MEKANISME DISTRIBUSI BEBAN PADA *LOAD BALANCING WEB SERVER* DENGAN ALGORITMA *LEAST CONNECTION* DAN *MULTI-AGENT SYSTEM* PADA LINGKUNGAN VIRTUAL

Web Server Load Balancing Mechanism with Least Connection Algorithm and Multi-Agent System

AFIYAH RIFKHA RAHMIKA
D082192010

PROGRAM STUDI S2 TEKNIK INFOMATIKA
DEPARTEMEN TEKNIK INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS HASANUDDIN
GOWA
2023

MEKANISME DISTRIBUSI BEBAN PADA *LOAD BALANCING WEB SERVER* DENGAN ALGORITMA *LEAST CONNECTION* DAN *MULTI-AGENT SYSTEM* PADA LINGKUNGAN VIRTUAL

AFIYAH RIFKHA RAHMIKA
D082192010

PROGRAM STUDI S2 TEKNIK INFOMATIKA
DEPARTEMEN TEKNIK INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS HASANUDDIN
GOWA
2023

PENGAJUAN TESIS

MEKANISME DISTRIBUSI BEBAN PADA *LOAD BALANCING WEB SERVER* DENGAN ALGORITMA *LEAST CONNECTION* DAN *MULTI AGENT SYSTEM* PADA LINGKUNGAN VIRTUAL

Tesis

Sebagai Salah Satu Syarat untuk Mencapai Gelar Magister

Program Studi Teknik Informatika

Disusun dan diajukan oleh

AFIYAH RIFKHA RAHMIKA

D082192010

Kepada

FAKULTAS TEKNIK

UNIVERSITAS HASANUDDIN

GOWA

2023

TESIS

MEKANISME DISTRIBUSI BEBAN PADA *LOAD BALANCING WEB SERVER* DENGAN ALGORITMA *LEAST CONNECTION* DAN *MULTI-AGENT SYSTEM* PADA LINGKUNGAN VIRTUAL

AFIYAH RIFKHA RAHMIKA
D082192010

Telah dipertahankan di hadapan Panitia Ujian yang dibentuk dalam rangka Penyelesaian Studi Program Magister Teknik Informatika Fakultas Teknik Universitas Hasanuddin
pada tanggal 16 Februari 2023
dan dinyatakan telah memenuhi syarat kelulusan.

Menyetujui,

Pembimbing Utama,

Dr-Eng. Zulkifli Tahir, S.T., M.Sc.
NIP. 19840403 201012 1 004

Pembimbing Pendamping,

Dr. Eng. Ady Wahyudi Paundu, S.T., M.T.
NIP. 19750313 200912 1 003

Dekan Fakultas Teknik
Universitas Hasanuddin,

Prof. Dr. Eng. Ir. Muhammad Isran Ramli, S.T., M.T.
NIP. 19730926 200012 1 002

Ketua Program Studi
S2 Teknik Informatika

Dr. Ir. Zahir Zainuddin, M.Sc.
NIP. 19640427 198910 1 002

PERNYATAAN KEASLIAN TESIS DAN PELIMPAHAN HAK CIPTA

Yang bertanda tangan di bawah ini:

Nama : Afiyah Rifkha Rahmika

Nomor Mahasiswa : D082192010

Program Studi : S2 Teknik Informatika

Dengan ini menyatakan bahwa, tesis berjudul "Mekanisme Distribusi Beban Pada *Load Balancing Web Server* dengan *Algoritma Least Connection* dan *Multi-Agent System* Pada Lingkungan Virtual" adalah karya saya dengan arahan dari komisi pembimbing (Dr.Eng. Zulkifli Tahir, S.T., M.Sc., dan Dr-Eng. Ady Wahyudi Paundu, S.T., M.T.,). Karya ilmiah ini belum diajukan dan tidak sedang diajukan dalam bentuk apa pun kepada perguruan tinggi mana pun. Sumber informasi yang berasa atau dikutip dari karya yang diterbitkan maupun tidak diterbitkan dari penulis lain telah disebutkan dalam teks dan dicantumkan dalam Daftar Pustaka tesis ini. Sebagian dari isi tesis ini telah dipublikasikan di Jurnal/Prosiding (CommIT (Communication and Information Technology) Journal) sebagai artikel dengan judul *"Web Server Load Balancing Mechanism with Least Connection Algorithm and Multi-Agent System"*.

Dengan ini saya limpahkan hak cipta dari karya tulis saya berupa tesis ini kepada Universitas Hasanuddin.

Gowa, 20 Februari 2023

Yang menyatakan

Afiyah Rifkha Rahmika

KATA PENGANTAR

Puji dan syukur penulis panjatkan kepada Allah SWT. karena berkat rahmat dan karunia-Nya sehingga tesis yang berjudul "**Mekanisme Distribusi Beban Pada *Load Balancing Web Server* Dengan Algoritma *Least Connection* dan *Multi-Agent System* Pada Lingkungan Virtual**" ini dapat diselesaikan sebagai salah satu syarat dalam menyelesaikan jenjang Strata-2 pada Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin.

Penulis menyadari bahwa dalam penyusunan dan penulisan laporan tesis ini tidak lepas dari bantuan, bimbingan serta dukungan dari berbagai pihak, dari masa perkuliahan sampai dengan masa penyusunan tesis. Oleh karena itu, penulis dengan senang hati menyampaikan terima kasih kepada:

1. Kedua Orang tua penulis, Bapak Alm. Ir. H. Bahrun Rafid dan Ibu Ir. Hj. Ramlawati Ahmad yang selalu menjadi motivasi terbesar dalam penyelesaian perkuliahan ini yang tidak pernah putus memberikan dukungan, doa, dan semangat serta selalu sabar dalam mendidik penulis sejak kecil;
2. Saudara kembar penulis, Afiyah Rifdha Rahmika, S.Ds yang dengan sangat sabar menemani dan memberikan semangat kepada penulis selama penyusunan tesis;
3. Bapak Dr.Eng. Zulkifli Tahir, S.T., M.Sc selaku pembimbing I dan Bapak Dr. Eng. Ady Wahyudi Paundu, ST., M.T. selaku pembimbing II yang telah memberikan waktu, tenaga, pikiran, dukungan moril maupun materil serta perhatian yang luar biasa untuk mengarahkan penulis dalam penyusunan tesis;
4. Bapak Dr. Eng. Muhammad Niswar, S.T., M.InfoTech. Bapak Dr. Amil Ahmad Ilham, S.T., M.IT. dan Bapak Ir. Zahir Zainuddin, M.Sc. selaku dosen penguji yang telah memberikan kritik dan saran yang membangun sehingga laporan tesis ini menjadi lebih baik;
5. Bapak Dr. Ir. Zahir Zainuddin, M.Sc selaku Ketua Program Studi Teknik Informatika Fakultas Teknik Universitas Hasanuddin yang telah memberikan motivasi, bimbingan, dan semangatnya selama masa perkuliahan penulis;

6. Para sahabat, teman-teman, kakak-kakak dan adik-adik di Laboratorium *Computer Based System* Pascasarjana UNHAS yang telah memberikan begitu banyak bantuan, keceriaan dan pengalaman manis selama proses perkuliahan;
7. Teman-teman Pascasarjana UNHAS Angkatan 1 atas dukungan dan semangat yang diberikan selama ini;
8. Ibu Yuanita serta segenap Staf Departemen Magister Teknik Informatika yang telah banyak membantu penulis selama pengurusan administrasi;
9. Orang-orang terkasih yang tidak sempat dituliskan oleh penulis;

Akhir kata, penulis berharap semoga Allah SWT. Senantiasa berkenan membalas segala kebaikan dari semua pihak yang telah banyak membantu. Semoga Tesis ini dapat memberikan manfaat bagi pengembangan ilmu. Aamiin ya Rabbal Alamin.

Gowa, 20 Februari 2023

Afiyah Rifkha Rahmika

ABSTRAK

Afiyah Rifkha Rahmika. Mekanisme Distribusi Beban Pada *Load Balancing Web Server* Dengan Algoritma *Least Connection* dan *Multi-Agent System* Pada Lingkungan Virtual. (dibimbing oleh **Zulkifli Tahir,** dan **Ady Wahyudi Paundu**).

Kebutuhan pengguna internet terhadap suatu website terus meningkat seiring dengan pengembangan layanan ini yang dilakukan secara terus menerus. Semakin banyak jumlah pengguna internet yang mengakses sebuah website, maka semakin berat beban web server dalam merespon permintaan tersebut sehingga dapat terjadi kondisi *overload* pada *web server*. Hal ini akan menurunkan kinerja website dalam memberikan kepuasan kepada pengguna internet, sehingga dibutuhkan sebuah mekanisme yang mampu menyeimbangkan beban dari web server agar website dapat bekerja secara optimal. Penelitian ini bertujuan untuk mengembangkan mekanisme *load balancing* berdasarkan kondisi *load* dan jumlah koneksi aktif terkecil dari *web server* menggunakan kombinasi algoritma *Least Connection* dan metode Multi-Agent System (LC-MAS). Agen yang diletakkan pada masing-masing *web server* memastikan kondisi *load* dibawah 80% kemudian algoritma menghitung jumlah koneksi yang sedang ditangani untuk selanjutnya dipilih menjadi *web server* yang merespon permintaan selanjutnya. Pengukuran kinerja dilakukan dengan menguji mekanisme LC-MAS dengan mekanisme yang hanya menggunakan algoritma *Least Connection* (LC) dengan mengirimkan sejumlah HTTP *request*. Hasil penelitian menunjukkan mekanisme *load balancing* LC-MAS memperoleh kinerja yang lebih baik dibandingkan dengan algoritma Least Connection (LC), dengan rata-rata waktu respon sebesar 1338.8 ms, 20.07 % error, dan nilai throughput sebesar 125 tps saat menangani 1500 HTTP *request*. Mekanisme *load balancing* dengan LC-MAS mampu mendistribusikan jumlah HTTP request secara lebih merata tanpa menimbulkan kondisi *idle* atau *overload*.

Kata Kunci: Load Balancing, Web Server, Least Connection, Multi-Agent System, Virtualisasi

ABSTRACT

Afiyah Rifkha Rahmika. Web Server Load Balancing Mechanism with Least Connection Algorithm and Multi-Agent System. (Supervised by **Zulkifli Tahir,** and **Ady Wahyudi Paundu**).

Demands for information over the internet become massively increased through the continuous expansion of web applications. Therefore, generating powerful and efficient server architecture for web servers is a must to satisfy internet users and avoid the system being overloaded. Implementing load balancing methods has proven to tackle the problems. A load balancer is needed to manage and distributes the workload equally among the servers to make the website performance better. The main contribution of this research focuses on developing a new mechanism for load balancing to distribute incoming HTTP request in web applications by combining the Least Connection algorithm and Multi-Agent System (LC-MAS). Proposed mechanism will distribute the request based on load condition and fewest number of active connections where existing research did not consider resources condition in developing load balancing mechanisms. This research was tested in two scenarios using 500, 1000, and 1500 requests. The performance of this proposed mechanism is measured through the values of Average Response Time, Throughput, and Error Percentage. The results show that the proposed mechanism (LC-MAS) distributes the workload more equally than LC with average response time for 1500 request is 1338.8 millisecond, 20.07% error, and 125 transactions per second. The LC-MAS makes the web application performance is much better when the request is increased. The LC-MAS helps in the utilization of system resources and improves system robustness.

Keywords: Load Balancing, Web Server, Least Connection, Multi-Agent System, Virtualization

DAFTAR ISI

Halaman

DAFTAR GAMBAR

Nomor Halaman

DAFTAR TABEL

Nomor Halaman

BAB I
PENDAHULUAN

1.1 Latar Belakang

Pada era digital ini penggunaan internet dikalangan masyarakat sudah menjadi hal yang biasa. Aplikasi internet seperti *website* terlibat secara luas dalam semua aspek kehidupan sehari-hari karena jutaan pengguna menggunakan internet untuk mengakses *website* dari manapun. Konten yang ditampilkan juga sangat beragam dan interaktif sehingga mendorong pengguna internet untuk mengakses *website* setiap saat. Pengguna ini meminta skalabilitas tinggi, ketersediaan, dan keandalan dalam memberikan respon cepat untuk aplikasi *website* yang mereka akses kapan saja[1].

Website seperti layanan *e-commerce* bahkan menerima jutaan koneksi dari pengguna internet secara simultan dalam satu hari karena suatu kondisi dimana *website* tersebut sering dikunjungi. Contohnya pada layanan *e-commerce* yang mengadakan promo besar-besaran di akhir tahun atau pada tanggal tertentu, sistem informasi akademik suatu universitas ketika pendaftaran mahasiswa baru atau saat *deadline* pengumpulan berkas calon mahasiswa, dan pengumpulan tugas mahasiswa melalui *e-learning*. Kondisi ini membuat permintaan akses dari pengguna internet yang harus ditangani oleh *website* tersebut semakin meningkat.

Web server menjadi bagian penting dari infrastruktur internet sebab sebuah aplikasi *website* hanya dapat berjalan di atas sebuah *web server*. Membangun arsitektur *web server* yang baik dan handal sangat penting untuk mengelola trafik dan *request* dalam jumlah yang besar guna melindungi bisnis dari risiko kegagalan *(down)* sistem, kesalahan pemrosesan pesan, kehilangan data transaksional, dan

penurunan performa. Untuk aplikasi *website* yang ramai, *down* sejenak merupakan masalah yang serius karena bisa menghilangkan banyak uang[2].

Semakin banyak jumlah pengguna yang mengakses suatu *web*site, maka semakin berat pula beban sebuah *web server* dalam menerima *request*. Banyaknya jumlah *request* pada suatu *web server* akan mengakibatkan terjadinya *overloading* bahkan kemungkinan *server* akan mengalami *down*. Disisi lain, sebuah *web server* yang handal seharusnya mampu menangani *request* dari pengguna yang cukup besar. Oleh karena itu dibutuhkan penyeimbang beban *server* yang mengatur dan membagi beban kerja *server* secara merata[3].

Teknologi yang dapat diterapkan untuk menyeimbangkan beban pada *server* yaitu teknologi *load balancing*. *Load balancing* adalah teknik untuk mendistribusikan beban trafik dari sebuah layanan pada sekumpulan *server* atau perangkat jaringan secara seimbang ketika ada permintaan dari pengguna. Dengan ini trafik dapat berjalan secara optimal, memaksimalkan *throughput*, memperkecil waktu respon, memanfaatkan *resources* dengan baik sehingga dapat meningkatkan efisiensi hasil yang dibuat oleh *server*. Selain itu jika ada satu *server* yang gagal, layanan tetap dapat berjalan menggunakan *server* yang masih ada karena terdapat banyak *server* yang bekerja untuk melayani *request*[4].

Proses penyeimbangan beban pada sistem *load balancing* memiliki metode dan algoritma tersendiri. Algoritma *load balancing* dibagi menjadi dua jenis yaitu statis dan dinamis. Algoritma statis membagi beban berdasarkan nilai awal yang ditetapkan sebelum *request* dieksekusi oleh *server*. Algoritma dinamis membagi beban berdasarkan kondisi *real* sebuah *server* setelah *request* dieksekusi. Terdapat beberapa parameter tertentu seperti waktu respon, *throughput*, *resources utilization*,

fault tolerance, skalabilitas, dll yang digunakan untuk analisis efisiensi algoritma *load balancing* yang diterapkan[5].

Penelitian [6] menganalisa performansi dari penerapan teknologi *load balancing* menggunakan algoritma *Least Time First Byte* dan *Multi Agent System* (LFB-MAS). Penelitian ini menggunakan kondisi *resources* dari *server backend* dan waktu respon tercepat terhadap *byte* pertama pada data yang menjadi prioritas dalam melakukan *load balancing*. Simulasi dilakukan dengan membandingkan hasil performansi dari algoritma *Weighted Least Connection* (WLC) pada penelitian sebelumnya. Algoritma LFB-MAS mendapatkan hasil yang lebih baik dibandingkan dengan WLC, dengan waktu respon 16.190 ms, *error* 0.00%, dan nilai *throughput* sebesar 43.0 request per second. Akan tetapi penulis menambahkan bahwa nilai *response time* dan *throughput* sistem membutuhkan peningkatan untuk menghasilkan kinerja sistem yang lebih baik dibandingkan sebelumnya.

Waktu respon yang diperoleh jauh lebih lama dibandingkan penelitian sebelumnya dengan algoritma WLC. Hal ini terjadi karena sistem harus mengecek waktu respon *byte* pertama dari setiap *request* yang masuk. LFB-MAS memang menghasilkan waktu respon yang lebih lama dikarenakan LFB-MAS memproses lebih banyak *request* yang dapat diproses sehingga harus menunggu sampai pemrosesan *request* tersebut selesai, sedangkan LFB tanpa agen dan WLC mendapatkan waktu respon lebih singkat karena ketika *request* yang banyak tidak dapat diproses, pemrosesan *request* langsung selesai sehingga nilai error yang dihasilkan lebih tinggi.

Seperti yang diketahui bahwa kebutuhan aplikasi *web* saat ini masih membutuhkan performansi yang lebih tinggi, tidak hanya memberikan waktu respon yang cepat ke pengguna namun sebuah aplikasi *web* dengan performa yang sangat baik juga harus mampu memproses semua permintaan dari pengguna guna meminimalkan nilai *error* dalam menangani permintaan sehingga dibutuhkan mekanisme *load balancing* yang lebih baik.

Berdasarkan permasalahan pada penelitian [6], maka pada penelitian ini peneliti mengusulkan sebuah mekanisme pendistribusian beban pada aplikasi *web* dengan menerapkan teknologi *load balancing* menggunakan algoritma adalah *Least Connection* dan *Multi Agent System* (LC-MAS) untuk meningkatkan kinerja aplikasi web yang lebih baik. Pemilihan algoritma bertujuan untuk menghilangkan proses dimana sistem harus mengecek waktu respon *byte* pertama dari setiap *request* yang masuk yang menyebabkan waktu respon dari *backend server* semakin meningkat.

Algoritma LC mendistribusikan beban kerja secara efektif ke seluruh *server* sesuai dengan kapasitasnya. *Server* dengan kapasitas yang lebih kuat akan memenuhi *request* lebih cepat sehingga pada saat tertentu kemungkinan memiliki jumlah koneksi yang masih diproses jauh lebih sedikit daripada *server* dengan kapasitas yang rendah. Akan tetapi ketika hanya mempertimbangkan jumlah koneksi aktif, penggunaan sumber daya (*resources*) dari setiap *server* tidak dapat dimaksimalkan sehingga dibutuhkan metode untuk memantau kondisi *resources* dari setiap *server*. Penelitian [7] memperlihatkan hasil metrik kinerja beberapa algoritma dalam melakukan *load balancing*. Algoritma LC unggul dalam metrik

performance, resource utilization, dan *throughput* akan tetapi kurang baik pada metik *resources utilization*.

Pada beberapa algoritma seperti algoritma WLC dan Fixed Weighting, bobot beban yang harus ditangani *server* telah ditentukan sebelumnya sedangkan kondisi *real* yang sering terjadi adalah spesifikasi *server* tidak merata sehingga dapat terjadi ketidakseimbangan beban antara *server* dengan spesifikasi rendah dan *server* dengan spesifikasi tinggi. Pemberian bobot pada *server* tidak dapat dikira-kira dalam artian tidak diketahui kapan *server* dengan spesifikasi rendah sedang menangani permintaan yang besar dan kapan *server* dengan spesifikasi tinggi sedang menangani permintaan yang kecil. Agar penggunaan *resources* lebih efisien, maka metode MAS ditambahkan untuk mengecek kondisi setiap *backend server* secara berkala.

Pada umumnya *load balancing* tidak menyediakan metode apapun untuk memonitoring kondisi *resources server*. *Load balancing* dapat diterapkan dengan menggunakan metode *Multi Agent Sistem* (MAS). Metode ini menggunakaan agen-agen yang memiliki kemampuan khusus untuk terus-menerus melakukan tugas yang sebelumnya diberikan dalam suatu lingkungan yang khusus. Agen dalam sistem ini dapat berpindah dari satu *node* menuju *node* yang lainnya secara bebas sesuai dengan tugas yang telah diberikan. Agen bergerak ini dapat diprogram untuk memonitor keadaan *resources* dari *server*. Para agen berkomunikasi melalui *node* dalam kelompok jaringan dalam melakukan tugasnya untuk mendapatkan informasi *resources*. Dengan integrasi metode ini, proses untuk mendapatkan informasi menjadi lebh cepat dan efisien[8].

Penelitian ini juga mengusulkan desain arsitektur mekanisme *load balancing* yang dapat diterapkan pada lingkungan virtual dengan mengatur beban lalu lintas pada jalur koneksi sebuah *node* (mesin virtual/vm). Seperti yang kita ketahui bahwa layanan internet saat ini rata-rata telah berpindah dari manual ke lingkungan virtual. Hal ini bertujuan untuk memudahkan proses pembangunan sistem serta meminimalisir biaya yang harus dikeluarkan ketika harus menyewa perangkat fisik. Lingkungan virtual dapat memudahkan dalam mensetup lingkungan sistem yang diharapkan. Dalam arsitektur ini terdapat satu vm sebagai *server load balancer* dan *server* utama untuk menyimpan informasi dari para agen dan beberapa vm sebagai *web server*. Agen ditempatkan di semua vm dengan tugas yang berbeda. *Agen server* utama bertugas meminta informasi *resources* dari vm yang berperan sebagai *web server*. Informasi ini kemudian dijadikan sebagai acuan untuk *server load balancer* memutuskan kondisi *web server* layak atau tidak untuk menerima *request* berdasarkan parameter yang diatur. Parameter tersebut adalah nilai ambang (*threshold*) dari CPU dan Memori sebesar 80%. Ketika *resources* tidak melebihi *threshold* maka dikategorikan **normal** dan dapat menerima *request*. Ketika *resources* melebihi *threshold* maka dikategorikan ***overload*** dan tidak dapat menerima *request*. Setelah mengecek kondisi *web server*, maka algoritma *least connection* akan menghitung jumlah koneksi aktif terkecil yang sedang ditangani dan *request* tersebut dialihkan ke *web server* yang memenuhi kedua syarat ini. Penelitian ini menggunakan dua skenario pengujian yaitu membandingkan antara *load balancing* menggunakan algoritma *least connection* tanpa *agen* dan *load balancing* menggunakan algoritma *least connection* dan *multi agent system*.

Mekanisme yang disulkan kemudian diuji untuk memperoleh analisa performasi *load balancing* dengan algoritma LC-MAS. Hasil pengujian dari setiap skenario akan dijadikan sebagai bahan analisa kuantitatif dengan mengambil nilai dari beberapa parameter berupa waktu respon, *throughput* (*request/*s), jumlah error, dan *resources utilization*. Penelitian ini berfokus pada aspek efisiensi dari mekanisme yang diusulkan sehingga diharapkan dapat memberikan performansi yang lebih baik dibandingkan dengan metode dan algoritma *load balancing* yang telah diuji pada penelitian sebelumnya dan mampu menjadi referensi baru dalam menerapkan teknologi *load balancing*.

1.2 Rumusan Masalah

Berdasarkan pada latar belakang di atas, maka rumusan masalah pada penelitian ini yaitu

1. Bagaimana merancang dan mengembangkan mekanisme *load balancing* pada *web server* di lingkungan virtual dalam memenuhi kebutuhan dari website yang sampai saat ini masih membutuhkan kinerja yang lebih tinggi?
2. Bagaimana kinerja mekanisme distribusi *load balancing* pada *web server* di lingkungan virtual yang telah dirancang dan dikembangkan?
3. Bagaimana pengaruh kinerja mekanisme distribusi *load balancing* pada *web server* di lingkungan virtual ketika ditambahkan metode Multi-Agent System?

1.3 Tujuan Penelitian

Berdasarkan rumusan masalah maka tujuan yang akan dicapai dari penelitian ini:

1. Merancang dan mengembangkan mekanisme *load balancing* pada *web server* di lingkungan virtual dengan kombinasi algoritma *Least Connection* dan metode Multi-Agent System untuk memenuhi kebutuhan dari *website* yang masih membutuhkan kinerja yang lebih tinggi.
2. Melakukan pengujian dan analisis terhadap kinerja mekanisme distribusi *load balancing* pada *web server* di lingkungan virtual yang telah dirancang dan dikembangkan.
3. Mengetahui pengaruh penambahan metode Multi-Agent System berdasarkan kinerja mekanisme distribusi *load balancing*.

1.4 Manfaat Penelitian

Penelitian ini memberikan manfaat berupa pengetahuan (*knowledge*) tentang mekanisme dalam mengelola jumlah request yang besar pada aplikasi *web* dengan teknologi *load balancing*. Pengetahuan ini dapat dijadikan sebagai referensi bagi organisasi atau perusahaan dalam meningkatkan keandalan sistem aplikasi *web* mereka. Hasil analisa kinerja yang dilakukan secara komprehensif dan akurat memberikan manfaat bagi akademisi sebagai literatur ketika akan meneliti pada topik serupa. Manfaat lain dari penelitian ini yaitu tersedianya rekomendasi rancangan mekanisme pendistribusian beban menggunakan teknologi *load balancing* pada aplikasi *web* dengan beberapa skenario.

1.5 Batasan Masalah

Adapun batasan masalah dalam penelitian ini yaitu:

1. Mekanisme *load balancing* berfokus pada pendistribusian permintaan atau *request* pada *website* yang diterapkan di lingkungan virtualisasi.

