JUSTIN
Jurnal Sistem dan Teknologi Informasi
p-ISSN : 2460-3562 / e-ISSN : 2620-8989
DOI: 10.26418/justin.v11i4.71607
Vol. 11, No. 4, Oktober 2023

Analisis Skalabilitas Web Server Apache Tomcat, Node.Js Dan Go Pada Protokol Hypertext Transfer Protocol (HTTP) Dan Message Queue Telemetry Transport (MQTT)

Steven Pragestu[a1], Herry Sujaini[a2], Eva Faja Ripanti[a3]
Program Studi Informatika, Fakultas Teknik, Universitas Tanjungpura
Jl. Prof. Dr. H. Hadari Nawawi, Pontianak 78124
[1]stevenpragestu@student.untan.ac.id
[2]hs@untan.ac.id
[3]evaripanti@untan.ac.id

Abstrak

Dalam era digital saat ini, web server memainkan peran yang sangat penting dalam melayani permintaan pengguna melalui protokol Hypertext Transfer Protocol (HTTP). Apache Tomcat adalah sebuah web server yang dapat menjalankan Java web application. Node.js adalah runtime environment JavaScript yang memiliki kecepatan eksekusi yang tinggi dan model non-blocking I/O. Web server Go adalah web server yang dibangun menggunakan bahasa pemrograman Go yang menawarkan kecepatan eksekusi yang tinggi, pengelolaan memory yang efisien, dan kemampuan penanganan concurrency yang baik. Selain protokol HTTP, protokol Message Queuing Telemetry Transport (MQTT) juga penting dalam industri Internet of Things (IoT). MQTT adalah protokol ringan yang digunakan untuk pertukaran pesan antara perangkat IoT. Penelitian mengenai analisis skalabilitas web server Apache Tomcat, Node.js, dan Go pada protokol HTTP dan MQTT dilakukan untuk mengetahui kemampuan dari masing-masing platform dalam menangani peningkatan beban kerja dan menemukan web server yang memiliki skalabilitas terbaik. Penelitian ini dilakukan secara offline pada topologi jaringan Local Area Network (LAN), menggunakan Mosquitto broker sebagai MQTT broker dan metode load testing dalam pengujian web server. Adapun skenario pengujian yang digunakan pada penelitian ini adalah masing-masing web server akan dikirim pesan yang berformat JSON oleh 200-1000 client setiap 500 ms selama 5 menit dan QoS 2 untuk publish-subscribe yang disimulasikan dengan menggunakan aplikasi Apache Jmeter. Dari analisis hasil pengujian, didapatkan bahwa pada web server Go pada protokol HTTP memiliki tingkat skalabilitas yang terbaik dibandingkan dengan web server pembanding lainnya dikarenakan memiliki peningkatan throughtput yang stabil seiring dengan meningkatnya jumlah user dan memiliki nilai throughput yang tertinggi.

Kata kunci: web server, protokol pesan, skalabilitas, pengujian beban, throughput.

Scalability Analysis of Apache Tomcat, Node.Js and Go Web Servers on Hypertext Transfer Protocol (HTTP) and Message Queue Telemetry Transport (MQTT)

Abstract

In today's digital era, web servers play a very important role in serving user requests via the Hypertext Transfer Protocol (HTTP) protocol. Apache Tomcat is a web server that can run Java web applications. Node.js is a JavaScript runtime environment that has high execution speed and a non-blocking I/O model. The Go web server is a web server built using the Go programming language which offers high execution speed, efficient memory management, and good concurrency handling capabilities. Apart from the HTTP protocol, the Message Queuing Telemetry Transport (MQTT) protocol is also important in the Internet of Things (IoT) industry. MQTT is a lightweight protocol used for message exchange between IoT devices. Research on the scalability analysis of the Apache Tomcat, Node.js, and Go web servers on the HTTP and MQTT protocols was carried out to understand the ability of each platform to handle increasing workloads and find the web server that has the best scalability. This research was carried out offline on a Local Area Network (LAN) network topology, using the Mosquitto broker as an MQTT broker and a load testing method in web server testing. The test scenario used in this research is that each web server will be sent messages in JSON format by 200-1000 clients every 500 ms for 5 minutes and QoS 2 for publish-subscribe which is simulated using the Apache Jmeter application. From the analysis of test results, it was found that the Go web server on the HTTP protocol has the best level of scalability compared to other comparative web servers because it has a stable throughput increase along with the increase in the number of users and has the highest throughput value.

