Jurnal Teknologi dan Manajemen Industri Terapan (JTMIT) Vol. 4, No. 4, Dsember 2025 pp. 2154 - 2165
P-ISSN: 2829-0232 E-ISSN: 2829-0038

# Implementasi Layer-7 Load Balancing Dengan Sistem Geo-Aware Failover Untuk Meningkatkan Reliabilitas Layanan Berbasis Web

**Wildy Sheverando[1], Agustinus Fritz Wijaya[2]**
[1,2)] Program Studi Informatika, Fakultas Teknologi Dan Desain, Universitas Bunda Mulia
Jl. Lodan Raya No. 2, Jakarta Utara, 14430, Indonesia
Email: [1]shiwildy@gmail.com, [2]agustinus.wijaya@bundamulia.ac.id

## ABSTRAK

*Sistem load balancing merupakan komponen penting dalam menjaga kinerja dan ketersediaan layanan pada infrastruktur jaringan berskala besar, namun banyak implementasi yang masih menggunakan algoritma statis seperti Round Robin yang belum mampu menyesuaikan kondisi kesehatan server dan latensi jaringan secara real-time, sehingga berpotensi meningkatkan latensi dan downtime ketika salah satu server mengalami gangguan. Penelitian ini bertujuan untuk merancang dan mengimplementasikan sistem Layer-7 Load Balancer berbasis Geo-Aware Failover dengan perhitungan Exponential Weighted Moving Average (EWMA) sebagai solusi untuk meningkatkan stabilitas distribusi beban, reliabilitas layanan, serta pengambilan keputusan adaptif pada lingkungan multi-region. Implementasi dilakukan menggunakan OpenResty (Nginx + LuaJIT) yang dilengkapi modul deteksi lokasi pengguna, pemeriksaan kesehatan backend, dan pembaruan nilai latensi secara berkelanjutan. Pengujian performa dan reliabilitas dilakukan menggunakan Apache JMeter dan PyTest dalam skenario normal, failover, dan failback. Hasil pengujian menunjukkan bahwa sistem mampu mempertahankan waktu respons rata-rata pada rentang 208–300 ms, transisi failover kurang dari 1 detik, serta tingkat ketersediaan layanan di atas 99%, dengan peningkatan performa yang lebih baik dibandingkan metode Round Robin konvensional. Dengan demikian, penelitian ini memberikan kontribusi berupa pengembangan mekanisme load balancing adaptif berbasis lokasi dan latensi yang dapat meningkatkan performa, kontinuitas layanan, dan kemampuan adaptif pada arsitektur layanan berbasis web berskala global.*

***Kata kunci**: Web Service Reliability, Layer-7 Load Balancing, Geo-Aware Failover, High Availability, Adaptive Latency-Based Routing.*

## *ABSTRACT*

*Load balancing systems are essential to maintaining performance and service availability within large-scale network infrastructures. However, many existing implementations still rely on static algorithms such as Round Robin, which are not capable of adapting to real-time server health and network latency, potentially resulting in increased latency and downtime when a backend server fails. This research aims to design and implement a Layer-7 Load Balancer based on Geo-Aware Failover and Exponential Weighted Moving Average (EWMA) calculations to improve load distribution stability, service reliability, and adaptive decision-making in multi-region environments. The system is implemented on OpenResty (Nginx + LuaJIT) equipped with modules for user geolocation detection, active backend health checking, and continuous latency evaluation. Performance and reliability testing were conducted using Apache JMeter and PyTest across normal, failover, and failback scenarios. The results show that the system can maintain an average response time of 208–300 ms, achieve failover in less than 1 second, and sustain service availability above 99%, outperforming the conventional Round Robin method. Therefore, this research provides a contribution in the form of an adaptive load-balancing mechanism based on geographic proximity and latency, which enhances performance, service continuity, and adaptive capability in globally distributed web service architectures.*

***Keywords:** Web Service Reliability, Layer-7 Load Balancing, Geo-Aware Failover, High Availability, Adaptive Latency-Based Routing.*

## Pendahuluan

Perkembangan layanan berbasis web dan sistem informasi terdistribusi meningkat pesat seiring dengan pertumbuhan layanan e-commerce, *streaming*, *financial technology*, serta aplikasi *real-time* lainnya yang membutuhkan performa tinggi dan *high availability* [1], [2], [3]. Pada arsitektur layanan global, pengguna berasal dari lokasi geografis yang sangat beragam sehingga perbedaan rute jaringan dan jarak fisik berdampak langsung pada *Round Trip Time (RTT)*, *throughput*, serta pengalaman akses pengguna akhir [4], [5]. Ketika permintaan meningkat, server tunggal dapat mengalami *overload* yang memicu *latency spike* dan penurunan kualitas layanan [6].

*Load balancing* menjadi pendekatan utama untuk mendistribusikan trafik ke banyak *backend* agar performa tetap stabil [7]. Namun, sebagian besar implementasi *Layer-7 load balancing* di lingkungan produksi masih mengandalkan algoritma statis seperti *Round Robin* dan *Least Connection*, yang tidak memperhitungkan kondisi kesehatan *backend* maupun dinamika jaringan saat permintaan terjadi [8], [9]. Jika salah satu server mengalami kegagalan atau degradasi kinerja, sistem tetap mengarahkan trafik ke server tersebut sehingga menyebabkan *downtime*, penolakan koneksi, dan potensi *service disruption* [10], [11].

Beberapa studi mencoba meningkatkan kinerja metode statis dengan menambahkan indikator performa seperti *connection weight* atau *queue utilization* [8], [12], namun masih belum memanfaatkan informasi latensi aktual dan kedekatan geografis pengguna. Padahal penelitian menunjukkan bahwa kedekatan regional dapat mengurangi latensi secara signifikan dan berdampak langsung pada *quality of experience (QoE)* pengguna [13], [14]. Selain itu, penelitian mengenai *geo-aware routing* pada *content delivery* umumnya hanya fokus pada optimasi pemilihan *edge node* dan tidak memiliki mekanisme pemulihan otomatis (*failover*) ketika terjadi kegagalan layanan [15], [16], [17].

Adapun penelitian pada domain *adaptive load balancing* cenderung hanya melakukan evaluasi pada lingkungan simulasi atau jaringan lokal (*private network*) tanpa mempertimbangkan fluktuasi latensi pada jaringan publik yang sesungguhnya [18], [19], [20]. Beberapa pendekatan menggunakan *ping-based RTT monitoring*, tetapi nilai RTT yang berfluktuasi ekstrem dapat menyebabkan pemilihan *backend* menjadi tidak stabil dan sering berubah (thrashing). Oleh karena itu, diperlukan mekanisme penstabil nilai latensi seperti *Exponential Weighted Moving Average (EWMA)* yang terbukti mampu memberikan tren latensi lebih representatif untuk pengambilan keputusan adaptif [21], [22].

Dari tinjauan literatur tersebut, terlihat bahwa masih terdapat research gap:

1. *Layer-7 load balancing* berbasis geo-aware failover.
2. Adaptasi pemilihan *backend* berdasarkan latensi aktual yang distabilisasi.
3. Evaluasi langsung pada jaringan internet publik multi-region, bukan simulasi.

