IMPLEMENTASI *WEB SERVER CLUSTER* MENGGUNAKAN METODE *LOAD BALANCING* PADA *CONTAINER* DOCKER, LXC, DAN LXD

IMPLEMENTATION OF WEB SERVER CLUSTER USING LOAD BALANCING METHOD ON DOCKER, LXC, AND LXD CONTAINER

Rivaldy Arif Pratama[1], Ratna Mayasari, S.T., M.T.[2], Danu Dwi Sanjoyo, S.T., M.T.[3]
[1,2,3]Prodi S1 Teknik Telekomunikasi, Fakultas Teknik Elektro, Universitas Telkom
[1]rivaldy.arif@gmail.com, [2]ratnamayasari@telkomuniversity.ac.id, [3]danudwj@telkomuniversity.ac.id

Abstrak

Load balancing merupakan salah satu metode pada *cluster computing* yang dapat digunakan untuk meningkatkan skalabilitas serta mengurangi beban kerja sebuah *server* dengan cara mendistribusikan beban trafik ke beberapa *server cluster* secara seimbang agar trafik dapat berjalan dengan optimal, sehingga tidak menyebabkan *server* tersebut kelebihan beban (*overload*) atau bahkan *down*.

Dalam penerapannya, teknologi virtualisasi berbasis *container* dapat digunakan untuk mengimplementasikan *load balancing* dengan memanfaatkan HAProxy sebagai *load balancer*. *Container* merupakan sebuah teknologi virtualisasi yang dapat memungkinkan program yang berjalan didalamnya berhubungan langsung dengan linux *kernel* pada *host operating system*. Tidak seperti *Virtual Machine*, *container* tidak menggunakan *hardware* untuk virtualisasi. Pada beberapa penelitian, *container* dipercaya sebagai *platform* yang ringan dibanding *hypervisor*, sehingga menjadi alasan yang tepat dan memungkinkan untuk melakukan proses *load balancing* pada *container* dalam tugas akhir ini.

Tugas akhir ini akan diimplementasikan sistem *load balancing* yang dijalankan diatas tiga *container* yaitu Docker, LXC, dan LXD. Penelitian ini bertujuan untuk mengetahui kinerja dari ketiga *container* tersebut dari sisi layanan *web server* dan *resource utilization* pada saat *server* menggunakan *load balancing* dan tidak menggunakan *load balancing* (*single server*). Dari hasil pengujian yang dilakukan diketahui bahwa kinerja *server* yang menggunakan *load balancing* memiliki hasil yang lebih baik dibandingkan *single server*. Pada penelitian ini juga diketahui bahwa Docker menunjukkan hasil performansi yang lebih baik pada parameter *throughput*, *response time*, dan *request per second*. Sedangkan LXC menunjukkan hasil yang lebih baik pada parameter *request loss* dan CPU *utilization*. Dari segi *fairness*, algoritma *Round Robin* lebih *fair* dibanding *Least Connection* dengan nilai *fairness index* mencapai 1.

Kata kunci : ***Load Balancing, Cluster Computing, Container, Docker, LXC, LXD***

Abstract

Load balancing is a method in cluster computing that can be used to increase the scalability and decrease the server workloads by distributing traffic load to several clusters of server in equals, in order that the traffic can run optimally, so that it doesn't cause the server becomes overload or down.

In it's implementation, the container-based virtualization can be used to implement the load balancing by using HAProxy image as the load balancer. Container is the virtualization technology that allows the program run directly connects with linux kernel in host operating system. Different from Virtual Machine, container doesn't use hardware to virtualize. In some researches, it's believed that container is more lightweight than hypervisor, so that it allowed to process load balancing on the container in this final project.

This final project will implement load balancing system that are run on three containers, those are Docker, LXC, and LXD. This research is to know the performance from those three containers from web server services side and resource utilization when server are using load balancing and not (single server). From the research, we know that the performance of server using load balancing had better result than the single server. In this final project we also got that Docker showed better performance on throughput, response time, and request per second. While LXC showed better result on request loss and CPU utilization parameters. In terms of fairness, Round Robin algorithm is more fair than Least Connection with the fairness index is 1.

