IMPLEMENTASI DAN PENGUJIAN PERFORMA SERVER HIGH AVAILABILITY MENGGUNAKAN METODE HORIZONTAL SCALING DENGAN STUDI KASUS: MARKETPLACE XYZ

Achmad Nur Fauzy[1], Muhammad Anis Al Hilmi[*2], Ilman Kadori[3]
Program Studi Rekayasa Perangkat Lunak, Politeknik Negeri Indramayu[1,2]
Program Studi Informatika, Universitas Swadaya Gunung Jati (UGJ) Cirebon[2,3]
achmadnurf22@gmail.com[1], muhammadanisalhilmi.dosen@ugj.ac.id*[2],
ilmankadori.dosen@ugj.ac.id[3]

Abstrak

Marketplace daring telah populer di kalangan masyarakat Indonesia belakangan ini. Hal ini berdampak pada trafik pengunjung yang terus meningkat, sehingga kinerja server dalam melayani permintaan menjadi semakin berat. Dampak lain yang timbul adalah penurunan kinerja web server disertai lag atau gangguan jaringan yang menyebabkan respon halaman web menjadi lambat. Masalah ini dapat ditangani menggunakan load balancing, yaitu metode yang membagi trafik ke beberapa server sehingga tidak terjadi penumpukan beban pada satu server. Pengujian 10.000 permintaan dengan Apache Bench menghasilkan waktu respons 7,725 ms untuk klaster server (3 server) menggunakan algoritma round-robin. Sebagai perbandingan, algoritma least connection menghasilkan waktu respon 7,496 ms, sementara single server menghasilkan 119,656 ms. Pada uji stres menggunakan JMeter dengan 10.000 permintaan, klaster server menghasilkan waktu respons 60 ms, sedangkan single server menghasilkan 39.418 ms. Pada implementasi failover, load balancing bekerja dengan baik dalam membagi permintaan ke server yang tersedia. Failover adalah proses pengalihan permintaan dari server yang mati/rusak ke server lain yang aktif.

Kata Kunci: *Load balancing, Marketplace Daring, Web Server, Failover, Algoritma round-robin dan least connection*

Abstract

Online marketplaces have become popular among the Indonesian public lately. This trend has an impact on the increasing visitor traffic, making the server's performance in handling requests heavier. Another impact that arises is the decline in web server performance accompanied by lag or network disruptions that cause web page responses to become slow. This issue can be addressed using load balancing, which is a method that distributes traffic across multiple servers to prevent load accumulation on a single server. Testing 10,000 requests with Apache Bench resulted in a response time of 7.725 ms for a server cluster (3 servers) using the round-robin algorithm. In comparison, the least connection algorithm produced a response time of 7.496 ms, while a single server resulted in 119.656 ms. In a stress test using JMeter with 10,000 requests, the server cluster produced a response time of 60 ms, while a single server resulted in 39.418 ms. In the failover implementation, load balancing works well in distributing requests to the available servers, ensuring that even during high traffic or server failures, the system maintains optimal performance and minimizes response times. Failover is the process of redirecting requests from a dead/broken server to another active server.

***Keywords**: Load balancing, Marketplace Daring, Web Server, Failover, Algoritma round-robin and least connection*

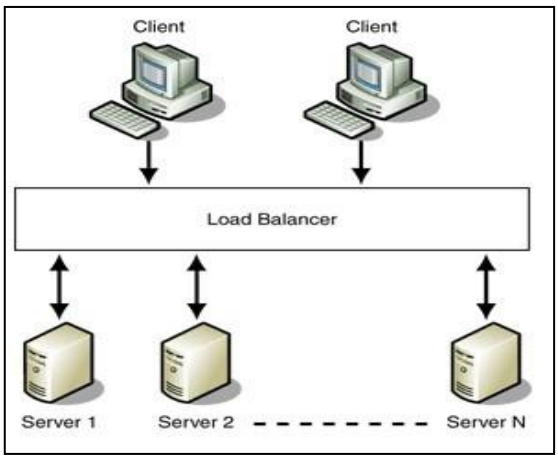
PENDAHULUAN

Saat ini, banyak situs *web* yang hampir selalu dikunjungi oleh publik sebagai pengguna internet dan akan terus berkembang. Permintaan layanan pun meningkat. Salah satu layanan dan aplikasi *web* yang paling populer adalah pasar *online/marketplace*. Pasar *online* memiliki fitur yang membutuhkan *server* yang selalu tersedia atau terus tersedia, yaitu fitur flash sale, yang merupakan aktivitas atau acara dari pasar online. Dengan fitur ini, pengguna akan mendapatkan diskon pada setiap pembelian. Dalam contoh aktivitas ini, situs *web* pasar online XYZ pasti telah mengalami banyak lonjakan layanan atau lalu lintas tinggi. Meningkatnya jumlah lalu lintas juga dapat menyebabkan kinerja *server* yang melayani permintaan menjadi lebih berat (Dani & Suryawan,