2. Parameter pengujian untuk menentukan kinerja dari mekanisme yang dikembangkan hanya waktu respon, *throughput*, dan persentasi error yang dihasilkan.
3. *Website* yang digunakan dibangun menggunakan CMS Wordpress dengan konten sederhana.
4. Skenario pengujian dibantu dengan tools Apache Jmeter untuk menciptakan jumlah *request*.

1.6 Sistematika Penulisan

Adapun sistematika penulisan pada penelitian ini yaitu:

Bab I Pendahuluan

Bab ini berisi penjelasan tentang latar belakang yang menjabarkan alasan dilakukannya penelitian terkait *load balancing* berdasarkan peluang penelitian dan uraian penelitian awal tentang *load balancing* yang dilakukan, terkait rumusan masalah, tujuan, manfaat, ruang lingkup serta sistematika penulisan penelitian dibahas pada bagian ini.

Bab II Tinjauan Pustaka

Bab ini berisi penjelasan tentang landasan teori yang digunakan dalam penelitian seperti manajemen virtualisasi, teknologi *load balancing*, algoritma *least connection,* metode *Multi-Agent System,* aplikasi website, web server, dan beberapa landasan teori lainnya. Diuraikan pula tentang tinjauan pustaka yang merupakan penjelasan tentang hasil-hasil penelitian sebelumnya yang berkaitan dengan penelitian yang dilakukan. Dalam bab ini juga diuraikan tentang kerangka pemikiran yang merupakan penjelasan tentang kerangka berpikir untuk

memecahkan masalah yang sedang diteliti, termasuk menguraikan objek penelitian serta state of the art dari beberapa penelitian terkait.

Bab III Metodologi Penelitian

Bab ini berisi tentang tahapan penelitian, waktu dan lokasi penelitian, instrumen penelitian, tahap persiapan, gambaran umum sistem, struktur dataset, skenario pengujian.

Bab IV Hasil dan Pembahasan

Bab ini berisi penjabaran hasil penelitian berdasarkan teknik evaluasi kinerja sistem yang digunakan. Pada bagian ini hasil dan pembahasan terbagi atas dua yaitu berdasarkan hasil perancangan mekanisme *load balancing* dan kinerja mekanisme yang dikembangkan. Hasil analisis kinerja mekanisme dirangkum dalam bentuk tabel, grafik dan gambar.

Bab V Kesimpulan dan Saran

Bab V berisi kesimpulan terhadap hasil yang didapatkan dalam penelitian ini, yang merujuk pada rumusan masalah dan saran pengembangan dari penelitian ini untuk menyempurnakan kekurangan-kekurangan atau capaian-capaian yang belum tercapai pada penelitian ini, sehingga kedepannya penelitian yang dilakukan dapat dikembangkan dan bisa memperoleh hasil yang jauh lebih baik.

BAB II
TINJAUAN PUSTKA

II.1. Penelitian Terdahulu

2.1 Virtualisasi

Virtualisasi adalah abstraksi sumber daya komputer dan memisahkan lapisan perangkat lunak dari lapisan perangkat keras serta mengisolasi aplikasi yang sedang berjalan dari perangkat keras yang digunakan. Pengertian virtualisasi dalam lingkungan IT secara mendasar adalah melakukan isolasi terhadap satu sumber daya komputasi dengan yang lainnya. Dengan memisahkan *layer-layer* yang berbeda dalam *logic stack* memungkinkan fleksibilitas yang lebih tinggi karena tidak diperlukan lagi konfigurasi tiap elemen untuk dapat berkerja bersama-sama. Virtualisasi memberikan fleksibilitas yang lebih baik untuk penyediaan sumber daya IT dibandingkan dengan penyediaan di lingkungan non-virtual karena dapat membantu mengoptimalkan pemanfaatan sumber daya dan mengirimkan sumber daya dengan lebih efisien. Pengumpulan sumber daya didukung dalam virtualisasi, di mana sumber daya dapat dikelola secara terpusat untuk meningkatkan fleksibilitas dengan mendukung perubahan dinamis dalam kebutuhan bisnis[9].

Menurut [10], virtualisasi dilakukan dengan menambahkan lapisan virtualisasi yang terdiri dari hypervisor dan monitor mesin virtual yang ditempatkan diantara perangkat keras dan sistem operasi yang menggunakannya. Menggunakan lapisan ini untuk virtualisasi sistem, satu mesin komputer dapat menjalankan beberapa sistem operasi secara bersamaan dalam mesin virtual, di mana sumber daya komputer seperti CPU, memori, dan penyimpanan secara dinamis dipartisi dan dibagi antara mesin virtual yang berbeda.

Pengenalan teknologi virtualisasi terutama berfokus pada tiga bidang: *Hardware Virtualization, Presentation Virtualization, dan Application Virtualization* [11].

a) *Hardware Virtualization*: Konsep dasar dari virtualisasi perangkat keras sangat sederhana yaitu gunakan perangkat lunak untuk membuat mesin virtual (*virtual machine* / VM) yang "meniru" komputer fisik. Dengan menyediakan beberapa VM sekaligus memungkinkan menjalankan beberapa sistem operasi secara bersamaan pada satu mesin fisik. Ketika digunakan pada mesin klien, pendekatan ini sering disebut virtualisasi desktop, sedangkan menggunakannya pada sistem *server* dikenal sebagai virtualisasi *server*.

b) *Presentation Virtualization*: Pendekatan ini memungkinkan pembuatan sesi virtual, masing-masing berinteraksi dengan sistem desktop jarak jauh. Aplikasi yang dijalankan dalam sesi tersebut mengandalkan virtualisasi presentasi untuk memproyeksikan antarmuka pengguna mereka dari jarak jauh. Setiap sesi mungkin hanya menjalankan satu aplikasi, atau mungkin menyajikan penggunanya dengan desktop lengkap yang menawarkan beberapa aplikasi. Dalam kedua kasus tersebut, beberapa sesi virtual dapat menggunakan salinan aplikasi yang sama.

c) *Application Virtualization*: Pendekatan ini memungkinkan pengguna untuk mengakses dan menggunakan aplikasi dari komputer yang terpisah dari komputer tempat aplikasi diinstal. Cara paling umum untuk memvirtualisasikan aplikasi adalah pendekatan berbasis *server*. Artinya, administrator IT menerapkan aplikasi jarak jauh dari *server* di dalam pusat

data organisasi atau melalui layanan *hosting*. Admin IT kemudian menggunakan perangkat lunak virtualisasi aplikasi untuk mengirimkan aplikasi ke *desktop* pengguna atau perangkat lain yang terhubung. Pengguna dapat mengakses dan menggunakan aplikasi seolah-olah itu dipasang secara lokal di komputer mereka, dan tindakan pengguna disampaikan kembali ke *server* untuk dieksekusi.

Penerapan virtualisasi *server* memberikan banyak keuntungan diantaranya, penghematan biaya karena terjadi penurunan anggaran yang dikeluarkan untuk pembelian *server* baru, daya listrik, perangkat pendingin, serta biaya *service* dan *maintenance*, mulai ulang otomatis mesin virtual jika terjadi kegagalan perangkat keras *server* host lengkap (*zero downtime maintenance*), *load balancing*, manajemen penyimpanan, isolasi antar mesin virtual berarti infrastruktur yang mendasari dapat dibagikan antara kelompok yang berbeda (*collabortion*).

2.2 Load Balancing

Load balancing adalah teknik untuk mendistribusikan beban trafik terhadap sebuah layanan yang ada pada sekumpulan *server* atau perangkat jaringan ketika ada permintaan dari pemakai. Dengan *load balancing* trafik dapat berjalan secara optimal, memaksimalkan *throughput* dan memperkecil waktu tanggap dan menghindari *overload* pada salah satu *server*. *Load balancing* dapat diterapkan pada lingkungan fisik dan lingkungan virtual.

Berikut beberapa definisi dari *load balancing*:

a) Menurut [12], *load balancing* adalah metode jaringan komputer untuk mendistribusikan beban kerja di berbagai sumber daya komputasi, seperti komputer, *cluster* komputer, tautan jaringan, unit pemrosesan pusat, atau

drive disk. Teknik load balancing bertujuan untuk mengoptimalkan penggunaan sumber daya, memaksimalkan throughput, meminimalkan waktu respons, dan menghindari *overload* salah satu sumber daya. Menggunakan beberapa komponen dengan *load balancing* sebagai ganti satu komponen dapat meningkatkan keandalan melalui redundansi.

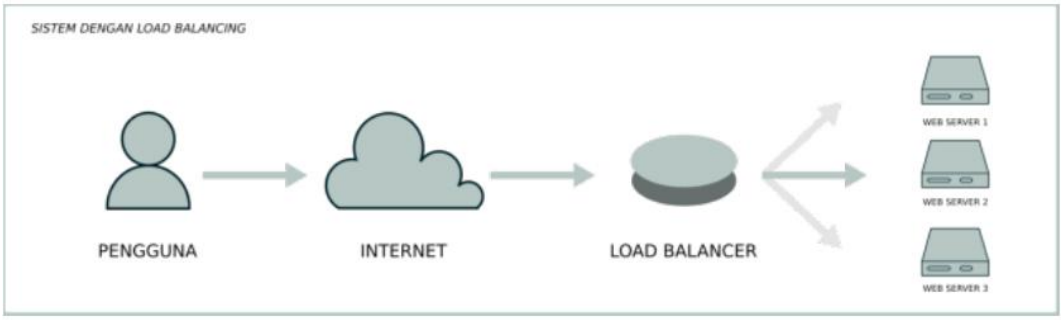
b) Menurut [13], *load balancing* adalah teknik untuk mendistribusikan beban trafik pada dua atau lebih jalur koneksi secara seimbang, agar trafik dapat berjalan optimal, memaksimalkan *throughput*, memperkecil waktu tanggap dan menghindari *overload* pada salah satu jalur koneksi. *Load balancing* digunakan pada saat sebuah *server* telah memiliki jumlah user yang telah melebihi maksimal kapasitasnya.

c) Menurut [14], *load balancing* merupakan teknik yang dapat mendistribusikan beban kerja, jaringan, atau lalu lintas aplikasi merata di beberapa *server*. *Load balancing* mampu meningkatkan fungsi sistem dengan memungkinkan peningkatan keandalan dan kinerja karena menghilangkan terjadinya satu titik kegagalan.

d) Menurut [15], *load balancing* adalah proses membagi beban total ke masing-masing *node* dari sistem terdistribusi untuk melakukan pemanfaatan sumber daya secara lebih cepat dan efisien dengan menghindari situasi dimana beberapa *node* banyak dimuat sementara *node* lain melakukan sedikit pekerjaan. *Load balancing* memastikan semua *node* dalam jaringan akan bekerja kurang lebih sama.

Gambar 1 merupakan gambaran tentang layanan *website* yang menggunakan sistem *load balancing*. *Load balancer* akan mengatur trafik yang datang dan

menentukan *web server* mana yang akan melayani trafik tersebut berdasarkan algoritma yang digunakan.



Gambar 1 Proses Load Balancing

Algoritma *load balancing* berkonsentrasi pada meminimalkan konsumsi sumber daya, meningkatkan skalabilitas, menghindari kemacetan (*bottleneck*), serta mengurangi penyediaan sumber daya yang berlebih. Algoritma *load balancing* pada dasarnya ada 2 jenis tergantung dari metode implementasi yang digunakan [16]:

a) Algoritma static *load balancing* merupakan algoritma *load balancing* yang tidak bergantung pada status sistem saat ini. Pada tahap awal, sebelum permintaan masuk ke *server*, telah diputuskan bahwa di*server* mana permintaan akan dieksekusi. Beberapa contoh dari algoritma statik *load balancing* yaitu Min-min, Round-Robin, OLB dan Max-Min.

b) Algoritma dinamik *load balancing* bekerja dengan cara penyeimbang beban menganalisis statistik beban saat ini dari setiap *server* yang tersedia dan mengeksekusi permintaan pada *server* yang sesuai. Beberapa contoh dari algoritma dinamik *load balancing* yaitu Least Connection, Honey Bee Foraging, Ant Colony Optimization dan Throttled.

Menurut [7] beberapa metrik kualitatif yang dianggap penting untuk *load balancing* dalam mengukur performa algoritma yang digunakan adalah:

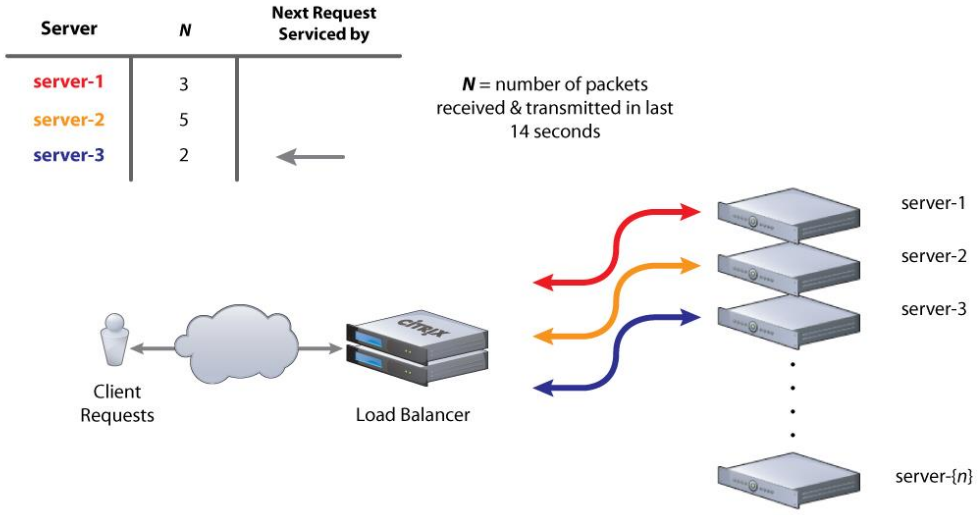
a) Throughput: Jumlah total tugas yang telah menyelesaikan eksekusi disebut *throughput*. *Throughput* yang tinggi diperlukan untuk kinerja sistem yang lebih baik.
b) Fault Tolerant: Kemampuan algoritme untuk bekerja dengan benar dan seragam bahkan dalam kondisi kegagalan di sembarang node dalam sistem.
c) Migration Time: Waktu yang dibutuhkan dalam migrasi atau transfer tugas dari satu mesin ke mesin lain dalam sistem.
d) Response Time: Waktu minimum yang diperlukan sistem terdistribusi yang menjalankan algoritme penyeimbangan beban tertentu untuk merespons.
e) Resource Utilization: Sejauh mana sumber daya sistem digunakan.
f) Scalability: Kemampuan untuk menyeimbangkan beban sistem dengan jumlah simpul yang terbatas. Metrik ini harus ditingkatkan.
g) Performance: Mewakili efektivitas sistem setelah melakukan load balancing. Jika semua di atas parameter terpenuhi secara optimal maka akan sangat tinggi meningkatkan kinerja sistem.

2.3 Algoritma Least Connection (LC)

Algoritma *load balancing least connection* bekerja dengan mengarahkan *request* menuju *server* yang paling sedikit memiliki koneksi yang sedang tersambung dengan terus mengupdate catatan koneksi tersambung secara *real time*, sehingga dapat diketahui *server* mana yang sedang menangani banyak *request* dan *server* mana yang tidak. Algoritma dapat bekerja dengan baik pada *web server* dengan spesifikasi *hardware* yang sama.

Penjadwalan ini termasuk salah satu algoritma penjadwalan dinamik, karena memerlukan perhitungan koneksi aktif untuk masing-masing *real server* secara

dinamik. Metode penjadwalan ini baik digunakan untuk melancarkan pendistribusian ketika *request* yang datang sangat banyak. Ilustrasi pendistribusian beban *request* dalam algoritma ini dapat dilihat pada Gambar II.2.



Gambar 2 Distribusi Beban Pada Algoritma Least Connection

Ilustrasi diatas menggambarkan *cluster server* yang saling terkoneksi untuk mengelola permintaan atau *request* dari pengguna internet yang mengakses sebuah layanan. *Server* tersebut menangani jumlah permintaan yang dikirim dan diterima sebanyak (N) dalam waktu 14 detik terakhir. *Server-1* sedang menangani 3 *request*, *Server-2* sedang menangani 5 *request*, *Server-3* sedang menangani 2 *request*. Algoritma LC kemudian menghitung jumlah koneksi aktif yang sedang ditangani oleh *server* untuk kemudian menentukan *request* selanjutnya akan diteruskan ke *server* mana. Dari gambar dilihat bahwa *request* selanjutnya diteruskan ke *Server-2* yang dalam waktu tersebut menangani *request* yang paling sedikit.

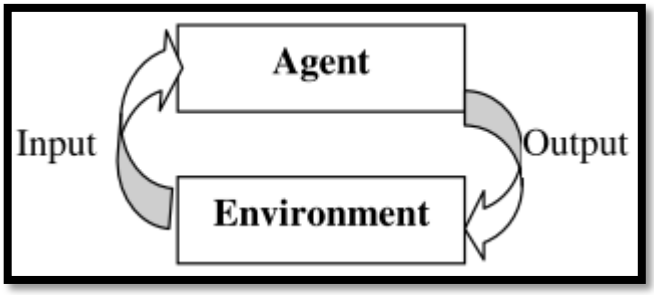
Penelitian [17] menjelaskan bahwa *server* dengan spesifikasi *hardware* yang bervariasi, kerja algoritma ini bisa menjadi tidak optimal, disebabkan oleh adanya TCP_WAIT_TIME. Kondisi *wait time* menyebabkan koneksi yang belum diproses disimpan dalam antrian. *Server* dengan spesifikasi rendah akan mengalami waktu

proses yang lebih lama dibandingkan dengan *server* spesifikasi tinggi, sehingga menyebabkan waktu *reply* yang tinggi untuk koneksi yang dilayani oleh *server* dengan spesifikasi rendah.

Algoritma ini menetapkan koneksi pengguna ke setiap server yang terhubung dengan pengguna. Arsitektur keseimbangan beban dirancang berdasarkan jumlah koneksi pengguna. Server distributor terhubung ke permintaan HTTP, dan setiap pengguna baru mengirimkan permintaan HTTP yang akan ditetapkan ke jumlah minimum koneksi dengan server [18] [14].

2.4 Multi-Agent System (MAS)

Multi-Agent System dibangun dari agent-agent yang cerdas (*intelligent agents*). *Intelligent agent* adalah sebuah sistem komputer yang independen, yang mampu melakukan *action* terhadap kebutuhan lingkungannya, khususnya kebutuhan dirinya sendiri, sehingga sebuah *agent* dapat menyelesaikan pekerjaan yang didelegasikan kepadanya secara lebih baik (*delegation*), serta mampu secara cerdas mengelola pengetahuannya (*intelligence*), juga memberikan kontribusi pada lingkungannya (*interconnection*). Setiap *agent* memiliki kemampuan dapat menerima informasi dari luar, yang selanjutnya dapat memberikan aksi ke lingkungan [19]. Sebuah *agent* mampu menerima masukan dari lingkungan di sekitarnya dan memberikan keluaran sesuai dengan masukan tersebut secara otomatis. Interaksi keduanya dapat dilihat pada Gambar 2. Agent memiliki kemampuan atau karateristik seperti kooperatif, mandiri, dapat berkomunikasi antar agent, dan bergerak (mobile). Agent dapat berpindah dari satu node ke node yang lain dengan bebas sesuai dengan tugas yang diberikan.



Gambar 3 Interaksi Agent dan Lingkungannya

Agent tidak selamanya beroperasi sendirian tapi kadang-kadang bisa melibatkan *agent-agent* lain sehingga disebut *multi-agent system*. Dalam interaksi antar *agent* dibutuhkan sebuah protokol yang menangani interaksi tersebut. Contoh protokol komunikasi dua *agent* yang yaitu:

- Menawarkan sebuah aksi yang harus dijalankan
- Menerima tawaran aksi tersebut
- Menolak tawaran aksi tersebut

Berdasarkan contoh sederhana dari protokol yang ada di atas, maka interaksi antara *agent* A dan *agent* B bisa berupa:

- *Agent* A menawarkan sebuah aksi yang harus dijalankan oleh *Agent* B
- *Agent* B mengevaluasi tawaran dari *Agent* A
- *Agent* B menerima atau menolak tawaran tersebut

Multi-Agent System (MAS) merupakan sebuah perangkat lunak berbasis sistem terdistribusi menggunakan agent dengan kemampuan khusus dan *autonomous* (mandiri) pada sebuah jaringan [20]. Sistem *multi-agent* dibangun atas 3 kategori yang umum, yaitu, *provider agent, service requester agent*, dan *middle agent. Provider agent* (penyedia layanan) seperti pencarian informasi atau mememcahkan masalah yang spesifik, *Requester Agent* adalah agent yang membutuhkan layanan tersebut dari *provider agent*, dan *Middle Agent* digunakan

untuk mencari kecocokan antar agent. Contoh sistem *multi-agent* keuangan, Pemantauan perawatan kesehatan berbasis agen seluler dirancang untuk memberi tahu penyedia perawatan yang bertanggung jawab tentang kelainan secara otomatis.

II.2. Metode Penyelesaian Masalah

2.5 State of the Art Penelitian

Penelitian-penelitian terkait yang telah dilakukan sebelumnya dapat dilihat pada Tabel 1 dibawah ini.

Tabel 1 State of the Art

No	Judul Karya Ilmiah, Nama, Tahun Terbit, dan Penerbit	Objek Dan Permasalahan	Metode Penyelesaian	Kinerja
1.	**Topik yang diusulkan:** Mekanisme Distribusi Beban Pada Load Balancing Web Server Dengan Algoritma Least Connection dan Multi-Agent System Pada Lingkungan Virtual	**Objek:** Web Server dan Algoritma Load Balancing **Permasalahan:** Bagaimana merancang dan menguji mekanisme pendistribusian beban dengan teknologi *load balancing* untuk memenuhi kebutuhan dari aplikasi web yang sampai saat ini masih membutuhkan performansi yang lebih tinggi?	Algoritma Least Connection dan Multi-Agent System	Distribusi permintaan yang masuk dibagi dengan hanya mengecek kondisi resources server dan koneksi aktif terkecil yang ditangani server pada saat itu sehingga diharapkan mampu menghasilkan performansi yang lebih baik dibandingkan metode penyelesaian pada penelitian terdahulu, khususnya waktu respon dan nilai error
2.	**Judul:** Round-robin Algorithm in HAProxy and Nginx Load Balancing Performance Evaluation a Review **Penulis:**	**Objek:** Web Server dan Algoritma Load Balancing **Permasalahan:** Bagaimana evaluasi kinerja load balancer yang	Round Robin Least Connection HAProxy NGINX NGINX KA	HAProxy menunjukkan kinerja yang lebih baik di antara NGINX dalam konfigurasi default dengan menerapkan algoritma Round-Robin. Jika