Submitted 09-09-2023; Revised 09-10-2023; Accepted 09-10-2023

Keywords: web server, messaging protocol, scalability, load testing, throughput.

I. PENDAHULUAN

Dalam era digital saat ini, web server memainkan peran yang sangat penting dalam melayani permintaan pengguna melalui protokol Hypertext Transfer Protocol (HTTP). Web server Apache Tomcat, Node.js, dan Go adalah beberapa web server yang menjalankan aplikasi yang ditulis dengan bahasa pemrograman yang populer berdasarkan TIOBE Index, yaitu Java, Javascript dan Go [1].

Apache Tomcat adalah sebuah web server yang dapat menjalankan Java web application [2]. Apache Tomcat sendiri menjadi pilihan populer bagi pengembang web application yang menggunakan teknologi Java Enterprise Edition (Java EE), dimana menurut survey yang dilakukan oleh Jetbrains, sebanyak 59% dari pengembang aplikasi Java menggunakan Apache Tomcat sebagai application server [3] .

Node.js adalah runtime environment JavaScript yang memungkinkan pengembang aplikasi untuk menjalankan bahasa pemrograman JavaScript di sisi server. Node dibangun di atas mesin JavaScript V8 [4]. Dengan kecepatan eksekusi yang tinggi dan model non-blocking I/O, Node.js sering digunakan dalam pengembangan aplikasi real-time, seperti aplikasi chat atau streaming [5]. Jika dibandingkan dengan web server lebih populer seperti Apache HTTP, Nginx dan Python, menurut penelitian-penelitian sebelumnya Node.js memiliki performa yang lebih baik dibandingkan dengan web server tersebut [6], [7].

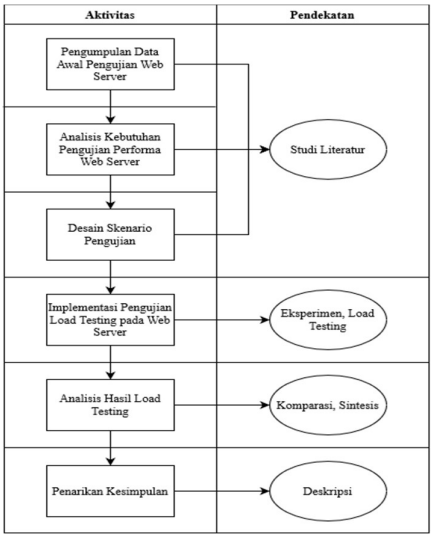
Web server Go adalah web server yang dibangun menggunakan bahasa pemrograman Go, Go sendiri adalah bahasa pemrograman yang dikembangkan oleh Google. Go menawarkan kecepatan eksekusi yang tinggi, pengelolaan memory yang efisien, dan kemampuan penanganan concurrency yang baik. Go telah digunakan secara luas dalam pengembangan aplikasi berkinerja tinggi dan sistem yang membutuhkan skalabilitas tinggi [8], [9].

Selain protokol HTTP, protokol Message Queuing Telemetry Transport (MQTT) juga penting dalam industri Internet of Things (IoT) dan aplikasi berbasis sensor. MQTT adalah protokol ringan yang digunakan untuk pertukaran pesan antara perangkat IoT. MQTT memungkinkan perangkat untuk berkomunikasi secara efisien dalam kondisi jaringan yang terbatas [10], [11].

Penelitian mengenai analisis skalabilitas web server Apache Tomcat, Node.js, dan Go pada protokol HTTP dan MQTT dilakukan untuk mengetahui kemampuan dari masing-masing platform dalam menangani peningkatan beban kerja. Dengan membandingkan skalabilitas ketiga web server ini, diharapkan pengembang aplikasi dapat memilih web server yang paling sesuai dengan kebutuhan mereka dalam menghadapi tantangan yang terkait dengan performa web server.

II. METODOLOGI PENELITIAN

Pada Gambar 1, diperlihatkan bahwa terdapat beberapa tahap dalam metodologi penelitian. Tahapan-tahapan tersebut terdiri dari:



Gambar 1. Metode Penelitian

A. *Pengumpulan Data Awal Pengujian Web Server*

Tahap ini mencakup pengumpulan data awal pengujian web server pada penelitian ini yang dilakukan dengan cara mengumpulkan literatur-literatur seperti jurnal, buku, dan literatur lainnya terkait metode-metode pengujian performa web server.

