ANALISIS PERBANDINGAN PERFORMA SERVER WEBSOCKET DENGAN MENGGUNAKAN PAYLOAD JSON, BINARY SERIALIZATION, DAN PROTOBUF DENGAN MENGGUNAKAN METODE LOAD TESTING

Raden Mohamad Rishwan, E Haodudin Nurkifli, Arip Solehudin
Informatika, Universitas Singaperbangsa Karawang
Jl. HS.Ronggo Waluyo, Puseurjaya, Telukjambe Timur, Karawang, Jawa Barat - Indonesia
radenrishwan@gmail.com

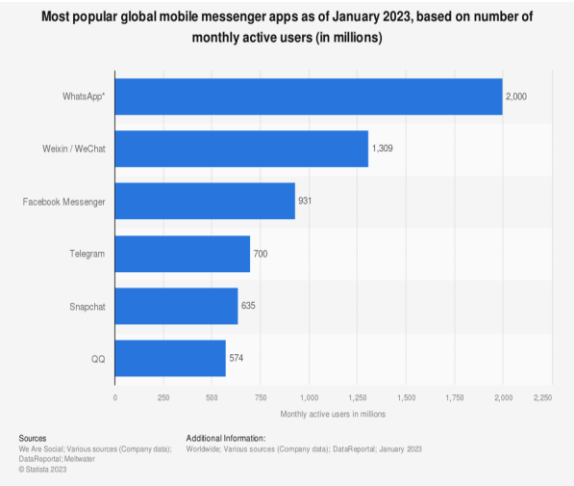
ABSTRAK

Dalam beberapa tahun terakhir dimana teknologi dipaksa untuk berjalan dengan lebih interaktif dan responsive baik itu untuk pengguna maupun aplikasi. Dimana solusinya yaitu dengan menggunakan websocket. Namun, penggunaan *websocket* memakan banyak *resource* dikarenakan koneksi pengguna yang disimpan di server. Salah satu cara untuk mengurangi penggunaan *resource* yaitu dengan cara merubah payload yang digunakan untuk mengirim dan menerima data antara server dan pengguna. Aplikasi dibangun menggunakan golang untuk servernya dan *flutter* untuk aplikasi *mobile* dengan menggunakan metode waterfall lalu dilakukan tes menggunakan metode *load testing* dengan menggunakan K6. Untuk mendapatkan data penggunaan CPU, memori, dan *network traffic* dengan menggunakan *node exporter* dan untuk menampilkan datanya dengan menggunakan *grafana*. Hasil yang didapat bahwa, penggunaan JSON mendapatkan hasil tertinggi baik itu untuk kasus pesan pribadi maupun pesan grup. Sementara *binary serialization* dan *protobuf* menunjukkan penggunaan resource yang lebih optimal dibanding JSON, dimana *protobuf* sedikit lebih baik dalam penggunaan memori namun memiliki penggunaan CPU yang relatif sama dengan *binary serialization*. Kesimpulan yang didapat bahwa dengan menggunakan *binary serialization* atau *protobuf* mampu mengurangi *resource* yang digunakan oleh server.

Kata kunci : *websocket, benchmark, load testing, json, binary serialization, protocol buffer.*

1. PENDAHULUAN

Dalam beberapa tahun terakhir, teknologi semakin berkembang pesat. Dimana salah satu dampak dari perkembangan teknologi yaitu aplikasi dipaksa supaya berjalan lebih interaktif dan responsif untuk pengguna. Salah satu solusinya yaitu aplikasi tersebut dibuat *realtime* seperti aplikasi chat. Dimana aplikasi chat seperti whatsapp dan telegram memanfaatkan teknologi *websocket* supaya aplikasinya berjalan lebih interaktif dan responsif.



Gambar 1. Grafik penggunaan aplikasi mobile messanger pada januari 2023 berdasarkan pengguna aktif bulanan

Berdasarkan data yang didapatkan dari statista, pengguna aplikasi chat pada januari 2023 pengguna aktif bulannya mencapai lebih dari 500 juta dari 6 aplikasi terpopuler. Bahkan, aplikasi Whatsapp memiliki 2 miliar pengguna per bulannya.

