IMPLEMENTASI *LOAD BALANCING* WEB *SERVER* DENGAN HAPROXY MENGGUNAKAN ALGORITMA ROUND ROBIN

Muhammad Arigoh Waluyo [1)], Fery Antony[2)] dan Candra Setiawan [3)]

Program Studi Sistem Komputer UNIVERSITAS INDO GLOBAL MANDIRI
JL. Jend Sudirman No. 629, Palembang 30129, Sumatera Selatan
Email: 2019310051@students.uigm.ac.id [1)], feryantony@uigm.ac.id [2)], chandarsetiawan@uigm.ac.id [3)]

***Abstract*— Ketika banyak orang yang menggunakan atau login ke platform *online* atau website pada saat yang bersamaan, akan membebani *server hosting* website karena tidak dapat memproses *request* yang masuk. Masalah ini biasanya terjadi ketika *server* web adalah mesinnya sendiri atau *server* tunggal yang meng*hosting* situs web yang harus berbagi sumber dayanya seperti prosesor dan memori. Untuk mengatasi masalah tersebut digunakan teknik *load balancing* untuk mengatasi beban *server* yang diakses secara bersamaan dengan jumlah akses yang tinggi. Salah satu aplikasi load balancing yang digunakan adalah Haproxy. Pengujian ini dilakukan menggunakan simulasi beban server web dengan 100, 500, 1000, 2000, 2500 hingga 3000 pengguna secara bersamaan. Untuk membebani *server* dengan jumlah akses yang telah ditentukan, digunakan aplikasi penghasil lalu lintas, Apache JMeter. Hasilnya, web *server* berjalan normal dan mampu melayani akses dari pengguna. Pengujian dilakukan dengan menghitung *throughput* dan *responses time* antara web *server* dan haproxy *server*. Melalui Haproxy beban *throughput* dibagi antara satu *server* dan lainnya. Pengujian membuktikan bahwa Haproxy mampu membagi beban antar *server* web.**

***Kata Kunci*— Haproxy, Load Balancing, Web Server**

I. PENDAHULUAN

Perkembangan dalam beberapa tahun terakhir, kemajuan teknologi telah mengalami perkembangan yang signifikan., dan orang lebih termotivasi untuk menemukan dan mengembangkan teknologi baru dengan tujuan membantu orang bekerja secara efektif dan efisien. Pengolahan data menjadi informasi dapat dilakukan dengan cepat dan mudah menggunakan komputer. Teknologi informasi diharapkan menjadi media yang dapat mengontrol kebutuhan dan keinginan setiap pengguna.

Pentingnya internet saat ini tak dapat disangkal, menjadi sebuah kebutuhan yang tidak bisa diabaikan karena semakin banyak orang yang mencari informasi melalui internet dan mengunjungi website. Oleh karena itu website sebagai bagian dari internet perlu sering dikunjungi oleh para penggunanya dan terus berkembang. Meski begitu, web *server* memiliki kapasitas yang berbeda dalam menangani permintaan pengguna. Sebagian besar situs web dapat beroperasi dengan lancar selama hanya beberapa pengguna yang mengaksesnya pada saat yang bersamaan. Namun, ada masalah ketika ribuan pengguna mengakses situs atau *website* secara bersamaan. [1] Dengan semakin banyaknya pengguna internet, maka perlu dilakukan penyesuaian *routing* dan beban trafik jaringan, serta pengaturan alamat IP agar dapat mengoptimalkan jaringan internet secara efektif. Hal ini bertujuan untuk mencapai keseimbangan dan pemerataan dalam mengatur beban trafik jaringan. [2]

Masalah ini biasanya terjadi dengan *server* web mesin tunggal atau *server* web tunggal. Solusinya adalah dengan menggunakan *Scaling, Caching, Content Delivery Network* (CDN) dan *load balancer*. Penyeimbang beban sering digunakan untuk menangani kelebihan beban di *server* web. *Load balancer* adalah perangkat lunak yang membantu mendistribusikan beban dan menangani permintaan dari pengguna. Beban kueri atau distribusi akses pengguna didistribusikan secara merata di semua *server* yang ada, yang juga meningkatkan kinerja *server*. Haproxy sebagai load balancer digunakan untuk mengkoordinasikan distribusi beban. Selain itu, Haproxy menyertakan perangkat lunak gratis yang andal untuk ketersediaan tinggi, penyeimbangan beban, dan proksi TCP dan HTTP. [2]

Load balancing mengimplementasikan algoritma Round Robin. Hal ini dikarenakan dengan menggunakan metode algoritma ini, setiap permintaan dari *client* dialihkan ke

Pertama Muhammad Arigoh Waluyo adalah mahasiswa di Universitas Indo Global Mandiri di Palembang, 30129, INA. Ia telah menyelesaikan masa studinya di program studi sistem komputer dalam waktu 3,5 tahun. (email: 2019310051@students.uigm.ac.id).

