Analisis Perbandingan Kinerja Algoritma Load Balancer NGINX pada Studi Kasus PRS

William Chen, Agustinus Noertjahyana, Justinus Andjarwirawan
Program Studi Informatika Fakultas Teknologi Industri Universitas Kristen Petra
Jl. Siwalankerto 121 – 131 Surabaya 60236
Telp. (031) – 2983455, Fax. (031) – 8417658
E-Mail: chenwilliam862@gmail.com, agust@petra.ac.id, justin@petra.ac.id

ABSTRAK

Semakin berkembangnya teknologi tentu menimbulkan masalah yang baru, dimana aktivitas *client* akan semakin banyak untuk mengakses *server* yang ada. Salah satu contoh yaitu aktivitas Pendaftaran Rencana Studi Online pada Universitas Kristen Petra. Dimana kebanyakan mahasiswa melakukan *login* secara bersamaan ke dalam sistem PRS yang mengakibatkan akses terhadap *server* menjadi penuh.

Penelitian ini akan melakukan uji coba dengan implementasi *container* menggunakan *docker*. Teknologi *containerization* merupakan cara yang sangat efektif untuk menjalankan suatu aplikasi. Dimana dengan *containerization*, akan mengatur jalan masuknya dari *client* menuju ke *server* melalui bantuan *docker* serta dengan beberapa algoritma yang dimilikinya yaitu *Round Robin, hash* dan *Least Connection*.

Dengan pengujian yang dilakukan dapat menemukan algoritma dari *load balancer* NGINX apakah yang paling efektif serta efisien ketika mendistribusikan *workload* kepada *web service* yang ada pada studi kasus PRS online. Sehingga proses PRS kedepannya dengan menggunakan algoritma yang ada menjadi lebih efektif dan efisien.

Kata Kunci: *Docker, Load Balancer, NGINX, Container, Round Robin, Hash, Least Connection*

ABSTRACT

The development of technology will certainly create new problems, where more client activity will be available to access existing servers. One example is the activity of registering an online study plan at Petra Christian University. Where most students log in simultaneously into the PRS system which results in full access to the server.

This study will conduct a trial with the implementation of a container using a docker. Containerization technology is a very effective way to run an application. Where with containerization, it will set the entrance from the client to the server via the docker assistance and with some of the algorithms it has, namely Round Robin, hashes and Least Connection.

With the tests carried out it can find the algorithm of the NGINX load balancer which is the most effective and efficient when distributing workload to the web service available in the online PRS case study. So that the PRS process going forward using existing algorithms becomes more effective and efficient.

Keywords: *Docker, Load Balancer, NGINX, Container, Round Robin, Hash, Least Connection*

1. PENDAHULUAN

Teknologi telah banyak berkembang dan telah banyak membantu masyarakat untuk menyelesaikan masalah – masalah yang ada, salah satunya adalah teknologi *containerization*. Teknologi *containerization* merupakan cara yang sangat efektif untuk menjalankan suatu aplikasi. Untuk melakukan proses *containerization* sendiri terdapat beberapa software yang mendukung, antara lain: *Docker*, *Kubernetes*, dll.

Docker sendiri merupkan *software* yang bersifat *opensource* yang dibuat untuk memudahkan pembuatan aplikasi, menjalankan dan juga mengembangkannya menggunakan porses *containerization* [12]. Selain itu, *Docker* juga memiliki keunggulan sebagai berikut: proses *deployment* aplikasi yang cepat, memiliki kemudahan untuk dipindah – pindahkan ke mesin yang lain, pengontrolan terhadap versi dan komponen, *Lightweight footprint and minimal overhead*,dll [9].

Di penelitian ini, penulis akan mengimplementasikan sistem PRS online Universitas Kristen Petra dengan teknologi *containerization*. Dimana garis besar sistem PRS online mempunyai persamaan dengan sistem *flash sale* yaitu kedatangan user dalam jumlah yang besar dalam waktu bersamaan sehingga menimbulkan kenaikan yang sangat tajam pada *traffic*, dan jumlah *request* terhadap barang yang lebih besar daripada jumlah barang yang diinginkan.