Dengan pengguna yang dapat mencapai angka sebanyak itu memerlukan server dengan spesifikasi tinggi untuk menanganinya yang otomatis menambah biaya lebih untuk menyewa lebih banyak server. Dimana cara kerja websocket yaitu menyimpan state atau koneksi pengguna di server yang membuat server memerlukan banyak aplikasi untuk menyimpan koneksi tersebut. Salah satu cara untuk memperkecil *traffic* dari client yaitu dengan mengubah payload pada *websocket* server.

Studi ini yaitu melakukan tes dan membandingkan hasil dengan cara load testing, yaitu melakukan tes dengan menaikan koneksi secara bertahap.

2. TINJAUAN PUSTAKA

2.1. Websocket

Websocket merupakan salah satu teknologi yang memungkinkan melakukan komunikasi dua arah (klien - server) tidak seperti HTTP yang hanya berkomunikasi 1 arah saja [1]. Websocket biasanya berjalan diatas HTTP untuk melakukan koneksi ke klien. Selain itu, protokol dari Websocket merupakan stateful yang berarti koneksi antara klien dan server akan tetapi hidup hingga salah satu pihak (klien atau server) mengakhirinya.

2.2. Unified Modeling Language (UML)

UML (Unified Modeling Language) merupakan sebuah teknik penggambaran visual yang dimanfaatkan dalam proses perancangan dan pengembangan software berorientasi objek. UML berperan sebagai panduan standar atau cetak biru yang memuat beragam alur proses bisnis [2].

2.3. Load Testing

Load testing merupakan salah satu cara untuk melakukan pengujian dengan cara menambah beban kepada aplikasi secara bertahap untuk melihat perubahan yang terjadi pada aplikasi [3]. Sebagai contoh misalnya suatu website yang di uji coba dengan cara menambah user yang mengunjungi website tersebut dalam satu waktu dan terus ditambah pengunjungnya.

2.4. Waterfall

Waterfall termasuk salah satu metode dalam SDLC (*Software Development Life Cycle*) atau Siklus Hidup Pengembangan Perangkat Lunak. Dalam metode waterfall, pengembangan software dilakukan secara bertahap dan berurutan, dimana tiap tahapan harus diselesaikan terlebih dahulu sebelum berpindah ke tahapan selanjutnya [4].

2.5. Docker

Docker merupakan platform pernagkat lunak *open-source* paling populer yang digunakan untuk mengelola dan menjalankan container [5].

2.6. Container

Container merupakan salah satu teknologi yang memungkinkan untuk mengemas sebuah aplikasi dan menjalankannya secara terisolasi dari lingkungan komputasi utamanya [6].

2.7. Promotheus

Prometheus adalah sistem pemantauan sumber terbuka (*open source*) yang digunakan untuk mengumpulkan, mengolah, dan memvisualisasikan metrik dan data kinerja aplikasi dan infrastruktur [7]. Prometheus dibangun untuk memantau sistem yang sangat skala besar dan dirancang dengan fokus pada kehandalan, skalabilitas, dan pengoperasian yang sederhana.

2.8. Grafana

Grafana adalah sebuah platform visualisasi dan analisis data yang digunakan untuk memonitor dan memvisualisasikan data dari berbagai sumber, termasuk sistem pemantauan seperti *prometheus* [7]. Grafana memberikan kemampuan untuk membuat dashboard interaktif, grafik, dan panel visual yang memudahkan pemahaman dan analisis data secara real-time.

2.9. Node Exporter

Node exporter adalah perangkat lunak untuk mengumpulkan data metrik yang digunakan di dalam ekosistem *promotheus* [8]. Alat ini biasanya digunakan untuk mengumpulkan metrik seperti penggunaan CPU, *memory*, I/O, *network traffic*, dll.

2.10. Golang

Golang merupakan salah satu bahasa pemrograman *open-source* yang dikembangan oleh Google. Bahasa ini dirancang oleh Robert Griesemer dan kawan - kawan dan diumumkan ke publik pertama kali pada tahun 2009 [9]. Tujuan bahasa Go yaitu untuk menyediakan bahasa pemrograman yang modern, efisien, dan sederhana untuk membangaun sistem perangkat lunak.

2.11. Flutter

Flutter adalah *framework* pengembangan aplikasi yang bersifat terbuka (*open source*) yang dikembangkan dan disponsori oleh google yang digunakan untuk mengembangkan aplikasi di berbagai platform seperti android dan IOS [10].