Kedua Fery Antony sedang bersama dosen fakultas ilmu komputer program studi sistem komputer, universitas indo global mandiri Palembang 30129 INA. Gelar master dari Universitas Indonesia, Depok 16424 INA. Saat ini sedang menjalani pendidikan gelar Doktor dalam bidang ilmu komputer. di Universitas Sriwijaya, Palembang, 30128 INA (email: feryantony@uigm.ac.id).

Ketiga Candra Setiawan dengan dosen fakultas ilmu komputer program studi sistem komputer, universitas indo global mandiri Palembang 30129 INA. dia lulus dengan gelar master di Institut Teknologi Bandung (ITB) di bidang telekomunikasi. (email: candrasetiawan@uigm.ac.id).

beberapa *real server*, sehingga beban pada beberapa server dapat dibagi secara merata. Penambahan *server* yang menggunakan algoritma Round Robin dan mesin *load balancer* secara tidak langsung mengurangi beban pada *server* sehingga meminimalisir *overload*. [3] Pengembangan algoritma Round Robin yaitu Round Robin Weighted, namun dengan penambahan kondisi baru. Akan mempertimbangkan dan mengalihkan beban *server* dengan kemampuan yang lebih besar. [4]

Penggunaan metode penelitian algoritma Round Robin berguna untuk menganalisis proses *responses time* pada web *server* sehingga dapat diketahui peningkatan kinerja dari *Responses Time*, *Network Throughput* dan *Packet Loss*. parameter di *server* web. Berdasarkan uraian diatas dengan demikian judul penelitian ini adalah "Implementasi *Load Balancing* Web *Server* Menggunakan Haproxy dengan Algoritma Round Robin".

Berdasarkan konteks yang telah dijelaskan sebelumnya, rumusan permasalahan dalam penelitian ini adalah sebagai berikut:

1. Mengetahui cara mengatasi masalah umum saat *server* web kelebihan beban.
2. Implementasi *load balancing* web *server* menggunakan Haproxy dengan algoritma Round Robin.
3. Mengukur performa *load balacning* pada *server* web Memakai Haproxy dengan algoritma Round Robin pada tiga kriteria yaitu *responses time*, *throughput* dan *packet loss*.

Tujuan penelitian para peneliti adalah:

1. Dapat memastikan stabilitas *server* melalui pengelolaan yang efektif.
2. Mampu mengimplementasikan *load balancing server* web menggunakan Haproxy dengan algoritma Round Robin.
3. Dapat mengukur kinerja *load balancing* menggunakan Haproxy dengan algoritma Round Robin dengan parameter *responses time*, *throughput* dan *packet loss*.

Manfaat penelitian dari peneliti adalah:

1. Membantu mengurangi kemungkinan terjadinya *overload* pada web *server*.
2. Penggunaan algoritma Round Robin dan Haproxy untuk load balancing pada penelitian sebelumnya.
3. Optimalisasi penggunaan sumber daya *server*.
4. Hasil tes dapat digunakan sebagai bahan pengembangan untuk penelitian selanjutnya.

II. PEMBAHASAN

A. Load Balancer

Load balancer adalah perangkat yang mendistribusikan lalu lintas jaringan atau aplikasi saat melewati sekelompok *server*. *Load balancing* digunakan ketika jumlah pengguna melebihi kapasitas *server* dengan membagi beban lalu lintas secara merata melalui dua atau lebih saluran koneksi. [5]

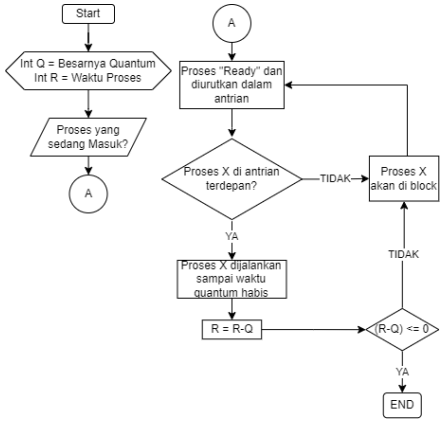
B. Web Server

Perangkat lunak yang menjadi fondasi atau dasar utama. World Wide Web (WWW) adalah Web *server* dibuat sekitar tahun 1980-an. Web Server berfungsi untuk menerima *request* dari *client* melalui program berbagai peramban web seperti Netscape, Navigator, Internet Explorer, Mozilla Firefox, dan beragam program peramban lainnya. [6]

C. Algoritma Round Robin

Salah satu metode yang umumnya digunakan untuk load balancing adalah Algoritma Round Robin. Algoritma ini memakai konsep berbagi waktu, dimana setiap aktivitas atau tugas diberikan waktu pemrosesan CPU tertentu yang dikenal juga sebagai waktu kuantum, yang berfungsi untuk mengatur batasan waktu waktu pemrosesan (kebanyakan antara 1 hingga 100 milidetik). Pada algoritma ini antrian diproses secara berulang (iteratif).