B. *Analisis Kebutuhan Pengujian Performa Web Server*

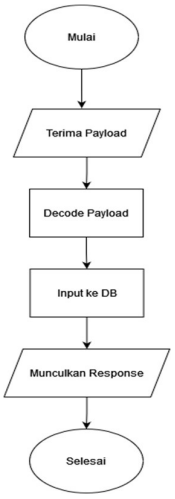
Tahap ini penulis melakukan analisis kebutuhan untuk pengujian performa web server dengan menggunakan data-data awal mengenai pengujian web server yang telah dikumpulkan, setelah itu didapatkan informasi-informasi mengenai metode-metode pengujian web server seperti metode load testing dan informasi mengenai alat dan bahan apa yang dibutuhkan dalam melakukan pengujian web server.

C. *Desain Skenario Pengujian*

Tahap ini penulis melakukan desain skenario pengujian yang dibagi menjadi 2 bagian yaitu perancangan setup perangkat client-server untuk pengujian dan perancangan kasus pengujian. Berdasarkan hasil analisis kebutuhan pengujian performa *web server* yang telah dilakukan, penulis melakukan perancangan setup perangkat client-server untuk pengujian, dimana perancangan skenario pengujian pada penelitian ini mengacu pada penelitian yang telah dilakukan oleh peneliti lainnya [6],[12] yaitu menggunakan topologi Local Area Network (LAN) dalam lingkungan pengujian untuk mengurangi masalah-masalah kualitas jaringan saat pengujian [13] yang dapat dilihat pada Gambar 2. Perangkat keras yang digunakan oleh peneliti pada penelitian ini 2 kabel LAN, 1 buah PC client, 1 buah PC server, dan 1 buah gateway. Untuk perangkat lunak yang digunakan pada penelitian ini adalah Sistem Operasi Windows 11, Apache Tomcat 10.1 dengan Java JDK 1.8.0_221, Node.js 18.15.LTS, Go 1.20.2, MariaDB 10.11.6, Mosquitto MQTT Broker v 2.0.15 dan Apache Jmeter 5.5 + mqtt-jmeter plugin.

Gambar 2. Setup Jaringan

Selain itu juga dilakukan perancangan kasus pengujian, mengacu pada kasus-kasus pengujian yang telah dilakukan pada penelitian sebelumnya [6], [12] penelitian ini menggunakan metode load testing untuk menguji performa dari web server. Load testing adalah teknik performance testing yang mana respon sistem diukur dalam berbagai load condition [14]. Pengujian ini membantu menentukan bagaimana software berperilaku ketika beberapa user mengakses software secara bersamaan. Load testing diperlukan untuk membuat simulasi akses aplikasi website secara simultan. Cara ini lebih baik dibandingkan dengan harus mengundang sekian belas, atau puluh orang sekaligus untuk mengakses sebuah website [15], [16]. Pada penelitian ini skenario load testing yang dirancang oleh penulis adalah menggunakan aplikasi Apache Jmeter untuk mensimulasikan request-request client [17] terhadap aplikasi dengan alur yang ditunjukkan oleh Gambar 3, dengan load sebanyak 200, 400, 600, 800 dan 1000 request selama 5 menit dengan interval setiap sampel yang dikirim adalah 500 ms. Adapun pengaturan tambahan untuk protokol MQTT adalah level QoS MQTT yang digunakan adalah QoS2. Level QoS2 ini digunakan agar publisher/subscriber hanya mengirimkan/menerima data hanya 1 kali tanpa duplikasi [18]. Data yang dikirim oleh client berupa data JSON dengan format sebagai berikut { "clientId": value, "temp": value, "humidity": value, "light": value }.



Gambar 3. Diagram Alir Aplikasi pada Web Server untuk Pengujian

D. *Implementasi Pengujian Load Testing pada Web Server*

Tahap ini penulis mengimplementasikan pengujian web server berdasarkan skenario pengujian yang telah didesain. Setelah pengujian dilakukan didapatkan hasil pengujian load testing yang berupa data-data seperti jumlah paket yang terkirim, jumlah paket yang diterima yang dikumpulkan oleh aplikasi Jmeter dengan plugin mqtt-jmeter. Data-data seperti status, waktu awal proses, dan waktu akhir proses setiap request pengujian didapatkan dari log web server. Nilai throughput dari setiap pengujian akan dihitung dengan memanfaatkan data jumlah transaksi yang berhasil diproses oleh web server dan durasi pengujian [19].

