PA3 Report

Tom King

Notes

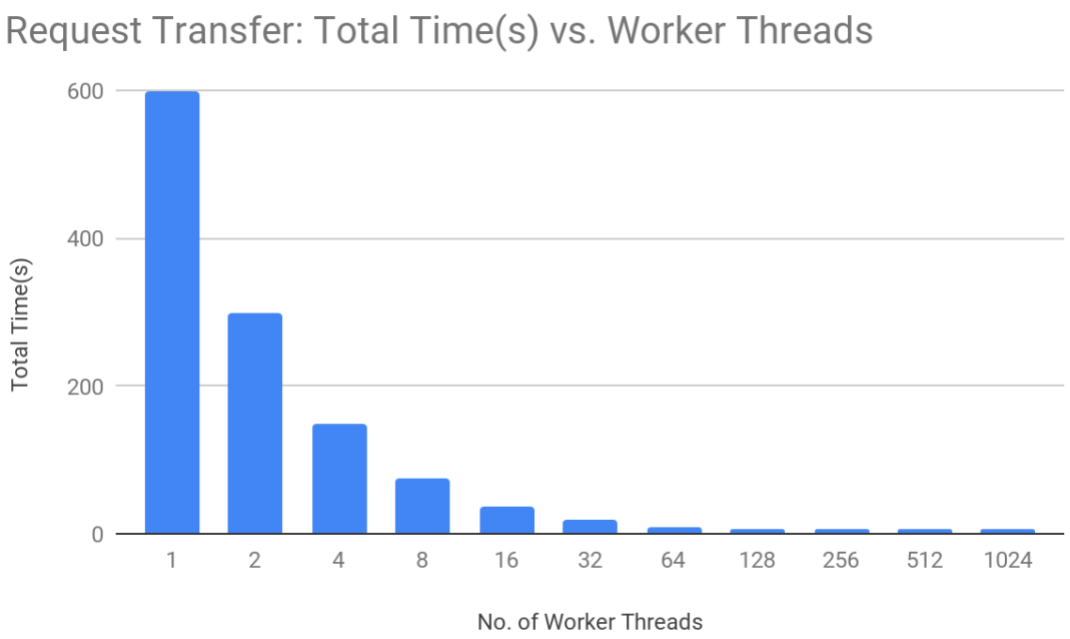
- The program uses std::mutex and condition variable for bounded buffer and I was also going to use std::thread for threading but I saw the piazza post about using pthreads if the class was taken before and so I used it
- I've completed the bonus assignment with a real time histogram and file copy rate view
- The program structure is fairly simple to follow with 3 structs for the three types of threads and accompanying functions. I load variables into them using a pthread and use an array to join all the threads at the end. The bounded buffer uses std mutex variables to function. I've rigor tested it in another program but you can see the results by commenting in the print statements to make sure it's working
- I left the real time histogram on for timing tests. It refreshes every 1 second
- All testing was done on amazon aws ec2 instance with modified ulimit to 8192
- I've only attached a 1MB file in BIMDC because of space issues but I used other files for testing

Charts

Request Transfer Chart 1 - Total Time vs Worker Thread Count

Holding n constant at 15000 and bounded buffer size constant at 50, the following results were recorded.

w	Total Time(s)
1	598.707375
2	298.67322
4	149.043011
8	74.331679
16	37.092575
32	18.585955
64	9.36542
128	5.188222
256	5.7631
512	6.030784
1024	6.622241