Untuk menjawab kekosongan riset tersebut, penelitian ini mengembangkan sebuah mekanisme *Layer-7 Load Balancer* berbasis Geo-Aware Failover + EWMA-based adaptive routing yang diimplementasikan pada OpenResty agar keputusan *routing* dapat dilakukan secara *real-time* pada tingkat aplikasi [23].

Sistem ini dirancang untuk mampu:

1. Mendeteksi kegagalan *backend* dan melakukan *failover* otomatis dalam waktu sub-detik,
2. Memulihkan aliran trafik secara *failback* setelah server pulih.
3. Menjaga kestabilan latensi pada kondisi jaringan publik yang dinamis.

Evaluasi dilakukan dengan pengujian performa dan reliabilitas melalui skenario normal, failover, dan failback di lingkungan multi-region (Asia–US) sehingga menggambarkan kondisi operasional nyata [18], [19], [24]. Dengan demikian, kontribusi utama penelitian ini adalah peningkatan reliabilitas dan performa layanan web dengan pendekatan adaptif berbasis lokasi dan latensi, yang dapat diimplementasikan pada berbagai layanan berskala besar, termasuk *multi-region SaaS*, *edge computing*, dan sistem web modern lainnya [25].

## Metode Penelitian

Penelitian ini menggunakan metode *experimental research*, yaitu metode yang menekankan pada pengujian langsung terhadap sistem yang sedang dibangun pada lingkungan nyata. Pendekatan ini dipilih karena tujuan utama penelitian adalah mengevaluasi performa dan reliabilitas sistem *Layer-7 Load Balancer* berbasis *Geo-Aware Failover* ketika dijalankan di jaringan internet publik, bukan hanya di lingkungan simulasi. Dengan cara ini, hasil yang diperoleh dapat menggambarkan perilaku sistem pada kondisi operasional sesungguhnya, termasuk fluktuasi latensi, rute jaringan internasional, dan potensi kegagalan server yang terjadi sewaktu-waktu.

### 1. Lingkungan Pengujian

Untuk mendukung metode *experimental research* yang digunakan, sistem diuji pada tiga server yang ditempatkan di lokasi geografis berbeda. Satu *server* digunakan sebagai *load balancer* yang ditempatkan di Singapura, sedangkan dua *server* lainnya berperan sebagai *backend* yang ditempatkan di Indonesia dan Amerika Serikat. Semua server menggunakan sistem operasi Debian 13 dengan spesifikasi yang disesuaikan untuk kebutuhan pengujian. Penempatan ini dipilih untuk mensimulasikan kondisi layanan global, di mana perbedaan jarak fisik dan rute jaringan akan secara langsung mempengaruhi nilai latensi.

### 2. Tahapan Penelitian

Secara garis besar tahapan penelitian ini meliputi beberapa hal yakni:

1. Analisis kebutuhan, untuk mengidentifikasi kelemahan metode *load balancing* tradisional seperti *Round Robin* dan menentukan kebutuhan sistem yang adaptif.
2. Perancangan arsitektur, yang mencakup desain *Layer-7 Load Balancer*, mekanisme *Geo-Aware Failover*, *health check*, pengukuran *RTT*, dan perhitungan *EWMA*.

3. Implementasi sistem, menggunakan *OpenResty* (*Nginx + LuaJIT*) sebagai *platform* utama, dengan logika *routing* dan *monitoring* yang ditulis menggunakan bahasa Lua.
4. Pengujian sistem, yang dilakukan langsung di jaringan internet publik menggunakan *Apache JMeter* dan *PyTest*.
5. Analisis hasil, untuk mengevaluasi performa, reliabilitas, serta kelebihan dan kekurangan sistem yang dikembangkan.

## 3. Metode Pengujian

Pengujian dalam penelitian ini dibagi menjadi dua kelompok utama, yaitu pengujian performa dan pengujian reliabilitas. Pengujian performa dilakukan dengan *Apache JMeter* untuk mengukur waktu respons rata-rata, *throughput*, serta kestabilan sistem ketika menerima *requests* yang tinggi. Sementara itu, pengujian reliabilitas dilakukan dengan mensimulasikan skenario kegagalan *backend*, seperti mematikan salah satu *server* selama pengujian berlangsung, dan mengamati bagaimana sistem melakukan *failover* dan *failback.*

Seluruh pengujian dijalankan pada jaringan internet publik sehingga setiap hasil yang diperoleh mencerminkan kondisi operasional yang realistis. Dengan pendekatan ini, efektivitas arsitektur yang diusulkan dapat dinilai secara lebih objektif.

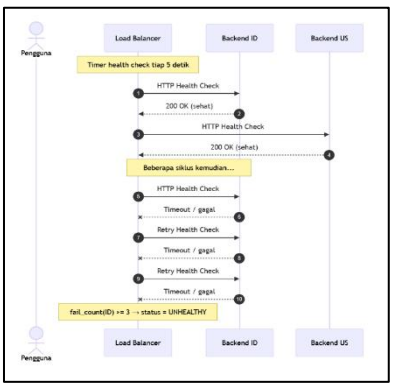
## 4. Perancangan Sistem

Perancangan sistem dalam penelitian ini difokuskan pada pengembangan mekanisme *adaptive load balancing* yang mampu menjaga kontinuitas layanan meskipun salah satu backend mengalami degradasi performa atau kegagalan total. Pendekatan ini bekerja pada Layer-7 sehingga load balancer tidak hanya melihat alamat IP dan port, tetapi dapat menganalisis karakteristik permintaan seperti *HTTP header* serta pola trafik untuk menentukan backend yang paling sesuai. OpenResty dipilih sebagai platform inti karena memiliki kemampuan untuk mengeksekusi logika berbasis Lua secara langsung pada reverse proxy, yang membuat proses pengambilan keputusan routing berlangsung secara cepat dan berorientasi kondisi aktual jaringan. Dengan desain ini, arah trafik tidak lagi bersifat statis seperti *Round Robin*, tetapi dinamis dan bergantung pada performa backend serta kedekatan geografis dengan pengguna.

Agar sistem mampu bereaksi secara akurat terhadap perubahan kondisi jaringan publik, load balancer dilengkapi modul *health monitoring* berkala dan mekanisme pengukuran Round-Trip Time (RTT) yang diperhalus menggunakan metode *Exponential Weighted Moving Average* (EWMA). Nilai EWMA ini berfungsi sebagai indikator utama dalam menentukan backend yang sedang berada dalam kondisi optimal. Ketika backend menunjukkan respons lambat atau gagal merespons beberapa kali berturut-turut, sistem akan menjalankan *failover* secara otomatis ke backend alternatif yang masih sehat dan memiliki nilai latensi terbaik. Sebaliknya, jika backend yang sebelumnya bermasalah telah kembali stabil, sistem memasukkannya kembali ke rotasi melalui proses *failback* yang terkontrol. Dengan perancangan ini, sistem dapat mempertahankan ketersediaan layanan tetap tinggi, waktu respons terjaga, dan pengalaman pengguna tetap konsisten walaupun terdapat gangguan di salah satu bagian infrastruktur.

