Concurrent and Parallel Systems Assignment – High Throughput TCP Forum Server and Test Harness

Evaluation Report

By Jacob Whitehead

# Experiment Descriptions

For my experiments I chose to vary the number of reader and poster threads in a variety of different ways. Firstly, I wanted to test the server using an equal number of reader and poster threads to view how this affects the performance. I started by conducting 10 runs using just 1 reader and 1 poster thread, meaning a total of 2 threads were running. I then progressed both the reader and poster thread count to 2 for 10 runs and then 3 for 10 runs. I ran each configuration 10 times to get an average for the test, as there are a wide variety of things that can affect the results, such as other processes being run by the CPU, as well as how to CPU scheduler behaves. After this, I wanted to vary the number of threads so that the poster and reader thread counts were different, so I ran the test harness using 1 poster and 2 reader threads and took the average from the 10 runs I conducted. I also reversed the numbers, so that I was running with 2 poster and 1 reader thread. After this I also conducted a similar experiment, where I ran both with 1 poster and 3 reader threads, as well as vice versa, meaning 3 poster and 1 reader thread. My final experiment was to run the test harness using 5 reader and 5 poster threads, providing a total of 10; this is the most performance intensive test I decided to run, which is shown by the results that were produced.

# Analysis and Interpretation of Data

Based on the results from my experiments, I calculated the mean averages for each of the configurations that I tested. I then plotted these average values on scatter graphs to allow me to see if there any trends, and to show the relationship between the number of threads connected to the server and how it performs. I also wanted to distinguish whether poster or reader threads have any sort of effect on how the server does in high throughput situations, compared to the other.

I plotted a graph for the total number of threads in use against the overall throughput of the server, based on an average of the reader and poster thread averages, for each configuration. The graph demonstrates a clear trend line, showing an inverse relationship between the total number of threads being used by the test harness, and the performance of the server. My results show that when a smaller number of threads are utilised, then the performance of each thread is drastically increased; this is because of the fact that the more threads that exist, the more work the CPU scheduler will have to do, which slows down execution. Another limiting factor may also be the number of cores in the CPU from the computer that I performed my experiments on, as generally server CPUs will have a lot more cores than a standard desktop, meaning more threads can be running concurrently.

I also wanted to assess whether poster or reader threads have any effect on the performance – for example in situations where there are more of one type. Unsurprisingly, both the graphs for read and post request averages follow a similar trend to the graph for the system as a whole, showing an inverse relationship, meaning that as the number of threads increases, the number of requests per second that the server can process becomes lower.

One thing I observed is that in every single one of my tests, the average number of read requests was always lower than the number of post requests. I believe this is caused by the fact that the read requests do not utilise a standard mutex within the server implementation, but instead use a shared mutex, meaning that many reader threads can access the critical area concurrently – however post requests (which take longer to complete) fully lock the thread, meaning no other thread can perform a lock. If the thread is locked while the post is being performed, it means that the reader thread must simply wait; I think this is the reason why generally my harness shows that there are fewer read requests performed.

# Future Server Improvements

There are a variety of things that could be changed within the server implementation to make it quicker and more efficient. Firstly, the implementation of the request parser could potentially be improved, as currently it performs regular expression matches for each type of request until it either finds one that matches or doesn’t recognise the request and responds appropriately. Regular expression matching isn’t particularly efficient in C++, unless some additional library or smart code is utilised, and so I believe this will be negatively affecting the performance of the server. If I were to write code that matches against the request type quicker, perhaps by only checking certain characters at the start of the request to determine which type it is, then the performance will likely be greatly improved. The current implementation checks each type of request in succession, so the requests which are checked later are going to be negatively impacted by the fact that several regex matches occur, when they don’t need to if the algorithm could immediately determine the request type.

Another way that could potentially improve the performance of the server would be to run it using a dedicated server CPU, which has considerably more cores than a standard desktop CPU, and so more threads can be executed in parallel. Modifying the server design to make use of a thread pool for individual requests would also make it more efficient; currently the server creates one thread per client that connects to it and processes requests sequentially, but it would be possible for that thread to create more to process a larger volume of requests. There could be a threshold at which the server instantiates additional threads to process requests when it is under high load from the clients.

Certain patterns could also be implemented to improve the efficiency of the server, such as the consumer producer pattern, which involves a queue or priority queue to track and process requests in a strategic order, so that the server can produce a higher throughput. If you prioritise post requests then the server would be more accurate, but prioritising read requests may improve efficiency, as read requests can occur concurrently, unlike post requests which lock fully around critical areas, such as when data is saved to memory.