Docker sendiri memerlukan sebuah mekanisme untuk mengatur proses – proses yang terdapat pada *container*, mekanisme yang digunakan antara lain dalah *load balancer*. Mekanisme yang akan diterapkan pada penelitian ini adalah *load balancer NGINX*. *NGINX* sendiri memiliki beberapa pilihan dalam melakukan pengaturan terhadap algoritma untuk mendistribusikan *workload*, yaitu: *Round Robin*, *Hash*, dan *Least Connection*

Penelitian ini berfokus untuk mengetahui algoritma mana yang paling efektif dan efisen ketika mendistribusikan *workload* kepada *web service* yang ada pada studi kasus PRS.

Parameter yang akan diujikan untuk penelitian ini adalah CPU dan memory load, server response time, total waktu yang dibutuhkan untuk merespon ke client, request yang gagal.

2. DASAR TEORI

2.1 Container

Container merupakan unit standar dari software yang memaketkan kode dan semua dependensinya, agar aplikasi dapat berjalan dengan cepat dan terpercaya melalui suatu lingkungan *computing* menuju ke lingkungan yang lainnya [3].

Ketika membahas tentang proses Virtualisasi, seringkali *container* dibandingkan dengan *Virtual Machine* dimana perbedaan yang menonjol adalah *container* memperbolehkan aplikasi digabungkan dengan *library* dan *dependencies*, sehingga menyediakan lingkungan komputasi yang terisolasi untuk menjalankan servis dari *software.* Disisi lain container juga tidak mengisolasi *operating system*, perangkat keras, serta *kernel*. Sedangkan *virtual machine* melakukan isolasi terhadap keselutruhan sistem

2.2 Docker

Docker merupakan platform software open source yang melakukan *containerization* yang diciptakan untuk memudahkan pembuatan, menyebarkan, dan menjalankan aplikasi dengan menggunakan container. Secara garis besar, proses *containerization* akan mengumpulkan sebuah software menjadi sebuah file sistem yang lengkap yang memiliki semua kebutuhan software untuk dapat berjalan (*runtime, code, system tools or system libraries*). *Docker* juga memberikan tambahan layer untuk *abstraction* dan juga *automation* dari virtualisasi pada level *operating system* dari *operating system* Linux. *Docker* menggunakan proses isolasi yang berbeda terhadap fitur – fitur dari Linux kernel dan juga memperbolehkan *container* independen untuk berjalan pada sebuah Linux *instance* dengan menggunakan file sistem yang berbasis *union-capable* [12].

2.3 Load Balancer

Load Balancer merupakan sebuah perangkat yang melakukan tugas sebagai *reverse proxy* dan melakukan distribusi terhadap jaringan atau proses *trafic* dari aplikasi melalui beberapa server. *Load balancer* digunakan untuk meningkatkan kapasitas dan *reliability* terhadap suatu aplikasi. Teknologi ini sangat membantu kinerja dari suatu aplikasi dengan cara mengecilkan beban dari server yang terhubung dengan aplikasi, mengatur dan mempertahankan *session* dari aplikasi dan jaringan, dan juga melakukan *application-spesific tasks.*

Pada umumnya, *Load balancer* dikelompokkan menjadi 2 kelompok, yaitu: *Layer* 4 dan *Layer* 7. Dimana *load balancer layer* 4 mengerjakan tugas yang berhubungan dengna *layer network* dan *transport protocol* seperti IP,TCP,FTP,UDP. Sedangkan *load balancer layer* 7 bertugas untuk melakukan distribusi *request* terhadap data yang ditemukan pada protokol lapisan aplikasi seperi HTTP. *Request* yang diterima akan melakukan distribusi berdasarkan algoritma yang dikonfigurasikan. Algoritma yang pada umumnya digunakan pada *load balancer* antara lain adalah *Round robin, weighted round robin, least connection, dan least response time* [4].

2.4 NGINX

NGINX adalah server HTTP yang gratis, *open-source* yang memiliki performa yang sangat tinggi ,dapat berfungsi sebagai *reverse proxy*, dapat berfungsi sebagi server *proxy* IMAP/POP3, dan juga . Mekanisme NGINX dikenal karena memiliki performa yang sangat tinggi ketika menjalankan proses, *stability, rich feature set,* konfigurasi yang mudah, dan penggunaan resource yang rendah. Tidak seperti kebanyakan server, NGINX tidak selalu bergantung pada *threads* untuk menangani *request,* melainkan NGINX menggunakan *scalable event-driven (asynchronus) architecture.* Arsitektur ini menggunakan memori yang lebih kecil daripada memori yang diprediksikan. Meskipun NGINX tidak menangani *request* yang sangat banyak, NGINX memiliki keuntungan yaitu performa yang tinggi dan memori yang kecil. NGINX dapat diaplikasikan mulai dari VPS yang kecil hingga server dengan *cluster* yang sangat besar [6].

