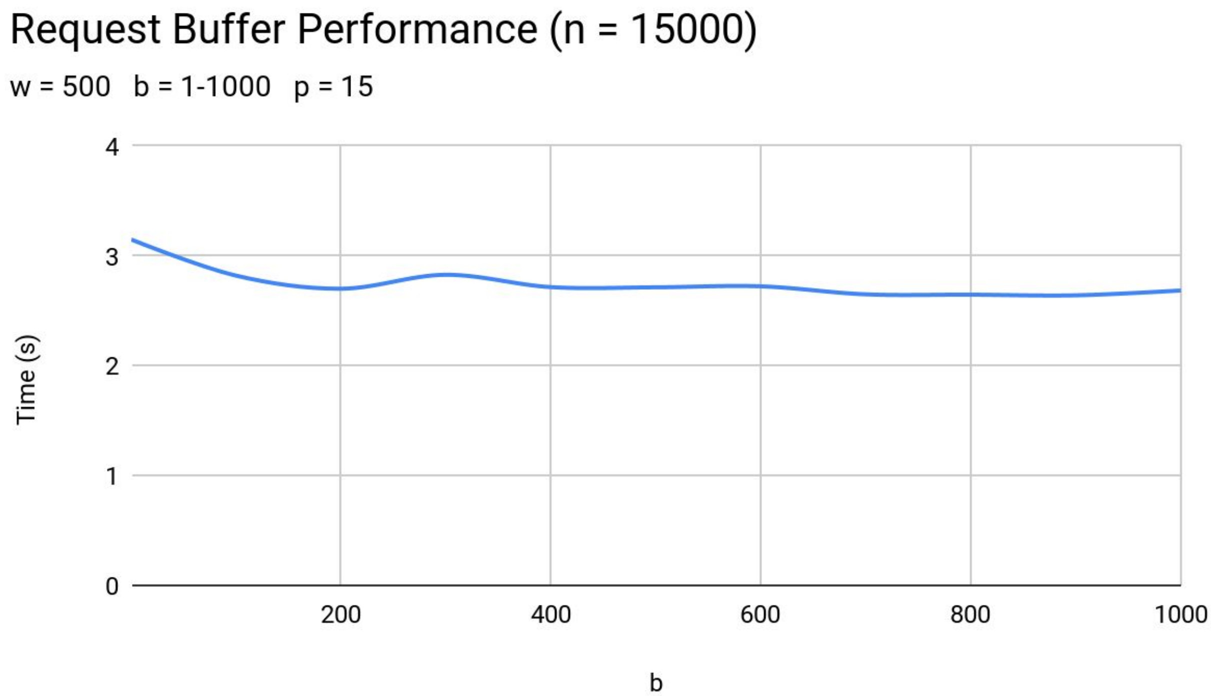


As you can see in the graph, increasing the number of worker threads drastically improves performance until around $w = 300$. The graph is non-linear due to the rapid fluctuations in the data when w is between 50-200. The performance cap when $w > 300$ is likely due to the fact that the overhead of creating a worker is constant, but the amount of data that each worker processes decreases with every additional worker. Too many workers will take a lot of time to manage, but will not significantly contribute to the data processing.

Table 1:

w	Time (seconds)
50	11.662
75	7.779
100	5.91
125	4.749
150	3.971
200	3.259
250	2.675
300	2.546
500	2.79
1000	2.852
2000	2.876

Data requests graph 2: Varying request buffer size [1-1000]

