ANALISIS PERFORMANSI LOAD BALANCING WEIGHTED LEAST CONNECTION DENGAN HAPROXY

Muhammad Algazali[1], La Surimi[2], La Ode Muhammad Bahtiar Aksara[3], Jumadil Nangi[4], Rizal Adi Saputra[5], Asa Hari Wibowo[6]
[1,2,3,4,5,6]Teknik Informatika, Teknik, Universitas Halu Oleo
[1]Muhammadalgazali999@gmail.com, [2]lasurimi@uho.ac.id, [3]bahtiar.aksara@uho.ac.id,
[4]jumadilnangi@uho.ac.id, [5]rizaladisaputra@uho.ac.id, [6]asa.hari@uho.ac.id

Abstrak

Aplikasi *web* telah menjadi bagian esensial dalam kehidupan digital masyarakat modern. Berdasarkan survei APJII 2023–2024, sebanyak 79,5% penduduk Indonesia atau sekitar 221,6 juta jiwa merupakan pengguna internet, sehingga menuntut layanan *web* yang cepat dan andal. Permasalahan yang muncul adalah beban server yang tidak merata, terutama saat terjadi lonjakan trafik, yang berdampak pada penurunan performa layanan. Penelitian ini bertujuan untuk menganalisis kinerja server *web* dengan menerapkan algoritma *Weighted Least Connection* (WLC) pada HAProxy, menggunakan skema bobot server 1:2, 1:3, 2:3, serta skema *single* server. Pengujian dilakukan pada dua *instance virtual* dengan spesifikasi berbeda di *VirtualBox*, menggunakan Apache JMeter untuk simulasi beban. Parameter yang diukur meliputi *throughput*, waktu respons, penggunaan CPU, tingkat kesalahan, dan ketersediaan sistem. Hasil pengujian menunjukkan bahwa skema WLC 1:2 menghasilkan performa terbaik dengan waktu respons 32,602 ms, *throughput* 122.080 *request*/s, dan tingkat kesalahan 30,51%, lebih efisien dibandingkan skema WLC 1:3. Tanpa *load balancer*, performa server menurun drastis dengan tingkat kesalahan mencapai 40%. Dengan demikian, skema WLC 1:2 terbukti lebih andal dan efisien dalam mendistribusikan beban, terutama pada kondisi trafik tinggi.

Kata kunci: Analisis Performansi, HAProxy, *Load Balancing*, *Web* Server, *Weighted Least Connection*,

1. Pendahuluan

Perkembangan teknologi informasi telah mendorong perubahan signifikan dalam cara manusia mengakses informasi, terutama melalui aplikasi berbasis *web* yang terhubung ke internet. Dengan jumlah pengguna internet di Indonesia mencapai 221.563.479 jiwa atau 79,5% dari total populasi, kebutuhan akan layanan *web* yang cepat dan andal terus meningkat. Angka ini naik sebesar 14,7% sejak tahun 2018, yang saat itu baru mencapai 64,8% (Prasetiyo et al. 2024) . Aplikasi *web* kini menjadi bagian penting dalam kehidupan sehari-hari, menjadikan keberadaan *web* server sebagai infrastruktur utama yang mendukung ketersediaan informasi secara online (Riska and Alamsyah 2021).

Peningkatan jumlah pengguna internet berbanding lurus dengan meningkatnya jumlah permintaan akses layanan *web*, yang berpotensi menyebabkan kelebihan beban (*overload*) pada server. Untuk mengatasi permasalahan ini, dilakukan pendekatan *horizontal scaling*, yaitu dengan menambahkan server tambahan sebagai replika untuk menampung lonjakan trafik (Ali and Abdullah 2019). Namun, distribusi trafik ke banyak server ini memerlukan pengaturan yang optimal agar beban kerja tidak timpang, sehingga dibutuhkan algoritma *load balancing* yang efektif untuk memastikan performa tetap stabil (Ilham 2024).

Terdapat berbagai algoritma dalam teknik load balancing, seperti *Round Robin*, *Weighted Round Robin*, *Least Connection*, dan *Weighted Least Connection* (WLC). Namun, tantangan utama dalam optimalisasi algoritma tersebut adalah ketidakseimbangan beban kerja, terutama saat kapasitas server berbeda-beda. Salah satu algoritma yang dianggap lebih efisien adalah *Weighted Least Connection* (WLC), karena memperhitungkan jumlah koneksi aktif dan bobot kapasitas server (Komaruddin et al. 2019). WLC memungkinkan distribusi beban yang lebih seimbang berdasarkan kemampuan masing-masing server dalam menangani koneksi.

Dalam implementasinya, *HAProxy* menjadi salah satu perangkat lunak *load balancer* yang mendukung algoritma WLC serta menyediakan fitur *high availability* dan *fault tolerance*. Kombinasi *HAProxy* dan WLC mampu meningkatkan performa sistem, mempercepat waktu respons, dan menjaga ketersediaan layanan secara optimal (Riskiono and Pasha 2020).

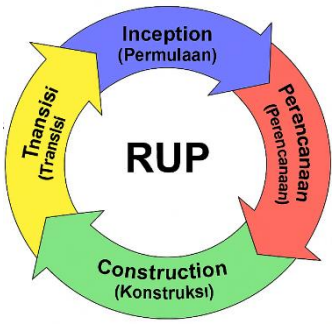
Dalam penelitian terdahulu yang dilakukan oleh Muhammad Naufal Ammar Rizqi dan I Kadek Dwi Nuryana (2022) menunjukkan bahwa WLC memiliki kinerja lebih baik dibandingkan *Weighted Round Robin* (WRR) dalam hal *throughput*, waktu respons, dan penanganan *error request* (Rizqi and Dwi Nuryana 2022). Hal ini memperkuat efektivitas WLC dalam mendistribusikan lalu lintas jaringan dan memastikan pembagian beban secara optimal. Selain itu, penelitian oleh Molion Surya Pradan dan Aditya Prapanca (2019) menguji algoritma WLC

menggunakan berbagai skema pembobotan, seperti rasio 2:1, 3:1, dan 2:3. Hasil penelitian menunjukkan bahwa skema pembobotan dapat mempengaruhi performa sistem, di mana konfigurasi rasio 2:3 mampu meningkatkan *throughput* sebesar 113 kb/s dan menurunkan jumlah *error request* hingga 986 (Pradana and Prapanca 2020).