Masalah yang dihadapi oleh algoritma Round Robin adalah menentukan ukuran waktu kuantum yang tepat. Jika kuantum waktu yang dipilih terlalu kecil, beberapa proses mungkin tidak dapat diselesaikan dalam satu kuantum waktu. Kondisi ini menyebabkan banyak perubahan proses (*switching*), yang membutuhkan waktu CPU untuk ketika berpindah dari satu proses ke proses lainnya (waktu peralihan konteks). Sebaliknya, jika ukuran *time quantum* terlalu besar, algoritma Round Robin akan berperilaku serupa dengan algoritma First Come First Served. Kuantum waktu yang ideal adalah sekitar 80% dari total proses yang memiliki waktu *burst* CPU kurang dari 1 kali kuantum. [7]



Gambar 1. Diagram alir Algoritma Round Robin [8]

Penjelasan mengenai Q sebagai nilai kuantum dan R sebagai nilai waktu yang diberikan kepada proses dapat diturunkan dari flowchart yang diberikan. Ketika sebuah proses tiba, proses tersebut akan ditempatkan dalam antrian siap (*ready*). Jika antrian telah mencapai bagian depan, proses akan dieksekusi sampai waktu kuantum habis. Namun, jika antrian belum mencapai bagian depan, proses akan diblokir dan kembali ditempatkan dalam antrian. Setelah sebuah proses dalam antrian dieksekusi selama waktu kuantum, nilai R (waktu pemrosesan) akan dikurangi dengan nilai Q (waktu kuantum). Jika hasil pengurangan (R-Q) kurang dari atau sama dengan 0, berarti proses dalam antrian telah selesai. Namun, jika hasil pengurangan (R-Q) lebih besar dari 0, proses akan diblokir dan kembali ditempatkan dalam antrian. Contoh penjadwalan

menggunakan algoritma Round Robin digunakan untuk menjelaskan langkah-langkah dalam menghitung penjadwalan dengan metode Round Robin.

TABEL I
CONTOH PERHITUNGAN ALGORITMA ROUND ROBIN

Proses	***(Arrival Time)***	**(Waktu Pemrosesan)**
P1	0	7
P2	2	5
P3	6	4
P4	10	2

Dari tabel, terlihat terdapat 4 proses yang memiliki nilai kedatangan (*Arrival Time*) dan waktu pemrosesan (*Burst Time*) yang berbeda-beda. Kedatangan adalah waktu ketika proses mulai antri, sedangkan *Burst Time* adalah waktu yang dibutuhkan untuk mengeksekusi satu proses dalam antrian. Nilai kuantum adalah waktu yang diperlukan agar proses dapat berjalan sepenuhnya.

D. Domain Name System

DNS merupakan sebuah *server* yang bertugas mengonversi alamat IP menjadi alamat nama dan sebaliknya, yaitu dari alamat nama menjadi alamat IP. *Server* DNS memberikan identifikasi berupa nama komputer dalam jaringan komputer, sedangkan dalam konteks Internet, *server* DNS bertanggung jawab untuk menerjemahkan nama menjadi alamat IP.

Tidak mungkin mengelola sistem hierarki DNS yang terdiri dari satu *server* saja. Sistem DNS hierarkis ini terbagi menjadi beberapa zona, di mana setiap zona mencakup semua host dalam domain dan dapat berupa *level* domain apa pun. Dalam sebuah jaringan, setiap zona harus memiliki *server* namanya sendiri. *Server* nama ini bertanggung jawab untuk menerjemahkan antara alamat IP dan nama host di dalam zona tersebut. Jika Anda ingin membuat zona baru dalam domain Anda, kami dapat membuat *server* nama utama yang akan meneruskan permintaan pemetaan ke *server* nama resmi apa pun. Penggunaan lebih dari satu *server* nama bertujuan untuk membagi beban kerja dan berfungsi sebagai sistem cadangan untuk *server* nama tersebut.

Perbedaan antara *server* nama primer (*primary name server*) dan *server* nama sekunder (*secondary name server*) terletak pada bagaimana *server* nama tersebut. mendapatkan informasinya. *Server* nama utama mendapatkan informasinya dari data yang disimpan di *harddisk* nya, sedangkan *server* nama sekunder mendapatkan informasinya dari data salinan yang diterimanya dari *server* nama utama. Oleh karena itu, untuk setiap perubahan DNS, kita hanya perlu mengubah informasi *primary name server*. [9]

E. Haproxy

Haproxy merupakan solusi proxy yang gratis, memiliki kecepatan dan keandalan tinggi, serta menyediakan fitur load balancing dan proxy untuk aplikasi berbasis TCP dan HTTP. Haproxy cocok untuk menghadapi lalu lintas yang sangat tinggi dan digunakan oleh beberapa situs web paling populer di dunia. Kelebihan Haproxy adalah kemampuannya untuk dengan mudah dan tanpa risiko terintegrasi dengan arsitektur yang sudah ada. [1]

