KING ABDULAZIZ UNIVERSITY

THE COLLEGE OF ENGINEERING

Operating Systems

EE463 – Spring 2023

Benchmark Test Report

Name	ID
SAAD ALI SADAGAH AL JEHANI	1935151
KHALED MAMNDOUH NASSER AL DAHASI	1935129
NAWAF SAMI MUALLA Al-HARBI	1936576

Tuesday, June 6, 2023

Introduction

In this report we will test the server using Apache benchmarking tool. To gather about how the server will operate under heavy load. The server already has implemented the Thread Pool architecture, and a FIFO queue and multiple overload handling policies. Our objective in this report is to test the server using all overload handling policies, and different sizes of queue and different concurrent requests at a time. We must set a min_rate and must find a max_rate where our server starts getting errors.

The number of cores in the test system is 8 cores. We will start benchmarking by setting the min_rate to 10 and increasing it by 10 each time. The tests will be repeated with different amounts of pool sizes and buffer sizes.

Testing the server with default handling method (BLCK):

While testing different Pool sizes and different Queue sizes. We found that increasing the pool size will always increase the max_rate. But at the same it increases the reply time. Thus, it is our choice to which we see fit. We see the optimum error free sizes are the ones with the minimum reply time. And at the same time, it uses the least pools and the least queue size. When testing with different values for the queue we found that increasing the queue size does increase the max_rate. And

