Analisis Unjuk Kerja Load Balancing Web Server Menggunakan Virtualisasi Berbasis Container Docker Swarm

Arnanda Satria Wibawa*[1], Bongga Arifwidodo[2], Eka Wahyudi[3]
[1,2,3] S1-Teknik Telekomunikasi, Fakultas Teknik Telekomunikasi dan Elektro,
Insititut Teknologi Telkom Purwokerto
E-mail: *[1]**15101008@st3telkom.ac.id**, [2]bongga@ittelkom-pwt.ac.id,
[3]0617117601@ittelkom-pwt.ac.id

Abstrak

Meningkatnya jumlah trafik data mempengaruhi kinerja web server sedangkan penggunaan server tunggal masih banyak digunakan. Untuk mengatasi permasalahan tersebut, dibutuhkan teknik load balancing. Load balancing yang dipakai adalah zevenet ce. Load balancing memiliki beberapa algoritma diantaranya adalah round robin dan least connection. Untuk mengetahui algoritma load balancing mana yang lebih baik dalam menangani koneksi dari client maka dibutuhkan pengukuran response time, CPU Utilization dan Memory usage web server. Web server dibangun menggunakan docker swarm. Pengujian dilakukan sebanyak 20 kali percobaan pada setiap parameter kemudian diambil rata-rata untuk response time dan nilai CPU Utilization dan memory usage tertinggi selama percobaan. Tiap algoritma diberikan beban sebanyak 500 dengan 100 konkurensi, 2000 dengan 400 konkurensi, dan 5000 koneksi dengan 1000 konkurensi menggunakan H2load Benchmark. Hasil penelitian menunjukkan bahwa round robin bekerja lebih optimal dan cepat dibandingkan least connection karena memiliki nilai response time yang lebih kecil, CPU Utilization yang lebih besar dan memory usage yang lebih kecil dengan nilai response time pada 500 koneksi dengan 100 konkurensi sebesar 6,28 second, 19,83 second pada 2000 koneksi dengan 400 konkurensi dan 30,19 second pada 5000 koneksi dengan 1000 konkurensi. Persentase CPU Utilization web server pada koneksi 500-5000 node manager sebesar 62,56%-78,55% dan node worker sebesar 61,99%-62,32%. Nilai Memory usage web server pada koneksi 500-5000 sebesar 344,10MB-602,44MB pada node manager dan 319,72MB-586,70MB pada node worker.

Kata Kunci*— Web Server, Load Balancing, Zevenet, Least Connection, Round Robin, Docker Swarm*

1. PENDAHULUAN

Dewasa ini, manusia semakin bergantung pada teknologi khususnya dalam bidang telekomunikasi dalam usaha mereka memperoleh berbagai macam informasi. Hal ini menyebabkan angka pertukaran data naik setiap harinya sehingga kecepatan dan kehandalan web server sangat dibutuhkan. Tetapi, saat ini server tunggal masih banyak digunakan sedangkan sistem ini memiliki titik lemah dalam menangani permintaan data yang semakin bertambah banyak [1]. Oleh karena itu dibutuhkan teknik load balancing untuk menangani banyaknya permintaan data yang terjadi. Load balancing menggabungkan beberapa server dan bekerja dengan membagi beban trafik secara merata kepada setiap server sehingga resiko terjadinya kemacetan traffik dan beban server yang berlebihan saat menangani koneksi bersamaan dapat ditekan. Pada penelitian Faris Muslim Azmi [2] mengimplementasikan load balancing dalam sebuah sistem merupakan cara yang tepat dalam usaha membagi beban server saat banyaknya

koneksi yang masuk secara bersamaan. Zevenet merupakan salah satu load balancer yang bersifat open source dan dapat berjalan pada virtual machine. Zevenet memiliki banyak algoritma load balancing salah satunya least connection dan round robin. Least Connection bekerja dengan cara membagi beban sesuai dengan banyaknya koneksi yang sedang dilayani oleh server, beban koneksi yang masuk akan dilayani oleh server dengan jumlah koneksi yang lebih sedikit. Sedangkan round robin bekerja dengan cara membagi koneksi yang masuk ke semua server secara merata tanpa memperhatikan kapasitas server ataupun koneksi yang sedang dilayani oleh server. Docker merupakan salah satu virtualisasi berbasis kontainer. Namun, mengelola banyak kontainer untuk menjalankan sebuah service adalah tugas yang tidak mudah. Untuk itu, docker memperkenalkan docker swarm. Docker swarm dapat membantu mengelompokkan beberapa kontainer menjadi satu kelompok sehingga dapat diatur dalam tempat yang sama yaitu swarm. Docker swarm mempunyai dua node, yaitu node manager dan node worker. Node manager berfungsi untuk mengatur keanggotaan sedangkan node worker berfungsi menjalankan swarm service di docker swarm [1].