F. Nginx

Nginx, juga dikenal dengan sebutan "Engine-x", adalah sebuah web *server open source*. Selain berfungsi sebagai web *server*, Nginx juga dilengkapi dengan fitur-fitur seperti *reverse* proxy, HTTP cache, dan *load balancer*. Pengembangan Nginx dilakukan oleh Igor Sysoev dan dirilis ke publik pada bulan Oktober 2004. Pada saat peluncurannya, Igor meyakinkan masyarakat bahwa Nginx dapat menjadi solusi untuk masalah kinerja web *server* saat memiliki lebih dari 10.000 koneksi yang aktif secara bersamaan. [10]

G. Quality of Service (Qos)

Quality of Service didefinisikan dalam banyak hal, salah satunya adalah waktu respon *server*. Misalnya, situs web yang membedakan layanannya dalam hal waktu respon dari pelanggan, misalnya anggota platinum mendapatkan waktu respons yang lebih baik daripada pengguna emas atau perak. *Load balancing* dapat digunakan untuk membedakan anggota berdasarkan persyaratan paket.

1) ***Responses Time***

Salah satu faktor menarik terkait *load balancing* dan QoS adalah waktu respons saat mengakses sumber daya. Misalnya, berapa lama pengguna harus menunggu untuk melihat halaman web pada kunjungan pertama mereka? Waktu respon selalu dapat diterima sebagaimana mestinya. Beberapa penelitian telah menunjukkan bahwa situs web berkualitas buruk dapat mempengaruhi citra perusahaan, menguranginya, dan bahkan berdampak negatif terhadap persepsi keamanan pengguna. [11]

2) ***Packet Loss***

Packet loss merujuk pada jumlah paket IP yang tidak berhasil sampai ke tujuan selama proses transmisi. Salah satu faktor penyebabnya adalah adanya antrian yang melebihi kapasitas yang tersedia. [12]

TABEL II
KATEGORI *PACKET LOSS*

Kategori	*Packet loss* (%)	Nilai
Sangat Bagus	0-2	4
Bagus	3-14	3
Cukup	15-24	2
Buruk	>25	1

Persamaan perhitungan *Packet Loss*:

$$Packet\ Loss = \frac{(Paket\ data\ dikirim - Paket\ data\ diterima)}{Paket\ data\ yang\ dikirim} \times 100\ \%$$

3) ***Throughput***

Throughput merupakan ukuran kecepatan transfer data efektif yang diukur dalam kilobit per detik (kbps). *Throughput* dihitung dengan membagi jumlah total paket yang berhasil tiba ke tujuan selama interval waktu tertentu dengan durasi interval waktu tersebut.

Beberapa aplikasi, seperti media *streaming* langsung dan sistem interaktif, biasanya memerlukan *throughput* yang tinggi atau wajar, seperti 128 kbps untuk *streaming* audio. Dalam skenario seperti itu, waktu respon mungkin tidak memainkan peran penting selama penundaan yang terjadi dapat diterima. Oleh karena itu, perusahaan harus

merancang faktor kunci untuk dipertimbangkan saat merancang infrastruktur *load balancing* dengan ketersediaan tinggi. [11]

Throughput adalah kecepatan efektif transfer data yang diukur dalam kilobit per detik (kbps). Untuk menghitung *throughput*, jumlah total paket yang berhasil diterima di tujuan selama interval waktu tertentu dibagi dengan durasi interval waktu tersebut. [12]

TABEL III
KATEGORI THROUGHPUT

Kategori	*Throughput* (%)	Nilai
Sangat Bagus	100%	4
Bagus	75%	3
Cukup	50%	2
Buruk	<25&	1

Persamaan perhitungan *throughput*:

$$Throughput = \frac{\text{Paket data } diterima}{Lama\ Pengamatan}$$

$$Persentase\ (\%) = \frac{Nilai\ Throughput}{50} x100\%$$

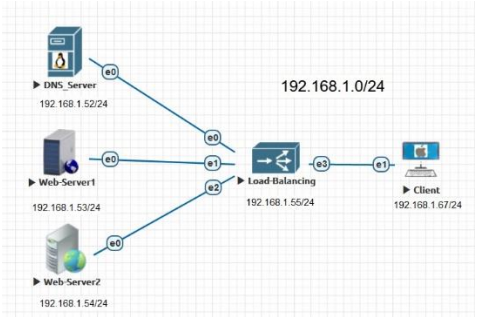
H. Apache JMeter

JMeter atau Apache JMeter adalah sebuah aplikasi *opensource* yang dibangun dengan menggunakan bahasa pemrograman Java, yang dapat digunakan untuk melakukan pengujian kinerja. Bagi para insinyur QA, JMeter dapat digunakan untuk melakukan pembebanan (*load testing*) pada aplikasi web, aplikasi FTP, dan pengujian server basis data.