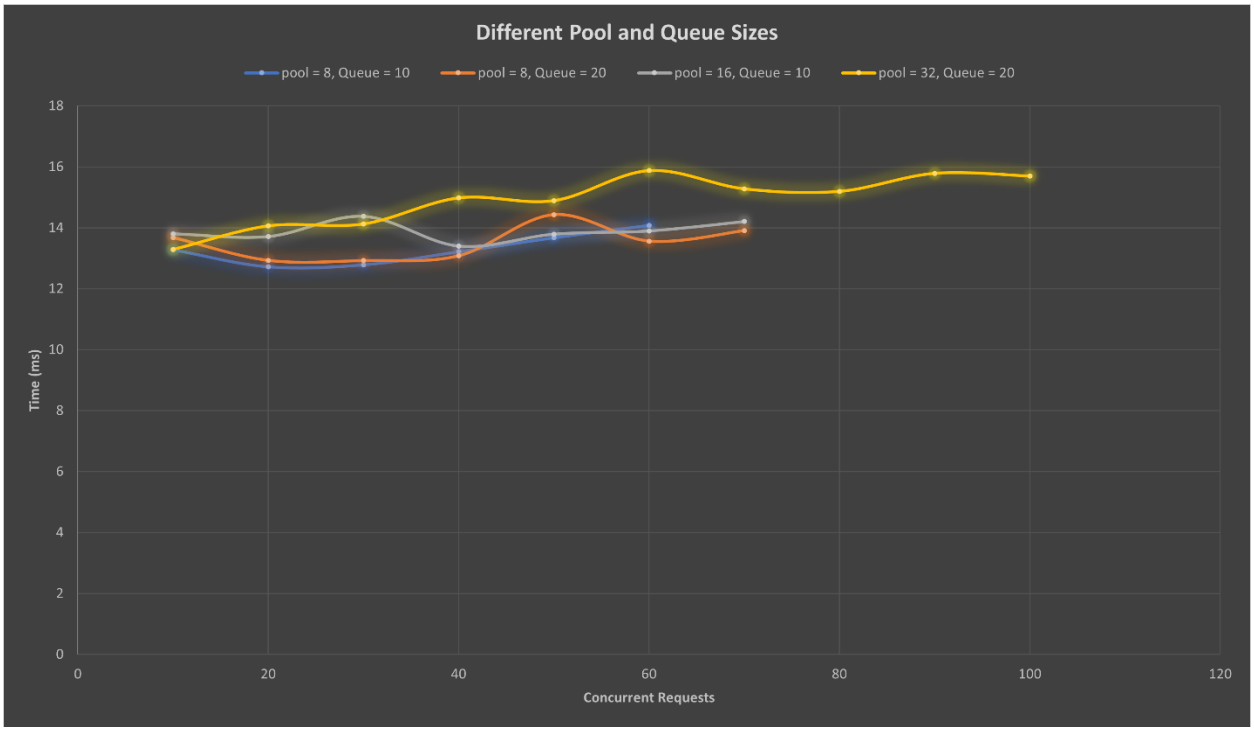

Figure 1 different pool and queue sizes

increasing the pool size does increase the max_rate as well but more pools are expensive. And it is not worth the increase. Thus, we found that the optimum error free settings are eight threads in the thread pool. Which is the same as the number of cores in the system. And the queue size is 20.

Table 1 max rate of different pool and queue sizes

Pool and Queue Sizes	Max_rate	Avg Reply Time (milliseconds)
Pools = 8, Queue =10	60	14.08
Pools = 8, Queue =20	**70**	**13.908**
Pools = 16, Queue =10	70	14.203
Pools = 32, Queue =20	100	15.695

Testing other overload handling methods:

Now that we have the optimal error-free setting, we use it to test the server when it's using DRPT and DRPH. We also compare the results with the simple server and also with the previous results of BLCK. Shown in figure 2 is the reply time for the different servers. We can see how the simple server has the longest reply time out of the others, for DRPT the maximum error-free request rate was 20 concurrent requests. After that no errors happen necessarily, what really happens is that a lot of requests get refused more and more. This same thing happens also in DRPH where the max rate is also 20 for the same reason as we consider that a lot of refusals is an error. As for reply time, in this max rate for DRPT its equal to 13.007 milliseconds and for DRPH its equal to 13.856 milliseconds.

Compared with the simple server, our server is better due to simple server having reply time of 21.248 milliseconds at the same max rate which is 20. As for “request rate at which reply-time

