International Journal of Scientific Research in Computer Science, Engineering and Information Technology
ISSN : 2456-3307 (www.ijsrcseit.com)
doi : https://doi.org/10.32628/CSEIT2174127

Lightron : A GUI Integrated, Rust Based Web Server

Apurva Solanki[1], Aryan Parekh[1], Gaurav Chawda[1], Mrs. Geetha S.[2]

[1]Diploma Student, Department of Computer Engineering, Shri Bhagubhai Mafatlal Polytechnic, Mumbai, Maharashtra, India

[2]Lecturer, Department of Computer Engineering, Shri Bhagubhai Mafatlal Polytechnic, Mumbai, Maharashtra, India

Article Info
Volume 7, Issue 4
Page Number: 554-560

Publication Issue :
July-August-2021

Article History
Accepted : 07 Aug 2021
Published : 14 Aug 2021

ABSTRACT

Day by day, the number of users are increasing on the internet and the web servers need to cater to the requests constantly, also if compared to the past years this year, due to a global pandemic and lockdown in various countries, the requests on the web have surged exponentially. The complexity of configuring a web server is also increasing as the development continues. In this paper, we propose a Lightron web server, which is highly scalable and can cater many requests at a time. Additionally, to ease users from the configuration of the web server we introduced Graphical User Interface which is beginner friendly.

Keywords : Web Server, Concurrency, Rust, GUI, Asynchronous, Non-Blocking, HTTP.

I. INTRODUCTION

Whenever a user accesses the web, he/she uses an HTTP client program called web browser to fetch contents from what's called a web server which is an HTTP server program. HTTP client and server normally use TCP as transport layer protocol. Whenever a user visits a particular website, the browser will send an HTTP GET request with some metadata stored in HTTP request headers to the web server. The web server will parse the request and prepare the HTTP response. If the server doesn't find the requested content then it will respond with a 404 status code, alternatively, if the server finds the requested content it will respond with a 200 status code. There are many types of status codes i.e. Informational, Successful, Redirects, Client errors, Server errors. websites can also be dynamically generated which is integrated with some database engines, but in this paper, we'll be going to limit our discussion to static websites only.

Traffic on the web servers is increasing rapidly due to many clients connecting to the web server in such a short time interval. That's why the server must cope up with many requests at a time, therefore support for concurrency and parallelism is indispensable. There are many concurrency models available but each one has its drawbacks. Configuring a web server is also a tedious task and if not configured properly it can be vulnerable to security attacks.

Considering the above problems, In this paper, we introduce a Lightron web server that can handle many requests at a time by utilizing multiple cores of the processor and also make use of the non-blocking

IO operations provided by the operating system. We have also solved the complication of configuration by introducing a lightweight, user-friendly GUI.

II. BACKGROUND

A. HTTP Protocol

Hyper Text Transfer Protocol is one of the most widely used web protocols. It is a stateless protocol in which each transaction is independent of all other transactions. There are HTTP clients which in most of the cases are web browsers who initialize the transactions by sending HTTP requests which are replied to by the HTTP server's response. Mainly there are 2 types of HTTP requests: GET & POST. GET is used to fetch content from the server and POST is used to submit content to the server. In HTTP requests, there are numerous headers like the method: which is used to indicate whether request type is GET or POST, URI or HOST: which is used to indicate server's domain name and in some cases requested file, version: which is used to indicate HTTP protocol versions. On the other hand, HTTP response contains headers like status code: which is used to denote the result of transaction, content-type: specifies what kind of file is transferred, server: which contains server name and version.

There are multiple versions of HTTP protocol [1] namely HTTP/1.0, HTTP/1.1, HTTP/2, HTTP/3 in which HTTP/3 is still in the development phase. HTTP/1.1 supports pipelining, virtual hosting, and chunked responses. HTTP/2 is faster than HTTP/1.1 and supports additional features such as server push and multiplexing.

B. HTTP over TLS (HTTPS)

For Encrypting transactions between client and server, HTTPS is used. Secure HTTP or HTTPS is achieved using Transport Layer Security protocol. For RSA-based encryption process, The client first sends a ClientHello message with some random bytes called "Client Random" and the server replies with a ServerHello message with some random bytes called "Server Random" and an SSL certificate for authentication purposes. Now the client creates a "premaster secret" which is also random bytes but encrypted with the server's public key and sends it to the server. The server then decrypts the premaster secret and both client and server create a session key based on client random, server random, and premaster secret. Now all the communication between client and server will be encrypted with a session key. There is another encryption process called Diffie-Hellman which is moderately different than RSA.