Aplikasi Apache JMeter adalah web analisis yang menggunakan bahasa pemrograman Java berjalan di sistem operasi apa pun dan memungkinkan anda menguji stabilitas dan kinerja *server* web anda. Jenis pengujian yang dapat dijalankan di JMeter adalah web *test plan*, *load test*, dan *stress test*. [13]

III. METODE PENELITIAN

Metode penelitian adalah cara penulis memahami pembahasan masalah dan pemecahan masalah dalam suatu sistem. Jurnal ini menjelaskan metodologi penelitian perancangan sistem dan metode penelitian yang digunakan oleh penulis.

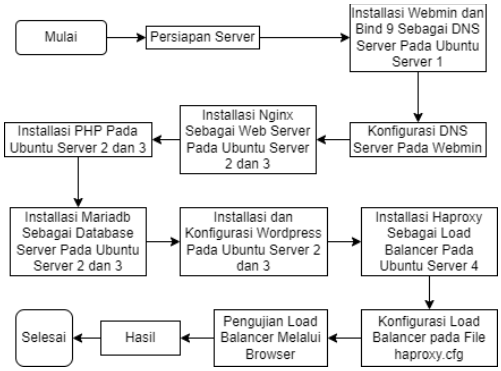


Gambar 2. Topologi yang akan digunakan

Dalam Gambar 2, ditampilkan skema jalur yang akan digunakan dalam sistem load balancing dengan menggunakan empat *server* berbasis OS Ubuntu, yang masing-masing telah diinstal dengan web dinamis. Selanjutnya, Haproxy akan digunakan sebagai solusi *load balancing* dan diinstal pada OS Ubuntu yang akan terhubung dengan klien.

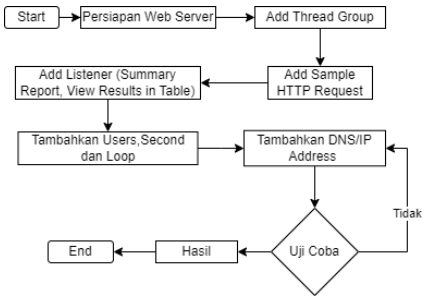
Gambar 3. *Flowchart* Kerangka Kerja Penelitian (*Framework*)

Pada Gambar 3 dilakukan tahapan Penelitian dimulai dengan mengidentifikasi masalah, dan dilanjutkan dengan persiapan OS dan tools yang digunakan, selanjutnya melakukan desain topologi pada EVE-NG kemudian dilanjutkan dengan instalasi seperti menginstal Ubuntu 22.04 kemudian Bind9 sebagai *server* DNS, Nginx sebagai *server* web dan Haproxy. sebagai penyeimbang beban. Setelah itu, konfigurasikan kedua *server* yang diuji dengan Haproxy dan siapkan pengujian *Quality of Service* (QoS) diuji menggunakan parameter *Responses Time*, *Packet Loss*, dan *Throughput*. Uji coba tersebut berhasil mencapai tahap evaluasi hasil yang diuji, kemudian penelitian tersebut diselesaikan.



Gambar 4. *Flowchart* Proses *Preprocessing Load Balancing*

Proses *preprocessing* merupakan proses persiapan untuk melakukan *load balancing* pada *server* web. Proses *preprocessing load balancing* terdiri dari persiapan *server*, instalasi DNS *server* pada Ubuntu *server* 1, konfigurasi DNS *server* melalui instalasi Webmin, Nginx, MariaDB dan WordPress pada Ubuntu *server* 2 dan 3. Instalasi Haproxy sebagai *load balancer* pada Ubuntu *server* 4, demikian juga sebagai pengujian load balancing menggunakan *server* IP Haproxy. Berikut *flowchart* proses *load balancing preprocessing* Informasi tersebut dapat ditemukan pada Gambar 4.



Gambar 5. Diagram Alir prosedur pengujian kualitas layanan (*Quality of Service*/QoS)

Prosedur pengujian kualitas layanan (*Quality of Service*/QoS) adalah proses pengujian web *server* yang telah dibuat dan mendapatkan hasil pada parameter *responses time*, *packet loss* dan *throughput*. Prosedur pengujian *Quality of Service* (QoS) terdiri dari menyiapkan web *server*, *add thread group*, *add sample* HTTP *request*, *add listener* (*summary report* dan *view results in table*), *add user*, *second* dan *loop*, menambahkan server DNS dan alamat IP. Berikut *flowchart* prosedur pengujian *Quality of Service* (QoS) dapat dilihat pada Gambar 5.