E. *Analisis Hasil Load Testing*

Tahap ini penulis melakukan analisis terhadap hasil pengujian dari pengujian yang telah dilakukan, adapun fokus disini adalah skalabilitas web server. Skalabilitas web server pada penelitian ini difokuskan pada kemampuan web server dalam menangani peningkatan load yang diberikan. Analisis skalabilitas web server dilakukan dengan membandingkan nilai throughput setiap web server dengan peningkatan beban/jumlah user yang didapatkan dari hasil load testing. Web server dengan skalabilitas terbaik adalah web server yang nilai throughput-nya terus meningkat seiring dengan peningkatan jumlah user dan memiliki nilai throughput yang tertinggi.

F. *Penarikan Kesimpulan*

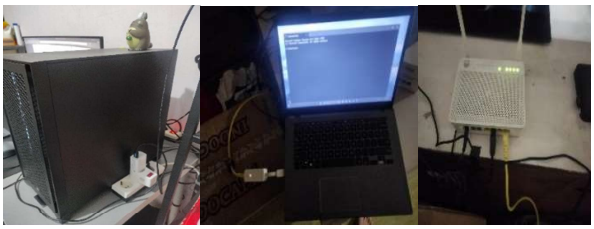
Tahap ini merupakan tahap akhir dari penelitian ini. Pada tahap ini kesimpulan dirumuskan berdasarkan hasil analisis pada pengujian yang telah dilakukan.

III. HASIL DAN PEMBAHASAN

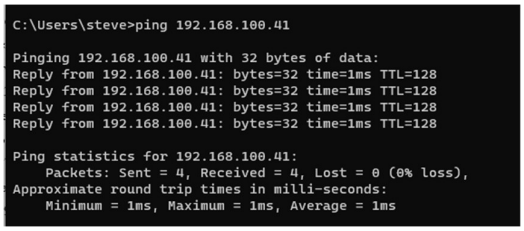
Bagian ini memaparkan secara terstruktur hasil yang didapatkan pada penelitian meliputi implementasi, hasil throughput web server dan analisis skalabilitas web server.

A. Implementasi

Hasil dari implementasi setup perangkat client-server untuk pengujian dapat dilihat pada Gambar 4 yang menunjukkan perangkat client-server yang digunakan untuk pengujian dan Gambar 5 yang menunjukkan kualitas jaringan antara client-server yang menghasilkan latency bernilai sekitar 1ms. Untuk konfigurasi aplikasi di server seperti konfigurasi web server, MQTT broker dan lainnya ditunjukkan pada Tabel I.



Gambar 4. Perangkat client untuk pengujian



Gambar 5. Hasil ping client ke server

TABEL I
PENGATURAN APLIKASI DI SERVER

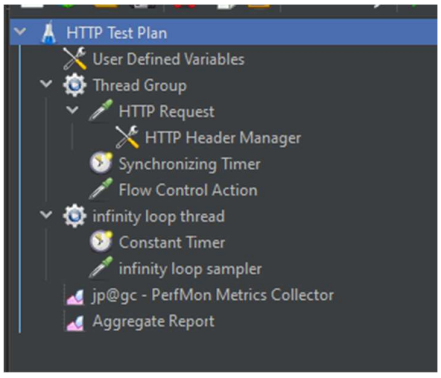
No	Item	Pengaturan
1	MariaDb	max_connections = 20000;

2	Mosquitto	max_connection = -1;
		max_queued_messages = 0;
		max_inflight_messages = 0;

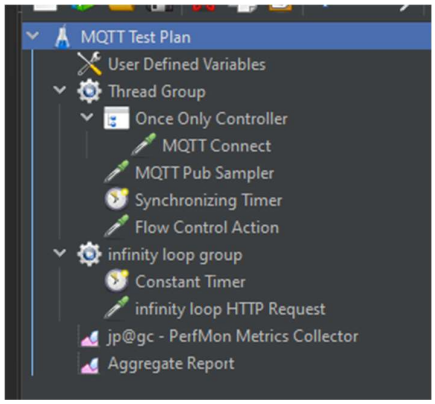
3	Web server Apache Tomcat	maxConnections = 10000; maxThreads = 10000;

4	MQTT subscriber pada setiap web server	keepAlive = 60 detik;
		cleanSession = false;
		autoReconnect = true (Tomcat);

Hasil dari implementasi perancangan kasus pengujian berupa test plan yang digunakan pada aplikasi Apache Jmeter saat melakukan load testing. Penulis membuat 2 buah test plan yaitu test plan pengujian web server untuk menguji web server pada protokol HTTP yang ditunjukkan oleh Gambar 6 dan test plan untuk menguji web server pada pada protokol MQTT yang ditunjukkan oleh Gambar 7.



