(2025), 3 (4): 787- **Scientica** 3021-8209
Jurnal Ilmiah Sain dan Teknologi

# ANALISIS PERFORMA *WEB SERVER* RUST DAN GO PADA PROTOKOL *HYPERTEXT TRANSFER PROTOCOL* (HTTP)

**Jander Supardi [1], Heri Priyanto [2], Alfian Abdul Jalid [3]**
Fakultas Teknik, Universitas Tanjungpura
janderloc12@gmail.com

**Abstract (English)**
Web servers play a crucial role in serving requests from client browsers by delivering web content to users. This study analyzes the performance of two web servers, Rust and Go, in handling HTTP requests. The aim of the research is to compare the performance of the two web servers in terms of resource usage (CPU and memory), speed, stability, and scalability. Testing was conducted using the load testing method with Apache JMeter, simulating user loads ranging from 200 to 1000 users. The results show that Rust is more efficient in resource utilization compared to Go. At low loads (200–400 users), Go has lower CPU usage, but at high loads (600–1000 users), Rust outperforms with significantly lower CPU usage. For example, at a load of 600 users, Rust only uses 0.47% CPU, while Go reaches 5.21%. In all test scenarios, Rust is also more memory-efficient, using 52.28 MB at a load of 800 users, compared to Go, which consumes 267.01 MB. In terms of speed, Go performs more optimally than Rust based on throughput and processing time metrics. A notable difference is observed at a load of 600 users, where Go achieves an average throughput of 60.043 transactions per second (tps), while Rust only reaches 8.047 tps. Go also has lower processing times compared to Rust, with a significant difference at a load of 800 users, where Go records 173.47 ms, while Rust reaches 7,408.53 ms. In terms of stability, Go also outperforms Rust, as indicated by a lower average coefficient of variation (CV). For example, at 200 users, Go's coefficient of variation is 29.11%, lower than Rust's 277.63%. Rust excels in scalability, with a throughput increase of 46.057% at 1000 users, compared to Go's 2.074%. In conclusion, Go is more optimal for low loads and applications requiring speed, while Rust is better suited for high loads and applications requiring resource efficiency.

**Article History**
Submitted: 12 April 2025
Accepted: 15 April 2025
Published: 16 April 2025

**Key Words**
web server, Rust, Go, HTTP, performance, load testing

**Abstrak (Indonesia)**
*Web server* memegang peran penting dalam melayani permintaan dari *browser client* dengan mengirimkan konten web kepada pengguna. Penelitian ini menganalisis performa dua *web server*, yaitu Rust dan Go, dalam menangani permintaan HTTP. Tujuan penelitian adalah membandingkan performa kedua *web server* dalam hal penggunaan sumber daya (CPU dan memori), kecepatan, stabilitas dan skalabilitas. Pengujian dilakukan menggunakan metode *load testing* dengan Apache JMeter, dengan simulasi beban pengguna mulai dari 200 hingga 1000 pengguna. Hasil penelitian menunjukkan bahwa Rust lebih rendah dalam pemanfaatan sumber daya dibanding Go. Pada beban rendah (200–400 pengguna) Go lebih rendah dalam penggunaan CPU, namun pada beban tinggi (600–1000 pengguna) Rust unggul dengan penggunaan CPU yang lebih rendah. Sebagai contoh, pada beban 600 pengguna Rust hanya menggunakan 0,47% CPU sementara Go mencapai 5,21%. Dalam semua skema pengujian Rust juga lebih hemat memori, dengan penggunaan 52,28 MB pada beban 800 pengguna, dibandingkan Go yang mengonsumsi 267,01 MB. Dari segi kecepatan Go memiliki performa lebih optimal dibanding Rust berdasarkan nilai *throughput* dan *processing time*. Perbedaan mencolok pada beban 600 pengguna Go memiliki rata-rata *throughput* 60,043 transaksi per detik (tps) sedangkan Rust hanya mencapai 8,047 tps. Go juga memiliki waktu pemprosesan yang lebih rendah dibanding Rust perbedaan menonjol pada beban 800 pengguna dimana Go memiliki nilai 173,47 ms sedangkan Rust 7.408,53 ms. Dari segi stabilitas, Go juga lebih unggul dari Rust ditunjukan rata-rata nilai koefisien variasi (CV) yang lebih rendah. Sebagai contoh pada 200 pengguna nilai koefisien variasi Go sebesar 29,11% lebih rendah dibanding Rust yang mencapai 277,63%. Rust lebih unggul dalam skalabilitas, dengan peningkatan throughput sebesar 46,057% pada 1000 pengguna,

**Sejarah Artikel**
Submitted: 12 April 2025
Accepted: 15 April 2025
Published: 16 April 2025

**Kata Kunci**
*web server*, Rust, Go, HTTP, Performa, *load testing*

dibandingkan Go yang hanya 2,074%. Kesimpulannya, Go lebih optimal untuk beban rendah dan aplikasi yang memerlukan kecepatan, sementara Rust lebih cocok untuk beban tinggi dan aplikasi yang memerlukan efisiensi sumber daya.

## 1. PENDAHULUAN

*Web server* memegang peranan penting untuk melayani *request* dari *browser client* seperti google chrome, firefox, safari dan *software browser* lainnya lalu menampilkan hasil dari proses berupa data-data yang dibutuhkan *client* kembali ke *web browser* (Amrullah, Nugroho, and Ramadhan 2023). Hingga April 2024, terdapat 5,44 miliar pengguna internet di seluruh dunia, yang setara dengan 67,1 persen dari populasi global (Statista 2024). Berdasarkan survei yang dilakukan pada Juli 2024, terdapat lebih dari 1,1 miliar situs web yang aktif dan terhubung ke internet (Netcraft 2024). *Hypertext Transfer Protocol* (HTTP) merupakan salah satu protokol komunikasi yang paling banyak digunakan di dunia. Berdasarkan survei yang dilakukan juli 2024, protokol default http digunakan oleh 86,0% dari semua situs web (W3techs 2024).

Rust dan Go adalah dua bahasa pemrograman yang relatif baru dan sedang naik daun dalam pengembangan *web server*. Go pada juli 2023 berada pada peringkat ke-13, sekarang pada juli 2024 berada pada peringkat ke-7. Rust melonjak dari posisi 17 pada juli 2023 ke posisi 13 pada juli 2024 (TIOBE 2024). Sebagai *web server* golang meiliki keungulan dari segi kecepatan, skalabilitas dan pemanfaatan *memory* (Pragestu, Sujaini, and Faja 2023). Rust merupakan proyek *open source*, yang berarti dapat digunakan secara gratis dan diubah sesuai kebutuhan. Dari hal perfoma Rust menawarkan layanan yang cepat dan optimal dalam pengelolaan memori sehingga sangat cocok untuk aplikasi yang berkinerja tinggi. Rust sering digunakan untuk pengembangan sistem dan perangkat lunak yang memerlukan kontrol tinggi terhadap sumber daya perangkat keras (Steve Klabnik 2023).

Tujuan dari penelitian ini adalah untuk membandingkan performa *web server* Go dan Rust dalam menangani permintaan klien pada protokol HTTP, termasuk kecepatan, penggunaan sumber daya, stabilitas dan skalabilitas.

## 2. METODE PENELITIAN

Pengumpulan data awal pengujian *web server* dalam penelitian ini dilaksanakan dengan cara melakukan pengumpulan informasi dari literatur-literatur seperti jurnal, buku, dan literatur penelitian lainnya yang berkaitan dengan metode-metode penelitian yang akan dilaksanakan dalam pengujian performa *web server*.(Ridwan et al. 2021).

## 3. HASIL DAN PEMBAHASAN

### Implementasi

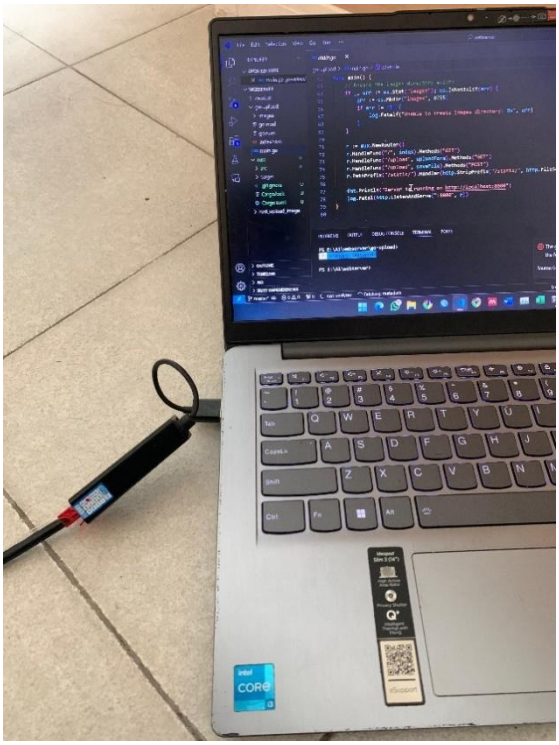
Setelah penjelasan dan pembahasan metodologi penelitian yang akan dilakukan, maka langkah berikutnya dalam proses penelitian adalah implementasi atau penerapan dari kerangka metodologi yang telah dirancang. Dalam tahap ini, dilakukan eksekusi langkah-langkah yang telah disusun untuk mengumpulkan data, menganalisis informasi yang relevan, dan menjalankan prosedur-prosedur yang telah ditetapkan untuk mencapai tujuan penelitian yang telah ditetapkan sebelumnya.