	Luthfian Hadi. Pramono, Robby Cokro Buwono, Yanuar Galih Waskito **Tahun:** 2018 **Penerbit:** Journal of Chemical Information and Modeling	menggunakan algoritma Round Robin dan Least Connection?		konfigurasi Keep Alive pada NGINX aktif, maka NGINX memiliki performa yang sangat baik dan lebih cepat dari yang lain. Algoritma Least Connection menghasilkan hasil yang lebih baik untuk beberapa pengujian, seperti permintaan per detik dan kecepatan transfer (KB/detik).
3	**Judul:** Performance comparisons of web server load balancing algorithms on HAProxy and Heartbeat **Penulis:** Agung B. Prasetijo, Eko D. Widianto, Ersya T. Hidayatullah **Tahun:** 2017 **Penerbit:** IEEE International Conference on Information Technology, Computer, and Electrical Engineering, ICITACEE	**Objek:** Web Server dan Load Balancer **Permasalahan:** Bagaimana merancang sebuah sistem yang dapat membagi beban secara merata dengan memastikan ketersediaan web server ketika server utama mengalami down?	Least Connection, Round Robin, Source, Heartbeat, HAProxy	Algoritma Least Connection terbukti berperan lebih baik dalam hal throughput, waktu respons, jumlah koneksi, dan mengurangi jumlah total permintaan yang gagal. Meskipun, Round Robin adalah yang terbaik ketika sistem menghadapi lalu lintas tinggi.
4.	**Judul:** Web Server Farm Design Using Personal Computer (PC) Desktop **Penulis:** Iqsyahiro Kresna A, Yusep Rosmansyah **Tahun:** 2018 **Penerbit:** IEEE International Conference on Information Technology	**Objek:** Web Server Farm atau Clustering Web Server dan Load Balancer **Permasalahan:** Bagaimana merancang arsitektur cluster web server pada sebuah PC Desktop dan bagaimana performansi yang dihasilkan dari penerapan	Least Connection, Round Robin	Hasil penelitian sangat baik dalam penerapan menggunakan hardware dengan spesifikasi yang tinggi. Dari hasil pengujian Load Balancer, performansi yang dihasilkan oleh algoritma Least Connection lebih baik daripada algoritma Round Robin. Rata-rata waktu Downtime

	and Electrical Engineering (ICITEE)	algoritma load balancing pada PC Desktop		pada pengujian High Availability atau failover adalah 6,587 detik.
5.	**Judul:** Load Balancing Evaluation Tools for a Private Cloud: A Comparative Study **Penulis:** Sahand Kh. Saeid, Tara Ali Yahiya **Tahun:** 2018 **Penerbit:** The Scientific Journal of Koya University Volume VI, No 2(2018)	**Objek:** Penerapan load balancing pada web server di lingkungan private cloud. **Permasalahan:** Bagaimana mengatasi kemacetan lalu lintas pada server yang menyediakan layanan di cloud?	Least Connection, NGINX, HAProxy	Hasil penelitian sangat baik karena diterapkan pada private cloud dan memiliki banyak skenario pengujian. Waktu respons HAProxy sebesar 3.124 s sedangkan NGINX sebesar 3.297 s terhadap permintaan yang masuk dan HAProxy dapat menerima lebih banyak permintaan dengan status beban berat (3201 *request*). Namun CPU Utilization NGINX hanya 20% dan lebih rendah daripada HAProxy yang hampir mencapai 50%
6.	**Judul :** Load Balancing Algorithms Round-Robin (RR), Least-Connection and Least Loaded Efficiency **Penulis:** Dr. Mustafa ElGili Mustafa **Tahun:** 2017 **Penerbit:** International Journal of Computer and Information Technology	**Objek:** Penerapan teknologi load balancing pada 8 HTTP Server **Permasalahan:** Bagaimana meningkatkan efisiensi pendistribusian trafik dengan load balancing pada HTTP Server yang disimulasikan dengan Opnet?	Round Robin, Least Connection, Least Loaded	Hasil penelitian kurang baik karena masih berupa simulasi. Penggunaan CPU pada web server akan lebih tinggi jika menggunakan algoritma Least Loaded dibandingkan dengan dua algoritma lainnya yang memiliki tingkat konsumsi CPU yang sama. Round robin mendistribusikan beban kira-kira sama kecuali server pertama yang menerima jumlah permintaan

				tertinggi dibandingkan dengan 7 server lainnya. Algoritma Least Connection membagi beban paling adil daripada dua algoritma lainnya.
7.	**Judul :** Model of load balancing using reliable algorithm with Multi-Agent System **Penulis:** Muhammad Faizal Afriansyah, Maman Somantri, Munawar Agus Riyadi **Tahun:** 2016 **Penerbit:** International Conference on Electrical Engineering, Computer Science and Informatics	**Objek:** Penerapan teknologi load balancing pada web server di lingkungan virtual **Permasalahan:** Bagaimana menangani masalah peningkatan aktivitas trafik jaringan yang berpengaruh pada beban kerja sistem web pada skala besar?	Least Time First Byte, Multi Agent System, NGINX	Hasil penelitian kurang baik karena salah satu parameter penting load balancing yaitu respon time pada penetilian ini masih cukup besar.
8.	**Judul :** Web Server Load Balancing Based On Memory Utilization Using Docker Swarm **Penulis:** Mochamad Rexa Mei Bella, Mahendra Data, Widhi Yahya **Tahun:** 2018 **Penerbit:** International Conference on Sustainable Information Engineering and Technology	**Objek:** Load Balancing pada Docker **Permasalahan:** Bagaimana mendistribusikan beban server pada Docker Swarm berdasarkan resources utilization agar terhindar dari distribusi beban yang tidak merata?	Memory Utilization, Docker Swarm	Hasil penelitian menunjukkan nilai CPU dan Memory utilization yang dihasilkan lebih tinggi pada node worker 1 namun perbedaannya tidak jauh dari nilai ambang batas yang ditentukan. Hasil ini menunjukkan bahwa masih diperlukan perbaikan untuk memperoleh nilai utilization dibawah ambang batas
9.	**Judul :** Methodology for Load Balancing in Multi-Agent System Using SPE Approach	**Objek:** Algoritma Load Balancing pada SPE (Software	Multi-Agent System, Round Robin, Random,	Hasil penelitian menunjukkan nilai respon time dari algoritma yang diusulkan mencapai

	Penulis: S. Ajitha **Tahun:** 2021 **Penerbit:** Journal of Security Issues and Privacy Concerns in Industry 4.0 Applications	Performance Engineering) **Permasalahan:** Bagaimana menyeimbangkan beban kerja agen pada MAS dengan mengembangkan algoritma pada JADE dan NetLogo?	Mixed Selection	nilai terbaiknya saat diimplementasikan pada JADE, begitupun nilai server utilization menunjukkan hasil yang lebih baik dibandingkan algoritma yang telah ada.
10.	**Judul :** Multi-Agent Cognitive System for Optimal Solution Search **Penulis:** Victor, Ababii Viorica, Sudacevschi Silvia, Munteanu Dimitrie, Bordian Dmitri, Calugari Ana, Nistiriuc Sergiu, Dilevschi **Tahun:** 2018 **Penerbit:** International Conference on DEVELOPMENT AND APPLICATION SYSTEMS	**Objek:** Optimasi pada teori permainan **Permasalahan:** Bagaimana menemukan solusi paling optimal dari sebuah Multi-Agent System berdasarkan konsep Nash Equilibrium?	Multi-Agent System, Nash Equilibrium	Hasil penelitian menunjukkan Semakin banyak agen yang terlibat dalam aksi di lingkungan, semakin cepat dan semakin nyata pengaruh ini.

Tabel diatas merupakan sebuah ringakasan yang memuat beberapa penelitian terdahulu yang memiliki keterkaitan dengan penelitian yang akan dilakukan. Tabel diatas berupa judul, penulis, penerbit, tahun, objek dan permasalahan, metode penyelesaian, kinerja dan korelasi.

Berdasarkan uraian beberapa keaslian penelitian pada Tabel II.1, maka pada penelitian ini akan merancang sebuah mekanisme load balancing untuk menyeimbangkan beban permintaan (*request*) pada aplikasi web. Dalam penerapannya, *load balancer* akan menggunakan sebuah algoritma untuk membagi

beban secara merata dan dikombinasikan dengan metode yang dapat mengecek kondisi *resources server* secara berkala guna mencapai pemanfaatan *resources* yang maksimal. Algoritma dan metode yang digunakan adalah Least Connection dan Multi-Agent System.

Dengan usulan rancangan mekanisme ini, *request* yang masuk dibagi hanya dengan mengecek kondisi *resources server* dan koneksi aktif terkecil yang ditangani server pada saat itu sehingga sistem tidak perlu lagi menghitung jumlah *byte* data yang masuk seperti yang dilakukan pada penelitian sebelumnya (LFB-MAS) dan diharapkan mampu menghasilkan performansi yang lebih baik, khususnya waktu respon dan nilai *error*. Penelitian ini diuji pada lingkungan virtual dengan menggunakan *software* manajemen virtualisasi. Penelitian ini memiliki dua skenario pengujian dengan tiga kali percobaan yang berbeda. Hasil pengujian dari setiap skenario akan dianalisa untuk mengetahui sejauh mana kinerja yang dihasilkan oleh rancangan mekanisme yang diusulkan.

2.6 Tingkat Keaslian (Level Orisinalitas) Topik Penelitian yang diusulkan

Pengembangan topik penelitian yang diusukan ini dapat dilihat dari Tabel 2. Tabel ini berisi kesesuaian antara referensi pada tabel State of the Art dengan metode penyelesaian yang digunakan dalam penelitian.

Tabel 2 Matriks Metode Penyelesaian

No. Referensi dari Tabel II.1	Metode Penyelesaian									
	a	b	c	d	e	f	g	h	i	j
1 (Topik yang diusulkan)				✓		✓				
2		✓		✓			✓			
3	✓	✓		✓						
4				✓						
5		✓	✓	✓						
6					✓	✓				
7					✓	✓				
8										✓
9		✓				✓		✓	✓	
10						✓				

Keterangan:

Nomor Referensi:

1. Analisis Performansi Load Balancing Web Server Dengan Algoritma Least Connections dan Multi-Agent System Pada Lingkungan Virtual
2. Round-robin Algorithm in HAProxy and Nginx Load Balancing Performance Evaluation a Review
3. Performance comparisons of web server load balancing algorithms on HAProxy and Heartbeat
4. Web Server Farm Design Using Personal Computer (PC) Desktop
5. Load Balancing Evaluation Tools for a Private Cloud: A Comparative Study
6. Load Balancing Algorithms Round-Robin (RR), Least-Connection and Least Loaded Efficiency
7. Model of load balancing using reliable algorithm with Multi-Agent System
8. Web Server Load Balancing Based On Memory Utilization Using Docker Swarm
9. Methodology for Load Balancing in Multi-Agent System Using SPE Approach

10. Multi-Agent Cognitive System for Optimal Solution Search

Metode Penyelesian:

a. Heartbeat
b. Round Robin
c. Least Loaded
d. Least Connection
e. Least Time First Byte
f. Multi Agent System
g. Source
h. Random Selection
i. Mixed Selection
j. Memory Utilization

Berdasarkan pengembangan topik penelitian tentang load balancing pada web server yang dapat dilihat dari Tabel 2 maka topik penelitian ini memenuhi level keaslian atau level orisinalitas tingkat 4 sesuai dengan tingkat keaslian dari pedoman penulisan proposal yang dikeluarkan oleh Program Studi Magister Teknik Informatika UNHAS dilihat pada Tabel 3. Topik penelitian yang diusulkan memenuhi tingkat keaslian 4 karena metode penyelesaian yang diusulkan yaitu algoritma Least Connection dan metode Multi-Agent System belum digunakan untuk menyelesaikan masalah kelebihan beban pada web server. Metode yang diusulkan juga dikategorikan sebagai metode penyelesaian baru karena menggabungkan beberapa metode lama (poin b) untuk menyelesaikan objek dan permasalahan yang sama.

Tabel 3 Level Orisinalitas Berdasarkan Standar Proposal Program Studi Magister Teknik Informatika Universitas Hasanuddin

Level orisinalitas	**Objek penelitian**	**Jenis permasalahan**	**Metode penyelesaian masalah**	**Status orisinal proposal/penelitan**
1	xxxxx	Tidak jelas, sangat minim untuk level S2	xxxxxx	Orisinalitas tidak ada
2	Terpakai (sudah digunakan sebelumnya)	Serupa dengan objek yang sama	Sudah digunakan untuk objek dan jenis permasalahan sebelumnya	Sangat tidak orisinal (replikasi)
3	Baru	Serupa dengan sebelumnya	Sama untuk masalah yang serupa	Kurang
4	Terpakai (sudah digunakan sebelumnya)	Serupa dengan objek yang sama	Lain yang tersedia (belum digunakan untuk masalah yang sama)	Minimalis
5	Baru	Serupa dengan sebelumnya	Lain yang tersedia	Minimalis
6	Terpakai	Baru	Tersedia	Orisinal
7	Baru	Baru	Tersedia	Orisinal
8	Terpakai	Serupa	Baru	Orisinal+Novelty
9	Baru	Serupa	Baru	Orisinal+Novelty
10	Terpakai	Baru	Baru	Sangat Orisinal+Novelty
11	Baru	Baru	Baru	Sangat Orisinal+Novelty

Kategori metode penyelesaian baru:

a. Modifikasi dari metode yang lama
b. Penggabungan beberapa metode lama
c. Metode yang baru dikembangkan dan atau belum pernah digunakan sebelumnya
d. Selalu menghasilkan keluaran yang lebih baik

C. Target Hipotesis Penelitian

Target hipotesis pada penelitian ini adalah nilai dari metrik parameter performansi yang dihasilkan aplikasi web setelah diterapkan mekanisme yang

diusulkan dapat lebih baik. Penelitian ini ditargetkan memiliki nilai-nilai parameter yang lebih baik dibandingkan penelitian terdahulu, seperti nilai waktu respon yang lebih kecil, nilai *throughput* yang lebih besar, nilai *error* yang dihasilkan sangat minim sehingga jumlah *request* yang berhasil ditangani oleh sistem dapat lebih banyak dan penggunaan *resources* dari setiap *server* memiliki nilai yang stabil atau dengan kata lain tidak *overload* atau *underload*. Penelitian ini juga ditargetkan menghasilkan analisis performansi yang detail dan akurat sehingga mampu menjadi referensi dalam implementasi *real* dilapangan.

D. Kerangka Pikir

Kerangka pikir dapat dilihat pada Tabel 4 yang menjelaskan alur penelitian yang akan dilakukan.

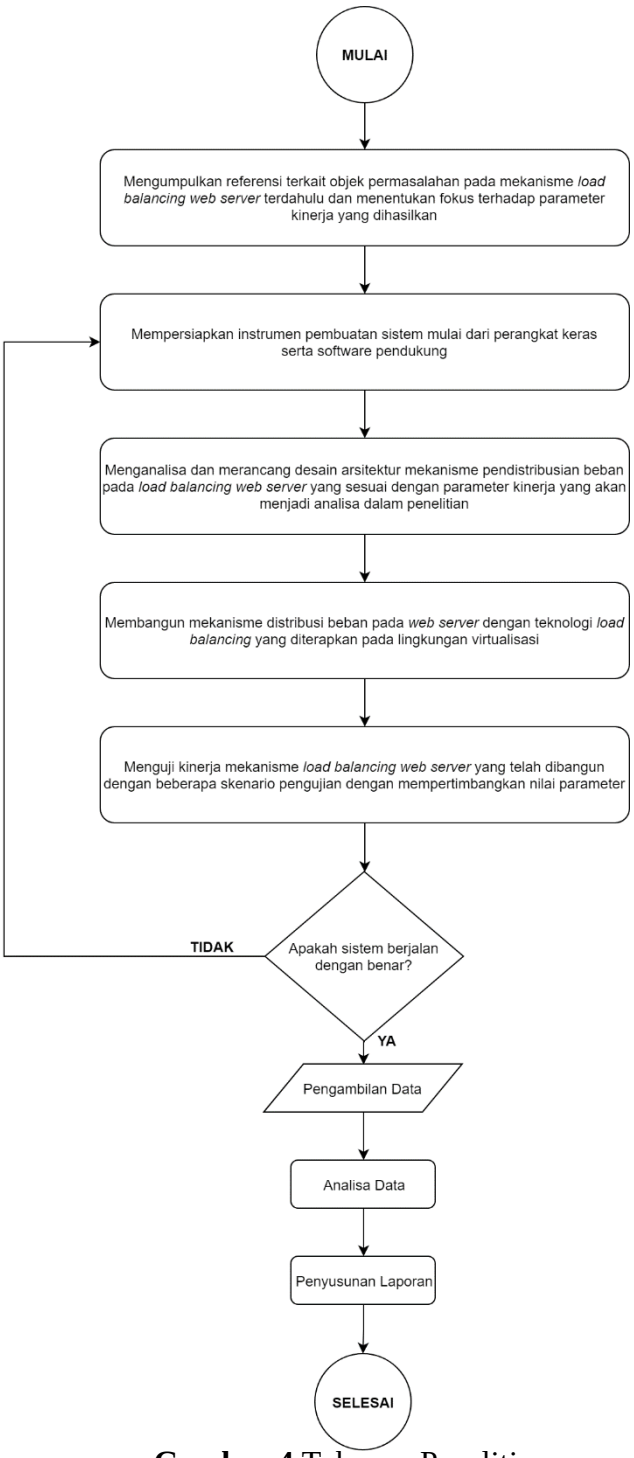
Tabel 4 Kerangka Pikir Penelitian

Masalah • Kinerja website saat banyak pengguna internet yang mengunjunginya semakin menurun karena kelebihan beban kerja pada web server • Aplikasi web sampai saat ini masih membutuhkan kinerja yang lebih tinggi • Mekanisme *load balancing* yang terdahulu kurang memperhatikan aspek *resource* pada *server*
Solusi Merancang dan mengembangkan mekanisme *load balancing* pada lingkungan virtualisasi dengan memperhatikan aspek *resource* dalam mendistribusikan beban kerja *web server* untuk mencapai kinerja maksimum pada website.
Metode Penyelesaian Kombinasi algoritma *Least Connection* dan *Multi-Agent System* dalam mendistribusikan *request* dari pengguna internet berdasarkan kondisi *load* dan jumlah koneksi aktif terkecil yang sedang ditangani *web server*
Hasil Mekanisme *load balancing* dapat mendistribusikan beban *web server* secara merata tanpa menyebabkan kondisi *overload* atau *idle* pada *web server* yang membuat kinerja *website* semakin tinggi.

BAB III
METODOLOGI PENELITIAN

III.1 Tahapan Penelitian

Diagram alur tahapan penelitian dapat dilihat pada Gambar 4 dibawah ini.



Gambar 4 Tahapan Penelitian

III.2 Teknik Pengumpulan Data

Data yang digunakan pada penelitian ini adalah nilai parameter yang menggambarkan kinerja mekanisme *load balancing* yang dibangun. Data tersebut diperoleh dari hasil pengujian masing-masing skenario, seperti nilai rata-rata waktu respon, *throughput* (*transaction/*s), dan jumlah error.

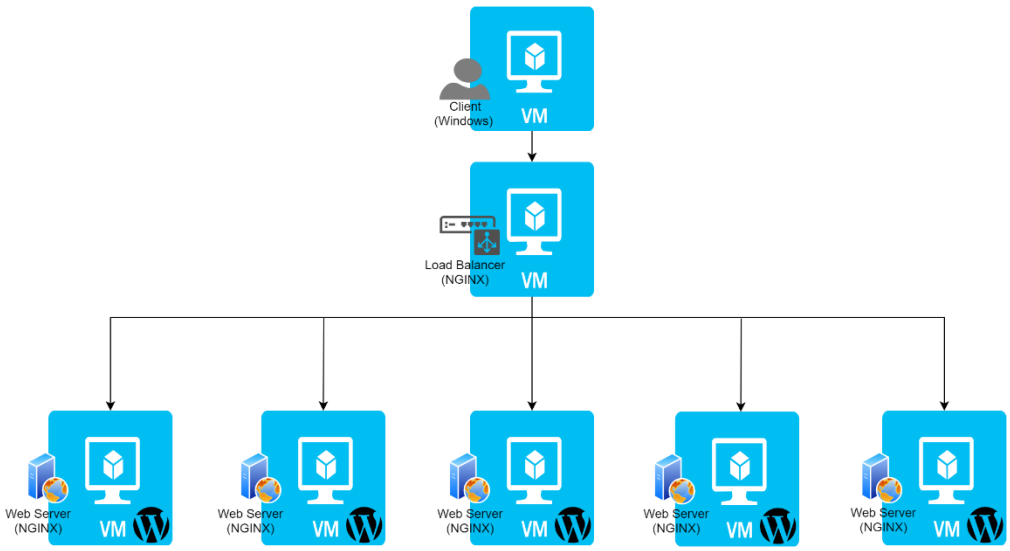
Hasil pengujian dari setiap skenario dianalisa secara kuantitatif kemudian ditampilkan dalam bentuk grafik dan tabel. Parameter tersebut adalah:

- Waktu Respon (ms): Parameter yang merupakan waktu minimum yang dibutuhkan dari sebuah sistem yang menerapkan algoritma penyeimbangan beban tertentu untuk merespon. Semakin kecil nilai waktu respon maka semakin bagus pula kinerja sistem.
- *Throughput* (request per second): Parameter yang merupakan jumlah keseluruhan permintaan yang telah dieksekusi oleh sistem. Parameter ini mencerminkan kapasitas server dalam hal seberapa banyak beban yang dapat ditangani. Sistem membutuhkan nilai *throughput* tinggi untuk mencapai kinerja yang lebih baik. *Throughput* diukur sebagai jumlah total transaksi dalam waktu tertentu atau transaksi per detik (TPS).
- Jumlah *error*: Parameter yang menunjukkan jumlah koneksi yang tidak berhasil dibuat karena gagal dieksekusi atau diproses oleh *server backend* dan direpresentasikan dalam persen. Persentase *error* dapat dilihat pada kolom %Error pada tab Aggregate Report dari software benchmark Apache JMeter™.

III.3 Rancangan Sistem

3.3.1 Rancangan Arsitektur *Load Balancing*

Mekanisme *load balancing web server* diimplementasikan pada lingkungan virtual menggunakan perangkat lunak PROXMOX VE sebagai *platform* manajemen virtualisasi dan Linux Debian 7.11 sebagai sistem operasinya. Mesin virtual dibangun untuk bertindak sebagai mesin server fisik. Mesin virtual ini berisi dua jenis layanan yang menggunakan NGINX, yaitu *load balancer* dan *web server*. Layanan *web server* berisi situs *web* dengan konten sederhana yang dibuat menggunakan CMS Wordpress. Secara keseluruhan sistem terdiri atas satu buah mesin *server* yang berperan sebagai *Host*. *Server host* dibagi menjadi 7 mesin virtual diantaranya 5 mesin virtual berperan sebagai *web server*, 1 mesin virtual berperan sebagai *server load balancer*, dan 1 mesin virtual berperan sebagai *client*. Arsitektur sistem dapat dilihat pada Gambar 5.



Gambar 5 Arsitektur Load Balancing

Seluruh mesin virtual tersebut saling terkoneksi melalui LAN virtual pada PROXMOX dengan konfigurasi *bridge* dari setiap *Network Interface Card* (NIC) mesin virtual. Mesin virtual yang berperan sebagai *load balancer* memiliki dua NIC yaitu NIC publik dan NIC privat. NIC publik terkoneksi langsung ke *client*, sedangkan NIC privat digunakan untuk interkoneksi antar *web server*.

Arsitektur pada penelitian ini terbagi menjadi dua bagian, yaitu *main server* dan *backend server*. *Main server* bertugas sebagai *load balancing server* dan pusat informasi yang menyimpan data dari para agen. *Backend server* bertugas sebagai *web server* yang memuat konten aplikasi CMS Wordpress.

3.3.2 Rancangan Algoritma *Least Connection*

Algoritma LC membagi permintaan atau *request* dari *client* diantara *backend server* dengan melihat jumlah koneksi aktif terkecil yang sedang ditangani pada saat itu. Berikut pseudocode dari algoritma ini:

```
for (j=0; j<n; j++) {
for (i=j+1; i<n; i++) {
if (C(Si) < C(Sj))
m=i;
}
return Sj;
}
return NULL;
```

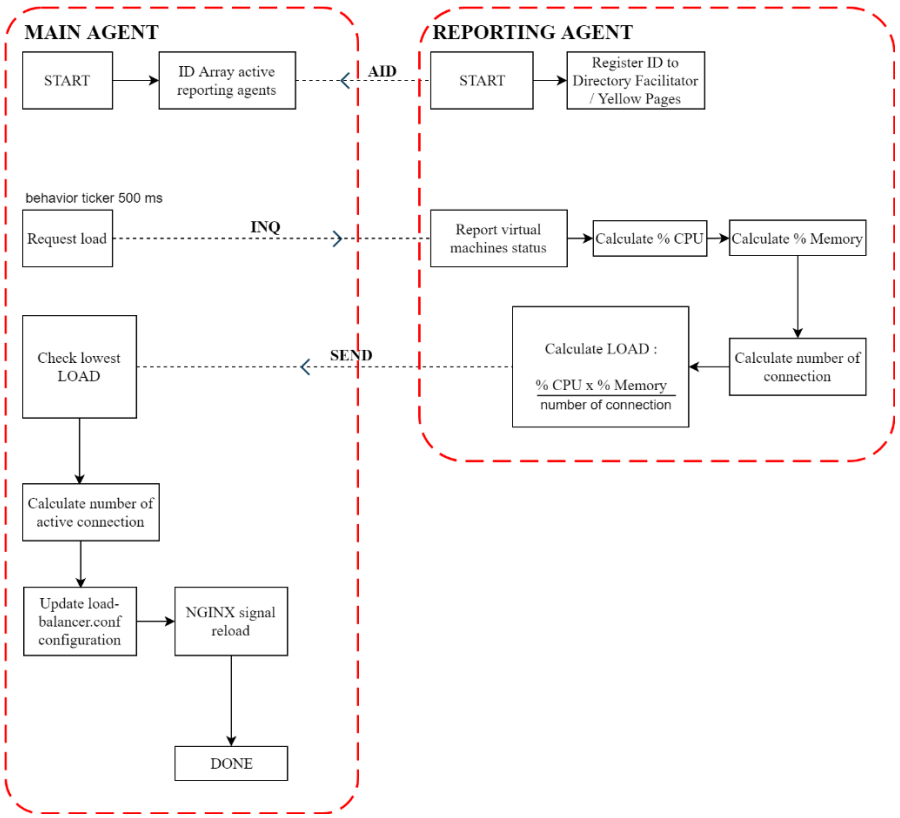
S mewakili sekelompok server dan C mewakili jumlah koneksi dari server Si dan Sj saat ini. Namun jumlah koneksi yang ditangani oleh server berbanding terbalik dengan kondisi beban, sehingga algoritma ini membuat distribusi beban tidak merata.

Algoritma *Least Connection* diimplementasikan pada agen utama. File konfigurasi *load-balancer.conf* semula bersifat statik yang terletak pada NGINX diubah menjadi file konfigurasi yang bersifat dinamis. File ini berisi alamat IP *web server* dengan koneksi terkecil dan kondisi *reources* nya tidak melebihi ambang batas. Perubahan ini dilakukan karena secara *default* daftar *server* yang terletak pada file konfigurasi *load-balancer.conf* tidak dapat berubah sehingga sulit untuk merutekan permintaan ketika 2 syarat (kondisi resources dan jumlah koneksi terkecil) dijadikan acuan dalam membagi beban.

3.3.3 Rancangan Multi-Agent System

Pada penelitian ini *Multi-Agent System* diprogram menggunakan bahasa pemrograman Java. MAS bekerja dengan memanfaatkan agen-agen pintar yang menjalankan tugas sesuai fungsinya masing-masing. Seluruh agen dibangun menggunakan bahasa pemrograman Java dengan library Java Agent Development Framework (JADE).

Penelitian ini menggunakan dua jenis agen, yaitu *main* agen dan *reporting* agen dengan fungsi yang berbeda. *Main agent* berfungsi untuk meminta informasi kondisi terkini dari *resources server* kepada *reporting agent*. *Reporting* agen berfungsi untuk menerima permintaan *main* agen dan mengembalikan informasi *resources* ke *main* agen. Adapun alur kerja dari dua jenis agen dalam penelitian ini dapat dilihat pada Gambar 6.



