Sheffield Hallam University

An Overview and Analysis of the Delivered Server

Concurrent and Parallel Systems Report

Aiden Moncavage

Computer IP: 10.72.85.4[12]

Terminology

Rqs/s - Requests per second

5-5, 4-1, X-Y (etc) - Regarding testing: Number of Writers tested with Number of Readers

Throughput Experiments Evaluation

For testing my server I had utilized Sergio's Reference Client for all of my throughput tests to produce results similar to those that will be produced to evaluate my grade. I want to make it very clear that I used Sergio's Reference Client against my implementation of the server. First let's take a look at the high throughput tests that I've conducted. 5 readers and 5 writers were utilized to produce these result.[1] Over the course of the 15 tests I've run there's a lot of variance in average requests per second that my server was able to handle; however, with that said the variance is somewhat consistent leading me to believe that the cause of the variance is a Window's task management related issue. With my upper results I averaged around 9830.5 requests per second (rqs/s) with my best being 10342 rqs/s and for my lower results I averaged 8500.5 rqs/s with the worst being 8315.63 rqs/s. Throughout all of my tests performed I only see this split of upper and lower results with the high throughput tests of 5 writers and 5 readers; whenever I performed tests with 4 writers and 5 readers[2] and tests of 5 writers and 4 readers[3] there may be one or two outliers but nothing that was constantly producing either an "upper result" or a "lower result" that seems to be the case of the 5-5 testing. Looking through all of the tests performed, specifically the chat of averages,[4] it's not extremely clear whether the server inherently has a read or write priority. Taking a closer look at the results produced it does seem overall that in instances when the system is handling a lower number of writers to that of readers or vice versa that the lower number of readers/writers is not what is prioritized, with the exception being 1-5. All of the lower throughput testing that was performed was extremely consistent.[10,11] As previously stated there's very little variation within the results produced throughout the lower or standard throughput results; those being 1-1, 1-2 up until 3-2 and 3-3 tests. For all of these standard tests there's little variation between the averages and the individual results, they're all within a standard deviation. My server seems to be able to consistently handle anything that is equal to or less than double the number of cores the CPU has, meaning that as long as the total number of writers and readers is less than 8 on a 4 core CPU the results will remain constant when testing my server against Sergio's reference client. However, my server has much more consistency when running my own test harness against it.

Evaluation & Comparison of Test Harness and Reference Client

I ran tests to compare my test harness against my own server and against Sergio's server. While I did only record 6 tests, 3 for Sergio's server and 3 for my server, I was running tests in the lab earlier to ensure that my test harness was working properly so I feel these 6 tests accurately reflect the data. Both servers were able to consistently average over 10,500 rqs/s, sometimes nearing 11,000 rqs/s. I would also like to say that previously I was achieving 20,000 rqs/s by pairing my test harness with my server, but while I actually recorded my results that didn't seem to be the case; I know Mike saw that it was averaging

20,000 rqs/s. I've also heard students say that they've gotten different results on different days of the week, so potentially that is the case here, nonetheless I'm unbothered with results. I was able to achieve such high results when testing the test harness against my own server because of how I'm able to create post and read requests. I do so in an extremely simple manner to avoid putting unnecessary processing on the CPU. The code between the readers and writers are entirely the same except for the line of the request where the string sent changes the first 4 characters from "post" to "read" and vice versa. When a thread is created, it generates 2 ints for sending/reading messages: a topic id and a message. After creating TCPClient and connecting to the server it enters into a 10 second loop that sends post/read requests in the format of "{read/post}@{topic id}#{message}++" where for the first poster, poster 0, it's first messages will be "post@0#1" "post@0#2" "post@0#3" up until 5000 messages in a topic where it will increment the topic id, the next messages being "post@1#5001" "post@1#5002" and so on until the 10 seconds pass. To avoid writing over preexisting data, the second poster, poster 1, will start at topic id 1000 so that one poster thread will have to reach 50,000 messages before moving onto a topic that's already existed. All readers and writers also have a 10 int array which counts how many requests are sent per second and an incrementing access int; every second that passes the access inc increments and the message count increments in the next array position.