### Setup Perangkat *Client-Server* Pengujian

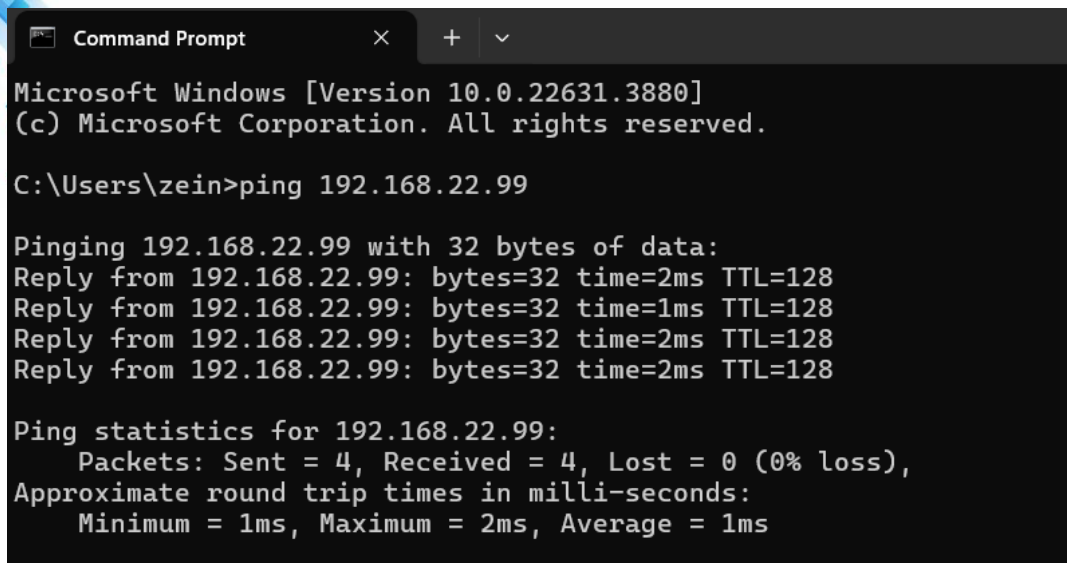
Tahap awal pada setup perangkat *client-server* dilakukan dengan instalasi perangkat lunak yang diperlukan untuk pengujian, instalasi ini dilakukan pada komputer *server* dan juga pada komputer *client* dengan masing – masing komputer menggunakan sistem operasi windows 11 dengan rincian spesifikasi yang telah dirincikan pada bab sebelumnya. Dalam tahapan setup perangkat *client*-server untuk pengujian, penelitian ini menggunakan topologi jaringan LAN, dimana topologi ini diimplementasikan untuk mengurangi permasalahan pada

kualitas jaringan selama proses pengujian, dimana permasalahan yang dapat dihindari seperti permasalahan *bandwith* dan *latency* hal ini disesuaikan seperti tahapan yang telah dibuat pada tahapan 3.3.3.1. Hasil implementasi topologi jaringan LAN yang diterapkan dalam pengujian dapat dilihat pada Gambar 3.1 yang menunjukkan perangkat klien yang digunakan untuk pengujian, Gambar 3.2 yang menunjukkan perangkat server yang digunakan pada pengujian, dan Gambar 3.3 yang menunjukkan hasil ping antara *client*-server yang menunjukkan hasil *latency* sebesar 1 ms.

*Gambar 3. 1 Komputer Client*

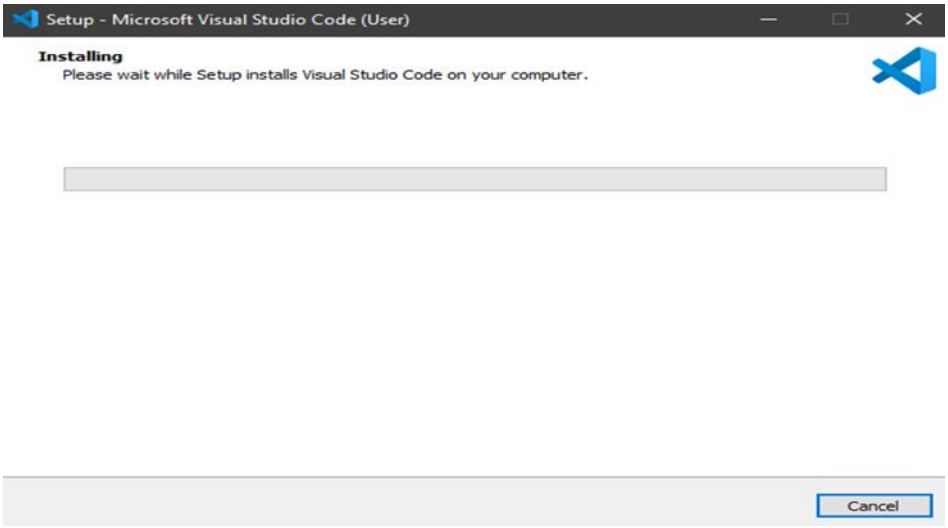


*Gambar 3. 2 Komputer Server*




*Gambar 3. 3 Hasil Ping dari Client ke Server*

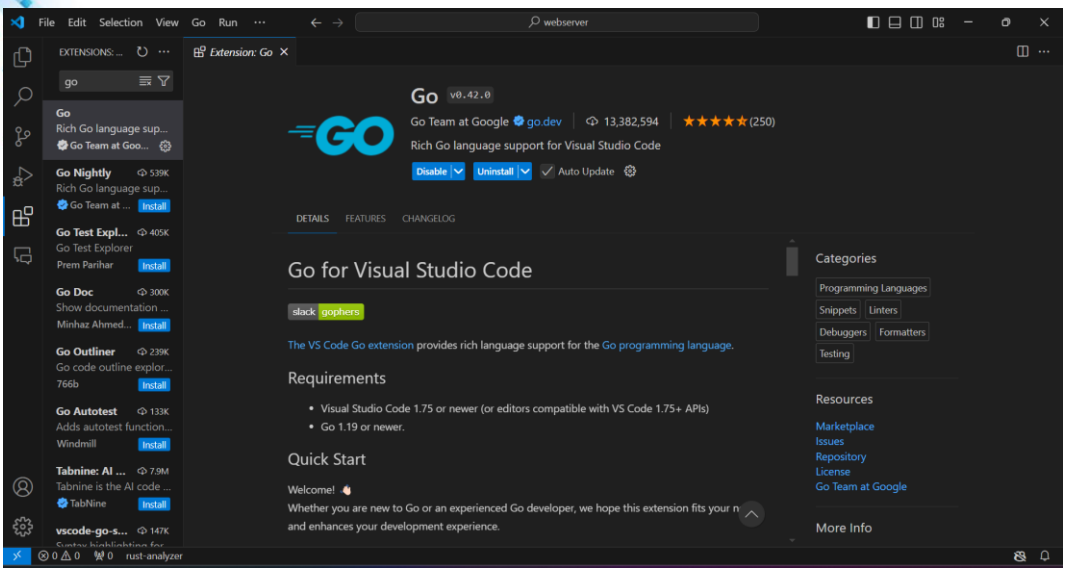
*Selanjutnya pada tahap setup perangkat client-server adalah melakukan instalasi perangkat lunak yang dibutuhkan untuk pengujian, perangkat lunak yang diimplementasikan pada pengujian ini disesuaikan seperti yang ditunjukkan pada poin 3.1.1. Sistem operasi yang digunakan pada perangkat client dan server adalah Windows 11, sistem operasi ini sendiri telah tersedia langsung dari perangkat.*




*Gambar 3. 4 Instalasi Visual Studio Code 1.92.0*

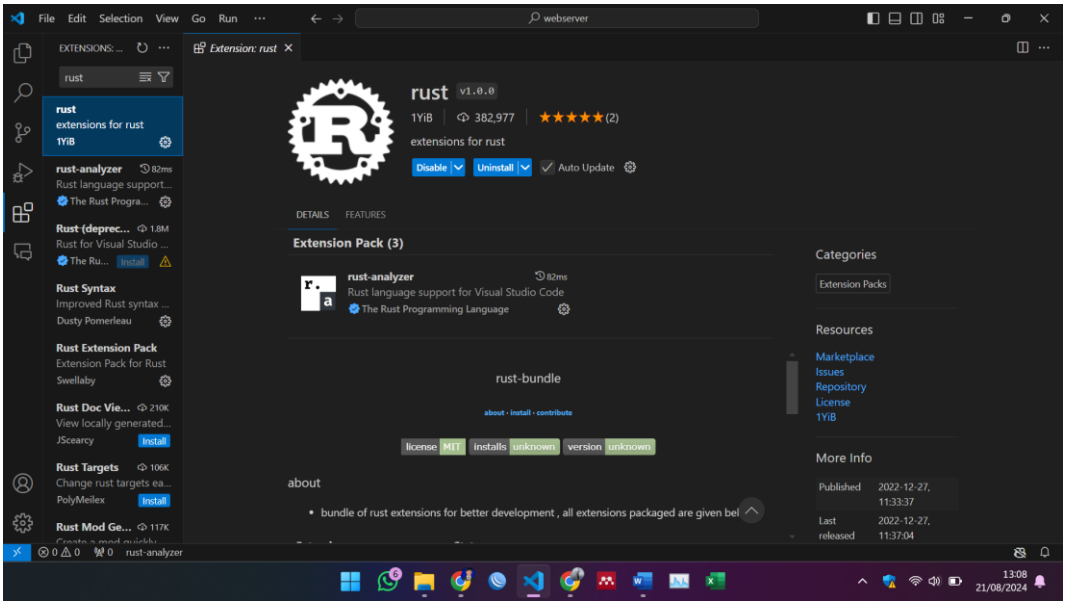
Gambar 3.4 menunjukkan instalasi Visual Studio Code 1.92.0 yang digunakan untuk merancang *web server* serta menjalankan *web server*.

(2025), 3 (4): 787- **Scientica** 3021-8209
Jurnal Ilmiah Sain dan Teknologi



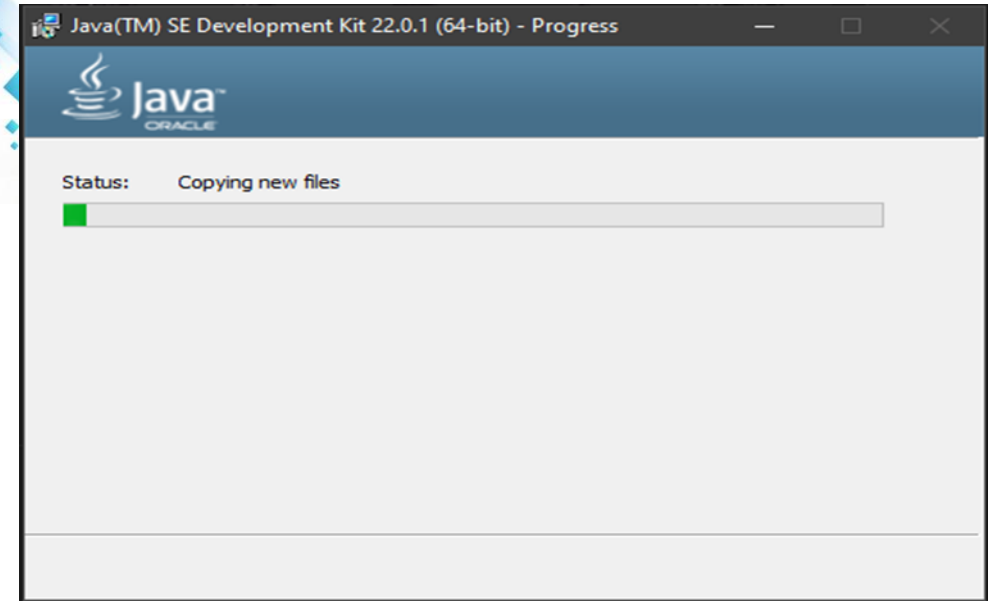

*Gambar 3. 5 Instalasi Go (Golang) v0.42.0*

Gambar 3.5 menunjukan hasil instalasi Go (Golang) vo.42.0 yang digunakan sebagai bahasa pemrograman pengembangan atau perancangan *web server* yang akan diujikan.