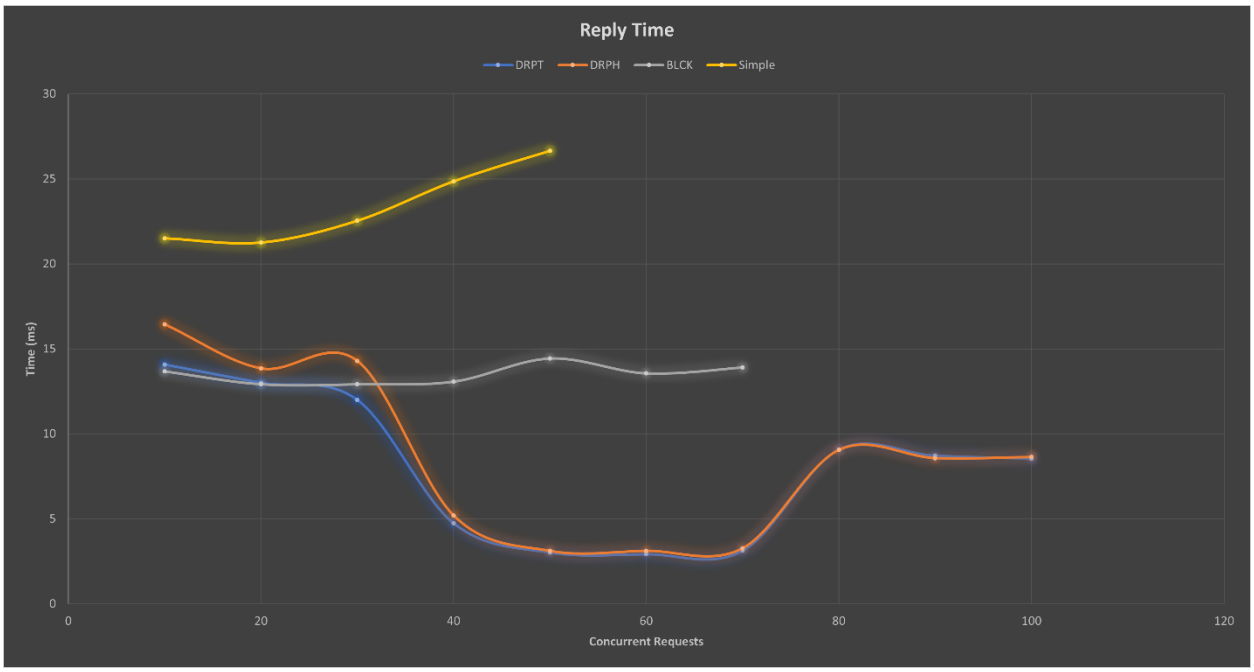

Figure 2 Reply Times

latency increases” we don’t seem to have a “Flat” behavior for reply to times in this setting for both DRPH and DRPT, it changes as soon as we test with our min rate which is 10 requests. As for testing when it’s 25% above the maximum rate, we test for when its 25% above 20 requests. We take approximately 30 requests and for DRPT we see that the refused requests are equal to, and the reply time is 12.00 milliseconds and for DRPH we see that the refused requests is equal to and the reply time is 14.296 milliseconds. Be aware that we take refusals as errors, because otherwise no “errors” happen as discussed earlier.

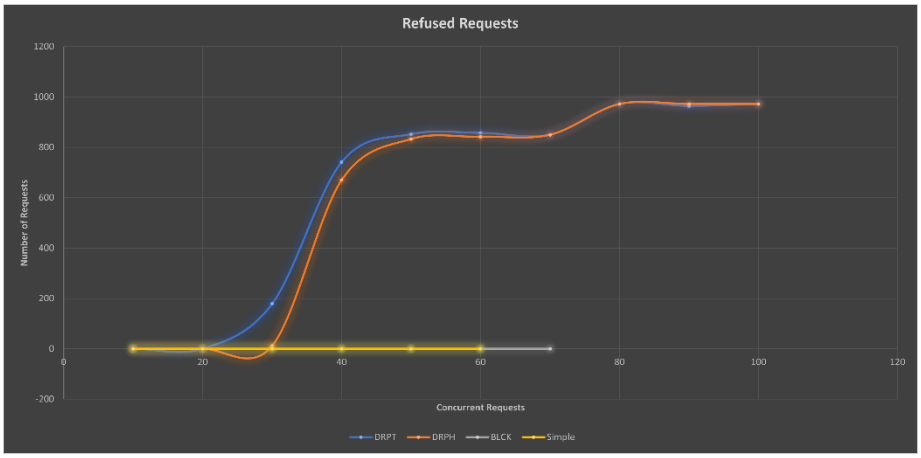

Figure 3 Refused Requests

Even though DRPT and DRPH have better reply to times than BLCK, this is due to a lot of refusals happening in the main thread of DRPH and DRPT. Shown in figure 3 it can be seen that refusals for BLCK and Simple are 0, which is expected as they don't refuse requests. But this increase of refusals for DRPH and DRPT theoretically in our design should not happen. From our investigations we concluded that it is happening due to worker threads being "held" in the serve method of serveWebRequest class when refusals are happening at the same time, not allowing workers to finish their serve methods. The same thing can be concluded from figure 4, which shows the opposite which is completed requests. These low completed requests for DRPH and DRPT are actually mostly the first few requests that the worker gets held in their serve method. After some refusals, the workers conclude their serving and that's why they get completed.