Potential Server Improvements

While only formally documented on my home system with a 8 core CPU (although testing in the lab yielding the same results, just not documented), according to the CPU profiler in Visual Studio Community 2019 the server tends to spend more time in the shared mutex lock of the read function than the writers' individual mutex lock of the data, or anywhere else in system, excluding Sergio's TCP connection code. During testing that I performed at my home system (with an 8 core CPU) I averaged 36,283.3 rqs/s with this shared mutex in release mode.[5] Upon testing the removal of the shared mutex lock on the readers in debug release and running in debug mode, the read requests went from averaging 105412 requests per reader thread with the server averaging 9651 rqs/s to the server averaging 151501 requests per reader thread with the server averaging 10981.7 rqs/s against Sergio's Reference Client.[6,7] However the total post requests ended up decreasing significantly: dropping from 438089 total post requests to 340669 total post requests producing a clear preference for read requests over post requests whereas the submitted server has no clear preference. I do not believe you would need the shared mutex to ensure that readers are reading accurate data as anytime the server's message storage is modified by a poster, the poster locks the mutex, thereby preventing other threads from accessing the data. If the approximate 12% boost in requests per second is to be accurate then my server could potentially go from handling on average 9121.23 rqs/s to 10,215.76 rqs/s thereby potentially yielding faster results than Sergio's own Reference Server when tested in the lab.

Appendix

1

High throughput testing results done in the lab (5-5)

	Writers	Readers	Avg Post Rqst	Avg Read Rqst	Avg Rqst / Sec	
1	5	5	87941	100732	9433.3	
2	5	5	86650	86740	8669.49	
3	5	5	86924	86968	8694.62	
4	5	5	87502	80958	8422.92	
5	5	5	98688	108177	10342	
6	5	5	87333	87411	8735.65	
7	5	5	79784	86529	8315.63	AVG of BEST
8	5	5	100416	100984	10069.9	9830.531429
9	5	5	100588	88492	9453.91	
10	5	5	105049	83214	9413.17	
11	5	5	95238	104772	9997.54	AVG of WORST
12	5	5	80398	87258	8382.76	8500.595
13	5	5	86985	80194	8358.7	
14	5	5	80811	87689	8424.99	
15	5	5	100803	101276	10103.9	
AVERAGES	**5**	**5**	**91007.33333**	**91426.26667**	**9121.232**	

2

High throughput testing results done in the lab (4-5)

	Writers	Readers	Avg Post Rqst	Avg Read Rqst	Avg Rqst / Sec
1	4	5	91563	95018	9347.37
2	4	5	91479	94842	9334.73
3	4	5	82573	94682	8930.03
4	4	5	100539	115031	10858.9
5	4	5	83624	96077	9054.27
6	4	5	91981	88721	9016.99
7	4	5	92472	89062	9057.16
8	4	5	91036	94442	9292.84
9	4	5	109319	108206	10825.6
10	4	5	92086	88691	9019.97
AVERAGES	**4**	**5**	**92667.2**	**96477.2**	**9473.786**