2017). Akibatnya, kinerja *server web* menurun, dan sering terjadi lag atau gangguan pada jaringan, menyebabkan respons yang lama dari halaman *web*. Jika tidak ditangani atau diperbaiki, hal ini dapat mengakibatkan *server* mati/tidak berfungsi.

Masalah lalu lintas tinggi dapat diatasi dengan menggunakan *load balancing*. *Load balancing* adalah metode Horizontal Scaling (Umam et al., 2018). *Load balancing* juga merupakan metode untuk membagi lalu lintas antara beberapa *server* sehingga tidak terjadi penumpukan beban pada satu *server* (Prakoso, 2020). Ketersediaan *server* akan dipertahankan atau tetap berjalan dengan menggunakan metode penyeimbangan beban ini; yaitu, jika salah satu *server* tidak melayani permintaan dari pengguna (*server* mati atau *down*), maka *server* lain akan secara otomatis mengambil alih permintaan dari pengguna sehingga pengguna tidak menyadari bahwa *server* sedang *down*. Beberapa algoritma penyeimbangan beban membagi beban *server*, yaitu round robin dan *least connection*. Algoritma round-robin sederhana dan sering digunakan dalam penyeimbangan beban (Riskiono & Pasha, 2020). Algoritma round-robin akan membagi beban secara bergantian dari satu *server* ke *server* berikutnya (Diarjo & Mulyana, 2017). Algoritma *least connection* adalah algoritma penjadwalan dinamis karena algoritma ini membutuhkan perhitungan dinamis koneksi langsung ke setiap *server*. Dalam penelitian ini, penulis menggunakan algoritma round-robin dan *least connection* sebagai perbandingan karena algoritma ini dapat membantu proses penyeimbangan beban menjadi lebih baik dan bahkan dapat membantu dalam mengelola lalu lintas atau layanan dari pengguna ke *server web* yang akan ditangani.

Ketersediaan Tinggi adalah pembagian sistem yang terdiri dari 2 *server* atau lebih mesin *server* yang dikenal sebagai *node* (Riawati et al., 2022). Prinsip kerja *High availability* adalah ketika *server* utama tidak dapat menyediakan layanan, *server* yang dibagi akan secara otomatis menggantikan tugas *server* utama (Erlianto et al., 2020). Konsep ini berkaitan dengan kemampuan suatu sistem untuk mengatasi gangguan, kerusakan perangkat keras, crash/*down*, kesalahan jaringan, dan bahkan kegagalan *server* yang disebabkan oleh perangkat lunak yang gagal menjalankan tugasnya dengan benar (Rahayu, 2021). *Load balancing* adalah metode dalam komputasi klaster yang dapat meningkatkan skalabilitas dan mengurangi beban kerja *server* dengan mendistribusikan beban lalu lintas ke beberapa klaster *server* secara seimbang sehingga lalu lintas dapat berjalan optimal, tidak membebani *server* (overload) atau bahkan *down* (Pratama et al., 2018).



Gambar 1. Skenario *Load balancing* (Rahmatulloh & Msn, 2017)

Algoritma round-robin adalah algoritma yang paling sederhana dan banyak digunakan oleh perangkat penyeimbang beban (Riskiono & Pasha, 2020). Cara kerja algoritma ini adalah dengan menjadwalkan secara berurutan sesuai dengan *server* yang tersedia. Jika klien 1 membuat permintaan, maka akan menuju ke *server* 1; setelah permintaan dari klien satu, maka akan menuju ke *server* 2, dan seterusnya hingga semua *server* memiliki bagian beban permintaan yang sama sesuai dengan jumlah *server* (Wicaksono, 2019). Algoritma koneksi paling sedikit adalah algoritma penjadwalan dinamis karena algoritma ini membutuhkan perhitungan dinamis koneksi langsung ke