2.12. Protocol Buffer

Protobuf (*Protocol Buffer*) merupakan format data yang di compile dan di serialize ke sebuah bit yang dipresentasikan ke decimal value [11]. Protobuf bisa digunakan di berbagai pemrograman. Protobuf memiliki format data struktur yang mendukung berbagai tipe data. Berikut merupakan proses bagaimana protobuf melakukan *serialize* dan *deserialize*.

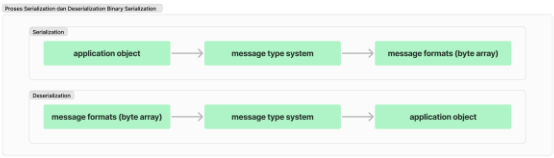
Gambar 2. Proses encode dan decode *protocol buffer*

2.13. Javascript Object Notation (JSON)

JSON (*Javascript Object Notation*) merupakan salah satu *scheme*-less dan textual *serilization* yang diadaptasi dari bahasa pemrogramaan *Javascript* [12]. JSON biasanya digunakan untuk melakaukan pengiriman data antara server dan *klien*.

2.14. Binary Serialization

Binary Serialization merupakan sebuah proses untuk melakukan kompilasi sebuah data struktur atau sebuah objek ke sebuah format yang dapat disimpan atau di kirim (sebagai contoh dikirim melalui internet) yang nantinya akan di dekompilasi agar dapat dibaca kembali [13].



Gambar 3. Proses encode dan decode *binary serialization*

3. METODE PENELITIAN

Pada tahapan ini terdapat 4 tahapan yaitu studi pustaka, Perancangan server, Perancangan tes, dan terakhir yaitu mengumpukan data tes.

Gambar 4. Tahapan Penelitian

3.1. Studi Pustaka

Pada tahapan ini yaitu melakukan studi pustaka yaitu mencari referensi mengenai websocket, dan payload yang digunakan yaitu JSON, *Binary Serialization*, dan *Ptotobuf*.

3.2. Perancangan Aplikasi

Setelah melakukan studi pustaka, selanjutnya yaitu pembuatan aplikasi dan server. Aplikasi dibuat untuk memastikan bahwa server yang dibuat sudah berjalan memenuhi kebutuhan. Dimana aplikasi dibangun dengan menggunakan metode Waterfall

3.3. Perancangan Tes

Setelah melakukan studi pustaka, akan dibuat skenario tes yang nantinya digunakan untuk melakukan *load testing* untuk server.

3.4. Mengumpulkan Data Tes

Setelah membuat aplikasi dan juga skenario tes, selanjutnya yaitu melakukan pengumpulan data tes.

4. HASIL DAN PEMBAHASAN

4.1. Perancangan Aplikasi

Server dirancang dengan menggunakan metode waterfall yang dimana memiliki beberapa tahapan.

4.2. Requirements

Pada tahap ini yaitu melakukan analisis hal apa saja yang diperlukan. Dimana untuk mengetahuinya penulis melakukan penelitian dengan menggunakan apikasi chat popular yang menggunakan teknologi websocket. Dimana hasilnya dibagi menjadi 2 yaitu *Functional* dan *Non-Functional*.

a. *Functional Requirement*

Kebutuhan functional yaitu kebutuhan yang berhubungan lansung dengan pengguna. Dimana aplikasi harus memungkinkan untuk melakukan komunikasi secara 2 arah baik itu server dengan klien ataupun klien dengan klien. Selain itu, aplikasi harus memiliki fiutr *chat* baik itu *private message* (antar klien) atau pesan grup baik itu bersifat *open public* ataupun tertutup.

b. *Non-Functional Requirement*

Kebutuhan non fungsi adalah kebutuhan yang tidak langsung dengan pengguna. Dimana berikut merupakan kebutuhan non fungsi.

- Aplikasi harus *realtime* (dengan batas wajar response dibawah 300 *miliseconds*).
- Aplikasi harus memiliki antarmuka yang familiar dengan aplikasi chat mainstream.

4.3. Analysis

Hasil dari analisis yaitu merupakan list fitur yang akan dibuat untuk server. Berikut merupakan fitur yang harus dipenuhi dan di implementasikan pada server.