Keywords : Load Balancing, Cluster Computing, Container, Docker, LXC, LXD

1. Pendahuluan

Penggunaan *web* sebagai salah satu media penyampaian dan penerimaan informasi saat ini semakin banyak digunakan, karena dengan mudah pengguna mendapatkan informasi yang dibutuhkan dalam waktu yang relatif singkat dan dapat diakses dimana saja. Meningkatnya permintaan akan kebutuhan informasi dalam internet menyebabkan trafik dalam internet juga semakin padat. Semakin meningkatnya jumlah trafik dalam internet akan menyebabkan beban kerja pada suatu penyedia layanan informasi yaitu *web server* akan mengalami kelebihan beban, sehingga dapat menyebabkan *server* tersebut *down*.

Untuk mengatasi masalah kelebihan beban dalam *server* dapat diterapkan konsep *cluster computing*, yaitu suatu teknologi menggabungkan beberapa komputer/*server* yang bekerja bersama-sama yang seolah-olah merupakan satu sistem tunggal [1]. Dalam pengimplementasiannya, sistem *cluster computing* dibangun menggunakan metode *load balancing* yaitu teknik untuk mendistribusikan beban trafik ke beberapa *server cluster* secara seimbang [2], agar trafik dapat berjalan dengan optimal. Penerapan *load balancing* pada *web server cluster* sangatlah penting dan dapat menjadi solusi dalam menangani beban *server* yang terlalu berat akibat banyaknya permintaan (*request*) sehingga dapat meningkatkan skalabilitas pada sistem terdistribusi [3].

Dengan memanfaatkan HAProxy sebagai *load balancer* diharapkan *server* dapat menangani permasalahan beban yang sangat berat (*overload*) terhadap permintaan (*request*) [3] tanpa mengalami kelebihan beban. *Load balancer* dibangun dalam sebuah *container*, yang merupakan teknologi virtualisasi pada level OS yang dapat mengisolasi lingkungan program pada *host operating system*. Dengan kelebihan *container* yang mudah dikelola dan sangat ringan, sehingga memungkinkan untuk melakukan proses *load balancing* dalam penelitian tugas akhir ini.

Pada tugas akhir ini akan dilakukan *load balancing* menggunakan HAProxy pada *multi container/web server cluster* untuk mendapatkan hasil yang maksimal, dengan menerapkannya pada ketiga *container* yaitu Docker, LXC, dan LXD untuk melihat dan menguji performansi yang dihasilkannya berdasarkan parameter uji *throughput*, *response time*, *request* per *second*, *request loss*, dan CPU *utilization*.

2. Dasar Teori

2.1 Cluster Computing

Cluster komputer merupakan jenis komputasi dimana beberapa *node* atau komputer dibuat untuk dijalankan sebagai satu entitas. Berbagai *node* yang terlibat dalam *cluster* saling terhubung satu sama lain menggunakan jaringan berkecepatan tinggi, seperti gigabit Ethernet. *Nodes* atau komputer-komputer tersebut bekerja sama secara kooperatif untuk mengerjakan suatu pekerjaan atau perhitungan intensif yang tidak layak untuk dijalankan pada satu komputer.

Ada dua alasan utama menggunakan *cluster* komputer dibanding *single* komputer yaitu dari segi performansi dan toleransi kesalahan. *Cluster* komputer menyediakan perhitungan yang tinggi dengan menggunakan pemrograman paralel, yang menggunakan banyak prosesor secara bersamaan untuk sejumlah atau satu masalah [4]. Tidak hanya itu, *cluster* komputer juga menyediakan kecepatan pemrosesan lebih cepat, kapasitas penyimpanan yang lebih besar, integritas data yang lebih baik, dan kehandalan yang superior seiring dengan ketersediaan sumber daya yang lebih luas [5].