setiap *server* (Hakim et al., 2019). Algoritma ini juga dapat melayani dengan membagi lebih banyak permintaan ke *server* pusat berdasarkan koneksi yang tersedia lebih sedikit; algoritma koneksi paling sedikit juga mengasumsikan bahwa semua *server* backend memiliki kemampuan komputasi yang sama (Triangga et al., 2019). HAProxy adalah perangkat lunak gratis, berkecepatan tinggi, dan andal dalam mengimplementasikan ketersediaan tinggi, penyeimbangan beban, dan proxy untuk aplikasi berbasis TCP dan HTTP (Azizah, 2017). Perangkat lunak Apache memiliki alat untuk membuat *server web* (Azizah, 2017). Keuntungan Apache adalah lebih mudah untuk diatur; Apache juga dapat digunakan sebagai penyeimbang beban dengan menambahkan beberapa modul untuk dikonfigurasi (Azi et al., 2023). Apache Bench adalah alat (*tool*) dari organisasi Apache yang digunakan untuk mengukur kinerja pada *server web* HTTP (Chandra, 2019). JMeter adalah perangkat lunak sumber terbuka berbasis Java yang dirancang untuk menguji beban perilaku fungsional dan mengukur kinerja pengujian beban dari *server web* (Khadafi et al., 2017).

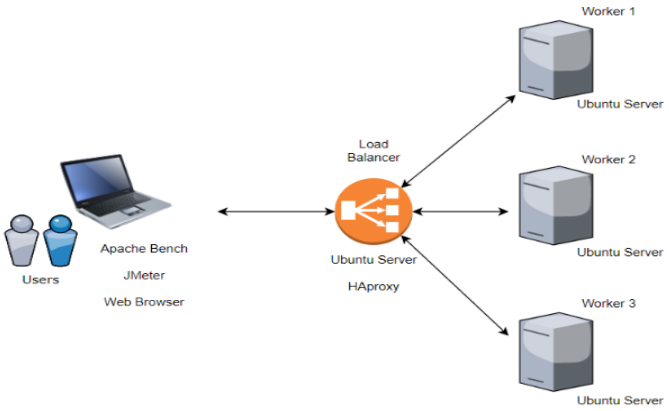
PENELITIAN RELEVAN

Dalam rangka mencapai high availability pada server, berbagai metode dan studi kasus telah dilakukan untuk meningkatkan performa dan reliabilitas sistem, khususnya dalam konteks marketplace dan web server. (Pribadi et al., 2020) menyoroti pentingnya failover clustering sebagai solusi untuk memastikan ketersediaan layanan web server yang tinggi. Penelitian tersebut menunjukkan bahwa sistem failover clustering mampu bekerja sesuai konsepnya dan memberikan tingkat availability sebesar 99,90%, bahkan saat terjadi kegagalan server. Hasil pengujian juga mengindikasikan bahwa performa web server dengan failover clustering tetap memuaskan, meskipun performa tanpa failover sedikit lebih baik pada beberapa parameter, namun secara keseluruhan konfigurasi ini menunjukkan keunggulan dalam hal availability saat terjadi kegagalan. Selain itu, pengembangan sistem yang mampu mengatasi beban secara horizontal juga menjadi fokus utama dalam studi lain. Implementasi arsitektur microservices dalam pengembangan marketplace sewa properti menunjukkan bahwa skalabilitas horizontal dapat meningkatkan agility dan kapasitas sistem dalam menghadapi pertumbuhan pengguna dan fitur (Pandega & Widyaro, 2026). Pengujian upload file dalam konteks ini menegaskan bahwa penambahan node secara horizontal mampu mendukung peningkatan throughput dan performa secara efektif. Dalam konteks sistem berbasis cloud dan virtualisasi, sistem auto scaling yang responsif dan high availability juga menjadi perhatian. Sistem ini, seperti yang dibahas dalam analisis performa pada sistem web berbasis clustering menggunakan AWS, menunjukkan bahwa auto scaling dapat meningkatkan performa dan ketersediaan layanan secara otomatis sesuai dengan kebutuhan beban kerja (Asiari, 2021). Pendekatan ini memungkinkan sistem untuk menyesuaikan kapasitas secara dinamis, sehingga mendukung kebutuhan marketplace yang mengalami fluktuasi trafik. Secara keseluruhan, berbagai studi menunjukkan bahwa implementasi dan pengujian performa server high availability dengan metode horizontal scaling dapat dilakukan secara efektif melalui kombinasi failover clustering, dan auto scaling. Pendekatan ini tidak hanya meningkatkan ketersediaan layanan tetapi juga mendukung pertumbuhan dan skalabilitas sistem marketplace secara berkelanjutan.

