Journal of Applied Informatics and Computing (JAIC)
Vol.10, No.1, February 2026, pp. 900~909
e-ISSN: 2548-6861

Performance of Load Balancing Algorithms on Homogeneous and Heterogeneous Servers in On-Premise Environments

Tsabitah Avia Aulia Faridah [1]*, Galura Muhammad Suranegara [2]*
* Departement of Telecommunication system, Universitas Pendidikan Indonesia
tsabitahavia@upi.edu [1], galurams@upi.edu [2]

Article Info

Article history:

Received 2025-10-31
Revised 2025-12-01
Accepted 2025-12-29

Keyword:

Load Balancing,
NGINX,
Server Performance,
On-Premise System.

ABSTRACT

This research evaluates the performance of Round Robin, IP Hash, and Random Allocation algorithms in a homogeneous server environment, as well as Least Response Time, Least Connection, and Weighted Least Connection algorithms in a heterogeneous server environment implemented on on-premise servers. This study was motivated by the need to improve traffic management efficiency in local server infrastructure, where system performance is greatly influenced by resource diversity and distribution strategies. The experimental method was applied using NGINX and NGINX Plus as load balancing platforms, with Apache JMeter as a testing tool with low, medium, and high load test scenarios, while Netdata monitored the load distribution in real-time. Performance evaluation was based on six key metrics: throughput, latency, error rate, load distribution, CPU usage, and memory consumption. The results show that in a homogeneous environment, static algorithms such as Round Robin, IP Hash, and Random Allocation maintain stable performance with consistent throughput and minimal latency. Conversely, in a heterogeneous environment, dynamic algorithms, such as Weighted Least Connection, achieve lower latency and more balanced resource utilization. These findings highlight that algorithm selection must match system characteristics: static algorithms are more suitable for small-scale, uniform deployments, while dynamic approaches are recommended for heterogeneous or large-scale systems that require adaptive load management. Overall, weight-based dynamic approaches demonstrate superior scalability and resilience under high workloads.

I. PENDAHULUAN

Perkembangan teknologi informasi yang pesat telah mengubah sistem komputasi modern dalam menyediakan layanan yang cepat, efisien, dan andal. Seiring meningkatnya ketergantungan masyarakat terhadap layanan digital, tuntutan terhadap sistem server dengan *high availability* dan *low latency* semakin tinggi. Peningkatan jumlah permintaan pengguna (*request*) yang bersifat dinamis sering kali menimbulkan *bottleneck* pada server tertentu, menyebabkan *overload*, peningkatan waktu respons, bahkan kegagalan layanan [1]. Untuk mengatasi permasalahan tersebut, mekanisme *load balancing* menjadi salah satu solusi fundamental dalam menjaga kinerja dan kestabilan sistem [2].

Load balancing merupakan proses distribusi beban kerja secara merata di antara beberapa server guna mengoptimalkan pemanfaatan sumber daya, meningkatkan *throughput*, dan menghindari penumpukan beban pada satu titik [3]. Dengan penerapan strategi *load balancing* yang tepat, sistem dapat mempertahankan kinerja optimal meskipun menghadapi variasi permintaan yang tinggi dan kondisi trafik yang tidak stabil. Selain itu, mekanisme ini juga berperan penting dalam menjaga kontinuitas layanan serta meningkatkan efisiensi pemanfaatan sumber daya komputasi secara keseluruhan [4].

Dalam praktiknya, *load balancing* diterapkan pada dua jenis lingkungan server, yaitu server homogen dan heterogen. Server homogen memiliki spesifikasi perangkat keras dan lunak yang seragam sehingga pembagian beban relatif seimbang. Sebaliknya, server heterogen memiliki variasi

komponen seperti CPU, memori, dan konfigurasi sistem yang menimbulkan perbedaan kemampuan pemrosesan tiap *node* [5]. Kondisi ini menuntut algoritma *load balancing* yang adaptif terhadap kapasitas dan kondisi aktual server agar sistem tetap stabil dan efisien.

Berbagai algoritma *load balancing* telah dikembangkan untuk meningkatkan efisiensi dan kinerja sistem terdistribusi [6]. Secara umum, algoritma tersebut dapat dikategorikan menjadi dua jenis, yaitu statis dan dinamis, berdasarkan cara pengambilan keputusan dalam proses distribusi beban. Algoritma statis membagi beban tanpa mempertimbangkan kondisi server secara *real-time*, sedangkan algoritma dinamis menyesuaikan pembagian beban berdasarkan informasi aktual dari setiap server [7]. Keduanya memiliki pendekatan dan karakteristik berbeda tergantung pada tingkat keseragaman serta kemampuan sumber daya yang dimiliki oleh sistem [8].

Algoritma statis seperti *Round Robin*, bekerja dengan mendistribusikan permintaan secara bergiliran [9]. IP Hash menggunakan fungsi *hash* dari alamat IP klien untuk memastikan *session persistence* [10]. Serta *Random Allocation* membagi beban kerja ke server secara acak tanpa pola tertentu [11]. Pendekatan statis ini efektif diterapkan pada sistem homogen, di mana kemampuan pemrosesan antarserver relatif seragam dan distribusi beban tidak memerlukan penyesuaian kompleks.

Sementara itu, algoritma dinamis seperti *Least Connection* yang mengarahkan permintaan baru ke server dengan jumlah koneksi aktif paling sedikit, sedangkan *Least Response Time* memilih server yang memiliki waktu respons tercepat sekaligus beban koneksi terendah pada saat tersebut [12]. Adapun *Weighted Least Connection* dikembangkan dari algoritma *Least Connection* dengan prinsip pembagian beban kerja sesuai kapasitas pemrosesan masing-masing server yang telah ditetapkan sebelumnya [13]. Pendekatan dinamis ini umumnya lebih efektif diterapkan pada sistem heterogen, karena dapat beradaptasi terhadap perbedaan kapasitas sumber daya dan fluktuasi beban jaringan.

Berbagai penelitian telah dilakukan untuk mengevaluasi dan mengoptimalkan kinerja algoritma *load balancing* pada sistem terdistribusi. Penelitian [14] membandingkan algoritma R*ound Robin, Throttled,* dan *Equally Spread Current Execution* pada lingkungan *cloud computing*, menunjukkan bahwa pemilihan algoritma memengaruhi *response time* dan *service time*. Penelitian [15] mengimplementasikan algoritma optimisasi *data center* untuk menyeimbangkan beban antar mesin virtual. Pengujian mereka menegaskan bahwa efisiensi pembagian beban dapat ditingkatkan melalui pengelolaan sumber daya yang mempertimbangkan kapasitas pemrosesan dan tingkat antrian tugas pada tiap server.

Penelitian [16] memperkenalkan kebijakan *load balancing* berskala besar pada server heterogen yang memanfaatkan informasi kecepatan server untuk dua tahap pengambilan keputusan. Hasilnya menunjukkan peningkatan stabilitas dan penurunan waktu respons dibandingkan pendekatan sebelumnya.