Cluster computing diklasifikasikan menjadi 3 kategori yaitu [1] :

1. *High-availability Clusters*, juga dikenal sebagai HA *cluster* atau *failover cluster* yaitu kelompok komputer yang mendukung aplikasi *server* yang dapat digunakan dengan andal dengan minimal *down time*. Mereka beroperasi dengan memanfaatkan komputer yang berlebihan dalam grup atau *cluster* yang menyediakan layanan lanjutan saat komponen sistem gagal. Tanpa *clustering*, jika *server* yang menjalankan aplikasi tertentu mogok, aplikasi tidak akan tersedia sampai *server* yang mogok diperbaiki. HA *clustering* memperbaiki situasi ini dengan mendeteksi kesalahan perangkat keras/perangkat lunak, dan segera memulai kembali aplikasi di sistem lain tanpa memerlukan intervensi administratif, sebuah proses yang dikenal sebagai failover.
2. *Load Balancing Clusters*, adalah metode jaringan komputer untuk mendistribusikan beban kerja ke beberapa sumber komputasi, seperti komputer, *cluster* komputer, *network links*, dll. *Load balancing* bertujuan untuk mengoptimalkan penggunaan sumber daya, memaksimalkan *throughput*, meminimalkan waktu respon, dan menghindari kelebihan sumber daya apapun. Ada beberapa metode *load balancing* seperti pendekatan algoritma *Round Robin*, *Least Connection*, dll. *Load balancing* sering diperlukan saat membangun solusi yang menangani permintaan klien dalam jumlah besar atau memiliki tuntutan keamanan dan redundansi yang tinggi.
3. *High Performance Computing Clusters*, menggunakan *node cluster* untuk melakukan penghitungan bersamaan. Sebuah *high performance cluster* memungkinkan aplikasi bekerja secara paralel, sehingga meningkatkan kinerja aplikasi.

2.2 Virtualisasi

Virtualisasi adalah teknik membuat versi maya (*virtual*) dari suatu sumber daya (*resource*) sehingga pada satu sumber daya fisik dapat dijalankan atau disimpan beberapa sumber daya *virtual* sekaligus, dengan syarat unjuk kerja masing-masing sumber daya *virtual* itu tidak berbeda signifikan dengan sumber daya fisiknya [6]. Tujuan dari virtualisasi adalah kinerja tingkat tinggi, ketersediaan, skalabilitas, keandalan, ketangkasan, atau untuk membuat dasar keamanan dan manajemen yang terpadu [7]. Hingga saat ini sumber daya yang dapat divirtualisasikan antara lain adalah perangkat keras komputer (*hardware*), media penyimpan data (*storage*), *operating system* (OS), layanan jaringan (*networking*) dan teknologi ini masih berkembang terus.

2.3 Virtualisasi Container

Container adalah teknik virtualisasi pada level sistem operasi dimana tiap proses atau aplikasi yang dijalankan tiap *container* memiliki *kernel* yang sama. Hal ini menjadi keuntungan sendiri dibandingkan virtualisasi pada level mesin (*Virtual Machine*), dimana *virtual machine* membutuhkan *kernel* sistem operasi yang berbeda-beda tiap aplikasi yang dijalankan [8].

2.4 Docker

Docker adalah suatu *platform* terbuka bagi pengembang perangkat lunak dan pengelola sistem jaringan untuk membangun, mengirim, dan menjalankan aplikasi-aplikasi terdistribusi. Dalam bukunya yang berjudul "*Build Your Own PaaS with Docker*" [9], Oskar Hane mengatakan bahwa Docker adalah suatu cara untuk memasukkan layanan kedalam lingkungan yang terisolasi, yang disebut *container*, sehingga layanan tersebut dapat dikemas menjadi satu bersama dengan semua pustaka dan *software* lain yang dibutuhkan dan juga pengembang dapat memastikan bahwa layanan akan jalan dimanapun Docker berjalan.

2.5 LXC

Linux *Container* (LXC) adalah teknologi virtualisasi ringan yang bergantung pada *userspaces* yang terisolasi untuk membuat *virtual container* yang memiliki *kernel* yang sama dengan *host OS*. LXC menggunakan fitur *kernel* Linux seperti *namespaces*, *Chroots*, *CGroups*, dan sebagainya, untuk mengisolasi proses kontainer. LXC berbeda dengan konsep *virtual machine* (VM) dengan berbagi *kernel* dan perangkat keras yang sama yang menghindarkan kebutuhan akan *hypervisor* [10].