# Investigation Into Factors That Influence Throughput

The main factor that affects the throughput of the server is the machine that it is being run on. A computer with a slower processor (in terms of clock speed) with less cores will perform worse than the machines in Cantor 9341. I conducted an experiment by running the server and test harness on my personal laptop so I could compare the results to that run on the machine I used in Cantor 9341.

Comparing the results of the test I conducted with 3 reader and 3 poster threads on my personal laptop, I can see that my laptop is significantly poorer at running the server and test harness. While the machine in Cantor 9341 resulted in an average throughput of 10757.64 requests per second, the experiment on my own laptop yielded a result of only 8910.0105 requests per second; this is a difference of 18.79%, which is significant, and shows how the CPU can influence the results.

Another thing that influences the performance of the server is the number of and running complexity of other applications and processes that are being executed by the CPU. For example, executing the server and test harness in a debug configuration, with two instances of Visual Studio running has quite a large negative impact on the performance of the system. Running the executable from the command line is a lot more efficient, as the two Visual Studio processes will take up lots of system resources. Running the server and client on two separate machines is also another way to improve the performance, as the CPU will be able to dedicate more threads to each process, and therefore process more requests per second.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *\*This set of results was obtained by running the server and test harness from my personal laptop so I could compare it to the results from the machine I used in Cantor 9341\** | | | | | | | |
| Poster thread count | Total post requests | Total post thread run time (s) | Average post requests per second per thread | Reader thread count | Total read requests | Total read thread run time (s) | Average read requests per second per thread |
| 3 | 288542 | 30.0002 | 9618.02 | 3 | 280267 | 30.0001 | 9342.2 |
| 3 | 258031 | 30.0001 | 8601 | 3 | 269245 | 30.0002 | 8974.76 |
| 3 | 270097 | 30.0001 | 9003.2 | 3 | 265370 | 30.0002 | 8845.62 |
| 3 | 265086 | 30.0002 | 8836.13 | 3 | 257805 | 30.0001 | 8593.48 |
| 3 | 263577 | 30.0002 | 8785.83 | 3 | 253492 | 30.0002 | 8449.67 |
| 3 | 268898 | 30.0001 | 8963.23 | 3 | 270987 | 30.0001 | 9032.87 |
| 3 | 288873 | 30.0002 | 9629.03 | 3 | 285230 | 30.0001 | 9507.62 |
| 3 | 271448 | 30.0001 | 9048.23 | 3 | 250747 | 30 | 8358.22 |
| 3 | 255046 | 30.0001 | 8501.49 | 3 | 244087 | 30.0002 | 8136.17 |
| 3 | 276625 | 30.0001 | 9220.79 | 3 | 262581 | 30.0002 | 8752.65 |

Mean read requests per second: 8799.36

Mean post requests per second: 9020.675

Total threads: 6

Mean total requests per second: 8910.0105

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Poster thread count | Total post requests | Total post thread run time (s) | Average post requests per second per thread | Reader thread count | Total read requests | Total read thread run time (s) | Average read requests per second per thread |
| 1 | 198408 | 10 | 19840.8 | 1 | 215634 | 10.0092 | 21543.6 |
| 1 | 198972 | 10 | 19897.1 | 1 | 198052 | 10 | 19805.1 |
| 1 | 195713 | 10.0001 | 19571.2 | 1 | 185547 | 10 | 18554.7 |
| 1 | 196135 | 10 | 19613.4 | 1 | 195397 | 10 | 19539.7 |
| 1 | 196871 | 10 | 19687.1 | 1 | 195058 | 10 | 19505.8 |
| 1 | 200002 | 10 | 20000.2 | 1 | 199271 | 10 | 19927 |
| 1 | 200774 | 10 | 20077.4 | 1 | 199878 | 10 | 19987.8 |
| 1 | 199209 | 10 | 19920.8 | 1 | 197703 | 10 | 19770.3 |
| 1 | 205486 | 10 | 20548.6 | 1 | 203780 | 10 | 20378 |
| 1 | 200313 | 10 | 20031.2 | 1 | 197087 | 10 | 19708.6 |