2.5 Round Robin

Round Robin adalah metode yang paling sederhana untuk melakukan distribusi *request* yang dilakukan oleh *client* terhadap sekelompok server. Yang dimana algoritma akan membagikan *request* oleh *client* secara berurutan kepada server secara bergantian, dan ketika sudah berada pada server yang terakhir maka algoritma akan mendistribusikan *request* kepada server pertama dan akan terus berulang [7].

Keuntungan utama dari metode load balancing ini adalah metode ini adalah pengimplementasian yang sangat mudah. Namun, metode ini tidak selalu akurat dan efektif dalam melakukan distribusi *workload*. Dikarenakan, metode ini selalu menganggap bahwa server memiliki kapasitas kerja yang sama(server sedang *up/down*, server sedang menangani *workload* yang sama dengan server lainnya, dan kapasitas hardware yang dimiliki).

Secara garis besar, Round Robin sendiri dapat dibagi menjadi beberapa contoh lagi, yaitu: Weighted Round Robin dan Dynamic Round Robin. Dimana Weighted Round Robin memiliki parameter yang diberikan secara manual oleh *administrator*. Parameter tersebut menentukan jumlah proporsi *workload* yang akan dikerjakan pada tiap servernya. Contohnya, apabila **web1** memiliki weight 6 ,**web2** memiliki weight 3 ,**web3** memiliki weight 1 maka *workload* yang akan diberikan kepada tiap server adalah 60% web1, 30% web2, dan 10% web3. Sedangkan Dynamic Round Robin akan menentukan parameter secara *real-time* berdasarkan status *idle* dan *load* dari server sekarang [7].

2.6 Hash

Hash adalah metode load balancing dimana load balancer menghitung "*hash value*" dari tiap request dari client. Penghitungan "*hash value*" tersebut berdasarkan dari kombinasi teks dan variabel pad NGINX yang diinginkan dan menggabungkan "*hash value*" tersebut dengan salah satu servernya. Sehingga semua request yang dilakukan oleh "hash value" yang sama akan selalu tertuju kepada server tersebut.

Metode *hash* sendiri pada *load balancer NGINX* sendiri dibagi menjadi *hash* dan *IP hash. IP Hash* (Khusus HTTP) adalah variant yang telah ditentukan oleh metode *hash* dimana "*hash value*" yang didapatkan berdasarkan *IP address* dari *client* yang melakukan *request*. Untuk melakukan algoritma load balancer digunakan perintah *ip_hash.*

2.7 Least Connection

Least Connection adalah metode *load balancing* dimana *load balancer* akan membandingkan jumlah user yang terhubung pada tiap server, dan kemudian load balancer akan mengirimkan request pada server yan memiliki jumlah user terhubung yang paling sedikit [8].

Arsitektur dari *load balancer* ini dibentuk berdasarkan banyaknya jumlah user yang terhubung dengan sistem. Server yang bertugas sebagai *distributor workload* terhubung dengan *HTTP Request* , dan tiap kali user mengirimkan *HTTP Request, Request* tersebut akan diteruskan kepada server yang memiliki jumlah user terhubung yang paling sedikit [10].

3. DESAIN SISTEM

3.1 Analisa Permasalahan

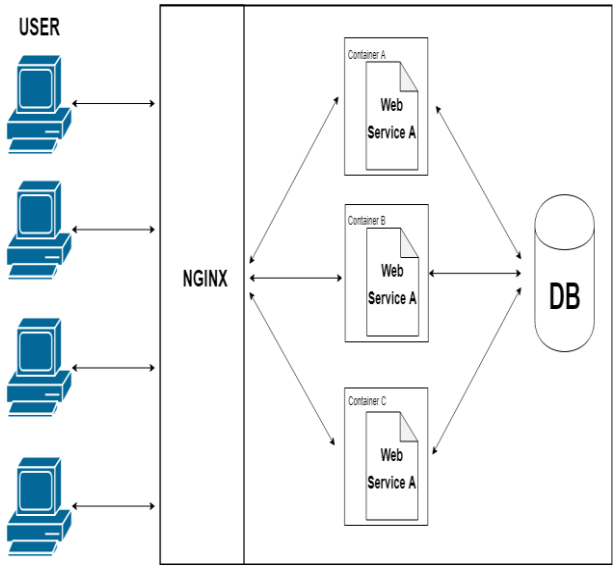
Penelitian tentang perbandingan algoritma *load balancer* yang dilakukan ini dilakukan berdasarkan permasalahan yang seringkali terjadi ketika mahasiswa dan dosen ketika melakukan proses PRS pada saat PRS I, antara lain:

- Diterapkannya pembagian waktu pada mahasiswa sehingga mahasiswa yang berbeda angkatan mengakses PRS pada waktu yang berbeda
- Banyaknya *traffic* data yang masuk ke dalam *server* Universitas Kristen Petra sehingga terkadang menyebabkan *server* Universitas Kristen Petra menjadi *down*.

Dengan adanya masalah seperti ini, salah satu solusi yang dapat diberikan adalah dengan menerapkan *load balancing*. Akan tetapi, *load balancing* sendiri memiliki beberapa algoritma yang dapat digunakan untuk melakukan pembagian *request* yang dimiliki. Oleh karena itu, maka dapat dilakukan pengujian untuk mengetahui algoritma apa yang efektif untuk menyelesaikan masalah tersebut.

3.2 Desain Implementasi Sistem

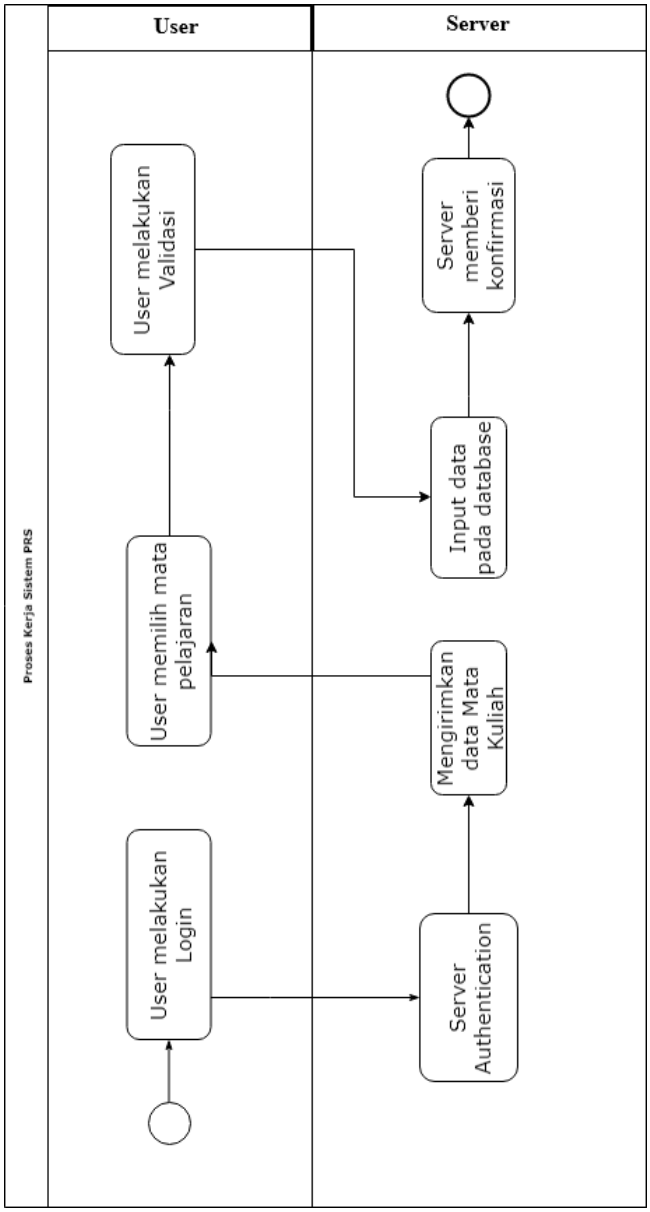
Desain dari sistem yang akan dibuat akan diimplementasikan pada satu sistem komputer, dimana dalam satu sistem komputer tersebut akan diinstal *docker* sebagai *platform* yang memberikan layanan untuk *containerization*. Kemudian pada *container* akan diinstall *NGINX* sebagai *load balancer* yang digunakan untuk pengujian. Selanjutnya *NGINX* akan membagikan *traffic* yang masuk kepada *container-container* yang lebih kecil dimana berisi *web service* dari PRS dan terhubung pada sebuah *database.* Pada Gambar 1 merupakan gambaran sederhana dari desain sistem yang akan diimplementasikan.



