Evaluasi kinerja load balancer menggunakan algoritma Round Robin dan Least Connection berbasis Docker Swarm

Tugas Akhir

diajukan untuk memenuhi salah satu syarat

memperoleh gelar sarjana

dari Program Studi Teknologi Informasi (Kampus Kota Surabaya)

Fakultas Informatika

Universitas Telkom

1202200399

Rey Dylanza

Program Studi Sarjana Teknologi Informasi (Kampus Kota Surabaya)

Fakultas Informatika

Universitas Telkom

Surabaya

2024

LEMBAR PENGESAHAN

Evaluasi kinerja load balancer menggunakan algoritma Round Robin dan Least Connection berbasis Docker Swarm

Load balancer performance evaluation using Round Robin and Least Connection algorithms based on Docker Swarm

NIM : 1202200399
Rey Dylanza

Tugas akhir ini telah diterima dan disahkan untuk memenuhi sebagian syarat memperoleh gelar pada Program Studi Sarjana Teknologi Informasi (Kampus Kota Surabaya)
Fakultas Informatika
Universitas Telkom

Surabaya, 16 Agustus 2024
Menyetujui

Pembimbing I,	Pembimbing II,
Oktavia Ayu Permata, S.T., M.T.	Kharisma Monika Dian Pertiwi, S.Kom., M.Kom
NIP. 19900006	NIP. 20950044

Ketua Program Studi
Sarjana Teknologi Informasi,

Bernadus Anggo Seno Aji, S.Kom., M.Kom.
NIP: 23929009

LEMBAR PERNYATAAN

Dengan ini saya, Rey Dylanza, menyatakan sesungguhnya bahwa Tugas Akhir saya dengan judul Evaluasi kinerja load balancer menggunakan algoritma Round Robin dan Least Connection berbasis Docker Swarm beserta dengan seluruh isinya adalah merupakan hasil karya sendiri, dan saya tidak melakukan penjiplakan yang tidak sesuai dengan etika keilmuan yang belaku dalam masyarakat keilmuan. Saya siap menanggung resiko/sanksi yang diberikan jika di kemudian hari ditemukan pelanggaran terhadap etika keilmuan dalam buku TA atau jika ada klaim dari pihak lain terhadap keaslian karya,

Surabaya, 16 Agustus 2024

Yang Menyatakan

Rey Dylanza

Evaluasi kinerja load balancer menggunakan algoritma Round Robin dan Least Connection berbasis Docker Swarm

Rey Dylanza[1], Oktavia Ayu Permata[2], Kharisma Monika Dian Pertiwi [3]

[1,2,3]Fakultas Informatika, Universitas Telkom, Surabaya
[1]reydylanza@students.telkomuniversity.ac.id, [2]oktapermata@telkomuniversity.ac.id, [3] kharismamonikadp@telkomuniversity.ac.id

Abstrak

Penggunaan teknologi clustering di lingkungan virtualisasi meningkatkan kapasitas dan kinerja server tanpa menambah server fisik. Dalam konteks ini, platform open-source Haproxy digunakan sebagai mesin web server load balancer yang memanfaatkan teknik virtualisasi Docker. Namun, penerapannya pada satu server dapat menimbulkan tantangan saat beban lalu lintas meningkat. Docker Swarm, yang mengelola cluster Docker di beberapa node, memungkinkan penerapan algoritma penyeimbangan beban, seperti Round Robin dan Least Connection, untuk meningkatkan kinerja load balancer. Penelitian ini mengevaluasi kinerja kedua algoritma tersebut dalam lingkungan virtualisasi lokal dengan mempertimbangkan kondisi riil web server. Pengujian membandingkan penggunaan sumber daya CPU dan RAM, throughput, dan response time, serta mempertimbangkan variasi beban kerja dan skenario lalu lintas. Hasil analisis menunjukkan bahwa algoritma Least Connection memiliki kinerja yang lebih baik dibandingkan dengan Round Robin, dengan skor akhir 0,4684 dibandingkan 0,4244. Efektivitas Least Connection dalam mendistribusikan beban kerja menjadikannya lebih disarankan untuk load balancing dengan beban tinggi, sehingga meningkatkan efisiensi sistem dan mengoptimalkan penggunaan sumber daya komputasi.

Kata kunci : Docker Swarm, Evaluasi Kinerja, Load Balancer, Virtualisasi, Web Server

Abstract

The use of clustering technology in virtualization environments increases server capacity and performance without adding physical servers. In this context, the open-source Haproxy platform is used as a web server load balancer engine that utilizes the Docker virtualization technique. However, its implementation on a single server can be challenging when the traffic load increases. Docker Swarm, which manages Docker clusters on multiple nodes, allows the implementation of load balancing algorithms, such as Round Robin and Least Connection, to improve load balancer performance. This study evaluates the performance of both algorithms in a local virtualization environment considering real-world web server conditions. The tests compare CPU and RAM resource usage, throughput, and response time, and consider variations in workload and traffic scenarios. The analysis results show that the Least Connection algorithm performs better than Round Robin, with a final score of 0.4684 compared to 0.4244. The effectiveness of Least Connection in distributing workloads makes it more recommended for load balancing with high loads, thereby increasing system efficiency and optimizing the use of computing resources.

Keywords: Docker Swarm, Performance Evaluation, Load Balancer, Virtualization, Web Server

1. Pendahuluan

a. Latar Belakang

Mengelola infrastruktur TI yang kuat dan andal merupakan tantangan bagi perusahaan yang menangani ratusan layanan setiap hari. *Server* adalah komputer yang menyediakan layanan melalui jaringan komputer, dan platform *web server* digunakan untuk menjalankan aplikasi [1]. Namun, peningkatan permintaan layanan dapat menyebabkan gangguan jika infrastruktur *server* tidak memadai. Untuk meningkatkan kapasitas dan kinerja *server* tanpa menambah *server* fisik, teknologi *clustering* di lingkungan virtualisasi dapat digunakan. Virtualisasi memungkinkan satu komputer berfungsi sebagai beberapa komputer virtual dengan arsitektur yang sama dengan komputer fisik [2].

Penelitian ini memanfaatkan platform *open-source Haproxy* sebagai mesin web server *load balancer*. *Haproxy* dipilih karena murah, cepat, andal, dan menawarkan ketersediaan tinggi, *failover*, serta *load balancing*. Selain itu, *Haproxy* dapat digunakan sebagai *proxy* untuk aplikasi berbasis *TCP* dan *HTTP* [3]. Untuk mendukung *load balancer*, teknik virtualisasi menggunakan *Docker* diterapkan. *Docker* memungkinkan pembangunan, pengujian, dan peluncuran aplikasi dalam *container* [4]. Namun,

penerapannya pada satu server dapat menimbulkan tantangan, terutama dengan peningkatan beban lalu lintas.

Docker Swarm, versi baru *Docker*, diciptakan untuk mengelola *cluster Docker* yang berjalan di beberapa node yang disebut *Swarm* [5]. Dalam konteks ini, penerapan algoritma penyeimbangan beban di *Docker Swarm* dapat meningkatkan kinerja *load balancer*. Algoritma penyeimbang beban yang umum digunakan adalah *Round Robin* dan *Least Connection*. Algoritma *Round Robin* mendistribusikan beban secara berurutan ke setiap node dalam interval waktu tertentu, sementara algoritma *Least Connection* membagi beban berdasarkan jumlah koneksi yang dibuat *node*. *Node* dengan koneksi paling sedikit akan menerima beban berikutnya [6].