IV. HASIL DAN PEMBAHASAN

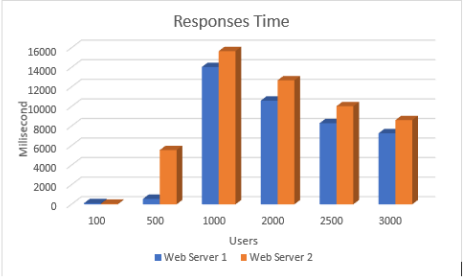
Jurnal ini akan menjelaskan lebih detail hasil pengujian implementasi *load balancing* web *server* menggunakan Haproxy dengan algoritma Round Robin. Setelah itu, hasilnya akan divalidasi melalui beberapa tahapan untuk mengecek apakah pelaksanaannya sesuai dengan tujuan yang diinginkan.

A. Perbandingan Hasil Web Server 1 dan 2

TABLE IV
HASIL PERBANDINGAN WEB *SERVER* 1 DAN 2

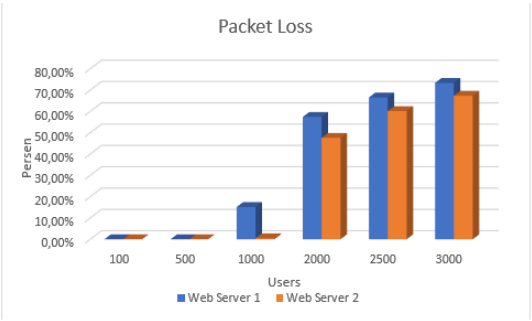
Users	*Web Server* 1			*Web Server* 2		
	Responses Time / Ms	*Packet Loss / %*	*Throughput / Sec*	*Responses Time / Ms*	*Packet Loss / %*	*Throughput / Sec*
100	118,6	0,00%	3,4	63,3	0,00%	3,4
500	558,7	0,00%	16,23	5545	0,00%	12,75
1000	14038,4	15,14%	17,48	15665,1	0,46%	12,46
2000	10598,9	57,41%	30,08	12679,2	47,57%	22,27
2500	8298,2	66,53%	40,24	10029	60,13%	29,46
3000	7264,1	73,34%	47,92	8609,4	67,33%	35,12

Hasil perbandingan permintaan pengguna menggunakan protokol HTTP setelah dilakukan pengujian disajikan pada Tabel 4. Tabel tersebut menunjukkan permintaan pengguna dari 100 hingga 3000 dan menghasilkan perbandingan antara web *server* 1 dan 2.



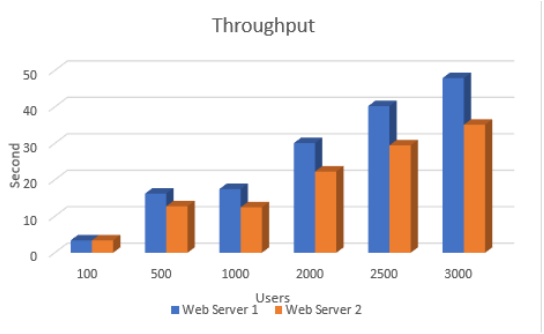
Gambar 6. Grafik *Responses Time* Perbandingan Web *Server* 1 dan 2

Berdasarkan Gambar 6 parameter *responses time* dibandingkan dengan web *server* 1 dan 2. Setelah dilakukan pengujian dengan 100 *user*, diketahui bahwa Waktu respon web *server* 2 jauh lebih rendah dibandingkan dengan web *server* 1. Namun, ketika dilakukan pengujian dengan 500 dan 1000 pengguna, hasilnya menunjukkan bahwa web *server* 1 memiliki waktu respon yang jauh lebih rendah dibandingkan dengan web *server* 2. Selama pengujian dengan jumlah *user* yang lebih besar yaitu 2500 dan 3000 *users*, hasil menunjukkan bahwa web *server* 2 kembali meunjukkan waktu respon yang lebih singkat. *responses time* dibandingkan dengan web *server* 1 Hal ini dikarenakan semakin banyak *user* yang mengakses *server* maka semakin berat beban yang harus ditangani oleh *server*, sehingga dapat mempengaruhi *responses time*.



Gambar 7. Grafik *Packet Loss* Perbandingan Web *Server* 1 dan 2

Berdasarkan Gambar 7 terlihat bahwa parameter *packet loss* membandingkan web *server* 1 dan 2. *Packet loss* pada web *server* 1 jauh lebih rendah dibandingkan web server 2. Penyebab meningkatnya *packet loss Network overload* terjadi ketika jaringan menerima terlalu banyak paket data dan tidak dapat menanganinya, sehingga beberapa paket data hilang.