Gambar 1. Desain sistem implementasi

3.3 Flowchart Proses Kerja Sistem PRS Universitas Kristen Petra

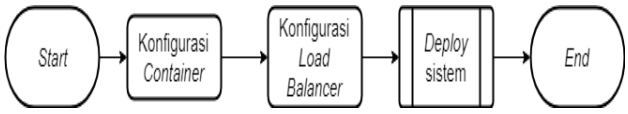
Proses kerja sistem PRS Universitas Kristen Petra pada saat ini dapat digambarkan pada Gambar 2.



Gambar 2. *Flowchart* sistem kerja PRS

3.4 *Flowchart* Proses Kerja Sistem Utama

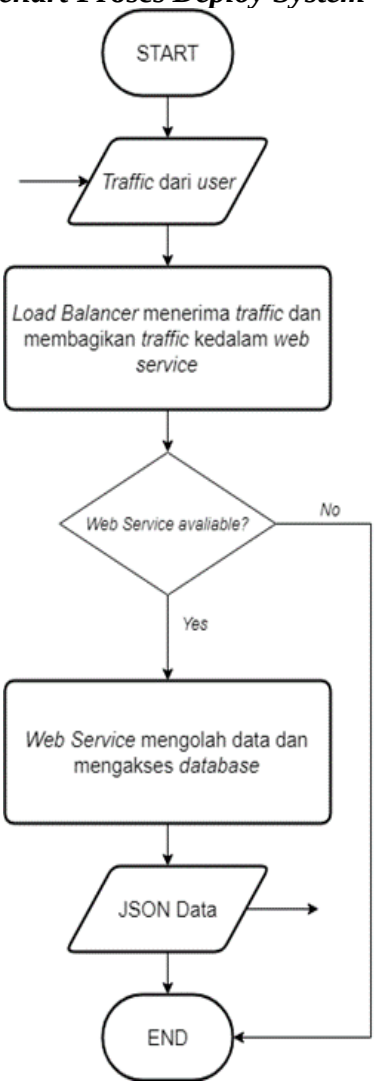
Berdasarkan desain dari sistem yang dijelaskan pada Gambar 1, maka keseluruhan proses kerja sistem dijelaskan pada Gambar 3 sebagai berikut.



Gambar 3. *Flowchart* keseluruhan kerja sistem

Proses pengerjaan yang dilakukan adalah diawali dengan melakukan konfigurasi pada *container docker*, kemudian dilanjutkan dengan konfigurasi kepada *load balancer NGINX*. Setelah selesai melakukan konfigurasi pada *container* dan *NGINX*, maka langkah terakhir sebelum melakukan pengujian adalah menerapkan web service yang telah dibuat kedalam sistem.

3.5 *Flowchart* Proses *Deploy System*



Gambar 4. *Flowchart* proses *deploy system*

Pada Gambar 4, proses kerja sistem dimulai dengan adanya *request* yang masuk dari *user* yang ingin melakukan pendaftaran pada sistem PRS. Kemudian proses yang masuk akan ditangkap oleh *NGINX* sebagai *load balancer*, sehingga *traffic* yang ada akan dibagikan dan diteruskan kepada *web service*. Sebelum masuk ke tahap selanjutnya, *load balancer* akan melakukan pengecekan apakah *web service* sedang dalam kondisi *available*. Apabila *web service* dalam kondisi *available* maka *web service* akan mengolah *traffic* dan mengakses database dan akan mengembalikan *JSON data* yang diinginkan dari *traffic* tersebut. Sedangkan apabila *web service* sedang tidak dalam kondisi *available* maka *traffic* akan ditahan pada *load balancer*.