METODE PENELITIAN

Gambar 2 merupakan aristektur sistem yang digunakan dalam penelitian dan spesifikasinya dijelaskan dalam Tabel 1.

e-ISSN : 2715-8756



Gambar 2. Arsitektur Sistem

Tabel 1. Spesifikasi

No	Item / Deskripsi	Spesifikasi
	Perangkat Keras	
1	5 komputer	- Sebagai *Load balancer*: 4GB RAM, Intel® Core™ i5-4460 CPU @ 3.20GHz - Sebagai *Client*/Testing: 4GB RAM, Intel® Core™ i5-4460 CPU @ 3.20GHz - Sebagai *Worker* 1,2,3: 4GB RAM, Intel® Core™ i5-4460 CPU @ 3.20GHz
2	Satu switch jaringan	5 port LAN
3	5 kabel LAN	-
	Perangkat Lunak	
1	4 Ubuntu *Server* (*worker* dan *load balancer*) 1 Ubuntu Desktop (*client*)	V 18.04 LTS
2	HAProxy Apache2 *Web Server* Apache Bench JMeter Mozilla Firefox	*Load balancer* *Web server* Perangkat Lunak *Load Testing* / Simulasi Trafik Browser *web* untuk *client*
	Konfigurasi Jaringan	
1	*Load balancer*	IP 192.168.100.2
2	*Client* / Tester	IP 192.168.100.1
3	*Worker*/*webserver* 1	IP 192.168.100.5
4	*Worker*/*webserver* 2	IP 192.168.100.4
5	*Worker*/*webserver* 3	IP 192.168.100.3

Konfigurasi *file host* dilakukan pada *server web* 1, 2, dan 3 ditampilkan pada Gambar 3, Gambar 4, dan Gambar 5, dengan perintah sudo nano etc/*host*s pada terminal *server web*.

```
127.0.0.1       localhost
192.168.100.5   webserver1
192.168.100.2   HAproxy
```

Gambar 3. Konfigurasi *Web Server* 1

```
127.0.0.1       localhost
192.168.100.4   webserver2
192.168.100.2   HAproxy
```

Gambar 4. W Konfigurasi *Web Server* 2

```
127.0.0.1       localhost
192.168.100.3   webserver3
192.168.100.2   HAproxy
```

Gambar 5. Konfigurasi *Web Server* 3

Setelah instalasi HAProxy, konfigurasikan algoritma round-robin pada *server* HAProxy. Selanjutnya, lakukan pemantauan HAProxy. Untuk memberikan informasi, termasuk status *server*, data yang ditransfer, waktu aktif, kecepatan sesi, dan lain-lain. Untuk konfigurasi pemantauan HAProxy, ditambahkan baris pada Gambar 6 ke *file* HAProxy.cfg:

```
#HAproxy monitoring config
#--------------------------------------
listen stats
        bind 192.168.100.2:8080
        stats enable
        stats show-node
        stats hide-version
        stats uri /stats
        stats realm HAproxy\ Statistic
        stats auth osboxes:osboxes
```

Gambar 6. Konfigurasi Monitoring HAProxy

Langkah selanjutnya adalah mengkonfigurasi *file* HAProxy.cfg, untuk menggunakan algoritma round-robin dan kemudian algoritma *least-connection*. Algoritma ini menyesuaikan untuk mengatur alamat *bind, frontend, backend, dan port*.

```
# HAproxy for web server
frontend web-frontend
        bin 192.168.100.2:80
        mode http
        default_backend web-backend

backend web-backend
        mode http
        balance roundrobin
        server webserver 1 192.168.100.5 check port 80
        server webserver 2 192.168.100.4 check port 80
        server webserver 3 192.168.100.3 check port 80
```