Gambar 6 Alur Kerja Multi-Agent System

Main agent dijalankan pada mesin *load balancer* sambil mengecek berapa agen yang statusnya aktif dari setiap *reporting agent*. Setelah *reporting agent* dijalankan pada masing-masing *web server* maka para agen ini akan mendaftarkan dirinya ke Yellow Pages JADE agar *Main agent* dapat mengenali ID masing-masing *reporting agent*. ID tersebut akan masuk kedalam array yang dibentuk oleh *Main agent*. Setelah array terbentuk maka *Main agent* akan meminta *reporting agent* untuk melaporkan kondisi *load* mereka setiap 500 ms. *Reporting agent* pada masing-masing mesin virtual menerima permintaan tersebut dan langsung menghitung persentase CPU dan Memori dan jumlah koneksi. Setelah informasi tersebut dikumpulkan, selanjutnya dilakukan kalkulasi *load*. Hasil kalkulasi tersebut dilaporkan ke *main agent*. *Main agent* lalu memilih *load* terendah dari list yang dikirimkan padanya untuk kemudian dihitung jumlah koneksi aktif terkecil dari mesin virtual dengan load terendah tersebut. Selanjutnya *main agent* melakukan update terhadap NGINX.

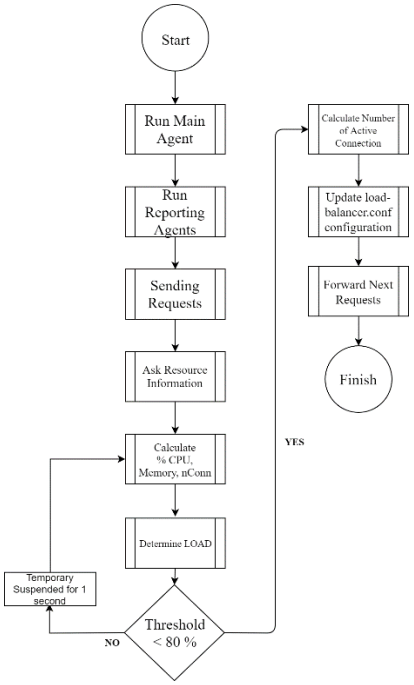
III.4 Metode Pengujian

Mekanisme *load balancing web server* diuji pada dua skenario pengujian untuk mengetahui kinerja dari mekanisme yang dibangun. Pada penelitian ini, Apache Jmeter digunakan untuk membantu skenario pengujian. Apache Jmeter menghasilkan lalu lintas HTTP atau HTTP *request* kemudian dikirim ke sekumpulan *web server*. Masing-masing skenario diuji dengan mengirimkan 1500 HTTP. HTTP *request* dikirim ke *web server* dengan waktu jeda selama 10 detik untuk setiap *request*.

a. Skenario 1

Skenario pertama menguji mekanisme *load balancing* menggunakan kombinasi Algoritma Least Connection yang dipadukan dengan Multi-Agent System (LC-MAS). Skenario ini menggunakan enam agen dimana satu *main* agen yang terletak pada *server load balancer* dan lima *reporting* agen yang terletak pada masing-masing *web server*. Pada skenario ini terdapat nilai *threshold* yang menjadi acuan sistem dalam mengambil keputusan saat mendistribusikan *request*. Nilai *threshold* pada penelitian ini dikategorikan menjadi dua kondisi, **normal** dan ***overload*.** Gambar 7 merupakan flowchart dari skenario 1.

Proses pertama yaitu menjalankan *main agent* lalu *reporting agents*. Selanjutnya dengan bantuan tool Apache Jmeter, sejumlah *requests* dikirimkan ke alamat IP *load balancer*. *Main agent* akan meminta informasi kondisi *resources* dari masing-masing *web server*. *Reporting agents* yang menerima permintaan tersebut langsung menghitung jumlah load dari masing-masing *web server* dan menentukan mana yang nilainya normal dan tidak.



Gambar 7 Flowchart Skenario 1

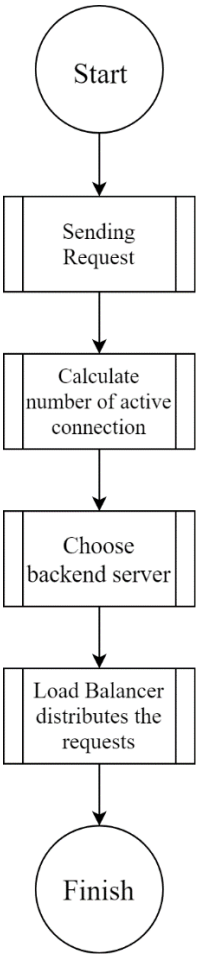
Ketika kondisi *resources* normal, *web server* dinyatakan layak menangani *request* sedangkan ketika kondisi *resources overload* maka dinyatakan tidak layak. Saat dinyatakan tidak layak, *backend server* diberi waktu 1 detik agar kondisi *resources* kembali normal. Daftar *web server* yang normal dikirim kembali ke *main agent* untuk dilakukan perhitungan jumlah koneksi aktif terkecilnya. *Main agent* kemudian memperbarui file konfigurasi *load-balancer.conf* yang lama. *Load balancer* kemudian meneruskan *request* selanjutnya. Tabel 5 memperlihatkan pemetaan alamat IP pada Skenario 1.

Tabel 5 Konfigurasi dan Spesifikasi Skenario 1

No.	VM ID	RAM	OS	KETERANGAN	Alamat IP
1.	112	2 GB	Linux Debian 7.11	Load balancer + agent server	10.163.12.112
2.	113			Web server 1	10.163.12.113
3.	114			Web server 2	10.163.12.114
4.	115			Web server 3	10.163.12.115
5.	116			Web server 4	10.163.12.116
6.	117			Web server 5	10.163.12.117
7.	118		Windows 10	User	10.163.12.118

b. Skenario 2

Skenario kedua menguji mekanisme *load balancing* menggunakan Algoritma Least Connection (LC). *Request* didistribusikan berdasarkan dengan jumlah koneksi aktif terkecil yang sedang ditangani *web server*. *Load balancer* menghitung jumlah koneksi aktif terkecil dari masing-masing *web server* untuk dibandingkan satu sama lain. *Web server* dengan jumlah koneksi aktif terkecil dipilih untuk menangani *request* selanjutnya. Gambar 8 merupakan flowchart dari skenario 2.



Gambar 8 Flowchart Skenario 2

Pada Skenario 2, *request* didistribusikan hanya berdasarkan satu kondisi, yaitu jumlah koneksi. File konfigurasi *load-balancer.conf* berisi 5 alamat IP

web server yang bersifat statik dan tidak dapat dilakukan perubahan. Tabel 6 memperlihatkan konfigurasi alamat IP pada Skenario 2.

Tabel 6 Konfigurasi dan Spesifikasi Skenario 2

No.	VM ID	RAM	OS	KETERANGAN	Alamat IP
1.	122	2 GB	Linux Debian 7.11	Load balancer	10.163.12.122
2.	123			Web server 1	10.163.12.123
3.	124			Web server 2	10.163.12.124
4.	125			Web server 3	10.163.12.125
5.	126			Web server 4	10.163.12.126
6.	127			Web server 5	10.163.12.127
7.	122		Windows 10	User	10.163.12.118

III.5 Instrumen Penelitian

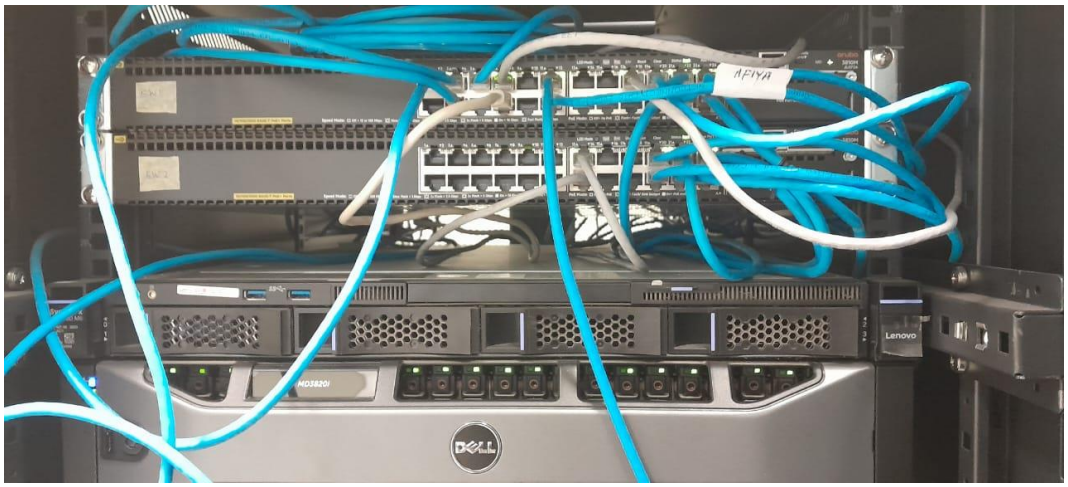
1. Software
   a. PROXMOX Virtual Environment 7.0-11
   b. Sistem Operasi Windows 10-64 bit
   c. Sistem Operasi Linux Debian 7.11
   d. NGINX
   e. Apache JMeter
   f. MariaDB
   g. PuTTY
   h. Microsoft Office Professional Plus 2013
   i. Browser Chrome
2. Hardware
   a. Perangkat Server DELL
   b. Laptop Lenovo 80F6 Intel® Core™ i3-5005U CPU @ 2.00GHz (4CPU) RAM 8GB HDD 500GB

BAB IV
HASIL DAN PEMBAHASAN

IV.1 Analisis Perancangan Mekanisme *Load Balancing*

Mekanisme *Load Balancing* dikembangkan pada lingkungan virtual menggunakan bantuan PROXMOX sebagai manajemen virtualisasi. Software ini diinstal pada perangkat *server* fisik untuk membuat mesin virtual dengan Linux Debian 7.11 sebagai sistem operasinya. Gambar 9 merupakan perangkat *server* yang digunakan dan Gambar 10 merupakan perangkat *switch* yang menyambungkan koneksi *server* yang masih menggunakan jaringan lokal kampus. Sedangkan untuk keperluan instalasi software pada server memanfaatkan kabel LAN sebagai alat untuk terhubung ke internet.

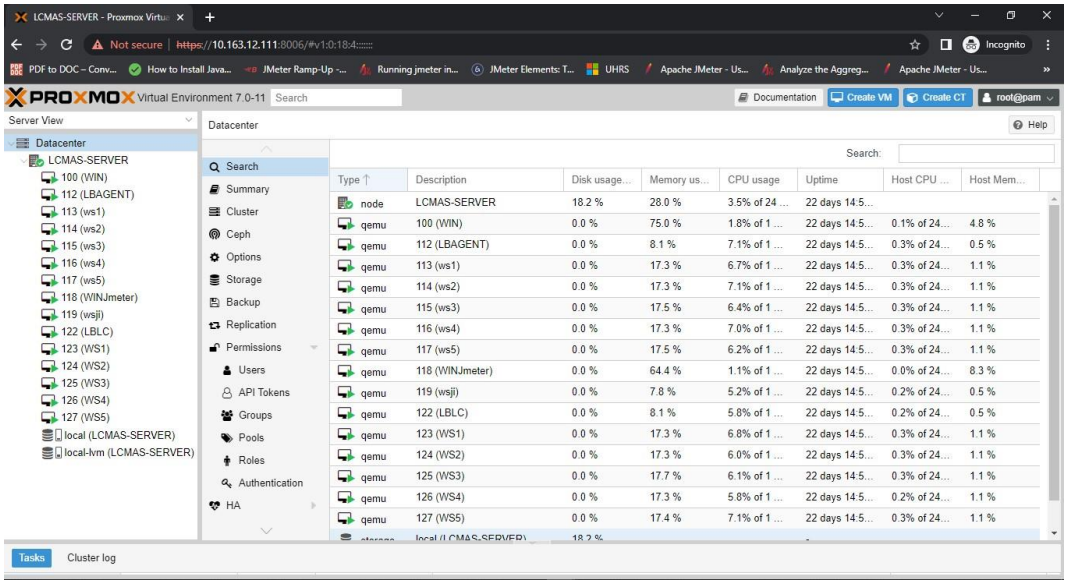
Gambar 9 Perangkat *Server* yang digunakan



Gambar 10 Perangkat *Switch* yang digunakan

NGINX berperan sebagai *web server* dan *load balancer* yang diinstal pada masing-masing mesin virtual. Terdapat 5 mesin virtual yang berisi aplikasi CMS Wordpress sederhana di masing-masing skenario. MariaDB digunakan sebagai

database dari wordpress tersebut. Gambar 11 merupakan tampilan dari data center pada penelitian ini, yaitu LC-MAS SERVER.

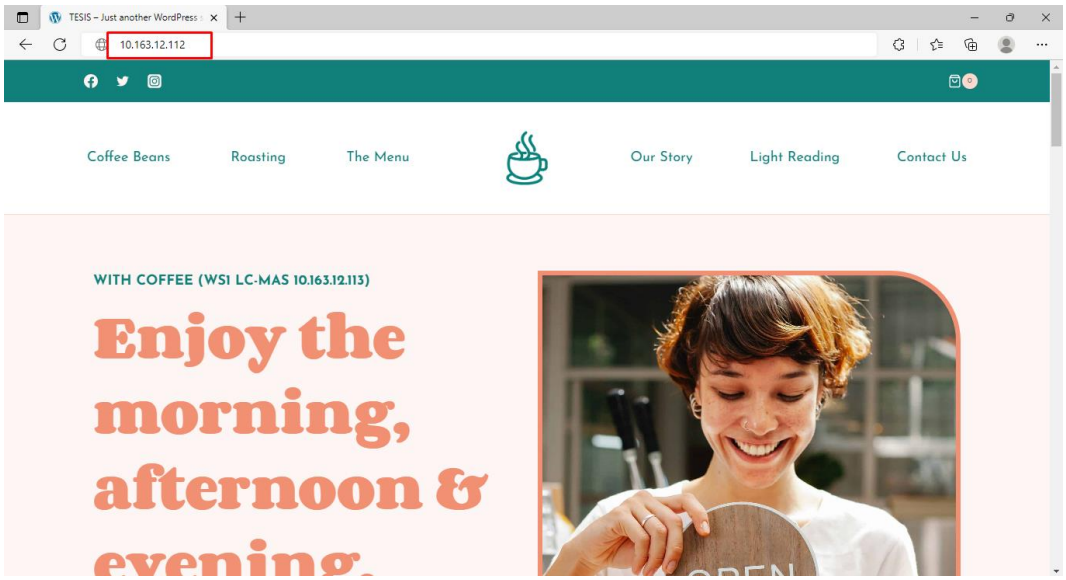


Gambar 11 LCMAS-SERVER Data Center

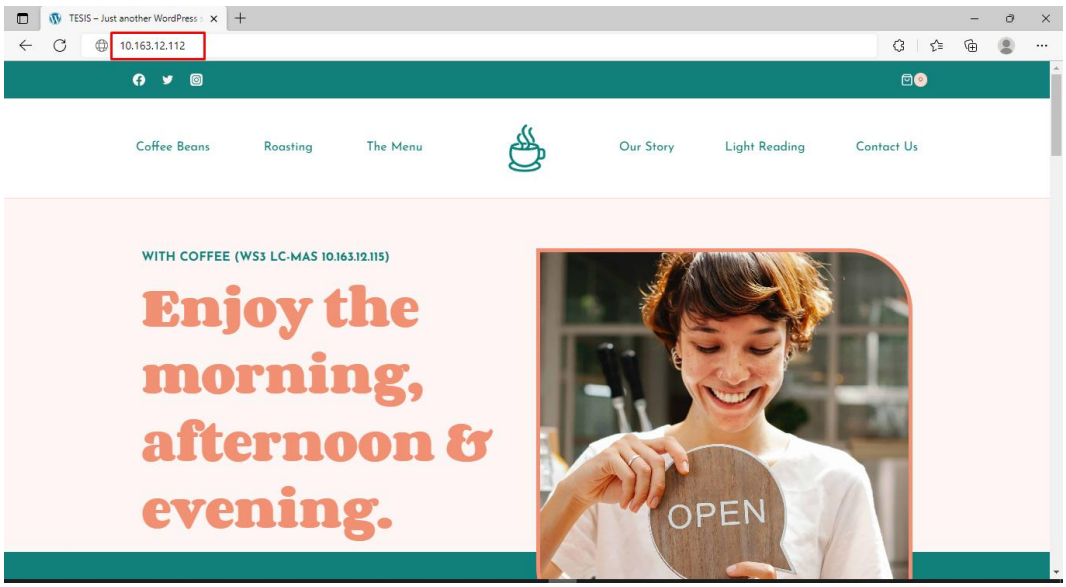
Masing-masing mesin virtual diberi nama agar lebih mudah dikenali. Pada skenario 1 (LC-MAS) mesin virtual diberi nama ws1 sampai ws5 sedangkan skenario 2 mesin virtual diberi nama WS1-WS5. Adapun nama lain seperti LB-AGENT dan LBLC merupakan nama mesin virtual dari *load balancer* kedua skenario. Mesin virtual WINJmeter digunakan sebagai *user* atau *client* yang menjalankan tools Apache Jmeter.

Masing-masing virtual mesin diberi tanda khusus pada konten wordpress yang dimuat untuk mengetahui apakah *load balancer* mendistribusikan *request* ke mesin virtual yang berbeda ketika alamat IP *load balancer* diakses. Gambar 12 memperlihatkan ketika alamat IP *load balancer* (10.163.12.112) pada Skenario 1 diakses, *request* tersebut diarahkan ke *web server* 1 dengan alamat IP 10.163.12.113. Selanjutnya ketika alamat IP *load balancer* kembali diakses, *request* tersebut diarahkan ke *web server* 3 dengan alamat IP 10.163.12.115 yang

dapat dilihat pada Gambar 13. Hal ini menunjukkan bahwa *load balancer* berhasil mendistribusikan *request* ke kluster *web server* yang ada dibelakangnya dan menandakan bahwa perancangan *load balancer* pada kluster *web server* di lingkungan virtualisasi ini berhasil.



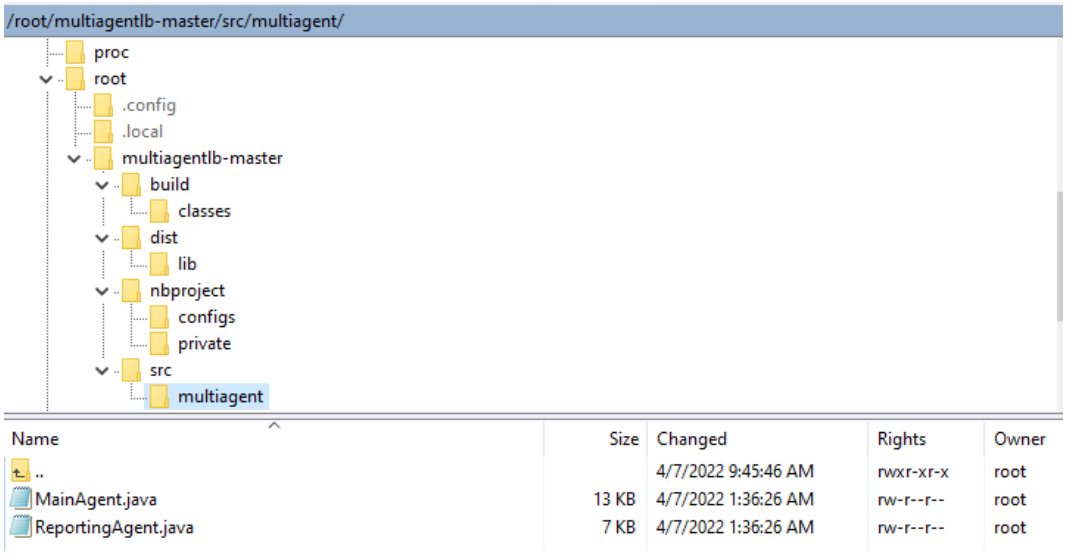
Gambar 12 Request Pertama Skenario 1 ditangani oleh WS 1



Gambar 13 Request Kedua Skenario 1 ditangani oleh WS 3

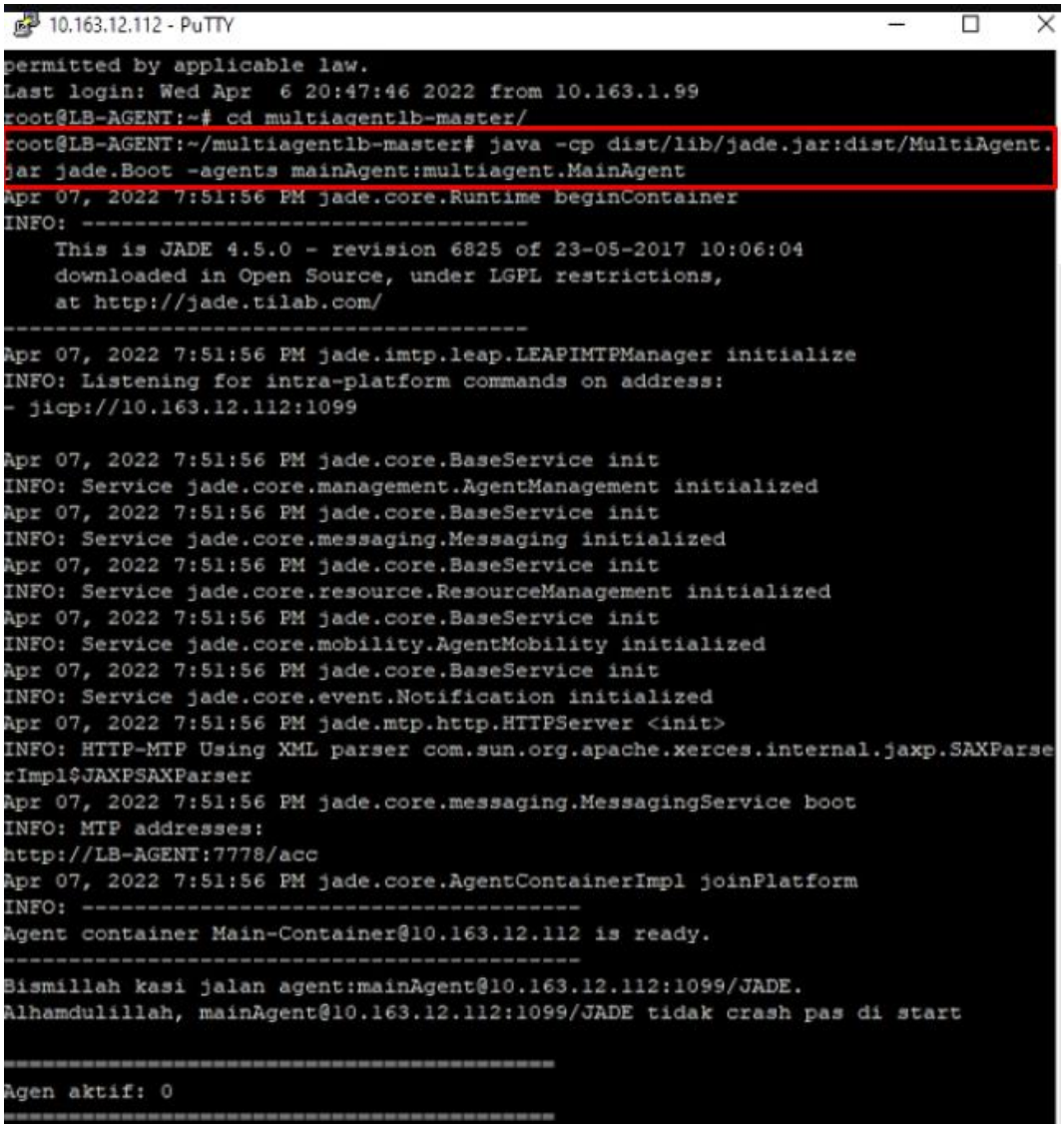
Multi-Agent System dibangun dengan memanfaatkan library JADE pada bahasa pemrograman Java dan dikembangan sesuai kebutuhan. Pada penelitian ini,

MAS dirancang sebagai alat monitoring secara *real-time* kondisi *resources server* dengan mengambil nilai CPU dan Memory untuk dijadikan perhitungan dalam menentukan *load server* yang menjadi salah satu pertimbangan untuk *load balancer* mendistribusikan *request*. Agen yang dibentuk terdiri dari 2 fungsi utama, yaitu *Main Agent* dan *Reporting Agents*. *Main Agent* diletakkan pada *server load balancer* dan diberi nama file MainAgent.java. *Reporting Agents* diletakkan pada masing-masing *web server* dan diberi nama file ReportingAgent.java. File tersebut disimpan pada direktori src\multiagent di dalam struktur direktori NGINX *web server*. Gambar 14 memperlihatkan struktur direktori dari MAS yang dikembangkan dalam penelitian ini.