Penelitian [17] mengusulkan model algoritma baru berbasis prioritas untuk meningkatkan efisiensi alokasi beban kerja. Hasil kajian mereka menunjukkan bahwa pendekatan dinamis dapat secara signifikan menurunkan waktu respons dan memperbaiki utilisasi sumber daya dibandingkan dengan metode statis. Selanjutnya, penelitian [18] dengan penulis yang sama, memperluas kajian tersebut dengan menyoroti penerapan *dynamic load balancing* dalam berbagai aplikasi dunia nyata, seperti sistem *e-commerce* dan layanan kesehatan digital, yang menuntut keandalan tinggi serta waktu respons rendah. Penelitian tersebut menegaskan bahwa efektivitas mekanisme *load balancing* bergantung pada kemampuan sistem untuk menyesuaikan diri dengan variasi beban dan spesifikasi perangkat keras secara dinamis.

Selain itu, penelitian [19] mengembangkan algoritma *hybrid load balancing* yang menggabungkan Round Robin, Least Connections, dan Weighted Response Time. Pendekatan berbasis *fitness selection* ini mampu menurunkan waktu respons hingga 48,5% dibanding Round Robin konvensional dan meningkatkan efisiensi melalui pemantauan kesehatan server.

Selanjutnya, penelitian [20] melakukan studi eksperimental untuk mengevaluasi efisiensi algoritma *Round Robin* dan *Least Connections* dalam lingkungan *cloud virtual*. Hasil pengujian menunjukkan bahwa algoritma *Round Robin* secara konsisten mengungguli *Least Connections* pada metrik utama seperti waktu respons rata-rata, *throughput*, dan pemanfaatan *bandwidth* jaringan. Temuan ini menegaskan bahwa meskipun *Least Connections* bersifat dinamis, efektivitasnya dapat menurun pada lingkungan dengan sumber daya yang seragam, sehingga pemilihan algoritma *load balancing* perlu disesuaikan dengan karakteristik sistem dan pola lalu lintas yang digunakan.

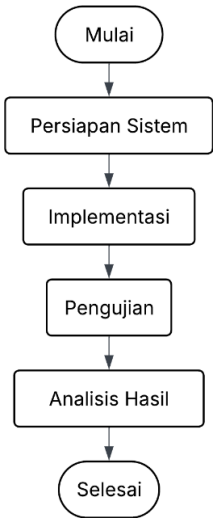
Meskipun berbagai studi tersebut telah memberikan kontribusi signifikan dalam memahami efektivitas algoritma *load balancing*, sebagian besar penelitian masih berfokus pada lingkungan *cloud* dan belum menjawab pertanyaan penting terkait karakteristik performa algoritma pada sistem *on-premise*, baik dalam kondisi server homogen maupun pada konfigurasi dengan variasi spesifikasi. Kondisi ini menjadi signifikan karena infrastruktur *on-premise* memiliki karakteristik operasional yang berbeda dari *platform cloud*, sehingga kinerja algoritma statis maupun dinamis berpotensi menunjukkan perilaku yang tidak sepenuhnya sejalan dengan temuan pada lingkungan virtualisasi berbasis *cloud*. Selain itu, ruang lingkup penelitian ini tidak mencakup mekanisme *failover*, karena fokus utama algoritma yang dikaji adalah distribusi beban (*load distribution*) antarserver. Oleh karena itu, kontribusi utama penelitian ini adalah (1) mengidentifikasi kesenjangan penelitian sebelumnya yang lebih berfokus pada lingkungan *cloud*, dengan menghadirkan evaluasi komparatif enam algoritma *load balancing* pada sistem *on-premise*, (2) menganalisis efektivitas algoritma statis seperti *Round Robin, IP Hash,* dan *Random Allocation*

pada lingkungan server homogen untuk menilai kestabilan performa dalam kondisi sumber daya seragam, (3) menganalisis adaptabilitas algoritma *Least Response Time, Least Connection,* dan *Weighted Least Connection* terhadap heterogenitas sumber daya untuk menilai pengaruh variasi spesifikasi server terhadap performa sistem, (4) menyajikan pengukuran kuantitatif berdasarkan empat parameter utama, seperti *throughput, latency, error rate,* dan *load distribution* pada tiga tingkat beban (*low, medium, high*) untuk menilai efisiensi dan stabilitas sistem, (5) menganalisis penggunaan CPU dan konsumsi memori sebagai parameter komparatif antara lingkungan homogen dan heterogen, serta (6) memberikan rekomendasi empiris terhadap algoritma *load balancing* yang paling sesuai untuk diterapkan pada kedua skenario lingkungan, baik homogen maupun heterogen, berdasarkan hasil pengujian eksperimental.

Penelitian ini dilakukan dalam lingkungan *on-premise* berskala terbatas yang terdiri dari enam *virtual machine*, sehingga hasil yang diperoleh lebih merepresentasikan kondisi eksperimental pada sistem kecil hingga menengah, bukan pada infrastruktur berskala besar.

II. METODE

Bagian ini menjelaskan metodologi yang digunakan untuk mengevaluasi performa enam algoritma *load balancing* pada dua jenis lingkungan server yang berbeda, yaitu server homogen dan server heterogen. Metode yang digunakan adalah metode eksperimen, karena memungkinkan dilakukan pengamatan langsung serta pengukuran kuantitatif terhadap performa sistem berdasarkan konfigurasi dan tingkat beban kerja tertentu. Alur penelitian ini terdiri atas beberapa tahapan, dimulai dari persiapan sistem, implementasi, pengujian, hingga analisis hasil, seperti yang ditunjukkan pada Gambar 1. Setiap tahap dirancang untuk memastikan akurasi dan konsistensi hasil eksperimen.



Gambar 1. Alur Penelitian

A. Persiapan Sistem

Lingkungan pengujian dibangun menggunakan *Proxmox Virtual Environment*, di mana setiap server diimplementasikan dalam bentuk *Virtual Machine* (VM). Semua VM dihubungkan dalam jaringan internal menggunakan Virtual LAN dan *gateway* yang sama, yang memungkinkan konektivitas antarserver secara efisien. Terdapat total delapan server, terdiri dari dua server *load balancer* dan enam server *backend*. Server *backend* dibagi menjadi dua kelompok:

- Tiga server homogen, dengan spesifikasi identik.
- Tiga server heterogen, dengan spesifikasi RAM berbeda.

Dengan demikian, perbedaan utama antara server homogen dan heterogen dalam penelitian ini terletak pada konfigurasi RAM masing-masing server. Tabel I menunjukkan spesifikasi perangkat keras dan perangkat lunak yang digunakan dalam penelitian ini.