Selain mengutamakan reliabilitas, perancangan sistem ini juga mempertimbangkan aspek skalabilitas dan kemudahan pengelolaan. Daftar backend disusun dalam konfigurasi terpisah sehingga penambahan server baru dapat dilakukan tanpa perubahan pada logika inti. Semua status backend disimpan pada memori bersama sehingga akses dan pembaruan informasi dapat dilakukan dengan latensi minimal. Arsitektur ini memungkinkan sistem untuk terus diperluas mengikuti pertumbuhan trafik, sekaligus memastikan setiap backend memperoleh trafik sesuai kapasitasnya. Dengan demikian, rancangan ini menjadi solusi yang tidak hanya responsif terhadap kegagalan, tetapi juga mampu menyesuaikan beban layanan di masa mendatang dengan usaha pemeliharaan yang relatif rendah.

### 4.1 Mekanisme *Health Check*

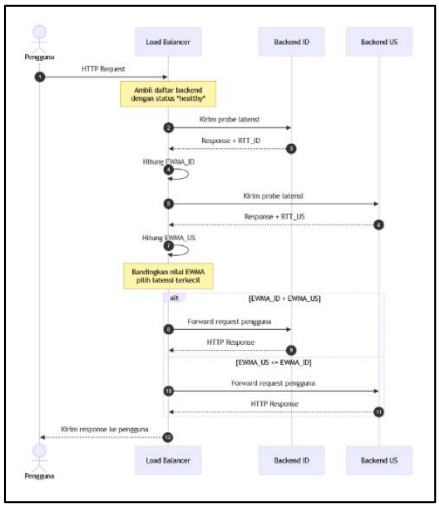



*Gambar 1. Mekanisme Health check*

Pada mekanisme *health check*, *load balancer* secara berkala melakukan *monitoring* terhadap setiap *backend server* untuk memastikan status kesehatannya. Proses ini berjalan otomatis menggunakan *timer* internal yang diatur pada interval tertentu seperti setiap 5 detik. Pada setiap siklus *monitoring*, *load balancer* mengirimkan permintaan *HTTP HEAD* ke masing-masing *backend* untuk memastikan bahwa server masih dapat memberikan respons dalam batas waktu yang telah ditentukan. Jika *backend* merespons dengan *status code* 200 atau respons sukses lainnya, maka server tersebut dianggap *healthy* dan tetap berada dalam rotasi distribusi trafik.

Namun apabila *backend* mulai mengalami gangguan dan gagal merespons pemeriksaan, *load balancer* akan mencatat kegagalan tersebut. Untuk mencegah kesalahan deteksi akibat gangguan jaringan sesaat, sistem menerapkan mekanisme *retry* beberapa kali secara berturut-turut. Jika jumlah kegagalan telah mencapai ambang batas seperti tiga kali berturut-turut, maka *backend* akan diberi status *unhealthy* dan sementara waktu dikeluarkan dari *server pool* aktif. Status tersebut tetap dipantau dalam pemeriksaan berikutnya sehingga ketika server kembali pulih, dapat dimasukkan lagi ke dalam rotasi melalui mekanisme *failback*. Dengan cara ini, sistem dapat memastikan bahwa permintaan pengguna selalu dialihkan ke *backend* yang paling responsif dan terpercaya untuk menjaga ketersediaan layanan secara berkelanjutan.

### 4.2 Mekanisme Pengukuran Latensi




**Gambar 2**. Mekanisme pengukuran Latensi

Pada mekanisme pengukuran latensi, *load balancer* melakukan evaluasi terhadap waktu respons setiap *backend server* untuk memastikan bahwa permintaan pengguna diteruskan ke server dengan performa yang terbaik pada saat itu. Setelah permintaan dari pengguna diterima, *load balancer* terlebih dahulu mengidentifikasi daftar *backend* yang berstatus *healthy*. Selanjutnya, *load balancer* mengirimkan *latency probe* berupa permintaan ringan ke masing-masing *backend*, kemudian menghitung nilai *Round Trip Time (RTT)* berdasarkan jeda waktu antara permintaan dikirim dan respons diterima kembali.

Nilai *RTT* yang diperoleh kemudian tidak langsung dijadikan dasar dalam pengambilan keputusan, karena kondisi jaringan publik dapat berubah secara dinamis sehingga berpotensi menghasilkan fluktuasi yang sangat tajam. Untuk itu digunakan metode *Exponential Weighted Moving Average (EWMA)* dalam menstabilkan data latensi. Setiap nilai *RTT* terbaru akan dipadukan dengan nilai *EWMA* sebelumnya sesuai nilai faktor pembobot ($\alpha$) yang ditentukan. Hasil perhitungan *EWMA* ini lebih dapat merepresentasikan kondisi latensi aktual namun tetap menjaga stabilitas agar keputusan pemilihan *backend* tidak berubah secara agresif.

Setelah semua nilai *EWMA* diperbarui, *load balancer* akan membandingkan performa setiap *backend* yang *healthy*, lalu memilih *backend* dengan latensi terendah sebagai server tujuan untuk memproses permintaan pengguna. Dengan mekanisme ini, sistem dapat melakukan *adaptive routing* secara real-time, sehingga pengalaman pengguna tetap optimal meskipun kondisi jaringan pada lintas wilayah berubah-ubah

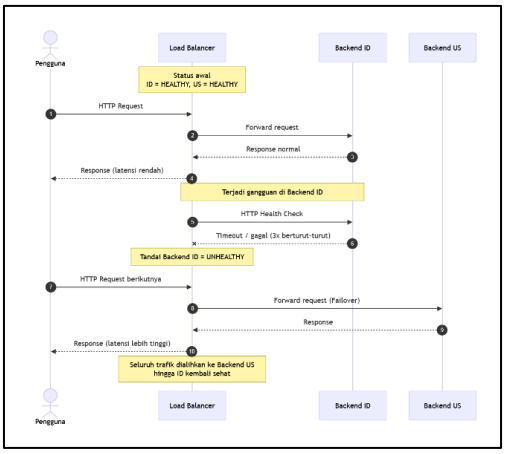
Rumus yang digunakan adalah:

$$EWMA = \alpha \times RTT_t + (1 - \alpha) \times EWMA_{t-}$$

Penjelasan rumus yang digunakan:

- $RTT_t$ adalah nilai RTT dari hasil pengukuran terbaru.
- $EWMA_{t-}$ Adalah nilai EWMA pada pengukuran sebelumnya.
- $\alpha$ adalah faktor pembobot antara 0 dan 1 yang menentukan seberapa besar pengaruh nilai RTT terbaru terhadap nilai EWMA. Semakin besar $\alpha$ semakin cepat EWMA mengikuti perubahaan RTT, semakin kecil $\alpha$ semakin stabil nilai EWMA dari waktu ke waktu

### 4.3 Mekanisme Failover

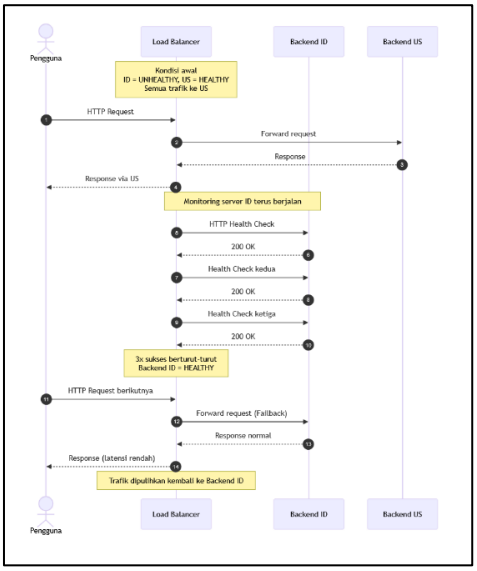



**Gambar 3**. Mekanisme Failover

Pada mekanisme *failover*, sistem akan mengalihkan seluruh trafik ke *backend server* lain yang masih dalam kondisi *healthy* ketika *backend* utama mengalami kegagalan. Proses ini dimulai dari pendeteksian kegagalan saat *load balancer* menjalankan *health check* berkala. Apabila *backend* utama gagal merespons beberapa kali secara berturut-turut dalam batas waktu tertentu, sistem akan menandai server tersebut sebagai *unhealthy*. Penandaan ini dilakukan agar *load balancer* tidak lagi meneruskan permintaan pengguna ke server yang bermasalah, sehingga mencegah terjadinya gangguan layanan seperti *request timeout* atau *service unavailable*.