Penelitian [1] tahun 2019 mengungkapkan bahwa penggunaan server tungaal belum mampu menangani permintaan data yang sangat banyak. Untuk itu, penulis menggunakan teknologi virtualisasi bernama docker swarm sebagai alat untuk membuat clustering web server untuk meningkatkan keandalan server. Untuk menyeimbangkan beban internal server, digunakan load balancing pada docker swarm sehingga permintaan dapat didistribusikan dengan baik ke web server. Algoritma load balancing yang digunakan yaitu algoritma round robin dan algoritma least connection. Sistem akan dibanjiri dengan 1000, 3000, dan 5000 koneksi untuk mendapatkan nilai throughput. Hasil pengujian menunjukkan load balancing yang menerapkan algoritma least connection memiliki throughput 15 Mbps pada 1000 koneksi, 17 Mbps pada 3000 koneksi dan 5000 koneksi. Sedangkan algoritma round robin memiliki nilai throughput 15 Mbps pada 1000 koneksi, 14 Mbps pada 3000 koneksi, dan 15 Mbps pada 5000 koneksi. Hasil menunjukkan bahwa algoritma least connection mempunyai kinerja yang lebih baik daripada algoritma round robin.

Penelitian [2] tahun 2019 tentang Perbandingan Kinerja HAproxy dan Zevenet dalam Pengimplementasian Multi Service Load Balancing. Dalam penelitian tersebut, penulis mengungkapkan bahwa setiap pengguna mempunyai kebutuhan layanan yang berbeda sehingga dibutuhkan sebuah sistem load balancing multi-service untuk menekan kemacetan trafik data dan mengurangi beban kerja server. Aplikasi load balancer HAproxy dan Zevenet akan dibandingkan kinerjanya dalam pengimplementasian multi-service load balancing dengan mengirimkan jumlah koneksi sebanyak 1000, 2500, dan 5000 HTTP dan FTP. Algoritma yang digunakan adalah algoritma round robin dan algoritma least connection dan parameter yang diuji adalah response time dan resource utilization. Dari pengujian tersebut diperoleh kesimpulan bahwa HAproxy mengungguli Zevenet pada parameter response time. Sedangkan Zevenet lebih unggul dalam parameter resource utilization untuk layanan HTTP dan FTP.

Penelitian [3] tahun 2019 tentang Analisis Perbandingan Kinerja HAproxy dan Zevenet sebagai Load Balancer Server. Peneliti menemukan bahwa sering terjadi kemacetan trafik data, fail, hingga kerusakan pada server yang disebabkan oleh peningkatan kebutuhan layanan dan tujuan yang beragam dari berbagai macam pengguna. Dengan load balancer kemacetan trafik data dari pengguna ke server dapat ditekan dan membantu pembagian beban kerja server. Pengujian kinerja HAproxy dan Zevenet dilakukan dengan menggunakan algoritma round robin dan dengan cara mengirimkan 3000 koneksi ke server sehingga didapatkan nilai pada parameter uji rata – rata koneksi dan tingkat errors. Hasil pengujian menunjukkan bahwa Zevenet memiliki nilai yang lebih baik daripada HAproxy dalam parameter rata – rata koneksi. Sedangkan HAproxy lebih stabil karena tidak memiliki nilai errors.

Berdasarkan permasalahan meningkatnya jumlah pertukaran data dan masih banyaknya penggunaan server tunggal, maka penerapan load balancing server pada cluster web server

menggunakan virtualisasi container berbasis docker swarm diharapkan dapat membuat kinerja web server menjadi lebih efisien.