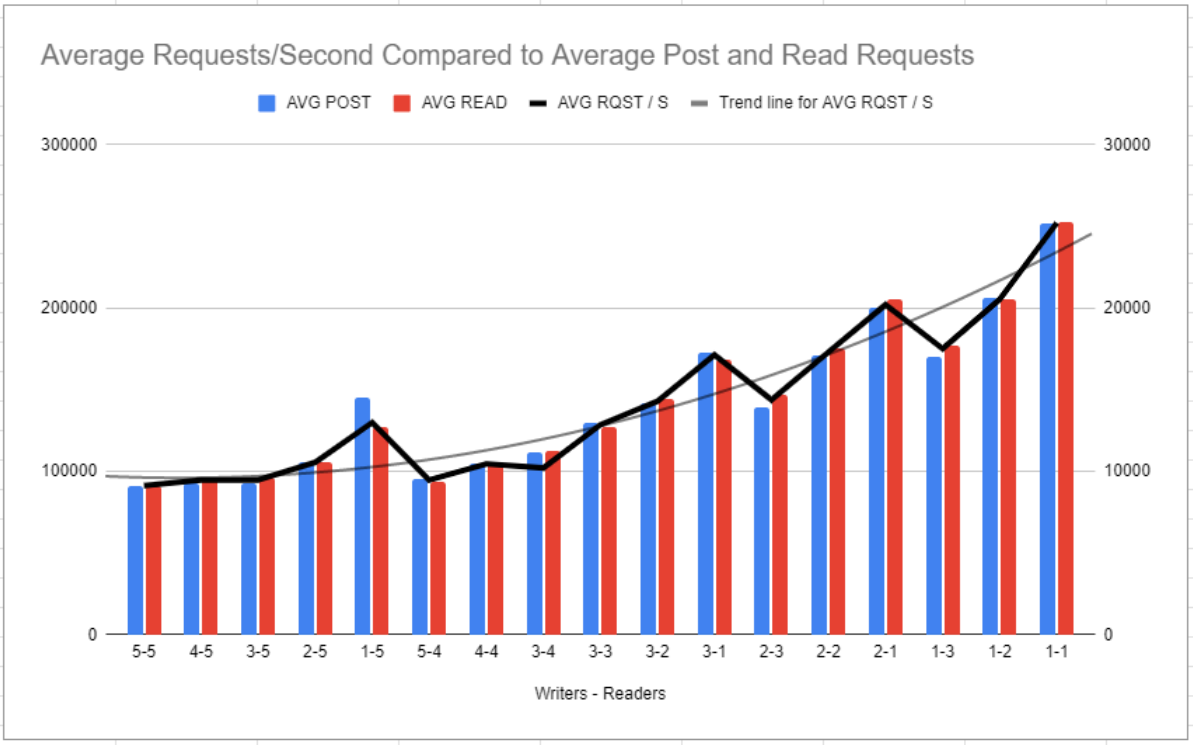
3

High throughput testing results done in the lab (5-4)

	Writers	Readers	Avg Post Rqst	Avg Read Rqst	Avg Rqst / Sec
1	5	4	94409	91096	9293.03
2	5	4	94302	90982	9282.47
3	5	4	88064	99911	9332.71
4	5	4	87373	99138	9260.24
5	5	4	113744	86956	10183.8
6	5	4	94799	91668	9340.74
7	5	4	94261	91058	9283.75
8	5	4	108821	108085	10849.3
9	5	4	85180	88679	8677.5
10	5	4	93823	90660	9241.7
AVERAGES	5	4	95477.6	93823.3	9474.524

4

Chat of Average rqs/s compared to Average post and read requests with a trend line for average rqs/s (using the right y-axis)



5

My Server against Sergio's Reference Client **at home with a 8 core CPU** without removing the Reader's shared mutex

```
Total poster requests: 1981185.
Average requests per poster thread: 396237.
Total reader requests: 1647152.
Average requests per reader thread: 329430.
Total requests: 3628337.
Average requests per thread: 362833.
Average requests per thread per second: 36283.3.
Press any key to continue . . .
```

6

My Server against Sergio's Reference Client **at home with a 8 core CPU** without removing the Reader's shared mutex in "debug" while running in debug mode **BEFORE** removing the reader's shared mutex lock

```
Total poster requests: 438089.
Average requests per poster thread: 87617.
Total reader requests: 527064.
Average requests per reader thread: 105412.
Total requests: 965153.
Average requests per thread: 96515.
Average requests per thread per second: 9651.43.
Press any key to continue . . .
```