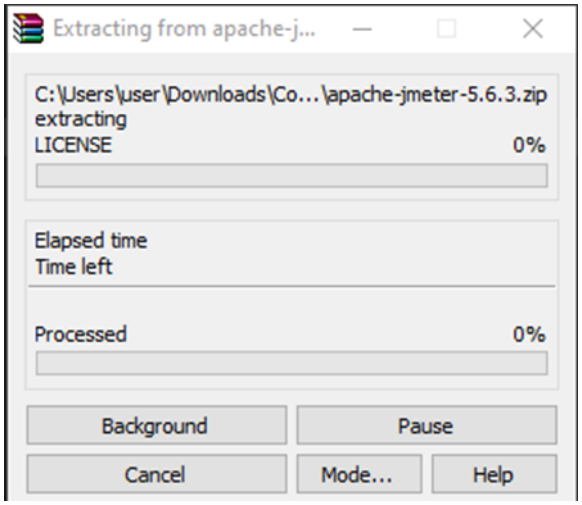

*Gambar 3. 6 Instalasi Rust v1.0.0*

Gambar 3.6 menunjukan hasil instalasi Rust v1.0.0 yang digunakan sebagai bahasa pemrograman pengembangan atau perancangan web server yang akan diujikan.



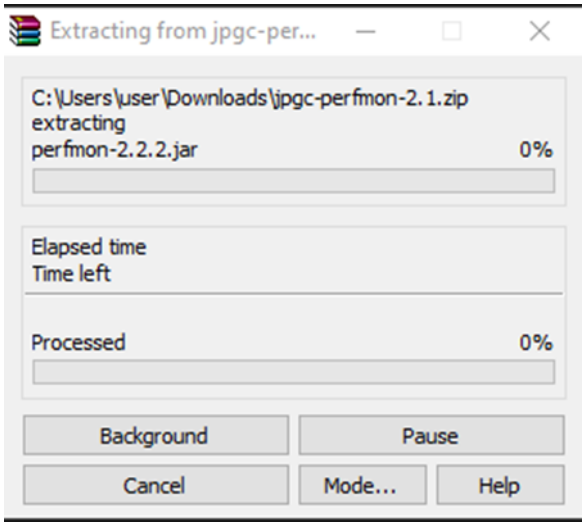

*Gambar 3. 7 Instalasi Java JDK 22.0.1*

Gambar 3.7 menunjukkan hasil instalasi Java JDK 22.0.1 yang digunakan menjalankan perangkat lunak Apache Jmeter.



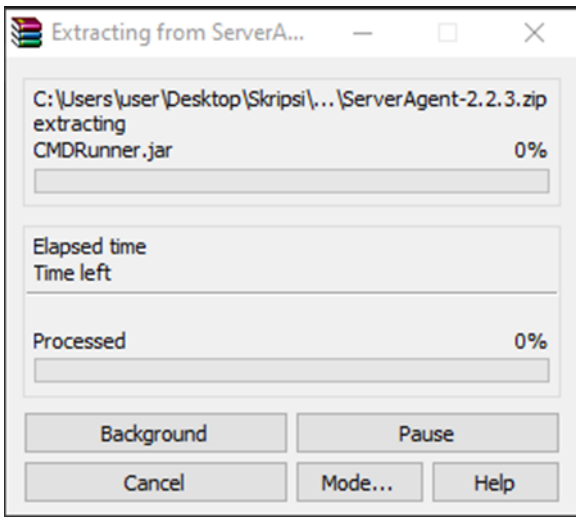

*Gambar 3. 8 Instalasi Apache Jmeter 5.6.3*

Gambar 3.8 menunjukkan hasil instalasi Apache Jmeter 5.6.3 yang digunakan sebagai perangkat lunak pengujian web server.



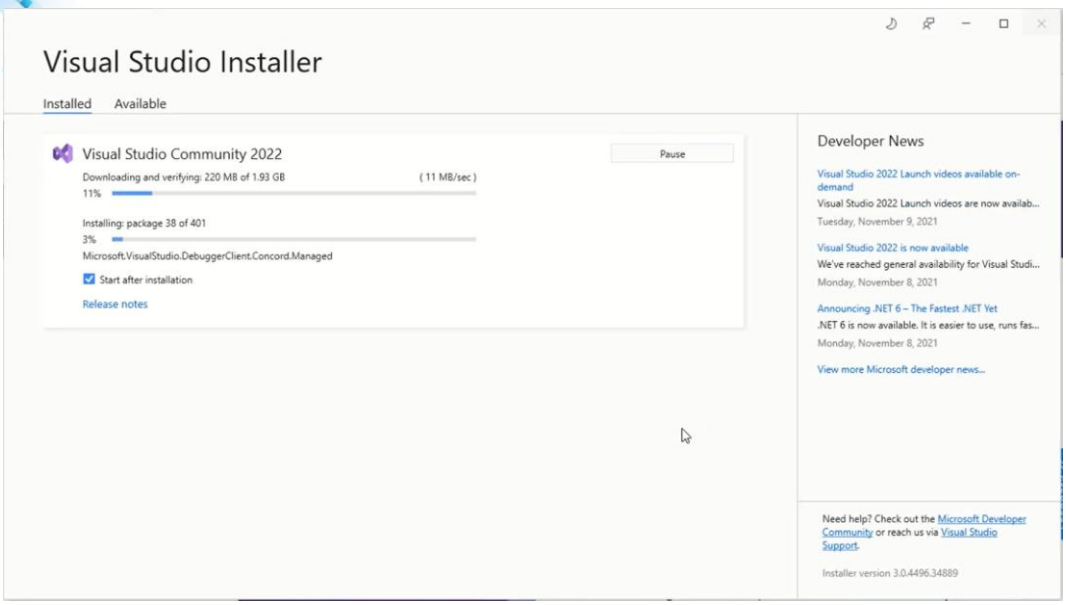

*Gambar 3. 9 Instalasi Perfmon Plugin 2.2.2*

Gambar 3.9 menunjukkan hasil instalasi Perfmon plugin 2.2.2 yang digunakan sebagai plugin Jmeter guna memantau kinerja sumber daya *server* secara *real-time* selama *load testing* proses ini dilakukan dengan mengekstrak file Perfmon plugin pada folder '/lib' pada aplikasi Jmeter.




*Gambar 3. 10 Instalasi Server Agent 2.2.3*

Gambar 3.10 menunjukkan hasil instalasi *Server Agent* 2.2.3 yang digunakan bersamaan dengan plugin Perfmon guna menangkap data pemanfaatan CPU dan memori pada sisi *server* selama proses pengujian *load testing*.



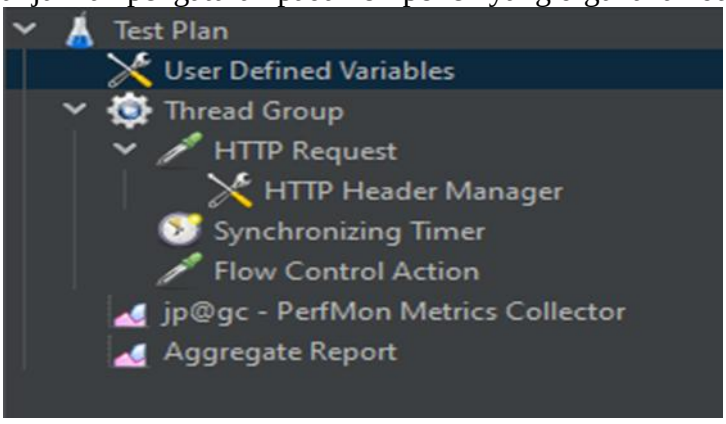

*Gambar 3. 11 Instalasi Visual Studio Community 2022 v17.10.1*

Gambar 3.11 menunjukan hasil instalasi *visual studio community* 2022 v17.10.1 yang diperlukan untuk menjalankan *web server* Rust.

**Perancangan Kasus Pengujian**

Dalam tahapan perancangan kasus pengujian, berdasarkan hasil rancangan pada poin 3.3.3.2, beberapa program dibuat pada aplikasi Visual Studio Code untuk setiap *web server*. *Source code* program-program tersebut dapat ditemukan di Lampiran 1 untuk kode program aplikasi pada *web server* Go, serta di Lampiran 2 untuk kode program aplikasi pada web server Rust.

Simulasi pengujian *load testing* dilakukan secara *offline* menggunakan Apache Jmeter 5.6.3 dengan tambahan plugin perfmon pada sisi *server*, serta program *Server Agent* diinstal pada sisi server. Rancangan pengujian (*Test Plan*) Jmeter pada penelitian ini dapat dilihat pada Gambar 3.15 yang menunjukan rancangan pengujian untuk pengujian load testing web server dan Tabel 3.1 menunjukkan pengaturan pada komponen yang digunakan dalam Test Plan.




*Gambar 3. 12 Test Plan Pengujian Web Server*

*Tabel 3. 1 Konfigurasi Komponen Test Plan Apache Jmeter.*

| No | Komponen | Deskripsi | Pengaturan |
|---|---|---|---|
| 1 | *Test plan* | Rencana Pengujian | *Run Thread Consecutively* |
| 2 | *User defined variables* | Mendefinisikan variabel yang digunakan pada *load testing* | *Server* : 192.168.22.99; *Port* : 8080; *User* : 200, 400, 600, 800, 1000; (sesuai *load* pengujian); *Path* : /upload (*target path*); |
| 3 | *Thread group* | Grup *thread* yang mengeksekusi skenario yang sama | *Users* : ${user}; *Ramp-up period* : 0; *Loop count* : infinite; *Specify Thread liftime* (*duration*) : 300; |
| 4 | HTTP *request sampler* | Mensimulasikan *request* HTTP | *Protocol* : http; *Server Name or IP* : ${*server*} *Port Number* : ${port} HTTP *Request* : POST; *Path* : ${path} Files Upload:{ File Path: C:\Users\husne\Downloads\egy_.jpeg; *Parameter Name*: File; MIME *Type: image/jpeg*}; |
| 5 | HTTP *Header Manager* | Memberikan header pada setiap HTTP *request* sampler | *Content-Type* : *image/jpeg*; |
| 6 | *Synchronizing Timer* | Mengumpulkan *threads* hingga sejumlah *threads* yang ditetapkan tercapai, dan dirilis serentak | *Number of Simulated Users to Group by* : ${user}; |
| 7 | *Flow Control Action* | Memberikan *pause* kepada semua *thread* pada durasi yang ditetapkan | *Durations* : 500 ms; |
| 11 | jp@gc – *Perfmon Metrics Colector* | Monitoring metrik yang didapat oleh *Server*Agent | Sesuai IP target *server* dan PID *web server* |
| 12 | *Aggregate Report* | Memberikan *report* pengujian | - |