Berdasarkan hasil-hasil tersebut, penelitian ini fokus pada analisis performa algoritma WLC dengan skema pembobotan 1:2, 1:3, dan 2:3 untuk menentukan konfigurasi optimal dalam mendistribusikan beban kerja secara seimbang.

2. Metode

Penelitian ini menggunakan metode *Rational Unified Process* (RUP) yang terdiri dari empat tahap: Permulaan, Perencanaan, Konstruksi, dan Transisi (Hakimin et al. 2021). Gambar 1 menunjukkan diagram alur metode RUP yang digunakan dalam penelitian ini, yang menggambarkan proses iteratif dari perencanaan hingga transisi sistem.



Gambar 1. Diagram Metode *Rational Unified Process* (RUP)

Gambar 1 menunjukkan alur metode *Rational Unified Process* (RUP) yang terdiri dari empat tahap. Pada Tahap *Inception*, dilakukan observasi dan studi pustaka untuk memahami infrastruktur *web* server serta teori terkait *load balancing* dan HAProxy, yang menjadi dasar ruang lingkup dan tujuan penelitian berbasis algoritma *Weighted Least Connection* (WLC). Tahap Perencanaan mencakup analisis kebutuhan, perancangan sistem, konfigurasi HAProxy, penyusunan *flowchart*, dan penetapan metrik seperti *response time*, *throughput*, dan kestabilan saat *failover*. Tahap Konstruksi melibatkan implementasi dan pengujian awal sistem pada skenario satu *node*, *multi-node*, dan tanpa *load balancing* menggunakan Apache JMeter. Pada Tahap Transisi, dilakukan *Load Testing* dan *Failover Testing* untuk mengukur efektivitas algoritma WLC dan ketahanan sistem.

2.1. Kebutuhan Sistem

Untuk menunjang proses implementasi dan pengujian sistem *load balancing*, digunakan lingkungan simulasi berbasis perangkat lunak VirtualBox. VirtualBox digunakan sebagai sarana untuk membangun mesin-mesin virtual, baik untuk server *backend* maupun *load balancer*, sehingga memungkinkan pengujian dilakukan secara lokal tanpa memerlukan infrastruktur fisik tambahan. Dalam simulasi ini, setiap mesin virtual menjalankan sistem operasi dan aplikasi pendukung layanan web. Selain itu, digunakan juga perangkat lunak tambahan seperti alat uji performa untuk mengevaluasi efektivitas mekanisme *load balancing* yang diterapkan. Dengan pendekatan ini, kondisi nyata sistem dapat direpresentasikan secara efisien dan terukur dalam lingkungan virtual, sekaligus memudahkan transisi atau integrasi ke infrastruktur *cloud* apabila dibutuhkan.

Perangkat PC yang digunakan sebagai *load balancer* berperan sebagai tempat instalasi HAProxy, yang bertugas mendistribusikan beban ke server *backend* secara merata. Spesifikasi lengkap perangkat tersebut disajikan pada Tabel 1 berikut:

Tabel 1. Spesifikasi Perangkat PC Load Balancer

Nama Perangkat	Spesifikasi
Notebook/PC	PC-Desktop HP
Processor	Intel Core i5 2.50 GHz
Core CPU	4 CORE (4 *Logical Core*)
OS	Ubuntu Server
Memori	RAM 6 GB DDR3
Harddisk	1 TB HDD

Selanjutnya, laptop yang digunakan untuk menjalankan beberapa mesin virtual *backend* melalui VirtualBox berfungsi sebagai server utama dalam pengujian sistem. Rincian spesifikasinya dapat dilihat pada Tabel 2 berikut:

Tabel 2. Spesifikasi Perangkat VirtualBox Server

Nama Perangkat	Spesifikasi
Notebook/PC	ACER Aspire Lite 14 (AL14-3IP)
Processor	Intel (R) N100 800 MHz
Core CPU	4 CORE (8 Logical Core)
OS	*Windows* 11 *Home Language*
Memori	RAM 16 GB DDR4
Harddisk	512 GB SSD

Setiap mesin virtual *backend* dikonfigurasi dengan sumber daya yang berbeda, disesuaikan dengan kebutuhan pengujian dan skenario beban yang diterapkan. Konfigurasi masing-masing mesin virtual ditampilkan pada Tabel 3 berikut:

Tabel 3. Spesifikasi Mesin Virtual Back-End

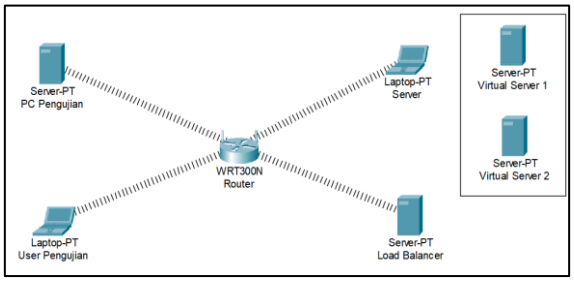
Nama	Spesifikasi BE 1	Spesifikasi BE 2
CPU	2 vCPU	3 vCPU
Memory	2 GB	3 GB
Storage	20 GB	20 GB

2.2. Arsitektur Sistem

Arsitektur sistem digunakan untuk mendefinisikan komponen – komponen desain topologi jaringan berdasarkan arsitektur *hardware* dan *software* yang lebih spesifik secara terstruktur. penelitian ini diterapkan sistem menggunakan algoritma *Weighted Least Connection*. Hal ini berguna untuk membantu beban kerja server yang digunakan dalam pengaksesan *web* dengan kapasitas

yang cukup tinggi dan *traffic* yang padat sehingga *web* dapat diakses dengan lancar.