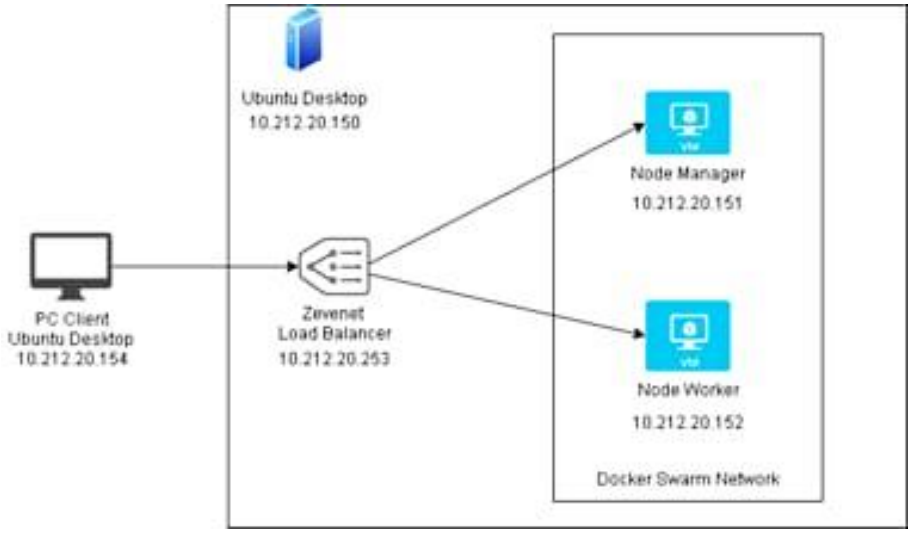
2. METODE PENELITIAN

2.1 Kebutuhan Perangkat

Terdapat beberapa perangkat yang digunakan dalam penelitian ini untuk perancangan system yaitu terdiri dari *hardware* dan *software* antara lain:

1. *Hardware*: *Server*: PC Ubuntu 22.04 LTS, Processor i7-7700, 8 CPU 3,60 GHz, RAM 8 GB, SSD 100 GB
2. *Software*: OS Ubuntu Server 22.04 (RAM 1GB dan vCPU 1 *Core* pada masing-masing VM Ubuntu *server*).
3. *Tools* pendukung: *Zevenet* (*Tools load balancer*), *Virtualbox* (*tools* virtualisasi), dan *h2load benchmark* (*tools* pengujian load balancing pada *web server*).

2.2 Perancangan Desain Sistem



Gambar 1. Perancangan Desain Sistem

Rancangan topologi jaringan pada gambar 1. terdiri dari sebuah PC *client* dan sebuah PC *Server*. Di dalam PC *server* akan dibangun tiga buah *virtual machine* menggunakan *virtualbox* yaitu sebuah *virtual machine load balancer zevenet* dan dua buah *virtual machine ubuntu server* dimana kedua *virtual machine* ini akan dijadikan menjadi satu *cluster* yang sama menggunakan *docker swarm*. *Docker swarm* mempunyai dua jenis *node* yang masing-masing mempunyai tugas yang berbeda. *Node* utama atau *node manager* bertugas untuk menemukan *service* di dalam satu *cluster* yang sama, manajemen *cluster*, dan memantau kinerja *node* lainnya. *Node* kedua disebut sebagai *node worker* yang tugasnya adalah menerima perintah dari *node manager*. *Docker swarm* mempunya kelebihan yaitu jika salah satu *service* mati, maka *service* akan digantikan dengan yang masih aktif.

Tabel 1. Daftar Virtual Machine

VM	Alamat IP	Sistem Operasi	RAM	vCPU	Jumlah *Container*
Load Balancer	10.212.20.253	Debian	2 GB	2 *Core*	-
Node Manager	10.212.20.151	Ubuntu Server	1 GB	1 *Core*	1
Node Worker	10.212.20.152	Ubuntu Server	1 GB	1 *Core*	1

Tabel 1. Diatas merupakan spesifikasi yang dibutuhkan dalam pembuatan sistem pada *container docker* dan jumlah *container* yang dibutuhkan dalam penelitian ini. Semua *container* akan menjalankan *web server nginx*. Penggunaan sumber daya *virtual machine ubuntu server* akan dipantau menggunakan perintah *docker stats* dengan menjalankan serangkaian tes dan uji coba dengan menggunakan *h2load benchmark.*

2.3 Load Balancing

Load balancing merupakan teknik yang bekerja dengan cara mendistribusikan atau membagi beban kerja, jaringan, atau lalu lintas data ke beberapa *server*. Hal ini akan menjadikan *server* lebih andal dan kinerja meningkat karena titik kegagalan dapat ditekan [1]. Ada beberapa algoritma *load balancing* diantaranya adalah algoritma *round robin* dan algoritma *least connection*.

3.1.2 Algoritma Round Robin

Algoritma *round robin* bekerja dengan cara membagi beban ke seluruh *server* yang ada secara urut [2]. Misalnya, jika terdapat dua *server* di dalam *cluster* maka *request* pertama akan diteruskan menuju *server* 1, *request* kedua akan diteruskan menuju *server* 2, *request* ketiga akan diteruskan menuju *server* 1 lagi, dan seterusnya akan seperti itu siklusnya. Semua *server* diperlakukan setara tanpa memperhatikan jumlah koneksi yang masuk atau *response time* yang dialami setiap *server* [3].