**Hasil**

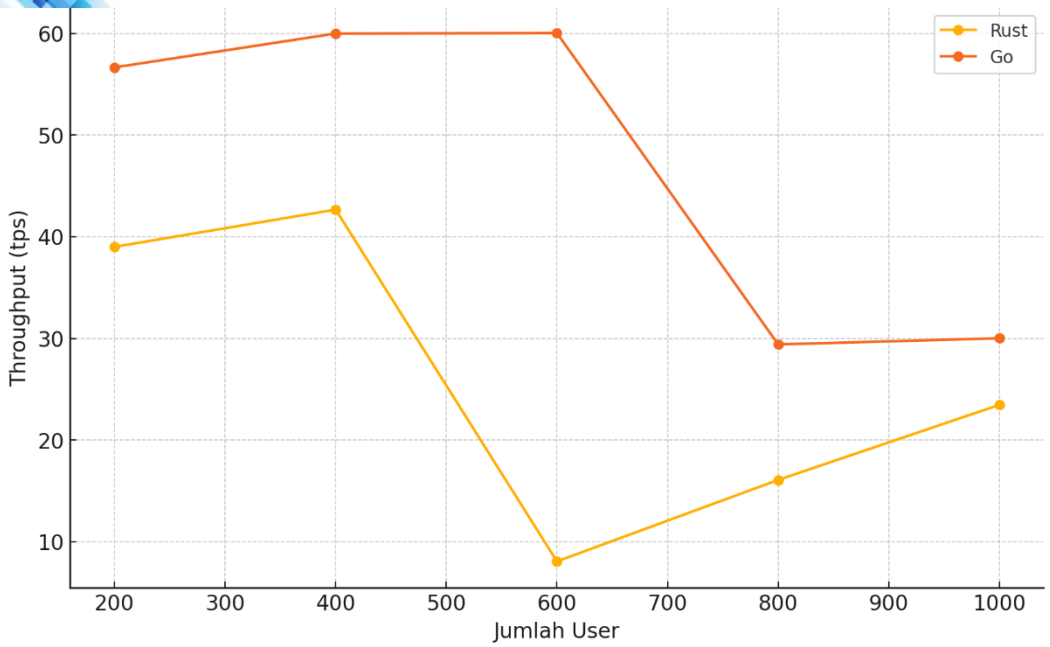
Setelah tahapan implementasi dan tahapan pengujian *load testing* selesai dilaksanakan. Pada tahapan ini, hasil pengujian *load testing* akan dihitung menggunakan aplikasi Excel. Kemudian hasil perhitungan akan disajikan dalam berbagai bentuk tampilan, termasuk grafik dan tabel. Informasi yang akan disajikan mencakup pemanfaatan CPU dan memori, *throughput* dan *processing time* pada setiap tahapan pengujian, serta analisis perbandingan performa dari masing masing *web server.*

**Throughput dan Processing time**

Nilai *throughput* dan *processing time* pada masing – masing *web server* didapat dari pengolahan data hasil pengujian. Hasil *throughput* didapatkan dengan membandingkan berapa banyak transaksi yang berhasil dilakukan selama waktu pengujian sedangkan *processing time* dihasilkan dengan menghitung rata – rata waktu yang diperlukan untuk menyelesaikan semua transaksi. Data disajikan pada Tebel 3.2.

*Tabel 3. 2 Hasil Throughput dan Processing Time Web Server*

| *Web server* | Jumlah *User* | Jumlah Transaksi | Error (%) | Total Durasi (s) | *Throughput* (tps) | Rata-Rata *Processing time* (ms) | Standar Deviasi *Processing time* (ms) |
|---|---|---|---|---|---|---|---|
| Rust | 200 | 11.800 | 2.39 | 300 | 39 | 3727,10 | 10347,61 |
| Rust | 400 | 12.800 | 9.48 | 300 | 42,67 | 4964 | 5415,20 |
| Rust | 600 | 2414 | 19.97 | 300 | 8,0467 | 5859,30 | 6194,28 |
| Rust | 800 | 4802 | 19.39 | 300 | 16,0667 | 7408,53 | 5048,43 |
| Rust | 1000 | 7037 | 32.60 | 300 | 23,4567 | 8431 | 5198,41 |
| Go | 200 | 17000 | 0.00 | 300 | 56,6667 | 2254 | 655,99 |
| Go | 400 | 18000 | 0.00 | 300 | 60 | 4037 | 1620,84 |
| Go | 600 | 18013 | 0.00 | 300 | 60,0433 | 5788 | 2803 |
| Go | 800 | 8821 | 0.00 | 300 | 29,40 | 11122 | 8504,90 |
| Go | 1000 | 9003 | 0.00 | 300 | 30,01 | 15589 | 10824,41 |




*Gambar 3. 13 Diagram hasil throughput masing-masing web server*

Hasil pengujian disajikan dalam Tabel 3.2 yang mencakup berbagai metrik utama seperti jumlah pesan yang diterima, durasi total pengujian, *throughput*, rata-rata waktu pemrosesan, dan standar deviasi dari waktu pemrosesan. Gambar 3.37 menunjukkan bahwa *throughput* server yang diimplementasikan dengan Rust pada awalnya meningkat seiring dengan jumlah pengguna, mencapai puncaknya sekitar 16,07 tps pada 800 pengguna. Namun, throughput turun tajam menjadi sekitar 23,45 tps pada 1000 pengguna, yang menunjukkan penurunan efisiensi. Sebaliknya, Go mampu mempertahankan *throughput* yang tinggi dan stabil di berbagai beban pengguna, dengan puncaknya mencapai 60,0433 tps pada 600 pengguna. Go mempertahankan performanya di semua tingkat, menunjukkan kemampuannya untuk menangani transaksi per detik secara konsisten seiring dengan bertambahnya jumlah pengguna. Waktu pemrosesan rata-rata juga menunjukkan perbedaan yang signifikan antara kedua bahasa pemrograman ini. Rust menunjukkan peningkatan waktu pemrosesan seiring dengan bertambahnya pengguna, dimulai dari 3727,10 ms pada 200 pengguna dan meningkat hingga 8431 ms pada 1000 pengguna ini menunjukkan bahwa Rust mengalami peningkatan latensi seiring dengan skalabilitasnya. Di sisi lain, rata-rata waktu pemrosesan pada Go berfluktuasi sedikit namun tetap relatif rendah, seperti 2254 ms pada 400 pengguna dan 4037 ms pada 600 pengguna. Hal ini menunjukkan bahwa Go dapat mempertahankan latensi yang relatif rendah meskipun jumlah pengguna bertambah.

Selain itu, jumlah transaksi pada server Rust awalnya meningkat seiring dengan bertambahnya jumlah pengguna. Namun, setelah jumlah pengguna melebihi 600, jumlah transaksi menurun secara signifikan, terutama pada 1000 pengguna. Hal ini menunjukkan bahwa Rust mungkin mengalami kesulitan dalam menangani beban yang lebih tinggi secara efektif. Sedangkan Go secara konsisten menangani peningkatan jumlah pengguna dengan peningkatan jumlah transaksi yang stabil hingga mencapai 1000 pengguna, dan tetap pada tingkat kesalahan nol. Hal ini menunjukkan bahwa Go lebih teroptimasi untuk menangani jumlah pengguna yang besar tanpa mengalami penurunan kinerja. Untuk tingkat error Rust meningkat seiring dengan bertambahnya jumlah pengguna. Mulai dari 2,39% pada 200 pengguna, tingkat kesalahan naik hingga 32,60% pada 1000 pengguna, yang menunjukkan adanya masalah skalabilitas. Go mampu mempertahankan tingkat kesalahan 0% di semua jumlah pengguna, yang menunjukkan kestabilan dan ketahanan yang baik terhadap

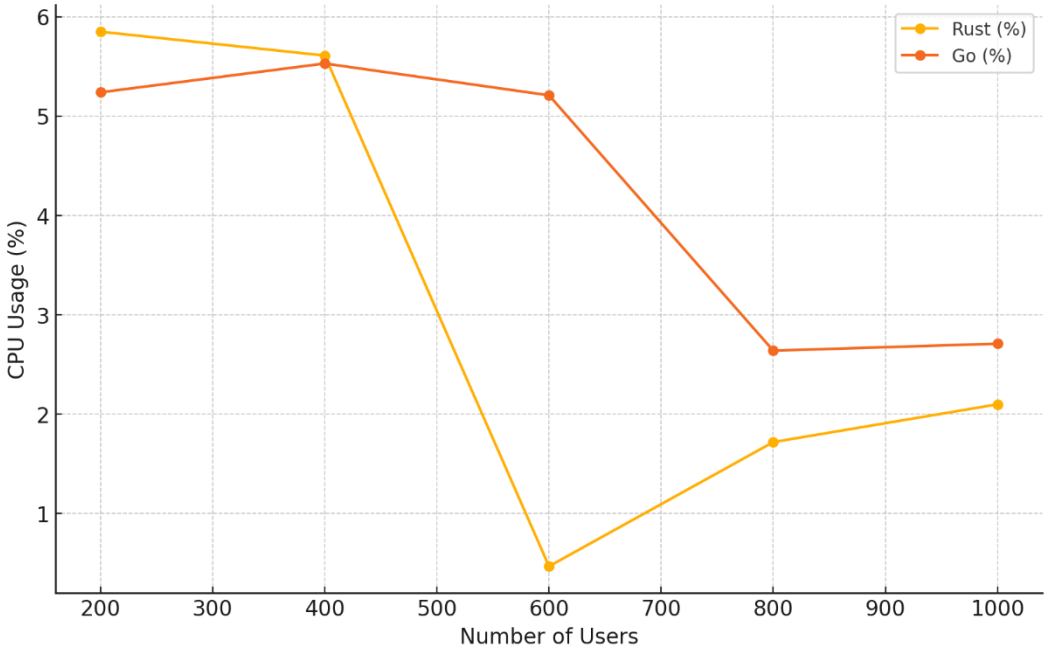
peningkatan beban. Analisis dari nilai standar deviasi waktu pemrosesan pada Rust lebih tinggi dibandingkan dengan Go, terutama pada beban pengguna yang lebih rendah, yang menunjukkan menunjukkan adanya variabilitas dan potensi ketidakpastian dalam waktu respons seiring peningkatan beban. Go, dengan standar deviasi yang lebih rendah, menunjukkan kinerja yang lebih konsisten dan dapat diprediksi, terutama pada jumlah pengguna yang lebih sedikit. Secara keseluruhan, dari hasil perbandingan ini dapat disimpulkan bahwa Go lebih mampu menangani beban pengguna yang lebih besar dengan kinerja yang lebih efisien dan konsisten dibandingkan dengan Rust.