Tabel 1. Analisis Fitur Aplikasi

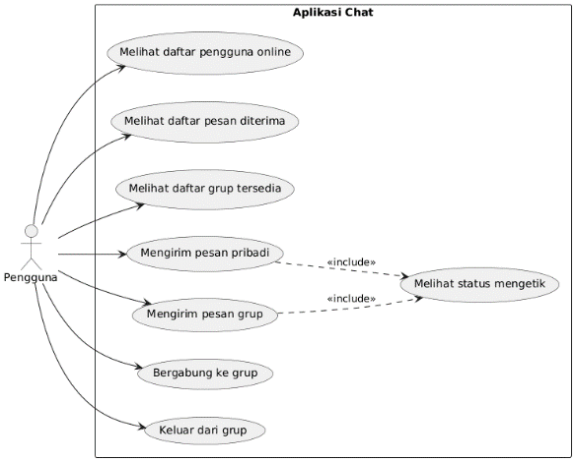
Fitur	Penjelasan
Chat	Pengguna dapat melakukan komunikasi 2 arah yaitu antara *klien* A dengan *klien* B.
Grup	Pengguna dapat membuat sebuah Group atau Room dan dapat di atur baik menjadi private ataupun public.
Notifikasi	Pengguna dapat melihat bahwa user sedang mengetik atau pengguna mendapatkan notifikasi Ketika ada yang keluar atau masuk ke dalam grup.

4.4. Design

Hasil pada tahap ini dibagi menjad 2 yaitu diagram *Usecase* dan diagram *Class.*

a. *Usecase Diagram*

Diagram *Usecase* adalah salah satu jenis diagram UML yang menggambarkan interaksi antara pengguna (actor) dengan sistem. Diagram ini menunjukkan fungsionalitas yang diharapkan dari sebuah sistem dan bagaimana sistem tersebut berinteraksi dengan dunia luar. Berikut merupakan diagram *usecase* dari aplikasi yang dibuat.

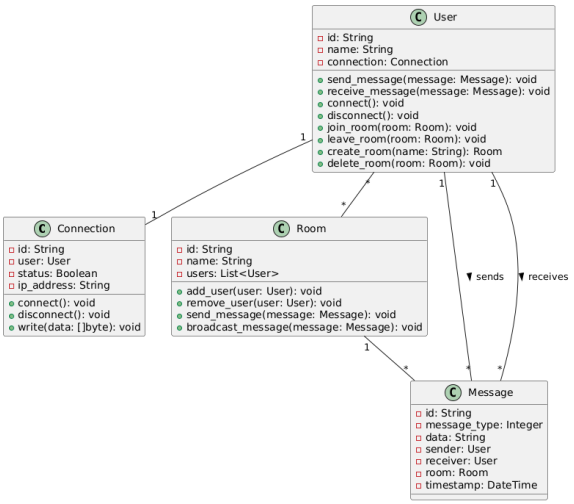


Gambar 5. Diagram *Usecase* untuk aplikasi

b. *Class Diagram*

Class diagram berfungsi untuk menampilkan objek dalam sistem dengan cara mendefinisikan suatu kelas yang akan diimplementasikan pada aplikasi yang dibuat. fungsinya yaitu untuk

menggambarkan perilaku sebuah sistem dari luar atau external.

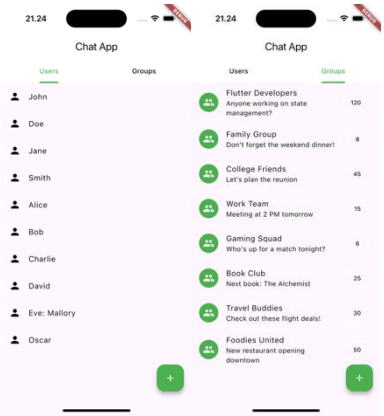


Gambar 6. Diagram *Class* untuk aplikasi.