4. PENGUJIAN SISTEM

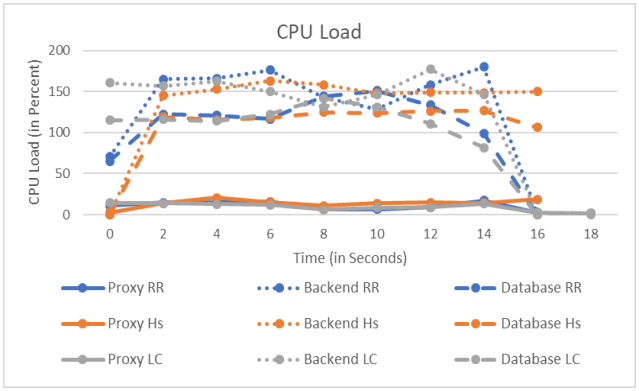
Pengujian dilakukan dengan menguji sistem yang dibuat digunakan spesifikasi komputer seperti berikut ini. Server yang digunakan untuk melakukan pengujian memiliki spesifikasi sebagai berikut:

- Server A
  - Processor : Intel Core i5-4570
  - RAM : 8 GB
  - IP Address : 192.168.11.49
- Server B
  - Processor : Intel Core i5-3340
  - RAM : 16 GB
  - IP Address : 192.168.11.48

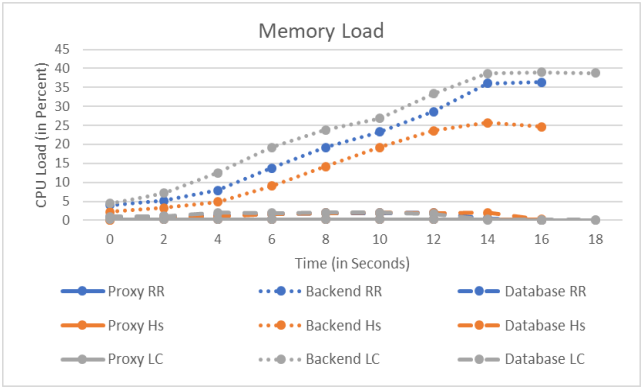
Kemudian untuk komputer yang digunakan sebagai client yang bertugas untuk menembakkan request memiliki spesifikasi sebagai berikut:

- Komputer 1
  - Processor : Intel Core i5-3340
  - RAM : 8 GB
  - IP Address : 192.168.11.110
- Komputer 2
  - Processor : Intel Core i5-3340
  - RAM : 8 GB
  - IP Address : 192.168.11.111
- Komputer 3
  - Processor : Intel Core i5-3340
  - RAM : 8 GB
  - IP Address : 192.168.11.112

Hasil dari pengujian yang dilakukan pada sistem yang diterapkan pada arsitektur diatas adalah sebagai berikut



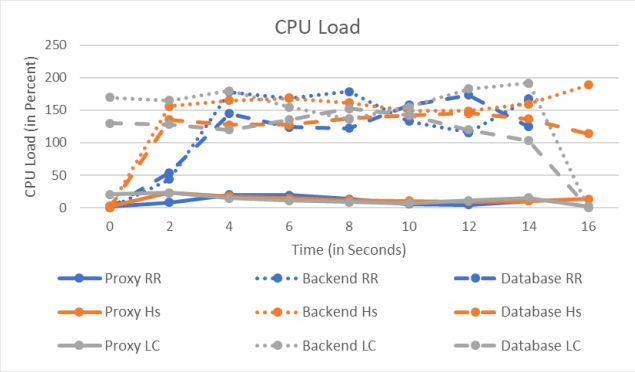
Gambar 5. CPU Load server A 1000 Request



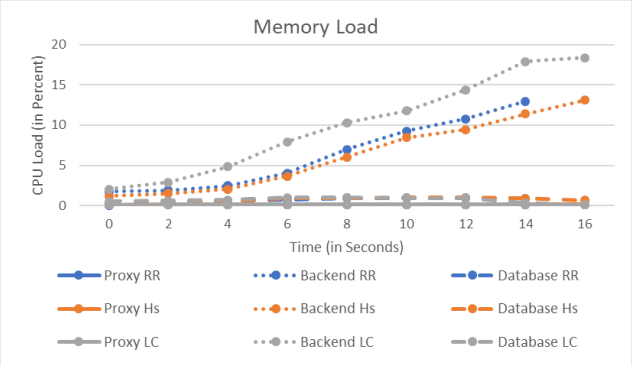
Gambar 6. *Memory Load server* A 1000 *Request*

Pada pengujian Server A 1000 request memiliki CPU Load seperti pada Gambar 5 dan *Memory load* pada Gambar 6. Pada pengujian dibandingkan 3 algoritma yang telah disebutkan diawal, yaitu: *Round Robin* (RR), *Hashing* (Hs), dan *Least Connection* (LC). Selain data *CPU load* dan *Memory load*, pengujian juga mendapatkan hasil *success rate*, dan *server response time. Success rate* dari algoritma Round Robin , Hashing, dan Least Connection berturut turut adalah 65,4%, 82,3%, dan 65,2%. Server response

time dari algoritma Round Robin , Hashing, dan Least Connection berturut turut adalah 16 *sec*, 17 *sec*, dan 18 *sec*.