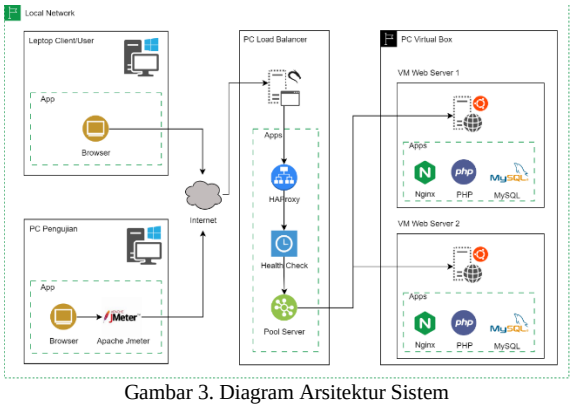
Gambaran desain topologi penelitian atau perancangan sistem dengan algoritma *Load Balancing Weighted Least Connection* secara arsitektur *hardware* dapat dilihat pada Gambar 2 berikut ini.



Gambar 2. Desain Topologi Pengujian Sistem

Gambar 2 menunjukkan bahwa setiap perangkat dalam jaringan dikoneksikan secara *wireless* melalui *router* yang digunakan sebagai pusat komunikasi dalam jaringan lokal (*Local Area Network* atau *LAN*). Secara tujuan, perangkat yang ideal digunakan adalah *access point*, namun karena keterbatasan alat, penelitian ini memanfaatkan *router* yang memiliki fungsi *access point* untuk membentuk jaringan nirkabel lokal. Topologi yang digunakan adalah topologi *star*, karena semua perangkat terhubung ke satu titik pusat, yaitu *router*, yang menjadi jalur utama pertukaran data antar perangkat.

Selanjutnya, alur kerja server dalam memproses permintaan dari klien melalui HAProxy yang telah dikonfigurasi digambarkan dalam Gambar 3 berikut.



Gambar 3. Diagram Arsitektur Sistem

Gambar 3 Diagram Arsitektur Sistem menunjukkan alur kerja server dalam memproses request klien melalui HAProxy yang telah dikonfigurasi. PC pengujian menggunakan Apache JMeter untuk mensimulasikan beban dan terhubung ke VirtualBox yang menjalankan beberapa mesin virtual. Request dari JMeter dikirim ke HAProxy *Load Balancer*, lalu didistribusikan ke server web berdasarkan algoritma *Weighted Least Connection* (WLC). *Request* diteruskan ke salah satu dari dua server yang menjalankan aplikasi *dummy* sebagai objek uji, kemudian diproses dan respons dikembalikan ke HAProxy, lalu ke PC pengujian. Selama proses, HAProxy juga melakukan *health check* untuk menjaga kestabilan dan ketersediaan aplikasi.

2.3. Perancangan Data

Perancangan data pada sistem ini bertujuan menjelaskan proses pengumpulan, pengolahan, dan analisis data untuk menjawab rumusan masalah. Data dikumpulkan melalui simulasi dan eksperimen guna menentukan variabel-variabel yang berpengaruh terhadap performas server, seperti distribusi beban, *throughput*, *response time*, *error rate*, penggunaan CPU, penggunaan memori, dan *availability*. Pengujian dilakukan menggunakan Apache JMeter untuk *load testing* dan *Sysstat* untuk *monitoring* performa server. Variabel-variabel tersebut dijelaskan lebih lanjut dalam Tabel 4 berikut.

Tabel 4. Variabel Data Load Testing dan Failover

Variabel	Fungsi	Alat Uji
Load Distribution	Mengukur seberapa beban dibagi di antara server.	*Log HAProxy*
Throughput	Mengukur jumlah data yang diproses per detik.	Apache JMeter
Response Time	Mengukur waktu prosesan *request-respons*	Apache JMeter
Error Rate	Menghitung persentase *request* yang gagal.	Apache JMeter
CPU *Usage*	Mengukur persentase penggunaan CPU server.	*Sysstat*
Memory Usage	Mengukur jumlah memori yang digunakan server.	*Sysstat*
Availability	Mengukur persentase waktu layanan tersedia.	*Uptime Log*

Pada Tabel 4 terdapat beberapa parameter utama dalam pengujian performa *load testing* yaitu *Throughput*, *Response Time*, CPU *Utilization*, *Memory Utilization*, *Error Rate* (Qomariyah et al. 2023). Adapun *failover* diukur dengan parameter *availability* untuk mengetahui ketersediaan system ketika dijalankan.

a. *Throughput* adalah jumlah data yang dikirim dalam satu satuan waktu, biasanya diukur dalam kilobyte per detik (KB/s), dengan rumus

$$Throughput(\text{KB/s}) = \frac{\sum Data\ dikirim\ (KB)}{Durasi\ (s)} \quad (1)$$

b. *Response time* merujuk pada waktu yang dibutuhkan server untuk merespons request pengguna, dihitung dengan rumus

$$Response\ Time\ (\text{Ms}) = \text{Temp Respon (Ms)} - \text{Temp } Request\ (ms) \quad (2)$$

c. CPU *utilization* menunjukkan persentase penggunaan CPU selama pengujian, dengan rumus

$$\text{CPU } Utilization\ (\%) = \frac{\sum \text{Penggunaan CPU}}{Durasi\ Pengujian\ (s)} \quad (3)$$

d. *Memory utilization* mengukur seberapa banyak memori yang digunakan oleh aplikasi selama pengujian, dihitung dengan rumus

$$\text{RAM } Utilization\ (\%) = \frac{\sum \text{Penggunaan RAM}}{Durasi\ Pengujian\ (s)} \quad (4)$$

e. *Error rate* mengindikasikan frekuensi kesalahan yang terjadi selama operasional aplikasi, yang menjadi indikator penting dalam mengukur keandalan sistem di bawah beban.