Gambar 8. Grafik *Throughput* Perbandingan Web *Server* 1 dan 2

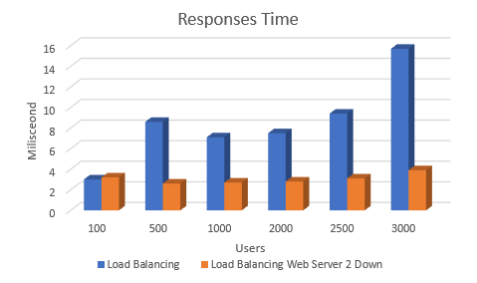
Berdasarkan Gambar 8, parameter *throughput* dapat dilihat dari perbandingan web *server* 1 dan 2, menunjukkan bahwa web *server* 2 jauh lebih rendah dibandingkan web *server* 1. Hal ini dikarenakan jika *throughput* lebih rendah maka jumlah permintaan atau data yang dapat dikirim atau diterima dalam waktu tertentu akan lebih sedikit, yang dapat menyebabkan keterlambatan pengiriman data atau respon pada aplikasi atau sistem tertentu.

B. Perbandingan Hasil Load Balancing dan Load Balancing Web Server 2 Down

TABLE V
HASIL PERBANDINGAN LOAD BALANCING DAN LOAD BALANCING WEB SERVER 2 DOWN

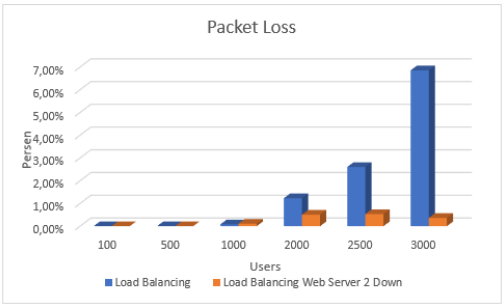
Load Balancing				*Load Balancing Web Server 2 Down*		
Users	*Response Time /ms*	*Packet Loss / %*	*Throughput / Sec*	*Responses Time /ms*	*Packet Loss / %*	*Throughput / Sec*
100	3	0,00%	3,4	3,2	0,00%	3,4
500	8,6	0,00%	7,36	2,6	0,00%	15,87
1000	7,1	0,08%	10,38	2,7	0,11%	18,3
2000	7,5	1,23%	22,85	2,8	0,51%	33,67
2500	9,4	2,60%	28,24	3,1	0,53%	42,52
3000	15,7	6,86%	32,03	3,9	0,36%	50,24

Hasil perbandingan permintaan pengguna menggunakan protokol HTTP setelah dilakukan pengujian disajikan pada Tabel 5. Tabel tersebut menunjukkan permintaan pengguna dari 100 hingga 3000 dan menghasilkan perbandingan antara *load balancing* dan *load balancing* web *server* 2 *down*.



Gambar 9. Grafik *Responses time* perbandingan *load balancing* dan *load balancing web server* 2 *down*

Berdasarkan Gambar 9 menunjukkan parameter *responses time* untuk perbandingan *load balancing* dan web *server* 2 *down load balancing*. Waktu respons pada penyeimbangan beban *server* web 2 jauh lebih rendah daripada *load balancing*. Performa *server* web akan lebih baik dengan tingkat waktu respon yang lebih rendah dan *throughput* yang lebih tinggi. [14] Jika *throughput* lebih kecil, maka lebih sedikit anjuran atau data yang dapat dikirim atau diterima dalam waktu tertentu, yang dapat menyebabkan keterlambatan transmisi data atau respon terhadap aplikasi atau sistem tertentu.



Gambar 10. Grafik *Packet loss* perbandingan *load balancing* dan *load balancing web server* 2 *down*

Berdasarkan Gambar 10 dapat dilihat parameter *packet loss*, perbandingan *load balancing* dan *down load balancing* web *server* 2. *Packet loss* pada web *server* 2 *down load balancing* jauh lebih rendah dibandingkan dengan *load balancing*. Penyebab meningkatnya kehilangan paket Kelebihan beban jaringan terjadi ketika jaringan menerima terlalu banyak paket data dan tidak mampu menanganinya, mengakibatkan beberapa paket data hilang.

Gambar 11. Grafik *Throughput* perbandingan *load balancing* dan *load balancing web server* 2 *down*

Berdasarkan Gambar 11 terlihat bahwa parameter *throughput* yang membandingkan *load balancing* dan *load balancing* web *server* 2 menunjukkan *load balancing* yang jauh lebih rendah dibandingkan dengan *down load balancing* web *server* 2. Hal ini dikarenakan jika *throughput* lebih rendah maka jumlah permintaan atau data yang dapat dikirim atau diterima dalam waktu tertentu akan lebih sedikit, yang dapat menyebabkan keterlambatan pengiriman atau respon data pada aplikasi atau sistem tertentu.