Dalam lingkungan riil *web server*, dimana lingkungan riil *web server* merujuk pada situasi di mana *web server* digunakan dalam operasi yang sebenarnya, melayani permintaan dari pengguna nyata dengan berbagai tingkat beban dan kondisi jaringan yang berubah-ubah. Performa *load balancer* tidak hanya diukur berdasarkan satu parameter saja. Sebaliknya, berbagai parameter kinerja seperti penggunaan *CPU*, penggunaan memori, *throughput* jaringan, dan waktu respons harus diperhitungkan secara bersama-sama. Namun saat ini dalam evaluasi kinerja *load balancer* sering kali terbatas pada pengukuran parameter individual tanpa mempertimbangkan interaksi antara berbagai parameter tersebut. Hal ini dapat menyebabkan hasil yang kurang akurat dalam menggambarkan kinerja load balancer dalam kondisi dunia nyata.

Penelitian ini berfokus pada evaluasi kinerja dua metode *load balancer*, *Round Robin* dan *Least Connection*, dalam lingkungan *Docker Swarm* pada *virtual machine*. Pengujian membandingkan penggunaan sumber daya CPU dan RAM, *throughput*, dan *response time*, serta mempertimbangkan kondisi riil lingkungan *web server* dengan variasi beban kerja dan skenario lalu lintas, menggunakan pembobotan antara keempat parameter tersebut untuk menentukan algoritma terbaik yang paling efisien dan andal guna memberikan gambaran yang lebih jelas mengenai performa *load balancer* dalam situasi sebenarnya.

b. Topik dan Batasannya

Penelitian ini menggunakan beberapa batasan untuk melakukan *load testing*. *Apache Bench* digunakan untuk mensimulasikan dan mengukur beban pada *web server*, sedangkan *VirtualBox* pada *virtual machine* digunakan untuk sebagai lingkungan pengujian. *Web server* dan *load balancer* yang digunakan adalah *Haproxy*. Pengujian melibatkan empat node, terdiri dari satu *node manager* dan tiga *node worker*. Kinerja sistem dievaluasi berdasarkan metrik penggunaan CPU, penggunaan memori, *throughput*, dan waktu respons. Jumlah *request* yang diuji mencakup 10.000, 15.000, 20.000, dan 25.000 untuk menilai penanganan beban kerja yang berbeda. Selain itu, konfigurasi CPU dan RAM yang digunakan meliputi CPU dengan 1 *Core*, 2 *Core*, dan 3 *Core*, serta RAM sebesar 1 GB, 2 GB, 4 GB, dan 6 GB untuk melihat dampak berbagai sumber daya pada kinerja sistem. Penelitian ini juga menggunakan *Docker Compose* versi 3.8 untuk mengatur lingkungan kontainer. MySQL versi 8.0 digunakan sebagai basis data untuk WordPress versi 6.2.2. Selain itu, Haproxy versi 2.8.2 digunakan sebagai *load balancer* dalam pengujian ini.

c. Tujuan

Penelitian ini mensimulasikan penggunaan *Docker Swarm* dalam lingkungan virtualisasi desktop untuk menguji kinerja algoritma *load balancer*. Pengujian dilakukan dengan menggunakan dua algoritma *load balancing*, yaitu *Round Robin* dan *Least Connection*. Tujuan utama penelitian ini adalah untuk menentukan algoritma *load balancing* yang paling efektif berdasarkan beberapa parameter kinerja utama, seperti penggunaan *CPU*, penggunaan memori, *throughput* jaringan, dan waktu respons. Selain itu, penelitian ini juga mempertimbangkan kondisi riil lingkungan *web server* dengan menerapkan pembobotan antara keempat parameter tersebut, sehingga dapat memberikan gambaran yang lebih komprehensif tentang performa *load balancer* dalam situasi dunia nyata. Dengan mengumpulkan dan menganalisis data dari berbagai skenario pengujian, penelitian ini bertujuan untuk memberikan rekomendasi mengenai algoritma *load balancing* yang dapat memberikan kinerja optimal dalam pengelolaan beban kerja pada *Docker Swarm*. Hasil dari penelitian ini diharapkan dapat memberikan wawasan yang bermanfaat bagi pengembang dan administrator sistem dalam mengoptimalkan distribusi beban pada lingkungan virtualisasi.

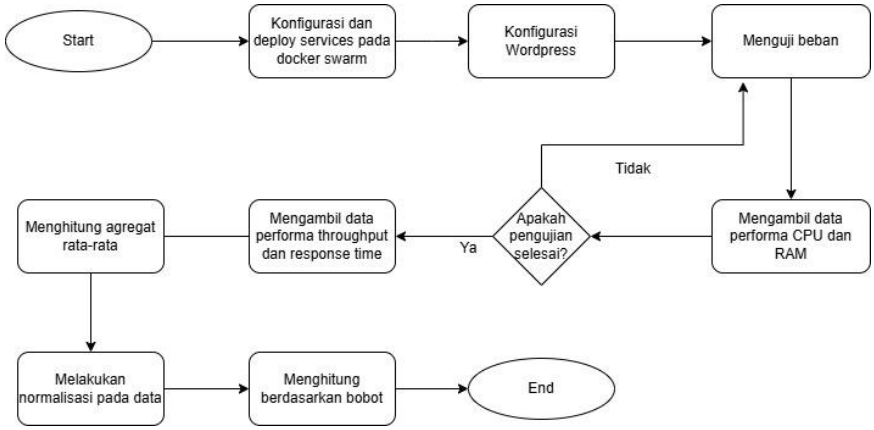
2. Studi Terkait

Penelitian yang sudah dilakukan dijadikan referensi sebagai pengembangan penelitian terkait implementasi *load balancing* pada *Docker Swarm* dengan algoritma *Round Robin* dan *Least Connection* sebagai teknik *load balancing*. Penelitian yang dijadikan referensi adalah sebagai berikut.

