Michael Laucella
Multicore programming
Lab 3

Analytics Report

**note - all graphs are located in the analyze folder just trail down to the respective folders
- for some reason the allRun graph had some kind of interference and dragged one of the lines far to the right so instead of starting at 0 it starts at about 200k, also don't know why the plot is coming out that way the individual graphs tell a different story as none of them start at 0 and curve. They are the exact same data points I just used all of them in one graph so I found it strange the way it came out, look at the individual total graphs for a clearer picture.
- I used openmp wall clock for the timing the reason for this is if you look at the individual stats.csv you will see almost no time, this is likely due to the cryptopp library making use of some other io device like a gpu and therefore not adding to the cpu time

For the mean, median, max, min: you can run the program make sure to compile with the ST=1 flag, but to make life easier I also had python analyze my output and put all the stats from every run in one file stats.txt in the analyze folder. The order is the same as the file array in stats.py

From analyzing this file we notice first the lowest average is from one thread running the reader writer kv store using the regular listener queue (I also used a lock free queue). Obviously enough it also contains the lowest median. The reason the 1 thread most likely has a better average time is because it doesn't have the problem of waiting, what I mean by this is that if there is a writer it is blocked until the readers currently in the kv store exit this would increase the time for the server response as the number of threads grow. Now you would expect there would be a caveat here although 1 thread performs the individual server calculations (lower latency) its throughput, the amount it can serve in a period of time is lower than the higher threaded times, however for 1000 sessions with 100 requests each the finish time was 37s on 1 thread compared to 35s on 32. The threads between fluctuate some are higher and some lower than 1 and 32 oddly enough. Potential reasoning behind this could be contention on the system, inferior algorithms for instance the queue class being used here has an enqueue and dequeue lock and also a semaphore which could be removed if a sentinel node model was used instead. Building on that point the lock free structure although having horrible performance was getting better and better ad the threads increased both the client and server times were decreasing at a noticeable rate, likely due to the fact that each thread can perform some operation and is always either grabbing from, inserting into, or fixing the queue and CAS operations are themselves cheaper than mutex operations (in terms of time). This shows the impact the queue itself has on the run times observed. Also form observing the graphs there are many noticeable patterns within them where we can see how some outside influence such as other users on the machine may have drained performance, towards the end of chart 1 with regular queue and read write

locks, there is a spike from server time going from almost nothing to the same as that seen in 32. And these random spikes occur quite often and randomly some lasting longer than others. Another interesting pattern when viewing the stats.txt file the global locks for the most part offer very little to the time increase than the reader write locks, this is very strange as I noted to earlier the CPU time clocks were practically 0 for most requests, whereas on my system (running earlier tests the graphs should be in the older commits) there is noticeable CPU differences. This leads me to think that the kv store itself doesn't cause the majority of the time differences, the queue is obviously having an impact as can be seen by viewing the lock frees graphs, I'm assuming though that also having to run more concurrent MD5's or sending and receiving connections may have played a role in the time shifts as well. This theory is further backed up by the pie charts there are two (in every folder) but within it we see that the vast majority of requests are gets but even with that they all generally took about the same time (in terms of the wall clock) (also confirmed by the overlapping charts of each types request times). As for the client side there's also the reality that the connections themselves are still bottlenecked at the very beginning they come in through the main thread meaning they face a period of serialization, and on top of that there is also the fact that they are highly reliant on the transfer speed of the network and the contention that may be on it.

--for times sake im excluding lock free queue with global lock
--server stats

Regular Queue, with global lock KV

Threads	Average	Max	Min	Median
1	.01885205	.067116	8.9e-05	.018538
2	.01921910	.05592	8.9e-05	.018898
4	.01908297	.06061	8.9e-05	.018708
8	.01877035	.063985	8.8e-05	.018447
16	.01842874	.068069	8.7e-05	.018108
32	.01915216	.067748	9.4e-05	.018782

Regular Queue, with reader/writer KV

Threads	Average	Max	Min	Median
1	.00638408	.051658	0	0
2	.01971801	.061877	8.7e-05	.019335
4	.01917270	.072023	8.7e-05	.018861

8	.01853583	.05437	8.7e-05	.01823
16	.01920072	.062036	9.9e-05	.018847
32	.01815809	.05257	9.9e-05	.0178675

--note that lock free queue has a 250ms sleep if queue is empty which may have impacted the max times for all the queues as they all see to be around the same time

Lock Free Queue, with reader/writer KV

Threads	Average	Max	Min	Median
1	.26668502	.319109	8.6e-05	.268402
2	.14548477	.289631	.00011	.019335
4	.07611344	.250325	9.5e-05	.066693
8	.05337333	.241162	9.3e-05	.0450055
16	.03728912	.239835	.000102	.03357
32	.03007656	.238395	9.9e-05	.027531

--client stats

Regular Queue, with reader/writer KV

Threads	Time (s)	Max (s)	Min (s)	Median (s)
1	37.090	37.1	21.7	36.5
2	38.985	62	22.1	38.5
4	37.983	72.4	22.9	37.5
8	36.7	54.6	24	36.5
16	37.985	62.3	22.3	38.5
32	35.889	52.8	21.9	35.5

Global Lock Queue, with reader/writer KV

Threads	Time (s)	Max (s)	Min (s)	Median (s)
1	37.347	67.3	23.8	37.5

2	38.046	56.6	25.3	38.5
4	37.776	60.8	22.5	37.5
8	37.132	64.2	22.4	36.5
16	36.458	66	23.1	36.5
32	37.848	67.7	22.2	37.5