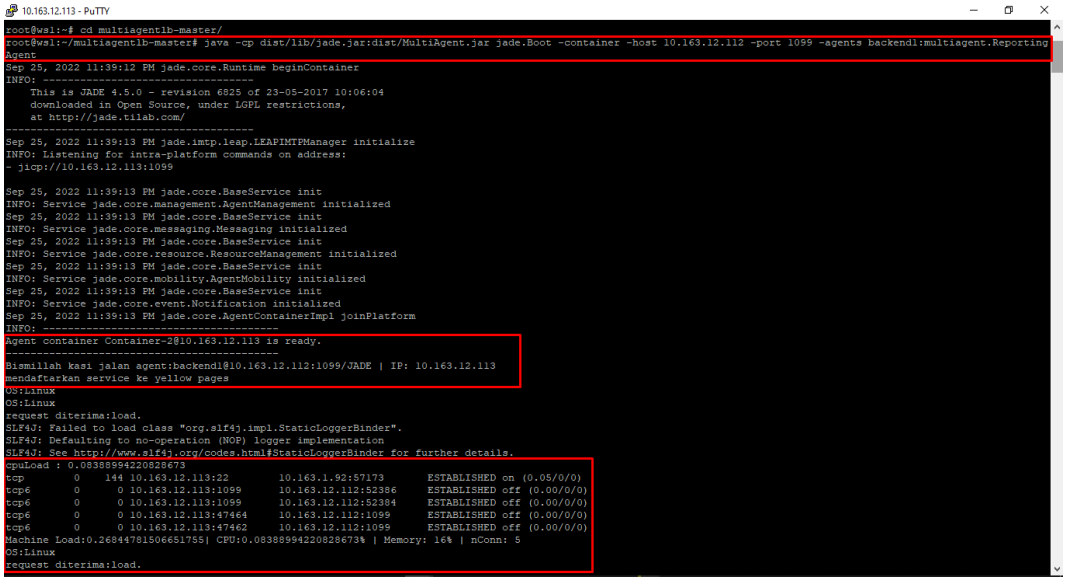
Gambar 14 Struktur Direktori Multi-Agent System

Untuk mengetahui apakah MAS berjalan dengan baik, kedua jenis agen diberi inisialisasi pesan yang menandakan ketika mereka dijalankan, tidak terjadi *crash*. Gambar 15 memperlihatkan ketika *Main Agent* pada Skenario 1 dijalankan.



Gambar 15 Start Main Agent

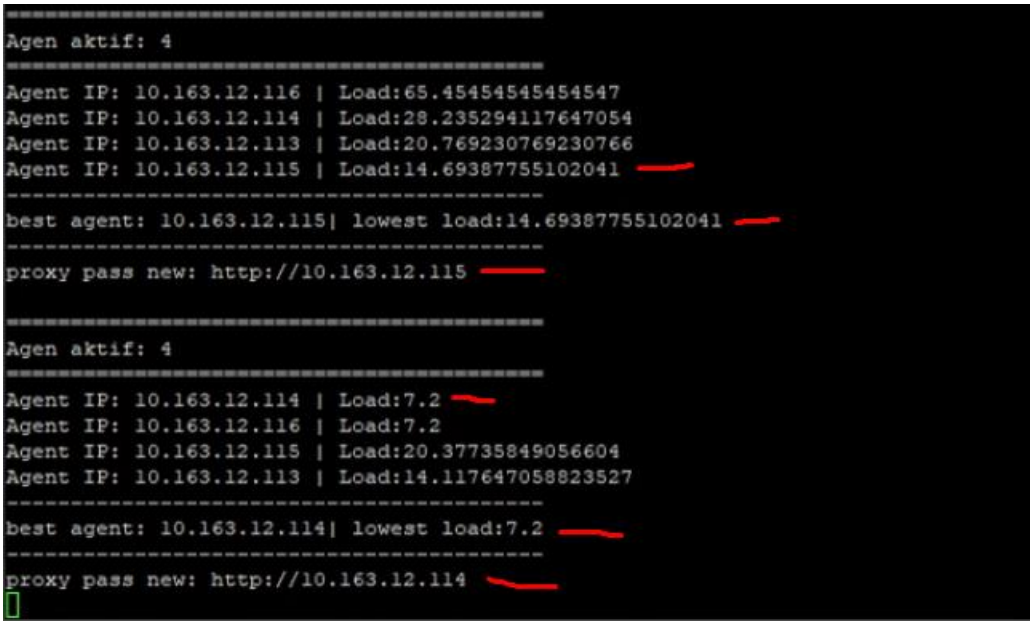
Main agent pada penelitian ini diatur untuk selalu mengecek *reporting agent* yang berstatus aktif. Aktivitas ini bekerja secara *looping* sampai *main agent* dimatikan. Demi menghindari tumpukan file log yang berisi status *reporting agent* yang dapat menyebabkan peningkatan ukuran penggunaan Memori pada mesin virtual , maka *main agent* diberi behavior untuk selalu meminta laporan setiap 500 ms saja. Sehingga ketika *main agent* tidak meminta status maka *reporting agent* tidak perlu mengirimkan file log. Hal ini berbeda dengan penelitian sebelumnya yang mengatur *reporting agent* selalu melaporkan status *load* setiap 1 detik.



Gambar 16 Start Reporting Agent

Gambar 16 memperlihatkan ketika *Reporting Agent* pada Skenario 1 dijalankan pada ws1 (10.163.12.113). Setelah sintaks dijalankan, sebuah container jade akan terbentuk. Container ini yang akan berkomunikasi dengan Main Container (main agent). Host dan port pada container ini yang akan didengar oleh Main Container agar dapat berkomunikasi dengan container lain pada platform (gui) jade. Ketika container berhasil terbentuk, agent mendaftarkan dirinya ke yellow pages JADE (DF).

Gambar 17 memperlihatkan tampilan dari *main agent* ketika *reporting agent* telah aktif. Terdapat 4 agen aktif dengan jumlah load yang berbeda-beda pada saat mekanisme dijalankan. Update pertama alamat IP 10.163.12.115 yang merupakan ws3 dari Skenario 1 memperoleh *load terendah* dan menjadi agen terbaik sehingga *request* selanjutnya dikirimkan ke ws3.



Gambar 17 List Reporting Agent Aktif

Pada update kedua, alamat IP 10.163.12.114 yang merupakan ws2 dari Skenario menjadi agen terbaik karena memiliki load paling rendah. Walaupun jumlah *load* antara ws 2 dan ws 4 (10.163.12.116) sama, namun *main agent* kembali menghitung julmlah koneksi aktif terkecil dari keduanya dan dipilih ws 2 karena memiliki jumlah koneksi aktif terkecil dibandingkan ws 4.

Penerapan algoritma *least connection* digabungkan dengan fungsi utama *main agent*. Perhitungan jumlah koneksi aktif terkecil tidak lagi dilakukan di file konfigurasi default NGINX yang umumnya mengatur alamat IP backend server yang akan menerima *request*. Dengan memanfaatkan fitur NGINX reload signal, maka pada bagian *proxy pass* dalam file *load_balancer.conf* akan berubah sesuai 2 kondisi yaitu *load* dan jumlah koneksi aktif terkecil.

IV.2 Analisis Kinerja Mekanisme *Load Balancing*

Analisis pada hasil pengujian mekanisme *load balancing* dilakukan untuk mengetahui kinerja dari mekanisme tersebut. Pengujian yang dilakukan pada penelitian ini berupa *load testing* untuk melihat bagaimana kapasitas kinerja sistem

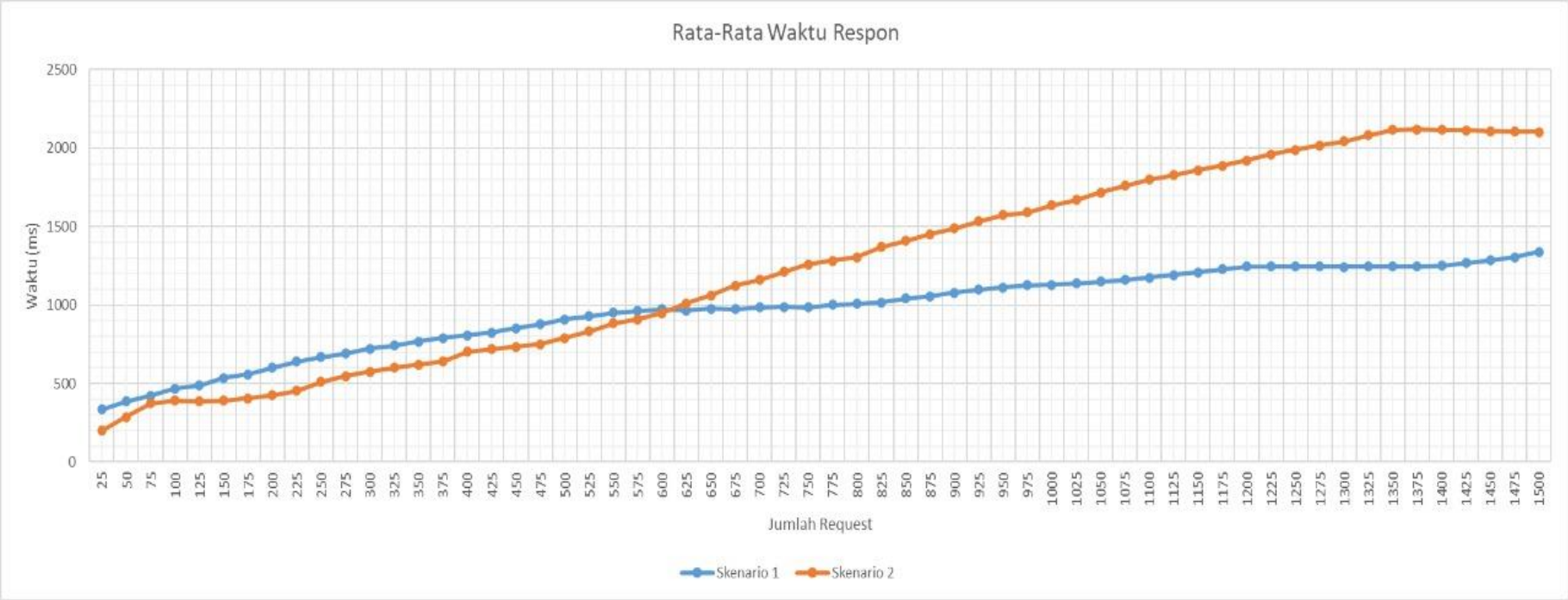
terhadap waktu respon sebuah transaksi atau permintaan. Ketika kinerja sistem memerlukan waktu yang cukup lama dan transaksi atau permintaan menjadi tidak stabil, hal ini berarti mekanisme telah mencapai atau telah berada pada kapasitas maksimumnya. Setelah mengetahui keadaan mekanisme ketika mencapai kapasitas maksimumnya dan pada saat kapan serta bagaimana mekanisme ini mencapai kapasitas maksimumnya, maka dapat dilakukan identifikasi penyebab mekanisme *load balancing* tersebut menjadi tidak stabil. Jika masalah atau penyebab terjadinya ketidakstabilan tersebut telah terpenuhi, maka dapat disediakan solusi untuk mengatasi masalah tersebut.

Kinerja dari mekanisme *load balancing* ini dilihat dari nilai parameter yang dihasilkan. Adapun parameter yang paling berpengaruh terhadap kinerja sebuah sistem yang dijadikan bahan analisa pada penelitian ini adalah:

4.2.1 Rata-Rata Waktu Respon

Waktu respon dihitung mulai saat request dikirimkan sampai request tersebut selesai dieksekusi. Parameter ini merepresentasikan waktu yang dibutuhkan sebuah mekanisme *load balancing* yang mengaplikasikan algoritma tertentu untuk merespon sebuah request. Semakin kecil nilai waktu respon maka kinerja sistem tersebut semakin baik. Pada tools Apache Jmeter, parameter ini dapat dilihat dari dashboard Aggregate Report dan Summary Report di kolom Average. Nilai rata-rata waktu respon diperoleh dari total seluruh waktu respon dibagi dengan jumlah request.

Gambar 18 merupakan tampilan grafik dari nilai rata-rata waktu respon kedua skenario. Adapun rincian nilai rata-rata waktu respon yang diperoleh dari kedua skenario dapat dilihat pada Tabel 7.



Gambar 18 Grafik Rata-Rata Waktu Respon Skenario 1 dan 2

Tabel 7 Rata-rata Waktu Respon Skenario 1 dan 2

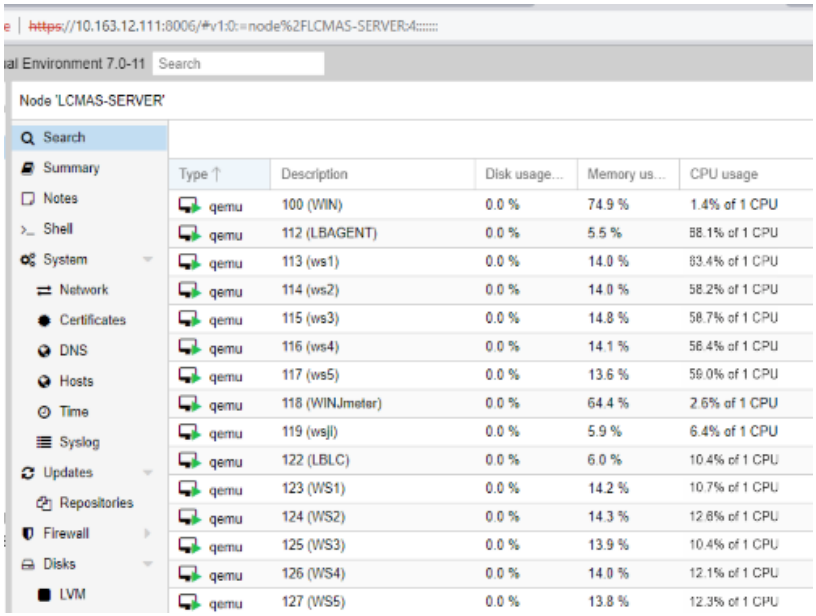
Jumlah Request	Rata-Rata Waktu Respon (ms)	
	Skenario 1	Skenario 2
0	0	0
25	336.16	198.64
50	384.94	287.14
75	423.98	374.61
100	466.97	392.72
125	486.58	386.96
150	536.10	392.55
175	557.22	405.44
200	601.25	424.68
225	640.49	452.40
250	668.90	508.96
275	690.30	548.00
300	721.08	574.51
325	743.34	599.89
350	766.76	619.72
375	790.29	641.68
400	807.77	701.33
425	823.46	719.17
450	852.57	733.65
475	877.33	750.96
500	908.55	791.31
525	928.64	833.73
550	952.49	882.15
575	963.15	908.88
600	974.69	948.40
625	**966.15**	**1010.82**
650	975.59	1060.56
675	974.35	1125.03
700	984.61	1160.67
725	987.90	1211.25
750	985.71	1260.48
775	1001.37	1281.24
800	1008.81	1305.21
825	1016.74	1369.92
850	1041.94	1409.53
875	1054.70	1450.85
900	1077.90	1489.78
925	1097.72	1533.50
950	1112.87	1573.36

975	1126.89	1591.32
1000	1130.90	1635.58
1025	1138.34	1669.56
1050	1149.13	1716.54
1075	1161.41	1760.36
1100	1175.70	1801.02
1125	1192.22	1826.66
1150	1210.05	1858.93
1175	1228.82	1887.14
1200	1244.06	1920.74
1225	1246.86	1960.03
1250	1246.78	1988.62
1275	1246.01	2017.51
1300	1243.44	2043.82
1325	1246.05	2082.48
1350	1246.16	2115.06
1375	1246.03	2120.23
1400	1251.89	2116.40
1425	1267.94	2112.43
1450	1283.76	2108.46
1475	1306.11	2104.77
1500	1338.81	2101.29

Berdasarkan grafik dan tabel diatas dapat dilihat bahwa rata-rata waktu respon dari kedua skenario mengalami peningkatan seiring dengan bertambahnya jumlah request. Rata-rata waktu respon pada Skenario 1 (LC-MAS) sedikit lebih besar dibandingkan Skenario 2 (LC) saat awal-awal mengeksekusi request. Perbandingan kedua skenario juga terbilang kecil dengan berkisar 200 ms-500 ms. Nilai waktu respon dari setiap request dapat dilihat pada Tabel 9 di bagian **Lampiran a**.

Namun ketika jumlah request yang dieksekusi menyentuh angka 600-an nilai rata-rata waktu respon pada Skenario 1 mengalami penurunan secara perlahan. Hal ini berbanding terbalik dengan nilai rata-rata waktu respon pada Skenario 2 yang semakin meningkat.

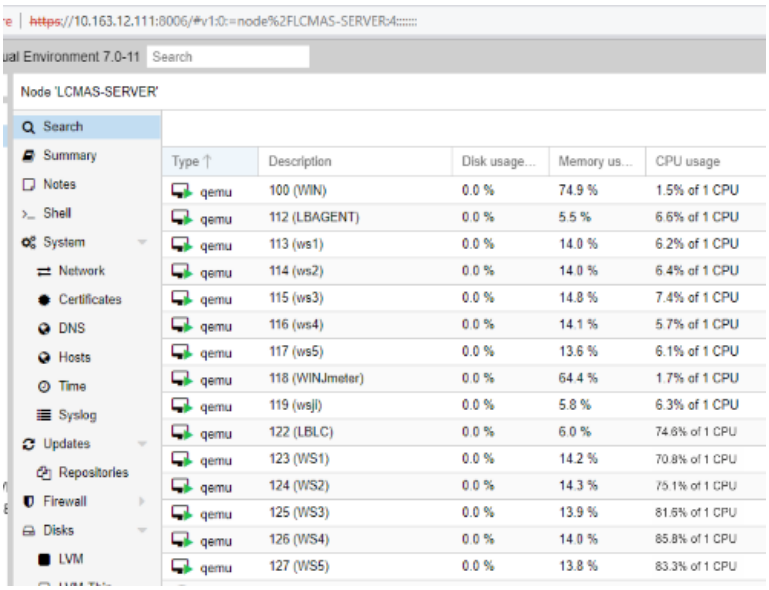
Berdasarkan hasil monitoring CPU yang dilakukan, penyebab nilai rata-rata waktu respon pada Skenario 1 sedikit lebih besar diawal pengujian yaitu terdapat aktivitas tambahan dari main agen dan reporting agen dimana setiap 500 ms keduanya saling berkomunikasi untuk mengumpulkan hasil monitoring resource mesin virtual. Aktivitas ini dapat menyebabkan sistem menjadi sangat sibuk sehingga mempengaruhi kondisi load yang berdampak pada jeda waktu terhadap antrian request selanjutnya untuk dieksekusi. Gambar 19 dan 20 memperlihatkan nilai penggunaan CPU dari kedua skenario. Kedua gambar ini merupakan salah satu dari 5 nilai CPU awal dan akhir yang didokumentasikan pada saat pengujian berlangsung. Dokumentasi 5 nilai awal dan akhir persentase CPU pada kedua skenario dapat dilihat pada **Lampiran b.**



Gambar 19 Persentase CPU Skenario 1 dan 2 di Awal Pengujian

Gambar diatas memperlihatkan persentase penggunaan CPU dari kedua skenario diawal pengujian. Dapat dilihat bahwa persentasi CPU untuk mesin virtual pada Skenario 2 hanya menggunakan sekitar 10 persen dimana angka tersebut jauh lebih kecil dibandingkan Skenario 1 yang langsung mencapai angka 60%. Hal ini

memperlihatkan bahwa saat request dalam jumlah yang besar masuk bersamaan, main agen dan reporting agen membutuhkan waktu lebih untuk beradaptasi dalam melakukan monitoring resource.



Gambar 20 Persentase CPU Skenario 1 dan 2 di Akhir Pengujian

Berdasarkan gambar diatas, Skenario 1 mengalami penurunan persentase CPU yang sangat drastis ketika seluruh request hampir selesai dieksekusi. Nilai rata-rata waktu respon hanya sekitar 7 persen diakhir pengujian. Hal ini menandakan main agen dan reporting agen mampu menstabilkan jumlah load agar mesin virtual tidak *overload* atau *idle*. Dapat dilihat juga bahwa ketika request terus bertambah dan ditingkatkan, kinerja dari kombinasi kedua algoritma dan metode pada Skenario 1 semakin baik dari sisi waktu respon. memperlihatkan perubahan load pada web server Skenario 1.

Berbanding terbalik dengan Skenario 2 dimana persentase penggunaan CPU dari seluruh mesin virtual berkisar 80%. Hal ini yang menyebabkan nilai waktu respon pada Skenario 2 semakin meningkat saat jumlah request semakin bertambah. Nilai rata-rata waktu respon pada Skenario 2 semakin besar karena pada skenario

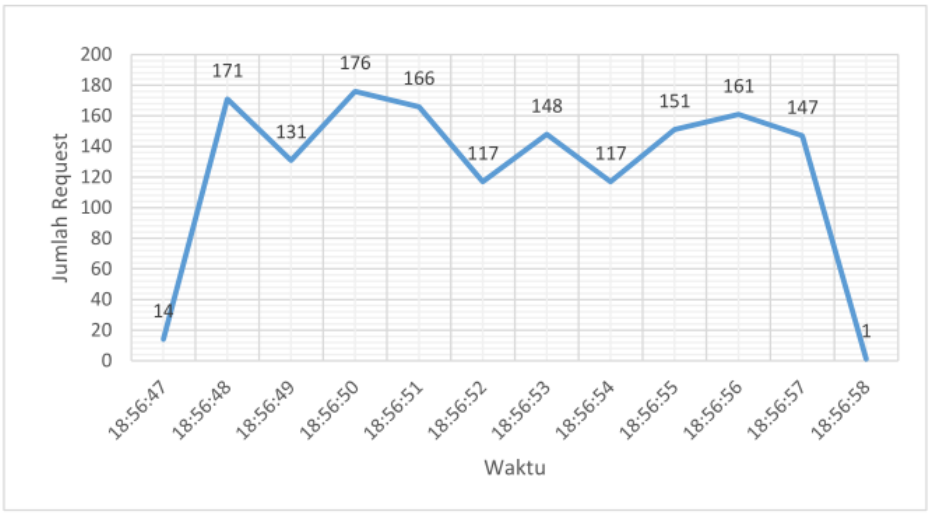
ini *load balancer* hanya mendistribusikan request berdasarkan jumlah koneksi backend server tanpa mempertimbangkan kondisi resources backend server tersebut apakah masih bisa menangani request atau harus diistirahatkan sehingga terjadi kelebihan beban pada backend server yang mempengaruhi waktu respon. Pendistribusian request pada Skenario 2 juga tidak memperhatikan mana web server yang overload dan mana web server yang idle.

4.2.2 Throughput

Pengujian throughput bertujuan untuk mengetahui berapa transaksi atau request yang dapat diproses oleh server dalam satuan waktu tertentu. Dalam penelitian ini, nilai throughput merepresentasikan jumlah request yang dapat diselesaikan dalam waktu 1 detik. Nilai throughput yang besar menandakan sistem bekerja dengan baik. Throughput dihitung sebagai total request yang masuk dibagi total waktu eksekusi. Satuan waktu dihitung dari awal request pertama dikirimkan hingga akhir request terakhir selesai diseksekusi, termasuk waktu jeda tiap request. Gambar dibawah ini menunjukkan grafik jumlah request yang ditangani masing-masing skenario dalam satu detik.

A. 1500 Request

   Waktu ramp-up yang diatur untuk mengirimkan 1500 request ke sistem adalah 10 detik dimana interval dari setiap request adalah 0.007 detik (10/1500). Gambar 21 memperlihatkan grafik jumlah request yang mampu diselesaikan oleh backend server setiap 1 detik pada Skenario 1.



Gambar 21 Grafik Throughput Skenario 1

Distribusi request pada Skenario 1 untuk 1500 request dimulai pada jam 18:56:47 dan berakhir pada jam 18:56:58. Waktu eksekusi yang dibutuhkan adalah 12 detik. Gambar 22 memperlihatkan grafik jumlah request yang mampu diselesaikan oleh backend server setiap 1 detik pada Skenario 2.

Gambar 22 Grafik Throughput Skenario 2

Distribusi request pada Skenario 2 untuk 1500 request dimulai pada jam 18:43:06 dan berakhir pada jam 18:43:11. Waktu eksekusi yang dibutuhkan adalah 6 detik.

Berdasarkan grafik diatas dapat dilihat waktu eksekusi Skenario 1 jauh lebih lama dibandingkan dengan Skenario 2. Hal ini dipengaruhi oleh jumlah

request yang mampu diselesaikan dalam 1 detik. Backend server pada Skenario 2 langsung "dipaksa" menyelesaikan lebih banyak request dalam 1 detik. Pembagian request yang tidak merata tersebut disebabkan oleh proses routing request yang tidak mempertimbangkan kondisi load server dimana request hanya dibagi secara terus menerus berdasarkan jumlah koneksi aktif. Backend server yang menangani request menjadi kewalahan karena adanya potensi hanya 1 node yang bekerja menangani seluruh request tersebut di detik pertama. Hal ini jauh berbeda dengan Skenario 1 dengan tetap mempertahankan stabilitas jumlah request yang ditangani sampai waktu eksekusi selesai. Pembagian request yang tidak merata berdampak pada nilai waktu respon dan mengakibatkan beberapa request gagal dieksekusi sehingga memperbesar nilai *error*.

Nilai throughput pada Skenario 2 lebih besar dari Skenario 1. Hal ini menujukkan bahwa backend server pada Skenario 2 lebih tangguh dalam menangani request sekali eksekusi. Backend server dapat menangani request yang lebih banyak sehingga waktu eksekusi jauh lebih singkat dibandingkan Skenario 1. Namun, besarnya nilai throughput dapat mempengaruhi kualitas waktu respon dari setiap request.