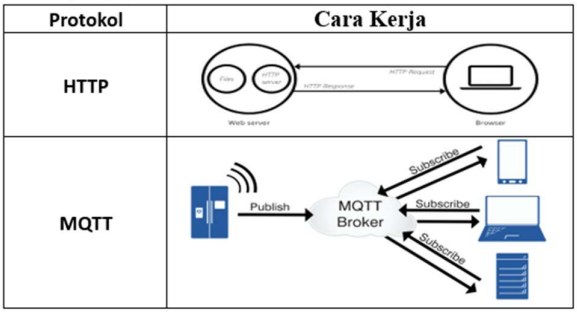
Gambar 6. Test Plan pengujian web server pada protokol HTTP



Gambar 7. Test Plan pengujian web server pada protokol MQTT

B. *Hasil Throughput Web Server*

Hasil throughput web server didapatkan dengan menggunakan data dari hasil pengujian yang telah dilakukan dan ditunjukkan pada Tabel II. Untuk perhitungan nilai throughput pada penelitian ini, dikarenakan pada implementasi protokol MQTT, client tidak mengirimkan pesan secara langsung kepada web server, tidak seperti pada implementasi protokol HTTP, melainkan melalui MQTT broker dan client tidak mendapatkan response dari server seperti yang ditunjukkan pada Gambar 8, oleh karena itu perhitungan throughput web server MQTT tidak bisa disamakan dengan perhitungan throughput HTTP yang biasanya menggunakan request. Oleh karena itu pada penelitian nilai throughput dihitung dengan menggunakan jumlah transaksi yang berhasil diproses oleh web server dibagi dengan total durasi pengujian.



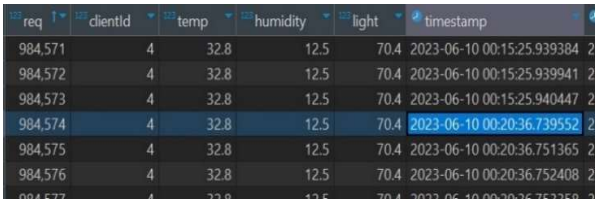
Gambar 8. Cara kerja protokol HTTP VS MQTT

TABEL II
HASIL PENGUJIAN WEB SERVER DAN PROTOKOL

No	Web Server	Protokol	Jumlah Users	Total Pesan yang Diterima	Total Durasi Pengujian (s)	Throughput (tps)
1	Go	HTTP	200	120000	317.7279	377.6816578
2	Go	HTTP	400	240000	332.849102	721.0474613
3	Go	HTTP	600	360000	338.401574	1063.824839
4	Go	HTTP	800	480000	343.688295	1396.614336
5	Go	HTTP	1000	600000	365.975599	1639.453564
6	Go	MQTT	200	120000	320.429677	374.4971475
7	Go	MQTT	400	240000	345.545537	694.553899
8	Go	MQTT	600	360000	371.397653	969.3114566
9	Go	MQTT	800	922146	1355.475323	680.3119056
10	Go	MQTT	1000	1924413	3231.224944	595.5676356
11	Node.js	HTTP	200	120000	364.317955	329.3826131
12	Node.js	HTTP	400	240000	432.185036	555.3176996
13	Node.js	HTTP	600	360000	495.387739	726.7034924

No	Web Server	Protokol	Jumlah Users	Total Pesan yang Diterima	Total Durasi Pengujian (s)	Throughput (tps)
14	Node.js	HTTP	800	480000	561.032255	855.565782
15	Node.js	HTTP	1000	600000	629.784356	952.7070564
16	Node.js	MQTT	200	120000	320.829344	374.0306248
17	Node.js	MQTT	400	240000	336.871335	712.4381776
18	Node.js	MQTT	600	360000	367.311858	980.0935967
19	Node.js	MQTT	800	480000	1254.327302	382.6752389
20	Node.js	MQTT	1000	600000	2845.318303	210.8727165
21	Apache Tomcat	HTTP	200	120000	333.719268	359.5836726
22	Apache Tomcat	HTTP	400	240000	359.62976	667.3530022
23	Apache Tomcat	HTTP	600	360000	391.92675	918.5389872
24	Apache Tomcat	HTTP	800	480000	435.350769	1102.55921
25	Apache Tomcat	HTTP	1000	600000	491.503421	1220.744301
26	Apache Tomcat	MQTT	200	120000	320.627	374.2666712
27	Apache Tomcat	MQTT	400	240000	338.15351	709.7368293
28	Apache Tomcat	MQTT	600	360000	369.237338	974.9826547
29	Apache Tomcat	MQTT	800	480000	1384.584171	346.6744818
30	Apache Tomcat	MQTT	1000	600000	1944.939879	308.4928262