Gambar 7. *CPU Load server* B 1000 *Request*



Gambar 8. *Memory Load server* B 1000 *Request*

Pada pengujian Server B 1000 request memiliki *CPU Load* pada Gambar 7 dan *Memory load* pada Gambar 8. Pada pengujian dibandingkan 3 algoritma yang telah disebutkan diawal, yaitu: *Round Robin* (RR), *Hashing* (Hs), dan *Least Connection* (LC). Selain data *CPU load* dan *Memory load*, pengujian juga mendapatkan hasil *success rate*, dan *server response time. Success rate* dari algoritma Round Robin , Hashing, dan Least Connection berturut turut adalah 60,3%, 72,1%, dan 62,1%. Server response time dari algoritma Round Robin , Hashing, dan Least Connection berturut turut adalah 15 *sec*, 17 *sec*, dan 15 *sec*.

5. KESIMPULAN

Berdasarkan hasil pengujian yang dilakukan pada sistem, maka dapat disimpulkan bahwa :

- Algoritma Load Balancer Hashing lebih efektif dan efisien untuk diterapkan pada studi kasus PRS dikarenakan penurunan *success rate* yang kecil ketika request yang ditembakan bertambah dan juga *success rate* yang tinggi ketika diberikan 1000 *request* dibandingkan dengan algoritma lainnya Algoritma Round Robin memiliki waktu yang paling cepat dalam menjalankan semua request.
- Kemampuan *Processor* lebih berpengaruh terhadap *success rate* daripada RAM dikarenakan *Server* A yang memiliki *processor* lebih bagus memiliki *success rate* yang lebih tinggi daripada *Server* B yang memiliki *memory* lebih banyak
- *Request* dengan *Method POST* menggunakan resource yang lebih ringan daripada *Request* dengan *Method GET*

6. DAFTAR PUSTAKA

[1] Andhikari, M. & Amgoth, T. 2018. Heuristic-Based Load-Balancing Algorithm for IaaS Cloud. *Future Generation Computer System*, *81*, p.156-165. DOI= https://doi.org/10.1016/j.future.2017.10.035

[2] Bokhari, M. & Hasan, F. 2019. Performance Analysis Of Dynamic Load Balancing Algorithm for Multiprocessor Interconnection Network. *Perspective in Science*, *8*, p.564-566. DOI= https://doi.org/10.1016/j.pisc.2016.06.021.

[3] Docker. *What is a Container?* .URI= https://www.docker.com/resources/what-container.

[4] F5. *Load Balancer* . URI= https://www.f5.com/services/resources/glossary/load-balancer

[5] Google. *Container at Google*. URI= https://cloud.google.com/containers/

[6] NGINX. *Welcome to NGINX Wiki!*. URI= https://www.nginx.com/resources/wiki/.

[7] NGINX. *What Is Round-Robin Load Balancing?*. URI= https://www.nginx.com/resources/glossary/round-robin-load-balancing/

[8] Nommensen, P. 2015. *Choosing an NGINX Plus Load-Balancing Technique*. URI= https://dzone.com/articles/choosing-an-nginx-plus-load-balancing-technique

[9] Redhat. *Advantages of using Docker*. URI= https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/7.0_release_notes/sect-red_hat_enterprise_linux-7.0_release_notes-linux_containers_with_docker_format-advantages_of_using_docker

[10] Shang L, & Yun Y, & Suang H. 2016. CLB: A novel load balancing architecture and algorithm for cloud services. *Computers & Electrical Engineering*, *58*, p.154-160. DOI= https://doi.org/10.1016/j.compeleceng.2016.01.029

[11] Showell, J. 2015. Containerisation vs Virtualisation – what's the difference?. URI= http://www.serverspace.co.uk/blog/containerisation-vs-virtualisation-whats-the-difference.

[12] Vivek R. 2017. *Docker: A Favourite in the DevOps World*. URI= http://opensourceforu.com/2017/02/docker-favourite-devops-world/

[13] Xu, J. 2018. *Developing a flash sale system*. URI= https://hackernoon.com/developing-a-flash-sale-system-7481f6ede0a3