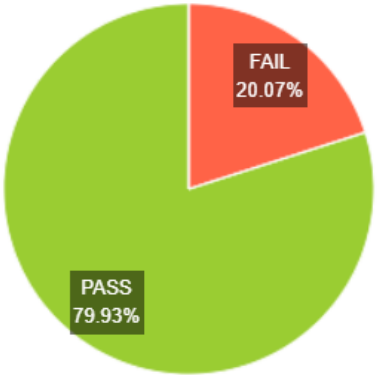
Hasil pengujian parameter throughput ini menunjukkan kemampuan server dari kedua skenario sangat baik ketika menangani request yang semakin meningkat seiring dengan bertambahnya jumlah request yang diuji. Hal ini membuktikan bahwa keberadaan mekanisme load balancing telah berhasil mendistribusikan permintaan secara merata tanpa salah satu dari server yang kelebihan beban.

4.2.3 Presentasi Error

Pengujian parameter error merupakan pengujian yang dilakukan untuk memeriksa apakah sistem dapat menangani berbagai error yang mungkin terjadi dikemudian hari. Parameter ini mengacu pada jumlah request yang gagal atau tidak diproses oleh sistem. Pengujian ini penting dilakukan karena dapat mengukur seberapa besar kesalahan yang dihasilkan oleh metode dan algoritma yang diterapkan dalam sistem tersebut untuk dipertimbangkan kembali sebelum membuat keputusan apakah sistem tersebut cocok digunakan atau tidak. Nilai error didapatkan dari jumlah request yang gagal dieksekusi dibagi dengan total request dikali 100%. Berikut analisa dari penyebab terjadinya error dari kedua skenario.

A. Skenario 1 (LC-MAS 1500 request)

Jumlah request yang gagal dieksekusi pada Skenario ini yaitu sebanyak 301 request dari total 1500 request dengan tipe error 502 (Bad Gateway). Gambar 23 menunjukkan grafik nilai error dari Skenario 1 saat mengeksekusi 1500 request diambil dari dashboard Apache Jmeter.



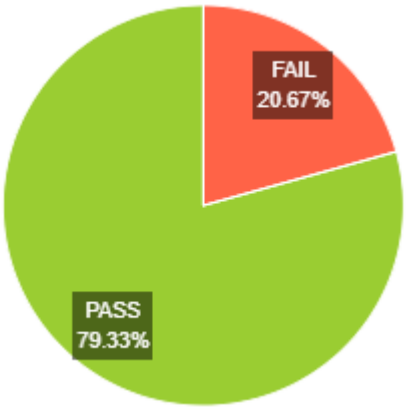
Gambar 23 Grafik Persentase Error Skenario 1

Error ini cukup sering terjadi pada protokol HTTP. Hal ini disebabkan oleh beberapa kondisi, seperti server yang kelebihan beban karena banyaknya request yang mencoba mengakses aplikasi website secara bersamaan serta

aktivitas agen yang mempengaruhi penggunaan CPU dan Memori yang membuat sistem menjadi sibuk. Overload pada server juga mempengaruhi waktu respon.

B. Skenario 2 (LC 1500 request)

Jumlah request yang gagal dieksekusi pada Skenario ini yaitu sebanyak 310 request dari total 1500 request dengan tiga tipe error. Gambar 24 menunjukkan grafik nilai error dari Skenario 2 saat mengeksekusi 1500 request diambil dari dashboard Apache Jmeter.



Gambar 24 Grafik Persentase Error Skenario 2

Tabel 8 menunjukkan jenis error pada Skenario 2.

Tabel 8 Jenis Error Skenario 2

No	Error Message	Number of Errors	% Error
1	Non HTTP response code: org.apache.http.NoHttpResponseException/ Non HTTP response message: 10.163.12.122:80 failed to respond	268	17.87%
2	Non HTTP response code: java.net.SocketException/ Non HTTP response message: Connection reset	39	2.60%
3	Non HTTP response code: java.net.SocketException/ Non HTTP response message: Software caused connection abort: recv failed	3	0.20%
TOTAL		**310**	**20.67%**

Beberapa faktor penyebab terjadinya error pada Skenario 2 adalah:

1. Pesan error pertama menunjukkan bahwa alamat IP load balancer tidak dapat merespon. Hal tersebut terjadi karena server sangat sibuk melayani request yang masuk. Load balancer juga harus menghitung jumlah koneksi aktif terkecil dari backend server yang bekerja dibelakangnya sehingga server menjadi sibuk dan tidak dapat merespon permintaan dengan baik.
2. Pesan error ini disebabkan oleh Apache Jmeter gagal dalam menerima respon dari request yang dikirim. Waktu koneksi habis saat request sedang diproses sehingga perlu diulang untuk menyelesaikan seluruh permintaan.
3. Pesan error ini menunjukkan bahwa koneksi jaringan tidak stabil. Pesan ini mengisyaratkan bahwa server tidak menunggu data apapun dari klien. Tipe ini jarang terjadi dan tidak dapat diprediksi. Dalam beberapa kasus, me-reboot atau memulai ulang sistem adalah salah satu opsi untuk mengatasi masalah ini.

BAB V
KESIMPULAN DAN SARAN

V.1 Kesimpulan

Setelah melakukan perancangan dan pengembangkan mekanisme load balancing untuk mendistribusikan beban request pada web server di lingkungan virtual agar mampu memberikan kinerja website yang lebih tinggi, maka berdasarkan hasil perancangan dan pengembangan serta analisis yang dilakukan pada penelitian ini, maka diperoleh kesimpulan sebagai berikut:

1. Mekanisme *load balancing* yang dirancang dan dikembangkan berhasil mendistribusikan permintaan atau *request* dari sebuah *website* secara merata dan efisien ke masing-masing *web server* di lingkungan virtualisasi dengan menerapkan algoritma *Least Connection* dan metode Multi-Agent System, sehingga kedepannya dapat diimplementasikan untuk memenuhi kebutuhan dari *website* yang masih membutuhkan kinerja yang lebih tinggi. Mekanisme ini juga dapat dimanfaatkan untuk mengatasi salah satu permasalahan yang sering terjadi pada data center, yaitu kelebihan beban kerja pada *server*.
2. Mekanisme *load balancing* yang dikembangkan menggunakan kombinasi algoritma *Least Connection* dan metode Multi-Agent System (LC-MAS) memperoleh kinerja yang lebih baik dibandingkan dengan algoritma *Least Connection* (LC), dengan rata-rata waktu respon sebesar 1338.8 ms, 20.07 % *error*, dan nilai *throughput* sebesar 125 tps saat menangani 1500 *request*. Dari hasil pengujian Skenario 2 (LC) memang menghasilkan rata-rata waktu respon yang jauh lebih kecil dibandingkan dengan Skenario 1 (LC-MAS) karena adanya proses pengecekan *resources* setiap 500 ms sehingga

membutuhkan waktu yang lebih lama untuk web server memberikan respon namun ketika jumlah *request* semakin bertambah, rata-rata waktu respon sangat dipengaruhi. Nilai rata-rata waktu respon ini berpengaruh pada persentase *error* yang dihasilkan karena pada skenario LC-MAS, *request* diproses sampai selesai meskipun membutuhkan waktu lebih, lain halnya dengan skenario LC yang tidak dapat memproses banyak *request* sehingga persentase error yang dihasilkan lebih besar dibandingkan LC-MAS.

3. Penggunaan metode Multi-Agent System menghasilkan waktu respon yang lebih merata dibandingkan tanpa Mult-Agent System. Hal ini dapat dilihat dari tabel waktu respon setiap *request* yang stabil tanpa mengalami lonjakan atau penurunan nilai yang sangat jauh. Selain itu penggunaan metode ini sangat berpengaruh pada distribusi *request*, dimana dengan metode ini setiap *backend server* yang bekerja memperoleh jumlah *request* yang merata tanpa ada salah satu yang menangani banyak *request* sendirian. Dengan metode Multi-Agent System, kontrol *resources* dari *backend server* juga dapat dilakukan sehingga meminimalisir terjadinya kondisi *idle* atau *overload* pada *server*.

V.2 Saran

Adapun saran untuk pengembangan penelitian ini selanjutnya antara lain:

1. Menerapkan *load balancing* pada lingkungan *cloud computing* dengan memanfaatkan beberapa provider seperti AWS, GCP, dan semacamnya.
2. Menambah skenario pengujian seperti penambahan *server database* atau *storage* yang dipisah agar lebih kompleks.

3. Menambah parameter seperti *scalability*, *failover*, dan parameter pengukuran lainnya untuk lebih mempertajam kinerja *load balancing*.
4. Mengembangkan algoritma lain untuk diterapkan pada arsitektur *load balancing* guna memperoleh kinerja yang lebih baik.

DAFTAR PUSTAKA

[1] M. Aggarwal, "Dynamic Load Balancing Based on CPU Utilization and Data Locality in Distributed Database Using Priority Policy," *Int. Conf. Softw. Technol. Eng. Dyn.*, pp. 388–391, 2010.

[2] M. R. M. Bella, M. Data, and W. Yahya, "Web Server Load Balancing Based On Memory Utilization Using Docker Swarm," *3rd Int. Conf. Sustain. Inf. Eng. Technol. SIET 2018 - Proc.*, pp. 220–223, 2018.

[3] I. Rijayana, "Teknologi Load Balancing Untuk Mengatasi Beban Server," *Seminar Nasional Aplikasi Teknologi Informasi*, vol. 2005, no. Snati. pp. 79–75, 2005.

[4] L. H. Pramono, R. C. Buwono, and Y. G. Waskito, "Round-robin Algorithm in HAProxy and Nginx Load Balancing Performance Evaluation a Review," *J. Chem. Inf. Model.*, vol. 53, no. 9, pp. 1689–1699, 2018.

[5] K. D. Patel and T. M. Bhalodia, "An efficient dynamic load balancing algorithm for virtual machine in cloud computing," *2019 Int. Conf. Intell. Comput. Control Syst. ICCS 2019*, no. Iciccs, pp. 145–150, 2019.

[6] M. Faizal Afriansyah, M. Somantri, and M. Agus Riyadi, "Model of Load Balancing Using Reliable Algorithm with Multi-Agent System," in *International Conference on Recent Trends in Physics (ICRTP 2016)*, 2016, vol. 755, no. 1.

[7] P. Geetha and C. R. R. Robin, "A comparative-study of load-cloud balancing algorithms in cloud environments," in *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing, ICECDS 2017*, 2017, pp. 806–810.

[8] X. J. Shen *et al.*, "Achieving dynamic load balancing through mobile agents in small world P2P networks," *Comput. Networks*, vol. 75, no. PartA, pp. 134–148, 2014.

[9] O. AbdElRahem, A. M. Bahaa-Eldin, and A. Taha, "Virtualization Security: A Survey," in *2016 11th International Conference on Computer Engineering & Systems (ICCES)*, 2016, pp. 473–490.

[10] J. Sahoo, S. Mohapatra, and R. Lath, "Virtualization: A survey on concepts, taxonomy and associated security issues," in *2nd International Conference on Computer and Network Technology, ICCNT 2010*, 2010, pp. 222–226.

[11] A. Agarwal, R. Luniya, M. Bhatnagar, M. Gaikwad, and V. Inamdar, "Reviewing the world of virtualization," in *Proceedings - 3rd International Conference on Intelligent Systems Modelling and Simulation, ISMS 2012*, 2012, pp. 554–557.

[12] A. B. M. Moniruzzaman, M. Waliullah, and M. S. Rahman, "A high availability clusters model combined with load balancing and shared storage technologies for web servers," *Int. J. Grid Distrib. Comput.*, vol. 5, no. 12, pp. 109–120, 2015.

[13] N. Guna, “IMPLEMENTASI LOAD BALANCING DENGAN ALGORITMA LEAST CONNECTION MENGGUNAKAN DIGITALOCEAN LOAD BALANCERS,” 2020.

[14] S. L. Chen, Y. Y. Chen, and S. H. Kuo, “CLB: A novel load balancing architecture and algorithm for cloud services,” *Comput. Electr. Eng.*, vol. 58, pp. 154–160, 2017.

[15] M. Xu, W. Tian, and R. Buyya, “A survey on load balancing algorithms for virtual machines placement in cloud computing,” *Concurr. Comput.* , vol. 29, no. 12, pp. 1–16, 2017.

[16] D. H. Youm, “Load Balancing Strategy using Round Robin Algorithm,” *Asia-pacific J. Converg. Res. Interchang.*, vol. 2, no. 3, pp. 1–10, 2016.

[17] L. Zhu, J. Cui, and G. Xiong, “Improved dynamic load balancing algorithm based on Least-Connection Scheduling,” in *Proceedings of 2018 IEEE 4th Information Technology and Mechatronics Engineering Conference, ITOEC 2018*, 2018, no. Itoec, pp. 1858–1862.

[18] S. Saranya and D. . Manicka Chezian, “Comparative analysis on load balancing techniques in cloud computing,” *Indian J. Sci. Technol.*, vol. 3, no. 10, pp. 352–357, 2016.

[19] A. Bone, “Multi-Agent System sebagai solusi pembangunan perangkat lunak dalam menjamin keberlangsungan hidup perangkat lunak agent input environment,” *J. Apl. Teknol. Inf.*, vol. 16, no. 1, pp. 6–11, 2005.

[20] M. A. Metawei, S. A. Ghoneim, S. M. Haggag, and S. M. Nassar, “Load balancing in distributed multi-agent computing systems,” *Ain Shams Eng. J.*, vol. 3, no. 3, pp. 237–249, 2012.

LAMPIRAN

Lampiran a.

Tabel **9** Waktu Respon 1500 Request

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
1	288	592
2	302	88
3	304	189
4	352	196
5	345	197
6	347	203
7	351	98
8	404	114
9	432	99
10	409	217
11	97	224
12	279	115
13	115	136
14	502	131
15	377	234
16	374	239
17	425	148
18	428	273
19	173	130
22	415	289
23	420	300
24	440	160
25	487	300
26	493	320
27	490	194
28	246	174
29	251	335
30	246	199
31	555	182
32	554	188
33	719	358
34	308	356
35	295	361
36	305	393
37	614	398
38	614	367
39	619	600
40	353	607
41	358	297
42	352	296
43	677	276
44	680	304
45	678	627
46	103	294
47	99	313
48	410	643
49	408	643
50	416	666
51	126	309
52	743	317
53	746	298
54	742	303
55	469	676
56	174	690
57	468	689
58	161	693
59	478	707
60	172	367
61	221	722
62	217	736
63	822	740

20	170	156
21	168	138

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
64	829	338
65	827	754
66	532	750
67	532	371
68	527	376
69	230	374
70	884	771
71	270	775
72	277	774
73	285	391
74	910	401
75	910	417
76	909	407
77	603	423
78	607	839
79	605	847
80	614	428
81	923	256
82	344	261
83	339	267
84	354	854

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
87	716	870
88	395	456
89	385	299
90	723	299
91	411	291
92	437	462
93	446	892
94	799	305
95	802	332
96	799	340
97	808	338
98	466	315
99	499	331
100	495	355
101	114	332
102	888	346
103	103	320
104	112	321
105	894	341
106	897	318
107	531	391
108	896	369
109	565	349

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
110	565	362
111	158	327
112	166	335
113	154	331
114	591	387
115	968	401
116	974	359
117	973	381
118	985	405
119	624	370
120	606	396
121	631	343
122	231	401
123	218	394
124	226	405
125	1056	414
126	1055	367
127	1053	395
128	680	382
129	681	402
130	1075	403
131	740	398
132	313	385

85	708	439
86	711	270

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
133	299	402
134	305	433
135	332	437
136	1156	451
137	1152	455
138	1157	428
139	781	419
140	787	440
141	1171	411
142	815	445
143	383	441
144	392	414
145	379	418
146	829	452
147	376	456
148	1226	409
149	1225	407
150	1231	463
151	856	411
152	865	461
153	1266	427

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
156	453	474
157	456	447
158	900	472
159	101	465
160	96	454
161	1312	471
162	1318	506
163	1317	485
164	93	469
165	468	468
166	936	479
167	933	495
168	1361	483
169	160	497
170	159	503
171	966	525
172	533	520
173	161	502
174	523	533
175	531	515
176	220	541
177	236	509
178	1042	548

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
179	243	507
180	597	547
181	603	498
182	590	511
183	1487	554
184	1063	506
185	1092	556
186	1087	530
187	1085	524
188	645	533
189	1519	589
190	1523	541
191	1526	539
192	620	559
193	270	634
194	264	640
195	1079	567
196	1071	597
197	267	609
198	1522	599
199	1545	646
200	1539	601
201	1579	607

154	888	500
155	447	507

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
202	677	610
203	693	599
204	688	640
205	1136	662
206	1142	683
207	350	704
208	346	731
209	1601	737
210	350	678
211	717	719
212	1161	708
213	1164	744
214	1653	721
215	1635	640
216	1655	655
217	765	687
218	758	664
219	765	654
220	1213	693
221	1662	727
222	1223	662

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
225	65	657
226	802	760
227	425	765
228	1230	755
229	1241	783
230	1630	767
231	77	756
232	76	743
233	482	781
234	846	783
235	843	786
236	838	659
237	1288	759
238	1679	811
239	1684	1685
240	1683	1286
241	116	696
242	1298	689
243	505	1365
244	497	1387
245	878	1682
246	1713	1342
247	1314	1679

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
248	1308	1378
249	515	1685
250	148	669
251	141	685
252	1751	681
253	911	711
254	919	678
255	908	700
256	176	710
257	558	1746
258	1367	725
259	1374	729
260	582	710
261	576	681
262	203	724
263	199	665
264	1794	709
265	1798	1762
266	1797	1764
267	1383	1466
268	1376	726
269	961	711
270	596	753

223	434	634
224	429	638
Request Ke-	**Waktu Respon (ms)**	
	Skenario 1	**Skenario 2**
271	236	705
272	1846	716
273	252	728
274	260	1501
275	644	1773
276	996	752
277	1019	730
278	1017	742
279	1877	760
280	644	733
281	562	745
282	1885	744
283	1461	734
284	1884	749
285	1877	765
286	1466	709
287	307	1572
288	548	689
289	1480	715
290	1485	740
291	1069	712

Request Ke-	**Waktu Respon (ms)**	
	Skenario 1	**Skenario 2**
294	602	751
295	1115	734
296	1095	787
297	612	816
298	1111	742
299	1103	749
300	618	743
301	370	742
302	1565	760
303	387	766
304	644	773
305	381	765
306	1575	763
307	1573	781
308	1582	770
309	1604	782
310	1172	769
311	697	776
312	431	781
313	707	780
314	709	1721
315	436	1719
316	431	756

Request Ke-	**Waktu Respon (ms)**	
	Skenario 1	**Skenario 2**
317	1195	785
318	1184	767
319	1189	775
320	1204	2003
321	739	805
322	1659	786
323	486	822
324	1674	818
325	1669	847
326	1664	777
327	497	818
328	493	817
329	783	839
330	1672	852
331	789	854
332	793	866
333	1271	858
334	1289	836
335	1294	811
336	1287	814
337	1300	823
338	833	800
339	1737	828

292	323	1875
293	333	1864

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
340	562	810
341	1736	829
342	1730	1837
343	1724	842
344	571	828
345	565	867
346	879	832
347	878	866
348	882	863
349	627	893
350	923	879
351	1379	875
352	1385	891
353	1395	895
354	1373	2160
355	1383	883
356	643	861
357	647	900
358	629	895
359	970	899
360	968	918

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
363	741	901
364	741	895
365	1502	907
366	1478	872
367	1490	897
368	1484	909
369	1502	920
370	746	911
371	740	924
372	1086	915
373	1081	917
374	1086	915
375	1505	914
376	792	2276
377	1146	2095
378	816	2275
379	804	2112
380	803	2036
381	1190	2017
382	1190	921
383	1194	915
384	1194	919
385	1194	948

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
386	1194	2095
387	1194	2060
388	1229	2299
389	1229	2295
390	876	2295
391	876	2128
392	876	2302
393	887	965
394	890	951
395	893	945
396	1257	1011
397	1257	1042
398	1257	1050
399	1257	930
400	1257	1018
401	1289	1028
402	1292	925
403	1287	985
404	1287	959
405	1287	973
406	1287	1075
407	1287	979
408	962	940

361	970	849
362	1069	905
Request Ke-	**Waktu Respon (ms)**	
	Skenario 1	**Skenario 2**
409	962	976
410	962	961
411	962	966
412	969	937
413	973	973
414	969	950
415	1336	960
416	1336	991
417	1336	1109
418	1336	995
419	1361	1105
420	1361	961
421	1361	1123
422	1361	1103
423	1361	1003
424	1361	1009
425	1403	1132
426	1415	1162
427	1402	994
428	1069	1016
429	1064	999

Request Ke-	**Waktu Respon (ms)**	
	Skenario 1	**Skenario 2**
432	1058	1199
433	1058	905
434	1058	1190
435	1058	1027
436	1058	912
437	1058	1170
438	1058	917
439	1058	895
440	1058	915
441	1058	900
442	1058	899
443	1058	894
444	1441	915
445	1441	904
446	1441	900
447	1441	954
448	1441	919
449	1441	945
450	1441	949
451	1441	1229
452	1441	960
453	1441	914
454	1441	1232

Request Ke-	**Waktu Respon (ms)**	
	Skenario 1	**Skenario 2**
455	1441	935
456	1441	947
457	1441	926
458	1479	1277
459	1479	1290
460	1479	1276
461	1479	964
462	1150	967
463	1503	922
464	1147	917
465	1502	1314
466	1120	1313
467	1498	1300
468	1144	1000
469	1144	987
470	1144	973
471	1144	1000
472	1144	1015
473	1144	942
474	1144	953
475	1144	1008
476	1144	942
477	1144	942

430	1069	875
431	1058	1141
Request Ke-	**Waktu Respon (ms)**	
	Skenario 1	**Skenario 2**
478	1561	1006
479	1561	941
480	1561	1000
481	1561	1017
482	1207	997
483	1194	1012
484	1197	989
485	1207	1012
486	1589	1010
487	1589	1003
488	1613	1019
489	1616	1019
490	1613	2969
491	1613	2969
492	1613	2975
493	1613	2975
494	1626	1043
495	1626	1049
496	1619	1028
497	1619	1025
498	1619	3001

Request Ke-	**Waktu Respon (ms)**	
	Skenario 1	**Skenario 2**
501	1619	3006
502	1266	3012
503	1282	3012
504	1276	3017
505	1290	1040
506	1290	1063
507	1290	1028
508	1290	1044
509	1290	1027
510	1290	1061
511	1290	1044
512	1290	1023
513	1290	1065
514	1290	1047
515	1290	1034
516	1290	1048
517	1343	1048
518	1354	3053
519	1372	3053
520	1378	1039
521	1378	1046
522	1378	1051
523	1378	1051

Request Ke-	**Waktu Respon (ms)**	
	Skenario 1	**Skenario 2**
524	1378	3070
525	1378	3070
526	1378	3075
527	1428	3075
528	1428	3080
529	1428	1074
530	1428	3086
531	1428	3086
532	1428	1063
533	1428	1088
534	1447	3105
535	1442	3105
536	1431	1060
537	1457	1067
538	1457	1056
539	1457	1056
540	1457	3121
541	1457	3121
542	1457	3126
543	1457	1093
544	1457	1125
545	1457	1099
546	1457	1129

499	1619	3001
500	1619	3006

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
547	1457	1172
548	1525	1148
549	1546	1148
550	1545	1116
551	1542	1147
552	1538	1164
553	1619	1150
554	1631	1143
555	1640	1192
556	1627	1149
557	1640	1161
558	1706	1165
559	1704	1127
560	1711	1206
561	1721	1159
562	1700	1159
563	95	1158
564	89	1139
565	111	1146
566	107	1145
567	1794	1176

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
570	1793	126
571	1800	126
572	208	3270
573	204	3270
574	199	3276
575	202	3276
576	1895	1184
577	1884	1162
578	1871	1178
579	1878	1168
580	1893	3292
581	277	3292
582	1957	3295
583	332	3295
584	1962	3302
585	1972	1161
586	342	1143
587	321	1169
588	1953	1157
589	67	1182
590	355	3315
591	2010	1156
592	364	1175

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
593	374	1187
594	383	1179
595	369	1162
596	363	3329
597	2045	3329
598	2035	1210
599	2057	1217
600	2044	1194
601	2031	1170
602	103	1159
603	111	1159
604	108	3359
605	104	3359
606	455	3364
607	461	3364
608	449	3368
609	458	3368
610	445	3371
611	2134	3371
612	2106	3376
613	2129	3376
614	2123	3381
615	2107	3381

568	1781	1137
569	1777	3260

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
616	187	3386
617	186	1247
618	183	3392
619	181	1193
620	545	3396
621	537	1194
622	524	1243
623	546	1246
624	533	1249
625	283	1249
626	283	3423
627	282	3423
628	282	1236
629	2201	1201
630	2213	1248
631	2227	1236
632	2220	3440
633	2235	3440
634	639	3447
635	619	3447
636	627	3451

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
639	402	1259
640	380	1216
641	407	1225
642	376	3465
643	2284	3465
644	2293	1249
645	2305	1235
646	2318	1276
647	2304	1274
648	704	3479
649	698	3479
650	714	3484
651	720	3484
652	729	3488
653	463	1241
654	79	1274
655	91	1269
656	88	1274
657	489	1266
658	493	1268
659	493	1268
660	2390	3507
661	492	3507

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
662	2400	3512
663	2377	3512
664	2384	3520
665	2366	3520
666	146	3524
667	140	3524
668	134	3531
669	807	3531
670	798	3538
671	805	3538
672	809	3542
673	817	3542
674	581	3547
675	2460	1306
676	2475	3553
677	2460	1304
678	2471	1301
679	2460	1301
680	595	3562
681	598	1317
682	590	1304
683	186	1314
684	200	3571