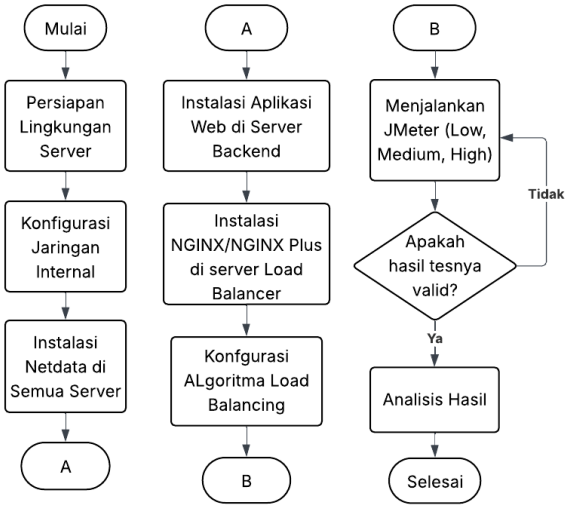
TABEL I
SPESIFIKASI HARDWARE AND SOFTWARE

Komponen	Deskripsi
Platform Virtualisasi	Proxmox VE 8.4.14
Jenis Server	Virtual Machine
Sistem Operasi	Ubuntu Server 22.04 (Jammy)
CPU	2 vCPU
Storage	32 GiB
RAM (Server Homogen)	2 GiB
RAM (Server Heterogen)	2 GiB /4 GiB / 6 GiB
Server Load Balancer 1	Nginx v1.18.0
Server Load Balancer 2	Nginx Plus v1.29.0 (nginx-plus-r35)
Alat Monitoring	Netdata v2.7.3
Alat Pengujian Beban	Apache JMeter v5.6.3

B. Implementasi

Tahap implementasi merupakan inti dari proses penelitian ini, yang mencakup seluruh rangkaian kegiatan mulai dari persiapan lingkungan server hingga tahap pengumpulan dan analisis data hasil pengujian. Alur tahapan implementasi sistem dapat dilihat pada Gambar 2, yang menggambarkan tahapan proses eksperimen untuk menguji kinerja enam algoritma *load balancing* pada dua jenis lingkungan server.

Penelitian ini diawali dengan pembuatan beberapa server pada *Proxmox Virtual Environment* (VE), yang masing-masing dikonfigurasi sesuai kebutuhan sistem, dengan total delapan server. Seluruh server dihubungkan dalam satu jaringan internal menggunakan Virtual LAN dan *gateway* yang sama. Pengaturan ini memastikan komunikasi antarserver berjalan stabil dalam jaringan terisolasi, serta memudahkan pemantauan performa selama pengujian berlangsung. Setiap server dijalankan dengan sistem operasi Ubuntu Server 22.04, yang digunakan karena stabilitasnya dan dukungan terhadap perangkat lunak server berbasis Linux.



Gambar 2. Flowchart Implementasi

Untuk memantau kinerja dan sumber daya selama proses pengujian, alat Netdata diinstal pada seluruh *node* baik pada server *load balancer* maupun server *backend* sebelum pengujian. Netdata digunakan untuk mengamati parameter seperti penggunaan *load distribution* antarserver secara *real-time*, penggunaan CPU, serta konsumsi memori.

Tahap selanjutnya adalah penyebaran aplikasi web sederhana pada masing-masing server *backend*. Aplikasi ini berfungsi untuk menampung permintaan HTTP yang dikirimkan oleh *load balancer* selama proses pengujian, sehingga setiap server dapat merespons trafik secara independen.

Dua server *load balancer* dikonfigurasi menggunakan dua versi perangkat lunak yang berbeda, yaitu NGINX dan NGINX Plus. NGINX digunakan untuk menerapkan tiga algoritma *load balancing* statis: *Round Robin, IP Hash,* dan *Random Allocation.* Sementara itu, NGINX Plus digunakan untuk tiga algoritma dinamis: *Least Response Time*, *Least Connection,* dan *Weighted Least Connection*.

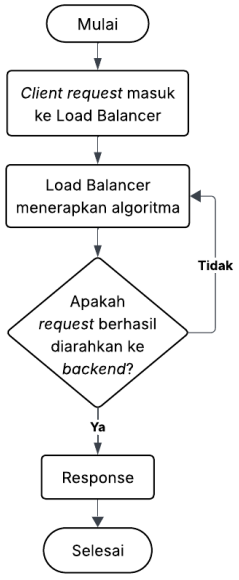
Pemilihan *platform load balancer* dalam penelitian ini merupakan keputusan metodologis yang dirancang untuk menjaga fokus analisis pada perilaku algoritma, bukan pada perbandingan antarplatform. NGINX digunakan untuk menguji algoritma statis, seperti *Round Robin, IP Hash,* dan *Random Allocation* karena *platform* ini menyediakan implementasi dasar yang lebih minimalis tanpa mekanisme pemantauan tambahan yang dapat memengaruhi sifat asli algoritma. Sebaliknya, algoritma dinamis seperti *Least Response Time, Least Connection,* dan *Weighted Least Connection* memerlukan dukungan fitur pemantauan koneksi dan respons secara *real-time*, sehingga NGINX Plus dipilih karena menyediakan kapabilitas tersebut sesuai dengan karakter operasional algoritmanya. Dengan pendekatan tersebut, pemilihan NGINX dan NGINX Plus berfungsi sebagai penyesuaian metodologis agar setiap algoritma diuji pada lingkungan yang paling sesuai dengan karakteristik operasionalnya. Fokus utama penelitian tetap diarahkan pada perilaku algoritma, sehingga *platform load balancer* digunakan sebagai infrastruktur pendukung yang menyediakan kapabilitas teknis yang diperlukan tanpa menjadi variabel utama dalam analisis.

Setiap algoritma diaktifkan secara bergantian dengan menyesuaikan konfigurasi yang menentukan pola distribusi permintaan ke server *backend*. Setelah konfigurasi diterapkan, dilakukan uji konektivitas untuk memastikan seluruh server *backend* dapat menerima permintaan dari *load balancer* dengan benar.

Untuk memastikan reproduktifitas eksperimental, setiap server di lingkungan Proxmox VE dikonfigurasi dengan parameter jaringan dan sumber daya yang tetap. Enam server *backend* dan dua *load balancer* dikerahkan dalam VLAN internal yang terisolasi. Dua server *load balancing* dikonfigurasi menggunakan NGINX v1.18.0 dan NGINX Plus v1.29.0 (nginx-plus-r35). Setiap *backend* dapat dijangkau melalui alamat IP privat.

Dengan seluruh lingkungan server yang telah terhubung, tahap berikutnya adalah menerapkan konfigurasi *load balancer* untuk masing-masing algoritma yang diuji. Setiap algoritma diaktifkan secara bergantian dengan menyesuaikan berkas konfigurasi pada NGINX maupun NGINX Plus agar pola distribusi permintaan mengikuti mekanisme kerja algoritma tersebut.