Setelah status *unhealthy* diterapkan, *load balancer* mengambil keputusan untuk melakukan *failover* dengan memilih *backend* alternatif yang masih dalam keadaan *healthy* dan memiliki nilai *latency* terbaik berdasarkan pengukuran terakhir. Dengan demikian, proses perpindahan trafik dilakukan secara otomatis tanpa memerlukan campur tangan administrator dan tanpa disadari oleh pengguna akhir. Seluruh permintaan baru akan diarahkan ke *backend* cadangan hingga *backend* utama kembali dapat merespons secara normal dan dinyatakan pulih melalui mekanisme *failback*. Pendekatan ini memastikan kontinuitas layanan tetap terjaga meskipun terjadi gangguan pada salah satu *backend*, sehingga meningkatkan *service availability* dan pengalaman pengguna secara keseluruhan.

### 4.4 Mekanisme Failback




**Gambar 4**. Mekanisme Failback

Pada mekanisme *failback*, sistem mengembalikan alur trafik ke *backend server* utama setelah server tersebut dinyatakan pulih dan stabil. Proses ini tidak dilakukan secara langsung begitu *backend* kembali merespons, tetapi melalui rangkaian *health check* yang dijalankan beberapa kali secara berturut-turut. Setelah sebelumnya *backend* utama diberi status *unhealthy* akibat kegagalan berulang, *load balancer* tetap melanjutkan proses pemantauan dengan mengirimkan *health check request* secara berkala. Jika pada beberapa siklus pemeriksaan berturut-turut *backend* utama kembali memberikan respons yang valid dan berada dalam batas waktu yang normal, sistem akan mengubah statusnya menjadi *healthy*.

Setelah status *healthy* dipulihkan, *load balancer* mulai menjalankan proses *failback* dengan mengalihkan trafik kembali ke *backend* utama secara terkontrol. Pemulihan ini dapat dikombinasikan dengan evaluasi nilai *latency* atau *EWMA* untuk memastikan bahwa kualitas respons *backend* utama memang sudah layak digunakan sebagai rute utama. Dengan pendekatan tersebut, *backend* yang baru pulih tidak langsung menerima beban penuh secara mendadak sehingga risiko gangguan ulang dapat dikurangi. Mekanisme *failback* ini memastikan bahwa sistem tidak hanya mampu bereaksi terhadap kegagalan melalui *failover*, tetapi juga dapat mengoptimalkan kembali jalur layanan ketika kondisi server *backend* sudah normal, sehingga performa dan konsistensi layanan tetap terjaga.

### 4.5 Pseudo-code Geo-Aware Layer-7 Load Balancing

*Algoritma 1 Adaptive Geo-Aware Layer-7 Load Balancing*

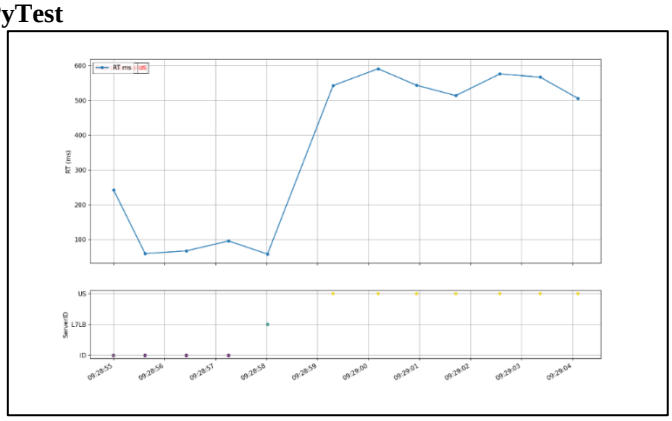
```
HEALTH_CHECK_INTERVAL = 5
FAIL_THRESHOLD       = 3
ALPHA                = 0.3

struct Backend {
   id
   region
   status
   fail_count
   rtt
   ewma
}

function background_monitoring(backends):
   every HEALTH_CHECK_INTERVAL seconds:
      for backend in backends:
         start = timestamp()
         result = probe(backend)
         latency = timestamp() - start

         if result == SUCCESS:
            backend.fail_count = 0
            backend.status = HEALTHY
            backend.rtt = latency
            if backend.ewma undefined:
               backend.ewma = backend.rtt
            else:
               backend.ewma = ALPHA * backend.rtt
                           + (1 - ALPHA) * backend.ewma
         else:
            backend.fail_count++
            if backend.fail_count >=
FAIL_THRESHOLD:
               backend.status = UNHEALTHY

function route_request(client_region, backends):
   healthy = filter(backends, status == HEALTHY)
   if healthy empty: return ERROR
   regional = filter(healthy, region == client_region)
   candidate = (regional not empty) ? regional : healthy
   best = null ; best_ewma = INF
   for backend in candidate:
      if backend.ewma defined and backend.ewma <
best_ewma:
         best = backend ; best_ewma = backend.ewma
   if best null: best = any(candidate)
   return best
```

Algoritma ini dijalankan sebagai bagian dari proses *monitoring* dan penanganan trafik pada *load balancer*, sehingga memastikan sistem tetap responsif terhadap perubahan kondisi jaringan publik secara real-time.

## Hasil Dan Pembahasan

Bagian ini menjelaskan hasil dari pengujian yang dilakukan terhadap sistem *Layer-7 Load Balancer* berbasis *Geo-Aware Failover*. Seluruh pengujian dilakukan di lingkungan internet 2160ublic agar hasil yang diperoleh mendekati kondisi operasional nyata. Selain menyajikan data hasil pengujian, bagian ini juga membahas pola yang muncul selama sistem diuji, termasuk bagaimana sistem merespons perubahan kondisi jaringan dan kegagalan pada backend.

### 1. Pengujian Failover PyTest




**Gambar 5**. Pengujian Failover menggunakan PyTest

Pada Gambar (5) memperlihatkan hasil pengujian proses *failover* yang dilakukan menggunakan *PyTest* untuk memantau perubahan kondisi respons *server* ketika *backend* utama (Indonesia) dimatikan secara sengaja. Pada grafik bagian atas, tampak pola perubahan nilai *Round Trip Time* (RTT) selama proses pengujian berlangsung. Sebelum terjadi *failover*, RTT berada pada kisaran rendah yaitu sekitar 40 hingga 120 ms, yang merupakan karakteristik normal ketika permintaan diarahkan ke *backend* Indonesia. Nilai RTT pada fase awal relatif stabil dengan variasi kecil yang masih dapat dianggap sebagai fluktuasi alami jaringan pada kondisi komunikasi normal.