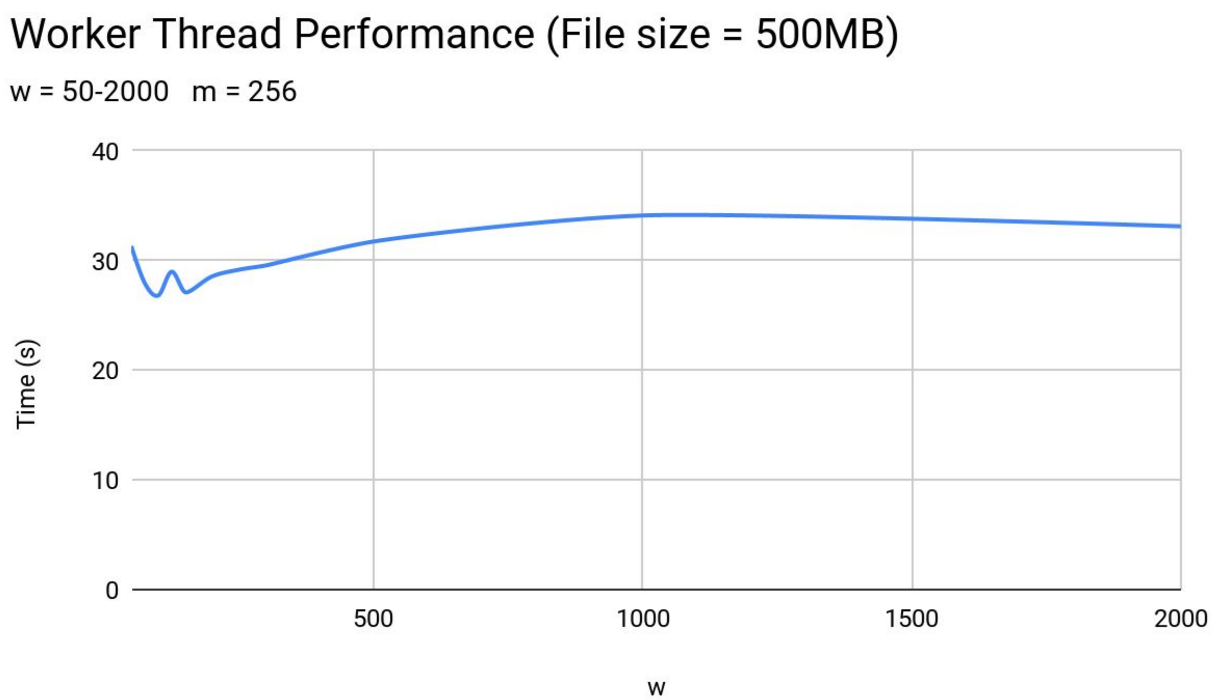


Size of the request buffer has no significant impact on client performance since the worker threads are able to request data faster than the client threads are able to insert it into the buffer. Therefore the performance impact scales linearly since the request buffer spends most of its time empty.

Table 2:

b	Time (seconds)
1	3.149
100	2.822
200	2.701
300	2.828
400	2.716
500	2.714
600	2.723
700	2.65
800	2.647
900	2.641
1000	2.685

File request graph 1: Varying number of worker threads [50-2000]

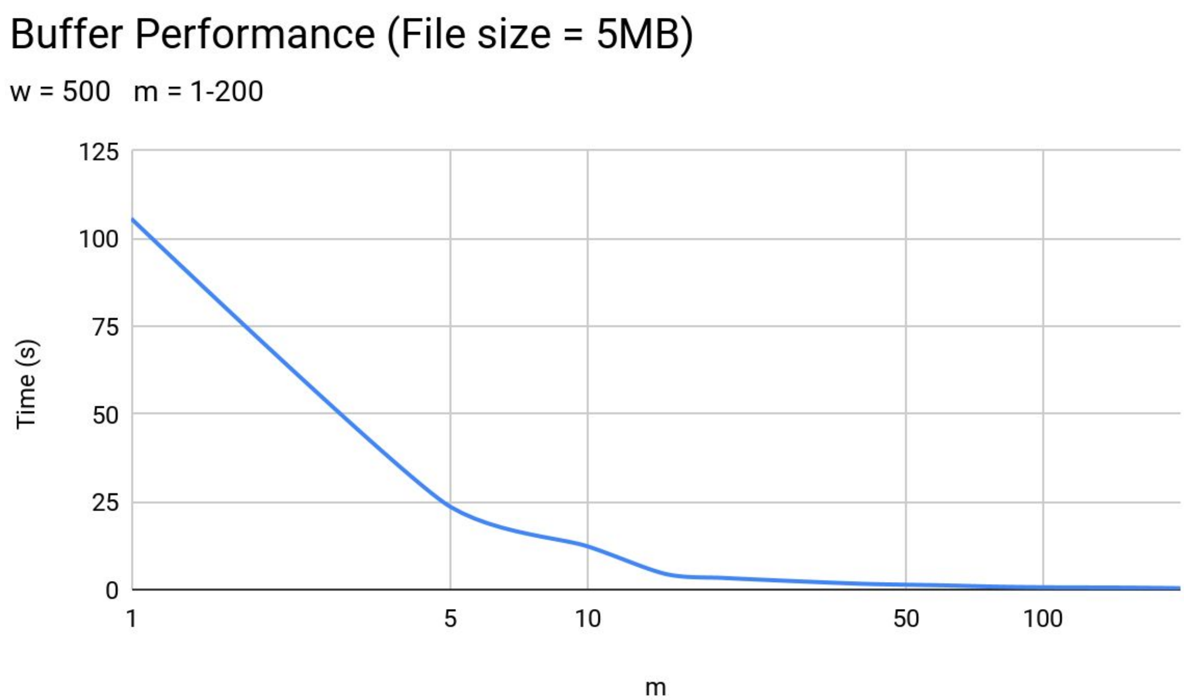


For file transfers, the time required for file transfer was not significantly affected by the number of worker threads. Even though each worker thread writes to it's own copy of the file and does not perform the write in a critical section, it seems as though the pwrite() function only returns once the I/O device has unblocked. In this scenario, even though multiple threads may be attempting to write to the file simultaneously, only one thread can be physically writing to the disk at any given time.

Table 3:

w	Time (seconds)
50	31.317
75	27.949
100	26.804
125	28.981
150	27.11
200	28.547
250	29.161
300	29.549
500	31.731
1000	34.107
2000	33.119

File request graph 2: Varying buffer size [1-200]



The buffer size had a large impact on file transfer speed. The graph is non-linear due to the rapid fluctuations in the data when *w* is between 1-10. The speed difference is due to the fact for a small buffer, each request from the server can only transfer a small number of bytes. As the buffer size increases, more information can be received with each request which improves transfer speed drastically.

Table 4:

m	Time (seconds)
1	105.698
5	23.653
10	12.348
15	4.328
20	3.295
40	1.65
60	1.204
80	0.817
100	0.654
150	0.556
200	0.398