Pada Tabel II, dapat dilihat bahwa web server Go, Node.js dan Apache Tomcat pada protokol HTTP mengalami kenaikan throughput yang signifikan seiring dengan bertambahnya jumlah user. Pada protokol MQTT, semua web server juga mengalami kenaikan throughput. Hanya saja pada saat pengujian dengan 800 dan 1000 user, semua web server mengalami penurunan throughput, hal ini dikarenakan adanya peningkatan total durasi yang diakibatkan oleh "jeda" pada Mosquitto broker ke web server (subscriber) seperti yang ditunjukkan pada Gambar 9, dimana pesan dikirimkan 5 menit kemudian. Selanjutnya juga ditemukan kegagalan Mosquitto Broker pada web server Go, dimana Mosquitto Broker mengirimkan pesan berlebih kepada web server, dimana jumlah pesan yang diterima oleh web server (subcriber) melebihi dari jumlah pesan yang dikirim oleh client (publisher).

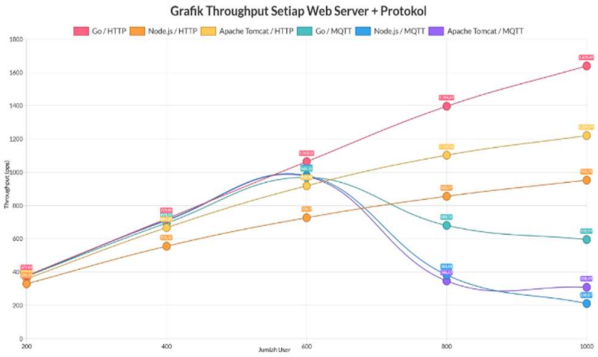


req	clientId	temp	humidity	light	timestamp
984,571	4	32.8	12.5	70.4	2023-06-10 00:15:25.939384
984,572	4	32.8	12.5	70.4	2023-06-10 00:15:25.939941
984,573	4	32.8	12.5	70.4	2023-06-10 00:15:25.940447
984,574	4	32.8	12.5	70.4	2023-06-10 00:20:36.739552
984,575	4	32.8	12.5	70.4	2023-06-10 00:20:36.751365
984,576	4	32.8	12.5	70.4	2023-06-10 00:20:36.752408

Gambar 9. Jeda publish pada Mosquitto broker

C. Analisis Skalabilitas Web Server

Analisis skalabilitas web server dilakukan terhadap hasil load testing yang telah dilakukan, skalabilitas setiap web server didapatkan dengan membandingkan nilai throughput dengan peningkatan jumlah user.



Gambar 10. Grafik Throughput Setiap Web Server dan Protokol

Pada Gambar 10 nilai throughput setiap web server pada protokol HTTP memiliki perbedaan nilai yang cukup jauh jika dibandingkan dengan nilai throughput setiap web server pada protokol MQTT yang hampir sama. Hal ini dikarenakan pada prokotol HTTP, web server menerima pesan secara bersamaan dari client sehingga nilai throughput web server tergantung pada kemampuan web server dalam menangani request dari sejumlah client secara bersamaan, dimana web server yang memiliki model pemrosesan multithread seperti web server Go dan Apache Tomcat memiliki keuntungan dimana web server ini dapat memproses request-request client secara bersamaan, di sisi lain web server Node.js yang memiliki model pemrosesan single thread event loop memiliki kecepatan yang paling rendah dikarenakan web server ini menangani request dari client satu per satu. Pada implementasi protokol MQTT, semua web server menerima pesan dari client melalui MQTT broker, dimana pesan ini dikirimkan oleh MQTT broker ke web server satu per satu yang mengakibatkan semua web server memproses pesan tersebut satu per satu juga, sehingga kecepatan setiap web server dalam menangani request dari client juga hampir sama.

Berdasarkan Gambar 10 yang menampilkan hasil pengujian throughput dari ketiga web server dan dua protokol, dilakukanlah analisis skalabilitas, dan didapatkan:

1. Web server Go/HTTP memiliki skalabilitas terbaik dari sisi throughput karena memiliki peningkatan throughput yang stabil seiring dengan peningkatan jumlah user dan memiliki nilai throughput yang paling tinggi.
2. Web server Apache Tomcat/HTTP memiliki skalabilitas dengan peringkat kedua dari sisi throughput karena memiliki peningkatan throughput yang stabil seiring dengan peningkatan jumlah user dan memiliki nilai throughput meskipun berada di bawah throughput web server Go/HTTP.
3. Web server Go/HTTP memiliki skalabilitas dengan peringkat ketiga dari sisi throughput karena memiliki

peningkatan throughput yang stabil seiring dengan peningkatan jumlah user dan memiliki nilai throughput paling rendah jika dibandingkan dengan web server lainnya pada protokol HTTP.