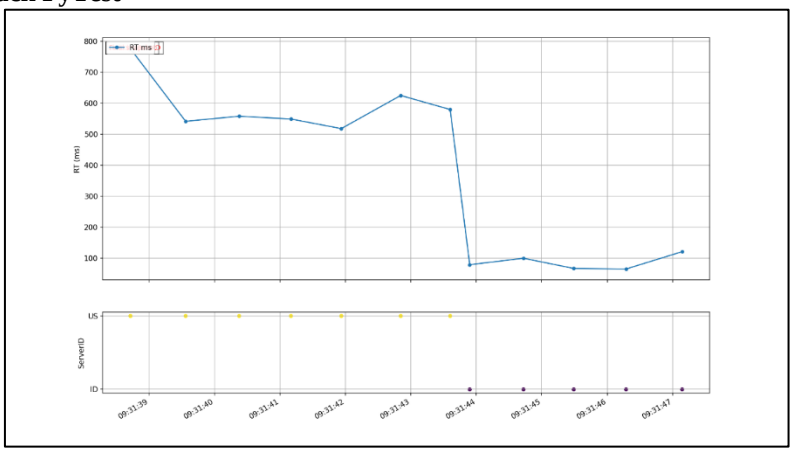
Perubahan signifikan mulai terjadi ketika *backend* Indonesia dimatikan. Pada titik ini, grafik menunjukkan lonjakan RTT yang cukup tajam. Lonjakan tersebut menandai momen perpindahan rute permintaan dari *backend* Indonesia ke *backend* Amerika Serikat sebagai *server* pengganti. Hal tersebut dapat diterima karena secara geografis jarak antara *client* dan *backend* Amerika Serikat jauh lebih besar sehingga waktu tempuh data otomatis meningkat. Setelah perpindahan berlangsung penuh, nilai RTT tampak bertahan pada rentang di atas 500 ms dan menjadi tren baru sepanjang *backend* utama masih tidak tersedia.

Pada grafik bagian bawah diperlihatkan pemilihan *backend* oleh sistem berdasarkan status kesehatan *server* yang terpantau. Pada fase awal, seluruh permintaan diarahkan ke *backend* Indonesia, ditunjukkan oleh penanda yang konsisten berada pada label ID. Ketika *backend* utama mengalami kegagalan, penanda berpindah ke label US dan tetap berada di sana hingga kondisi *backend* utama kembali normal. Perpindahan ini menunjukkan bahwa mekanisme *health check* bekerja sebagaimana mestinya, yaitu mendeteksi *server* yang tidak responsif lalu memicu pengalihan trafik secara otomatis tanpa campur tangan manual.

Dari kedua grafik tersebut dapat dilihat bahwa proses *failover* berjalan cepat dan efektif. Lonjakan RTT yang muncul hanya terjadi pada saat transisi perpindahan *backend*, dan setelah itu nilai RTT stabil pada rentang baru sesuai karakteristik *server* cadangan. Tidak terdapat indikasi *request* yang gagal (*error*), maupun jeda layanan yang terlalu lama yang dapat mengganggu pengguna. Dengan demikian, Gambar (5) memperlihatkan bahwa sistem berhasil mempertahankan kontinuitas layanan di tengah kondisi kegagalan, serta mampu memberikan pengalaman akses yang tetap konsisten walaupun *backend* utama tidak dapat digunakan.

Selain itu, hasil pengujian ini juga menguatkan bahwa logika pemilihan *server* yang berbasis *health check* dan pengukuran *latency* secara langsung mampu menyesuaikan diri dengan kondisi jaringan yang berubah-ubah. Observasi ini menjadi bukti bahwa sistem *load balancer* yang dirancang sudah memenuhi tujuan penelitian, yaitu meningkatkan reliabilitas layanan berbasis web dengan perpindahan trafik yang mulus dan dampak minimal bagi pengguna.

## 2 Pengujian Failback PyTest




**Gambar 6**. Pengujian Failback menggunakan PyTest

Pada Gambar (6) diperlihatkan perilaku sistem ketika *backend* utama (Indonesia) kembali aktif setelah sebelumnya dinonaktifkan dalam pengujian *failover*. Grafik bagian atas menunjukkan perubahan nilai *Round Trip Time* (RTT) selama proses transisi tersebut berlangsung. Pada fase awal grafik, seluruh permintaan masih diarahkan ke *backend* Amerika Serikat sehingga nilai RTT berada pada rentang yang lebih tinggi dan tampak fluktuatif. Kondisi ini dapat dipahami karena secara geografis koneksi ke *backend* AS memiliki jarak tempuh jaringan yang lebih jauh, sehingga waktu responsnya juga lebih besar dibandingkan dengan *backend* Indonesia.

Perubahan mencolok mulai terlihat ketika *backend* Indonesia terdeteksi kembali aktif. Pada titik tersebut, grafik menunjukkan adanya penurunan RTT yang cukup tajam dan terjadi dalam waktu yang relative singkat. Penurunan ini merupakan indikasi bahwa sistem telah mengalihkan trafik dari *backend* Amerika Serikat menuju *backend* Indonesia sebagai rute utama. Setelah perpindahan berlangsung, grafik memperlihatkan bahwa nilai RTT kembali stabil pada rentang rendah yang sesuai karakteristik normal saat pengguna dilayani oleh *backend* lokal.

Grafik bagian bawah turut memperkuat hasil tersebut dengan visualisasi pemilihan *backend* oleh sistem. Selama *backend* Indonesia belum dinyatakan sehat, semua marker berada pada label US. Setelah beberapa siklus pemeriksaan kondisi atau *health check* menunjukkan hasil positif, sistem secara otomatis memasukkan kembali *backend* Indonesia ke dalam rotasi layanan. Hal ini terlihat dari perpindahan marker pada grafik dari label US ke ID. Perubahan posisi marker tersebut menunjukkan bahwa sistem telah kembali memprioritaskan *backend* lokal dalam menangani permintaan pengguna.

Menariknya, perpindahan trafik tidak dilakukan secara mendadak, melainkan secara bertahap. Pendekatan ini penting untuk menghindari lonjakan beban mendadak pada *backend* yang baru saja pulih. Dengan memberikan waktu bagi *backend* Indonesia untuk kembali stabil, sistem dapat memastikan bahwa transisi tidak memunculkan masalah lanjutan seperti *overload* atau drop koneksi. Setelah proses peralihan selesai, nilai RTT terlihat kembali stabil pada kisaran rendah dan tidak terdapat lonjakan besar yang berulang, memberikan indikasi kuat bahwa *backend* Indonesia sudah siap menangani trafik secara optimal.

Secara keseluruhan, Gambar (6) menunjukkan bahwa mekanisme *failback* pada sistem bekerja sesuai dengan tujuan pengembangan. Sistem dapat mendeteksi kembalinya *backend* utama, memastikan kondisinya sudah stabil, lalu mengalihkan trafik secara aman, bertahap, dan efisien. Transisi yang mulus ini membuktikan bahwa sistem mampu mempertahankan kualitas layanan meskipun terjadi perubahan kondisi *backend* di belakang layar, sehingga dari sisi pengguna layanan tetap responsif tanpa gangguan berarti.