Setelah konfigurasi setiap algoritma tervalidasi, proses pengujian dilakukan menggunakan Apache JMeter. JMeter menghasilkan lalu lintas HTTP request menuju *load balancer* berdasarkan tiga skenario beban *low, medium,* dan *high*. Parameter seperti jumlah *thread, ramp-up time, dan loop count* ditetapkan sesuai nilai yang telah dirancang untuk masing-masing skenario.



Gambar 3. Flowchart Alur Distribusi Request oleh Load Balancer

Gambar 3 menggambarkan alur umum proses distribusi *request* pada sistem *load balancing* yang digunakan dalam penelitian ini. Ketika *client* mengirimkan *request*, *load balancer* akan menerima trafik tersebut dan menerapkan algoritma *load balancing* yang telah dikonfigurasi. Berdasarkan mekanisme kerja algoritma masing-masing, *load balancer* kemudian menentukan *backend* yang paling sesuai untuk memproses *request* tersebut. Jika proses pemilihan berhasil, *request* diarahkan ke *backend* terpilih dan sistem menghasilkan *response* kembali kepada *client*.

Pada setiap eksekusi skenario, hasil pengujian diperiksa untuk memastikan bahwa data yang dikumpulkan lengkap, konsisten, dan bebas dari anomali. Apabila ditemukan ketidaksesuaian, misalnya *error rate* yang tidak normal atau data yang tidak terbaca, konfigurasi pada NGINX atau JMeter disesuaikan dan skenario dijalankan ulang hingga diperoleh hasil yang valid.

Tahap akhir melibatkan pengumpulan seluruh data hasil pengujian dari JMeter dan Netdata. Dataset yang diperoleh mencakup metrik performa seperti *throughput, latency, error rate,* dan *load distribution*, serta parameter penggunaan CPU dan konsumsi memori. Seluruh metrik tersebut dianalisis secara kuantitatif untuk membandingkan performa setiap algoritma serta menilai efisiensi pembagian beban dan penggunaan sumber daya pada server homogen dan heterogen.

C. Testing

Pada tahap pengujian, Apache JMeter dikonfigurasi untuk mengirimkan *HTTP GET request* menuju laman *front-end* statis yang disajikan oleh setiap server *backend*. Laman ini berupa berkas HTML sederhana tanpa proses dinamis, pemanggilan API, maupun interaksi basis data, sehingga setiap permintaan menghasilkan waktu respons yang konsisten. Setiap skenario beban dijalankan dengan *Thread Group* sesuai Tabel VIII.

TABEL II
SKENARIO PENGUJIAN LOAD BALANCER

Skenario	Number of Threads (User)	Ramp-Up (second)	Loop Count	Total Request
Low	20	20	5	100
Medium	100	20	10	1000
High	300	20	10	3000

Ketiga skenario tersebut dirancang untuk mengukur kemampuan sistem dalam berbagai tingkat beban. Pada skenario *low*, pengujian dilakukan untuk mengamati distribusi awal beban dan kestabilan sistem saat menerima trafik ringan. Skenario *medium* digunakan untuk menilai performa sistem dalam kondisi normal dengan beban menengah, di mana server diharapkan dapat menyeimbangkan respons dan penggunaan sumber daya. Sementara itu, skenario *high* bertujuan untuk menguji batas kemampuan dan ketahanan sistem terhadap beban berat, termasuk kemungkinan terjadinya peningkatan *latency* dan *error rate*.

Dalam penelitian ini tidak diterapkan parameter *seed random* karena seluruh skenario pengujian menggunakan konfigurasi deterministik pada Apache JMeter (jumlah *threads*, *loop count*, dan *ramp-up time* ditetapkan secara eksplisit tanpa fungsi acak). Dengan demikian, setiap eksekusi menghasilkan pola trafik yang konsisten dan dapat direplikasi secara identik.

Hasil pengujian diambil melalui *Summary Report* dan *Aggregate Report* dari Apache JMeter untuk memperoleh nilai *throughput, latency,* dan *error rate*. Parameter *load distribution* diamati secara *real-time* menggunakan Netdata pada masing-masing server *backend*, di mana *snapshot* pemantauan diambil pada setiap skenario untuk memastikan pola distribusi trafik sesuai dengan algoritma yang sedang diuji. Selain itu, penggunaan CPU dan konsumsi memori dievaluasi setelah seluruh rangkaian pengujian selesai dijalankan pada kedua lingkungan server, baik homogen maupun heterogen, untuk menilai penggunaan sumber daya secara keseluruhan.

Setiap algoritma diuji sebanyak lima kali untuk memastikan hasil pengujian stabil dan sistem berfungsi dengan benar. Stabilitas hasil diukur dari konsistensi nilai metrik performa serta tingkat *error rate* yang mencapai 0%, yang menunjukkan bahwa seluruh permintaan HTTP berhasil diproses oleh *load balancer* tanpa kegagalan koneksi. Pengulangan dilakukan hingga seluruh konfigurasi dinyatakan valid dan tidak terdapat gangguan jaringan, sehingga hasil yang diperoleh mencerminkan performa aktual sistem dalam kondisi operasional yang optimal.

Untuk menjaga konsistensi hasil dan menghindari *bias sistem*, seluruh pengujian dilakukan pada jaringan lokal (*on-premise*) dengan konektivitas langsung antar-server menggunakan *Virtual LAN* di Proxmox tanpa penerapan *network emulation*. Kondisi jaringan berada dalam lingkungan terisolasi dan tidak terpengaruh lalu lintas eksternal, sehingga hasil pengujian merepresentasikan performa aktual sistem pada jaringan lokal yang stabil.

D. Analysis of Result

Data hasil pengujian dari JMeter dan Netdata kemudian diolah untuk memperoleh nilai rata-rata setiap parameter. Analisis dilakukan dengan membandingkan hasil antar algoritma dan antar tipe server (homogen dan heterogen) untuk melihat efisiensi pembagian beban serta stabilitas sistem. Setiap parameter dievaluasi secara komparatif, seperti *throughput* dan *latency* digunakan untuk menilai performa kecepatan sistem, *error rate* menggambarkan tingkat kestabilan sistem terhadap peningkatan beban kerja serta *load distribution* menunjukkan sejauh mana algoritma dapat membagi beban secara merata antarserver. Selain itu, penggunaan CPU dan konsumsi memori dievaluasi untuk menilai efisiensi penggunaan sumber daya pada masing-masing lingkungan server setelah seluruh skenario pengujian selesai dijalankan.

III. HASIL DAN PEMBAHASAN

Bagian ini menyajikan hasil pengujian dan analisis performa dari enam algoritma *load balancing* yang diimplementasikan pada dua jenis lingkungan server, yaitu homogen dan heterogen.