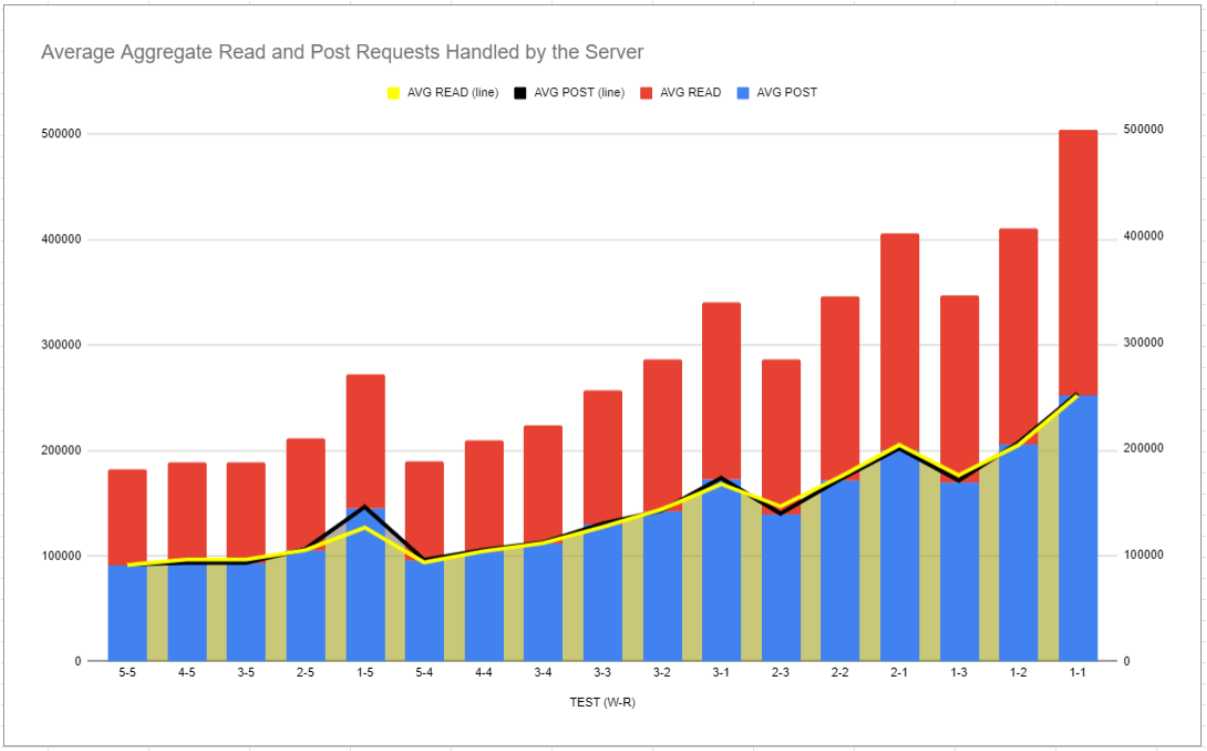
7

My Server against Sergio's Reference Client **at home with a 8 core CPU** without removing the Reader's shared mutex in "debug" while running in debug mode **AFTER** removing the reader's shared mutex lock

```
Total poster requests: 340669.
Average requests per poster thread: 68133.
Total reader requests: 757506.
Average requests per reader thread: 151501.
Total requests: 1098175.
Average requests per thread: 109817.
Average requests per thread per second: 10981.7.
Press any key to continue . . .
```

8

Chart of aggregated average read and post requests with area lines displaying whether read or post had a higher number of average requests.