## 3 Pengujian dengan JMeter

Pada Gambar (7) menampilkan hasil pengujian performa menggunakan *Apache JMeter*, yang dilakukan untuk melihat bagaimana sistem merespons beban tinggi pada saat terjadi *failover* maupun *failback*. Pengujian ini memanfaatkan ratusan *virtual users* yang secara bersamaan mengirimkan permintaan ke *load balancer*, sehingga perubahan performa pada setiap tahapan dapat diamati secara lebih jelas.

Pada grafik bagian atas terlihat bahwa waktu respons mengalami fluktuasi sesuai dengan kondisi *backend* yang sedang aktif. Pada fase awal, saat *backend* Indonesia masih berfungsi *normal*, nilai waktu respons cenderung stabil pada rentang rendah. Polanya terlihat lebih merata, dengan variasi kecil yang wajar pada lingkungan jaringan publik. Transisi pertama terjadi ketika *backend* Indonesia dimatikan untuk memicu *failover*. Pada titik ini, kurva waktu respons menunjukkan peningkatan signifikan. Lonjakan ini merupakan dampak langsung dari perpindahan rute permintaan ke *backend* Amerika Serikat yang memiliki jarak fisik lebih jauh, sehingga waktu tempuh paket data menjadi lebih panjang.


| Requests | Executions | | | Response Times (ms) | | | | | | | Throughput | Network (KB/sec) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Label | #Samples | FAIL | Error % | Average | Min | Max | Median | 90th pct | 95th pct | 99th pct | Transactions/s | Received | Sent |
| Total | 2800 | 20 | 0.80% | 403.71 | 2 | 3614 | 63.00 | 1105.00 | 2724.90 | 3422.00 | 123.13 | 17.77 | 0.00 |
| Failback | 500 | 0 | 0.00% | 247.58 | 43 | 2105 | 72.00 | 774.90 | 1080.95 | 1156.73 | 39.07 | 9.42 | 0.00 |
| Failback Confirm | 500 | 0 | 0.00% | 602.42 | 2 | 3614 | 9.00 | 3410.90 | 3422.00 | 3480.00 | 30.61 | 0.00 | 0.00 |
| Failover | 500 | 0 | 0.00% | 353.46 | 43 | 2101 | 81.00 | 809.90 | 1147.75 | 2042.88 | 29.79 | 7.19 | 0.00 |
| Failover Confirm | 500 | 0 | 0.00% | 606.84 | 2 | 3576 | 8.00 | 2474.60 | 2828.00 | 3319.91 | 25.57 | 0.00 | 0.00 |
| Normal | 500 | 20 | 4.00% | 208.23 | 39 | 1177 | 74.50 | 558.00 | 601.00 | 710.77 | 24.81 | 5.93 | 0.00 |

**Gambar 7**. Pengujian Failover menggunakan JMeter

Meski terjadi peningkatan waktu respons, grafik tetap menunjukkan bahwa sistem dapat mempertahankan kestabilan setelah transisi selesai. Waktu respons berada pada kisaran baru yang lebih tinggi namun konsisten, menandakan bahwa *backend* Amerika Serikat mampu menangani beban dengan baik tanpa adanya tanda-tanda penurunan performa yang ekstrem.

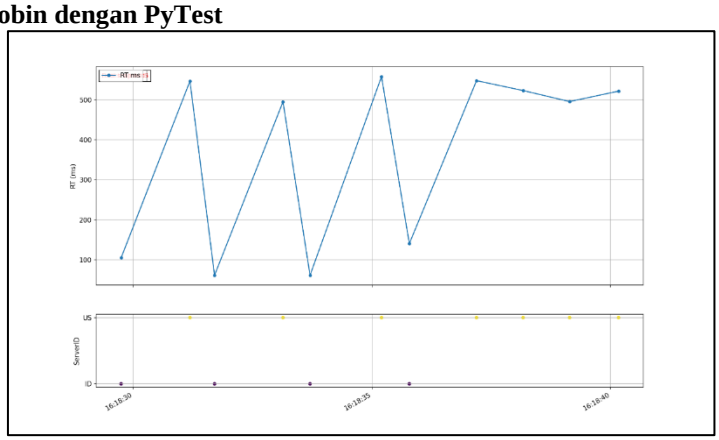
Pada tahap berikutnya, ketika *backend* Indonesia kembali diaktifkan, grafik menunjukkan pola penurunan waktu respons yang cukup jelas. Setelah proses kesehatan *backend* terdeteksi stabil kembali, sistem memulai proses *failback*. Pada titik tersebut, nilai waktu respons kembali turun dan mendekati rentang awal, yang mengonfirmasi bahwa *backend* Indonesia kembali menjadi *backend* utama. Transisi ini berlangsung dengan cukup mulus tanpa menyebabkan lonjakan waktu respons yang mengganggu.

Tabel statistik pada bagian bawah gambar memperkuat observasi dari grafik. Terlihat bahwa pengujian menghasilkan tingkat *error* yang sangat rendah bahkan mendekati 0% di sebagian besar skenario yang menunjukkan bahwa sistem mampu menangani permintaan secara konsisten meskipun *backend* berubah. Nilai *throughput* juga menunjukkan stabilitas yang baik, di mana selama periode *failover* throughput tetap dapat dipertahankan pada angka yang wajar. Perbedaan throughput antara fase *failover* dan normal dapat dijelaskan oleh kondisi jaringan lintas benua yang cenderung lebih lambat, sehingga berdampak langsung pada jumlah permintaan per detik yang dapat diproses.

Dari hasil statistik tersebut, terlihat pula bahwa rata-rata waktu respons pada fase *failover* lebih tinggi dibandingkan pada fase normal atau *failback*. Hal ini sesuai dengan teori bahwa pemilihan server yang berada secara geografis jauh akan memberikan latensi yang lebih tinggi. Namun demikian, perbedaan tersebut tetap berada dalam rentang yang dapat diterima dan tidak menimbulkan anomali yang dapat mengganggu keseluruhan performa sistem.

Secara keseluruhan, pengujian JMeter memperlihatkan bahwa sistem *load balancer* tidak hanya mampu mempertahankan kinerja pada kondisi normal, tetapi juga tetap stabil ketika terjadi *failover* maupun *failback*. Pergantian *backend* tidak menyebabkan lonjakan error ataupun kegagalan respons yang signifikan, dan waktu respons tetap berada dalam batas wajar. Hal ini menunjukkan bahwa mekanisme *failover* dan *failback* berjalan sesuai desain dan mampu mendukung kebutuhan aplikasi yang membutuhkan reliabilitas tinggi.

## 4 Pengujian RoundRobin dengan PyTest




**Gambar 8**. Pengujian Roundrobin dengan PyTest

Pada Gambar (8) menunjukkan hasil pengujian algoritma *Round Robin* yang digunakan sebagai pembanding terhadap pendekatan *Layer-7 Geo-Aware* yang dikembangkan pada penelitian ini. Dari grafik paling atas dapat terlihat bahwa nilai *RTT* berubah dengan sangat tajam dari satu titik ke titik berikutnya, membentuk pola naik turun yang ekstrem. Pola ini muncul karena *Round Robin* membagi beban secara bergantian antara *backend* Indonesia dan *backend* Amerika Serikat tanpa mempertimbangkan lokasi pengguna, kondisi jaringan, maupun performa *backend* pada saat itu. Ketika giliran *backend* Indonesia tiba, waktu respons berada pada kisaran rendah karena jarak klien relatif dekat namun pada saat rute berpindah ke backend Amerika Serikat, nilai *RTT* melonjak tinggi akibat jarak geografis dan rute jaringan internasional yang lebih panjang. Pergantian yang terjadi pada setiap permintaan inilah yang membuat grafik *RTT* tampak sangat tidak stabil dan mengalami ayunan besar dari nilai rendah ke nilai tinggi secara berulang.