a. Penelitian berjudul "Pengujian Kinerja Load Balancing Web Server Menggunakan Nginx Riverse Proxy Berbasis OS Centos 7" oleh Muhamad Rafli, dkk. Diteliti pada tahun 2022 meneliti cara untuk mengatasi kelebihan permintaan pengunjung yang menyebabkan *server web down* atau lambat dengan menggunakan *NGINX load balancer* untuk mendistribusikan beban kerja. Algoritma yang digunakan adalah *Round Robin*, *Least Connection*, dan *IP Hash*. Pengujian dilakukan dengan *Jmeter*, *SNMP*, dan *software* lainnya untuk 20.000, 30.000, dan 40.000 pengguna. Hasil menunjukkan *NGINX load balancer* menjaga kinerja *web server* dengan algoritma *Round Robin* sebagai yang terbaik, dengan *respons time* 36 ms dan *throughput* 223.3/sec [7].
b. Penelitian berjudul "Load Balancing Server Web Berdasarkan Jumlah Koneksi Klien Pada Docker Swarm" oleh Dimas Setiawan Afis, dkk. Diteliti pada tahun 2019, penelitian ini mengatasi tantangan *server web* dengan arsitektur *single backend* yang menerima banyak permintaan data. Solusinya menggunakan *Docker Swarm* untuk membangun klaster *web server* dan mengelola kontainer dengan algoritma *load balancer Round Robin* dan *Least Connection*. Hasil menunjukkan algoritma *Least Connection* lebih unggul dengan *throughput* 17 Mbps pada 3000 dan 5000 permintaan, dibandingkan dengan *Round Robin* yang hanya mencapai 14-15 Mbps [8].
c. Penelitian berjudul "Implementasi Load Balancing Web Server Dengan Haproxy Menggunakan Algoritma Round Robin" oleh Muhammad Arigoh Waluyo, dkk. Diteliti pada tahun 2023 dimana dalam penelitian ini ketika banyak pengguna mengakses platform *online* secara bersamaan, server hosting dapat kelebihan beban. Masalah ini diatasi dengan teknik *load balancing*, seperti menggunakan *Haproxy*. Pengujian dilakukan dengan simulasi beban menggunakan *Apache JMeter* untuk 100 hingga 3000 pengguna bersamaan. Hasilnya menunjukkan bahwa *web server* berjalan normal dan mampu melayani akses pengguna. *Throughput* dan *response time* diukur, menunjukkan bahwa *Haproxy* efektif membagi beban antara server web, membuktikan kemampuannya dalam *load balancing* [9].
d. Penelitian berjudul "Analisis Unjuk Kerja Load Balancing Web Server Menggunakan Virtualisasi Berbasis Container Docker Swarm" oleh Arnanda Satria Wibawa. Diteliti pada tahun 2022 Penelitian ini membahas implementasi *load balancing* menggunakan *Zevenet* pada *web server Apache* dengan algoritma *Round Robin* dan *Least Connection*. Hasil menunjukkan *Round Robin* lebih optimal dengan *response time* lebih kecil, *CPU utilization* lebih besar, dan *memory usage* lebih kecil dibandingkan *Least Connection*. Pada 500 koneksi dengan 100 konkurensi, *response time* sebesar 6,28 detik, dan 30,19 detik pada 5000 koneksi dengan 1000 konkurensi. *CPU utilization* pada *node manager* 62,56%-78,55% dan *node worker* 61,99%-62,32%. *Memory usage* pada *node manager* 344,10 MB-602,44 MB dan *node worker* 319,72MB-586,70MB. Implementasi ini serupa dengan penggunaan *Docker Swarm* untuk meningkatkan kinerja *server* [10].
e. Penelitian berjudul "Analisis Perbandingan Algoritma Static Round-Robin dengan Least-Connection Terhadap Efisiensi Load Balancing pada Load Balancer Haproxy" oleh Hasta Triangga. Diteliti pada tahun 2019 membahas penyeimbangan beban menggunakan *Haproxy* dengan algoritma *Static round-robin* dan *Least-connection* pada 4 *server web Nginx*. Percobaan melibatkan 20 PC klien yang melakukan permintaan HTTP secara bersamaan. Hasil menunjukkan algoritma *Static round-robin* lebih efisien, dengan rata-rata penggunaan CPU selama 1, 5, dan 15 menit sebesar 0,1%, 0,25%, dan 1,15%, serta *throughput* 14,74 kbps, total *delay* 64,3 ms, dan *jitter* 11,1 ms. Sedangkan algoritma *Least-connection* menghasilkan penggunaan CPU 0,1%, 0,3%, dan 1,25%, *throughput* 14,66 kbps, total *delay* 350,3 ms, dan *jitter* 24,5 ms [11].

3. Sistem yang Dibangun

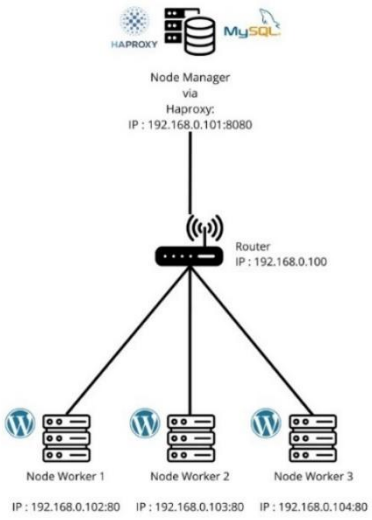
3.1 Alur penelitian



Gambar 1. Alur penelitian

Proses dimulai dengan persiapan awal, yaitu mengkonfigurasi dan *deploy* layanan ke dalam kluster *Docker Swarm*. Kemudian akan dilakukan konfigurasi pada *Wordpress* sebagai media untuk menguji nantinya. Setelah itu, beban koneksi dikirimkan ke layanan menggunakan *Apache Bench* untuk mensimulasikan kondisi nyata dengan banyak pengguna atau proses yang mengakses layanan secara bersamaan. Kemudian, data seperti penggunaan CPU dan RAM akan diambil. Pada titik ini, terdapat keputusan apakah pengujian sudah selesai atau belum. Jika belum, proses akan dilanjutkan dengan mengirimkan beban koneksi tambahan. Jika pengujian telah selesai, langkah berikutnya adalah mengumpulkan data performa yang mencakup metrik seperti waktu respons dan *throughput*. Proses ini diakhiri dengan analisis data dan pembuatan kesimpulan atau rekomendasi berdasarkan hasil pengujian.

3.2 Kerangka dan rancangan sistem



Gambar 2. Rancangan sistem load balancing

Pada gambar 2, menunjukkan arsitektur jaringan dari sistem yang terdiri dari 1 *node manager* yang mengatur kontainter, dan 3 *node worker* sebagai penyedia *services* yang terhubung melalui jaringan lokal menggunakan *router*. Pemilihan rancangan dengan 1 *node manager* dan 3 *node worker* dalam konfigurasi *Docker Swarm* diatas didasarkan pada beberapa pertimbangan. Pertama, dengan 1 *node manager* dapat meminimalkan kompleksitas dalam manajemen *cluster*, mengingat *node manager* bertanggung jawab atas pengaturan dan orkestrasi seluruh *cluster*. Satu *node manager* sudah cukup untuk lingkungan yang tidak terlalu besar atau untuk tahap pengujian, di mana redundansi dan toleransi terhadap kegagalan belum menjadi prioritas utama.

Pada bagian atas terdapat *Node Manager* yang bertindak sebagai pengelola node. Di dalam *Node Manager*, terdapat *HAProxy* yang memiliki alamat IP 192.168.0.101:8080. *HAProxy* berfungsi sebagai *load balancer* yang mengatur distribusi lalu lintas jaringan ke node-node pekerja di bawahnya. *Client* akan mengakses layanan web dari *IP* & *port HAProxy* tersebut. Selain *HAProxy*, didalam *node manager* juga terdapat *MySQL* sebagai *database* untuk *Wordpress*.

Router berada di tengah, menghubungkan *HAProxy* dengan tiga node pekerja, yaitu *Node Worker* 1, *Node Worker* 2, dan *Node Worker* 3. Setiap node pekerja memiliki alamat IP masing-masing: *Node Worker* 1 (192.168.0.102:80), *Node Worker* 2 (192.168.0.103:80), dan *Node Worker* 3 (192.168.0.104:80). Ketiga node pekerja ini masing-masing berisi *service Wordpress* serta menerima dan memproses permintaan yang didistribusikan oleh *HAProxy*.