f. Parameter *availability* sebagai tolok ukur untuk menilai ketahanan sistem, memastikan beban dapat dialihkan secara otomatis dari komponen yang gagal ke komponen cadangan tanpa mengganggu layanan, dihitung dengan rumus

$$Availability\ (\%) = \frac{\sum Waktu\ tersedia}{Durasi\ operasi\ (s)} \times 100\% \quad (5)$$

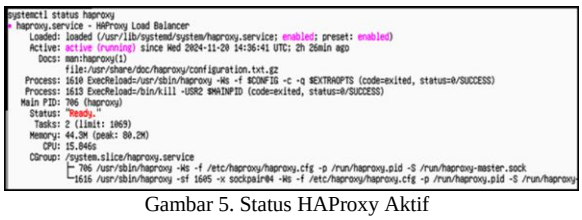
3. Hasil dan Pembahasan

Pada tahap ini dijelaskan penerapan sistem, hasil pengujian, serta analisis performa sistem. Tahapan ini mencakup proses instalasi dan konfigurasi *tools*, implementasi sistem, pengujian performa, serta pembahasan hasil yang diperoleh.

3.1. Instalasi Dan Konfigurasi

Implementasi sistem diawali dengan instalasi dan konfigurasi beberapa tools utama yang diperlukan untuk mendukung proses pengujian. NGINX digunakan sebagai *web* server untuk melayani permintaan pengguna, sedangkan HAProxy berperan sebagai *load balancer* dengan penerapan algoritma *Weighted Least Connection*. Untuk memonitor performa CPU dan memori server, digunakan *Sysstat*, sementara Apache JMeter digunakan sebagai alat untuk melakukan pengujian beban (*load testing*).

Gambar 4. Status NGINX Terinstal



Gambar 5. Status HAProxy Aktif

Gambar 4 memperlihatkan keberhasilan instalasi Nginx pada server *backend* (Node), sementara Gambar 5 menunjukkan keberhasilan instalasi HAProxy sebagai *load balancer*. Tahapan ini menandai bahwa penguji telah berhasil melakukan pemasangan dan konfigurasi perangkat pada server dan *load balancer*.

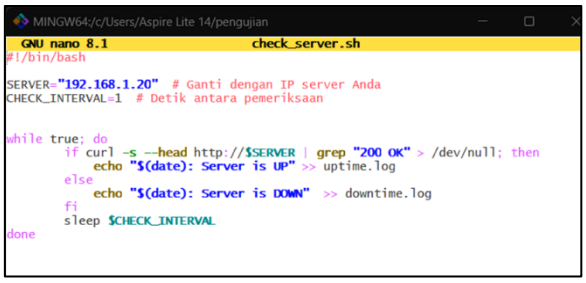
Selanjutnya, konfigurasi HAProxy dengan algoritma *Weighted Least Connection* menggunakan variasi bobot 1:2, 1:3, dan 2:3 dapat dilihat pada Gambar 6, Gambar 7, dan Gambar 8, yang menampilkan pengaturan bobot berbeda untuk mendistribusikan request antara server Node 1 dan Node 2 sesuai perbandingan tersebut.

```
option httpchk HEAD /
server node1 192.168.1.28:80 weight 1 check
server node2 192.168.1.29:80 weight 2 check
```
Gambar 6. Konfigurasi HAProxy WLC bobot 1:2

```
option httpchk HEAD /
server node1 192.168.1.28:80 weight 1 check
server node2 192.168.1.29:80 weight 3 check
```
Gambar 7. Konfigurasi HAProxy WLC bobot 1:3

```
option httpchk HEAD /
server node1 192.168.1.28:80 weight 2 check
server node2 192.168.1.29:80 weight 3 check
```
Gambar 8. Konfigurasi HAProxy WLC bobot 2:3

Untuk memastikan keberlanjutan layanan selama pengujian, dibuat skrip bash yang secara berkala memeriksa status *uptime* dan *downtime* server, skrip tersebut ditampilkan pada Gambar 9 berikut.



Gambar 9. *Monitoring Uptime* dan *Downtime*

Gambar 9 memperlihatkan konfigurasi *script bash* untuk menguji *uptime* dan *downtime* server dalam pengujian *fault tolerance* dengan skema *failover*. Pengujian ini bertujuan mengukur durasi proses *failover*, yaitu waktu yang dibutuhkan server cadangan untuk mengambil alih dan merespons *request* setelah kegagalan server utama.

3.2. Hasil Pengujian

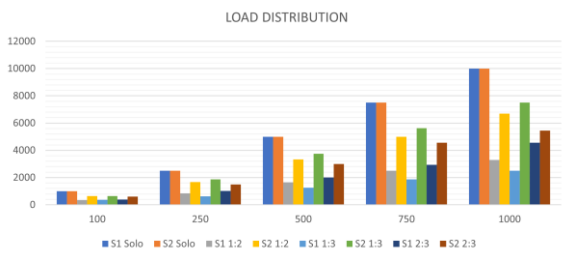
Pada skenario pengujian ini, sistem diuji menggunakan Apache JMeter untuk mensimulasikan beban dengan berbagai jumlah koneksi dan jenis *request*. Apache JMeter digunakan untuk mengukur parameter performansi seperti *throughput*, *response time*, dan *error rate* (Ismail et al. 2023). Htop digunakan untuk memantau penggunaan CPU dan memori secara *real-time*, sedangkan *Sysstat* digunakan untuk pengumpulan log aktivitas CPU dan memori selama pengujian. Log HAProxy dimanfaatkan untuk mencatat aktivitas distribusi *request* ke *node* server (Pentanugraha et al. 2024).