4.5. Coding & Implementation

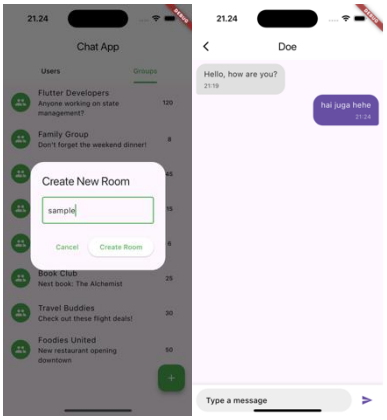
Pada tahap ini, yaitu membangun aplikasi. Dimana server dibangun menggunakan bahasa *Golang*. Untuk aplikasi menggunakan *Flutter* dan untuk monitoring menggunakan *Grafana* dan *Node Exporter*. Implementasi aplikasi dibagi menjadi 3 bagian yaitu impementasi server, implementasi aplikasi mobile, dan monitoring.

a. Implementasi Server
   Server Dibangun dari awal, yang dimana artinya pembuatan server HTTP lalu *websocket*. Hal ini dilakukan untuk mengurangi abstraksi jika menggunakan *library*. Dimana pembuatan server mengikuti standar RFC2616 dan untuk *websocket* dibangun mengikuti standar RFC6455.
b. Implementasi Aplikasi *Mobile*
   Berikut adalah hasil tampilan aplikasi yang dibangun menggunakan *flutter*.



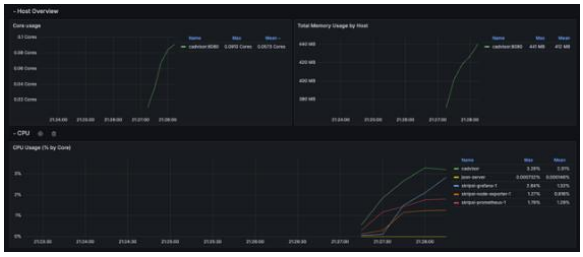
Gambar 7. Tampilan home pada aplikasi mobile

Selain itu, ada juga tampilan chat yang dimana *digunakan* untuk interaksi antar penggun dan juga tampilan *popup* untuk membuat grup baru.



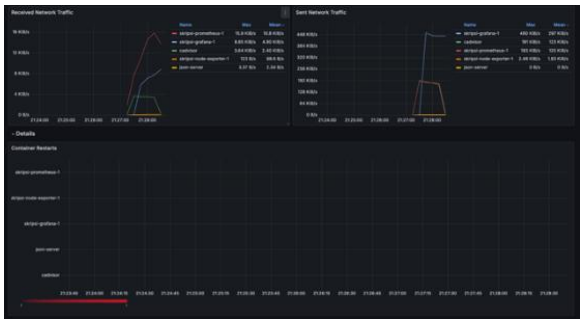
Gambar 8. Tampilan pembuatan grup dan chat.

c. Monitoring
   Untuk memonitoring penggunaan seperti penggunaan CPU, memori, dan *network traffic* yaitu dengan menggunakan *grafana* untuk menampilkan dat dan untuk mendapatkan datanya yaitu dengan menggunakan *node exporter*.



Gambar 9. Tampilan dashboard untuk penggunaan CPU dan memori

Gambar 10. Tampilan dashboard untuk pengunaan memori dan I/O



Gambar 11. Tampilan dashboard untuk network *traffic*

4.6. Testing

Setelah pembangunan aplikasi, selanjutnya yaitu melakukan pengetesan. Dimana pengetesan dilakukan dengan menggunakan Grafana k6.

a. Usecase

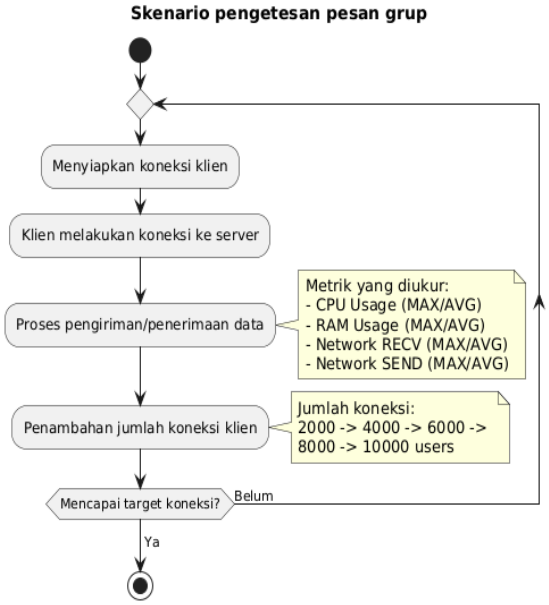
Berikut merupakan kasus atau usecase yang akan dilakukan pengujian.