3.1.3 Algoritma Least Connection

Algoritma *least connection* bekerja dengan membagi beban berdasarkan jumlah koneksi yang sedang dilayani *server*. *Server* dengan koneksi terendah akan melayani *request* yang masuk [4]. Algoritma ini termasuk salah satu algoritma yang dinamis karena perlu menghitung koneksi langsung untuk masing-masing server secara dinamis. Untuk *server* virtual, algoritma *least connection* baik untuk memperlancar distribusi beban ke *server* saat beban permintaan sangat bervariasi. Algoritma *least connection* mengasumsikan bahwa kemampuan pemrosesan semua *server* setara dan memberikan permintaan masuk kepada *server* dengan koneksi paling sedikit. Namun, kinerja sistem tidak ideal saat kemampuan pemrosesan *server* berbeda [5].

2.4 Docker

Docker menyediakan fasilitas untuk automasi aplikasi saat aplikasi tersebut di*deploy* ke dalam kontainer. Dalam lingkungan kontainer, dimana aplikasi divirtualisasikan dan dieksekusi. *Docker* menyediakan *lightweight environment* dimana kode dapat dieksekusi lebih efisien dan docker juga menyediakan fitur untuk proses kode dari komputer untuk di tes sebelum diproduksi.[6].

2.5 Docker Swarm

Swarm merupakan sekumpulan *docker host* atau *node* dimana saling terhubung melalui *overlay network*. Dengan demikian, pengguna dapat melakukan *deployment container* pada *multiple host* atau *node*. Dalam *docker swarm* terdapat *node manager* dan *node worker*. *Node manager* memiliki tugas untuk melakukan dan membuat *maintenance cluster service* sedangkan *node worker* berfungsi untuk menjalankan *container* atau layanan yang telah didefinisikan pada *node manager*. Kelebihan *docker swarm* adalah jika salah satu *node down* maka layanan akan dilakukan oleh *node* yang sedang aktif [7].

2.6 Virtualisasi

Virtualisasi adalah teknik yang memungkinkan pengguna untuk membuat versi virtual perangkat atau sumber daya seperti sistem operasi, jaringan, *server* dan perangkat penyimpanan. Virtualisasi memungkinkan sebuah komputer menjalankan beberapa sistem operasi secara serentak. Virtualisasi membagi sumber daya komputer utama menjadi beberapa sumber daya virtual dimana sistem operasi, *libraries*, dan program lainnya bersifat unik dan tidak terhubung ke sistem operasi komputer *host* dibawahnya [1]. Virtualisasi mengacu pada mesin virtual yang berperilaku seperti komputer nyata dengan sistem operasi. Perangkat lunak yang berjalan di mesin virtual yang terpisah dari sumber daya perangkat keras yang mendasarinya. Perangkat lunak yang mendasari mesin virtual pada perangkat keras *host* disebut *hypervisor* atau monitor mesin virtual. *Hypervisor* dalam dunia komputasi adalah *platform* dasar virtualisasi yang memungkinkan beberapa sistem operasi berjalan bersamaan dalam satu komputer.

2.7 Kontainerisasi

Virtualisasi berbasis *container* adalah cara terbaik dan mudah untuk membat *virtual user space*, ini dilakukan dengan memisahkan *kernel* dari sistem operasi utama (*host*). *User space* sistem operasi utama atau *host* dibagi menjadi *isolated user space* yang disebut *container*. Model *container* ini memiliki keunggulan *overhead* yang lebih rendah dibandingkan dengan metode berbasis *hypervisor* karena tidak perlu menjalankan sistem operasi tamu (*guest OS*) dan mesin virtual untuk setiap *virtualized environment*. Selain itu virtualisasi berbasis *container* dapat memberikan lebih banyak *user space*. Virtualisasi berbasis container digunakan sebagai *container* tempat menjalankan sistem operasi, *web server*, dan alat *benchmarking* [10].

2.8 Web Server

Web server merupakan perangkat lunak yang melayani request HTTP dari *web browser client* dan mengirimkan kode-kode dinamis ke *server* aplikasi. *Server* aplikasi ini bertugas untuk menerjemahkan dan memproses kode-kode dinamis menjadi kode-kode statis *HTML* yang selanjutnya dikirimkan ke *browser* oleh *web server*. *Web server* menggunakan protokol HTTP [4]. *Web server* mempunyai fungsi untuk menempatkan halaman *web*, sebagai penyimpanan atau *database* dan penggunaan aplikasi bisnis [9].

3. HASIL DAN PEMBAHASAN

3.1. Parameter Pengujian

Parameter pengujian yang dilakukan pada sistem load balancing web server yaitu: response time, CPU Utilization, dan Memory Usage.