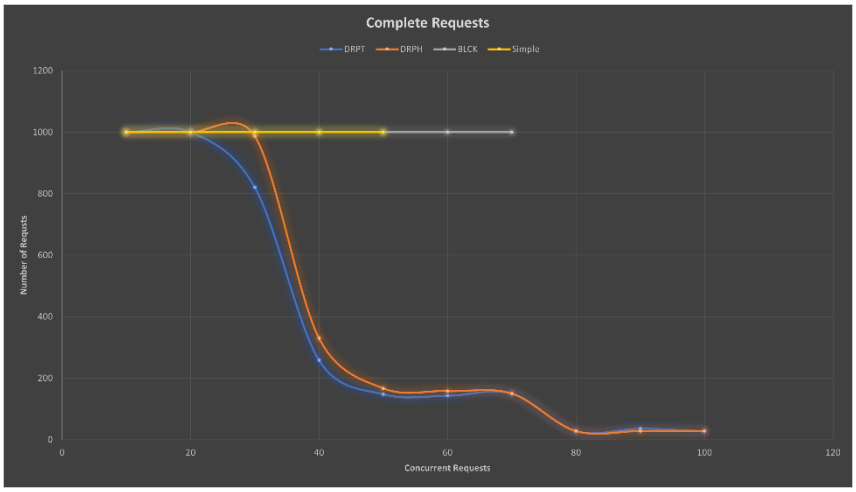

Figure 4 Complete Requests

Future work (Suggestions):

What could be done to improve the server is to solve the issue we reported in the DRPH and DRPT methods where the workers get held by the main thread. This could be solved by two approaches:

1. First improvement: Adding more semaphores to our program and classes ensuring that the main thread doesn't take too much time with the refuse method. This is essential to give the workers room to breathe and get their chance to use the serve method and finish it. The issue with this is that it could introduce unwanted delays or maybe even degenerate the server into a simple server again as workers wouldn't be serving at the same time.
2. Second improvement: Modifying the serveWebRequest class and in particular, the serve and refuse methods. We didn't have access to these methods and so we couldn't investigate what the issue could stem from. But what could be done is adding some synchronization tools to further improve these methods and make a remedy for the issue discussed to ensure that refusals and serves could happen in a synchronized manner.

Another suggestion would be to implement load balancing techniques. Load balancing can distribute incoming requests across multiple servers or resources, ensuring optimal resource utilization and improved response times. By implementing load balancing algorithms such as round-robin, least connection, or weighted round-robin, the server can effectively distribute the workload and handle high traffic situations more efficiently.

Appendix A:

BLCK	pool = 8, Queue = 10		
	Reply Time (ms)	Failed Requests	Complete Requests
10	13.276	0	1000
20	12.714	0	1000
30	12.779	0	1000
40	13.211	0	1000
50	13.675	0	1000
60	14.08	0	1000

BLCK	pool = 8, Queue = 20		
	Reply Time (ms)	Failed Requests	Complete Requests
10	13.677	0	1000
20	12.925	0	1000
30	12.92	0	1000
40	13.078	0	1000
50	14.432	0	1000
60	13.561	0	1000
70	13.908	0	1000

BLCK	pool = 16, Queue = 10		
	Reply Time (ms)	Failed Requests	Complete Requests
10	13.802	0	1000
20	13.712	0	1000
30	14.373	0	1000
40	13.399	0	1000
50	13.789	0	1000
60	13.895	0	1000
70	14.203	0	1000

DRPT	pool = 8, Queue = 20		
	Reply Time (ms)	Failed Requests	Complete Requests
10	14.08	0	1000
20	13.007	0	1000
30	12	179	821
40	4.749	741	259
50	3.028	852	148
60	2.929	857	143
70	3.126	850	150
80	9.076	972	28
90	8.717	964	36
100	8.553	972	28

DRPH	pool = 8, Queue = 20		
	Reply Time (ms)	Failed Requests	Complete Requests
10	16.447	0	1000
20	13.856	0	1000
30	14.296	12	988
40	5.205	670	330
50	3.117	833	167
60	3.117	842	158
70	3.271	850	150
80	9.059	972	28
90	8.575	972	28
100	8.639	972	28

Simple	Reply Time (ms)	Failed Requests	Complete Requests
10	21.499	0	1000
20	21.248	0	1000
30	22.533	0	1000
40	24.853	0	1000
50	26.646	0	1000

BLCK	pool = 32, Queue = 24		
	Reply Time (ms)	Failed Requests	Complete Requests
10	13.285	0	1000
20	14.06	0	1000
30	14.123	0	1000
40	14.98	0	1000
50	14.889	0	1000
60	15.876	0	1000
70	15.276	0	1000
80	15.194	0	1000
90	15.787	0	1000
100	15.695	0	1000