2.6 LXD

LXD diperkenalkan sebagai solusi sistem *container* OS yang ringan di atas Linux *Container* (LXC). Sementara Docker dirancang untuk meng-*host* aplikasi tunggal, LXD sebanding dengan *virtual machine* dengan kemampuan untuk meng-*host* beberapa *container* OS di satu *host* tunggal. LXD juga menggunakan Linux *Container* (LXC) APIs melalui liblxc dan satu set *Go bindings* untuk membuat dan mengelola *container* [11].

2.7 HAProxy

HAProxy *load balancer* merupakan sebuah aplikasi bersifat *open source* yang sangat cepat dan handal yang menawarkan fitur *high availability*, *load balancing*, dan *proxy* untuk aplikasi berbasis TCP dan HTTP. Aplikasi ini sangat cocok digunakan untuk *website* dengan trafik lalu lintas yang sangat tinggi [12]. HAProxy dipasang pada *front-end server*, sedangkan trafik yang diterima oleh *front-end server* akan diteruskan ke *back-end server*. Adapun algoritma penjadwalan *load balancing* yang bisa digunakan pada HAProxy yaitu [13] :

- *Round Robin*
- *Weighted Round Robin*
- *Static Round Robin*
- *Least Connection*

2.8 Web Server

Web server biasanya disebut juga sebagai HTTP *server* karena basisnya menggunakan *protocol* yang disebut *Hypertext Transfer Protocol* (HTTP). *Web server* merupakan perangkat lunak yang melayani permintaan HTTP (HTTP *request*) dari *web browser* dan menyediakan layanan akses ke suatu berkas, berkas tersebut dapat berupa *Hypertext Markup Language* (HTML), berkas *Javascript*, dan berkas *Perl* [14].

2.9 Apache HTTP Server

Apache merupakan *web server* unggul dibandingkan banyak *web server* berbasis unix lainnya dalam hal fungsi, efisiensi dan kecepatan. Apache juga merupakan sebuah aplikasi *multi-platform*, *responsive*, dan juga *open-source*. Apache dapat dijalankan di banyak sistem operasi (BSD, Linux, Microsoft Windows dan Novell Netware serta *platform* lainnya) karena merupakan perangkat lunak sumber terbuka dikembangkan oleh komunitas terbuka yang terdiri dari pengembang dibawah naungan Apache Software Foundation. Hampir 50% dari penyedia layanan *web* dalam sepuluh tahun terakhir ini menggunakan apache [15].

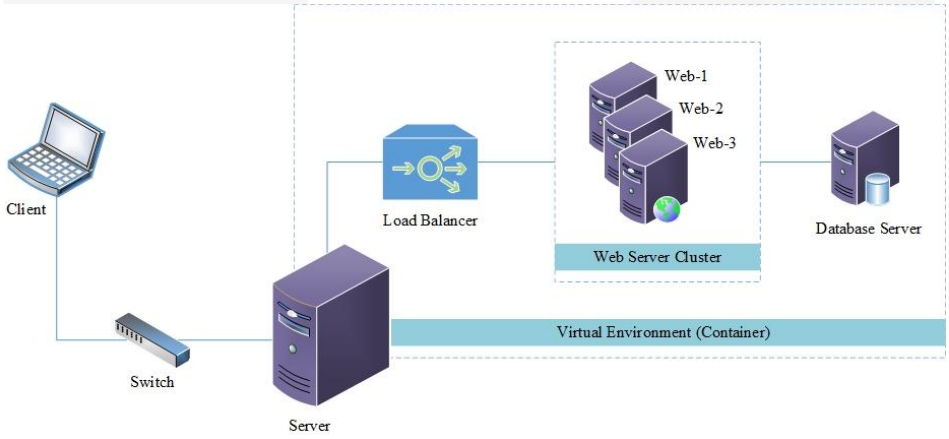
2.10 Redis

Redis (*Remote Dictionary Server*) adalah salah satu *database* dari dunia NoQSL yang berbasis *key-value store*. Redis bersifat *opensource* yang ditulis dalam bahasa pemrograman C. Sistemnya yang *in-memory* membuat pengambilan data dari Redis menjadi lebih cepat, namun dapat juga bersifat *persistent* bila ingin menyimpan data ke *disk*. Redis memiliki sejumlah *query* yang sangat mudah digunakan untuk menyimpan mulai dari data sederhana hingga data kompleks [16].