Pengujian dilakukan untuk menilai efektivitas setiap algoritma dalam mendistribusikan beban permintaan (*request distribution*), menjaga stabilitas kinerja sistem, serta memastikan efisiensi sumber daya komputasi dalam berbagai tingkat beban (*low, medium,* dan *high load scenarios*).

Tujuan utama dari analisis ini adalah untuk mengevaluasi sejauh mana mekanisme pembagian beban yang diterapkan oleh masing-masing algoritma berpengaruh terhadap parameter performa sistem.

Parameter-parameter tersebut dipilih karena merepresentasikan aspek utama dari kualitas layanan (*Quality of Service – QoS*) dalam konteks sistem *load balancing*, di mana *throughput* menggambarkan kecepatan pemrosesan permintaan, *latency* menunjukkan waktu respons rata-rata, *error rate* mengindikasikan tingkat kegagalan permintaan, *load distribution* mencerminkan pemerataan beban di antara server. Selain itu, penggunaan CPU dan konsumsi memori untuk menilai efisiensi penggunaan sumber daya pada masing-masing lingkungan server setelah seluruh skenario selesai dijalankan.

Secara keseluruhan, bagian ini dibagi menjadi dua tahap pembahasan utama, yaitu pertama, analisis hasil pengujian pada lingkungan server homogen, yang berfokus pada algoritma statis seperti *Round Robin, IP Hash,* dan *Random Allocation*, dan kedua, analisis pada server heterogen, yang mencakup algoritma dinamis yaitu *Least Response Time, Least Connection,* dan *Weighted Least Connection*.

Selanjutnya, pembahasan diperluas dengan analisis khusus terhadap parameter *throughput* dan *latency* berdasarkan hasil visualisasi grafik untuk melihat kecenderungan performa antar algoritma secara lebih mendalam. Sementara itu, parameter penggunaan CPU dan konsumsi memori disajikan dalam bentuk tabel untuk memberikan gambaran yang lebih terstruktur mengenai efisiensi penggunaan sumber daya komputasi setelah seluruh skenario pengujian dijalankan.

Setiap skenario pengujian pada kedua lingkungan, dijalankan sebanyak lima kali pengulangan untuk memastikan konsistensi dan reliabilitas hasil. Pada setiap pengulangan, diamati empat parameter utama, yaitu *throughput, latency, error rate,* dan *load distribution*. Nilai yang disajikan pada Tabel III dan Tabel IV merupakan rata-rata dari lima hasil pengukuran tersebut, sehingga data yang ditampilkan mencerminkan performa algoritma *load balancing* secara lebih stabil dan mengurangi potensi bias akibat fluktuasi sistem selama proses pengujian. Hasil pengulangan juga menunjukkan bahwa tidak terdapat perubahan nilai yang signifikan antar percobaan, sehingga variasi yang muncul pada data benar-benar merepresentasikan karakteristik dan variabilitas kapasitas server, bukan disebabkan oleh anomali sistem. Pendekatan ini memastikan bahwa interpretasi terhadap efektivitas algoritma pada kedua konfigurasi server dilakukan berdasarkan metrik yang terstandarisasi dan terverifikasi melalui pengukuran berulang.

TABEL III
HASIL UJI PADA SERVER HOMOGEN

Algoritma	Skenario	Throughput (kb/sec)	Avg Latency (ms)	Error Rate (%)	Load Distribution (%)		
					Server I	Server II	Server III
Round Robin	Low	6.84	34.20	0.00	11.86	10.37	9.95
	Medium	65.24	30.20	0.00	28.43	27.50	28.33
	High	194.30	31.40	0.00	21.91	21.81	21.86
IP Hash	Low	6.86	31.20	0.00	19.70	15.84	3.56
	Medium	65.49	30.00	0.00	27.94	14.72	3.86
	High	194.76	27.60	0.00	30.84	15.81	3.48
Random Allocation	Low	6.82	38.00	0.00	12.07	16.95	3.35
	Medium	65.00	32.40	0.00	11.96	14.55	3.11
	High	190.72	48.00	0.00	8.16	15.12	3.74

Pada pengujian homogen (server memiliki spesifikasi sama, masing-masing RAM 2 GiB), hasil pengujian menunjukkan bahwa seluruh algoritma mampu mempertahankan *throughput* yang meningkat secara proporsional terhadap kenaikan beban, dengan nilai berkisar 6.82–6.86 kb/sec pada beban rendah, meningkat menjadi 65.00–65.49 kb/sec pada beban sedang, hingga mencapai 190.72–194.76 kb/sec pada beban tinggi. Hal ini menunjukkan bahwa sistem mampu menskalakan performa sesuai peningkatan jumlah permintaan.

Dari sisi *latency*, seluruh algoritma menunjukkan waktu tanggapan dalam rentang 29–40 ms, meskipun terdapat variasi yang lebih mencolok pada algoritma *Random Allocation*. Pada algoritma ini, *latency* tercatat sebesar 38 ms pada beban rendah, menurun menjadi 32.40 ms pada beban sedang, namun kembali meningkat hingga 48 ms pada beban tinggi. Fluktuasi ini merupakan karakteristik wajar dari mekanisme pemilihan acak, karena beban antar server dapat terbagi tidak merata pada setiap skenario. Ketidakseimbangan tersebut semakin terasa pada beban tinggi, di mana satu server yang menerima lebih banyak permintaan dapat menyebabkan peningkatan *latency* secara signifikan. Variasi ini menunjukkan bahwa *Random Allocation* memiliki kestabilan lebih rendah dibandingkan algoritma yang mempertimbangkan kondisi server.

Untuk *error rate*, seluruh algoritma mencatat nilai 0%, menandakan tidak ada kegagalan permintaan selama proses pengujian, baik pada beban rendah hingga tinggi. Nilai *error rate* sebesar 0% ini menunjukkan tingkat reliabilitas sistem yang tinggi, di mana seluruh permintaan HTTP berhasil diteruskan dan diproses oleh *load balancer* tanpa adanya kegagalan koneksi maupun kesalahan konfigurasi. Kondisi ini juga mengindikasikan bahwa lingkungan pengujian berjalan stabil tanpa gangguan jaringan, sehingga hasil pengukuran merepresentasikan performa aktual sistem yang optimal.

Sementara pada *load distribution*, algoritma *Round Robin* tetap menunjukkan pembagian beban paling merata dengan variasi persentase antar server yang relatif kecil. Sebaliknya, *IP Hash* dan *Random Allocation* memperlihatkan variasi distribusi yang lebih besar hingga ±10%, dipengaruhi oleh sifat *hashing* dan pemilihan acak. Meskipun *throughput* tetap stabil, variasi ini berpotensi menimbulkan ketidakseimbangan beban pada skenario dengan jumlah *client* yang lebih besar.