C. Virtual Hosting

Hosting every single website with an individual server is quite expensive and inefficient. That's why support for virtual hosting is necessary. Through virtual hosting one can host many websites using a single web server. There are mainly 2 types of virtual hosting, Port-based & Domain Name based. Using port-based virtual hosting one can utilize multiple ports for each website hosted on a single machine and therefore only a single IP address is used. In domain-based virtual hosting, a single port and IP address is utilized, and the domain name of the website is used to distinguish between multiple websites. There is also an IP-based virtual hosting to host multiple websites on a single machine but utilizing multiple IP addresses.

D. Rust

The web servers which are popular in the software industry are mostly developed using the C programming language. Although C has lower overhead and is blazingly fast, it is not memory safe and many bugs arise while developing concurrent applications. Therefore, Mozilla has developed the Rust programming language [2] as an alternative.

In modern High-level programming languages such as Python and Java, garbage-collector is used to ensure

memory safety, but garbage-collector often comes with a price of performance drop. Rust uses an elegant approach for memory safety without using garbage-collector by using "Ownership & Borrowing". In Rust, each value has only one owner and when the owner goes out of scope the value will be discarded. There can be multiple shared references of the value but only one mutable reference. Also, there can be a transfer of ownership called move.

Rust also supports all modern features that high-level programming languages have such as Object-oriented programming, functional programming, and pattern matching. It also has an efficient C foreign function interface.

III. Working and Implementation

Lightron web server is developed to ease the configuration of websites in development environments as the other web server in the industry only supports terminal-based configuration which can be time-consuming for beginners. On the other hand, Lightron supports Graphical User Interface using which one can easily host websites in a development environment and focus more on website development rather than server configuration. The Lightron web server consists of two major components namely

- Lightron Core
- Lightron GUI

A. Lightron Core

This module represents the functional part of the Lightron web server. The function of this module is to first read the configuration file generated by Lightron GUI or the user itself. Then according to the configuration file, the server will allocate resources for each website mentioned.

Lightron web server is compatible with both HTTP and HTTPS protocol. The normal unencrypted HTTP protocol is served using the HTTP/1.1 version whilst encrypted HTTPS protocol is powered by a faster HTTP/2.0 version. In the Rust ecosystem, there are various crates to extend the functionality of the program. In the case of Lightron Hyperium H2 crate is used to implement HTTP/2.0 specification. Also, rustls crate provides support for transport layer security or the TLS handshake process. By using rustls Lightron can perform TLS version 1.3 handshake which reduces the time it takes to encrypt connections. Earlier TLS version 1.2 required two roundtrips to complete the TLS handshake but in version 1.3 that was reduced to only one round-trip [3]. Lightron also uses a TLS extension called Application-Layer Protocol Negotiation or ALPN [4] to specify which protocol version will get priority for further HTTPS communication; this will negotiate HTTP/2 communication directly in the TLS handshake itself rather than negotiating it later and wasting one extra roundtrip for it as in Connection-Upgrade process.

As described in the introduction section, Lightron supports both port-based, and domain name-based virtual hosting. In port-based virtual hosting Lightron creates separate kernel level threads for each website, this further isolates the behavior of individual websites and can utilize multi-core processors by spreading the load of the website across multiple cores.

Now in domain-based virtual hosting, For HTTP communication, the Server will make use of the HOST header in the request to identify multiple domain names and For HTTPS communication, the Server will make use of TLS Extension called SNI or Server Name Indication to distinguish SSL certificate and Private key for each virtually hosted website and select SSL Certificate and Private key in runtime so that each virtually hosted website can have its distinct SSL Certificate and Private key for secure communication.

Lightron can also utilize one of the important features of HTTP/2 which is Server-Push [5] [6]. In Server-Push, the server can send the web resources without the client requesting it. Using this the server does not have to wait for the client's request therefore it

reduces the latency and increases the throughput as in one request server is sending selected necessary dependencies for that file. But the Server-Pushed files need to be chosen wisely otherwise If the Server-Pushed file is not required by the client then there can be wastage in bandwidth.