Pada grafik bagian bawah, pergerakan titik antar *backend* juga terlihat bergantian dengan interval yang sangat pendek. Hampir setiap pengukuran menunjukkan perpindahan server, yang menandakan bahwa *Round Robin* tidak memiliki mekanisme untuk mempertahankan rute yang paling efisien. Pola pemilihan *server* yang terlalu sering berubah ini menyebabkan kondisi sistem menjadi tidak konsisten. Pada beberapa titik, pengguna merasakan respons cepat, namun pada titik lain, tiba-tiba terjadi peningkatan waktu respons yang cukup signifikan tanpa ada perubahan kondisi jaringan yang jelas. Ketidakstabilan ini diperburuk oleh sifat *Round Robin* yang tidak melakukan deteksi kualitas jaringan atau kondisi *backend*; ia hanya berputar secara mekanis mengikuti urutan yang ditentukan.

Jika dibandingkan dengan metode pada penelitian ini, pola *Round Robin* terlihat jauh lebih tidak teratur dan kurang responsif terhadap dinamika jaringan. Tidak ada proses penyaringan nilai latensi, tidak ada pertimbangan lokasi pengguna, dan tidak ada mekanisme adaptif yang memprioritaskan server dengan waktu respons terbaik. Akibatnya, setiap permintaan diperlakukan sama tanpa memperhatikan apakah backend yang diarahkan sedang berada pada kondisi optimal atau tidak. Situasi seperti ini sangat merugikan pada layanan yang bersifat real-time atau yang mengharuskan konsistensi waktu respons, karena variasi yang begitu mencolok dapat berdampak langsung pada pengalaman pengguna.

Dalam konteks praktik operasional, *Round Robin* juga rentan menimbulkan ketidak stabilan jangka panjang. Ketika server yang lebih jauh atau sedang mengalami peningkatan latensi mendapat giliran, pengguna yang diteruskan ke server tersebut akan merasakan penurunan performa walaupun backend lokal sebenarnya masih dapat memberikan respons yang jauh lebih baik. Kondisi ini hampir tidak mungkin dihindari dengan *Round Robin* karena algoritma tersebut tidak memiliki kemampuan untuk mengenali backend mana yang sedang mengalami penurunan performa. Hal ini berbanding terbalik dengan hasil pengujian menggunakan metode *Layer-7 Geo-Aware* yang memanfaatkan nilai latensi terolah, di mana sistem hanya berpindah *backend* ketika memang diperlukan, seperti ketika *backend* utama tidak sehat atau latensinya benar-benar naik secara konsisten.

Secara keseluruhan, hasil pengujian *Round Robin* ini menegaskan bahwa meskipun sederhana dan mudah diterapkan, algoritma tersebut tidak cocok digunakan pada skenario yang melibatkan distribusi trafik lintas wilayah geografis. Grafik menunjukkan bahwa Round Robin gagal menjaga kestabilan waktu respons dan cenderung menghasilkan pengalaman pengguna yang tidak konsisten. Dengan tidak adanya mekanisme seleksi berbasis performa, algoritma ini tidak mampu mengakomodasi jaringan modern yang sangat fluktuatif. Perbandingan ini memperkuat temuan penelitian bahwa pendekatan *Layer-7* dengan penyesuaian berbasis latensi dan lokasi pengguna memberikan kinerja yang jauh lebih stabil, adaptif, dan andal dibandingkan metode *Round Robin* tradisional.

## 5. Perbandingan Latensi Rata-rata antar backend

**Tabel 1**. Perbandingan Latensi Rata-rata antar backend (ms)

| Tujuan/Sumber | Backend Indonesia | Backend Amerika |
|---|---|---|
| Pengguna Asia | 45-60 ms | 200-240 ms |
| Pengguna Amerika | 210-250 ms | 40-70 ms |
| Cross-Region | 200+ ms | 200+ ms |

Tabel 1 menunjukkan bahwa latensi sangat dipengaruhi oleh lokasi geografis *backend server* yang dipilih. Pada kondisi normal, koneksi pengguna Asia yang diarahkan ke backend Indonesia berada pada rentang rendah yaitu 45–60 ms, sedangkan koneksi lintas-region menuju backend Amerika meningkat signifikan hingga 200–240 ms. Sebaliknya, pengguna Amerika mendapatkan latensi terbaik ketika trafik diproses oleh backend Amerika. Temuan ini menegaskan bahwa pemilihan backend berbasis kedekatan geografis merupakan faktor penting dalam menjaga performa dan pengalaman pengguna.

**Tabel 2**. Uji Performa dengan JMeter

| Skenario | Region Pengguna | Target Backend | Rata-rata | Minimum | Maksimum |
|---|---|---|---|---|---|
| Normal | Asia-ID | Indonesia | 40-120 ms | 35 ms | 130 ms |

| | | | | | |
|---|---|---|---|---|---|
| Failover | Asia-US | Amerika | 500-620 ms | 480 ms | 650 ms |
| Failback | Asia-ID | Indonesia | 35-100 ms | 32 ms | 110 ms |

Tabel 2 hasil pengujian *performance* menggunakan *Apache JMeter* dalam tiga skenario: normal, *failover*, dan *failback*. Pada kondisi normal, latensi berada pada rentang 40–120 ms yang merupakan kondisi ideal saat backend Indonesia aktif. Ketika terjadi *failover*, seluruh trafik dialihkan ke backend Amerika sehingga latensi meningkat tajam mencapai 500–620 ms. Setelah backend Indonesia pulih dan dilakukan *failback*, latensi kembali turun ke kisaran 35–100 ms. Perbandingan ini membuktikan bahwa mekanisme *Geo-Aware Failover* yang diterapkan mampu mempertahankan kontinuitas layanan dengan dampak minimum bagi pengguna.

## Simpulan

Penelitian ini berhasil mengimplementasikan *Layer-7 Load Balancer* berbasis *Geo-Aware Failover* dengan pemanfaatan metode *Exponential Weighted Moving Average (EWMA)* untuk pengambilan keputusan routing secara adaptif di lingkungan multi-region. Pengujian pada jaringan internet publik menunjukkan bahwa sistem mampu mempertahankan waktu respons rata-rata pada rentang 208–300 ms ketika backend utama beroperasi normal. Ketika backend mengalami kegagalan, mekanisme *failover* dapat berlangsung dalam waktu kurang dari satu detik tanpa memunculkan *downtime* yang dirasakan pengguna, serta layanan tetap tersedia dengan tingkat ketersediaan di atas 99%. Setelah backend utama pulih, proses *failback* berlangsung stabil dan memastikan beban kembali terdistribusi optimal.

Hasil ini memperlihatkan bahwa pendekatan adaptif berbasis lokasi dan tren latensi lebih unggul dibanding algoritma statis seperti *Round Robin* dalam menjaga reliabilitas dan performa layanan web berskala global. Kedepannya, sistem dapat dikembangkan dengan penambahan *security-aware routing* serta dukungan terhadap lebih banyak region dan konteks layanan seperti *edge computing* dan *multi-region SaaS*.