3.1.1 Response time

Response time merupakan waktu yang dibutuhkan oleh server dalam menangani request dari client. Nama lain response time adalah latency atau time to first byte (TTFB) yang merupakan total waktu tunggu dan waktu menjawab request [12]. Tujuan utama load balancing adalah untuk menyediakan response time terbaik kepada client dengan cara menyebarkan beban ke server dalam cluster. Jika response time server tinggi, akan ada request yang menunggu untuk ditangani server. Jika response time server rendah, request yang menunggu untuk ditangani juga rendah karena server melayani request dengan rate tinggi. Jika masing-masing server mempunyai response time yang berbeda-beda, request akan dikirimkan ke server dengan beban yang lebih sedikit [13]. Response time menggambarkan kecepatan web server dalam melayanai koneksi dari client. Response time dihitung dalam satuan millisecond (ms) [14]

3.1.2 CPU Utilization

CPU Utilization merupakan persentase banyaknya penggunaan sumber daya CPU yang digunakan oleh server ketika menangani request yang masuk. Peningkatan jumlah request mengakibatkan nilai CPU utilization ikut meningkat [12]. Lonjakan singkat dalam penggunaan CPU menunjukkan penggunaan sumber daya mesin virtual digunakan dengan sebaik-baiknya. Namun, jika penggunaan CPU untuk mesin virtual diatas 90% dan CPU idle diatas 20% akan memengaruhi kinerja mesin virtual tersebut [15].

3.1.3 Memory Usage

Memory usage adalah banyaknya RAM yang digunakan oleh server ketika memproses request yang masuk [12].

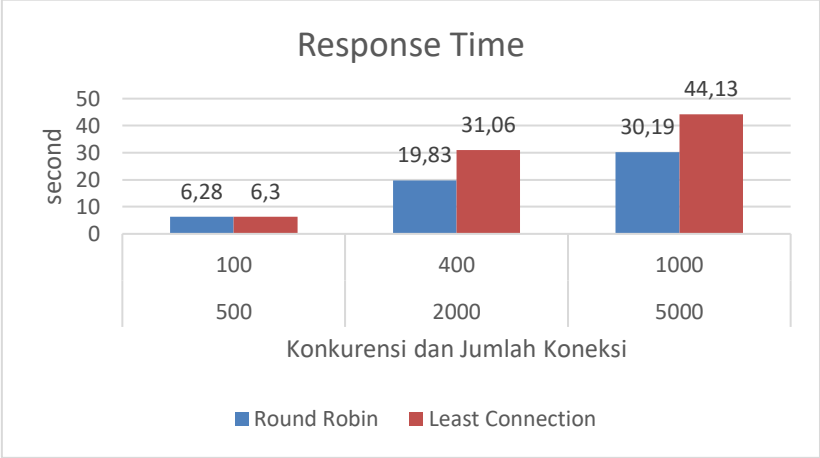
3.2. Skenario Pengujian

Skenario pengujian dilakukan untuk mengetahui unjuk kerja load balancing web server. Pengujian dilakukan untuk mengetahui kemampuan algoritma load balancing yaitu algoritma round robin dan algoritma least connection. Web server akan diberikan koneksi sebesar 500 koneksi dengan konkurensi 100 koneksi, 2000 koneksi dengan konkurensi 400 koneksi dan 5000 koneksi dengan konkurensi 1000 koneksi.