9

Screenshots of the summarized output for all 15, 5-5 tests conducted on my server against Sergio's Reference Client.
https://drive.google.com/drive/folders/1NAUqebwyZfRiI6KXRLIbaMLP6aWKAYsD?usp=sharing

10

All Averages across all tests performed. The left column being the test the averages were gathered from in the format of Writers to Readers (W-R) – stretched to make readable

TEST (W-R)	AVG POST	AVG READ	AVG RQST / S
5-5	91007.33333	91426.26667	9121.232
4-5	92667.2	96477.2	9473.786
3-5	92667.2	96477.2	9473.786
2-5	105583.5	105444.6	10553.093
1-5	145277.4	126779.3	12986.23
5-4	95477.6	93823.3	9474.524
4-4	104756.4	104359.4	10455.702
3-4	111814.7	112134.8	10211.74
3-3	129585.2	127448	12851.65
3-2	141909.1	144280.9	14285.77
3-1	172403.8	168104.6	17132.91
2-3	138957.7	147028.4	14379.87
2-2	171206.2	174887.7	17305.19
2-1	200377.6	205266.6	20200.7
1-3	170161.8	176657.1	17503.3
1-2	205834.4	204960	20525.12
1-1	251913.1	252239.5	25207.57

11

File of all testing data gathered in addition to the graphs.

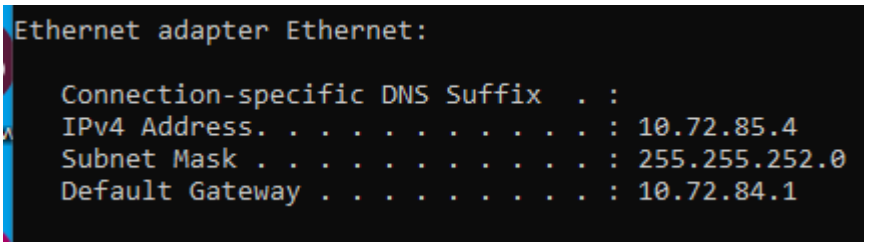
TestingData

https://docs.google.com/spreadsheets/d/1LC_z5oxzZv_n-C_GMdT0oSAtvSWmNwWL7i2vsL42kb0/edit?usp=sharing

(share link in case built-in file doesn't function)