## Daftar Pustaka

[1] H. Tussyadiah, R. M. Negara, and D. D. Sanjoyo, "Distributed gateway-based load balancing in software defined network," *Telkomnika (Telecommunication Comput. Electron. Control.*, vol. 18, no. 5, pp. 2352–2361, 2020, doi: 10.12928/TELKOMNIKA.V18I5.14851.

[2] F. Tapia, M. ángel Mora, W. Fuertes, H. Aules, E. Flores, and T. Toulkeridis, "From monolithic systems to microservices: A comparative study of performance," *Appl. Sci.*, vol. 10, no. 17, 2020, doi: 10.3390/app10175797.

[3] R. Tsyganok, "Methodology for Building Scalable Microservice Architectures on Go for High-Load E-Commerce Platforms," *Univers. Libr. Eng. Technol.*, vol. 01, no. 02, pp. 42–46, 2024, doi: 10.70315/uloap.ulete.2024.0102007.

[4] S. P. Sitorus, E. R. Hasibuan, and R. Rohani, "Analysis performance of content delivery network by used Rateless Code method," *Sinkron*, vol. 7, no. 4, pp. 2348–2359, 2022, doi: 10.33395/sinkron.v7i4.11651.

[5] Sree Priyanka Uppu, "Content Delivery Networks (CDNs) and live streaming: architecting scalable delivery for high-demand events," *World J. Adv. Res. Rev.*, vol. 26, no. 2, pp. 2153–2164, 2025, doi: 10.30574/wjarr.2025.26.2.1849.

[6] P. Dymora, M. Mazurek, and B. Sudek, "Comparative analysis of selected open-source solutions for traffic balancing in server infrastructures providing www service," *Energies*, vol. 14, no. 22, 2021, doi: 10.3390/en14227719.

[7] K. Prasojo and A. M. F. Fadhlurrahman, "Analisis Performa Load balancing pada Web Server Menggunakan Nginx," vol. 3, no. 1, pp. 30–39, 2025.

[8] I. Technology, "Evaluation and Comparison of Load Balancing Algorithm Performance in the Implementation of Weighted Least Connections and Round Robin in Cloud Computing Environment," *J. Comput. Sci. Inf. Technol. Telecommun. Eng.*, vol. 6, no. 1, 2025, doi: 10.30596/jcositte.v6i1.21731.

[9] 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.

[10] V. Mohammadian, N. J. Navimipour, M. Hosseinzadeh, and A. Darwesh, "Fault-Tolerant Load

Balancing in Cloud Computing: A Systematic Literature Review," *IEEE Access*, vol. 10, pp. 12714–12731, 2022, doi: 10.1109/ACCESS.2021.3139730.

[11] D. Sharmah and K. C. Bora, "A Survey on Dynamic Load Balancing Techniques in Cloud Computing," *Lect. Notes Electr. Eng.*, vol. 1088, no. January, pp. 273–282, 2024, doi: 10.1007/978-981-99-6855-8_21.

[12] H. F. Pérez and M. Galicia García, "Journal of Applied Research and Technology," *J. Appl. Res. Technol.*, vol. 18, pp. 245–258, 2020.

[13] Z. Li, Z. Li, and W. Zhang, "Quality-Aware Task Allocation for Mobile Crowd Sensing Based on Edge Computing," *Electron.*, vol. 12, no. 4, 2023, doi: 10.3390/electronics12040960.

[14] T. Edge *et al.*, "The Impact of Edge Computing on the Data Center The Impact of Edge Computing on the Data Center".

[15] D. Agung Rizky Ananta, P. Hari Trisnawan, and K. Amron, "Implementasi Load Balancing Web Server pada Software Defined Networking menggunakan Metode K-Nearest Neighbor (K-NN)," *J. Pengemb. Teknol. Inf. dan Ilmu Komput.*, vol. 4, no. 10, pp. 3598–3606, 2020, [Online]. Available: http://j-ptiik.ub.ac.id

[16] M. C. Saxena, M. Sabharwal, and P. Bajaj, "Review of SDN-based load-balancing methods, issues, challenges, and roadmap," *Int. J. Electr. Comput. Eng. Syst.*, vol. 14, no. 9, pp. 1031–1049, 2023, doi: 10.32985/ijeces.14.9.8.

[17] O. M. A. Alssaheli, Z. Z. Abidin, N. A. Zakaria, and Z. A. Abas, "Software Defined Network based Load Balancing for Network Performance Evaluation," *Int. J. Adv. Comput. Sci. Appl.*, vol. 13, no. 4, pp. 117–124, 2022, doi: 10.14569/IJACSA.2022.0130414.

[18] H. Allam, "Reliability at the Edge: SRE for Distributed Cloud and IoT Platforms," *Int. J. Emerg. Res. Eng. Technol.*, vol. 6, no. 2, pp. 39–52, 2025, doi: 10.63282/3050-922x.ijeret-v6i2p106.

[19] V. S. Ramdoss, "Optimizing System Performance: Load Balancers and High Availability," *Eastasouth J. Inf. Syst. Comput. Sci.*, vol. 1, no. 02, pp. 113–117, 2023, doi: 10.58812/esiscs.v1i02.435.

[20] E. J. García Fernández de Castro, E. J. García Puche, and D. Jabba Molinares, "Dynamic Low-Latency Load Balancing Model to Improve Quality of Experience in a Hybrid Fog and Edge Architecture for Massively Multiplayer Online (MMO) Games," *Appl. Sci.*, vol. 15, no. 12, pp. 1–29, 2025, doi: 10.3390/app15126379.

[21] M. N. A. Rizqi and I. K. Dwi Nuryana, "Analisis Perbandingan Kinerja Algoritma Weighted Round Robin dan Weighted Least Connection Menggunakan Load Balancing Nginx Pada Virtual Private Server(VPS)," *J. Informatics Comput. Sci.*, vol. 4, no. 01, pp. 67–75, 2022, doi: 10.26740/jinacs.v4n01.p67-75.

[22] A. Asghar, A. Abbas, H. A. Khattak, and S. U. Khan, "Fog Based Architecture and Load Balancing Methodology for Health Monitoring Systems," *IEEE Access*, vol. 9, pp. 96189–96200, 2021, doi: 10.1109/ACCESS.2021.3094033.

[23] X. Shi, Y. Li, C. Jia, X. Hu, and J. Li, "L7LB: High Performance Layer-7 Load Balancing on Heterogeneous Programmable Platforms," *IEEE INFOCOM 2023 - Conf. Comput. Commun. Work. INFOCOM WKSHPS 2023*, 2023, doi: 10.1109/INFOCOMWKSHPS57453.2023.10225882.

[24] A. Syarif Aziz, "Analisis Kinerja Web Server Apache, Nginx, Open Litespeed, Dan Open Resty," *J. Informatics Comput. Sci.*, vol. 11, no. 1, pp. 1–9, 2025.

[25] D. Aldossary, E. Aldahasi, T. Balharith, and T. Helmy, "A Systematic Literature Review on Load-Balancing Techniques in Fog Computing: Architectures, Strategies, and Emerging Trends," *Computers*, vol. 14, no. 6, pp. 1–42, 2025, doi: 10.3390/computers14060217.