3.3 Pengujian

Pengujian *load balancer* menggunakan *Docker Swarm* dilakukan dalam lingkungan virtual dengan beberapa node kontainer. Proses dimulai dengan membentuk *Docker Swarm* yang terdiri dari *node manager* menjalankan *service Haproxy* dan *database*. Konfigurasi *load balancer* dilakukan berdasarkan *IP address, port*, atau sumber daya yang digunakan. Dalam pengujian, *website* yang diuji memiliki ukuran sebesar 2,5 MB dan berbasis platform *WordPress*. Pengujian dilakukan menggunakan *software load testing* yang mensimulasikan *client* untuk mengirim sejumlah *request* ke aplikasi. Hasil pengujian ini digunakan untuk mengevaluasi kinerja *load balancer*. Pengujian ini juga mempertimbangkan kondisi riil lingkungan *web server* dengan variasi beban kerja dan skenario lalu lintas yang berbeda-beda, sehingga hasil yang diperoleh dapat merefleksikan performa *load balancer* dalam situasi yang lebih mendekati keadaan sebenarnya di lapangan., dan hasil pengujian yang dihasilkan selama pengujian berlangsung akan dicatat. Pada tabel 1 berikut adalah parameter yang akan dicatat selama pengujian berlangsung.

Tabel 1. Parameter Pengujian

No.	Parameter	Keterangan	Satuan
1	*CPU Usage*	Jumlah persentase *CPU* yang digunakan oleh sistem	%
2	*Memory Usage*	Jumlah kapasitas *RAM* yang digunakan oleh sistem	%
3	*Throughput*	Banyaknya data yang dapat ditransfer melalui jaringan dalam satu waktu tertentu	Kbps
4	*Respons Time*	Lama waktu yang dibutuhkan untuk mengirim dan menerima data	ms

Selama pengujian, *threshold response time* ditetapkan di bawah 150 ms sebagai batas maksimal yang diterima sebelum dianggap gagal. Angka ini dipilih untuk memastikan respon optimal, memberikan pengalaman pengguna yang baik, dan mempertahankan tingkat layanan yang diharapkan [12]. *Respons time* di bawah 150 ms menunjukkan kinerja yang diinginkan, sementara melebihi angka tersebut dianggap sebagai kegagalan kinerja web server, sesuai dengan detail pada Tabel 2.

Tabel 2. Threshold *Respons Time*

No.	Nilai *respons time*	Keterangan
1	< 50ms	Pengguna akan mengalami interaksi instan dan responsif tanpa hambatan apapun.
2	50ms – 100ms	Pengguna akan merasakan kecepatan dan kelancaran dalam interaksi.
3	100ms – 150ms	Meskipun masih dalam batas yang dapat diterima, pengguna mungkin mulai merasakan sedikit keterlambatan dalam interaksi.
4	> 150ms	Interaksi mungkin terasa kurang responsif, dan pengguna dapat mengalami frustrasi.

3.4 Skenario Kemampuan Host

Tabel 3. Skenario Pengujian Berdasarkan Batasan Kemampuan Host

No.	*Load Balancer*	*Worker*
1	CPU 1 *Core* & RAM 1, 2, 4 GB	CPU 1 *Core* & RAM 1, 2, 4, 6 GB
		CPU 2 *Core* & RAM 1, 2, 4, 6 GB
		CPU 3 *Core* & RAM 1, 2, 4, 6 GB
2	CPU 2 *Core* & RAM 1, 2, 4 GB	CPU 1 *Core* & RAM 1, 2, 4, 6 GB
		CPU 2 *Core* & RAM 1, 2, 4, 6 GB
		CPU 3 *Core* & RAM 1, 2, 4, 6 GB

Dari Tabel 3, dapat dilihat bahwa sistem memiliki kondisi yang berbeda pada *node load balancer/manager*, tergantung pada jumlah *core* CPU yang digunakan, yaitu antara 1 hingga 3 *core*. Setiap kondisi *core* CPU pada *node manager* juga memiliki variasi jumlah RAM, mulai dari 1 GB, 2 GB, 4 GB, hingga 6 GB. Selanjutnya, setiap kondisi pada *node manager* akan diuji dengan masing-masing kondisi pada *node worker*, yang juga memiliki kombinasi RAM yang bervariasi, mulai dari 1 GB, 2 GB, hingga 4 GB. Proses pengujian ini dilakukan untuk kedua kondisi *core* CPU pada *node worker*, yaitu 1 *core* dan 2 *core*.

3.5 Skenario Trafik Jaringan

Tabel 4. Skenario Pengujian Berdasarkan Trafik Jaringan

No.	*Request*	Keterangan
1	10000 *Request*	Mengirimkan 10000 *request*. Dengan concurrency level 100.
2	15000 *Request*	Mengirimkan 15000 *request*. Dengan concurrency level 150.
3	20000 *Request*	Mengirimkan 20000 *request*. Dengan concurrency level 200.
4	25000 *Request*	Mengirimkan 25000 *request*. Dengan concurrency level 250.

Untuk menguji kinerja sistem, empat skenario pengujian telah dirancang berdasarkan jumlah permintaan (*requests*) dan tingkat konkuren (*concurrency level*). Skenario pertama mengirimkan 10000 permintaan dengan tingkat konkuren 100, bertujuan mengamati kinerja sistem pada beban kerja ringan. Skenario kedua meningkatkan jumlah permintaan menjadi 15000 dengan tingkat konkuren 150 untuk mengukur respon sistem terhadap beban yang lebih tinggi. Pada skenario ketiga, jumlah permintaan dinaikkan menjadi 20000 dengan tingkat konkuren 200, menguji batas kemampuan sistem dengan beban kerja yang lebih berat. Skenario terakhir mengirimkan 25000 permintaan dengan tingkat konkuren 250, bertujuan mengidentifikasi batas maksimal kapasitas sistem di bawah beban yang sangat tinggi.

4. Evaluasi

4.1 Hasil Pengujian

Pengujian sistem yang dibangun untuk dapat membandingkan performa algoritma penyeimbang beban berbasis *Docker Swarm* telah memiliki hasil yang sudah diuji dengan menggunakan beberapa skenario yang telah dibuat pada tabel 3 dan tabel 4. Hasil pengujian untuk algoritma *Round Robin* tertera pada Tabel 5. Sedangkan untuk hasil pengujian pada algoritma *Least Connection* terdapat pada tabel 6.

Tabel 5. Hasil pengujian Round Robin

NO	Spek Node Manager	Spek Node Worker	Jumlah Request	CPU Node Manager (%)	CPU Node Worker 1 (%)	CPU Node Worker 2 (%)	CPU Node Worker 3 (%)	Memory Node Manager (%)	Memory Node Worker 1 (%)	Memory Node Worker 2 (%)	Memory Node Worker 3 (%)	Throughput (Kbps)	Response Time (ms)
1	CPU 1 core + RAM 1 GB	CPU 1 core + RAM 1 GB	10000	55	47	50	51	70	57	55	56	933	84
2	CPU 1 core + RAM 1 GB	CPU 1 core + RAM 1 GB	15000	56	48	57	53	72	64	68	64	879	87
3	CPU 1 core + RAM 1 GB	CPU 1 core + RAM 1 GB	20000	55	47	55	56	73	66	66	61	890	104
4	CPU 1 core + RAM 1 GB	CPU 1 core + RAM 1 GB	25000	54	57	54	53	75	61	62	66	746	131
285	CPU 2 core + RAM 4 GB	CPU 3 core + RAM 6 GB	10000	23	29	24	27	41	21	21	20	1564	73
286	CPU 2 core + RAM 4 GB	CPU 3 core + RAM 6 GB	15000	28	22	28	23	53	24	24	23	1342	79
287	CPU 2 core + RAM 4 GB	CPU 3 core + RAM 6 GB	20000	29	25	23	28	51	20	24	25	1284	86
288	CPU 2 core + RAM 4 GB	CPU 3 core + RAM 6 GB	25000	34	21	24	21	43	16	29	29	1080	91