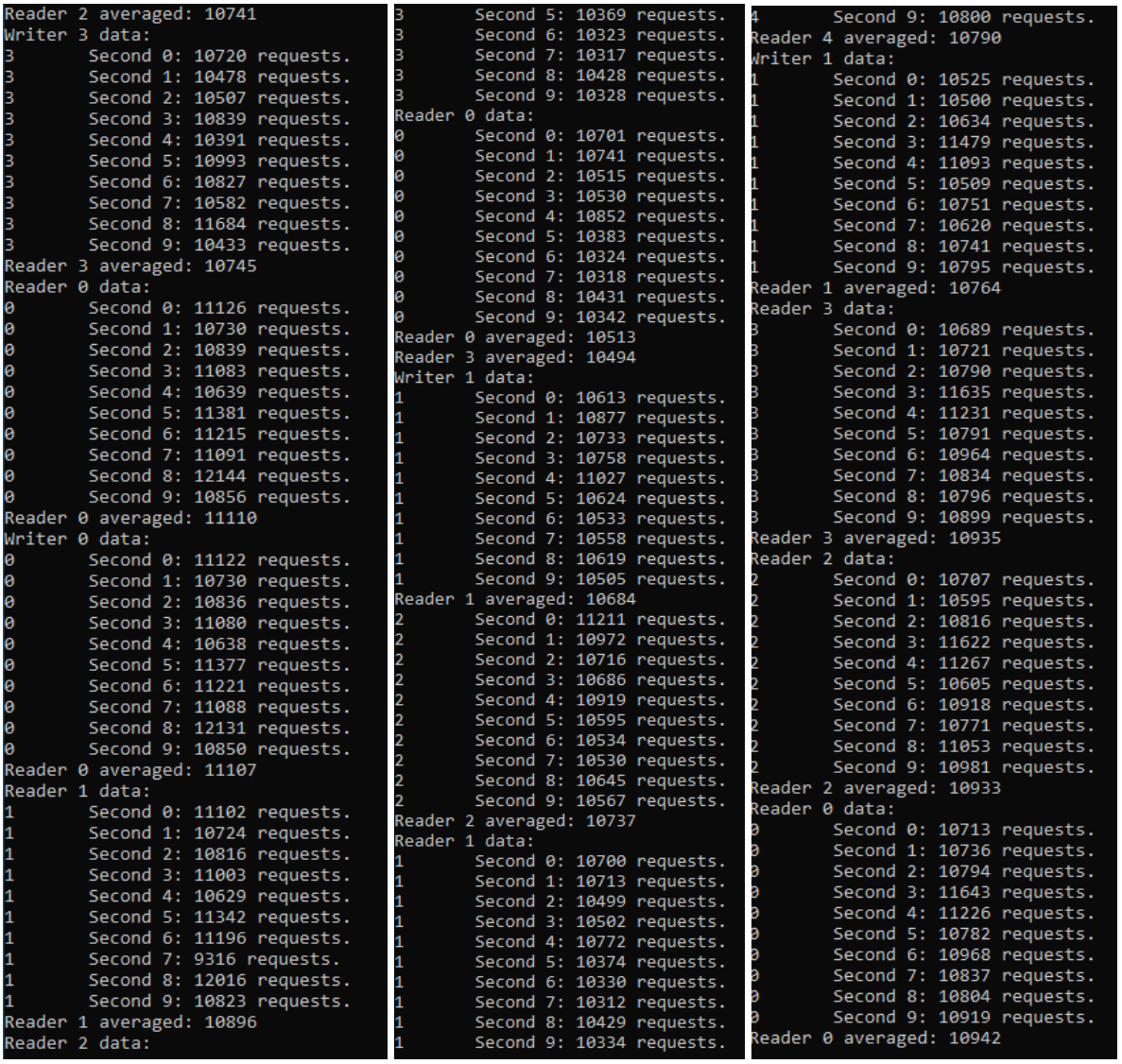
12

The Computer's IP config where the tests were run.