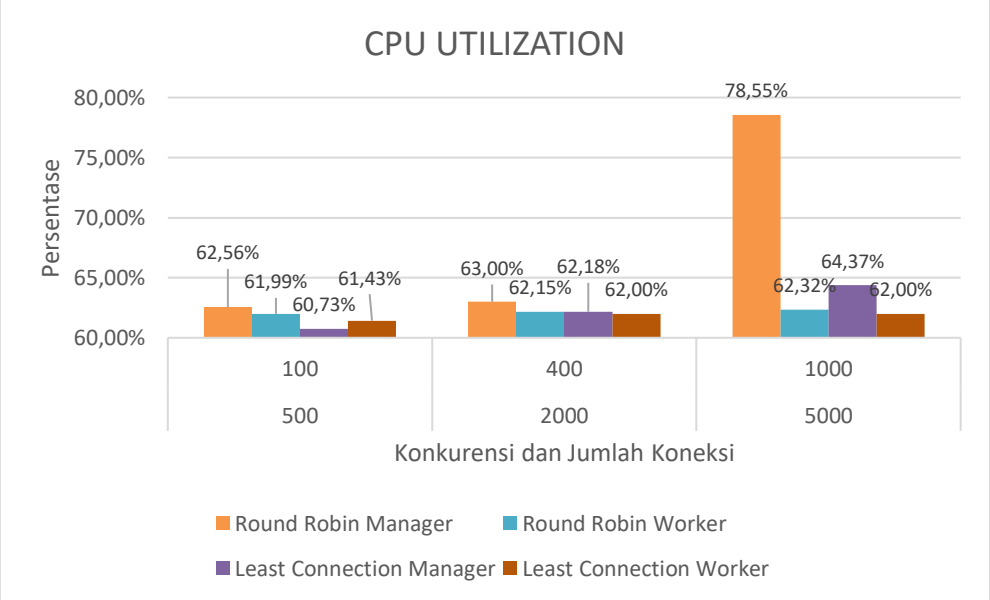
3.3. Hasil Pengujian

Nilai response time bertambah sebanding dengan bertambahnya jumlah koneksi dan jumlah konkurensi, semakin tinggi jumlah koneksi dan jumlah konkurensi maka semakin tinggi juga nilai response time yang diberikan oleh load balancing web server. Nilai rata-rata response time algoritma round robin lebih kecil dibandingkan dengan algoritma least connection pada jumlah koneksi 500, 2000 dan 5000. Nilai rata-rata response time kedua algoritma mengalami kenaikan nilai dari jumlah koneksi 500, 2000, dan 5000. Nilai response time algoritma round robin lebih unggul dibandingkan dengan algoritma least connection. Hal ini dikarenakan algoritma round robin hanya melakukan perhitungan terhadap jumlah koneksi yang masuk lalu mengarahkan ke web server tanpa harus melakukan perbandingan terlebih dahulu terhadap jumlah koneksi yang sedang dilayani oleh web server seperti yang dilakukan algoritma least connection. Dengan demikian, algoritma round robin memproses koneksi yang masuk lebih cepat daripada algoritma least connection sehingga mempengaruhi besarnya nilai response time.

P-ISSN: 2089-676X
E-ISSN: 2549-0796



Gambar 2. Hasil Response Time

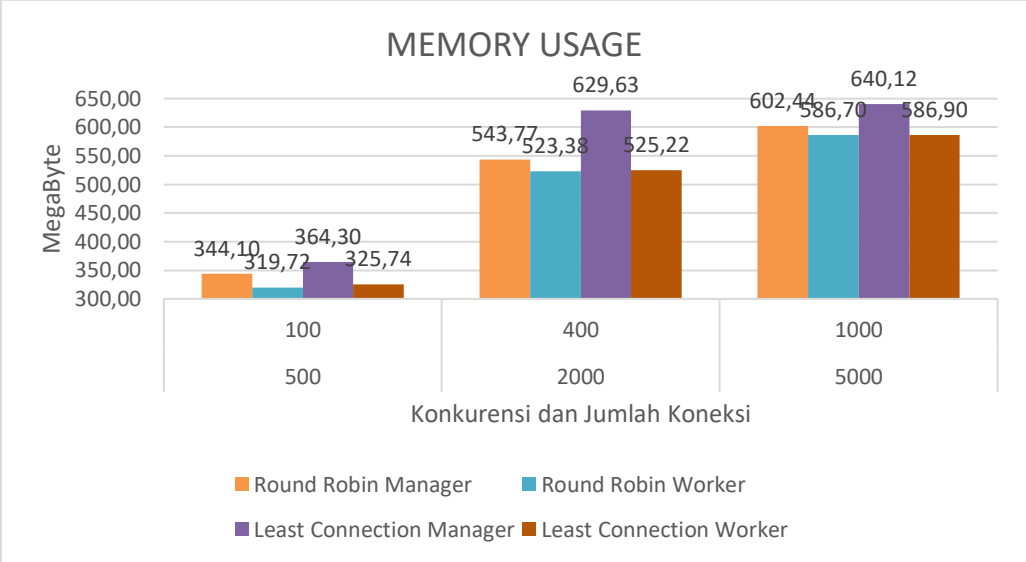


Gambar 3. Diagram CPU Utilization

Persentase CPU Utilization web server bertambah sebanding dengan bertambahnya request atau jumlah koneksi yang diberikan. Rata-rata persentase CPU Utilization algoritma round robin lebih besar dibandingkan dengan algoritma least connection pada jumlah koneksi 500 dengan konkurensi 100 koneksi, 2000 dengan konkurensi 400 koneksi dan 5000 dengan konkurensi 1000 koneksi baik pada node manager maupun node worker. Hasil yang diperoleh menunjukkan nilai CPU Utilization tertinggi masing-masing node mengalami kenaikan sebanding dengan bertambahnya jumlah koneksi dan konkurensi baik pada saat menggunakan algoritma round robin maupun algoritma least connection. Node manager rata-rata menggunakan CPU lebih tinggi dibandingkan dengan node worker karena node manager bertugas juga untuk menemukan service yang diminta oleh client di dalam cluster dan juga memberikan tugas kepada node worker untuk mengelola atau melayani koneksi yang masuk dari client. Saat CPU Utilization node worker lebih tinggi dibandingkan node manager menandakan bahwa node manager mengalami kelebihan koneksi sehingga menugaskan node worker untuk melayani koneksi

