Multi-threaded web server along with load testing

Niteen Arun Pawar (22M0800)

Load Generator and Load Testing for Web Server:
Project Overview: In this the task involves building a load generator and conducting a load test on a web server .The primary objective is to evaluate the web server's performance, robustness, and efficiency under varying levels of simulated user traffic. By subjecting the server to a significant number of requests, developers can identify potential bottlenecks, memory errors, and areas for optimization.

Load Generator:
The load generator is a software component designed to simulate a realistic distribution of user requests and generate a load on the web server. It generates and sends HTTP requests to the server, mimicking user interactions, and monitors the server's response time, throughput, and error rate. The load generator aids in measuring the server's performance metrics and uncovering potential issues. Load Testing: Load testing is the process of examining how a web server performs under different levels of simulated load. This entails subjecting the server to varying amounts of traffic to gauge its ability to handle concurrent requests without crashing, slowing down, or producing errors. Load testing provides insights into the server's scalability, stability, and resource consumption under stress conditions.
Valgrind and Memory Errors:
Before initiating the load test, it's essential to run Valgrind, a memory analysis tool, on the web server's code. Valgrind helps detect memory leaks, invalid memory access, and other memory-related issues that could potentially lead to crashes during load testing. By addressing these errors beforehand, developers ensure the server's stability and reliability during the test.

1. Load Generator Development: Create a load generator that generates HTTP requests with varying characteristics (GET, POST, etc.), simulating user interactions. This tool should be capable of controlling the request rate to replicate different levels of user load.
2. Load Test Execution: Execute the load test by subjecting the web server to the load generated by the load generator. Monitor metrics like response time, throughput, and error rate under different load levels.
3. Valgrind Analysis: Run Valgrind on the web server's code to identify and rectify memory errors and leaks. This step ensures the server's stability before load testing.
4. Performance Optimization: Based on load test results, identify performance bottlenecks and areas for improvement in the web server code. Optimize the code as necessary to enhance scalability and responsiveness.

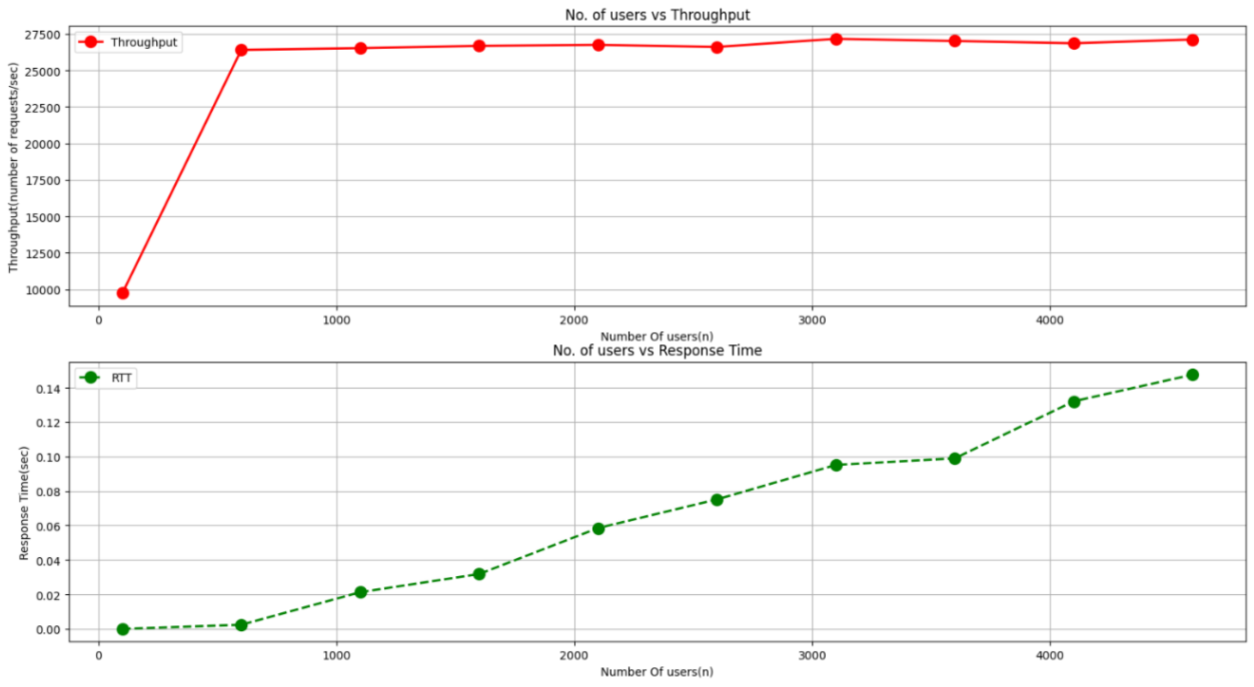
Response Time:

Response time, also known as latency or turnaround time, refers to the time it takes for a system, in this case, a web server, to respond to a user request. It includes the time it takes for the server to receive the request, process it, and send back the corresponding response to the user. Response time is a crucial metric in evaluating the user experience and the performance of a web server. A lower response time generally indicates a more responsive and efficient system, while a higher response time can lead to user frustration and may indicate performance issues.