Mean read requests per second: 19872.06

Mean post requests per second: 19918.78

Total threads: 2

Mean total requests per second: 19895.42

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Poster thread count | Total post requests | Total post thread run time (s) | Average post requests per second per thread | Reader thread count | Total read requests | Total read thread run time (s) | Average read requests per second per thread |
| 2 | 315978 | 20 | 15798.9 | 2 | 302102 | 20 | 15105.1 |
| 2 | 315603 | 20.0001 | 15780.1 | 2 | 304014 | 20.0001 | 15200.6 |
| 2 | 303702 | 20 | 15185.1 | 2 | 297774 | 20.0001 | 14888.7 |
| 2 | 313196 | 20.0001 | 15659.7 | 2 | 300931 | 20 | 15046.5 |
| 2 | 311091 | 20.0001 | 15554.5 | 2 | 304709 | 20.0001 | 15235.4 |
| 2 | 308011 | 20 | 15400.5 | 2 | 292596 | 20.0066 | 14625 |
| 2 | 313697 | 20.0001 | 15684.8 | 2 | 296580 | 20.0025 | 14827.2 |
| 2 | 312228 | 20 | 15611.4 | 2 | 302295 | 20 | 15114.7 |
| 2 | 315098 | 20.0001 | 15754.8 | 2 | 303987 | 20 | 15199.3 |
| 2 | 304209 | 20.0001 | 15210.3 | 2 | 295731 | 20.0001 | 14786.5 |

Mean read requests per second: 15002.9

Mean post requests per second: 15564.01

Total threads: 4

Mean total requests per second: 15106.6

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Poster thread count | Total post requests | Total post thread run time (s) | Average post requests per second per thread | Reader thread count | Total read requests | Total read thread run time (s) | Average read requests per second per thread |
| 3 | 333132 | 30.0004 | 11104.2 | 3 | 321573 | 30.0001 | 10719.1 |
| 3 | 333606 | 30.0001 | 11120.2 | 3 | 321656 | 30.0601 | 10700.4 |
| 3 | 329723 | 30.0002 | 10990.7 | 3 | 317375 | 30.0001 | 10579.1 |
| 3 | 323878 | 30.0002 | 10795.9 | 3 | 315677 | 30.0253 | 10513.7 |
| 3 | 323964 | 30.0002 | 10798.7 | 3 | 313754 | 30.0002 | 10458.4 |
| 3 | 326244 | 30.0001 | 10874.8 | 3 | 316843 | 30.0077 | 10558.7 |
| 3 | 326348 | 30.0002 | 10878.2 | 3 | 316613 | 30.0124 | 10549.4 |
| 3 | 324656 | 30.0002 | 10821.8 | 3 | 315008 | 30.0077 | 10497.6 |
| 3 | 340595 | 30.0001 | 11353.1 | 3 | 322626 | 30.0078 | 10751.4 |
| 3 | 322444 | 30.0001 | 10748.1 | 3 | 310329 | 30.0145 | 10339.3 |

Mean read requests per second: 10566.17

Mean post requests per second: 10948.57

Total threads: 6

Mean total requests per second: 10757.64

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Poster thread count | Total post requests | Total post thread run time (s) | Average post requests per second per thread | Reader thread count | Total read requests | Total read thread run time (s) | Average read requests per second per thread |
| 2 | 361595 | 20 | 18079.7 | 1 | 178014 | 10 | 17801.4 |
| 2 | 360852 | 20.0001 | 18042.6 | 1 | 176439 | 10 | 17643.9 |
| 2 | 361047 | 20.0001 | 18052.3 | 1 | 177220 | 10 | 17722 |
| 2 | 358108 | 20.0001 | 17905.3 | 1 | 177174 | 10 | 17717.4 |
| 2 | 374008 | 20 | 18700.4 | 1 | 169623 | 10 | 16962.3 |
| 2 | 363187 | 20.0001 | 18159.3 | 1 | 172202 | 10 | 17220.1 |
| 2 | 358738 | 20.0001 | 17936.8 | 1 | 176098 | 10 | 17609.8 |
| 2 | 366361 | 20.0001 | 18318 | 1 | 177360 | 10 | 17736 |
| 2 | 360759 | 20.0001 | 18037.9 | 1 | 176960 | 10 | 17696 |
| 2 | 360546 | 20 | 18027.3 | 1 | 168681 | 10 | 16868.1 |