TABEL IV
HASIL UJI PADA SERVER HETEROGEN

Algoritma	Skenario	Throughput (kb/sec)	Avg Latency (ms)	Error Rate (%)	Load Distribution (%)		
					Server I	Server II	Server III
Least Response Time	Low	6.82	39.00	0.00	11.82	10.36	3.06
	Medium	64.50	33.60	0.00	17.87	11.07	5.77
	High	193.32	32.40	0.00	32.56	23.73	20.68
Least Connection	Low	6.78	40.40	0.00	7.41	6.75	2.49
	Medium	64.95	29.00	0.00	14.59	13.37	4.39
	High	193.15	26.60	0.00	10.20	8.23	2.07
Weighted Least Connection	Low	6.79	38.20	0.00	3.44	3.63	3.89
	Medium	65.17	31.00	0.00	9.92	13.82	16.08
	High	192.06	26.00	0.00	31.08	35.24	39.13

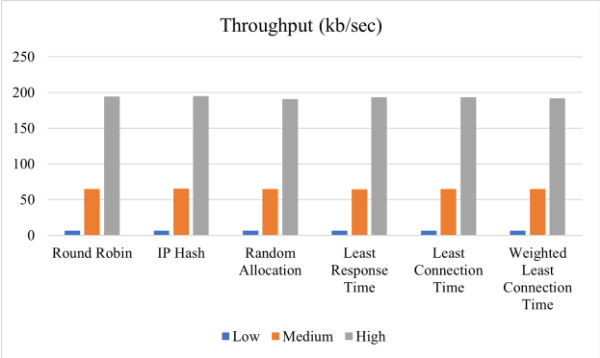
Pada pengujian heterogen, dengan perbedaan kapasitas RAM antar server (Server I: 2 GiB, Server II: 4 GiB, Server III: 6 GiB), seluruh algoritma menunjukkan peningkatan throughput sesuai dengan kenaikan beban, dengan kisaran 6.78–6.82 kb/sec pada beban rendah, meningkat menjadi 64.50–65.17 kb/sec pada beban sedang, dan mencapai 192.06–193.32 kb/sec pada beban tinggi. Pola ini menunjukkan bahwa sistem mampu mengakomodasi perbedaan kapasitas server seiring dengan dipertahankannya skalabilitas performa.

Dari segi *latency*, algoritma *Weighted Least Connection* menghasilkan waktu tanggapan terendah pada beban tinggi, yaitu 26 ms, diikuti oleh *Least Connection* dengan 26.60 ms. Sementara itu, algoritma *Least Response Time* menunjukkan *latency* sedikit lebih tinggi pada beban rendah (39 ms), namun menurun menjadi 32.40 ms pada beban tinggi. Nilai-nilai ini menggambarkan kemampuan sistem beradaptasi terhadap perbedaan kapasitas pemrosesan antar server.

Untuk *error rate*, seluruh algoritma juga menunjukkan nilai 0% pada seluruh skenario beban. Konsistensi hasil ini mengonfirmasi bahwa sistem tetap reliabel meskipun dijalankan pada lingkungan dengan sumber daya tidak seimbang. Tidak adanya permintaan yang gagal diproses menandakan bahwa *load balancer* mampu menangani heterogenitas sumber daya dengan baik, tanpa menyebabkan anomali koneksi ataupun kehilangan paket selama proses distribusi permintaan.

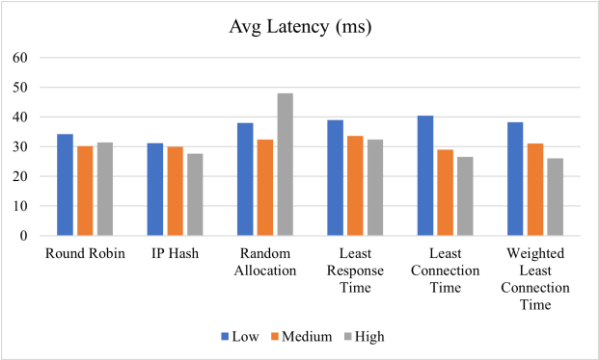
Pada *load distribution*, algoritma *Weighted Least Connection* menghasilkan pembagian beban paling proporsional terhadap kapasitas aktual tiap server. Sebaliknya, algoritma *Least Connection* dan *Least Response Time* masih cenderung memberikan beban yang tidak sebanding kepada server berkapasitas kecil. Pada algoritma *Least Connection*, misalnya pada skenario beban sedang, server kecil menerima 14.59% permintaan, sedangkan server besar hanya 4.39%, menunjukkan selisih sekitar 10% yang menandakan terjadinya pembebanan berlebih pada server dengan kapasitas paling rendah. Sementara itu, pada algoritma *Least Response Time*, server kecil menerima 17.87% permintaan dan server besar hanya 5.77%, dengan selisih lebih dari 12%. Kedua kondisi tersebut menggambarkan bahwa algoritma ini lebih sering memilih server kecil karena metrik waktu respons awal atau jumlah koneksi aktif yang rendah, tanpa mempertimbangkan kapasitas pemrosesan yang sebenarnya. Hal ini dapat menyebabkan *bottleneck* lokal pada server kecil, terutama pada skenario berskala besar.

Di sisi lain, algoritma *Weighted Least Connection* mampu mengatasi permasalahan ini karena menambahkan faktor pembobot (*weight*) sesuai spesifikasi server, sehingga beban yang diterima proporsional terhadap kapasitasnya. Hal ini karena *Weighted Least Connection* menormalkan jumlah koneksi terhadap bobot tiap server. Server dengan kapasitas lebih besar memiliki bobot lebih tinggi, sehingga meskipun menangani lebih banyak koneksi, rasio koneksi-per-*weight* tetap rendah. Dengan demikian, server kuat menerima beban lebih banyak secara terkontrol, sementara server kecil terhindar dari *overload*.



Gambar 3. Grafik Batang Throughput (kb/sec)

Gambar 3 memperlihatkan tren *throughput* (kb/sec) untuk masing-masing algoritma pada tiga tingkat beban (*low, medium,* dan *high*). Seluruh algoritma menunjukkan pola peningkatan *throughput* yang konsisten, yaitu sekitar 6–7 kb/sec pada beban rendah, 64–65 kb/sec pada beban sedang, dan 190–195 kb/sec pada beban tinggi. Keseragaman nilai ini mengindikasikan bahwa mekanisme pembagian beban pada seluruh algoritma berjalan efektif dan tidak menyebabkan bottleneck pada salah satu server, baik pada lingkungan homogen maupun heterogen. Dengan demikian, *throughput* sistem terutama dipengaruhi oleh tingkat beban yang diberikan, bukan oleh perbedaan algoritma yang digunakan.