13

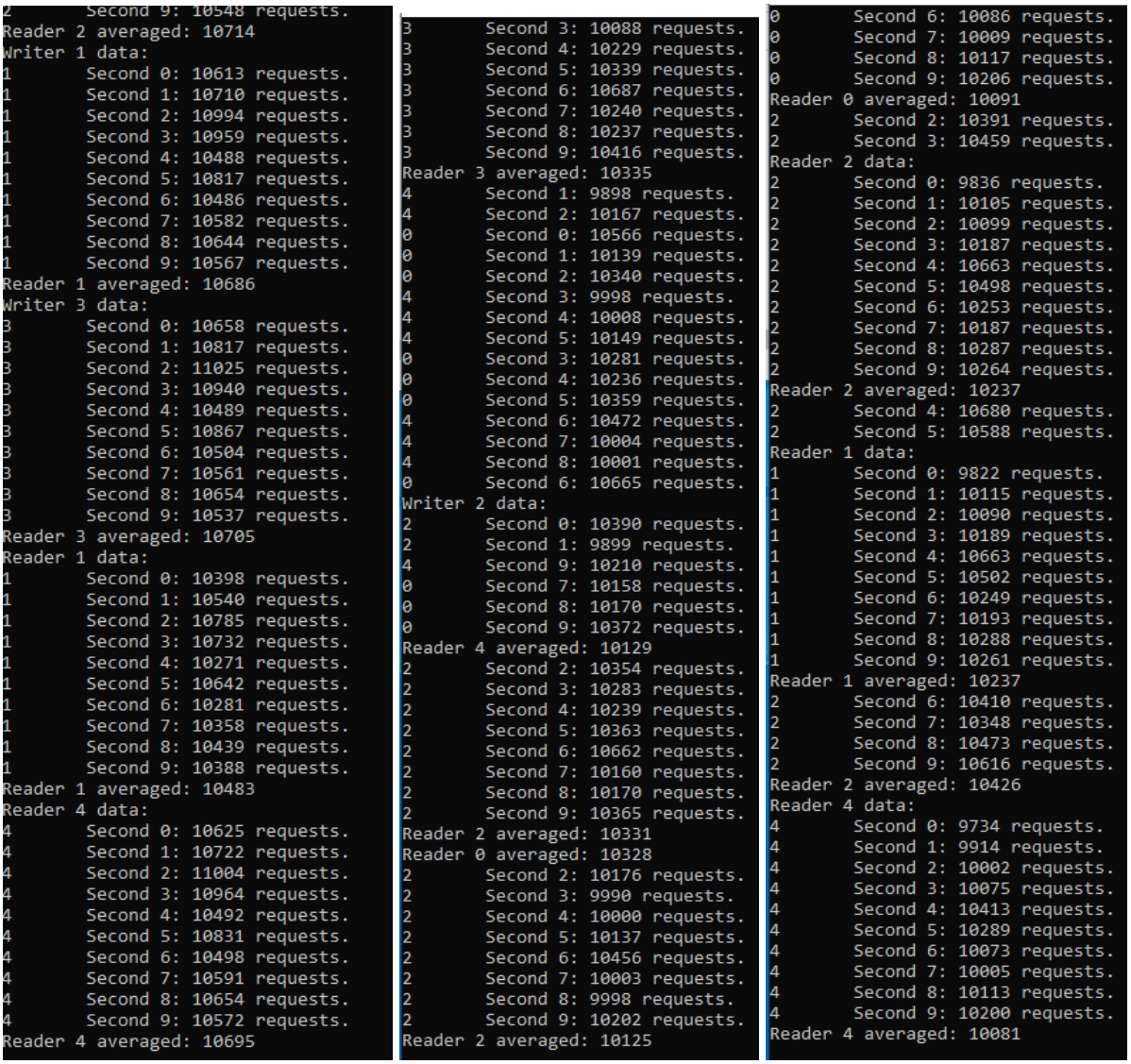
Results from testing my test harness against Sergio's Reference Server



Alternatively check the “TestHarnessResults” pictures “Sergio1” “Sergio2” “Sergio3” are his results