Gambar 7. Konfigurasi Round Robin

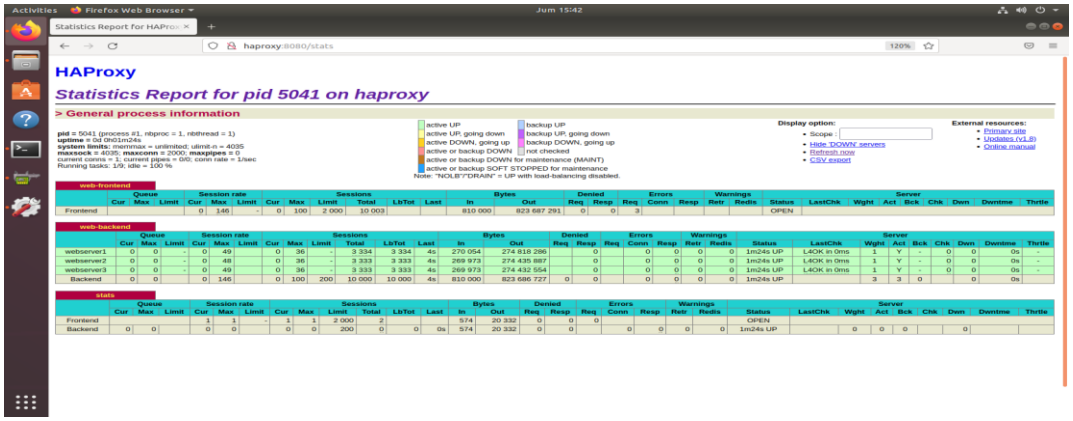
```
# HAproxy for web server
frontend web-frontend
        bin 192.168.100.2:80
        mode http
        default_backend web-backend

backend web-backend
        mode http
        balance leastconn
        server webserver 1 192.168.100.5 check port 80
        server webserver 2 192.168.100.4 check port 80
        server webserver 3 192.168.100.3 check port 80
```

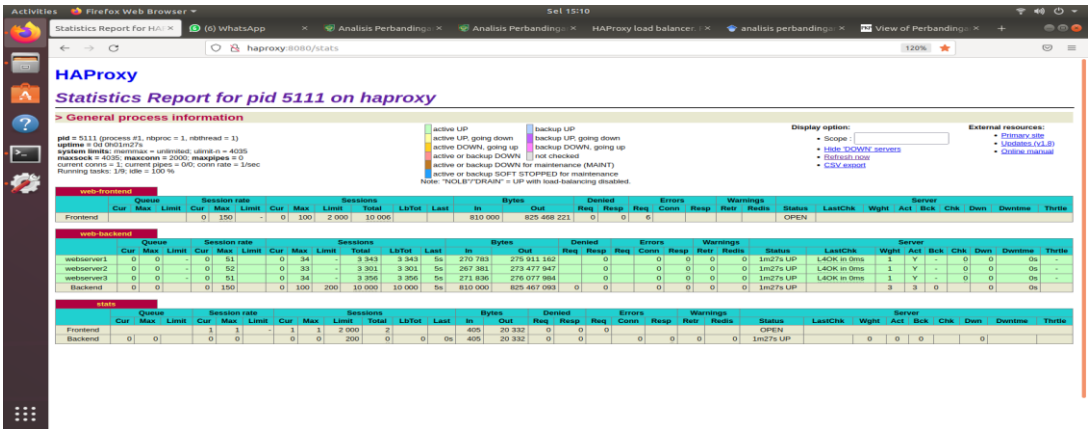
Gambar 8. Konfigurasi Algoritma *Least connection*

Tabel 2. Skenario Pengujian

Jenis Pengujian	Skenario
Pengujian Beban	Klaster *server*; 10.000 permintaan (Apache Bench & JMeter); algoritma round-robin & *least connection* Single *server*; 10.000 permintaan (Apache Bench & JMeter); algoritma round-robin & *least connection*
Pengujian Failover	Matikan *Worker* 2; 10.000 permintaan (Apache Bench & JMeter); algoritma round-robin & *least connection*



Gambar 9. Pengujian Apache Bench 10.000 permintaan klaster round-robin



Gambar 10. Pengujian Apache Bench 10.000 permintaan klaster least connection

HASIL DAN PEMBAHASAN

Pengujian Menggunakan Apache Bench

Pengujian berikut menggunakan Apache Bench dengan klaster *server* dan single *server* dengan 10.000 permintaan, dengan beban permintaan masing-masing 100/detik.

Pengujian ini dilakukan menggunakan algoritma round-robin dan *least connection* pada klaster *server*.

Tabel 3. Hasil Pengujian Apache Bench Klaster *Server* dengan Round-Robin

No	Request	Request per detik [#/dtk]	Waktu per Request [ms]	Kecepatan transfer [kbyte/dtk]