4. Web server Go/MQTT memiliki skalabilitas dengan peringkat keempat dari sisi throughput karena nilai throughput turun pada load dengan jumlah user 800 dan 1000 yang disebabkan oleh kegagalan pada broker Mosquitto, meskipun begitu dibandingkan web server pada protokol MQTT lainnya, web server Go/MQTT memiliki nilai throughput yang tertinggi.
5. Web server Apache Tomcat/MQTT memiliki skalabilitas dengan peringkat kelima dari sisi throughput karena nilai throughput turun pada load dengan jumlah user 800 dan 1000 yang disebabkan oleh kegagagalan pada broker Mosquitto, dan nilai throughput web server ini berada dibawah web server Go/MQTT.
6. Web server Node.js/MQTT memiliki skalabilitas dengan peringkat keenam dari sisi throughput karena nilai throughput turun pada load dengan jumlah user 800 dan 1000, dan memiliki nilai throughput terendah dibandingkan dengan web server lainnya pada protokol HTTP maupun MQTT.

Berdasarkan hasil analisa diatas, dapat diketahui juga bahwa web server pada protokol HTTP memiliki skalabilitas yang lebih baik dibandingkan dengan web server pada protokol MQTT, dimana 3 web server dengan protokol HTTP memiliki peringkat diatas semua web server dengan protokol MQTT.

IV. KESIMPULAN

Berdasarkan penelitian yang telah dilakukan, dimana penulis melakukan load testing terhadap web server Apache Tomcat, Node.js dan Go pada protokol HTTP/1.1 dan MQTT v3.1 menggunakan aplikasi Apache Jmeter, 1 buah Mosquitto MQTT broker dan dengan skenario pengiriman pesan ke web server selama 5 menit, interval pengiriman pesan sebesar 5 ms, jumlah user 200-1000 dan QoS 2 untuk MQTT, didapatkanlah kesimpulan yaitu:

1. Web server dengan performa terbaik dari sisi skalabilitas adalah Go pada protokol HTTP, dimana web server ini memiliki peningkatan throughtput yang stabil seiring dengan meningkatnya jumlah user dan memiliki nilai throughput yang tertinggi dibanding dengan web server pembanding lainnya.
2. Web server Go, Node.js dan Apache Tomcat pada protokol HTTP memiliki skalabilitas yang lebih baik dibandingkan dengan implementasinya pada protokol MQTT, hal ini dikarenakan terjadinya peningkatan total durasi pengujian pada pengujian dengan beban 800 dan 1000 user yang disebabkan oleh kegagalan pada Mosquitto MQTT Broker.

DAFTAR PUSTAKA

[1] TIOBE Index, "TIOBE Index for May 2023." Accessed: Jun. 02, 2023. [Online]. Available: https://www.tiobe.com/tiobe-index/

[2] A. D. Putra, W. Yahya, and A. Bhawiyuga, "Analisis Kinerja Dan Konsumsi Sumber Daya Aplikasi Web Server Pada Platform Raspberry P," *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*, vol. 3, no. 4, pp. 3513–3521, Apr. 2019.

[3] A. Sichkarenko, "Developer Ecosystem Survey 2022." Accessed: Jun. 02, 2023. [Online]. Available: https://www.jetbrains.com/lp/devecosystem-2022/java/

[4] Febrianto Widyoutomo, Hamidillah Ajie, and Widodo, "PENGEMBANGAN WEB SERVICE MODUL MAHASISWA PADA SISTEM INFORMASI AKADEMIK UNIVERSITAS NEGERI JAKARTA," *PINTER : Jurnal Pendidikan Teknik Informatika dan Komputer*, vol. 5, no. 1, pp. 68–75, Jun. 2021, doi: 10.21009/pinter.5.1.9.

[5] A. Mukasheva, S. Rakhmetulayeva, G. Astaubayeva, and S. Gnatyuk, "Developing a system for diagnosing diabetes mellitus using bigdata," *Eastern-European Journal of Enterprise Technologies*, vol. 5, no. 2(119), pp. 75–85, Oct. 2022, doi: 10.15587/1729-4061.2022.266185.