Pengujian dilakukan dengan variasi beban sebesar 100, 250, 500, 750, dan 1000 *request* selama

10 detik pada setiap skema algoritma *Weighted Least Connection* (WLC) (Karim, Primananda dan Yahya, 2019). Inisial yang digunakan dalam tabel hasil pengujian meliputi S1 (*Single* server lemah dengan 2 vCPU), S2 (server kuat dengan 3 vCPU), serta WLC (skema WLC 1:2, 1:3, dan 2:3) (Radtke and Ababei 2022). Load distribution diukur berdasarkan jumlah request yang tercatat dalam log HAProxy, *throughput*, *response time*, dan *error rate* diperoleh dari hasil pengujian menggunakan Apache JMeter, sedangkan CPU *usage* dan *memory usage* dipantau menggunakan *Sysstat* (Irwan, 2017). *Availability* dihitung berdasarkan data *uptime* server dari log eksekusi (Makpul 2021). Selanjutnya, hasil pengujian terhadap parameter *load distribution*, *throughput*, *response time*, *error rate*, *CPU utilization*, *memory utilization*, dan *availability* akan disajikan secara sistematis dalam bentuk tabel dan grafik pada bagian berikut.

Tabel 5. Tabel Pengujian *Load Distribution*

Req	Rata-Rata *Load Distribution* (Request/Test)							
	Single		WLC 1:2		WLC 1:3		WLC 2:3	
	S1	S2	S1	S2	S1	S2	S1	S2
100	1000	1000	349	651	362	638	394	606
250	2500	2500	835	1665	622	1878	1012	1488
500	5000	5000	1659	3341	1255	3745	2001	2999
750	7500	7500	2498	5002	1870	5630	2944	4556
1000	10000	10000	3302	6698	2500	7500	4556	5444

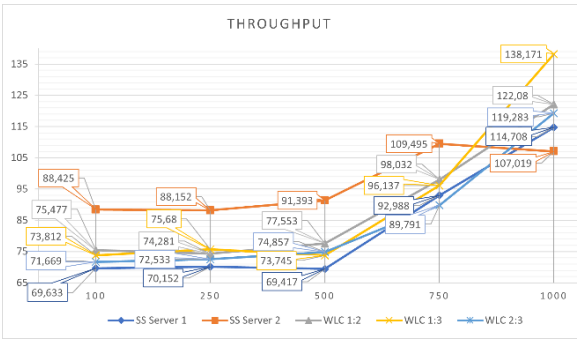


Gambar 10. *Bar Chart Load Distribution*

Tabel 5 menampilkan hasil pengukuran distribusi request antara dua server dalam berbagai skema pembebanan, seperti tanpa *load balancer* serta dengan algoritma *Weighted Least Connection* (WLC) menggunakan bobot 1:2, 1:3, dan 2:3. Gambar 10 menyajikan visualisasi hasil tersebut dalam bentuk grafik batang untuk memperjelas perbandingan. Tanpa penggunaan *load balancer*, seluruh request masuk ke satu server. Sebaliknya, dengan WLC, request didistribusikan sesuai proporsi bobot yang ditetapkan. Pada pengujian 10.000 *request*/10 detik, hasil distribusi mendekati rasio yang diharapkan, seperti 3.302:6.698 (1:2), 2.500:7.500 (1:3), dan 4.556:5.444 (2:3).

Tabel 6. Tabel Pengujian *Throughput Testing*

Req	***Throughput* (*request*/*s*)**				
	Tanpa LB		WLC	WLC	WLC
	S1	S2	1:2	1:3	2:3
100	69,6	88,4	75,5	73,8	71,7
250	70,2	88,2	74,3	75,7	72,5
500	69,4	91,4	77,6	73,7	74,9
750	93,0	109,5	98,0	96,1	89,8
1000	114,7	107,0	122,1	138,2	119,3

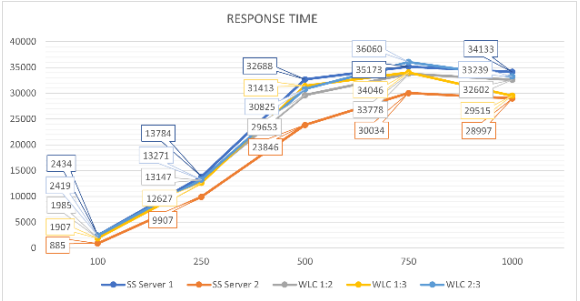


Gambar 11. *Line Chart Throughput*

Tabel 6 menampilkan perkembangan throughput pada tiap skema pengujian algoritma WLC, di mana *throughput* didefinisikan sebagai jumlah permintaan yang dapat diproses server per detik. Gambar 11 menyajikan visualisasi data tersebut dalam bentuk grafik garis. Berdasarkan Tabel 6, terlihat bahwa pada pengujian dengan 100, 250, 500, hingga 750 *request*, server mampu memproses permintaan dengan nilai yang relatif stabil. Namun, pada pengujian 1.000 *request*, performa server tanpa *load balancing* mulai menurun. Implementasi algoritma *Weighted Least Connection* menghasilkan performa yang lebih stabil dan unggul, dengan throughput lebih tinggi sekitar 20–30 *request* per detik dibandingkan tanpa *load balancer*. Skema terbaik ditunjukkan oleh WLC 1:3 dengan *throughput* tertinggi sebesar 138,171 *request* per detik pada beban maksimum.