3. Perancangan Sistem

3.1 Desain Model Sistem

Desain model secara keseluruhan dari sistem yang digunakan dalam tugas akhir ini dapat ditunjukkan pada gambar dibawah ini.



Gambar 3.1 Desain Model Sistem

Berdasarkan desain perancangan sistem diatas, sistem ini akan bekerja dengan cara melakukan *load balancing* terhadap permintaan *client* berupa HTTP *request* kepada *web server*. Sistem ini menggunakan HAProxy *Load Balancer* sebagai *tools* yang dapat digunakan untuk melakukan proses *load balancing*, Apache HTTP *server* sebagai *web server*, dan *container* sebagai wadah untuk pembuat lingkungan *virtual* serta sistem *clustering* di sisi *server*, adapun *container* yang digunakan yaitu Docker, LXC, dan LXD. Sedangkan pada sisi *client* akan dibangkitkan sejumlah *request* kepada *server* menggunakan *software* untuk *load testing* yaitu Siege. Pada sistem ini *load balancer* dan *web server* akan dibangun dan dijalankan diatas *container*, yang nantinya akan dilakukan perbandingan sejauh mana peningkatan kinerja *server* pada masing-masing *container* yaitu Docker, LXC, dan LXD saat dilakukan *load balancing* dengan menggunakan algoritma penjadwalan *Round Robin* dan penjadwalan *Least Connection* pada HAProxy *Load Balancer*.

3.2 Parameter Pengujian

Berdasarkan jurnal Shreyas Mulay dan Sanjay Jain [17] untuk melakukan pengujian pada sistem *load balancing*, parameter yang diukur meliputi *throughput*, *response time*, *request per second*, *request loss*, dan CPU *utilization*. Serta mengacu pada jurnal Saleem Bhatti dkk [18] untuk pengukuran parameter *fairness index*. Berikut adalah parameter yang diujikan :

1. *Throughput*

   Throughput adalah *bandwidth* aktual yang terukur pada suatu ukuran waktu tertentu dalam suatu jaringan. Pengujian *throughput* merupakan parameter variabel dari QoS (*Quality of Service*) dengan tujuan untuk melihat performansi jaringan dari segi kecepatan pengiriman paket yang dapat dikirim dengan memanfaatkan *bandwidth* yang tersedia.

$$Throughput = \frac{Jumlah\ data\ yang\ diterima}{waktu\ pengiriman\ data} \quad (3.1)$$

2. *Response Time*
   Response Time adalah waktu yang dibutuhkan untuk menyelesaikan satu *request* dan mengirimkannya kembali ke *client*. Pengujian *response time* bertujuan untuk mengukur seberapa cepat suatu *container* dapat menerima *request* dari *client*.
3. *Request* per *second*
   Pengujian *request* per *second* dilakukan dengan cara mengirimkan sejumlah layanan per satuan detik dari *client* yang akan dikirimkan ke *web server* guna untuk mengetahui kinerja pada suatu *server* yang menggunakan sistem *load balancing*. Hal ini bertujuan untuk melihat berapa besar *request* yang dapat ditangani oleh *server* yang berjalan diatas *container*.
4. *Request loss*
   Pada saat melakukan pengiriman paket, paket yang dikirim memiliki kemungkinan untuk dibuang karena *resources* menampung lebih dari kapasistas maksimum. Proses pembuangan paket ini disebut dengan *request loss*. Parameter *request loss* dapat diukur dengan menggunakan aplikasi siege dengan membebani *server* dengan *request* hingga *resource* pada *server* mencapai batas maksimum.
5. *Fairness Index*
   Pengukuran *fairness index* digunakan pada jaringan komputer untuk menentukan apakah *users* atau aplikasi telah menerima sumber daya yang adil. Sebuah *metric* yang digunakan secara umum untuk menaksir *fairness* adalah *Jain's Fairness Index* (JFI) dengan persamaan [18] :