Tabel 2. Kasus yang akan digunakan untuk *load testing*

No.	Usecase	Penjelasan
1.	Pesan pribadi	Pengguna terkoneksi secara bersamaan ke server dan melakukan pesan pribadi antar pengguna
2.	Pesan grup	Pengguna membuat grup dan masuk ke dalam grup yang dibuat lalu melakukan interaksi.

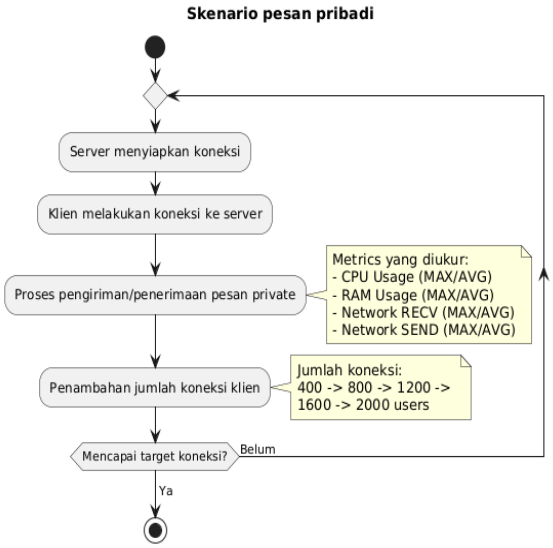
b. Skenario

Berikut merupakan skenario pengujian load testing yang dilakukan. Dimana untuk rentang waktu yang ditentukan pengguna akan bertambah.



Gambar 12. Skenario pengetesan pesan grup

Pada skenario pengetesan untuk pesan grup. Dimana pada periode waktu yang ditentukan akan ada penambahan koneksi klien ke server. Sebagai contoh, pada fase pertama, jumlah koneksi klien yaitu sebesar 2000. Setelah rentan waktu yang ditentukan, jumlah koneksi klien ke server ditambah hingga mencapai target.



Gambar 13. Skenario pengetesan pesan pribadi

Pada skenario pengetesan untuk pesan pribadi juga dilakukan hal yang sama, namun yang berbeda jumlah koneksinya lebih sedikit dikarenakan pada pesan pribadi memerlukan *resource* yang lebih banyak.

c. Spesifikasi server

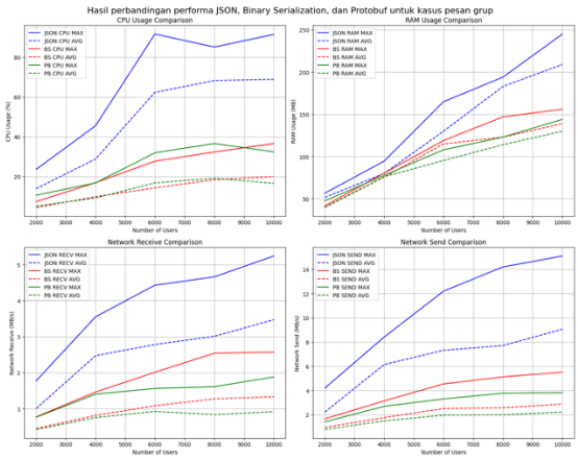
Untuk pengujian, yaitu dengan menggunakan spesifikasi pada tabel dibawah. Hal ini dilakukan supaya mendapatkan hasil yang konsisten dan adil.

Tabel 3. Spesifikasi server yang digunakan untuk pengujian

OS	Ubuntu 24.04.1 LTS x86_64
Memory	2 GB
Runtime	go1.23.4 linux/amd64
Kernel	6.1.112+
CPU	Intel Xeon (2) @ 2.200GHz

4.7. Hasil Pengetesan

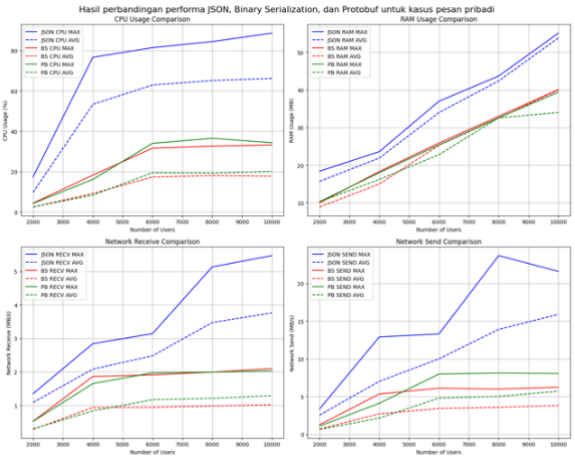
a. Hasil perbandingan untuk kasus pesan grup