Mean read requests per second: 17497.7

Mean post requests per second: 18125.96

Total threads: 3

Mean total requests per second: 17811.83

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Poster thread count | Total post requests | Total post thread run time (s) | Average post requests per second per thread | Reader thread count | Total read requests | Total read thread run time (s) | Average read requests per second per thread |
| 1 | 190784 | 10 | 19078.3 | 2 | 359107 | 20.0001 | 17955.3 |
| 1 | 182795 | 10 | 18279.5 | 2 | 359078 | 20 | 17953.9 |
| 1 | 181014 | 10 | 18101.3 | 2 | 353424 | 20 | 17671.2 |
| 1 | 180245 | 10 | 18024.4 | 2 | 355564 | 20 | 17778.2 |
| 1 | 180561 | 10 | 18056 | 2 | 347139 | 20.0001 | 17356.9 |
| 1 | 179553 | 10 | 17955.3 | 2 | 352419 | 20 | 17620.9 |
| 1 | 183180 | 10 | 18317.9 | 2 | 352615 | 20 | 17630.7 |
| 1 | 180516 | 10 | 18051.5 | 2 | 359110 | 20.0001 | 17955.4 |
| 1 | 179286 | 10 | 17928.5 | 2 | 353692 | 20 | 17684.6 |
| 1 | 180507 | 10 | 18050.7 | 2 | 352545 | 20 | 17627.2 |

Mean read requests per second: 17723.43

Mean post requests per second: 18184.34

Total threads: 3

Mean total requests per second: 17953.89

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Poster thread count | Total post requests | Total post thread run time (s) | Average post requests per second per thread | Reader thread count | Total read requests | Total read thread run time (s) | Average read requests per second per thread |
| 1 | 155334 | 10.0001 | 15533.3 | 3 | 452842 | 30.0001 | 15094.7 |
| 1 | 153732 | 10 | 15373.2 | 3 | 453435 | 30 | 15114.5 |
| 1 | 156603 | 10 | 15660.3 | 3 | 463344 | 30.0007 | 15444.4 |
| 1 | 153669 | 10 | 15366.9 | 3 | 454135 | 30.0001 | 15137.8 |
| 1 | 156541 | 10 | 15654.1 | 3 | 459728 | 30.0001 | 15324.2 |
| 1 | 159863 | 10 | 15986.3 | 3 | 454067 | 30.0001 | 15135.5 |
| 1 | 158947 | 10 | 15894.7 | 3 | 450799 | 30 | 15026.6 |
| 1 | 156890 | 10 | 15689 | 3 | 459983 | 30 | 15332.7 |
| 1 | 158581 | 10 | 15858 | 3 | 456733 | 30.0001 | 15224.4 |
| 1 | 155615 | 10 | 15561.5 | 3 | 457669 | 30.0001 | 15255.6 |

Mean read requests per second: 15209.04

Mean post requests per second: 15657.73

Total threads: 4

Mean total requests per second: 15433.39

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Poster thread count | Total post requests | Total post thread run time (s) | Average post requests per second per thread | Reader thread count | Total read requests | Total read thread run time (s) | Average read requests per second per thread |
| 3 | 453742 | 30.0001 | 15124.7 | 1 | 144026 | 10 | 14402.5 |
| 3 | 459409 | 30.0001 | 15313.6 | 1 | 147136 | 10 | 14713.6 |
| 3 | 462396 | 30.0001 | 15413.1 | 1 | 150417 | 10 | 15041.7 |
| 3 | 453448 | 30.0001 | 15114.9 | 1 | 147710 | 10 | 14771 |
| 3 | 451990 | 30.0001 | 15066.3 | 1 | 147143 | 10 | 14714.3 |
| 3 | 456599 | 30.0001 | 15219.9 | 1 | 148330 | 10 | 14833 |
| 3 | 465532 | 30.0001 | 15517.7 | 1 | 150720 | 10 | 15071.9 |
| 3 | 459471 | 30 | 15315.7 | 1 | 146699 | 10 | 14669.9 |
| 3 | 476838 | 30.0001 | 15894.5 | 1 | 153910 | 10 | 15391 |
| 3 | 476339 | 30.0001 | 15877.9 | 1 | 153629 | 10 | 15362.8 |