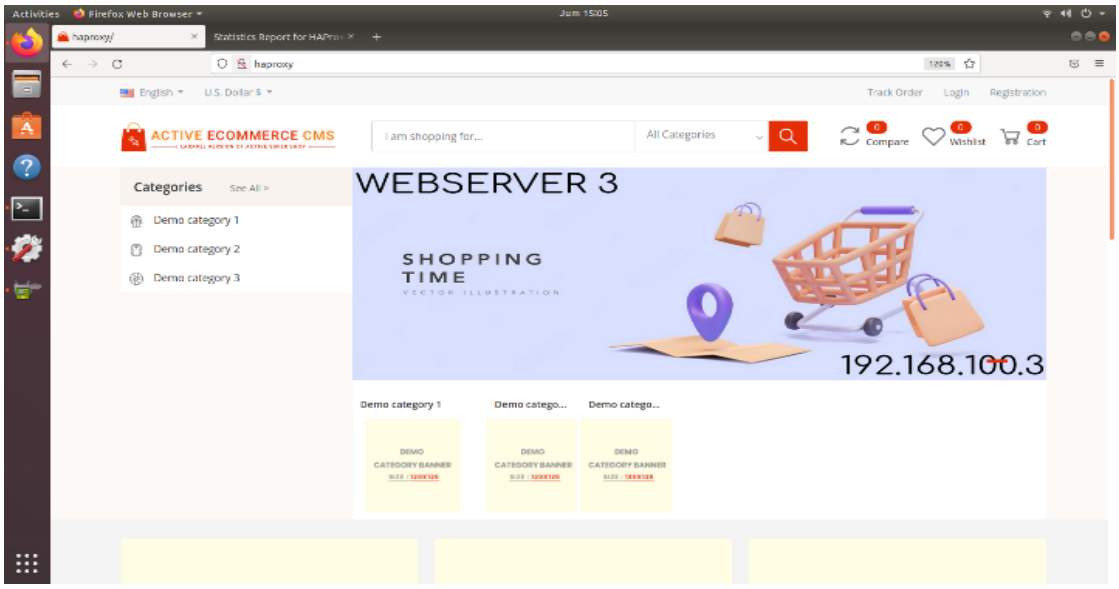
No	Request	Request per detik [#/dtk]	Waktu per Request [ms]	Kecepatan transfer [kbyte/dtk]
1	10.000	134,26	7,448	10.797,42

Tabel 3 menunjukkan hasil pengujian klaster *server* dengan algoritma round-robin untuk permintaan per detik, waktu per permintaan, dan kecepatan transfer. Hasil pengujian pada Tabel 3 dapat dijelaskan bahwa pengujian 10.000 permintaan menghasilkan 134,26 dtk untuk permintaan per detik, untuk waktu per permintaan menghasilkan 7,448 ms, dan untuk kecepatan transfer menghasilkan 10.797,42 kbyte/dtk.

Tabel 4. Hasil Pengujian Apache Bench Klaster *Server* dengan *Least connection*

No	Request	Request per detik [#/dtk]	Waktu per Request [ms]	Kecepatan transfer [kbyte/dtk]
1	10.000	133,40	7,496	10.750,99

Tabel 4 menunjukkan hasil pengujian klaster *server* dengan algoritma *least connection* untuk permintaan per detik, waktu per permintaan, dan kecepatan transfer. Hasil pengujian pada Tabel 4 dapat dijelaskan bahwa pengujian 10.000 permintaan menghasilkan 133,40 dtk untuk permintaan



Gambar 12. Pengujian akses aplikasi.

per detik, untuk waktu per permintaan menghasilkan 7,496 ms, dan untuk kecepatan transfer menghasilkan 10.750,99 kbyte/dtk. Pengujian klaster *server* dengan algoritma round-robin dan *least connection* juga dapat ditampilkan dalam laporan statistik dari HAProxy. Hasil pengujian dari Apache Bench ditampilkan dalam laporan statistik HAProxy. Gambar 9 menunjukkan hasil laporan statistik HAProxy dari pengujian Apache Bench dengan 10.000 permintaan yang dibagi oleh *load balancer* dengan algoritma round robin: *web server* 1 mendapatkan beban 3.334 permintaan, *web server* 2 mendapatkan beban 3.333 permintaan, dan *web server* 3 mendapatkan beban 3.333 permintaan — ditampilkan dalam tabel dari *web* backend pada bagian sessions di kolom total. Gambar 10 menunjukkan hasil laporan statistik HAProxy dari pengujian Apache Bench dengan 10.000 permintaan yang dibagi oleh *load balancer* dengan algoritma *least connection*: *web server* 1 mendapatkan beban 3.343 permintaan, *web server* 2 mendapatkan beban 3.301 permintaan, dan *web server* 3 mendapatkan beban 3.356 permintaan — ditampilkan dalam tabel dari *web*-backend pada bagian sessions di kolom total.

Pengujian *Failover*

Pada pengujian ini, terlihat bahwa kolom *web server* 2 berwarna merah pada Gambar 4.3, yang berarti *server* tersebut mati. Dalam pengujian, fungsi failover berjalan dengan baik; penulis mematikan *web server* 2 kemudian mencoba mengaksesnya menggunakan alat Apache Bench dengan 1.000 permintaan, di mana pembagian permintaan dialihkan ke *web server* 1 dan *web server* 3. Failover sendiri berfungsi untuk mengarahkan permintaan — jika satu atau lebih *server* mati, permintaan akan diarahkan ke *server* yang masih aktif.