**Hasil Pemanfaatan CPU**

Berikut adalah hasil pemanfaatan CPU masing – masing web server pada pengujian beban. Pemanfaatan CPU dihitung dengan menjumlahkan semua penggunaan CPU dalam memproses transaksi, lalu membagi dengan keseluruhan waktu pengujian. Hasil pemanfaatan CPU pada setiap beban pengujian dapat dilihat pada Tabel 3.3 dan Gambar 3.38 menunjukan grafik perbandingan antara Rust dan Go.

*Tabel 3. 3 Hasil Pemanfaatan CPU oleh Web Server*

| *Users* | *Web server* | |
|---|---|---|
| | Rust (%) | Go (%) |
| 200 | 5,85 | 5,24 |
| 400 | 5,61 | 5,53 |
| 600 | 0,47 | 5,21 |
| 800 | 1,72 | 2,64 |
| 1000 | 2,10 | 2,71 |




*Gambar 3. 14 Grafik pemanfaatan CPU masing-masing web server*

Pada 200 pengguna, Rust menunjukkan penggunaan CPU yang sedikit lebih tinggi (5,85%) dibandingkan Go (5,24%). Namun, ketika jumlah pengguna meningkat menjadi 400,

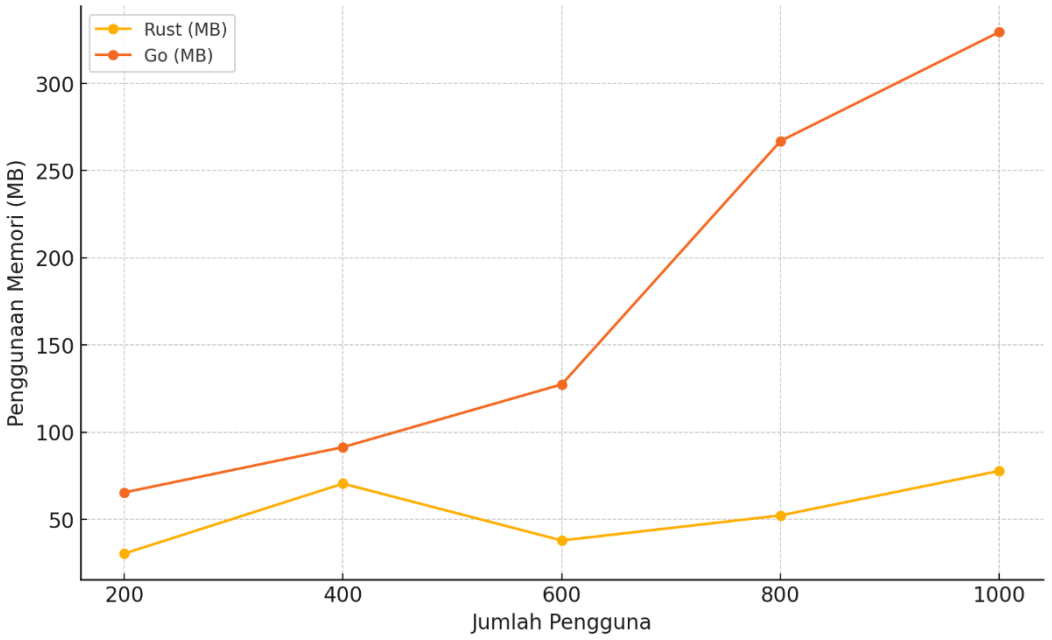
kedua server memiliki penggunaan CPU yang hampir sama, dengan Rust di 5,61% dan Go di 5,53%. Ketika jumlah pengguna mencapai 600, penggunaan CPU oleh Rust menurun drastis menjadi 0,47%, sementara Go tetap stabil di 5,21%. Seiring bertambahnya pengguna menjadi 800, penggunaan CPU oleh Rust meningkat menjadi 1,72%, tetapi masih lebih rendah dibandingkan Go yang berada di 2,64%. Pada 1000 pengguna, Rust terus menunjukkan peningkatan penggunaan CPU menjadi 2,10%, namun tetap lebih efisien dibandingkan Go yang berada di 2,71%. Dari pola ini, terlihat bahwa Rust mengalami fluktuasi yang signifikan dalam penggunaan CPU, namun cenderung lebih efisien saat menangani jumlah pengguna yang lebih besar, sementara Go menunjukkan penggunaan CPU yang lebih stabil namun sedikit lebih tinggi pada jumlah pengguna yang besar.

**Hasil Pemanfaatan Memori**

Berikut adalah hasil pemanfaatan *memory* pada masing – masing *web server* yang diujikan pada pengujian beban. Pemanfaatan *memory* dihitung dengan menjumlahkan penggunaan *memory* yang diperlukan untuk setiap skenario pengujian lalu membagi dengan total waktu pengujian. Tabel 3.4 menunjukan perbandingan pemanfaatan *memory* pada masing – masing *web server* dan Gambar 3.31 menunjukan grafik perbandingan pemanfaatan *memory* untuk setiap skenario pengujian.

*Tabel 3. 4 Hasil Pemanfaatan Memory oleh Web Server*

| *Users* | *Web Server* | |
|---|---|---|
| | Rust (MB) | Go (MB) |
| 200 | 30,40 | 65,45 |
| 400 | 70,53 | 91,45 |
| 600 | 37,92 | 127,44 |
| 800 | 52,28 | 267,01 |
| 1000 | 77,81 | 329,51 |




*Gambar 3. 15 Grafik Pemanfaatan Memory Masing-Masing Web Server*

Dari data yang ditampilkan, terlihat bahwa server web yang ditulis dalam Rust umumnya menggunakan lebih sedikit memori dibandingkan dengan yang ditulis dalam Go, di setiap level jumlah pengguna. Penggunaan memori pada Rust cenderung tidak linear, dengan penurunan signifikan pada 600 pengguna dibandingkan dengan 400 pengguna, sebelum kembali meningkat pada 800 dan 1000 pengguna. Sementara itu, penggunaan memori pada Go menunjukkan peningkatan yang stabil seiring bertambahnya jumlah pengguna. Dari hasil ini, dapat disimpulkan bahwa Rust lebih efisien dalam penggunaan memori dibandingkan Go.

**Analisis Hasil Pengujian**

**Analisis Pemanfaatan CPU dan Memori**

Analisis hasil pemanfaatan CPU dan *memory* ini berdasarkan hasil yang diperoleh dari pengujian *load testing.* Nilai rasio dalam penelitian ini digunakan untuk mempermudah mengidentifikasi serta membandingkan performa dari dua *web server* selanjutnya nilai rasio menjadi dasar untuk pengambilan keputusan. Pada Tabel 3.5 menunjukan perbandingan pemanfaatan CPU antara *web server* Rust dan Go. Rust menunjukkan penggunaan CPU yang lebih optimal dibandingkan dengan Go pada sebagian besar jumlah pengguna. Pada jumlah pengguna 200 dan 400, penggunaan CPU antara Rust dan Go cukup berimbang, dengan Go hanya sedikit lebih rendah dalam hal pemanfaatan CPU. Namun, perbedaan yang mencolok terjadi pada jumlah pengguna yang lebih tinggi. Pada 600 pengguna, terjadi perbedaan besar dalam pemanfaatan CPU, di mana Go menggunakan CPU lebih dari 11 kali lipat dibandingkan Rust. Ini mungkin menunjukkan bahwa Rust memiliki mekanisme yang lebih optimal dalam menangani jumlah beban pengguna yang meningkat, atau mungkin Go menghadapi masalah tertentu yang menyebabkan peningkatan pemanfaatan CPU yang signifikan pada titik ini.

Pada jumlah pengguna 800 dan 1000, Rust tetap menunjukkan performa yang lebih optimal dengan menggunakan CPU lebih sedikit dibandingkan dengan Go. Meskipun Go masih menggunakan lebih banyak CPU, perbedaannya tidak sebesar pada 600 pengguna, tetapi tetap menunjukkan bahwa Rust lebih stabil dan optimal dalam memanfaatkan CPU pada beban yang lebih tinggi. Secara keseluruhan, Rust cenderung lebih optimal dalam pemanfaatan CPU dibandingkan Go, terutama ketika jumlah pengguna meningkat. Hal ini menunjukkan bahwa Rust dapat menjadi pilihan yang lebih baik untuk aplikasi yang memerlukan pengelolaan sumber daya yang ketat dan kinerja yang stabil di berbagai kondisi beban.

*Tabel 3. 5 Tabel Perbandingan Pemanfaatan CPU*

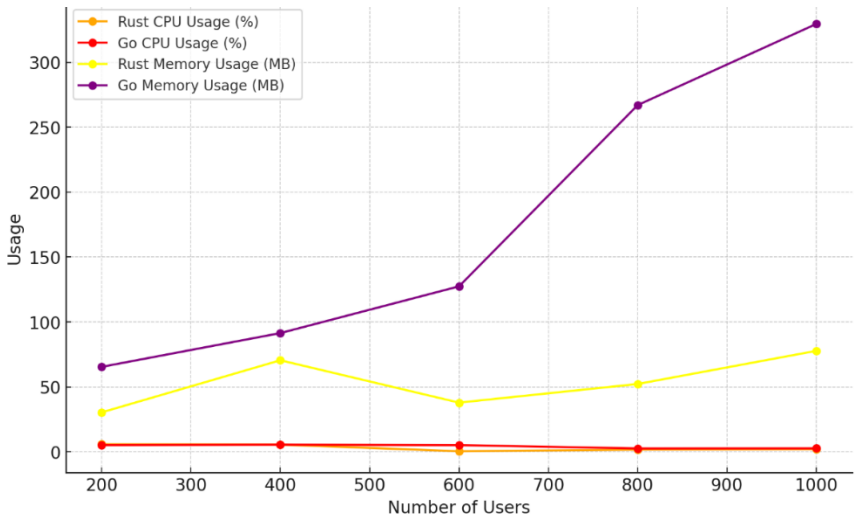
| Jumlah *User* | Pemanfaatan CPU Rust (%) | Pemanfaatan CPU Go (%) | Rasio Pemanfaatan CPU ( Go / Rust ) |
|---|---|---|---|
| 200 | 5,85 | 5,24 | 0,895 |
| 400 | 5,61 | 5,53 | 0,985 |
| 600 | 0,47 | 5,21 | 11,085 |
| 800 | 1,72 | 2,64 | 1,534 |
| 1000 | 2,10 | 2,71 | 1,290 |

Dari data yang ditampilkan dalam Tabel 3.6 terlihat bahwa Rust secara konsisten menunjukkan pemanfaatan memori yang jauh lebih optimal dibandingkan Go di semua jumlah pengguna. Pada setiap tingkat pengguna, Rust menggunakan lebih sedikit memori, yang menunjukkan bahwa Rust dioptimalkan untuk penggunaan memori yang minimal. Pada 200 pengguna, Rust menggunakan 30,40 MB memori, sedangkan Go menggunakan 65,45 MB. Ini

berarti Go menggunakan lebih dari dua kali lipat memori dibandingkan Rust. Hal ini menunjukkan bahwa bahkan pada tingkat beban pengguna yang lebih rendah, Rust lebih hemat dalam penggunaan memori. Perbedaan dalam pemanfaatan memori antara Rust dan Go menjadi semakin mencolok seiring bertambahnya jumlah pengguna. Pada 800 pengguna, Go menggunakan 267,01 MB memori, sedangkan Rust hanya menggunakan 52,28 MB. Ini berarti Go menggunakan lebih dari lima kali lipat memori dibandingkan Rust. Ini adalah indikasi bahwa Rust mempertahankan efisiensi memori yang lebih baik, bahkan saat beban pengguna meningkat. Secara keseluruhan, Rust menunjukkan keunggulan yang jelas dalam hal pemanfaatan memori dibandingkan Go, terutama ketika beban pengguna meningkat. Ini menjadikan Rust pilihan yang lebih baik untuk aplikasi yang memerlukan efisiensi memori tinggi dan kinerja yang stabil, sedangkan Go mungkin lebih cocok untuk situasi di mana sumber daya memori yang lebih besar tersedia dan efisiensi memori bukanlah faktor kritis.