Gambar 4. Grafik Batang Average Latency (ms)

Gambar 4 memperlihatkan variasi rata-rata *latency* untuk setiap algoritma pada tiga tingkat beban (*low, medium,* dan *high*). Pada lingkungan homogen, algoritma seperti *Round Robin* dan *IP Hash* menunjukkan pola *latency* yang relatif stabil di seluruh skenario, dengan perubahan nilai yang kecil antara beban rendah hingga tinggi. Namun, terdapat fluktuasi yang cukup mencolok pada *Random Allocation*, di mana *latency* menurun pada beban sedang tetapi meningkat signifikan pada beban tinggi. Hal ini sejalan dengan karakteristik pemilihan acak yang dapat menghasilkan ketidakseimbangan beban secara tak terduga.

Pada lingkungan heterogen, algoritma dinamis memperlihatkan pola yang lebih terstruktur. *Least Response Time* dan *Least Connection* menunjukkan *latency* yang cenderung menurun ketika beban meningkat, menggambarkan bahwa mekanisme adaptif mulai bekerja lebih efektif saat jumlah permintaan bertambah. Di sisi lain, *Weighted Least Connection* secara konsisten menghasilkan *latency* paling rendah pada beban tinggi, mencerminkan efektivitas algoritma ini dalam mendistribusikan permintaan berdasarkan kapasitas RAM masing-masing server. Secara keseluruhan, hal ini menunjukkan bahwa semakin baik algoritma memahami perbedaan kapasitas server, semakin rendah *latency* yang dihasilkan.

TABEL V
HASIL PERBANDINGAN LINGKUNGAN UJI

Lingkungan Uji	Penggunaan CPU (%)	Konsumsi Memori (MiB)
Homogen	15.71	404,74
Heterogen	12.49	439,92

Tabel V menunjukkan bahwa kedua lingkungan menghasilkan nilai penggunaan CPU dan memori yang berbeda setelah seluruh skenario pengujian dijalankan. Lingkungan homogen mencatat penggunaan CPU rata-rata sebesar 15.71%, sedangkan lingkungan heterogen sebesar 12.49%. Perbedaan ini menggambarkan variasi aktivitas sistem yang muncul dari mekanisme kerja algoritma pada masing-masing lingkungan selama pemrosesan *request*. Karena algoritma yang diuji berbeda (statis pada homogen dan dinamis pada heterogen), pola interaksi *backend* selama distribusi beban juga tidak identik, sehingga akumulasi aktivitas CPU yang terekam pada akhir pengujian menunjukkan nilai yang berbeda.

Pada parameter memori, lingkungan homogen mencatat 404.74 MiB, sedangkan lingkungan heterogen 439.92 MiB. Nilai ini menunjukkan bahwa masing-masing kelompok algoritma menghasilkan pola penggunaan memori yang tidak sama meskipun menjalankan rangkaian pengujian yang serupa. Algoritma statis pada lingkungan homogen bekerja dengan struktur keputusan tetap, sementara algoritma dinamis pada lingkungan heterogen menjalankan mekanisme pemilihan berdasarkan kondisi backend. Perbedaan cara kerja inilah yang tercermin pada variasi konsumsi memori agregat yang terukur.

Secara keseluruhan, perbedaan nilai CPU dan memori pada Tabel V tidak berasal dari perubahan perlakuan pengujian, melainkan merupakan hasil dari karakteristik operasional dua kelompok algoritma yang diuji dalam kondisi beban yang sama. Analisis ini sepenuhnya berdasarkan data yang tersaji dan mekanisme kerja algoritma yang digunakan, tanpa mengaitkan hasil dengan faktor-faktor yang tidak diukur secara eksplisit.

Temuan dari hasil pengujian ini selaras dengan beberapa penelitian terdahulu yang telah dibahas pada bagian sebelumnya. Penelitian [17] dan [18] menekankan bahwa efektivitas mekanisme *load balancing* bergantung pada kemampuan sistem untuk beradaptasi terhadap variasi beban dan karakteristik perangkat keras. Pendekatan dinamis yang mempertimbangkan kapasitas atau bobot server, seperti *Weighted Least Connection*, terbukti mampu mempertahankan stabilitas *throughput* serta menurunkan *latency*, sejalan dengan strategi berbasis prioritas yang

diusulkan. Sementara itu, temuan bahwa algoritma statis seperti *Round Robin* tetap efisien pada kondisi homogen juga sejalan dengan hasil penelitian [20], yang menunjukkan keunggulan algoritma tersebut dalam lingkungan dengan sumber daya seragam.

Dengan demikian, hasil pengujian ini memperkuat pandangan bahwa efektivitas *load balancing* sangat dipengaruhi oleh kemampuan algoritma dalam menyesuaikan strategi distribusi beban terhadap karakteristik dan kapasitas server yang digunakan. Pendekatan berbasis bobot dinilai lebih adaptif dan tahan terhadap variasi lingkungan, sehingga cocok diterapkan pada sistem berskala besar atau dengan spesifikasi server yang tidak seragam.

IV. KESIMPULAN

Penelitian ini menyimpulkan bahwa efektivitas algoritma load balancing bergantung pada keseragaman spesifikasi server dan karakteristik beban sistem. Pada lingkungan homogen, algoritma statis seperti *Round Robin, IP Hash,* dan *Random Allocation* terbukti cukup efisien karena pembagian beban sederhana sudah mampu menjaga kestabilan *throughput* dan *latency*. Oleh karena itu, pendekatan statis direkomendasikan untuk sistem berskala kecil hingga menengah dengan spesifikasi seragam, di mana prioritasnya adalah kesederhanaan konfigurasi dan stabilitas beban. Namun, algoritma statis sebaiknya tidak digunakan pada sistem dengan sumber daya yang bervariasi karena berpotensi menimbulkan ketimpangan beban antarserver.

Sebaliknya, pada lingkungan heterogen, algoritma dinamis seperti *Least Response Time, Least Connection,* dan terutama *Weighted Least Connection* menunjukkan performa yang lebih unggul karena mampu menyesuaikan distribusi beban berdasarkan kapasitas aktual setiap server. Pendekatan dinamis lebih tepat diterapkan pada sistem berskala menengah hingga besar, atau pada infrastruktur yang memiliki perbedaan kapasitas *hardware* antar *node*. Meskipun demikian, algoritma dinamis membutuhkan pemantauan sistem yang lebih intensif dan sumber daya tambahan pada *load balancer*.

Secara keseluruhan, penelitian ini menegaskan bahwa algoritma statis cocok untuk server homogen dengan beban ringan hingga menengah, sementara pendekatan dinamis berbasis bobot lebih sesuai untuk lingkungan heterogen karena mampu memanfaatkan variasi kapasitas server secara lebih efektif. Rekomendasi implementatif yang diberikan pada penelitian ini ditujukan untuk konteks on-premise berskala kecil sesuai ruang lingkup eksperimen.