B. Lightron GUI

This module is isolated from the Lightron core as this module contains different elements of GUI which is independent of the server. As we discussed above the web server configuration can be intimidating for beginners, GUI-based configuration can be helpful for them and therefore the main task of GUI is to generate the configuration file. The configuration file is the bridge between Lightron core and Lightron GUI. Popular web servers that are present in the industry use complex file formats for configuration files. The file formats like XML, JSON, YAML are complicated and error-prone that's why Lightron uses TOML (Tom's Obvious Minimal Language) file format which is pretty easy to understand and parse [7].

Lightron GUI is developed using FLTK or Fast Light ToolKit framework in Rust Programming language which makes it lightweight and the memory footprint of the GUI is also very low between 45-50MB only. Also, FLTK produces a very small binary around 5MB so including it in the setup file will not make a huge difference. Because FLTK & Rust is cross-platform the GUI and Whole web Server supports major operating systems.

GUI can also be useful for monitoring web server's status like current CPU load, current memory usage. GUI can also be used for troubleshooting a web server by reading or changing log levels to more verbose so one can pinpoint an exact issue. By using GUI, the user does not even need to see the configuration file as the GUI lists all the hosted websites and their parameters based on the configuration file.

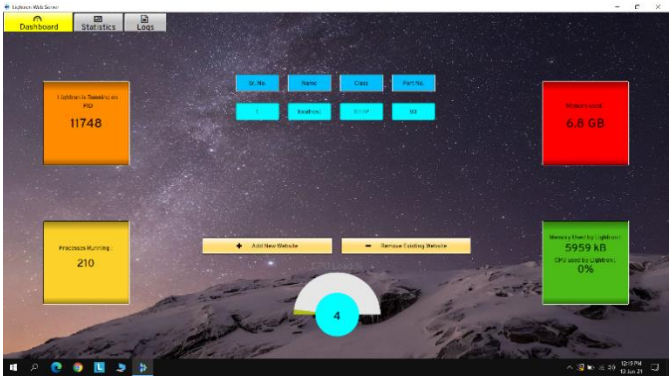

Figure 1. Lightron Web Server GUI Dashboard

C. Concurrency Model

There are various concurrency models or design patterns through which we can handle multiple requests simultaneously.

Let's start with a simple approach which is the Threadpool model. In this model, the group of spawned threads are ready to handle requests and are waiting for them. When a request arrives one of the free threads will be assigned to serve that request. In this way, we can parallelly serve multiple requests based on the number of threads spawned at the beginning and increase the throughput of the server. But this model comes with a drawback which is the number of threads spawned is fixed and spawning threads also comes with the cost of operating system overhead. Also, there can be a wastage of computation resources if the number of requests is far less than the number of threads because in this case, most threads will be in a waiting state doing nothing.

Then there is an event-driven approach, where servers can serve multiple requests using non-blocking IO operations that are provided by the operating systems. In this method only a single thread is used which is an event scheduler whose job is to accept the connection, read the requested file and send that back to the client but all these operations are asynchronous and non-blocking meaning that the thread does not have to wait for the completion of the event instead it will ask the operating system to notify on the completion of the event. The major constraint of this method is that the operating system must

support non-blocking IO operations. Although many operating systems like Linux and Windows support non-blocking network IO; they lack support or have very experimental support for non-blocking disk IO and because of that in some cases, the task becomes synchronous and blocking resulting in degraded performance.

Lightron web server is based on the Tokio framework [8] which is a runtime for writing asynchronous, event-driven, and non-blocking applications in the Rust programming language. Tokio is based on AMPED or Asymmetric Multi-Process event-driven architecture as described in [9]. This architecture is an extension of the above described event-driven architecture and resolves major constraints of it by introducing worker threads. By assigning blocking or synchronous tasks to the worker thread, the event scheduler will be free to assign other upcoming requests to available worker threads this way we can achieve maximum throughput and efficiency. The default number of worker threads is equal to the number of cores available in the working system. Tokio is not limited to AMPED architecture because it also supports other features like a work-stealing queue in which load amongst worker threads will be distributed uniformly and therefore all the cores of the CPU will be equally utilized.