*Tabel 3. 6 Tabel Perbandingan Pemanfaatan Memory*

| Jumlah *User* | Pemanfaatan *Memory* Rust (MB) | Pemanfaatan *Memory* Go (MB) | Rasio Pemanfaatan *Memory* ( Go / Rust ) |
|---|---|---|---|
| 200 | 30,40 | 65,45 | 2,152 |
| 400 | 70,53 | 91,45 | 1,296 |
| 600 | 37,92 | 127,44 | 3,360 |
| 800 | 52,28 | 267,01 | 5,107 |
| 1000 | 77,81 | 329,51 | 4,234 |




*Gambar 3. 16 Grafik Pemanfaatan CPU dan Memory*

Seperti yang ditunjukan pada Gambar 3.40 Rust secara keseluruhan menunjukkan efisiensi yang lebih tinggi dalam penggunaan CPU, terutama pada jumlah pengguna yang lebih besar (600 hingga 1000 pengguna). Pada 600 pengguna, misalnya, Rust hanya menggunakan 0,47% CPU, sementara Go menggunakan 5,21%. Hal ini menunjukkan bahwa Rust mampu mempertahankan penggunaan CPU yang rendah meskipun jumlah pengguna meningkat. Go menunjukkan efisiensi yang sebanding dengan Rust pada jumlah pengguna yang lebih rendah (200 dan 400 pengguna), tetapi mengalami peningkatan signifikan dalam penggunaan CPU

pada jumlah *user* yang lebih besar, yang dapat menunjukkan bahwa Go kurang efisien dalam menangani beban tinggi. Rust secara konsisten lebih hemat dalam penggunaan memori dibandingkan Go pada semua tingkat pengguna. Bahkan pada jumlah pengguna yang lebih kecil, perbedaan dalam penggunaan memori antara Rust dan Go sudah terlihat jelas, dengan Go menggunakan lebih dari dua kali lipat memori yang digunakan oleh Rust. Ketika jumlah pengguna meningkat, efisiensi memori Rust semakin jelas. Pada 800 dan 1000 pengguna, Go menggunakan lima kali lebih banyak memori dibandingkan Rust, menunjukkan bahwa Rust jauh lebih unggul dalam hal pengelolaan memori.

**Analisis Kecepatan *Web Server***

Berdasarkan analisis waktu pemrosesan yang ditampilkan dalam Tabel 3.7, terlihat bahwa *web server* Go secara umum memiliki waktu pemrosesan yang lebih cepat dibandingkan *web server* Rust pada sebagian besar skenario. Pada 200 pengguna, waktu pemrosesan rata-rata *web server* Go adalah 2254 ms, sedangkan *web server* Rust memerlukan 3727,10 ms untuk menyelesaikan transaksi yang sama, dengan rasio waktu pemrosesan *web server* Rust terhadap *web server* Go sebesar 1,653. Ini menunjukkan bahwa *web server* Rust memerlukan waktu sekitar 65,3% lebih lama dibandingkan *web server* Go untuk memproses transaksi pada tingkat pengguna ini. Saat jumlah pengguna meningkat menjadi 400, waktu pemrosesan untuk kedua *web server* mendekati nilai yang lebih seimbang, dengan *web server* Go memerlukan 4037 ms dan *web server* Rust 4964 ms, menghasilkan rasio waktu pemrosesan sebesar 1,229. Namun, pada 600 pengguna, waktu pemrosesan untuk kedua bahasa hampir sama, dengan rasio 1,012, yang menunjukkan bahwa kedua bahasa memiliki performa yang hampir setara dalam hal waktu pemrosesan pada jumlah pengguna ini.

Perbedaan yang signifikan muncul pada 800 pengguna, di mana *web server* Go menunjukkan peningkatan performa yang luar biasa dengan waktu pemrosesan hanya 173,47 ms, sementara *web server* Rust memerlukan waktu 7408,53 ms. Rasio waktu pemrosesan Rust/Go melonjak menjadi 42,707, yang berarti *web server* Rust sangat jauh lebih lambat dibandingkan *web server* Go pada jumlah pengguna ini. Namun, pada 1000 pengguna, situasi berbalik, dengan Go mengalami peningkatan waktu pemrosesan yang tajam hingga 15589 ms, sementara Rust berhasil menurunkan waktu pemrosesannya menjadi 8431 ms. Ini menghasilkan rasio waktu pemrosesan Rust/Go sebesar 0,540, yang menunjukkan bahwa pada beban yang sangat tinggi, *web server* Rust ternyata lebih cepat dalam memproses transaksi dibandingkan *web server* Go. Secara keseluruhan, *web server* Go cenderung memiliki waktu pemrosesan yang lebih baik dibandingkan *web server* Rust pada sebagian besar skenario, terutama pada jumlah pengguna yang lebih rendah dan menengah. Namun, pada beban pengguna yang sangat tinggi, seperti 1000 pengguna, *web server* Rust menunjukkan peningkatan kinerja yang signifikan, mampu mengungguli *web server* Go dalam hal kecepatan pemrosesan. Hal ini menunjukkan bahwa kinerja kedua bahasa dapat sangat bervariasi tergantung pada konteks beban kerja dan jumlah pengguna yang dilayani.

**Tabel 3. 7** Tabel Perbandingan Rata – Rata *Processing Time*

| Jumlah *User* | Rata-Rata *Processing time* Rust (ms) | Rata-Rata *Processing time* Go (ms) | Rasio *Processing time* (Rust / Go) |
|---|---|---|---|
| 200 | 3.727,10 | 2.254 | 1,653 |
| 400 | 3.964 | 3.037 | 1,229 |
| 600 | 5.859,30 | 5.788 | 1,012 |
| 800 | 7.408,53 | 173,47 | 42,707 |
| 1000 | 8.431 | 1.5589 | 0,540 |

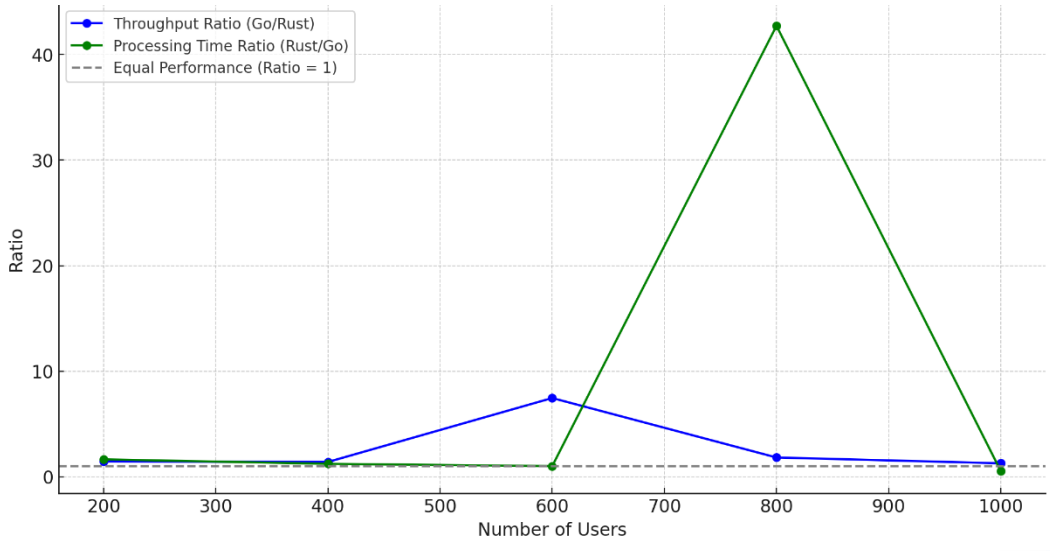
Berdasarkan analisis *throughput* yang ditampilkan dalam Tabel 3.8, terlihat bahwa *web server* Go secara umum memiliki keunggulan dalam hal jumlah transaksi yang dapat diproses

per detik (tps) dibandingkan dengan *web server* Rust. Pada 200 pengguna, Go mampu mencapai *throughput* sebesar 56,667 tps, sementara *web server* Rust hanya mencapai 39 tps, dengan rasio *throughput* Go terhadap *web server* Rust sebesar 1,453. Ini menunjukkan bahwa *web server* Go mampu memproses sekitar 45,3% lebih banyak transaksi per detik dibandingkan *web server* Rust pada tingkat pengguna ini. Seiring bertambahnya jumlah pengguna menjadi 400, *throughput web server* Go tetap stabil di angka 60 tps, sedangkan *web server* Rust meningkat menjadi 42,67 tps, yang membuat rasio *throughput* menjadi 1,406. Hal ini menunjukkan bahwa meskipun *web server* Rust mulai mengejar, *web server* Go masih memiliki keunggulan. Namun, pada 600 pengguna, perbedaan *throughput* menjadi sangat mencolok, di mana *web server* Go mencapai 60,043 tps dan *web server* Rust hanya 8,047 tps. Dengan rasio *throughput* Go/Rust sebesar 7,461, ini menunjukkan bahwa *web server* Go lebih dari tujuh kali lipat lebih optimal dalam memproses transaksi dibandingkan *web server* Rust pada jumlah pengguna ini.