Pembahasan

Penerapan metode horizontal scaling dalam proses *load balancing* berjalan dengan baik. *Load balancing* dapat memudahkan distribusi permintaan *client* secara merata ke setiap *web server*. Ini merupakan metode horizontal scaling sehingga *server* tidak mengalami overload dan dapat melayani 10.000 permintaan tanpa mengalami error. Penerapan algoritma round robin dan *least connection* juga berjalan sesuai yang diharapkan. Algoritma round-robin dapat meringankan beban setiap *web server* dalam memberikan layanan terhadap permintaan yang dibuat oleh *client* sehingga permintaan tersebut dapat diproses lebih cepat dibandingkan tanpa menggunakan klaster *server* karena dapat membagi beban secara cukup merata. Hal ini dapat dibuktikan dengan hasil pengujian menggunakan Apache Bench dengan 10.000 permintaan uji coba yang menghasilkan waktu respons 7,725 ms untuk klaster *server* dengan algoritma round-robin. Sebaliknya, algoritma *least connection* menghasilkan waktu respons 7,496 ms untuk 10.000 permintaan, dan untuk single *server* yang tidak menggunakan *load balancing* menghasilkan hasil 15,041 ms. Hasil pengujian menggunakan JMeter dalam uji coba 10.000 permintaan menghasilkan waktu respons 60 ms untuk klaster *server*, sementara hasilnya adalah 39.418 ms untuk single *server*. Dari pengujian menggunakan Apache Bench dan JMeter, klaster *server* yang mengimplementasikan *load balancing* dengan metode horizontal scaling dapat bekerja dengan baik karena dapat mempercepat proses permintaan dari beberapa permintaan sekaligus.

SIMPULAN

Pertama, *load balancing* dapat mengelola trafik atau permintaan layanan dari pengguna. *Load balancing* bekerja dengan baik; permintaan *client* dapat didistribusikan secara merata ke beberapa *server*. Kedua, dengan menggunakan metode horizontal scaling, penelitian ini dapat berjalan dengan baik. Penambahan sumber daya, sebagai metode horizontal scaling, dapat membantu penelitian ini mencapai tingkat ketersediaan atau availabilitas *server*. Algoritma round-robin dan *least connection*juga berkontribusi dalam performa *load balancing*, yang membagi trafik secara bergantian ke *server-server* yang tersedia untuk mencapai availabilitas pada *server* yang ada. Ketiga, implementasi *load balancing* dapat membantu marketplace daring XYZ, sebagai studi kasus dalam penelitian ini, karena dapat membagi beban secara tepat dan prosesnya lebih cepat dibandingkan single *server*. Keempat, pengujian *load balancing* menggunakan alat Apache Bench dan JMeter juga dapat memberikan perbedaan antara klaster *server* dan single *server*. Klaster *server* yang diuji pada Apache Bench dengan 10.000 permintaan menghasilkan rata-rata waktu respons 7,725 ms dengan algoritma round-robin. Algoritma *least connection* untuk pengujian yang sama menghasilkan waktu respons 7,496 ms, sementara single *server* menghasilkan waktu respons 15,041 ms. Kemudian dalam pengujian menggunakan JMeter dengan 10.000 permintaan, menghasilkan waktu respons 60 ms untuk klaster *server* dan untuk single *server*, waktu respons sebesar 39.418 ms. Dari pengujian yang dilakukan, terlihat bahwa penerapan *load balancing* dengan algoritma round-robin dan algoritma *least connection* dapat mempercepat permintaan layanan. Dalam penelitian ini, penulis memiliki keterbatasan dalam proses pengujian yang dilakukan; untuk proses penelitian ini, diharapkan dapat dikembangkan lebih lanjut dengan mencoba permintaan layanan yang lebih banyak dan menggunakan klaster *server database*.

DAFTAR PUSTAKA

Asiari, M. Z. (2021). *Analisis Kinerja Sistem Auto Scaling Pada Sistem Web Server Berbasis Clustering Menggunakan Sistem Virtual*. http://repository.unhas.ac.id/id/eprint/23398

Azi, M. N. A., Arifwidodo, B., & Wahyudi, E. (2023). Analisis Performansi Web Server Saat Menangani Permintaan Client Menggunakan Metode Reserve Proxy Caching dan Varnish. *Journal of Telecommunication, Electronics, and Control Engineering (JTECE)*, *5*(1), 14–21.