IV. RESULTS AND DISCUSSION

To experiment and benchmark Lightron web-server and Apache (Popular web-server in the industry) [10], We have used AB or Apache Benchmark tool [11] to load test and stress test web-servers in the host system. The host system has the following configuration, AMD E300 Dual Core CPU running a 1.3GHz, 4GB RAM with the frequency of 800MHz running Arch Linux with Kernel 5.10.43. All the results below are dynamic and may vary depending on various factors like configuration, room temperature, and OS.

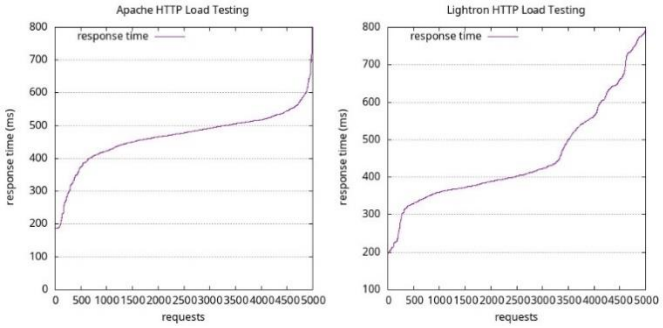

Figure 2. Graphical comparison of response time with respect to HTTP requests

First, with Fig. 2 we are flooding both the servers with 5000 unencrypted HTTP requests but at a time 500 requests are generated parallelly and sent to the server therefore 10 batches of 500 concurrent HTTP requests are sent to the server. By using this test, we can understand how the server behaves when many concurrent requests are coming and how the response time of the server increases in that period.

By analyzing the above graphs, we can see that when the number of requests reaches 750-800, the response time of Apache web is increasing rapidly around 400ms. On the other hand, the Lightron web server has a response time of 340ms which is comparatively less than the Apache web server.

When the number of requests goes from 1000 to 3500, the response time of the Apache web server goes from 410ms to 500ms but in the same scenario, the response time of the Lightron web server goes from 340ms to 450ms which is also lesser than Apache web server.

When requests rise above 3500 the Lightron web server's response time increases faster than Apache web server but at 5000 requests both the server has 800ms of response time.

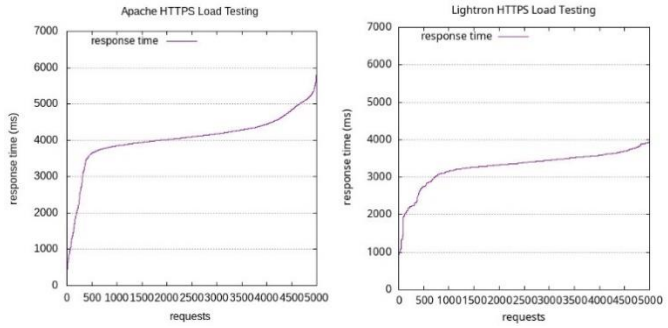

Figure 3. Graphical comparison of response time with respect to HTTPS requests

In the previous benchmark, we have only tested unencrypted HTTP protocol which is unlikely to be used in real-world scenarios therefore In this test we have used Encrypted HTTPS protocol which most organizations and companies are using because of security concerns and search engine indexing priorities. The benchmark setup is identical to the above test only difference is that both the web servers are configured to use HTTPS protocol with their default parameters.

By analyzing the above graph results, In Apache web server as the requests rise from 0 to 500 subsequent response time rapidly rises from 0ms to 3800ms; moreover, in Lightron web server the response time rises rapidly only from 0ms to 2700ms which is better than Apache web server. In the Apache web server, there is a steady incline in the response time about 2700ms to 4300ms for requests between 500 to 4000 and after that, there is a rapid increase in response time. In contrast, the Lightron web server's response time steadily grows from 3100ms to 4000ms for requests ranging between 1000 to 5000.

The peak response time for Apache is 5700ms and for Lightron it is 4000ms.

V. ACKNOWLEDGEMENTS

We appreciate Mohammed Alyousef for giving advice and suggestions to implement Graphical User Interface for the Lightron web server. Also, a shoutout to Rust & Tokio Documentation team for making helpful guides. We are also grateful to Apache Software Foundation for letting us use their products such as Apache Web Server and Apache Benchmark tool. Additionally, we want to thank unknown reviewers for making this paper more readable and understandable.