Pada 800 pengguna, *throughput web server* Go menurun menjadi 29,40 tps, sedangkan *web server* Rust meningkat menjadi 16,067 tps, dengan rasio Go/Rust sebesar 1,829. Ini menunjukkan bahwa meskipun *web server* Go masih unggul, selisih antara kedua bahasa mulai menurun. Pada 1000 pengguna, selisih ini semakin mengecil dengan *web server* Go mencapai 30,01 tps dan *web server* Rust mencapai 23,4567 tps, yang menghasilkan rasio Go/Rust sebesar 1,279. Secara keseluruhan, *web server* Go menunjukkan kemampuan yang lebih baik dalam memproses transaksi per detik di sebagian besar skenario, terutama pada jumlah pengguna yang lebih rendah dan menengah. Namun, seiring bertambahnya jumlah pengguna, *web server* Rust mulai menunjukkan peningkatan kinerja, mendekati dan bahkan mengejar performa *web server* Go pada jumlah pengguna yang lebih tinggi, seperti yang terlihat pada 1000 pengguna.

*Tabel 3. 8 Tabel Perbandingan Throughput*

| Jumlah *User* | *Throughput* Go (bps) | *Throughput* Rust (bps) | Rasio *Throughput* (Go / Rust) |
|---|---|---|---|
| 200 | 56,667 | 39 | 1,453 |
| 400 | 60 | 42,67 | 1,406 |
| 600 | 60,043 | 8,047 | 7,461 |
| 800 | 29,40 | 16,067 | 1,829 |
| 1000 | 30,01 | 23,4567 | 1,279 |




*Gambar 3. 17 Grafik perbandingan kecepatan web server berdasarkan rasio*

Rasio *Throughput* (Go/Rust) yang ditunjukan oleh garis biru, menunjukkan seberapa banyak *throughput web server* Go dibandingkan dengan *web server* Rust pada berbagai tingkat pengguna. Angka yang lebih tinggi dari 1 menunjukkan bahwa *web server* Go memiliki *throughput* yang lebih baik dibandingkan *web server* Rust. Kita melihat bahwa *web server* Go secara umum memiliki *throughput* yang lebih tinggi, terutama pada 600 pengguna, di mana *web server* Go mencapai hampir 7,5 kali lipat *throughput web server* Rust. Namun, pada 1000 pengguna, rasio ini mendekati 1, yang menunjukkan bahwa performa keduanya mulai seimbang.

Rasio Waktu Pemrosesan (Rust/Go) yang ditunjukan oleh garis hijau, menunjukkan perbandingan waktu pemrosesan antara *web server* Rust dan *web server* Go. Angka yang lebih tinggi dari 1 menunjukkan bahwa *web server* Rust memerlukan lebih banyak waktu dibandingkan *web server* Go, sedangkan angka di bawah 1 menunjukkan bahwa *web server* Rust lebih cepat. Pada 800 pengguna, waktu pemrosesan *web server* Go jauh lebih cepat dibandingkan *web server* Rust, dengan rasio mencapai lebih dari 42. Namun, pada 1000 pengguna, *web server web server* Rust menjadi lebih optimal, dengan waktu pemrosesan lebih cepat dibandingkan *web server* Go, yang terlihat dari rasio di bawah 1. Garis horizontal abu-abu menunjukkan titik di mana performa keduanya sama (rasio = 1).

Secara keseluruhan *web server* Go lebih baik dalam *throughput*, terutama pada jumlah pengguna menengah, namun *web server* Rust menunjukkan keunggulan dalam waktu pemrosesan pada beban pengguna yang sangat tinggi. Pilihan antara *web server* Rust dan *web server* Go akan bergantung pada kebutuhan spesifik aplikasi, apakah memprioritaskan *throughput* atau waktu pemrosesan, terutama di bawah beban tinggi.

**Analisis Stabilitas *Web Server***

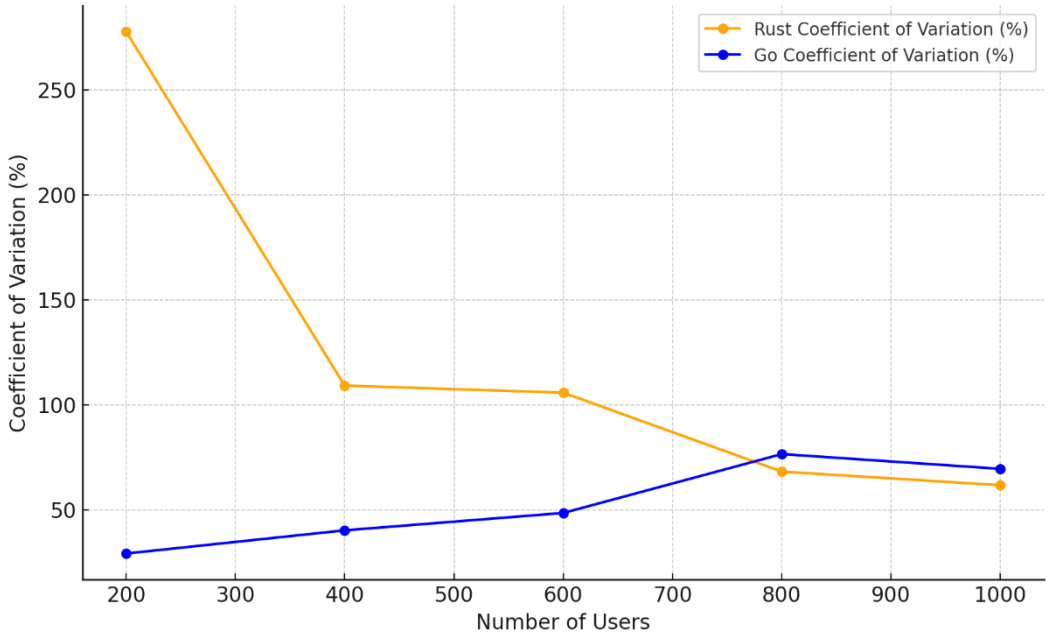
Berdasarkan analisis data koefisien variasi dari Tabel 3.9, terlihat bahwa stabilitas waktu pemrosesan *web server* Go umumnya lebih baik dibandingkan *web server* Rust pada jumlah pengguna yang lebih rendah hingga menengah. Pada 200 pengguna, Go menunjukkan koefisien variasi (CV) sebesar 29,11%, yang jauh lebih rendah dibandingkan dengan CV *web server* Rust yang mencapai 277,63%. Hal ini menunjukkan bahwa *web server* Go memiliki kinerja yang lebih konsisten dan stabil pada beban pengguna yang lebih rendah. Saat jumlah pengguna meningkat hingga 600 pengguna, *web server* Go masih menunjukkan stabilitas yang lebih tinggi dengan CV di bawah 50%, sedangkan *web server* Rust menunjukkan perbaikan stabilitas tetapi tetap lebih tinggi daripada *web server* Go.

Namun, seiring bertambahnya jumlah pengguna hingga 800 dan 1000, *web server* Rust menunjukkan peningkatan stabilitas yang signifikan. Pada 800 pengguna, koefisien variasi *web server* Rust menurun menjadi 68,14%, sedangkan *web server* Go mulai menunjukkan tanda-tanda penurunan stabilitas dengan CV sebesar 76,47%. Pada 1000 pengguna, *web server* Rust menjadi lebih stabil dengan CV sebesar 61,66%, dibandingkan dengan Go yang memiliki CV sebesar 69,44%. Ini menunjukkan bahwa *web server* Rust lebih mampu menjaga konsistensi kinerjanya pada beban pengguna yang tinggi, sedangkan *web server* Go mengalami penurunan stabilitas pada jumlah pengguna yang lebih besar.

**Tabel 3. 9** Tabel Perbandingan Koefisien Variasi pada *Web Server*

| *Web server* | Jumlah *User* | Rata-Rata *Processing time* (ms) | Standar Deviasi *Processing time* (ms) | Koefisien Variasi (%) |
|---|---|---|---|---|
| Rust | 200 | 3.727,10 | 10.347,61 | 277,63 |
| Rust | 400 | 3.964 | 5.415,20 | 109,08 |
| Rust | 600 | 5.859,30 | 6.194,28 | 105,72 |
| Rust | 800 | 7.408,53 | 5.048,43 | 68,14 |
| Rust | 1000 | 8.431 | 5.198,41 | 61,66 |

| *Web server* | Jumlah *User* | Rata-Rata *Processing time* (ms) | Standar Deviasi *Processing time* (ms) | Koefisien Variasi (%) |
|---|---|---|---|---|
| Go | 200 | 2.254 | 655,99 | 29,11 |
| Go | 400 | 3.037 | 1.620,84 | 40,15 |
| Go | 600 | 5.788 | 2.803 | 48,43 |
| Go | 800 | 11.122 | 8.504,90 | 76,47 |
| Go | 1000 | 15.589 | 10.824,41 | 69,44 |




*Gambar 3. 18 Grafik analisis stabilitas dari web server*

Garis Oranye pada grafik menunjukkan bahwa Rust memiliki koefisien variasi yang sangat tinggi pada 200 pengguna, yang mencerminkan ketidakstabilan kinerja. Namun, Rust menunjukkan peningkatan stabilitas yang signifikan seiring bertambahnya jumlah pengguna, terutama setelah 400 pengguna, dengan penurunan CV yang terus berlanjut hingga 1000 pengguna. Garis Biru menunjukkan bahwa Go memiliki koefisien variasi yang jauh lebih rendah, menunjukkan kinerja yang lebih stabil pada 200 pengguna. Seiring bertambahnya jumlah pengguna, CV Go cenderung meningkat, terutama setelah 600 pengguna, meskipun tetap lebih rendah daripada Rust pada sebagian besar skenario. Secara keseluruhan, grafik ini menunjukkan bahwa Go lebih stabil pada jumlah pengguna yang lebih rendah hingga menengah, sementara Rust meningkatkan stabilitasnya secara signifikan pada jumlah pengguna yang lebih tinggi, hingga menjadi lebih stabil daripada Go pada beban puncak (1000 pengguna).

**Analisis Skalabilitas Web Server**

Berdasarkan analisis data dari Tabel 3.10, terlihat bahwa web server Rust dan Go menunjukkan perilaku yang berbeda dalam hal skalabilitas saat jumlah pengguna meningkat. Rust menunjukkan kemampuan adaptasi di mana pada awalnya mengalami penurunan *throughput* yang signifikan ketika jumlah pengguna meningkat dari 400 ke 600, dengan penurunan sebesar 81,14%. Namun, setelah melewati titik kritis ini, Rust mampu pulih dengan peningkatan throughput yang signifikan sebesar 99,66% dari 600 ke 800 pengguna, dan terus meningkat sebesar 46,057% dari 800 ke 1000 pengguna. Hal ini menunjukkan bahwa Rust,

meskipun awalnya menghadapi tantangan dalam menangani peningkatan jumlah pengguna, memiliki kemampuan untuk beradaptasi dan meningkatkan performa seiring dengan bertambahnya beban pengguna.