Penelitian ini masih terbatas pada lingkungan pengujian dengan jumlah server dan konfigurasi yang kecil. Penelitian lanjutan disarankan untuk memperluas ruang lingkup menuju skala menengah dan besar, termasuk mengevaluasi *overhead monitoring* pada produksi, skenario *failover*, redundansi *load balancer*, serta pemanfaatan teknik optimasi adaptif atau kecerdasan buatan untuk mengatur *weight* secara otomatis berdasarkan kondisi waktu nyata.

DAFTAR PUSTAKA

[1] J. Idowu Akerele, A. Uzoka, P. Ugochukwu Ojukwu, and O. Jeremiah Olamijuwon, 'Optimizing Traffic Management for Public Services During High-Demand Periods Using Cloud Load Balancers', *Comput. Sci. IT Res. J.*, vol. 5, no. 11, pp. 2594–2608, 2024, doi: 10.51594/csitrj.v5i11.1710.

[2] A. G A and R. Pawar, 'Optimizing Cloud Application Performance: A Survey on Load Balancing Techniques', *Interantional J. Sci. Res. Eng. Manag.*, vol. 8, no. 5, pp. 1–5, 2024, doi: 10.55041/ijsrem34983.

[3] K. V. Kumar, G. Reddylatha, M. Sindhu, and K. Jayasree, 'A Comprehensive Survey of Load Balancing Techniques: From Classic Methods to Modern Algorithms', *Int. Res. J. Adv. Eng. Hub*, vol. 2, no. 02, pp. 287–296, 2024, doi: 10.47392/irjaeh.2024.0044.

[4] S. Shivaliya and V. Anand, 'Design of Load Balancing Technique for Cloud Computing Environment', *ECS Trans.*, vol. 107, no. 1, pp. 2911–2918, 2022, doi: 10.1149/10701.2911ecst.

[5] R. Kaur, S. Verma, Kavita, N. Z. Jhanjhi, and M. N. Talib, 'A Comprehensive Survey on Load and Resources Management Techniques in the Homogeneous and Heterogeneous Cloud Environment', *J. Phys. Conf. Ser.*, vol. 1979, pp. 1–27, 2021, doi: 10.1088/1742-6596/1979/1/012036.

[6] N. F. Hasani, 'Load Balancing in Distributed System', *Int. J. Basic Appl. Sci.*, vol. 14, no. 4, pp. 144–148, 2025, doi: 10.14419/kaejtm69.

[7] M. O. Oyediran, O. S. Ojo, S. A. Ajagbe, O. Aiyeniko, P. C. Obuzor, and M. O. Adigun, 'Comprehensive Review of Load Balancing in Cloud Computing System', *Int. J. Electr. Comput. Eng.*, vol. 14, no. 3, pp. 3244–3255, 2024, doi: 10.11591/ijece.v14i3.pp3244-3255.

[8] P. Ravi Kumar, S. Rajagopalan, and J. Charles P., 'Light Weight Native Edge Load Balancers for Edge Load Balancing', *Green Intell. Syst. Appl.*, vol. 3, no. 1, pp. 48–55, 2023, doi: 10.53623/gisa.v3i1.256.

[9] V. K. Prasad, D. Singh, and V. Gupta, 'Optimized Load Balancing Using Adaptive Algorithm in Cloud Computing with Round Robin Technique', *Int. J. Res. Appl. Sci. Eng. Technol.*, vol. 10, no. 7, pp. 134–149, 2022, doi: 10.53555/kuey.v30i2.6847.

[10] F. Apriliansyah, I. Fitri, and A. Iskandar, 'Implementasi Load Balancing Pada Web Server Menggunakan Nginx', *J. Teknol. dan Manaj. Inform.*, vol. 6, no. 1, pp. 18–26, 2020, doi: 10.3997/2214-4609.201801770.

[11] A. Jyoti, A. Yadav, D. Kumar, P. Mamoria, and V. Yadav, 'Classification & Technological Analysis of Load Balancing in Cloud Computing', *Recent Patents Eng.*, vol. 20, pp. 22–35, 2024, doi: 10.2174/0118722121358630241220044608.

[12] R. Gupta and O. P. Sharma, 'A Review Exploration of Load Balancing Techniques in Cloud Computing', *Educ. Adm. Theory Pract.*, vol. 30, no. 2, pp. 580–590, 2024, doi: 10.53555/kuey.v30i2.1600.

[13] S. A. Rahman and T. Y. Hadiwandra, 'Perbandingan Algoritma Weighted Least Connection dan Weighted Round Robin pada Load Balancing Berbasis Docker Swarm', *J. INOVTEK Polbeng - Seri Inform.*, vol. 8, no. 2, pp. 228–242, 2023, doi: 10.35314/isi.v8i2.3395.

[14] G. V. Gujar, S. R. Devane, and R. R. Deshmukh, 'Performance Comparison of Different Load Balancing Algorithms in Cloud Computing', *Int. J. Recent Innov. Trends Comput. Commun.*, vol. 11, no. 8s, pp. 747–754, 2023, doi: 10.17762/ijritcc.v11i8s.9286.

[15] N. Ranjan, B. Balkhande, S. Deokar, T. Kamble, C. Chaudhari, and S. T. Shirkande, 'Optimizing Cloud Computing Applications with a Data Center Load Balancing Algorithm', *Int. J. Recent Innov. Trends Comput. Commun.*, vol. 11, no. 10, pp. 320–331, 2023, doi: 10.17762/ijritcc.v11i10.8495.

[16] K. Gardner, J. Abdul Jaleel, A. Wickeham, and S. Doroudi, 'Scalable Load Balancing in The Presence of Heterogeneous Servers', *Elseiver*, vol. 145, pp. 1–19, 2021, doi: 10.1016/j.peva.2020.102151.

[17] J. Laha, S. Pattnaik, and K. Surjeet Chaudhury, 'Dynamic Load

Balancing in Cloud Computing: A Review and a Novel Approach', *EAI Endorsed Trans. Internet Things*, vol. 10, pp. 1–7, 2024, doi: 10.4108/eetiot.5387.

[18] J. Laha and S. Pattnaik, 'Dynamic Load Balancing in Cloud Computing: Improving Efficiency and Performance in Real Life Applications', *Int. Res. J. Adv. Eng. Hub*, vol. 2, no. 8, pp. 2192–2196, 2024, doi: 10.47392/irjaeh.2024.0298.

[19] M. Shahakar, S. A. Mahajan, L. Patil, and R. Mahajan, 'Design and Implementation of A Hybrid Load Balancing Algorithm for Optimized Server Response and Resource Utilization', *Int. J. Appl. Math.*, vol. 38, no. 1s, pp. 232–350, 2025.

[20] B. Alankar, G. Sharma, H. Kaur, R. Valverde, and V. Chang, 'Experimental setup for investigating the efficient load balancing algorithms on virtual cloud', *Sensors (Switzerland)*, vol. 20, no. 24, pp. 1–26, 2020, doi: 10.3390/s20247342.