Mean read requests per second: 14897.17

Mean post requests per second: 15385.83

Total threads: 4

Mean total requests per second: 15141.5

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Poster thread count | Total post requests | Total post thread run time (s) | Average post requests per second per thread | Reader thread count | Total read requests | Total read thread run time (s) | Average read requests per second per thread |
| 5 | 344281 | 50.0006 | 6885.54 | 5 | 332197 | 50.0712 | 6634.49 |
| 5 | 355735 | 50.0006 | 7114.61 | 5 | 339101 | 50.0439 | 6776.07 |
| 5 | 346109 | 50.0005 | 6922.12 | 5 | 331959 | 50.0629 | 6630.84 |
| 5 | 346904 | 50.0006 | 6937.99 | 5 | 332980 | 50.0169 | 6657.36 |
| 5 | 351255 | 50.0004 | 7025.04 | 5 | 343224 | 50.0634 | 6855.78 |
| 5 | 348610 | 50.0049 | 6971.51 | 5 | 336710 | 50.0402 | 6728.79 |
| 5 | 345451 | 50.0003 | 6908.98 | 5 | 331584 | 50.0664 | 6622.89 |
| 5 | 348361 | 50.0293 | 6963.14 | 5 | 332755 | 50.0191 | 6652.56 |
| 5 | 347172 | 50.0009 | 6943.32 | 5 | 340077 | 50.0804 | 6790.63 |
| 5 | 347689 | 50.0004 | 6953.72 | 5 | 338288 | 50.019 | 6763.19 |