VI. CONCLUSION & FUTURE SCOPE

This paper Introduces a new web server called Lightron which is developed using Rust Programming Language as opposed to a standard C programming language which is used by prominent web servers in the industry. Rust's memory safety, Scalability, and Speed make the Lightron web server robust and its performance as good as its counterpart. Using Tokio's Asynchronous, Event-driven, and work-stealing architecture, Lightron achieves maximum concurrency and efficiently manages the system's resources. By integrating Graphical User Interface with Lightron, beginners can easily configure a web server in their development environment and focus more on website development.

By implementing CGI or Common Gateway Interface, we can integrate PHP to the web server and server dynamic content over the web. The io_uring subsystem which was released in Linux Kernel version 5.1 provides a new interface for Asynchronous Disk Operations which is highly efficient and non-blocking, By using this we can achieve more concurrency and throughput in our web server. Windows is yet to release their implementation of io_uring but they will release in 21H2 as described in this [12]. HTTP/3 is an upcoming version of the HTTP protocol which will use QUIC-UDP instead of TCP as transport layer protocol and provides a significant performance boost in communication between client and server. By

using it the server's response time and latency will be reduced.

VII. REFERENCES

[1]. Abdullah, S. A., & Ahmad, A. M. (2016). HTTP/2 in Modern Web and Mobile Sensing based Applications Analysis, Benchmarks and Current Issues. J Electrical & Electronic System, 5, 193.

[2]. Fulton, K. R., Chan, A., Votipka, D., Hicks, M., & Mazurek, M. L. (2021). Benefits and Drawbacks of Adopting a Secure Programming Language: Rust as a Case Study. In Seventeenth Symposium on Usable Privacy and Security (SOUPS 2021) (pp. 597-616) USENIX.

[3]. Ralph Holz, Jens Hiller, Johanna Amann, Abbas Razaghpanah, Thomas Jost, Narseo Vallina-Rodriguez, and Oliver Hohlfeld. (2020). Tracking the deployment of TLS 1.3 on the web: a story of experimentation and centralization. SIGCOMM Comput. Commun. Rev. 50, 3 (July 2020), 3–15. DOI:https://doi.org/10.1145/3411740.3411742

[4]. Shoemaker, R. B. (2020). RFC 8737 Automated Certificate Management Environment (ACME) TLS Application-Layer Protocol Negotiation (ALPN) Challenge Extension.

[5]. Rosen, S., Han, B., Hao, S., Mao, Z. M., & Qian, F. (2017, April). Push or request: An investigation of HTTP/2 server push for improving mobile performance. In Proceedings of the 26th International Conference on World Wide Web (pp. 459-468).

[6]. Zimmermann, T., Rüth, J., Wolters, B., & Hohlfeld, O. (2017, June). How HTTP/2 pushes the web: An empirical study of HTTP/2 server push. In 2017 IFIP Networking Conference (IFIP Networking) and Workshops (pp. 1-9). IEEE.

[7]. Jonathan, H. (2020, July). The 3 Best Config File Formats. In https://jhall.io/posts/best-config-file-formats.

[8]. Tokio-rs, (2021). Tokio Runtime. In https://tokio.rs.

[9]. Pai, V. S., Druschel, P., & Zwaenepoel, W. (1999, June). Flash: An efficient and portable Web server. In USENIX Annual Technical Conference, General Track (pp. 199-212).

[10]. Apache Software Foundation, (2020). Apache HTTP Server Project (httpd). https://httpd.apache.org.

[11]. Apache Software Foundation, (2020). ab-Apache HTTP server benchmarking tool. https://httpd.apache.org/docs/2.4/programs/ab.html.

[12]. Yarden S. (2021, May). I/O Rings – When One I/O Operation is Not Enough. In https://windows-internals.com/i-o-rings-when-one-i-o-operation-is-not-enough

Cite this article as :

Apurva Solanki, Aryan Parekh, Gaurav Chawda, Mrs. Geetha S., "Lightron : A GUI Integrated, Rust Based Web Server", International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), ISSN : 2456-3307, Volume 7 Issue 4, pp. 554-560, July-August 2021. Available at

doi : https://doi.org/10.32628/CSEIT2174127

Journal URL : https://ijsrcseit.com/CSEIT2174127