Tabel 7. Tabel Pengujian *Response Time Testing*

Req	***Rata-Rata Response Time (ms/request)***				
	Tanpa LB		***WLC***	***WLC***	***WLC***
	S1	***S2***	***1:2***	***1:3***	***2:3***
100	2434	885	1985	1907	2419
250	13784	9907	13147	12627	13271
500	32688	23846	29653	31413	30825
750	35173	30034	33778	34046	36060
1000	34133	28997	32602	29515	33239



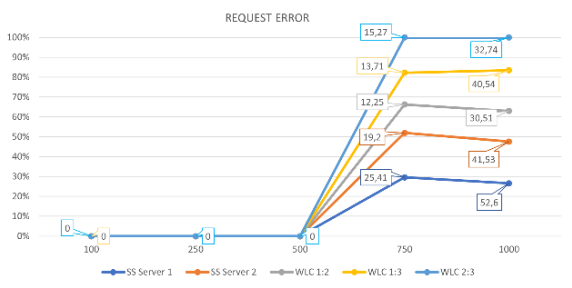
Gambar 12. *Line Chart Response Time*

Tabel 7 menyajikan waktu yang dibutuhkan server untuk merespons permintaan (*response time*) pada berbagai skema pengujian. Gambar 12 menampilkan grafik perbandingan dari nilai tersebut. Hasil pengujian menunjukkan bahwa pada skema pengujian dengan 100, 250, 500, 750, hingga 1.000 *request*, terdapat perbedaan yang signifikan antar skema. Pada skema *Weighted Least Connection (WLC)* 1:2, *response time* rata-rata tercatat sebagai

yang terbaik untuk beban rendah hingga menengah, yaitu *885 ms* pada *100 request* dan *28.997 ms* pada *1.000 request*. Skema *WLC 1:3* menunjukkan keunggulan pada beban tinggi dengan *response time 29.515 ms* pada *1.000 request*, meskipun sedikit lebih lambat dibandingkan *WLC 1:2* pada tingkat request rendah.

Tabel 8. Tabel Pengujian *Error Rate Testing*

Req	***Rata-Rata Request Error (%)***				
	Tanpa LB		***WLC 1:2***	***WLC 1:3***	***WLC 2:3***
	S1	***S2***			
100	0	0	0	0	0
250	0	0	0	0	0
500	0	0	0	0	0
750	25,41	19,2	12,25	13,71	15,27
1000	52,6	41,53	30,51	40,54	32,74



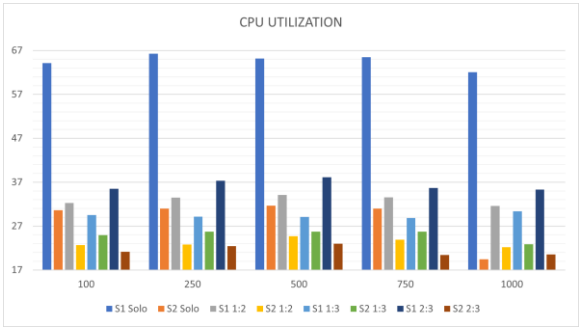
Gambar 13. *Line Chart Request Error*

Tabel 8 menyajikan tingkat error pada berbagai skema *load balancing* berdasarkan jumlah *request* per detik. Gambar 13 menampilkan visualisasi perbandingan tersebut dalam bentuk grafik. Hasilnya menunjukkan bahwa pada beban rendah (100 hingga 500 *request* per detik), tidak ada error yang terjadi. Namun, tanpa *load balancer*, tingkat error meningkat signifikan pada beban tinggi, yaitu 25,41% pada 750 *request* dan 52,6% pada 1.000 *request*. Implementasi algoritma *Weighted Least Connection* (WLC) 1:2 menghasilkan performa terbaik dengan tingkat error terendah, yakni 19,2% pada 750 *request* dan 41,53% pada 1.000 *request*. Skema WLC 1:3 juga menunjukkan error lebih rendah (13,71% dan 40,54%) dibandingkan WLC 2:3, yang mencatatkan error 15,27% pada 750 *request* dan 32,74% pada 1.000 *request*.

Tabel 9. Tabel Pengujian Utilitas CPU

Req	Rata-Rata CPU *Utilization* (%)							
	Single		**WLC 1:2**		**WLC 1:3**		**WLC 2:3**	
	S1	**S2**	**S1**	**S2**	**S1**	**S2**	**S1**	**S2**
100	64,16	30,62	32,28	22,69	29,54	24,96	35,47	21,13
250	66,29	30,96	33,5	22,81	29,13	25,72	37,34	22,42
500	65,19	31,68	34,12	24,67	29,06	25,74	38,14	22,97
750	65,49	30,98	33,52	23,88	28,87	25,74	35,68	20,43
1000	62,06	19,45	31,62	22,21	30,38	22,86	35,3	20,52

Tabel 9 menampilkan nilai CPU utilization, yaitu persentase penggunaan kapasitas CPU dalam memproses *request* pada berbagai skema *load balancing*. Gambar 14 menyajikan visualisasi data tersebut dalam bentuk grafik garis. Berdasarkan Tabel 9, terlihat bahwa pada pengujian dengan 100, 250, 500, 750, hingga 1.000 *request* per detik, penggunaan CPU server lemah mencapai sekitar 60%, sedangkan server kuat sekitar 30%.

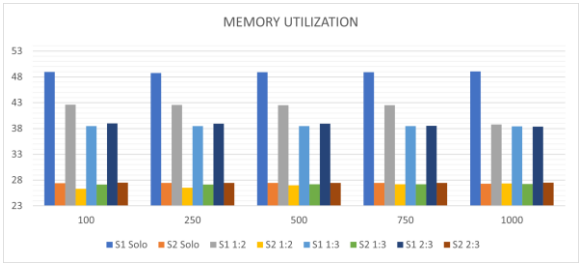


Gambar 14. *Bar Chart* CPU *Utilization*

Dengan penerapan algoritma *Least Connection*, performa server lemah menurun sementara server kuat meningkat pada skema dengan beban antara 1.000 hingga 10.000 *request* per detik, yang disebabkan oleh pembobotan distribusi *request*. Skema rasio 1:3 menunjukkan efektivitas terbaik dalam menurunkan beban server lemah, dengan perbedaan penggunaan CPU yang hampir seimbang sebesar dengan perbedaan 3 sampai 4%, dibandingkan dengan rasio 1:2 dan 2:3 yang memiliki perbedaan lebih besar, masing-masing 9-10% dan 15%.