Di sisi lain, Go menunjukkan perilaku yang lebih stabil pada tahap awal dengan peningkatan *throughput* dari 200 ke 400 pengguna, namun mulai menunjukkan batasan skalabilitasnya saat jumlah pengguna mencapai 600. Peningkatan *throughput* Go hanya sebesar 0,071% dari 400 ke 600 pengguna, yang mengindikasikan bahwa Go hampir mencapai kapasitas maksimalnya. Selanjutnya, Go mengalami penurunan *throughput* yang signifikan sebesar 51,035% ketika jumlah pengguna meningkat dari 600 ke 800, dan hanya menunjukkan sedikit peningkatan sebesar 2,074% ketika jumlah pengguna mencapai 1000. Ini menandakan bahwa Go mengalami kesulitan dalam menjaga performa dan stabilitas saat menghadapi peningkatan beban pengguna yang lebih tinggi. Secara keseluruhan, Rust menunjukkan skalabilitas yang lebih baik pada jumlah pengguna yang sangat tinggi, mampu beradaptasi dan meningkatkan performa setelah melewati titik kritis. Sementara itu, Go lebih stabil pada jumlah pengguna yang lebih rendah hingga menengah, tetapi menghadapi tantangan signifikan saat jumlah pengguna meningkat di atas 600. Pemilihan antara Rust dan Go sebaiknya disesuaikan dengan kebutuhan spesifik aplikasi, terutama terkait dengan ekspektasi jumlah pengguna dan kebutuhan skalabilitas.

**Tabel 3. 10** Perbandingan Peningkatan *Throughput* pada *Web Server*

| *Web server* | Jumlah *User* | *Throughput* (bps) | Peningkatan *Throughput* (%) |
|---|---|---|---|
| Rust | 200 | 39 | |
| Rust | 400 | 42,67 | 9,410 |
| Rust | 600 | 8,047 | -81,141 |
| Rust | 800 | 16,067 | 99,664 |
| Rust | 1000 | 23,467 | 46,057 |
| Go | 200 | 56,667 | |
| Go | 400 | 60 | 5,881 |
| Go | 600 | 60,043 | 0,071 |
| Go | 800 | 29,40 | -51,035 |
| Go | 1000 | 30,01 | 2,074 |

## Kesimpulan dan Saran

### Kesimpulan

Berdasarkan penelitian yang telah dilakukan melalui pengujian *load testing* pada protokol HTTP untuk membandingkan performa web server Rust dan Go dalam menangani berbagai beban. Hasil penelitian menunjukkan bahwa web server Rust cenderung lebih stabil dan optimal dalam penggunaan sumber daya dibandingkan Go. Pada beban 600 pengguna, Rust hanya menggunakan 0,47% CPU, sementara Go mencapai 5,21% ini menunjukkan efisiensi 11 kali lebih baik. Penggunaan memori juga lebih rendah pada Rust, dengan rata-rata 52,28 MB pada beban 800 pengguna, sementara Go mengonsumsi 267,01 MB pada beban yang sama. Ini berarti Go menggunakan lebih dari lima kali lipat memori dibandingkan Rust.

Dari segi kecepatan Go memiliki performa lebih optimal dibanding Rust berdasarkan nilai *throughput* dan *processing time*. Pada beban 600 pengguna, Go mencatatkan throughput rata-rata 60,043 transaksi per detik (tps), sedangkan Rust hanya mencapai 8,047 tps pada skenario yang sama. Pada beban 800 pengguna nilai processing time pada Go adalah 173,47 ms, sedangkan Rust mencatatkan waktu pemrosesan 7408,53 ms pada pengujian yang sama. Ini berarti Go dalam memproses data lebih dari 42 kali lipat lebih cepat dibandingkan Rust. Dari segi stabilitas terlihat bahwa web server Go memiliki kestabilan yang lebih baik

dibandingkan web server Rust. Pada 200 pengguna, Go menunjukkan koefisien variasi (CV) sebesar 29,11%, yang jauh lebih rendah dibandingkan dengan CV web server Rust yang mencapai 277,63%. Dari segi skalabilitas Rust lebih unggul dalam menghadapi jumlah pengguna, ditunjukan peningkatan throughput pada beban 1000 user mencapai 46.057%, sementara Go hanya 2.074%.

**Saran**

Berdasarkan penelitian yang telah dilakukan, terdapat beberapa saran yang yang dapat diberikan untuk pengembangan atau pengujian lebih lanjut:

1. Implementasi Protokol Lain.

   Uji kinerja *web server* menggunakan protokol lain seperti HTTPS, HTTP/2, WebSocket, atau gRPC untuk memperluas cakupan pengujian dan mendapatkan pemahaman lebih komprehensif tentang performa di berbagai kondisi.
2. Pengujian Keamanan

   Lakukan pengujian keamanan terhadap kedua *web server* untuk mengidentifikasi kekuatan dan kelemahan keamanan dalam menghadapi berbagai ancaman siber
3. *Stress Testing*

   Lakukan *stress testing* dengan menambahkan beban yang lebih besar untuk mengetahui batas maksimal kinerja dan skalabilitas *web server*.
4. Pengujian di Lingkungan Jaringan Berbeda

   Uji *web server* pada berbagai kondisi jaringan seperti jaringan dengan latensi tinggi, *bandwidth* terbatas, atau jaringan nirkabel untuk mendapatkan hasil yang lebih relevan dengan kondisi dunia nyata.
5. Pengujian di Sistem Operasi Berbeda

   Lakukan pengujian pada berbagai sistem operasi untuk menentukan apakah ada perbedaan signifikan dalam performa *web server* di lingkungan yang berbeda.

**Daftar Pustaka**

Ade Ismail, Ahmadi Yuli Ananta, Sofyan Noor Arief, and Elok Nur Hamdana. 2023. "Performance Testing Sistem Ujian Online Menggunakan Jmeter Pada Lingkungan Virtual." *Jurnal Informatika Polinema* 9(2):159–63. doi: 10.33795/jip.v9i2.1190.

Alkalah, Cynthia. 2016. "Framework for TCP Throughput Testing." 19(5):1–23.

Amrullah, Agit, Agung Nugroho, and Zekriansyah Ramadhan. 2023. "Perbandingan Kinerja Web Server Pada Penyedia Layanan Cloud Microsoft Azure Dan Amazon Web Services." *Jurnal Informatika Teknologi Dan Sains* 5(1):92–97. doi: 10.51401/jinteks.v5i1.2487.

Anastasi, G., E. Borgia, M. Conti, and E. Gregori. 2003. "IEEE 802.11 Ad Hoc Networks: Performance Measurements." *Proceedings - 23rd International Conference on Distributed Computing Systems Workshops, ICDCSW 2003* (September):758–63. doi: 10.1109/ICDCSW.2003.1203643.

Burke, John. 2022. "Throughput."

Chandra, Albert Yakobus. 2019. "Analisis Performansi Antara Apache & Nginx Web Server Dalam Menangani Client Request." *Jurnal Sistem Dan Informatika (JSI)* 14(1):48–56. doi: 10.30864/jsi.v14i1.248.

David Gourley, Brian Totty. 2002. "HTTP: The Definitive Guide." edited by L. Mui. Sebastopul: O'Reily Media.

GO. 2023. "Build Simple, Secure, Scalable Systems with Go." *GO*. Retrieved July 31, 2024 (https://go.dev/).

Kurniansyah, M. Iqbal, and Sinar Sinurat. 2020. "Sistem Pendukung Keputusan Pemilihan Server Hosting Dan Domain Terbaik Untuk WEB Server Menerapkan Metode VIKOR." *JSON (Jurnal Sistem Komputer Dan Informatika)* 2(1):14–23. doi: 10.30865/json.v2i1.2450.

Luh, Ni, Ayu Sonia, Kadek Suar, Ni Kadek, and Ayu Wirdiani. 2021. "Pengujian Stress Testing API Sistem Pelayanan Dengan Apache JMeter." *JITTER- Jurnal Ilmiah Teknologi Dan Komputer* 2(3).

Netcraft. 2023. "Survei Server Web Juli 2023." *Netcraft*. Retrieved (https://www.netcraft.com/blog/july-2024-web-server-survey/).

Permatasari, Desy Intan. 2020. "Pengujian Aplikasi Menggunakan Metode Load Testing Dengan Apache JMeter Pada Sistem Informasi Pertanian." *Jurnal Sistem Dan Teknologi Informasi (JUSTIN)* 8(1):135. doi: 10.26418/justin.v8i1.34452.

Pragestu, Steven, Herry Sujaini, and Eva Faja. 2023. "Analisis Skalabilitas Web Server Apache Tomcat , Node . Js Dan Go Pada Protokol Hypertext Transfer Protocol ( HTTP ) Dan Message Queue Telemetry Transport ( MQTT ) Scalability Analysis of Apache Tomcat , Node . Js and Go Web Servers on Hypertext Transfer." *JUSTIN (Jurnal Sistem Dan Teknologi Informasi)* 11(4):605–11. doi: 10.26418/justin.v11i3.71607.

Prayogo, Noval Agung. 2023. *Dasar Pemograman Golang*.

Ridwan, Muannif, Suhar AM, Bahrul Ulum, and Fauzi Muhammad. 2021. "Pentingnya Penerapan Literature Review Pada Penelitian Ilmiah." *Jurnal Masohi* 2(1):42. doi: 10.36339/jmas.v2i1.427.

Statista. 2023. "Number of Internet and Social Media Users Worldwide as of April 2023." *Ani Petrosyan*. Retrieved (https://www.statista.com/statistics/617136/digital-population-worldwide/#:~:text=Worldwide digital population 2024&text=As of April 2024%2C there,percent of the global population.).

Steve Klabnik, Carol Nichols. 2023. "The Rust Programming Language." united states: william pollock.

Sutriani, Elma, and Rika Octaviani. 2019. "ANALISIS DATA DAN PENGECEKAN KEABSAHAN DATA." *INA-Rxiv* 1–22.

TIOBE. 2023. "TIOBE Index for July 2023." *TIOBE*. Retrieved July 31, 2024 (https://www.tiobe.com/tiobe-index/).

W3techs. 2023. "Statistik Penggunaan Protokol Default Https Untuk Situs Web." *Web Technology Surveys*. Retrieved July 31, 2024 (https://w3techs.com/technologies/details/ce-httpsdefault).

Wahyuni, Molli. 2020. *Statistik Deskriptif Untuk Penelitian Olah Data Manual Dan SPSS Versi 25*. YOgyakarta: Bintang Pustaka Madani.