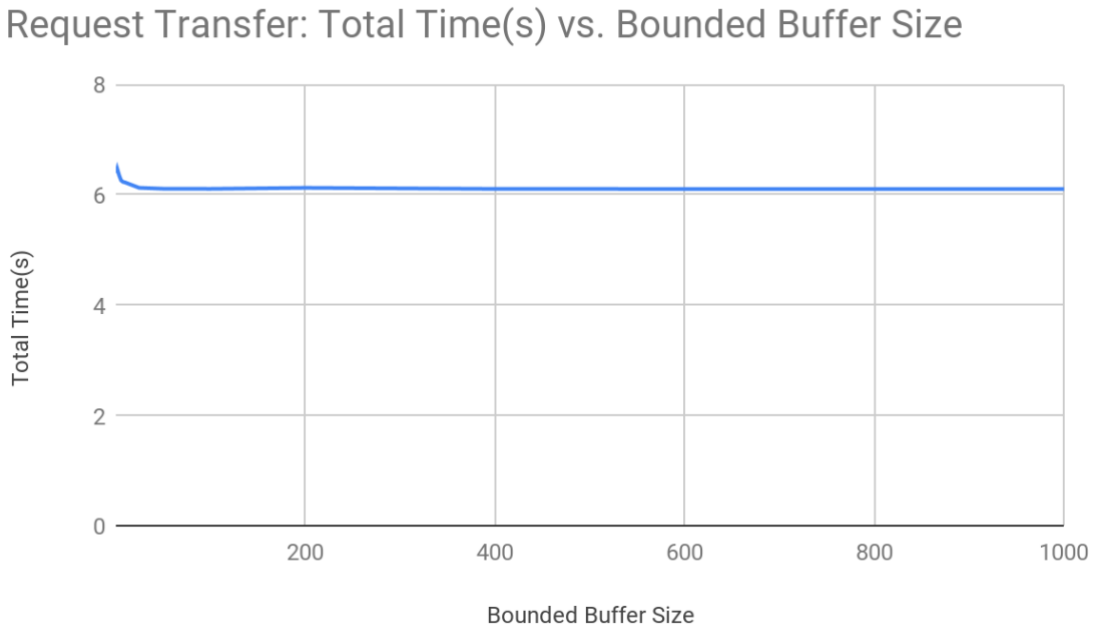


The results show time linearly decreasing up to a value w > 64 after which the slope stabilizes with small increases in time every time the worker count is doubled. This is because at the start the buffer size is 50 and because of the 15 patients, there is blocking on push requests (worker threads take longer than patient threads because of the delay in the dataserver). At some point greater than w > 64 this ratio turns around and you spend more time waiting to pop data points and also more time to quit all the threads. Thereby explaining the data.

Request Transfer Chart 2 - Total Time vs Bounded Buffer Size

Holding n constant at 15000 and worker thread count constant at 100, the following results were recorded.

b	Total Time(s)
1	6.39053
2	6.464669
4	6.34894
6	6.259885
8	6.232871
10	6.224227
25	6.126013
50	6.109387
100	6.107296
200	6.125392
400	6.106171
600	6.104844
800	6.104359
1000	6.103274

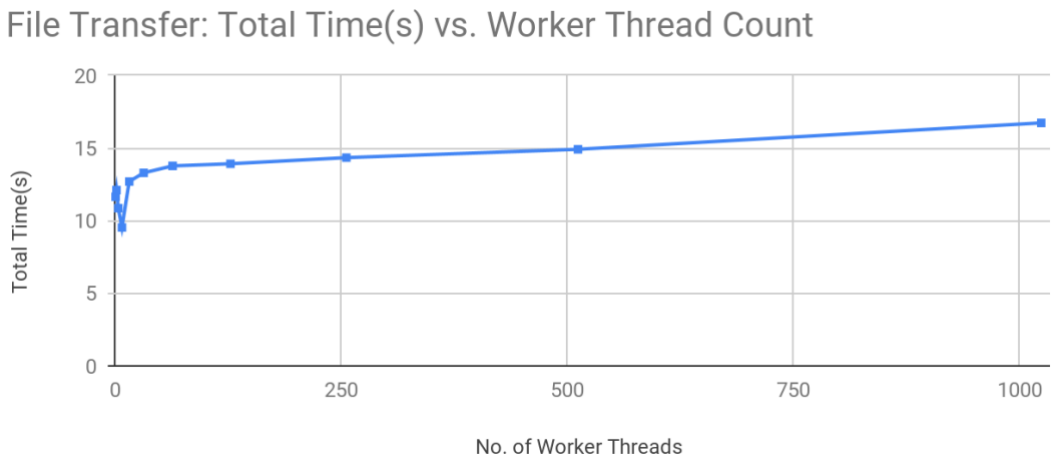


The important thing to notice on this chart is that the time stabilizes around a bounded buffer size of 25. This means that after this point, the gains/losses are marginal. This is because there aren't any more push blocks like there were prior this. The 15 patient threads are keeping the 100 worker threads occupied at a bounded buffer size of 25. Before this, the marginal gains going from 1 to 10 were because the patient threads were being blocked on the push requests owing to the worker threads not being able to empty it quick enough.

File Transfer Chart 1 - Total Time vs Worker Thread Count

Holding n constant at 15000, bounded buffer size constant at 20 and buffer size constant at 256 bytes, these results were recorded on a 100MB dat file created by truncate.

w	Total Time(s)
1	11.639414
2	12.099371
4	10.85536
8	9.525544
16	12.69047
32	13.288
64	13.767755
128	13.914989
256	14.335719
512	14.910785
1024	16.728923



Contrary to expectations this chart shows an odd curve slowing down at first but then steadily increasing. This can be explained because of the IO operation bottleneck. Specifically, the seeking operation which is repeated every time because the file is always opened at the start. This overhead and the increasing overhead of lock/unlocking and quitting increases with more threads.