Tabel 10. Tabel Pengujian Utilitas Memori

Req	Rata-Rata *Memori Utilization* (%)							
	Single		**WLC 1:2**		**WLC 1:3**		**WLC 2:3**	
	S1	**S2**	**S1**	**S2**	**S1**	**S2**	**S1**	**S2**
100	48,94	27,43	42,63	26,33	38,46	27,13	39	27,49
250	48,77	27,45	42,58	26,52	38,45	27,14	38,96	27,47
500	48,91	27,46	42,53	26,97	38,45	27,18	38,91	27,48
750	48,89	27,46	42,54	27,22	38,45	27,22	38,52	27,48
1000	49,08	27,31	38,78	27,33	38,39	27,25	38,37	27,49



Gambar 15. *Bar Chart Memory Utilization*

Tabel 10 menampilkan nilai pengujian *utilitas memory*, yaitu persentase penggunaan kapasitas memori dalam memproses *request* pada berbagai skema *load balancing*. Gambar 15 menyajikan visualisasi data tersebut dalam bentuk grafik garis. Hasil pengujian menunjukkan pola penggunaan memori yang mirip dengan penggunaan CPU. Tanpa *load balancing*, server lemah (S1) menggunakan memori tinggi (48,77%–49,08%), sedangkan server kuat (S2) menggunakan memori rendah (27,31%–27,46%), yang mengindikasikan ketidakseimbangan beban. Pada skema WLC 1:2, distribusi penggunaan

memori lebih merata, dengan S1 menggunakan 38,78%–42,63% dan S2 26,33%–27,33%. Skema WLC 1:3 menunjukkan efisiensi pada S1 (38,39%–38,46%), namun distribusi beban masih kurang optimal, sementara S2 tetap stabil pada 27,13%–27,25%. Pada WLC 2:3, penggunaan memori oleh S1 sedikit lebih tinggi (38,37%–39%) dibandingkan WLC 1:3, namun distribusi memori tetap tidak seimbang, dengan S2 stabil di 27,47%–27,49%.

Pengujian availability dilakukan dengan 4 skenario pengujian selama masing-masing 20 detik, di mana pada setiap pengujian, mulai detik ke-10 diterapkan skema mematikan server 1, server 2, atau kedua server secara bergantian. Pengujian ini mengukur ketersediaan layanan web, distribusi trafik, *response time*, jumlah *request error*, *request* yang berhasil, serta *request* yang diulang karena *timeout*. Ketersediaan sistem dihitung berdasarkan *up-time* dan *down-time* selama pengujian yang kemudian menjadi nilai *availability* berdasarkan waktu uji 20 detik. Hasilnya dapat dilihat pada Tabel 11 berikut.

Tabel 11. Tabel Pengujian *Availability*

Skema Server Selama Pengujian	Distribusi Server 1	Distribusi Server 2	*Request Retry*	Total *Request*	*Request Error*	*Response Time*	*Up-Time*	*Down-Time*	*Availability*
Normal (Semua aktif)	594 r	606 r	0 r	1200 r	0 r	26,3 ms	21 s	0 s	100%
Server 1 Dimatikan (t=10s)	462 r	783 r	45 r	1245 r	0 r	29,1 ms	15 s	4 s	71,4%
Server 2 Dimatikan (t=10s)	817 r	428 r	68 r	1268 r	0 r	30,7 ms	13 s	6 s	61,9%
Semua Server Dimatikan (mulai t=10s)	569 r	540 r	200 r	1109 r	200 r	29,7 ms	10 s	11 s	47,6%

Pengujian menunjukkan bahwa sistem mencapai ketersediaan penuh (100%) saat kedua server aktif, dengan seluruh *request* berhasil diproses tanpa *error*. Ketika salah satu server dimatikan pada detik ke-10, HAProxy mampu menjaga kelangsungan layanan dengan mengalihkan trafik ke server yang tersisa, meskipun terjadi *retry* (45 dan 68 *request*) serta peningkatan waktu respons (29,1 ms dan 30,7 ms), disertai *downtime* sementara selama 4–6 detik sehingga *availability* menurun menjadi 71,4% dan 61,9%. Penurunan paling signifikan terjadi saat kedua server dimatikan bersamaan, menyebabkan 200 *request error*, *downtime* 11 detik, dan *availability* turun menjadi 47,6%. Meskipun demikian, HAProxy tetap memproses *request* hingga tidak ada lagi *backend* tersedia, membuktikan kemampuannya dalam menangani *failover* dan redistribusi trafik secara adaptif.