Pengujian dengan algoritma *Round Robin* menunjukkan bahwa spesifikasi *hardware* sangat mempengaruhi performa sistem WordPress dalam menangani jumlah *request* besar. Pada spek CPU 1 *core* + RAM 1 GB, *throughput* turun drastis dan *response time* meningkat tajam hingga 131 ms pada 25,000 *request*, menandakan *bottleneck* signifikan. Sebaliknya, pada spek CPU 2 core + RAM 4 GB dan CPU 3 *core* + RAM 6 GB, sistem mampu menangani beban kerja lebih efektif, menjaga *throughput* di atas 1,000 Kbps dan *response time* tetap wajar (91 ms). Hasil ini menegaskan bahwa *hardware* yang lebih kuat meningkatkan efisiensi distribusi beban, menjaga performa stabil pada lalu lintas tinggi.

Tabel 6. Hasil pengujian Least Connection

NO	Spek Node Manager	Spek Node Worker	Jumlah Request	CPU Node Manager (%)	CPU Node Worker 1 (%)	CPU Node Worker 2 (%)	CPU Node Worker 3 (%)	Memory Node Manager (%)	Memory Node Worker 1 (%)	Memory Node Worker 2 (%)	Memory Node Worker 3 (%)	Throughput (Kbps)	Response Time (ms)
1	CPU 1 core + RAM 1 GB	CPU 1 core + RAM 1 GB	10000	73	56	52	53	60	69	68	67	1510	54
2	CPU 1 core + RAM 1 GB	CPU 1 core + RAM 1 GB	15000	74	54	57	51	62	61	67	66	1542	57
3	CPU 1 core + RAM 1 GB	CPU 1 core + RAM 1 GB	20000	77	59	53	56	64	61	68	68	1423	53
4	CPU 1 core + RAM 1 GB	CPU 1 core + RAM 1 GB	25000	71	54	52	59	68	68	69	69	1575	52
285	CPU 2 core + RAM 4 GB	CPU 3 core + RAM 6 GB	10000	34	22	23	29	34	13	21	18	1312	52
286	CPU 2 core + RAM 4 GB	CPU 3 core + RAM 6 GB	15000	33	26	30	25	33	15	23	24	1339	55
287	CPU 2 core + RAM 4 GB	CPU 3 core + RAM 6 GB	20000	36	25	27	27	37	14	28	26	1491	52
288	CPU 2 core + RAM 4 GB	CPU 3 core + RAM 6 GB	25000	38	29	25	28	40	16	30	29	1613	58

Pengujian algoritma *Least Connection* menunjukkan bahwa pada konfigurasi rendah (CPU 1 *core* + RAM 1 GB), CPU dan RAM mendekati kapasitas maksimal saat menangani 10.000-25.000 *request*, mengindikasikan potensi degradasi performa. Meskipun *throughput* stabil, waktu respons fluktuatif, menandakan efisiensi menurun. Sebaliknya, pada konfigurasi tinggi (CPU 2 *core* + RAM 4 GB untuk *node manager* dan CPU 3 *core* + RAM 6 GB untuk *node worker*), penggunaan CPU dan RAM lebih rendah dan stabil, dengan *throughput* yang lebih baik dan waktu respons yang konsisten, bahkan di bawah beban tinggi. Ini menunjukkan algoritma *Least Connection* lebih efisien pada sistem dengan sumber daya lebih besar.

Dalam menentukan algoritma terbaik maka diperlukan hasil agregat dari hasil pengujian berupa rata-rata pada setiap parameter yang telah ditentukan pada Tabel 1 yang tertera pada Tabel 7.

Tabel 7. Hasil agregat rata-rata

Algoritma	CPU	RAM	Throughput	Respons Time
Round Robin	36,89	42,65	1085.17	90.90
Least Connection	44,93	44,21	1456,32	55,20

4.2 Analisis Hasil Pengujian

Data agregat rata-rata kemudian akan dianalisis dengan melakukan normalisasi dengan menentukan nilai minimum dan nilai maksimum. Batas minimum penggunaan CPU dan memori ditetapkan pada 1% untuk memastikan sumber daya digunakan secara efektif, sementara batas maksimum 80% bertujuan mencegah *overload* yang dapat menyebabkan gangguan perangkat keras. Untuk *throughput*, batas maksimumnya tidak terbatas untuk memaksimalkan kapasitas transfer. Sedangkan untuk *response time*, batas maksimum 150 ms berdasarkan threshold tertinggi *response time* pada Tabel 2. Penentuan nilai minimum dan nilai maksimum seperti yang tertera pada tabel 8.

Tabel 8. Nilai Normalisasi

Parameter	Batas	Nilai	Satuan
CPU	Minimum	1	%
	Maksimum	80	%
Memory/ RAM	Minimum	1	%
	Maksimum	80	%
Throughput	Minimum	1	Kbps
	Maksimum	∞	Kbps
Response Time	Minimum	1	ms
	Maksimum	150	ms

Tabel 9. Bobot analisis

Parameter	Bobot
CPU	35%
Memory/RAM	20%
Throughput	10%
Response Time	35%

Dalam proses analisis, telah ditentukan pembobotan untuk setiap parameter pengujian yang akan digunakan. Persentase nilai bobot didasarkan pada kondisi riil lingkungan *web server* serta didasarkan pada keterkaitan masing-masing antara CPU dan RAM serta *Throughput* dan *Respons Time*. Dimana CPU memiliki pengaruh besar dan signifikan terhadap *respons time* dan *throughput*. Sebaliknya, RAM memiliki pengaruh sedikit terhadap keduanya [14]. Kemudian dari segi jaringan, *response time* lebih mempengaruhi pengalaman pengguna daripada nilai *throughput* [15]. Detail nilai pembobotan setiap parameter tertera pada Tabel 9.

```
Skor akhir Round Robin: 0.4244
Skor akhir Least Connection: 0.4686

Metode Least Connection menunjukkan kinerja yang lebih baik
```

Gambar 3. Hasil analisis

Berdasarkan Gambar 3, skor akhir yang diperoleh dari algoritma *Round Robin* adalah 0.4244, sedangkan algoritma *Least Connection* menghasilkan skor akhir yang lebih tinggi yaitu 0.4686. Berdasarkan hasil analisis ini, dapat disimpulkan bahwa algoritma *Least Connection* memiliki kinerja yang lebih baik dibandingkan dengan algoritma *Round Robin*. Hal ini menunjukkan bahwa *Least Connection* lebih efektif dalam mendistribusikan beban kerja, sehingga lebih disarankan untuk digunakan dalam implementasi *load balancing* dengan beban tinggi.