14

Results from testing my test harness against my server implementation



```
2       Second 9: 10548 requests.
Reader 2 averaged: 10714
Writer 1 data:
1       Second 0: 10613 requests.
1       Second 1: 10710 requests.
1       Second 2: 10994 requests.
1       Second 3: 10959 requests.
1       Second 4: 10488 requests.
1       Second 5: 10817 requests.
1       Second 6: 10486 requests.
1       Second 7: 10582 requests.
1       Second 8: 10644 requests.
1       Second 9: 10567 requests.
Reader 1 averaged: 10686
Writer 3 data:
3       Second 0: 10658 requests.
3       Second 1: 10817 requests.
3       Second 2: 11025 requests.
3       Second 3: 10940 requests.
3       Second 4: 10489 requests.
3       Second 5: 10867 requests.
3       Second 6: 10504 requests.
3       Second 7: 10561 requests.
3       Second 8: 10654 requests.
3       Second 9: 10537 requests.
Reader 3 averaged: 10705
Reader 1 data:
1       Second 0: 10398 requests.
1       Second 1: 10540 requests.
1       Second 2: 10785 requests.
1       Second 3: 10732 requests.
1       Second 4: 10271 requests.
1       Second 5: 10642 requests.
1       Second 6: 10281 requests.
1       Second 7: 10358 requests.
1       Second 8: 10439 requests.
1       Second 9: 10388 requests.
Reader 1 averaged: 10483
Reader 4 data:
4       Second 0: 10625 requests.
4       Second 1: 10722 requests.
4       Second 2: 11004 requests.
4       Second 3: 10964 requests.
4       Second 4: 10492 requests.
4       Second 5: 10831 requests.
4       Second 6: 10498 requests.
4       Second 7: 10591 requests.
4       Second 8: 10654 requests.
4       Second 9: 10572 requests.
Reader 4 averaged: 10695
```

```
3       Second 3: 10088 requests.
3       Second 4: 10229 requests.
3       Second 5: 10339 requests.
3       Second 6: 10687 requests.
3       Second 7: 10240 requests.
3       Second 8: 10237 requests.
3       Second 9: 10416 requests.
Reader 3 averaged: 10335
4       Second 1: 9898 requests.
4       Second 2: 10167 requests.
0       Second 0: 10566 requests.
0       Second 1: 10139 requests.
0       Second 2: 10340 requests.
4       Second 3: 9998 requests.
4       Second 4: 10008 requests.
4       Second 5: 10149 requests.
0       Second 3: 10281 requests.
0       Second 4: 10236 requests.
0       Second 5: 10359 requests.
4       Second 6: 10472 requests.
4       Second 7: 10004 requests.
4       Second 8: 10001 requests.
0       Second 6: 10665 requests.
Writer 2 data:
2       Second 0: 10390 requests.
2       Second 1: 9899 requests.
4       Second 9: 10210 requests.
0       Second 7: 10158 requests.
0       Second 8: 10170 requests.
0       Second 9: 10372 requests.
Reader 4 averaged: 10129
2       Second 2: 10354 requests.
2       Second 3: 10283 requests.
2       Second 4: 10239 requests.
2       Second 5: 10363 requests.
2       Second 6: 10662 requests.
2       Second 7: 10160 requests.
2       Second 8: 10170 requests.
2       Second 9: 10365 requests.
Reader 2 averaged: 10331
Reader 0 averaged: 10328
2       Second 2: 10176 requests.
2       Second 3: 9990 requests.
2       Second 4: 10000 requests.
2       Second 5: 10137 requests.
2       Second 6: 10456 requests.
2       Second 7: 10003 requests.
2       Second 8: 9998 requests.
2       Second 9: 10202 requests.
Reader 2 averaged: 10125
```

```
0       Second 6: 10086 requests.
0       Second 7: 10009 requests.
0       Second 8: 10117 requests.
0       Second 9: 10206 requests.
Reader 0 averaged: 10091
2       Second 2: 10391 requests.
2       Second 3: 10459 requests.
Reader 2 data:
2       Second 0: 9836 requests.
2       Second 1: 10105 requests.
2       Second 2: 10099 requests.
2       Second 3: 10187 requests.
2       Second 4: 10663 requests.
2       Second 5: 10498 requests.
2       Second 6: 10253 requests.
2       Second 7: 10187 requests.
2       Second 8: 10287 requests.
2       Second 9: 10264 requests.
Reader 2 averaged: 10237
2       Second 4: 10680 requests.
2       Second 5: 10588 requests.
Reader 1 data:
1       Second 0: 9822 requests.
1       Second 1: 10115 requests.
1       Second 2: 10090 requests.
1       Second 3: 10189 requests.
1       Second 4: 10663 requests.
1       Second 5: 10502 requests.
1       Second 6: 10249 requests.
1       Second 7: 10193 requests.
1       Second 8: 10288 requests.
1       Second 9: 10261 requests.
Reader 1 averaged: 10237
2       Second 6: 10410 requests.
2       Second 7: 10348 requests.
2       Second 8: 10473 requests.
2       Second 9: 10616 requests.
Reader 2 averaged: 10426
Reader 4 data:
4       Second 0: 9734 requests.
4       Second 1: 9914 requests.
4       Second 2: 10002 requests.
4       Second 3: 10075 requests.
4       Second 4: 10413 requests.
4       Second 5: 10289 requests.
4       Second 6: 10073 requests.
4       Second 7: 10005 requests.
4       Second 8: 10113 requests.
4       Second 9: 10200 requests.
Reader 4 averaged: 10081
```

Alternatively check the “TestHarnessResults” pictures “My1” “My2” and “My3” are my results