Gambar 13. Hasil perbandingan performa untuk kasus pesan pribadi

Berdasarkan hasil pengujian untuk kasus pesan grup, dimana JSON menunjukkan penggunaan rata-rata CPU sebesar 69%, memori sebesar 209MB, penerimaan data sebesar 3.5MB/s, dan pengiriman data sebesar 9MB/s pada 10.000 pengguna. Sedangkan *biinary serialization* mendapatkan penggunaan rata-rata CPU sebesar 20%, memori 139MB, penerimaan data 1.33MB/s, dan pengiriman data 3MB/s. Protobuf mendapatkan penggunaan rata-rata CPU sebesar 17%, memori 130MB, penerimaan data 1MB/s, dan pengiriman data 2MB/s pada jumlah pengguna yang sama.

b. Hasil perbandingan untuk kasus pesan pribadi



Gambar 14. Hasil perbandingan performa untuk kasus pesan pribadi

Hasil yang didapat pada kasus pesan pribadi dengan 2.000 pengguna, JSON mendapatkan penggunaan rata-rata CPU sebesar 66%, memori 54MB, penerimaan data 4MB/s, dan pengiriman data 16MB/s. binary serialization mendapatkan penggunaan rata-rata CPU 18%, memori 40MB, penerimaan data 1MB/s, dan pengiriman data 4MB/s. Sementara Protobuf mendapatkan penggunaan rata-rata CPU 20%, memori 34MB, penerimaan data 1.5MB/s, dan pengiriman data 6MB/s.

5. KESIMPULAN DAN SARAN

Berdasarkan hasil pengujian yang telah dilakukan, dapat disimpulkan bahwa merancang aplikasi dengan menggunakan metode waterfall mampu menghasilkan sistem yang lebih baik dikarenakan memiliki tahapan yang jelas dan testruktur. Selain itu, dengan menggunakan *node exporter* dan *prometheus* untuk mendapatkan data dan menampilkannya menggunakan dengan grafana dashboard dapat membantu dalam memantau kinerja sistem secara realtime dalam hal ini berarti server websocket dengan lebih baik. Selanjutnya berdasarkan hasil pengujian, penggunaan payload binary serialization dan protobuf mampu mengoptimalkan penggunaan *resource* pada server. Dimana pada skenario pesan grup, binary serialization dan protobuf mampu memangkas penggunaan CPU yang awalnya 69% menjadi sekitar 20% dan 17%, penggunaan memori yang awalnya 209MB menjadi 130MB, penerimaan data dari 3.5MB/s menjadi 1MB/s dan pengiriman data dari 9MB/s menjadi 3MB/s. Dan pada skenario pesan pribadi juga mendapatkan hasil yang mirip yaitu penggunaan binary serialization dan protobuf lebih optimal dibandingkan dengan JSON.

Untuk penelitian selanjutnya, ada beberapa saran yaitu dapat menggunakan format payload yang lebih eifisien dibandingkan yang sudah diuji. Selain itu, dapat juga melakukan pengujian dengan skenario yang lebih kompleks untuk mengevaluasi server yang sudah dibuat misalnya menambahkan fitur multimedia atau video call yang memerlukan resource yang lebih banyak.

DAFTAR PUSTAKA

[1] A. Diaconu, “The WebSocket Handbook,” pp. 17–37, 2022, [Online]. Available: https://pages.ably.com/hubfs/the-websocket-handbook.pdf

[2] M. Monika Septa Laia, E. Panca Saputra, J. Kramat Raya No, K. Kwitang, and J. Pusat, “PERANCANGAN SISTEM INFORMASI AKADEMIK SEKOLAH BERBASIS WEB STUDI KASUS SDN 075076 HILINAMONIHA,” *Jurnal Riset dan Aplikasi Mahasiswa Informatika (JRAMI)*, vol. 05, no. 1, pp. 164–172, 2024.

[3] H. M. Alghamdi, | Cor-Paul Bezemer, | Weiyi Shang, | Ahmed, E. Hassan, and P. Flora, “Towards Reducing the Time Needed for Load Testing.”