P-ISSN: 2089-676X
E-ISSN: 2549-0796

tersebut. Persentase CPU Utilization algoritma round robin lebih tinggi dibandingkan dengan algoritma least connection dikarenakan algoritma round robin meneruskan koneksi dari client menuju ke web server secara terus menerus sehingga CPU Utilization web server akan terus meningkat lebih cepat. Sedangkan algoritma least connection sebelum meneruskan koneksi dari client harus melakukan perbandingan jumlah koneksi yang sedang dilayani oleh web server terlebih dahulu sehingga akan menimbulkan jeda saat meneruskan koneksi dari client. Saat jeda, CPU Utilization web server menurun, kemudian akan naik kembali saat koneksi sudah dilayani oleh web server. Hal demikianlah yang menjadikan CPU Utilization algoritma least connection lebih kecil daripada algoritma round robin.



Gambar 4. Diagram Memory Usage

Pada gambar 4. dapat diketahui bahwa *memory usage* bertambah sebanding dengan bertambahnya jumlah koneksi dan konkurensi. *Pada pengujian memory usage web server, dapat diketahui bahwa nilai memory usage meningkat sebanding dengan peningkatan jumlah koneksi dan konkurensi. Memory usage node manager* lebih tinggi dibandingkan dengan *memory usage node worker* karena *node manager* menjalankan beberapa tugas secara bersamaan sehingga mempengaruhi besarnya *memory usage.* Sedangkan *node worker* tugas yang dijalankan secara bersamaan lebih sedikit daripada *node manager* sehingga nilai *memory usage*nya lebih kecil dibandingkan dengan *node manager*. Algoritma *round robin* mempunyai nilai *memory usage* yang lebih kecil dibandingkan dengan algoritma *least connection*. Algoritma *round robin* bekerja dengan meneruskan langsung koneksi dari *client* yang masuk menuju ke *web server* tanpa melakukan perbandingan koneksi yang sedang dilayani oleh *web server* terlebih dahulu sehingga proses dapat dilakukan lebih cepat. Dikarenakan proses yang lebih cepat, maka *memory usage* pun tidak setinggi algoritma *least connection*. Sedangkan algoritma *least connection* akan membandingkan dahulu koneksi dari *client* yang sedang dilayani oleh *web server* yang mengakibatkan adanya jeda pelayanan antara satu koneksi dengan koneksi yang lainnya. Hal ini menjadikan proses menjadi lebih lama sehingga *memory usage web server* saat algoritma *least connection* lebih tinggi dibandingkan dengan algoritma *round robin* karena proses yang sedang berjalan tersimpan di dalam *memori.*

4. KESIMPULAN

Berdasarkan hasil dari pembahasan mengenai analisis unjuk kerja load balancing web server menggunakan virtualisasi berbasis container docker swarm maka dapat diperoleh beberapa kesimpulan sebagai berikut:

1. Hasil pengukuran parameter response time pada load balancing web server menggunakan algoritma round robin menghasilkan nilai yang lebih baik daripada algoritma least connection. Yaitu 6,28 second pada 500 koneksi dengan 100 konkurensi, 19,83 second pada 2000 koneksi dengan 400 konkurensi, dan 30,19 second pada 5000 koneksi dengan 1000 konkurensi.
2. Hasil pengukuran parameter CPU Utilization pada node manager dan node worker menggunakan algortitma round robin menghasilkan nilai yang lebih tinggi dibandingkan dengan algoritma least connection. Yaitu node manager 62,56 % dan node worker 61,99 % pada 500 koneksi dengan 100 konkurensi, 63,00 % CPU Utilization node manager dan 62,15 % node worker pada 2000 koneksi dengan 400 konkurensi, 78,55 % CPU Utilization node manager dan 62,32 % node worker pada 5000 koneksi dengan 1000 konkurensi.
3. Hasil pengukuran parameter memory usage pada node manager dan node worker menggunakan algoritma round robin meghasilkan nilai yang lebih rendah dibandingkan dengan algoritma least connection. Yaitu node manager 344,10 MB dan node worker 319,72 MB pada 500 koneksi dengan 100 konkurensi, 543,77 MB memory usage node manager dan 523,38 MB node worker pada 2000 koneksi dengan 400 konkurensi, 602,44 MB memory usage node manager dan 586,70 MB node worker pada 5000 koneksi dengan 1000 konkurensi.
4. Algoritma round robin bekerja lebih cepat dan efisien daripada algoritma least connection.