[6] A. C. Rompis, "Perbandingan Performa Kinerja Node.js, PHP, dan Python dalam Aplikasi REST," *CogITo Smart Journal*, vol. 4, no. 1, p. 160, Jun. 2018, doi: 10.31154/cogito.v4i1.92.160-170.

[7] I. K. Chaniotis, K.-I. D. Kyriakou, and N. D. Tselikas, "Is Node.js a viable option for building modern web applications? A performance evaluation study," *Computing*, vol. 97, no. 10, pp. 1023–1044, Oct. 2015, doi: 10.1007/s00607-014-0394-9.

[8] M. D. Lusita, H. Hurnianingsih, and E. Rihyanti, "Aplikasi Bot Akademik BAAK STMIK Jakarta STI&K Platform Line Messenger Menggunakan Go Languages," *Jurnal Teknologi Sistem Informasi dan Aplikasi*, vol. 3, no. 1, p. 1, Feb. 2020, doi: 10.32493/jtsi.v3i1.4130.

[9] Y. Harjoseputro, Albertus Ari Kristanto, and Joseph Eric Samodra, "Golang and NSG Implementation in REST API Based Third-Party Sandbox System," *Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi)*, vol. 4, no. 4, pp. 745–750, Aug. 2020, doi: 10.29207/resti.v4i4.2218.

[10] Dian Rachmadini, Ira Puspasari, and Jusak, "Implementasi dan Analisis Fitur Keamanan Protokol MQTT pada Telehealthcare," *Journal of Technology and Informatics (JoTI)*, vol. 2, no. 1, pp. 10–19, Oct. 2020, doi: 10.37802/joti.v2i2.117.

[11] C. F. Permatasari and H. Dhika, "Optimasi Jalur Transfer Data dari HTTP menjadi MQTT pada IoT menggunakan Cloud Services," *JISA(Jurnal Informatika dan Sains)*, vol. 1, no. 2, Dec. 2018, doi: 10.31326/jisa.v1i2.446.

[12] I. Vilhelmsson, H. R. Faragardi, and J. Håstad, "A Performance Comparison of an Event-Driven Node.js Web Server and Multi-Threaded Web Servers," KTH, School of Electrical Engineering and Computer Science (EECS), 2021.

[13] V. P. Prakash and H. Virani, "Performance Testing and Analysis for a Stable Ethernet Communication Link," *International Journal of Science and Research (IJSR)*, vol. 3, no. 5, pp. 867–871, May 2014.

[14] A. Ismail, Ahmadi Yuli Ananta, Sofyan Noor Arief, and Elok Nur Hamdana, "Performance Testing Sistem Ujian Online Menggunakan Jmeter Pada Lingkungan Virtual," *Jurnal Informatika Polinema*, vol. 9, no. 2, pp. 159–164, Feb. 2023, doi: 10.33795/jip.v9i2.1190.

[15] L. W. Suwarsono, A. N. Aisha, and F. N. Nugraha, "The Role of E-Learning Readiness on Workload: Perspective Engineering and non-Engineering Students," *International Journal of Innovation in Enterprise System*, vol. 6, no. 01, pp. 85–94, Jan. 2022, doi: 10.25124/ijies.v6i01.165.

[16] D. I. Permatasari *et al.*, "PENGUKURAN THROUGHPUT LOAD TESTING MENGGUNAKAN TEST CASE SAMPLING GORILLA TESTING," in *Seminar Nasional Sistem Informasi*, Malang: Fakultas Teknologi Informasi - UNMER Malang, Sep. 2019, pp. 2018–2014.

[17] N. L. A. S. Ginasari, K. S. Wibawa, and N. K. A. Wirdiani, "Pengujian Stress Testing API Sistem Pelayanan dengan Apache JMeter ," *JITTER- Jurnal Ilmiah Teknologi dan Komputer*, vol. 2, no. 3, pp. 552–557, Nov. 2021.

[18] N. L. Husni, R. Vira, D. Andika, A. S. Handayani, and S. Rasyad, "Monitoring dan Analisis Kualitas Kinerja Jaringan Protokol Message Queue Telemetry Transport pada G-Bot (Garbage Robot)," *Jurnal Ampere*, vol. 7, no. 1, pp. 39–48, Jun. 2022.

[19] R. Herwanto, H. Sabita, and F. Armawan, "Measuring Throughput and Latency Distributed Ledger Technology: Hyperledger," *Journal of Information Technology Ampera*, vol. 2, no. 1, pp. 17–31, Jul. 2021, doi: 10.51519/journalita.volume2.isssue1.year2021.page17-31.