Azizah, S. (2017). Implementasi Load Balancing Web Server Menggunakan Haproxy. *Jurnal Manajemen Informatika*, (01), 11–19.

Chandra, A. Y. (2019). Analisis Performansi Antara Apache & Nginx Web Server Dalam Menangani Client Request. *JSI: Jurnal Sistem Informasi*, *14*(1), 48–56.

Dani, R., & Suryawan, F. S. T. (2017). *Perancangan dan Pengujian Load Balancing dan Failover Menggunakan NginX*.

Diarjo, A. A., & Mulyana, D. I. (2017). Penerapan Algoritma Round robin Dan Modulo Pada Load Balancing. *Jurnal CKI On SPOT*, *10*(1), 21–34.

Erlianto, A., Supendar, H., & Tutupoly, T. A. (2020). Implementasi High Availibility Virtualisasi Server Menggunakan Vmware Esxi Pada PT Grup Riset Potensial. *Jurnal Khatulistiwa Informatika*, *1*(2), 101–107.

Hakim, D. K., Yulianto, D. Y., & Fauzan, A. (2019). Pengujian Algoritma Load Balancing pada Web Server Menggunakan NGINX. *Jurnal Riset Dan Sain Teknologi*, *3*(2), 85.

Khadafi, S., Meilani, B. D., & Hidayat, S. A. (2017). Pengukuran Kompatibilitas Performa Komputer Server Menggunakan Jmeter Pada Raspberry Pi Dan PC Sebagai Layanan Web Server. *Prosiding Seminar Nasional Sains Dan Teknologi Terapan*, 157–162.

Pandega, B., & Widyaro, S. (2026). Implementasi Arsitektur Microservices Dalam Pengembangan Marketplace Sewa Properti. *Proceedings of the Informatics Conference*, *10*(21), 11–17. https://ojs.journals.unisel.edu.my/index.php/icf/article/view/408

Prakoso, R. B. B. (2020). *Analisis Kinerja Load Balancing Haproxy Dengan Metode Ratio Untuk Mengoptimalkan Kinerja Server Web*. Universitas Bumigora.

Pratama, R. A., Mayasari, R., & Sanjoyo, D. D. (2018). Implementasi Web Server Cluster Menggunakan Metode Load Balancing Pada Container Docker, Lxc. *EProceedings of Engineering*, *5*(3).

Pribadi, Y., Putra Negara, A. B., & Irwansyah, M. A. (2020). Analisis Penggunaan Metode Failover Clustering untuk Mencapai High Availability pada Web Server (Studi Kasus: Gedung Jurusan Informatika). *JUSTIN (Jurnal Sistem Dan Teknologi Informasi)*, *8*(2), 218–229. https://doi.org/10.26418/justin.v8i2.31965

Rahayu, D. T. (2021). *Implementasi High-Availability Server Pada Docker Swarm Menggunakan Metode Clustering*.

Rahmatulloh, A., & Msn, F. (2017). Implementasi Load Balancing Web Server menggunakan Haproxy dan Sinkronisasi File pada Sistem Informasi Akademik Universitas Siliwangi. *Jurnal Teknologi Dan Sistem Informasi*, *3*(2), 241–248.

Riawati, A. D., Irfan, M., Khaeruddin, K., & Faruq, A. (2022). High Availability Dynamic Sharding Database Server Dengan Metode Fail Over Dan Clustering. *Jurnal Manajemen Informatika Dan Sistem Informasi*, *5*(1), 1–10.

Riskiono, S. D., & Pasha, D. (2020). Analisis Metode Load Balancing Dalam Meningkatkan Kinerja Website E-Learning. *Jurnal TeknoInfo*, *14*(1), 22–26.

Triangga, H., Faisal, I., & Lubis, I. (2019). Analisis perbandingan algoritma static Round-Robin dengan Least-Connection terhadap efisiensi load balancing pada load balancer haproxy. *InfoTekJar: Jurnal Nasional Informatika Dan Teknologi Jaringan*, *4*(1), 70–75.

Umam, C., Handoko, L. B., & Rizqi, G. M. (2018). Implementation and analysis high availability network file system based server cluster. *Jurnal Transform.*, *16*(1), 31.

Wicaksono, H. (2019). *Implementasi Load Balancing Server Menggunakan HAPROXY pada Algoritma Round robin dan Algoritma Source*. Teknik Informatika, Fakultas Sains Dan Teknologi.