[4] A. Reservasi, L. Futsal, B. Web, T. Ardiansah, and D. Hidayatullah, “Penerapan Metode Waterfall Pada,” *Journal of Information Technology, Software Engineering, and Computer Science (ITSECS)*, vol. 1, no. 1, 2023, Accessed: Dec. 15, 2024. [Online]. Available: https://doi.org/10.58602/itsecs.v1i1.8

[5] A. Mahandis Shama and D. W. Chandra, “Implementasi Static Application Security Testing Menggunakan Jenkins CI/CD Berbasis Docker Container Pada PT. Emporia Digital Raya,” Sep. 2021. doi: https://doi.org/10.33884/jif.v9i02.3769.

[6] M. R. Alamsyah and H. I. Wahanani, “PENERAPAN DOCKER UNTUK MEMBANGUN INFRASTRUKTUR PRIVATE CLOUD STORAGE BERBASIS IAAS,” Jul. 2021. Accessed: Oct. 28, 2024. [Online]. Available: https://d1wqtxts1xzle7.cloudfront.net/107703756/161-libre.pdf?1700728719=&response-content-disposition=inline%3B+filename%3DPenerapan_Docker_Untuk_Membangun_Infrast.pdf&Expi

res=1730133303&Signature=A9tQblKrHCPgGrctWq8r~bK2h2UKgn-aMdCS9GlIH0Il-o9ArK-Bpq05RCgsBE4Tm9TwXrA6SWr234~GZmCKHMtsRkaa8ifPFuWUnuCRxhPZjnP60bqFLHFM5po2y6GA0OP6PdqaZr6MEDt3anynfh9yrTqjBE3VMIEkYG5EGD0R1sOdEvrBSduaZ427I-XfFr1Jh1ATBaizW00fwVc8y3nkQSovTMXfaMZwrfkiLj45MYneMLGH9f8b5FIz81PZ75VDdvGnVZ7X2tivelWeLT673obPLOFYIdswQ05ashpgdKVGRIvfG3~a-U5b0iErO5YOUdV1G9gjtWvvJ1uXcQ__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA

[7] Ramadoni, M. Zunus Amirudin, R. Fahmi, E. Utami, and M. Syukri Mustafa, "Evaluasi Penggunaan Prometheusdan Grafana Untuk Monitoring Database Mongodb," *JIP (Jurnal Informatika Polinema)*, vol. 2, no. 2, pp. 43–50, Feb. 2021, Accessed: Oct. 28, 2024. [Online]. Available: https://doi.org/10.33795/jip.v7i2.530

[8] B. Rasyidi and F. Pratama, "MALCOM: Indonesian Journal of Machine Learning and Computer Science Server Monitoring System at PT. XYZ Media Indonesia Based on Grafana and Prometheus Sistem Monitoring Server di PT. XYZ Media Indonesia Berbasis Grafana dan Prometheus," vol. 4, no. 4, pp. 1456–1465, Sep. 2024, doi: 10.57152/malcom.v4i4.1546.

[9] Suwarno and A. Putri Yulandi, "Analisis Performa Backend Framework: Studi Komparasi Framework Golang dan Node.js," vol. 8, no. 1, pp. 155–168, Feb. 2023, doi: http://dx.doi.org/10.30645/jurasik.v8i1.551.

[10] N. Sofi and R. Dharmawan, "PERANCANGAN APLIKASI BENGKEL CSM BERBASIS ANDROID MENGGUNAKAN FRAMEWORK FLUTTER (BAHASA DART)," *JTS*, vol. 1, no. 2, pp. 53–4, Jun. 2022, Accessed: Oct. 28, 2024. [Online]. Available: https://journal.admi.or.id/index.php/JTS/article/view/125

[11] C. Currier, "Protocol Buffers," in *Mobile Forensics – The File Format Handbook*, Springer International Publishing, 2022, pp. 223–260. doi: 10.1007/978-3-030-98467-0_9.

[12] B. Huang and Y. Tang, "Research on optimization of real-time efficient storage algorithm in data information serialization," *PLoS One*, vol. 16, no. 12 December, Dec. 2021, doi: 10.1371/journal.pone.0260697.

[13] J. C. Viotti and M. Kinderkhedia, "A Benchmark of JSON-compatible Binary Serialization Specifications," Jan. 2022, [Online]. Available: http://arxiv.org/abs/2201.03051