637	633	1265
638	648	1239
Request Ke-	**Waktu Respon (ms)**	
	Skenario 1	**Skenario 2**
685	584	3571
686	195	3576
687	250	3576
688	894	3581
689	241	1316
690	912	1308
691	891	1312
692	252	1310
693	907	1310
694	887	3595
695	670	3595
696	2546	1300
697	2578	1286
698	2578	1306
699	2563	1323
700	2560	1280
701	678	1280
702	698	3614
703	691	3614
704	686	3618
705	314	3618

Request Ke-	**Waktu Respon (ms)**	
	Skenario 1	**Skenario 2**
708	991	1335
709	969	1325
710	979	3640
711	1001	1345
712	1009	1332
713	352	1332
714	367	1311
715	374	1351
716	2672	3653
717	2671	1362
718	2671	3659
719	2670	3659
720	790	3664
721	2665	3664
722	771	3668
723	785	3668
724	775	3673
725	794	3673
726	408	3678
727	421	3678
728	417	3683
729	2758	3683
730	90	3688

Request Ke-	**Waktu Respon (ms)**	
	Skenario 1	**Skenario 2**
731	2753	3688
732	2663	3694
733	2776	3694
734	81	3697
735	2771	3697
736	75	3703
737	875	3703
738	474	1396
739	856	1390
740	870	1383
741	470	1383
742	887	1345
743	474	1345
744	866	1355
745	144	1355
746	130	1411
747	138	1411
748	556	3885
749	552	3885
750	547	1368
751	2768	1432
752	2773	1393
753	2776	1382

706	317	1314
707	310	1320

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
754	2774	1391
755	2778	1390
756	937	1399
757	961	1362
758	947	1369
759	969	1369
760	955	1383
761	178	1416
762	186	1410
763	194	1405
764	603	3934
765	617	3934
766	613	3939
767	627	1403
768	245	1402
769	233	1401
770	240	1393
771	2879	1393
772	2882	1394
773	2881	1394
774	2880	3956

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
777	1055	1458
778	1037	1451
779	1030	1455
780	1048	1447
781	726	1449
782	736	1460
783	722	1454
784	319	1437
785	333	1435
786	742	1447
787	338	1450
788	3008	1456
789	3011	1295
790	3013	1332
791	3017	1416
792	3019	4038
793	1117	4038
794	1130	4060
795	1138	4060
796	1153	1484
797	1146	3669
798	368	1430
799	367	1456

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
800	366	4071
801	769	4071
802	766	4076
803	764	3679
804	765	1494
805	428	1493
806	436	1439
807	438	1439
808	3100	4087
809	3115	4087
810	3109	4092
811	3125	4092
812	841	4096
813	3099	4096
814	844	4101
815	837	3702
816	831	4105
817	1207	4105
818	1228	4110
819	1222	4110
820	1237	1529
821	1217	1529
822	490	4118

775	2887	3956
776	1047	1465
Request Ke-	**Waktu Respon (ms)**	
	Skenario 1	**Skenario 2**
823	484	4118
824	499	4122
825	913	4122
826	900	4128
827	907	4128
828	895	4128
829	3197	4128
830	3191	4128
831	3186	4128
832	3206	4128
833	3180	1527
834	1306	1396
835	1321	1522
836	1315	1522
837	1302	1522
838	1296	3749
839	546	3749
840	560	2165
841	568	2165
842	963	3762
843	959	1560

Request Ke-	**Waktu Respon (ms)**	
	Skenario 1	**Skenario 2**
846	3218	1415
847	3233	1235
848	3254	1231
849	3226	1419
850	3160	3775
851	639	1411
852	631	1411
853	615	1411
854	1391	1411
855	1407	1411
856	1399	1411
857	1381	1261
858	1388	3812
859	663	1440
860	1051	3803
861	1065	1447
862	1047	1458
863	1062	3811
864	1464	3823
865	1453	3823
866	1461	3823
867	733	3823
868	729	3823

Request Ke-	**Waktu Respon (ms)**	
	Skenario 1	**Skenario 2**
869	737	3823
870	3271	3823
871	3264	3823
872	3171	3823
873	3276	3823
874	3171	3823
875	748	3847
876	1126	3847
877	1115	3847
878	1132	3847
879	1125	3847
880	797	3847
881	812	3847
882	804	3847
883	3279	3847
884	3184	3847
885	3281	3847
886	3179	3847
887	3172	3847
888	821	1510
889	1189	3870
890	1197	1320
891	1177	1320

844	966	3765
845	981	1540

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
892	1185	1320
893	877	1320
894	869	1320
895	879	1337
896	3198	1337
897	3187	1337
898	3183	1337
899	3198	3908
900	3279	3905
901	901	1528
902	1270	3912
903	1278	3907
904	1258	3908
905	1265	3912
906	934	3903
907	949	1330
908	945	1343
909	3189	3902
910	3220	3902
911	984	3902
912	3200	3902

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
915	1358	3902
916	1350	1378
917	1334	3942
918	1358	1368
919	1345	1373
920	1020	1383
921	1014	1382
922	1009	3955
923	3231	3951
924	3219	3951
925	3227	3951
926	1091	3951
927	3174	3951
928	3237	3951
929	1440	3951
930	1454	3951
931	1427	3951
932	1445	3951
933	1464	3951
934	1464	3951
935	1464	1391
936	1464	1391
937	1464	1391

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
938	1464	1391
939	1124	1391
940	1110	1391
941	1119	4000
942	1119	4000
943	1119	4000
944	1119	4000
945	1119	4000
946	1119	4000
947	1163	1424
948	3222	4014
949	3231	1430
950	3218	1430
951	3212	1430
952	3137	1430
953	3137	1430
954	3137	1430
955	3137	1430
956	3137	1430
957	1125	1436
958	1127	4045
959	1191	4040
960	1191	4028

913	3215	3902
914	3208	3902

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
961	1191	4041
962	1191	4032
963	1191	4048
964	1191	1406
965	1191	1442
966	1191	4047
967	1191	1435
968	1189	1456
969	1189	1456
970	1189	1456
971	1189	1456
972	1217	1456
973	1216	1456
974	1216	1456
975	1216	4074
976	1216	1443
977	1216	4080
978	1216	4066
979	1216	1481
980	1216	1463
981	1216	1461

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
984	1216	4091
985	1308	4091
986	1308	4091
987	1308	4091
988	1308	4091
989	1330	4091
990	1333	4091
991	1335	4091
992	1335	4091
993	1335	4091
994	1335	4091
995	1335	4091
996	1335	4091
997	1335	4181
998	1335	1468
999	1335	1515
1000	1335	1519
1001	1335	4177
1002	1335	4183
1003	1335	1531
1004	1427	1564
1005	1427	4186
1006	1427	1554

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
1007	1427	4189
1008	1427	4182
1009	1427	1559
1010	1437	1568
1011	1456	1545
1012	1454	1543
1013	1454	1558
1014	1454	4212
1015	1454	4212
1016	1454	1542
1017	1454	4214
1018	1454	1506
1019	1454	4195
1020	1454	1512
1021	1454	4197
1022	1454	4197
1023	1454	4197
1024	1454	4197
1025	1536	4197
1026	1541	4197
1027	1541	4197
1028	1541	4197
1029	1543	4322

982	1216	4091
983	1216	4091
Request Ke-	**Waktu Respon (ms)**	
	Skenario 1	**Skenario 2**
1030	1551	4275
1031	1541	1557
1032	1541	4254
1033	1541	4328
1034	1541	4328
1035	1541	4352
1036	1626	1558
1037	1619	1580
1038	1619	1563
1039	1619	4349
1040	1619	4356
1041	1619	4358
1042	1619	4354
1043	1619	1585
1044	1628	1595
1045	1622	4294
1046	1622	4294
1047	1634	4294
1048	1634	4294
1049	1634	4294
1050	1634	4294

Request Ke-	**Waktu Respon (ms)**	
	Skenario 1	**Skenario 2**
1053	1634	4294
1054	1634	1612
1055	1634	1612
1056	1634	1610
1057	1634	1609
1058	1634	1628
1059	1634	4320
1060	1634	1604
1061	1634	1615
1062	1715	4398
1063	1706	4395
1064	1706	4395
1065	1714	4395
1066	1723	4395
1067	1710	4395
1068	1710	4395
1069	1710	4395
1070	1710	4395
1071	1710	4395
1072	1710	4395
1073	1710	4395
1074	1710	4395
1075	1710	4395

Request Ke-	**Waktu Respon (ms)**	
	Skenario 1	**Skenario 2**
1076	1710	4395
1077	1710	4395
1078	1710	4395
1079	1710	4395
1080	1710	4395
1081	1710	4395
1082	1710	4455
1083	1710	4380
1084	1828	4460
1085	1816	4456
1086	1822	1666
1087	1818	1666
1088	1830	1666
1089	1830	4483
1090	1830	1649
1091	1830	4485
1092	1830	1661
1093	1830	4483
1094	1830	1674
1095	1830	1673
1096	1830	4488
1097	1830	1678
1098	1830	4490

1051	1634	4294
1052	1634	4294
Request Ke-	**Waktu Respon (ms)**	
	Skenario 1	**Skenario 2**
1099	1830	4424
1100	1830	4424
1101	1830	4424
1102	1830	4424
1103	1924	4424
1104	1917	1722
1105	1912	1685
1106	1899	1700
1107	1911	1697
1108	1911	1704
1109	1911	4533
1110	1911	4531
1111	1911	4531
1112	1911	1694
1113	1911	1694
1114	1911	1694
1115	1911	1694
1116	1911	4510
1117	1911	1759
1118	1911	1771
1119	1911	4601

Request Ke-	**Waktu Respon (ms)**	
	Skenario 1	**Skenario 2**
1122	2003	1742
1123	2011	4613
1124	1997	1752
1125	1991	1751
1126	1990	1792
1127	1990	4616
1128	1990	1773
1129	1990	1749
1130	1990	4622
1131	1990	1793
1132	1990	4666
1133	1990	1774
1134	1990	4666
1135	1990	1796
1136	1990	1779
1137	1990	4669
1138	1990	1793
1139	1990	4668
1140	1990	4669
1141	1990	4726
1142	1990	4730
1143	1990	1846
1144	1990	1827

Request Ke-	**Waktu Respon (ms)**	
	Skenario 1	**Skenario 2**
1145	2098	4765
1146	2084	4777
1147	2078	4775
1148	2091	1879
1149	2073	1864
1150	2073	4764
1151	2073	4769
1152	2073	1861
1153	2073	1851
1154	2073	1855
1155	2073	1874
1156	2073	4786
1157	2073	1890
1158	2073	1878
1159	2073	4798
1160	2073	4801
1161	2073	4800
1162	2073	1880
1163	2073	4796
1164	2073	1898
1165	2073	1889
1166	2073	1928
1167	2073	1923

1120	1911	4603
1121	1911	4615

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
1168	2073	4865
1169	2073	4874
1170	2073	4893
1171	2073	1935
1172	2190	1928
1173	2196	4897
1174	2184	1867
1175	2203	4884
1176	2178	1911
1177	2178	4904
1178	2178	4896
1179	2178	1959
1180	2178	4908
1181	2178	4947
1182	2178	1923
1183	2178	4927
1184	2178	1989
1185	2178	4950
1186	2178	4941
1187	2178	4952
1188	2178	4945

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
1191	2284	1915
1192	2279	2044
1193	2270	5015
1194	2294	1816
1195	2269	2016
1196	87	5021
1197	66	1831
1198	72	1946
1199	2351	5037
1200	2365	1836
1201	2380	5035
1202	2373	1996
1203	2358	5057
1204	114	5061
1205	130	2006
1206	123	5075
1207	162	2008
1208	2449	1873
1209	181	5077
1210	175	5086
1211	2461	2032
1212	2466	5092
1213	2439	5083

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
1214	2447	1887
1215	227	5103
1216	223	5150
1217	215	5151
1218	2543	1929
1219	2541	5161
1220	2541	2037
1221	2554	5162
1222	2563	1866
1223	293	1860
1224	289	5172
1225	285	5184
1226	336	1939
1227	346	1951
1228	333	1901
1229	2631	5187
1230	2644	5192
1231	2648	1873
1232	2640	5207
1233	2660	5208
1234	74	5222
1235	398	1973
1236	384	1967

1189	2178	1919
1190	2178	4955

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
1237	393	2000
1238	91	5235
1239	88	1915
1240	444	2000
1241	451	1889
1242	440	1908
1243	143	5282
1244	137	1853
1245	2724	1860
1246	2719	5314
1247	2747	5292
1248	2728	5295
1249	2731	1948
1250	140	5331
1251	496	2021
1252	490	1899
1253	504	2003
1254	199	5336
1255	197	2012
1256	190	1867
1257	2607	5342

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
1260	2597	1959
1261	2604	5379
1262	241	5361
1263	250	2031
1264	258	1909
1265	564	5402
1266	578	5452
1267	573	5433
1268	624	5439
1269	300	5455
1270	315	1894
1271	305	1852
1272	2701	5429
1273	2716	1899
1274	2724	2017
1275	2705	1890
1276	2731	5480
1277	655	1988
1278	645	1899
1279	662	2022
1280	371	5507
1281	364	1988
1282	380	1932

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
1283	715	1869
1284	724	5500
1285	732	5494
1286	721	5511
1287	2814	1895
1288	2801	2020
1289	2797	5531
1290	423	5550
1291	2826	1835
1292	2807	2006
1293	432	1933
1294	421	5747
1295	788	1838
1296	470	5581
1297	467	1975
1298	483	1889
1299	791	5756
1300	801	1892
1301	789	2003
1302	2896	5635
1303	2902	5809
1304	2910	5809
1305	2891	5817

1258	2831	1899
1259	2615	5369

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
1306	2885	5817
1307	516	1997
1308	515	5649
1309	533	1882
1310	95	5649
1311	88	2006
1312	861	1897
1313	103	5827
1314	866	5825
1315	856	5856
1316	865	1842
1317	576	1842
1318	572	5872
1319	587	1947
1320	147	5887
1321	142	1959
1322	2980	5890
1323	2998	1866
1324	2990	5879
1325	2981	1864
1326	2985	5908

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
1329	921	5910
1330	929	1853
1331	938	1982
1332	624	5933
1333	631	1858
1334	639	5939
1335	212	5958
1336	222	5955
1337	226	5961
1338	3080	5955
1339	3072	1900
1340	3071	1954
1341	3094	1833
1342	3102	1939
1343	1021	1871
1344	1014	6008
1345	1003	1875
1346	1002	6011
1347	716	1963
1348	701	1846
1349	711	1842
1350	294	1980
1351	296	1859

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
1352	301	1863
1353	362	1865
1354	1085	1982
1355	368	6057
1356	1072	1852
1357	1095	6064
1358	1080	1905
1359	369	1854
1360	786	1905
1361	780	1987
1362	774	1896
1363	807	6090
1364	3168	1895
1365	3188	1861
1366	3176	1830
1367	3183	1866
1368	3196	1951
1369	432	1829
1370	433	1987
1371	428	1935
1372	1135	1839
1373	1151	1997
1374	1144	1960

1327	150	5906
1328	941	5902
Request Ke-	**Waktu Respon (ms)**	
	Skenario 1	**Skenario 2**
1375	1160	1857
1376	850	1853
1377	844	1954
1378	859	1857
1379	846	1982
1380	1175	1891
1381	1168	1846
1382	490	1905
1383	484	1847
1384	3281	1974
1385	3268	2008
1386	3277	1980
1387	3265	1972
1388	3260	1891
1389	485	1899
1390	915	1980
1391	912	1861
1392	925	1859
1393	542	1823
1394	537	1948
1395	926	1950

Request Ke-	**Waktu Respon (ms)**	
	Skenario 1	**Skenario 2**
1398	3343	1833
1399	3363	1923
1400	3304	1850
1401	3345	1970
1402	3313	1851
1403	978	1973
1404	974	1870
1405	988	1837
1406	995	1869
1407	1044	1846
1408	3412	1968
1409	3323	1843
1410	3372	2006
1411	3337	1973
1412	3343	1958
1413	1061	1885
1414	1048	1908
1415	1083	1782
1416	1117	1799
1417	1127	1939
1418	1135	1980
1419	3397	1779
1420	3344	1946

Request Ke-	**Waktu Respon (ms)**	
	Skenario 1	**Skenario 2**
1421	3360	1807
1422	3368	1885
1423	3375	1946
1424	1154	1797
1425	1179	1826
1426	1185	1807
1427	1198	1830
1428	1221	1969
1429	3348	1975
1430	3363	1829
1431	3392	1833
1432	3266	1839
1433	3362	1962
1434	1258	1973
1435	1246	1874
1436	1254	1869
1437	1284	1880
1438	3336	1998
1439	3232	1847
1440	3334	1850
1441	3375	1902
1442	3329	1896
1443	1314	1899

1396	536	1831
1397	504	1930

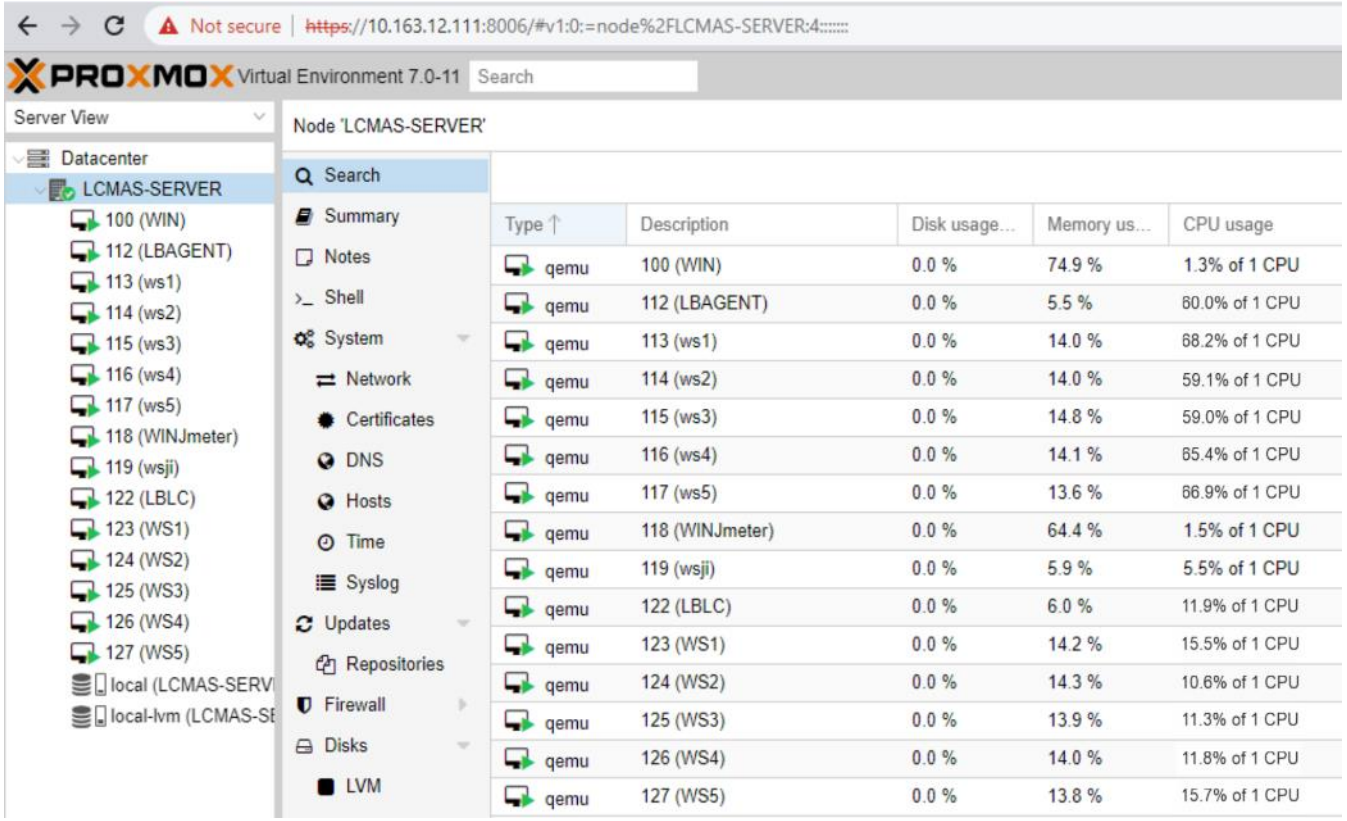
Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
1444	1309	1839
1445	1322	1843
1446	1352	1899
1447	1381	1825
1448	1376	1831
1449	1371	1864
1450	3224	1922
1451	3318	1917
1452	3344	1851
1453	3313	1853
1454	3341	1860
1455	1449	1832
1456	1478	1940
1457	1475	1843
1458	1490	1946
1459	3335	1846
1460	3344	1897
1461	3340	1869
1462	3364	1980
1463	3258	1980
1464	1539	1932

Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
1467	1550	1846
1468	1554	1821
1469	1558	1923
1470	3322	1831
1471	3332	1946
1472	3352	1910
1473	3341	1820
1474	3236	1863
1475	3307	1963
1476	3296	1967
1477	3319	1848
1478	3306	1844
1479	3223	1853
1480	3296	1951
1481	3301	1851
1482	3293	1821
1483	3288	1838
1484	3206	1858
1485	3298	1830
1486	3308	1832
1487	3296	1915
1488	3293	1919
1489	3216	1812

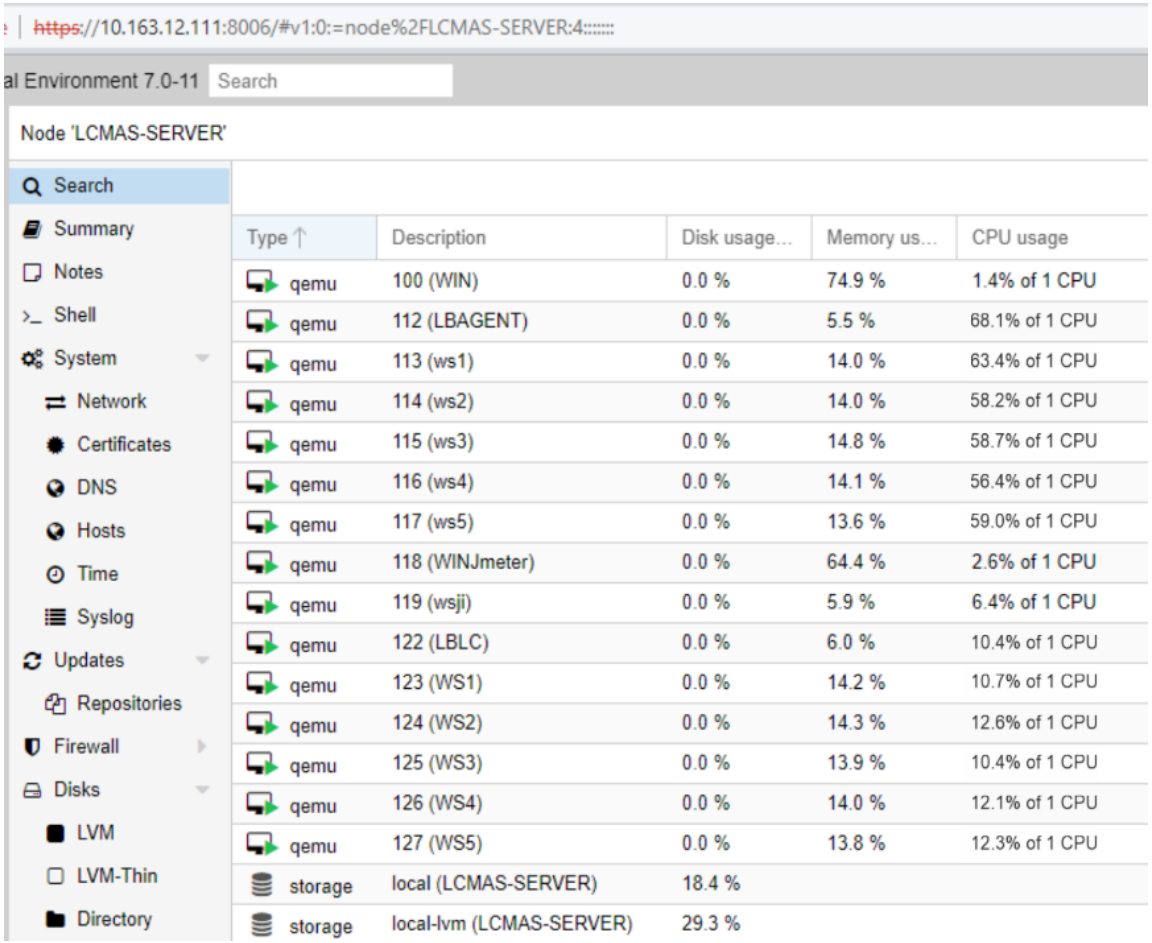
Request Ke-	Waktu Respon (ms)	
	Skenario 1	Skenario 2
1490	3298	1857
1491	3298	1988
1492	3293	1989
1493	3299	1866
1494	3190	1887
1495	3269	1897
1496	3251	2146
1497	3266	1900
1498	3263	1892
1499	3155	1911
1500	3177	1923

1465	1554	1848
1466	1566	1963

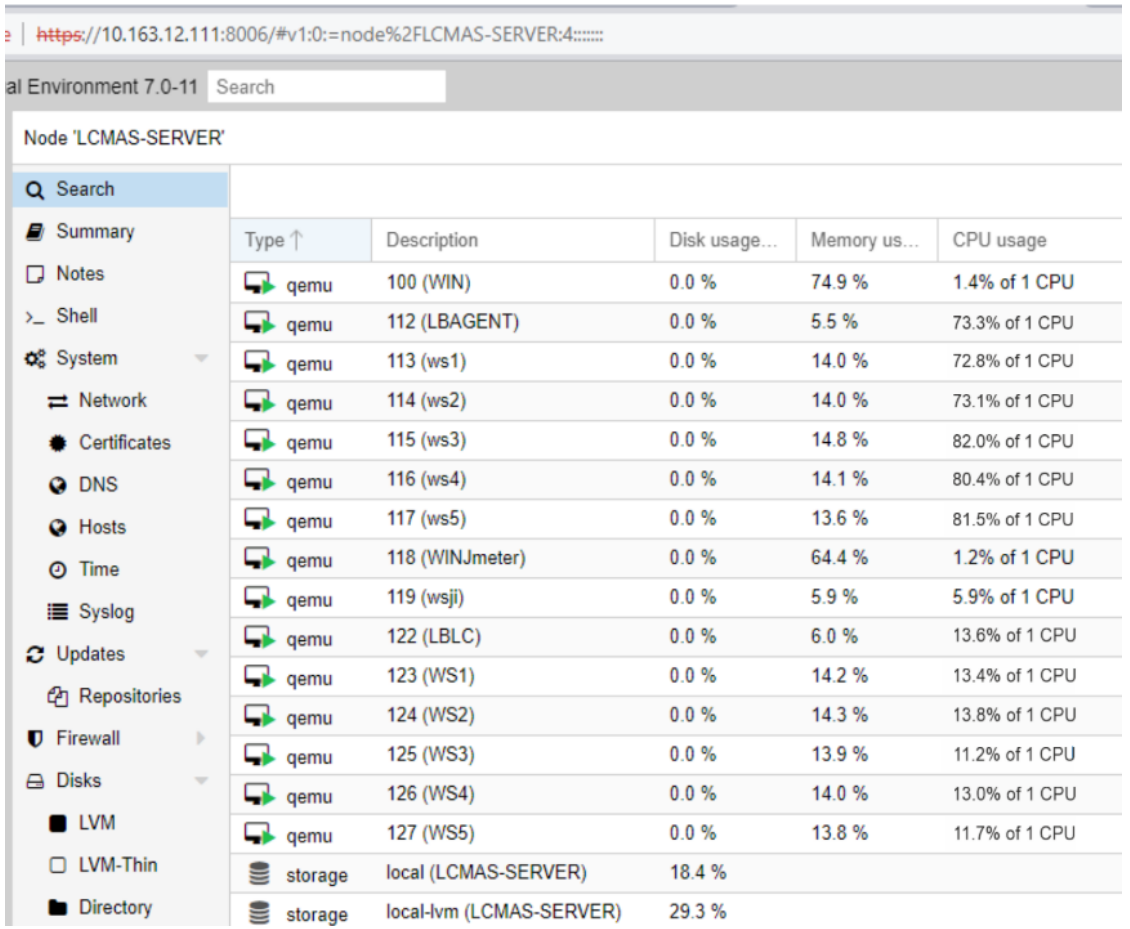
Lampiran b.



