Scalability Performance Report

Chelsea (Jiyuan) Lyu - jl1230, Shiyu Liu - sl846

Apr 8, 2024

1 Test on different cores CPU

To test the performance with different cores CPU, we use the following code:

```
# generate a CPU only use 1 core
sudo cgcreate -g cpu:/cpu_1
sudo cgset -r cpuset.cpus="0" cpu_1
# start and run the server on the 1-core CPU
sudo cgexec -g cpu:/cpu_1 ./server

# generate a CPU using 2 core
sudo cgcreate -g cpu:/cpu_2
sudo cgset -r cpuset.cpus="0,1" cpu_2
# start and run the server on the 2-core CPU
sudo cgexec -g cpu:/cpu_2 ./server

# generate a CPU using 4 core
sudo cgcreate -g cpu:/cpu_4
sudo cgset -r cpuset.cpus="0,1,2,3" cpu_4
# start and run the server on the 4-core CPU
sudo cgexec -g cpu:/cpu_4 ./server
```

Listing 1: Shell Commands for CPU Configuration

2 Test with different number of requests

To test with different numbers of requests, we write a Python script:

The py file generates and sends XML requests to a server, simulating concurrent client interactions. In the *scalability.py* file, we iterated through three distinct test cases, each with different requests: 10, 100, and 1000.

For each case, there are 10 parallel processes each responsible for sequentially sending its share of the total requests to ensure even distribution. This approach allowed us to measure the system's response time and throughput, quantitatively assessing its scalability under varying loads. The metrics collected include total execution time (s) and requests per second (req/s), offering insights into the system's performance scalability.

Num Of Cores	Average Total Time (seconds)	Average Throughput (requests/s)
1	0.03754701614	2683.264618
2	0.03385572433	2966.916236
4	0.03236045837	3124.154498

Table 1: The Time and Throughput when Num of Request = 10

Num Of Cores	Average Total Time (seconds)	Average Throughput (requests/s)
1	1.545272207	1546.107178
2	0.7460753441	4413.592708
4	0.9518029213	3066.290057

Table 2: The Time and Throughput when Num of Request = 100

Num Of Cores	Average Total Time (seconds)	Average Throughput (requests/s)
1	4.930967808	3865.982251
2	2.955434418	4381.320941
4	1.690238905	6308.936478

Table 3: The Time and Throughput when Num of Request = 1000

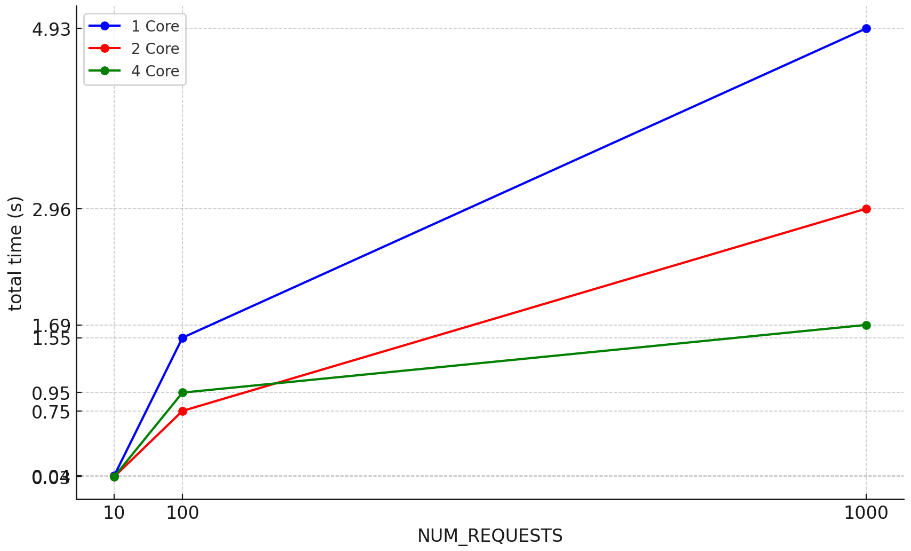

Figure 2: Average Execution Time Comparison

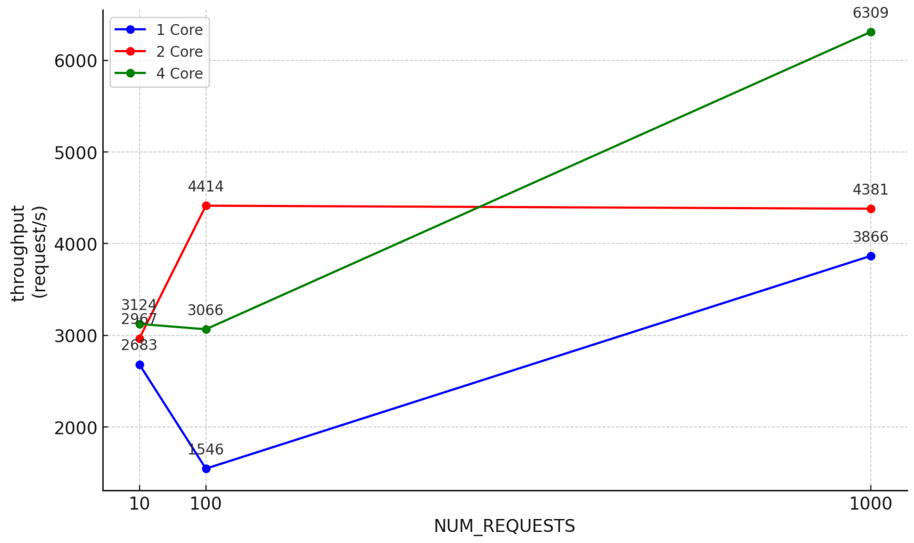

Figure 3: Average Throughput Comparison

3 Analyze & Conclusion

To analyze the performance, we ran each test 5 times and calculated their mean to present in the tables and graphs:

Table 1-3 presents the time and throughput for different numbers of requests (10, 100, 1000). From these tables, we can learn that the average throughput increases significantly while more cores are used for all cases of the number of requests.

Figure 2-3 presents the time and throughput for different numbers of requests (10, 100, 1000). From these figures, we can observe that the average execution time decreases significantly while more cores are used for all cases of the number of requests.

Therefore, we have enough evidence to support that if there are more cores on the CPU, our server is more efficient.

However, we can observe that the throughput for 100 requests is lower compared to 10 requests for both 1-core and 4-core, which is against our theoretical conclusion. This is because of that there could be a resource initialization phase—when dealing with 10 requests, the initialization overhead per request is less impactful than with 100 requests. What's more, this could be due to measurement noise since the data depends on a 5-times test, which is small data. We suggest testing more to calculate the stable performance.