5. Kesimpulan

Pengujian menunjukkan bahwa spesifikasi *hardware* sangat mempengaruhi performa sistem saat menggunakan algoritma *Round Robin* dan *Least Connection*. Pada spesifikasi rendah (CPU 1 *core* + RAM 1 GB), sistem mengalami *bottleneck* signifikan dengan *throughput* rendah dan waktu respons meningkat drastis saat menangani 25,000 *request*. Sebaliknya, pada spesifikasi lebih tinggi (CPU 2 *core* + RAM 4 GB dan CPU 3 *core* + RAM 6 GB), kedua algoritma mempertahankan *throughput* di atas 1,000 Kbps dengan waktu respons yang wajar, namun *Least Connection* menunjukkan keunggulan dalam mendistribusikan beban kerja secara lebih efisien. Skor akhir yang diperoleh, yaitu 0,4686 untuk *Least Connection* dan 0,4244 untuk *Round Robin*, menegaskan bahwa *Least Connection* lebih efektif dalam menjaga stabilitas sistem, terutama di bawah lalu lintas berat. Penelitian selanjutnya disarankan untuk menambah jumlah node dalam sistem guna memperoleh gambaran yang lebih komprehensif mengenai performa algoritma dalam skala yang lebih besar dan mengeksplorasi algoritma load balancing lainnya untuk optimasi lebih lanjut.

Daftar Pustaka

[1] A. Y. Priyawan, “Implementasi Pembuatan Mail Server dan Webmail Pada Rumah Sakit Siti Khodijah Sepanjang Sidoarjo”, Surabaya, Universitas Dinamika, 2017.

[2] R. Umar, “Review Tentang Virtualisasi,” Jurnal Informatika, vol. 7, no. 2, p.775-784, 2013.

[3] A. Wirawan, R. Gatra, H. Hidayat, dan D. Prasetyawan, “Implementasi Load Balancing dengan HAProxy di Sistem Informasi Akademik UIN Sunan Kalijaga,” JISKA (Jurnal Informatika Sunan Kalijaga), vol. 9, no. 1. Al-Jamiah Research Centre, hlm. 39–49, Jan 25, 2024. doi: 10.14421/jiska.2024.9.1.39-49.

[4] S. E. Prasetyo, A. Wijaya, “Analisis Load Balancing Menggunakan Docker Swarm,” CoMBInES (Conference on Management, Business, Innovation, Education and Social Sciences), vol. 1, no. 1, p. 527-538, 2021.

[5] N. Nguyen and D. Bein, "Distributed MPI cluster with Docker Swarm mode," 2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 2017, pp. 1-7, doi: 10.1109/CCWC.2017.7868429.

[6] H. Nasser, T. Witono “Analisis Algoritma Round Robin, Least Connection, Dan Ratio Pada Load Balancing Menggunakan Opnet Modeler,” INFORMATIKA Vol. 12, No. 1, p.25-32, 2016.

[7] M. Rafli, I. Fitri, A. Andrianingsih, “Pengujian Kinerja Load Balancing Web Server Menggunakan Nginx Riverse Proxy Berbasis OS Centos 7”, Jurnal Teknik Informatika dan Sistem Informasi Vol. 9, No. 3, p.1824-1840 2022. https://doi.org/10.35957/jatisi.v9i3.2185

[8] D. S. Afis, M. Data, dan W. Yahya, “Load Balancing Server Web Berdasarkan Jumlah Koneksi Klien Pada Docker Swarm”, J-PTIIK, vol. 3, no. 1, hlm. 925–930, Jan 2019.

[9] M. A. Waluyo, F. Antony, dan C. Setiawan, “Implementasi Load Balancing Web Server Dengan Haproxy Menggunakan Algoritma Round Robin,” Journal of Intelligent Networks and IoT Global, vol. 1, no. 1. Universitas Indo Global Mandiri, hlm. 46–52, Jul 10, 2023. doi: 10.36982/jinig.v1i1.3074.

[10] A. S. Wibawa, “Analisis Unjuk Kerja Load Balancing Web Server Menggunakan Virtualisasi Berbasis Container Docker Swarm”, Purwokerto, Institut Teknologi Telkom Purwokerto, 2022.

[11] H. Triangga, I. Faisal, dan I. Lubis, “Analisis Perbandingan Algoritma Static Round-Robin dengan Least-Connection Terhadap Efisiensi Load Balancing pada Load Balancer Haproxy,” InfoTekJar (Jurnal Nasional Informatika dan Teknologi Jaringan), vol. 4, no. 1. Universitas Islam Sumatera Utara, hlm. 70–75, Sep 25, 2019. doi: 10.30743/infotekjar.v4i1.1688.

[12] T. Day, Z. Mailloux, J. McManus, “The Effects of Latency, Bandwidth, and Packet Loss on Cloud-Based Gaming Services”, Massachusetts, Worcester Polytechnic Institute, 2019.

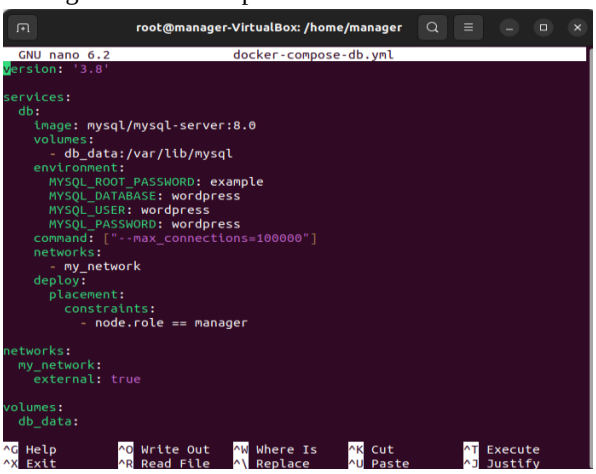
[13] T. T. Hanif, A. Adiwijaya, and S. Al-Faraby, "Analisis Churn Prediction Pada Data Pelanggan Pt. Telekomunikasi Menggunakan Underbagging Dan Logistic Regression," eProceedings of Engineering, vol. 4, no. 2, Aug. 2017.

[14] A. -P. Barzu, M. Carabas and N. Tapus, "Scalability of a Web Server: How Does Vertical Scalability Improve the Performance of a Server?," 2017 21st International Conference on Control Systems and Computer Science (CSCS), Bucharest, Romania, 2017, pp. 115-122, doi: 10.1109/CSCS.2017.22.

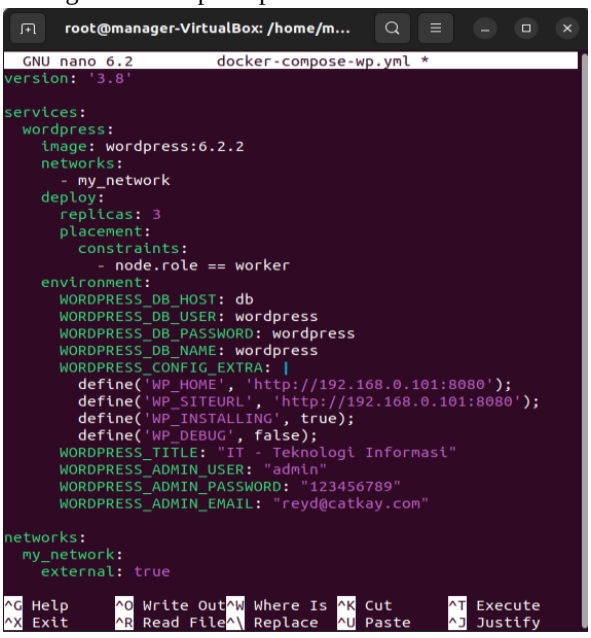
[15] N. J. Tochukwu and O. E. C. Mary, "Performance Evaluation of Web Servers using Response Time and Bandwidth," International Journal of Science and Engineering Applications, vol. 9, no. 12, pp. 133-138, 2020.