V. KESIMPULAN

Dengan mengacu pada hasil pengujian dan analisis yang telah dilakukan mengenai penggunaan algoritma Round Robin pada *load balancing* web *server*, Berikut adalah beberapa kesimpulan yang dapat diambil berdasarkan hasil pengujian dan analisis:

1. Untuk *load balancing* dengan Haproxy, berfungsi dengan baik. Menggunakan dua *server* web dengan Nginx. Haproxy merupakan solusi *load balancing* yang cukup populer dan dapat diintegrasikan dengan berbagai jenis web *server*.
2. Algoritma Round Robin pada HAProxy merupakan metode pembagian beban yang sederhana dan efektif, dimana setiap permintaan akan dialihkan secara bergantian ke setiap *server* yang tersedia. Meskipun algoritma Round Robin cukup sederhana, namun terdapat kelemahan yaitu tidak mempertimbangkan beban kerja sebenarnya pada masing-masing *server* sehingga kemungkinan terjadinya ketidakseimbangan beban kerja tetap ada.
3. Dalam implementasi *load balancing* perlu dilakukan *monitoring* secara berkala setiap *server* yang terhubung dengan Haproxy untuk memastikan semua *server* berjalan dengan baik dan tidak terjadi *overload* pada salah satu *server*.

UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada Bapak Candra Setiawan dan Bapak Fery Antony, kepada seluruh dosen sistem komputer di Universitas Indo Global Mandiri, Palembang. kepada agus, rizal dan diaz yang telah membantu selama pembuatan jurnal yang telah rela begadang semalaman membantu saya menyelesaikan jurnal ini.

REFERENSI

[1] S. Azizah and Asmunin, "Implementation of web server load balancing using Haproxy," 2017.

[2] B. Gofindo Malau, "Implementation of Internet Network Mikrotik Load Balancing at Pardamean Sibisa, Ajibata, Toba Samosir, North Sumatra," Journal of Computer Science and Technology (JCS-TECH), vol. 2, no. 1, pp. 20–29, May 2022, doi: 10.54840/jcstech.v2i1.23.

[3] G. Triono, T. Information, S. Tinggi, and T. Surabaya, "Implementation Of Load Balancing Using The Round Robin Algorithm In The Case Of New Student Registration Of First Junior High School Labschool Unesa Surabaya," 2015.

[4] M. Surya Pradana and A. Prapanca, "Analysis of the Load Balancing Performance of the Weighted Round Robin Algorithm in BPBD Infrastructure, East Java Province," 2019.

[5] A. Sumiati, P. Hari Trisnawan, and M. Ali Fauzi, "Implementation of Web Server Load Balancing with the Source IP Hash Algorithm on Software Defined Network (SDN)," 2020. [Online]. Available: http://j-ptiik.ub.ac.id

[6] Nurmiati Evy, "Analysis and Design of Web Servers on Mobile," 2012.

[7] M. Ismail, E. Wahyudi, and N. Iryani, "A Performance Analysis of F5 Big-Ip Ltm Load Balancing Using Least Connection and Round Robin Algorithms on Web Servers," 2022.

[8] Ansharullah Khairul, "Implementation of the Load Balancing System," 2016.

[9] A. Tedyyana, R. Kurniati, J. J. B. Alam, and S. Alam -Bengkalis - Riau, “Creating a Web Server Using a Dynamic Domain Name System on Dynamic IP”.2016. Available: www.namaanda.com',

[10] A. Y. Chandra, "Performance Analysis Between Apache & Nginx Web Server in Handling Client Requests," Journal of Systems and Informatics (JSI), vol. 14, no. 1, pp. 48–56, Nov. 2019, doi: 10.30864/jsi.v14i1.248.

[11] M. Rizky Pratama, "Load Balancing Performance Analysis With Round Robin Algorithm and Least Connection on a Web Server," Proceedings of Applied Science, vol. 1, no. 2, p. 1577, 2015.

[12] Apriza Uray Dini, Tjahjamooniarsih Neilcy, Imansyah Fitri, W Trias Pontia, and Kusumawardhani Eka, "Analysis of QOS (Quality Of Service) on Biznet Home Network Internet Services in Pontianak City," 2022.

[13] A. Dwi Astuti, A. Rahmatulloh, and M. F. Akbar Triadi, “Comparison of Haproxy and Nginx Load Balancing Performance.”

[14] R. Martha, Y. Firdaus, and K. A. Laksitowening, "Comparative Analysis of Response Time and Throughput on XML and DBMS as Data Storage Media," 2010.

Muhammad Arigoh Waluyo adalah mahasiswa di Universitas Indo Global Mandiri di Palembang. telah menyelesaikan masa studinya di program studi sistem komputer dalam waktu 3,5 tahun.

Fery Antony, dosen di Universitas Indo Global Mandiri Palembang dan mengajar di program studi sistem komputer. Ia lulus S1 dari Universitas Sriwijaya Palembang pada program studi teknik elektro, ia juga lulus S2 di Universitas Indonesia Depok pada program studi ilmu komputer. Saat ini sedang menjalani program studi S3 di Universitas Sriwijaya Palembang.

Candra Setiawan seorang dosen di Universitas Indo Global Mandiri Palembang dan mengajar di program studi sistem komputer. beliau lulus S1 di Universitas Sriwijaya Palembang program studi teknik elektro, beliau juga lulus pendidikan tingkat S2 di Institut Teknologi Bandung dengan program studi teknik elektro. bidang studi telekomunikasi.