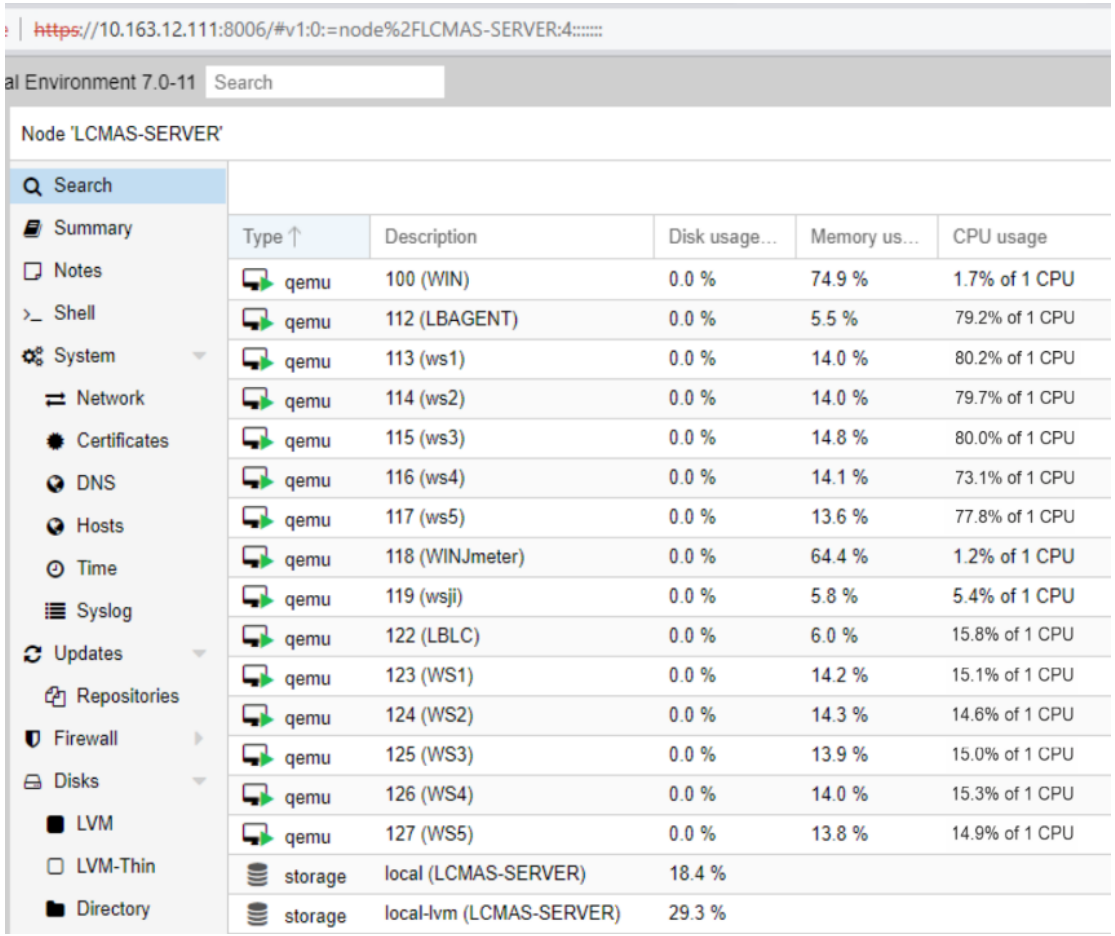
Gambar 25 Persentase CPU Awal – 1



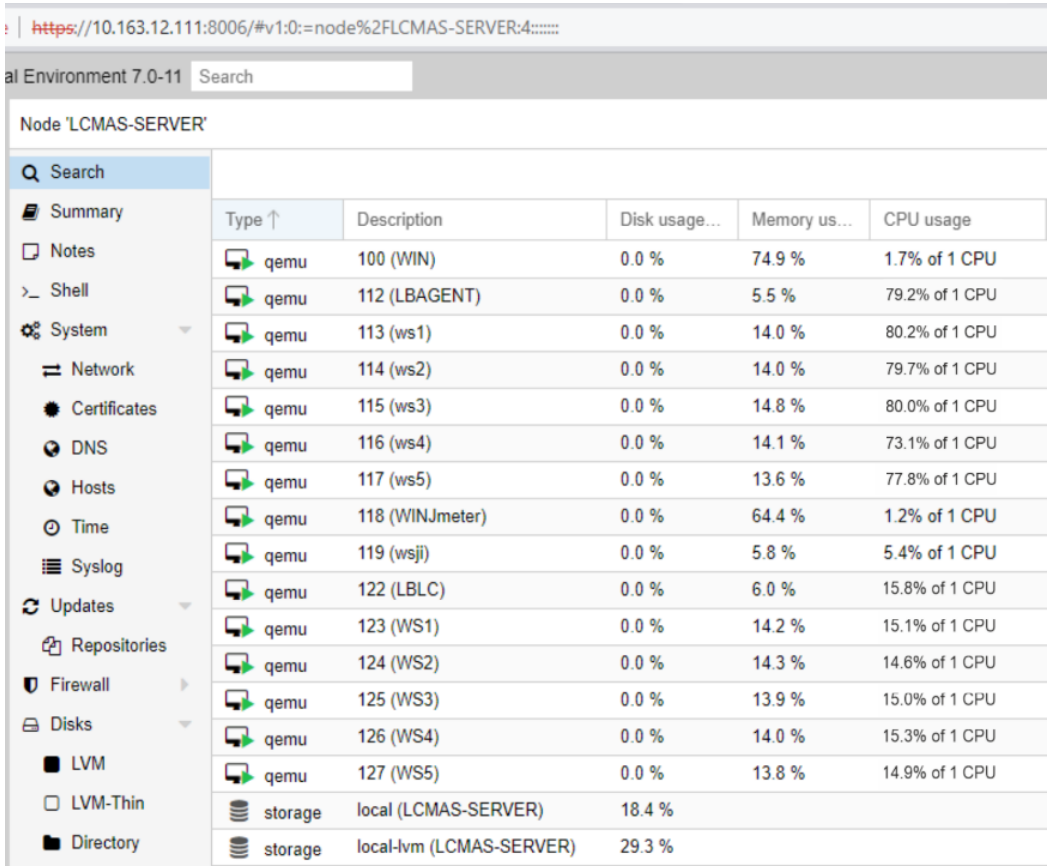
Gambar 26 Persentase Awal CPU – 2



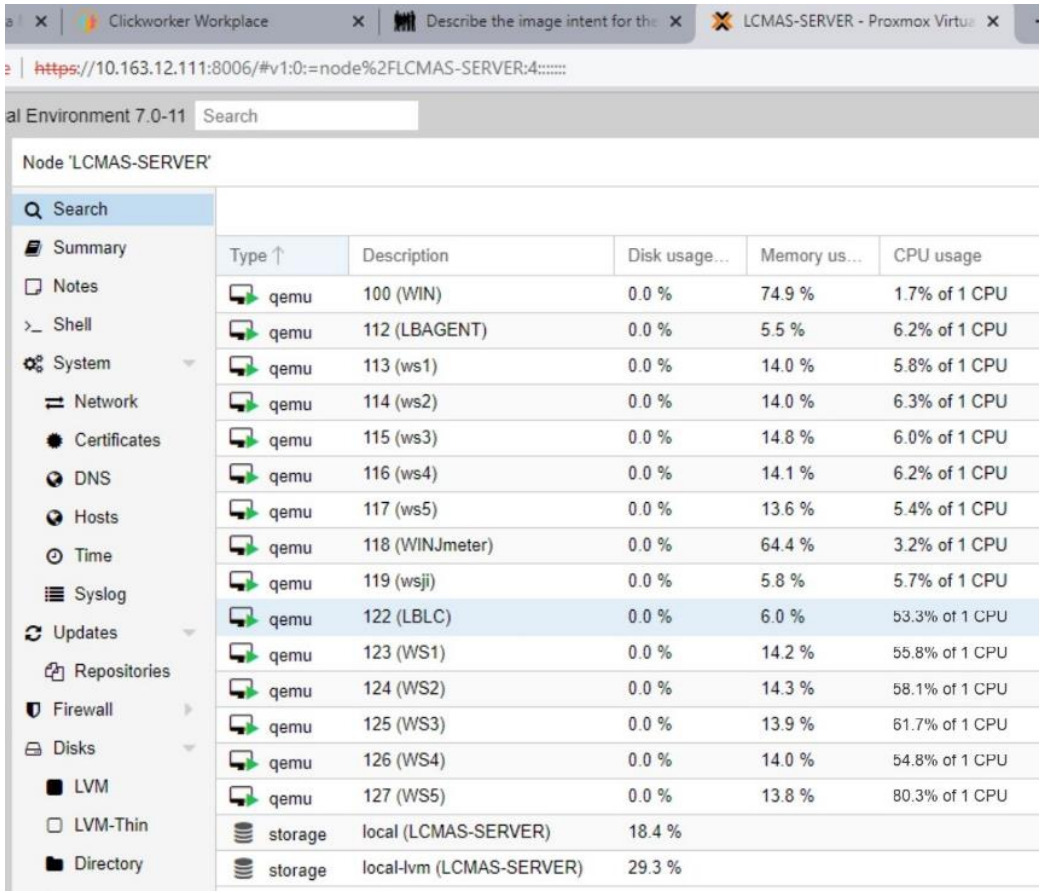
Gambar 27 Persentase Awal CPU – 3



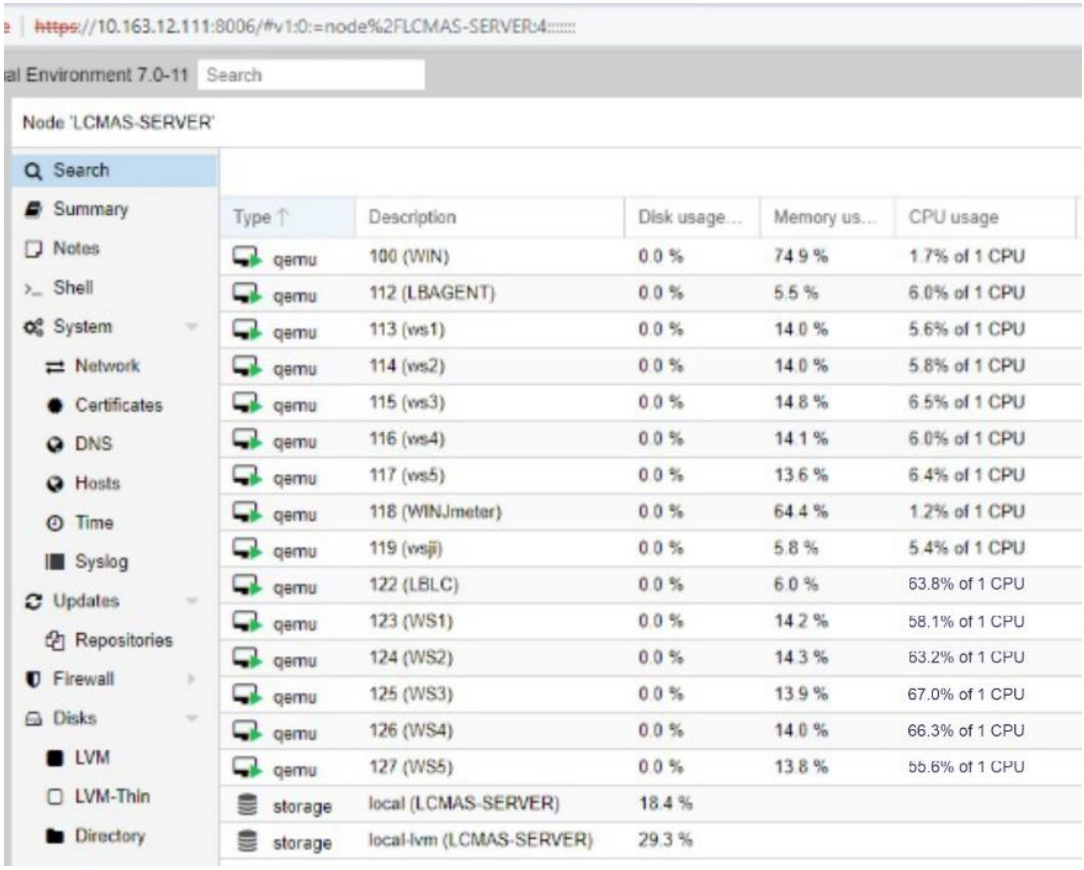
Gambar 28 Persentase Awal CPU – 4



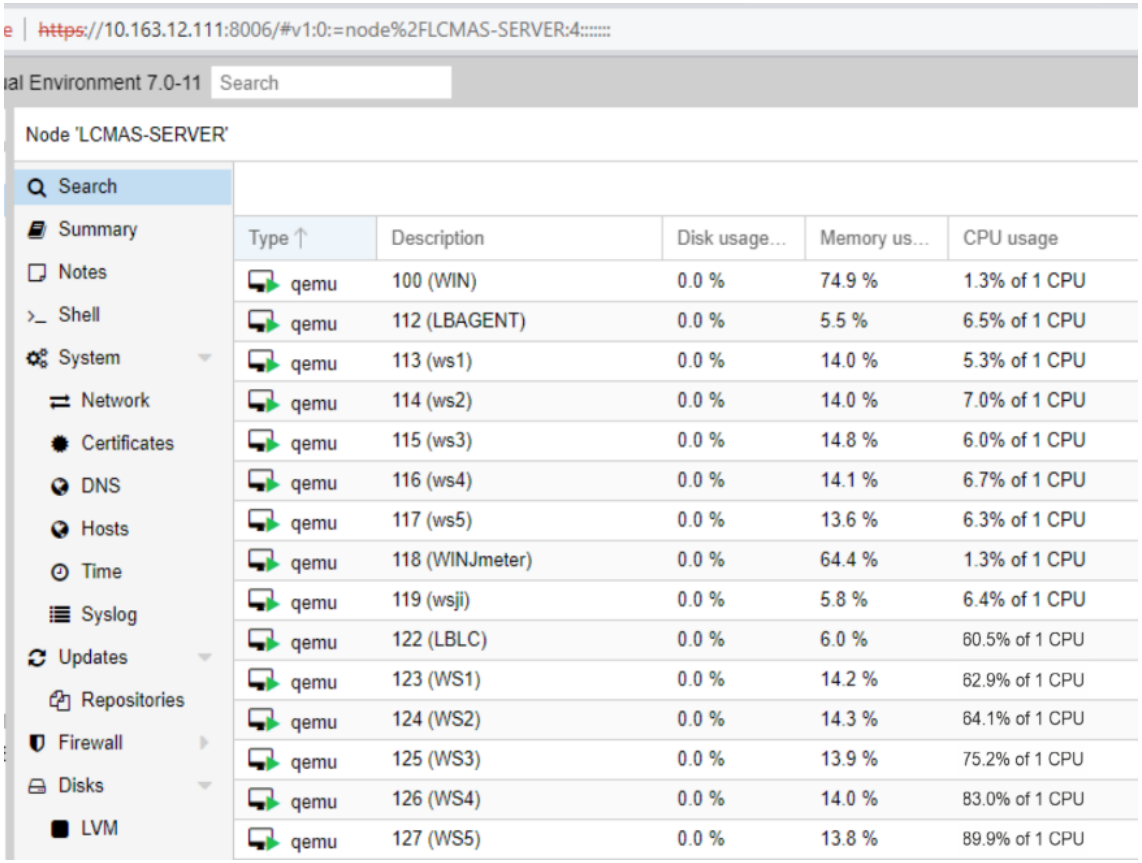
Gambar 29 Persentase Awal CPU – 5



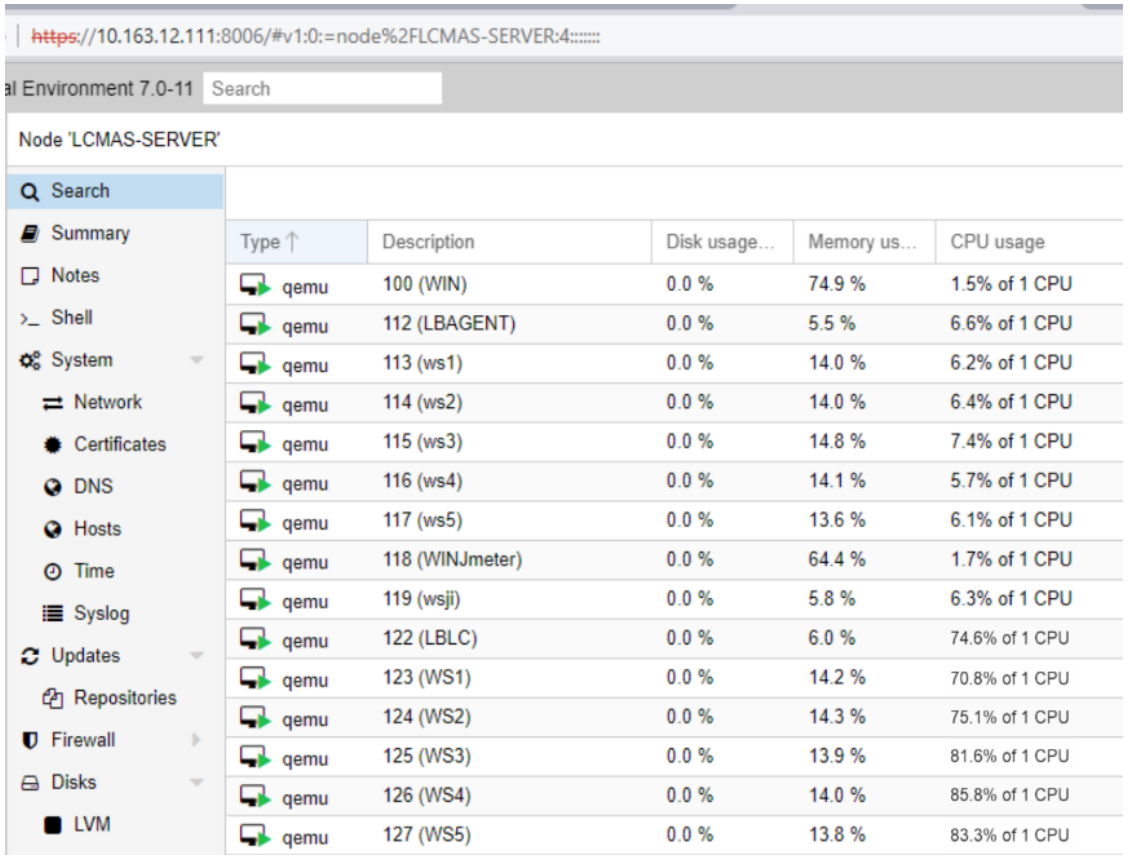
Gambar 30 Persentase Akhir CPU – 1



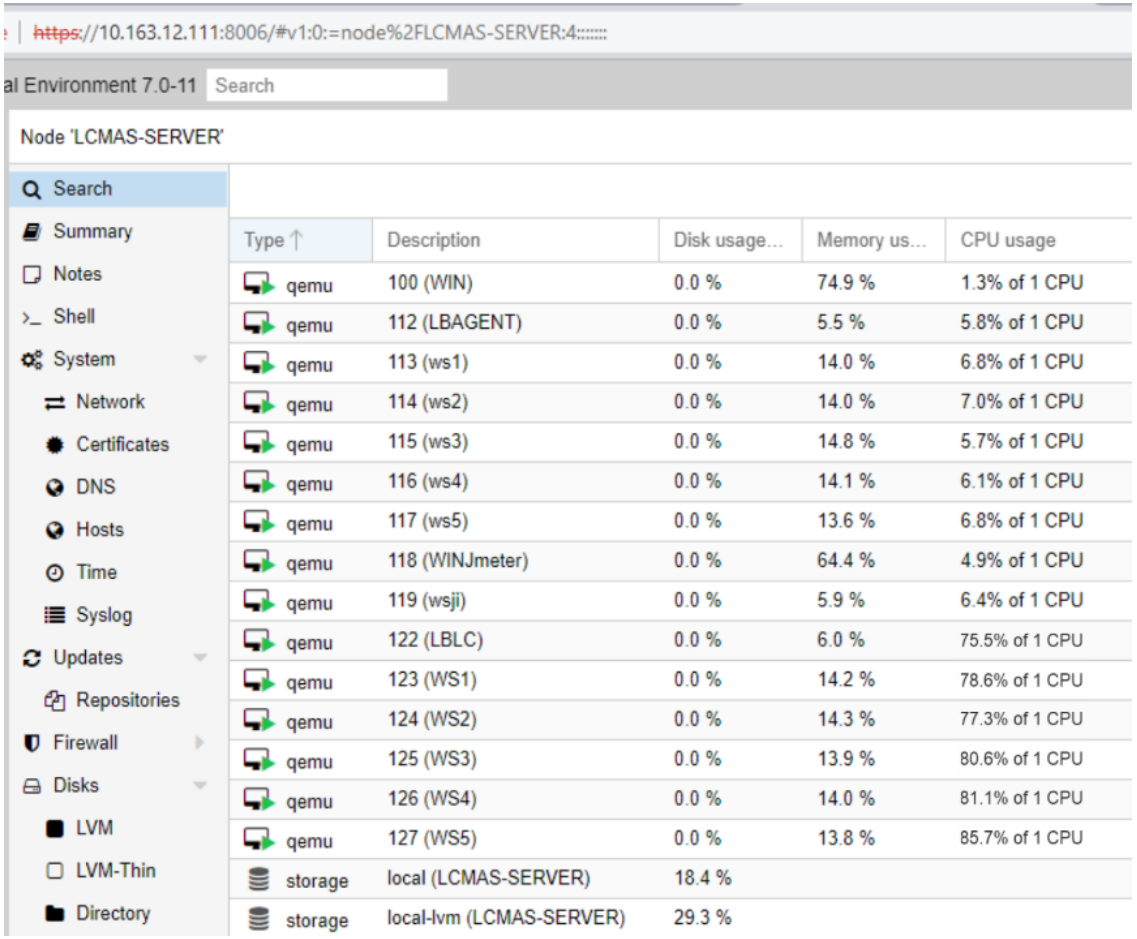
Gambar 31 Persentase Akhir CPU -2



Gambar 32 Persentase Akhir CPU – 3



Gambar 33 Persentase Akhir CPU – 4



Gambar 34 Persentase Akhir CPU - 5