Lampiran

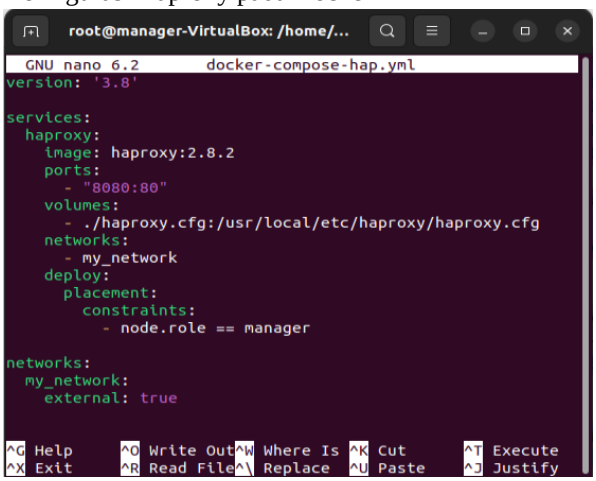
1. Konfigurasi Database pada docker



2. Konfigurasi Wordpress pada Docker



3. Konfigurasi Haproxy pada Docker



4. Konfigurasi load balancer Haproxy

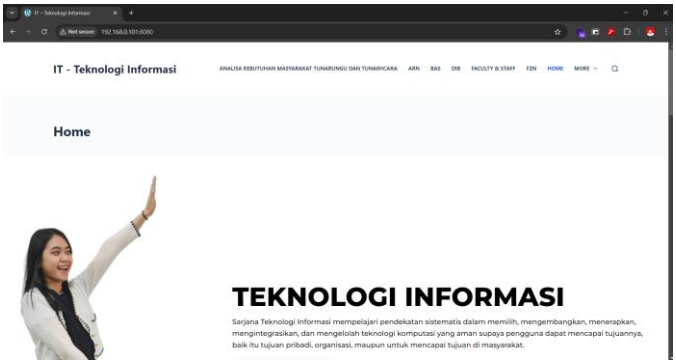
```
  GNU nano 6.2
global
    log stdout format raw local0

defaults
    log     global
    option  httplog
    option  dontlognull
    timeout connect 5000ms
    timeout client  50000ms
    timeout server  50000ms

frontend http_front
    bind *:80
    default_backend http_back

backend http_back
    balance leastconn
    server wordpress1 wordpress:80 check
    server wordpress2 wordpress:80 check
    server wordpress3 wordpress:80 check
```

5. Tampilan page wordpress untuk pengujian



6. Proses pengujian Apache Bench

```
root@manager-VirtualBox:/home/manager# ab -n 10000 -c 100 http://192.168.0.105:8080/
This is ApacheBench, Version 2.3 <$Revision: 1879490 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.0.105 (be patient)
Completed 1000 requests
Completed 2000 requests
Completed 3000 requests
Completed 4000 requests
Completed 5000 requests
Completed 6000 requests
Completed 7000 requests
Completed 8000 requests
Completed 9000 requests
Completed 10000 requests
Finished 10000 requests
```

7. Hasil pengujian Apache Bench dan mendapatkan data Throughput & Response Time

```
Server Software:        Apache/2.4.61
Server Hostname:        192.168.0.105
Server Port:            8080

Document Path:          /
Document Length:        101129 bytes

Concurrency Level:      100
Time taken for tests:   663.812 seconds
Complete requests:      10000
Failed requests:        0
Total transferred:      1015620000 bytes
HTML transferred:       1011290000 bytes
Requests per second:    15.06 [#/sec] (mean)
Time per request:       6638.117 [ms] (mean)
Time per request:       66.381 [ms] (mean, across all concurrent requests)
Transfer rate:          1494.12 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   1.4      0      18
Processing:    87 6597 4692.9   8524   20296
Waiting:       80 6086 4319.3   7904   17838
Total:         87 6597 4692.9   8524   20296

Percentage of the requests served within a certain time (ms)
  50%   8524
  66%   9374
  75%   9933
  80%  10328
  90%  11509
  95%  12519
  98%  13775
  99%  14735
 100%  20296 (longest request)
root@manager-VirtualBox:/home/manager#
```

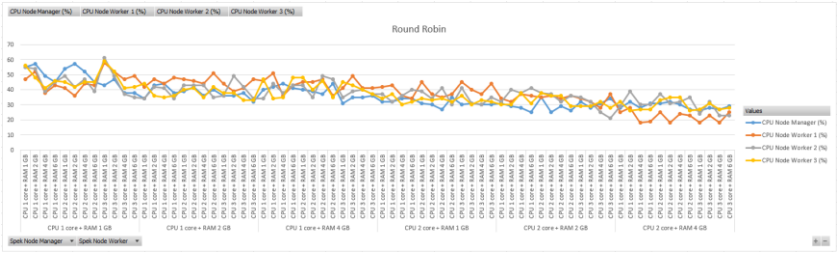
8. Mendapatkan data RAM

```
root@worker1-VirtualBox: /home/worker1
root@worker1-VirtualBox:/home/worker1# free | grep Mem | awk '{print $3/$2 * 100
.0 "%"}'
21,2873%
root@worker1-VirtualBox:/home/worker1#
```

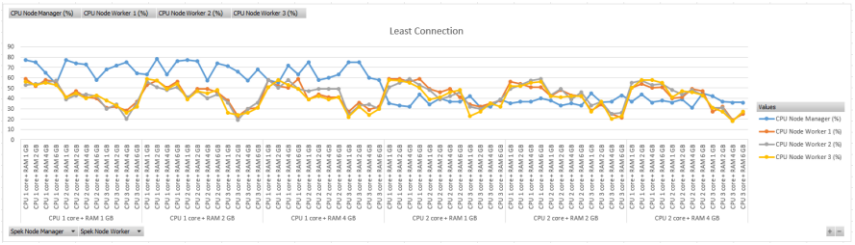
9. Data pemakaian CPU

```
Metric Name: CPU Load
Unit: %
Maximum: 100
Data Series: Guest Load
41 41 63 59 36 37 34 51 59 37 33 36 67 58 50 53 58 38 39 54 55 66 45 61 60 54 44 42 48 63 48 46 35 51
40 36 28 36 61 63 46 45 39 31 46 39 42 52 83 61 66 48 62 43 39 33 31 58 45 43 38 45 39 44 43 43 48 53
34 37 39 54 53 56 61 54 34 37 63 44 31 38 52 66 60 64 62 43 62 57 65 61 65 62 57 56 71 57 42 54 60 70
55 65 65 48 56 60 50 38 42 56 59 53 51 45 48 50 36 47
```