Mean read requests per second: 6711.26

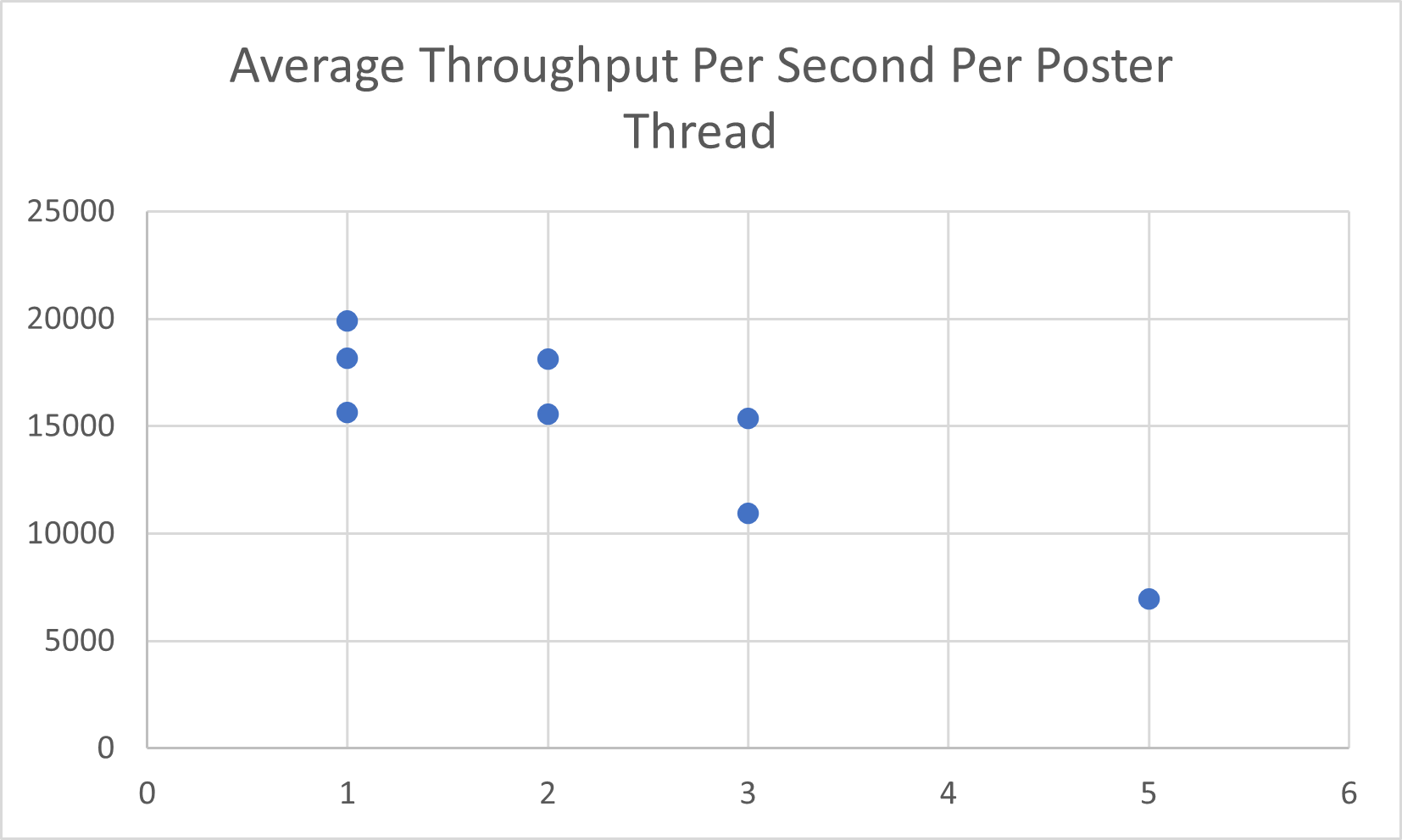
Mean post requests per second: 6962.597

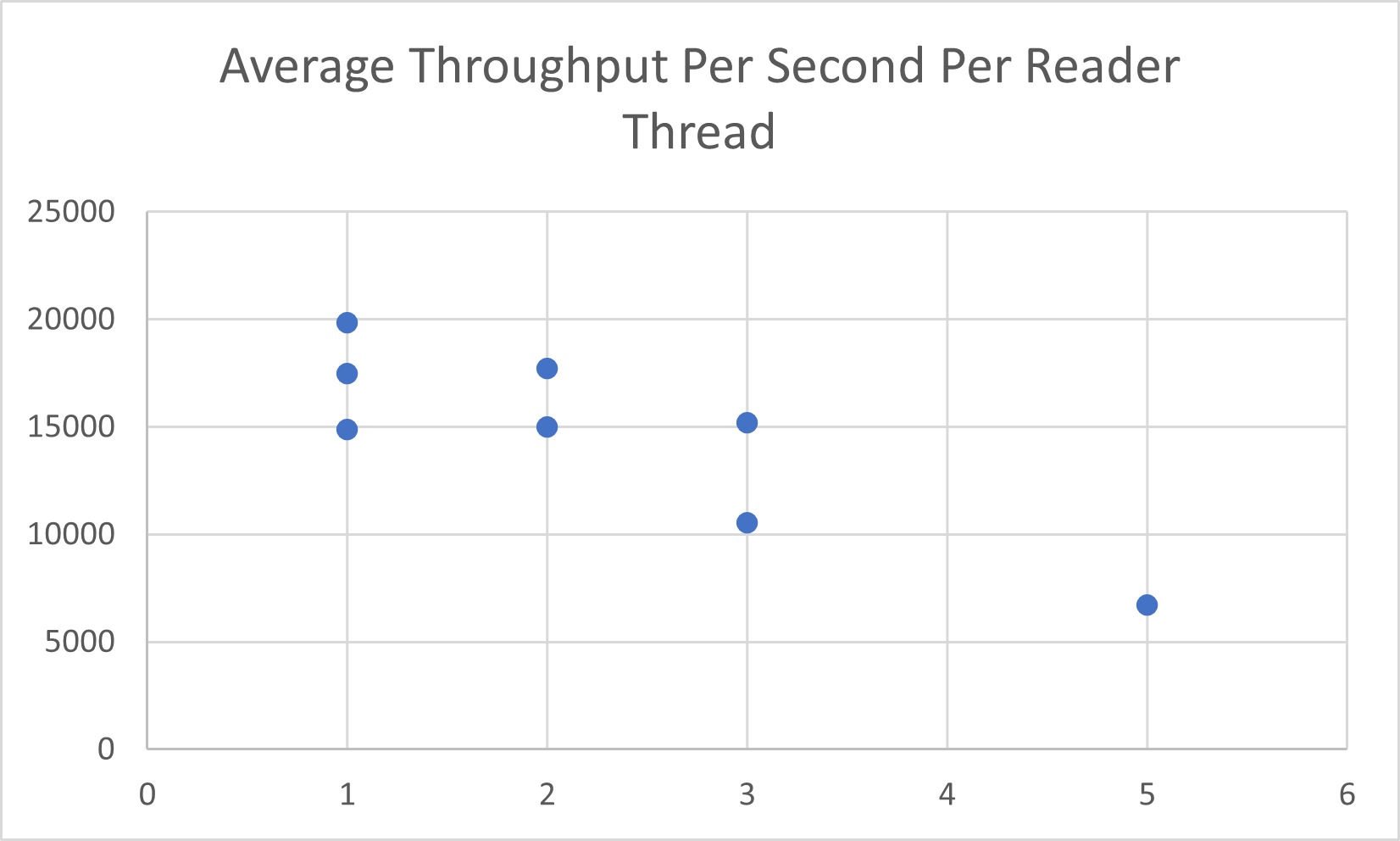
Total threads: 10

Mean total requests per second: 6836.929

Chart, scatter chart

Description automatically generated





Throughput tests were completed on the following machine in Cantor 9341 (IP 10.72.84.33):

Text

Description automatically generated