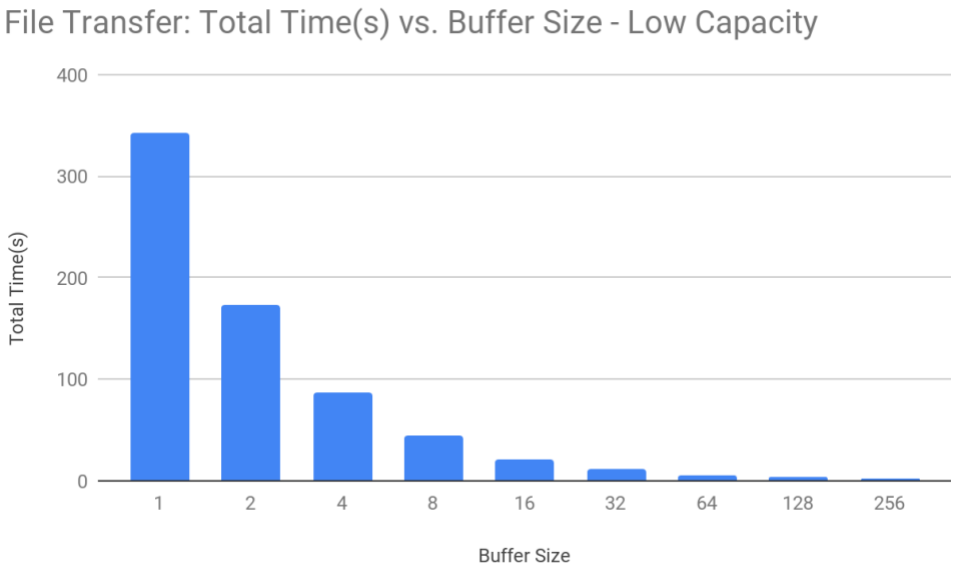
The initial drop in time can be explained by the balance factor being somewhere around 8. At this value, the overhead of the threads is discounted by the increased performance gotten by not letting IO operation time go to waste. But this increased performance from maximizing IO usage is soon overshadowed by the overhead of more threads as the thread count increases.

A far better implementation of this would be to segment the file into certain sections and having a thread take care of each section avoiding the seek overhead.

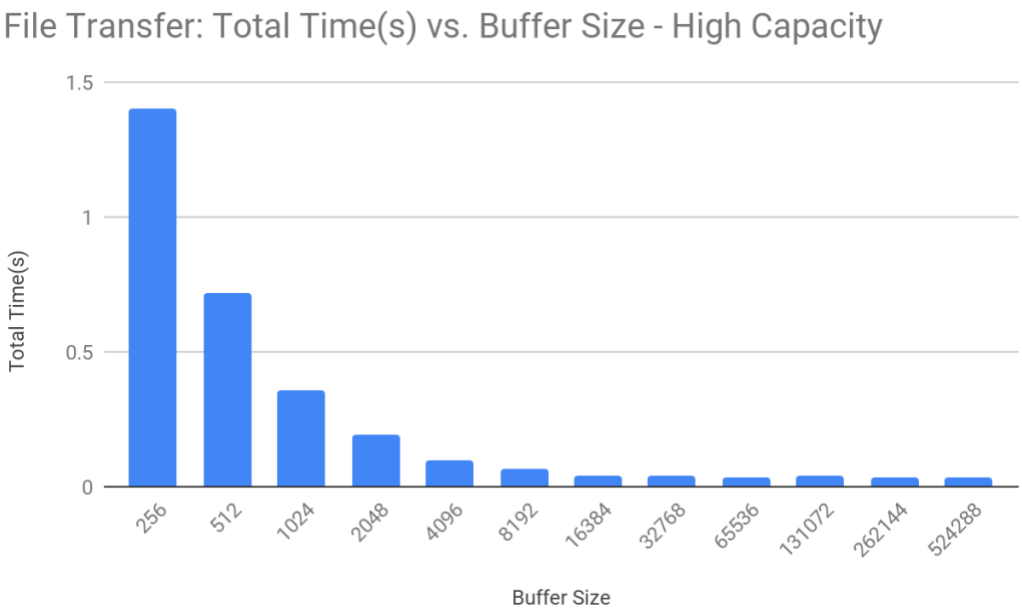
File Transfer Chart 2 - Total Time vs Buffer Size

Holding n constant at 15000, bounded buffer size constant at 20 and worker thread count constant at 100, these results were recorded on a 10MB dat file created by truncate.

m	Total Time(s)
1	342.15264
2	172.286039
4	86.041171
8	44.018123
16	21.084837
32	11.024785
64	5.467533
128	2.903789
256	1.398117



m	Total Time(s)
256	1.398117
512	0.719843
1024	0.35646
2048	0.194778
4096	0.098761
8192	0.066609
16384	0.043743
32768	0.0384
65536	0.037158
131072	0.037889
262144	0.036048
524288	0.031894



As you can see time decreases almost perfectly linearly with buffer size at low capacity here defined as less than 256 bytes. Considering high capacity, time continue to decrease linearly up to 8192 bytes at which point the time slows down in reduction. This is because the overhead of thread locking/unlocking with a lot of pop blocks in the bounded buffer hold it back. But despite this, time continues to decrease because increasing buffer size means fewer messages to send and that has a much higher impact.