DAFTAR PUSTAKA

[1] D. S. Afis, M. Data, and W. Yahya, “Load Balancing Server Web Berdasarkan Jumlah Koneksi Klien Pada Docker Swarm,” J. Pengemb. Teknol. Inf. dan Ilmu Komput. Univ. Brawijaya, vol. 3, no. 1, pp. 925–930, 2019.

[2] F. M. Azmi, M. Data, and H. Nurwasito, “Perbandingan Kinerja Haproxy dan Zevenet Dalam Pengimplementasian Multi Service Load Balancing,” J. Pengemb. Teknol. Inf. dan Ilmu Komput. Univ. Brawijaya, vol. 3, no. 1, pp. 253–260, 2019.

[3] J. K. Azhar and L. Nurhakim, “Analisis Perbandingan Kinerja Haproxy dan Zevenet sebagai Load Balancer Server,” no. January, 2020.

[4] Efrizal Zaida, Kupas Tuntas Teknologi Virtualisasi. Yogyakarta: ANDI OFFSET, 2013.

[5] C. G. Kominos, N. Seyvet, and K. Vandikas, “Bare-metal, virtual machines and containers in OpenStack,” Proc. 2017 20th Conf. Innov. Clouds, Internet Networks, ICIN 2017, pp. 36–43, 2017, doi: 10.1109/ICIN.2017.7899247.

[6] I. G. L. P. E. Supramana, Prismana, “Implementasi Load Balancing Pada Web Server Dengan Menggunakan Apache,” J. Manaj. Inform., vol. 5, no. 2, pp. 117–125, 2016, [Online]. Available: https://jurnalmahasiswa.unesa.ac.id/index.php/jurnal-manajemen-informatika/article/view/16413/14911.

[7] M. ElGili Mustafa, “Load Balancing Algorithms Round-Robin (RR), Least-Connection and Least Loaded Efficiency,” Int. J. Comput. Inf. Technol., vol. 1, no. 1, pp. 2279–0764, 2017.

[8] B. Bashari Rad, H. J. Bhatti, and M. Ahmadi, “An Introduction to Docker and Analysis of its Performance,” IJCSNS Int. J. Comput. Sci. Netw. Secur., vol. 17, no. 3, pp. 228–235, 2017.

[9] M. Rexa, M. Data, and W. Yahya, “Implementasi Load Balancing Server Web Berbasis Docker Swarm Berdasarkan Penggunaan Sumber Daya Memory Host,” J. Pengemb. Teknol. Inf. dan Ilmu Komput. Univ. Brawijaya, vol. 3, no. 4, pp. 3478–3487, 2019.

[10] A. Y. Chandra, "Analisis Performansi Antara Apache & Nginx Web Server Dalam Menangani Client Request," J. Sist. dan Inform., vol. 14, no. 1, pp. 48–56, 2019, doi: 10.30864/jsi.v14i1.248.
[11] K. A. Agung Nugroho, Widhi Yahya, "Analisis Perbandingan Performa Algoritma Round Robin dan Least Connection untuk Load Balancing pada Software Defined Network," J. Pengemb. Teknol. Inf. dan Ilmu Komput., vol. 1, no. 12, pp. 1568–1577, 2017.
[12] O. H. Jader, S. R. M. Zeebaree, and R. R. Zebari, "A state of art survey for web server performance measurement and load balancing mechanisms," Int. J. Sci. Technol. Res., vol. 8, no. 12, pp. 535–543, 2019.
[13] D. Sharma, "Response Time Based Balancing of Load in Web Server Clusters," 2018 7th Int. Conf. Reliab. Infocom Technol. Optim. Trends Futur. Dir. ICRITO 2018, pp. 471–476, 2018, doi: 10.1109/ICRITO.2018.8748373.
[14] H. Ilham Reza Wijaya, Rendy Munadi, "Analisis Kinerja Load Balancing Menggunakan Algoritma Dynamic Ratio Pada Beban Tiga Web Server Analysis Performance Load Balancing Using Dynamic Ratio Algorithm on Three Web Server Loads," e-Proceeding Eng., vol. 6, no. 1, pp. 1–8, 2019, [Online]. Available: https://openlibrarypublications.telkomuniversity.ac.id/index.php/engineering/article/view/8493.
[15] Vmware, "CPU (%)," 2019. https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.vsphere.monitoring.doc/GUID-FC93B6FD-DCA 7-4513-A45E-660ECAC54817.html.