Throughput:

Throughput is a measure of the number of tasks or transactions that a system can handle within a given time period. In the context of a web server, throughput refers to the number of requests the server can process and respond to in a certain amount of time. It is often expressed as requests per second (RPS) or transactions per second (TPS). Higher throughput indicates a system's ability to handle a larger volume of requests concurrently. Throughput is a critical metric for evaluating a server's capacity and scalability, as it provides insights into how well the system can handle increased loads without degrading performance.

Throughput And Response Time Graph:



The saturation of throughput in a load generator after some time is typically due to various factors related to the load generator itself, the system being tested, and the network environment. Here are some reasons why throughput might saturate:

Resource Limitations: Load generators, just like any other software, have limitations in terms of the hardware resources they can utilize. This includes CPU, memory, and network bandwidth. Once the load generator starts utilizing these resources to their maximum capacity, its ability to generate and send requests might become constrained, leading to a saturation in throughput.
Network Bottlenecks: The network connection between the load generator and the system being tested can become a bottleneck. If the network's bandwidth or latency becomes a limiting factor, the load generator might

not be able to send requests at a faster rate, leading to saturated throughput.

Why response time increases after sometime?

Resource Exhaustion: Load generators consume system resources such as CPU, memory, and network bandwidth. As the load generator runs over time, it might gradually exhaust these resources, leading to performance degradation. High CPU utilization or memory leaks, for example, can cause delays in generating and sending requests, resulting in increased response time.

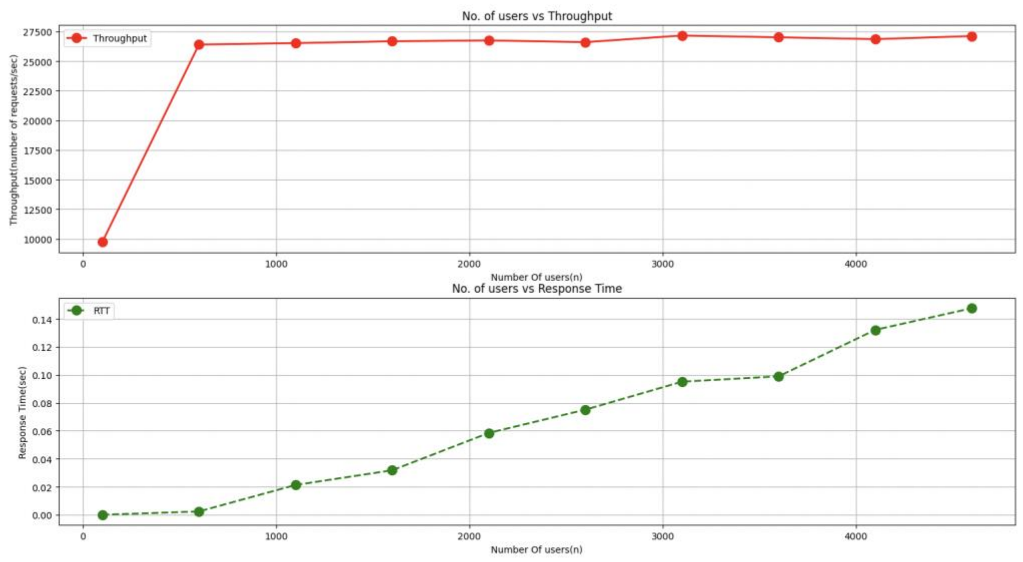
RESULTS:

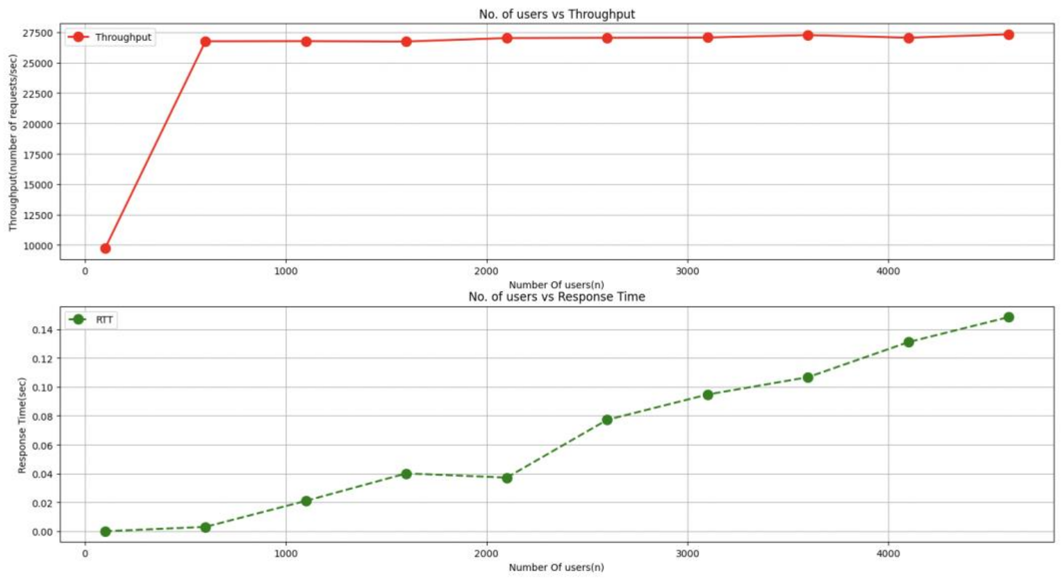
System Configuration:

Intel i5
8 core

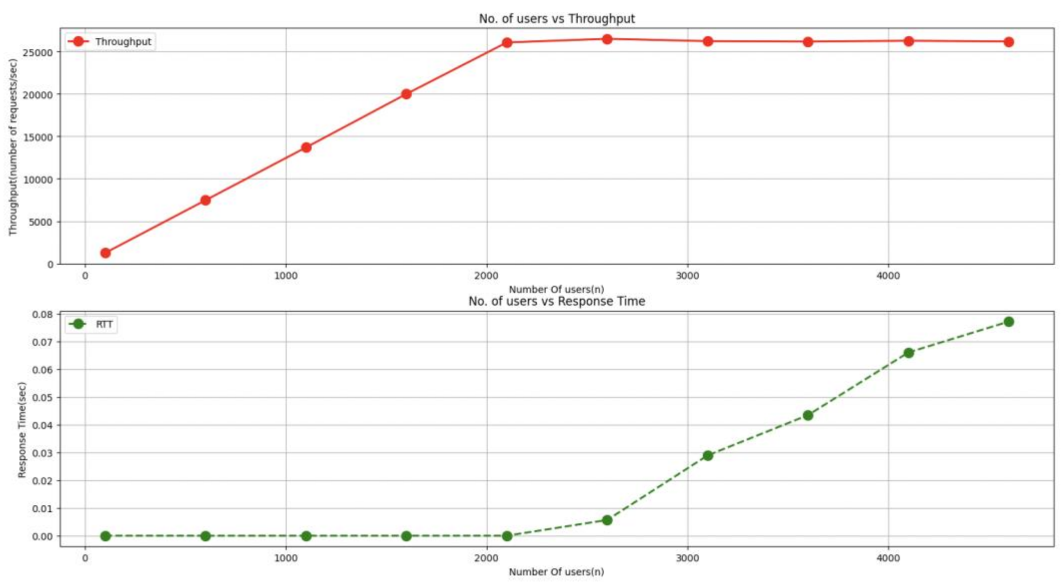
System configurations used

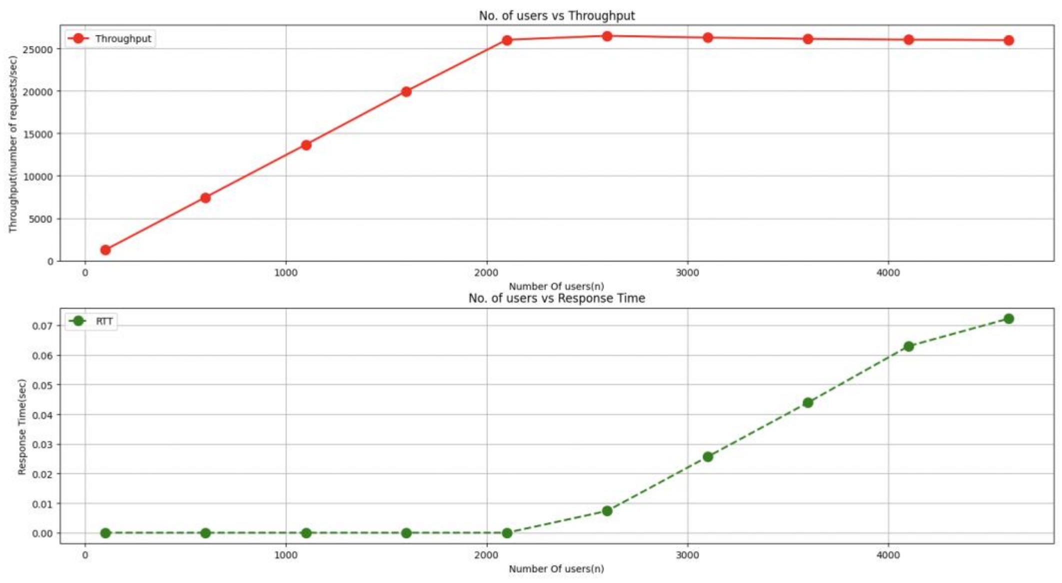
- Server set to core 0 of cp
- Load Generator set to core 1,2,3 of cpu





Think time of 0.01s ,No. of users incremented by 500 in each iteration upto 5100,test duration of 20 sec





Think time of 0.08s ,No. of users incremented by 500 in each iteration upto 5100,test duration of 30 sec

By using htop we are able to see that when we run the serve on a single core using taskset then that core gets utilised near to 100 but when we do not do taskset then the server utilizes remaining free cores and we do not get a flat line after a fixed interval of requests due to which we can conclude that our CPU is the bottleneck .