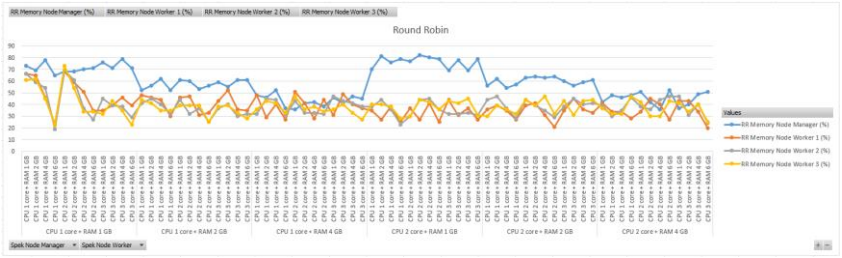
10. Grafik CPU Round Robin



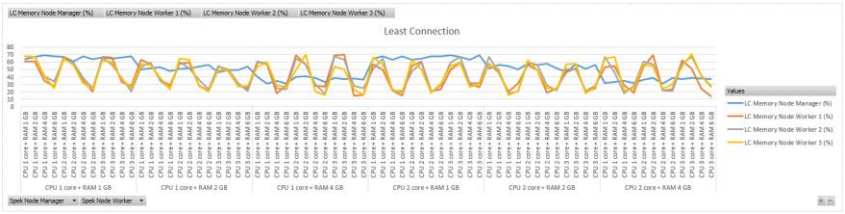
Grafik CPU Least Connection



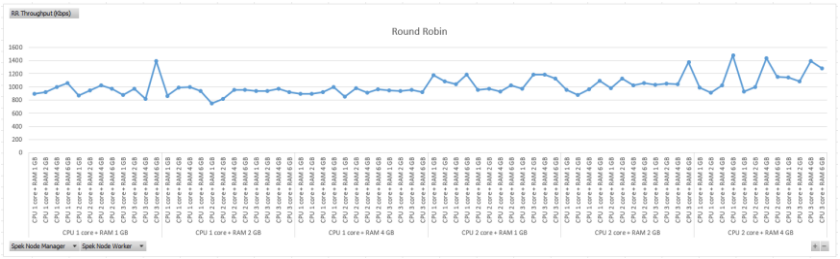
11. Grafik memori Round Robin



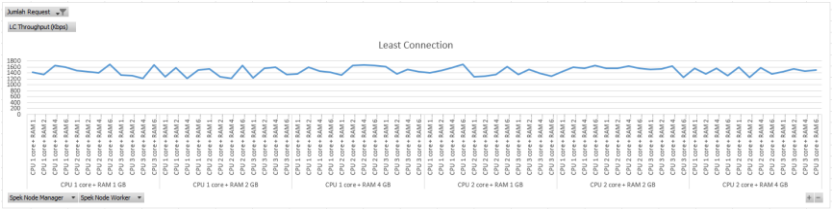
Grafik memori Least Connection



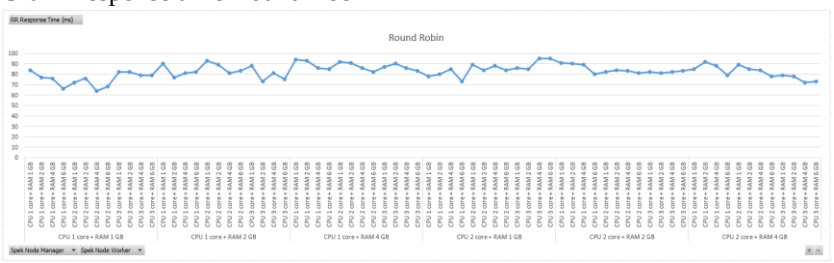
12. Grafik throughput Round Robin



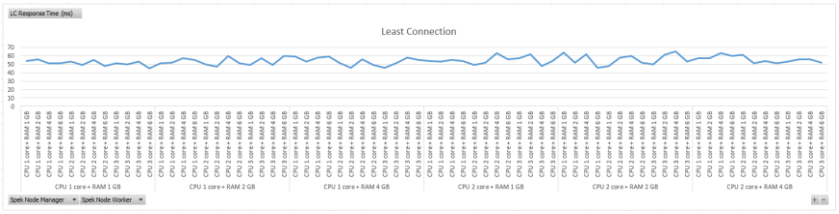
Grafik throughput Least Connection



13. Grafik response time Round Robin



Grafik response time Least Connection



14. Kode python

```
+ Kode + Teks

# Data pengujian untuk Round Robin
cpu_rr = 36.89
memory_rr = 42.65
throughput_rr = 1085.17
response_time_rr = 90.90

# Data pengujian untuk Least Connection
cpu_lc = 44.93
memory_lc = 44.21
throughput_lc = 1456.65
response_time_lc = 55.23

# Nilai maksimum dan minimum untuk normalisasi
max_cpu = 80 # Satuan %
min_cpu = 1 # Satuan %
max_memory = 80 # Satuan %
min_memory = 1 # Satuan %
max_throughput = float('inf')  # Menggunakan nilai tak hingga (infinity) dalam satuan Kbps
min_throughput = 1 # Satuan Kbps
max_response_time = 150 # Satuan ms
min_response_time = 1 # Satuan ms

# Fungsi untuk normalisasi
def normalize(value, min_value, max_value, inverse=False):
    normalized_value = (value - min_value) / (max_value - min_value)
    return 1 - normalized_value if inverse else normalized_value

# Normalisasi hasil pengujian Round Robin
cpu_rr_norm = normalize(cpu_rr, min_cpu, max_cpu, inverse=True)
memory_rr_norm = normalize(memory_rr, min_memory, max_memory, inverse=True)
throughput_rr_norm = normalize(throughput_rr, min_throughput, max_throughput)
response_time_rr_norm = normalize(response_time_rr, min_response_time, max_response_time, inverse=True)

# Normalisasi hasil pengujian Least Connection
cpu_lc_norm = normalize(cpu_lc, min_cpu, max_cpu, inverse=True)
memory_lc_norm = normalize(memory_lc, min_memory, max_memory, inverse=True)
throughput_lc_norm = normalize(throughput_lc, min_throughput, max_throughput)
response_time_lc_norm = normalize(response_time_lc, min_response_time, max_response_time, inverse=True)
```

```
+ Kode + Teks

# Bobot untuk masing-masing parameter
bobot_cpu = 0.35
bobot_memory = 0.20
bobot_throughput = 0.10
bobot_response_time = 0.35

# Hitung nilai yang dibobotkan untuk Round Robin
nilai_cpu_rr = cpu_rr_norm * bobot_cpu
nilai_memory_rr = memory_rr_norm * bobot_memory
nilai_throughput_rr = throughput_rr_norm * bobot_throughput
nilai_response_time_rr = response_time_rr_norm * bobot_response_time

# Jumlahkan hasil yang dibobotkan untuk Round Robin
skor_akhir_rr = nilai_cpu_rr + nilai_memory_rr + nilai_throughput_rr + nilai_response_time_rr

# Hitung nilai yang dibobotkan untuk Least Connection
nilai_cpu_lc = cpu_lc_norm * bobot_cpu
nilai_memory_lc = memory_lc_norm * bobot_memory
nilai_throughput_lc = throughput_lc_norm * bobot_throughput
nilai_response_time_lc = response_time_lc_norm * bobot_response_time

# Jumlahkan hasil yang dibobotkan untuk Least Connection
skor_akhir_lc = nilai_cpu_lc + nilai_memory_lc + nilai_throughput_lc + nilai_response_time_lc

# Bandingkan skor akhir
print(f"Skor akhir Round Robin: {skor_akhir_rr:.4f}")
print(f"Skor akhir Least Connection: {skor_akhir_lc:.4f}")

if skor_akhir_rr > skor_akhir_lc:
    print("\nMetode Round Robin menunjukkan kinerja yang lebih baik")
else:
    print("\nMetode Least Connection menunjukkan kinerja yang lebih baik")
```