4. Kesimpulan

Berdasarkan hasil pengujian selama 10 detik untuk setiap skema, algoritma *Weighted Least Connection* (WLC) 1:2 terbukti lebih efisien dalam mendistribusikan beban serta memberikan performa yang stabil pada beban rendah hingga menengah (100–750 *request*), dengan waktu respons yang cepat dan tingkat error yang rendah. Sebaliknya, WLC 1:3 menunjukkan *throughput* lebih tinggi namun memiliki distribusi beban yang kurang merata dan menghasilkan *error rate* lebih tinggi saat beban meningkat. Tanpa *load balancer*, sistem mengalami penurunan performa yang signifikan, dengan *error rate* mencapai 52,6%. Implementasi *failover* meningkatkan *availability* sistem menjadi 71,4% dan 61,9% saat salah satu server *backend* dimatikan pada detik ke-10. Walaupun terjadi *retry request* (masing-masing 45 dan 68 *request*), semua permintaan tetap mendapatkan respons, namun proses *retry* menyebabkan waktu respons meningkat. Pada skema saat kedua server dimatikan bersamaan, *availability* turun ke 47,6% dengan 200 *request error*, menunjukkan bahwa HAProxy tetap aktif mendistribusikan beban hingga semua *backend* gagal. Ini memperlihatkan keunggulan HAProxy dalam mempertahankan layanan hingga batas maksimal. Untuk penelitian selanjutnya, disarankan melakukan pengujian dengan jumlah server *backend* yang lebih banyak untuk mengevaluasi kemampuan HAProxy dalam menangani skala besar serta menjaga keseimbangan distribusi beban. Selain itu, durasi pengujian yang lebih panjang perlu diterapkan agar respons sistem dapat dievaluasi pada kondisi jangka menengah hingga panjang. Eksplorasi terhadap algoritma atau skema dinamis lainnya juga dianjurkan untuk mengkaji kemampuan adaptasi sistem terhadap perubahan beban dan mempercepat waktu pemulihan dalam lingkungan produksi.

Daftar Pustaka:

Ali, A. H. (2019). *A Survey on Vertical and Horizontal Scaling Platforms for Big Data Analytics*. International Journal of Integrated Engineering, 11(6), 138-150. https://doi.org/10.30880/ijie.2019.11.06.015

Hakimin, K., Jaroji & Subandri, M.A. (2021): *Penerapan Metode Rational Unified Process (RUP) Pada Pembuatan Aplikasi Public Speaking*, Seminar Nasional Industri dan Teknologi (SNIT), pp. 250–259.

Ilham, M. (2024): *Implementasi Sistem Load Balancing untuk Optimasi Kinerja pada web* server *Nginx Menggunakan Algoritma IP Hash*, Jurnal Teknik Informatika, Multimedia, dan Jaringan (INFOMEDIA), 9, pp. 60–68.

Irwan, D. (2017): *Service Availability dan Performa Sumber Daya Prosessor pada Infrastruktur* server *Virtual*, Jurnal Penelitian Ilmu Komputer, System Embedded & Logic, 5(1), pp. 42–50.

Ismail, A., Ananta, A.Y., Arief, S.N. & Hamdana, E.N. (2023): *Performance Testing Sistem Ujian Online Menggunakan Jmeter Pada Lingkungan Virtual*, Jurnal Informatika Polinema, 9(2), pp. 159–164. https://doi.org/10.33795/jip.v9i2.1190

Komaruddin, A.M., Sipitorini, D.M. & Rispian, P. (2019): *Load Balancing dengan Metode Round Robin Untuk Pembagian Beban Kerja web* Server, Siliwangi, 5(2), pp. 47–50. https://doi.org/10.37058/jssainstek.v5i2.1184

Makpul, Z.B.M. (2021): *Analisa Stabilitas MES* server *Untuk Mendukung Kegiatan Produksi*, Computer Based Information System Journal, 9(1), pp. 37–41. https://doi.org/10.33884/cbis.v9i1.3640

Pentanugraha, E., Saragih, A.S. & Christian, E. (2024): *Analisis Kinerja Load Balancing Web*server *Menggunakan HAProxy Terintegrasi Dengan Grafana Sebagai Monitoring Dan Notifikasi Telegram*, JOINTECOMS (Journal of Information Technology and Computer Science), 4(1), pp. 68–80. https://doi.org/10.47111/jointecoms.v4i1.13191

Pradana, M.S. & Prapanca, A. (2020): *Analisis Performa Load Balancing Algoritma Weighted Round Robin di Infrastruktur BPBD Provinsi Jawa Timur*, Journal of Informatics and Computer Science (JINACS), 1(2), pp. 109–114. https://doi.org/10.26740/jinacs.v1n02.p109-114

Prasetiyo, S.M., Gustiawan, R., Faarhat & Albani, F.R. (2024): *Analisis Pertumbuhan Pengguna Internet Di Indonesia*, Jurnal Buletin Ilmiah Ilmu Komputer dan Multimedia, 2(1), pp. 65–71.

Qomariyah, N., Subyantoro, E. & Asrowardi, I. (2023): *Penelitian Pendahuluan tentang Pengukuran Performance dan Load Testing pada Learning Management System (LMS)*, ROUTERS: Jurnal Sistem dan Teknologi Informasi, 1(2), pp. 122–126. https://doi.org/10.25181/rt.v1i2.3134

Radtke, T. & Ababei, C. (2022): *Performance Evaluation of the Weighted Least Connection Scheduling for Datacenters with BigHouse Simulator*, IEEE International Conference on Electro Information Technology, 2022-May, pp. 1–4. https://doi.org/10.1109/eIT53891.2022.9813846

Riska, R. & Alamsyah, H. (2021): *Penerapan Sistem Keamanan web Menggunakan Metode web Application Firewall*, *Jurnal Amplifier*: Jurnal Ilmiah Bidang Teknik Elektro dan Komputer, 11(1), pp. 37–42. https://doi.org/10.33369/jamplifier.v11i1.16683

Riskiono, S.D. & Pasha, D. (2020): *Analisis Metode Load Balancing Dalam Meningkatkan Kinerja Website E-Learning*, Jurnal Teknoinfo, 14(1), pp. 22. https://doi.org/10.33365/jti.v14i1.466

Rizqi, M.N.A. & Dwi Nuryana, I.K. (2022): *Analisis Perbandingan Kinerja Algoritma Weighted Round Robin dan Weighted Least Connection Menggunakan Load Balancing Nginx Pada Virtual Private* server *(VPS)*, Journal of Informatics and Computer Science (JINACS), 4(1), pp. 67–75. https://doi.org/10.26740/jinacs.v4n01.p67-75