$$Fairness\ Index = \frac{(\sum X_i)^2}{n \sum {X_i}^2} \quad (3.2)$$

   Dimana Xi merupakan nilai kelengkapan yang ditaksir aliran yaitu jumlah *request* yang diterima oleh setiap *web server cluster*, sedangkan n merupakan jumlah jalur yang ada. Nilai dari *fairness index* selalu berada diantara 0 dan 1. *Fairness index* sama dengan 1 artinya ada keseimbangan atau kewajaran (*fairness*) pada semua jalur yang ada.
6. CPU *Utilization*
   Pengujian ini bertujuan untuk melakukan peninjauan terhadap penggunaan *resources* oleh *container* dari segi penggunaan CPU (*processor*) untuk melihat apakah terjadi perbadaan penggunaan CPU pada masing-masing *container* saat menjalankan *load balancing*.

3.3 Skenario Pengujian

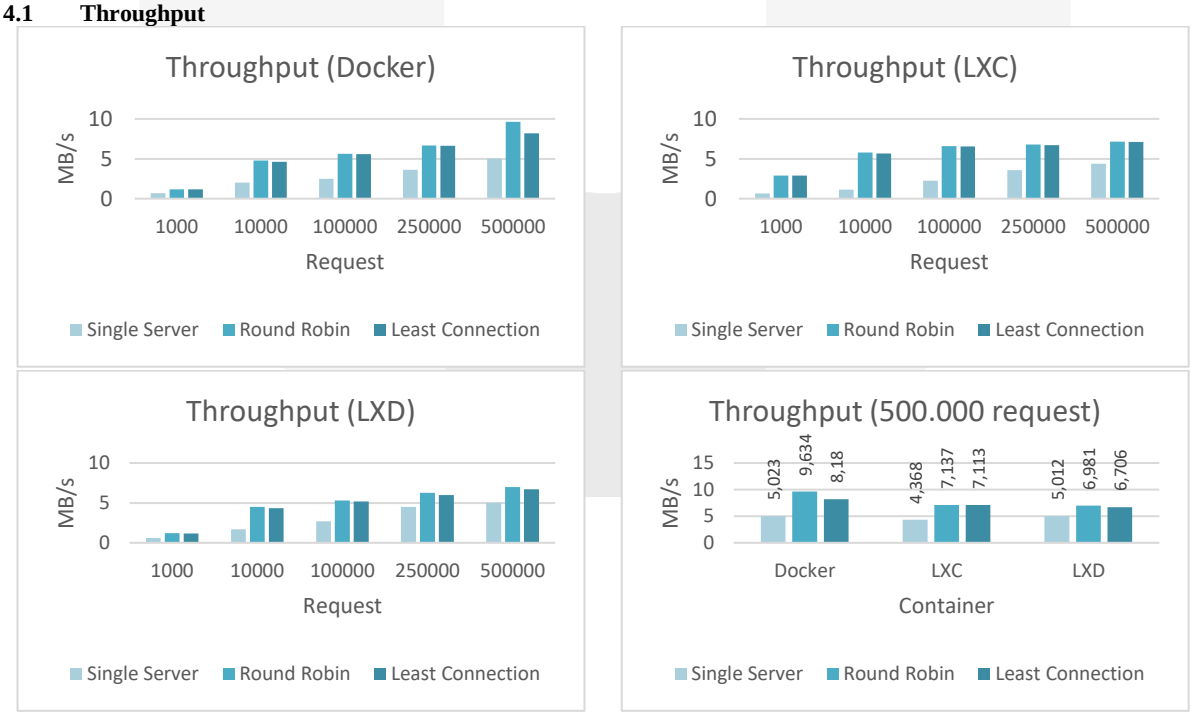
Pengujian akan dilakukan pada masing-masing *container*, yaitu pengujian untuk melihat kemampuan *load balancing* dan tanpa *load balancing* pada *container* Docker, pengujian pada *container* LXC, dan pengujian pada *container* LXD. Mengacu pada jurnal Sebastian Dabkiewicz [19], banyaknya permintaan (*request*) yang bisa digunakan untuk menguji performansi sebuah *web server* yaitu sebesar 1000, 10.000, 100.000, 250.000, dan 500.000 *request*. Adapun skenario pengujian adalah sebagai berikut:

1. Pengujian *load balancing* dengan algoritma *Round Robin*
2. Pengujian load balancing dengan algoritma *Least Connection*
3. Pengujian pada *Single Server* (tanpa *load balancing*)

4. Hasil Pengujian dan Analisis

Berikut adalah hasil pengujian dari 30 kali pengambilan data pada setiap parameter yang diuji :

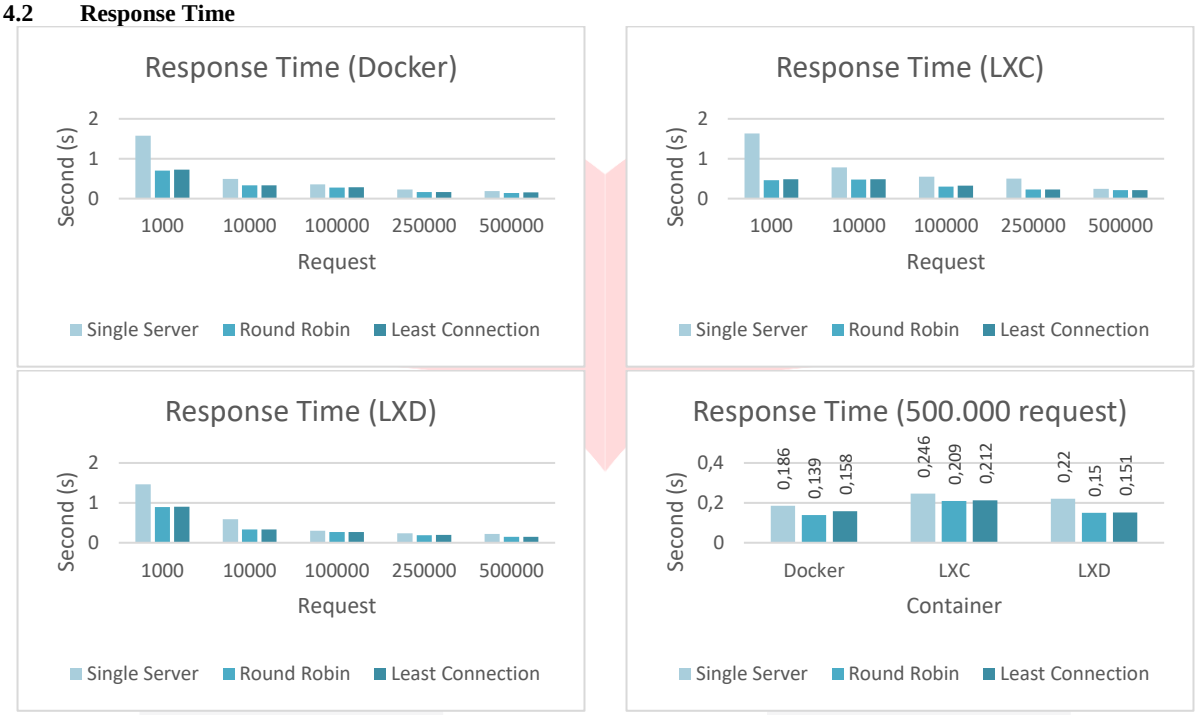
4.1 Throughput



Gambar 4.1 Grafik Hasil Pengujian *Throughput* pada Docker, LXC, dan LXD

Dari grafik diatas menunjukkan bahwa nilai *throughput* cenderung naik seiring bertambahnya jumlah *request* yang diberikan, semakin besar nilai *throughput* yang dihasilkan maka semakin bagus performansinya. Hasil pengukuran secara keseluruhan menunjukkan bahwa nilai *throughput* lebih besar saat sistem menggunakan *load balancing* dibandingkan dengan nilai *throughput* pada sistem yang tidak menggunakan *load balancing* (*Single Server*). Hal ini dikarenakan pada sistem yang menggunakan *load balancing* terdapat lebih dari satu *server* yang dapat melayani permintaan layanan yang datang secara bersamaan. Didapatkan nilai *throughput* maksimal pada Docker sebesar 9,634 MB/s dengan algoritma *Round Robin*.

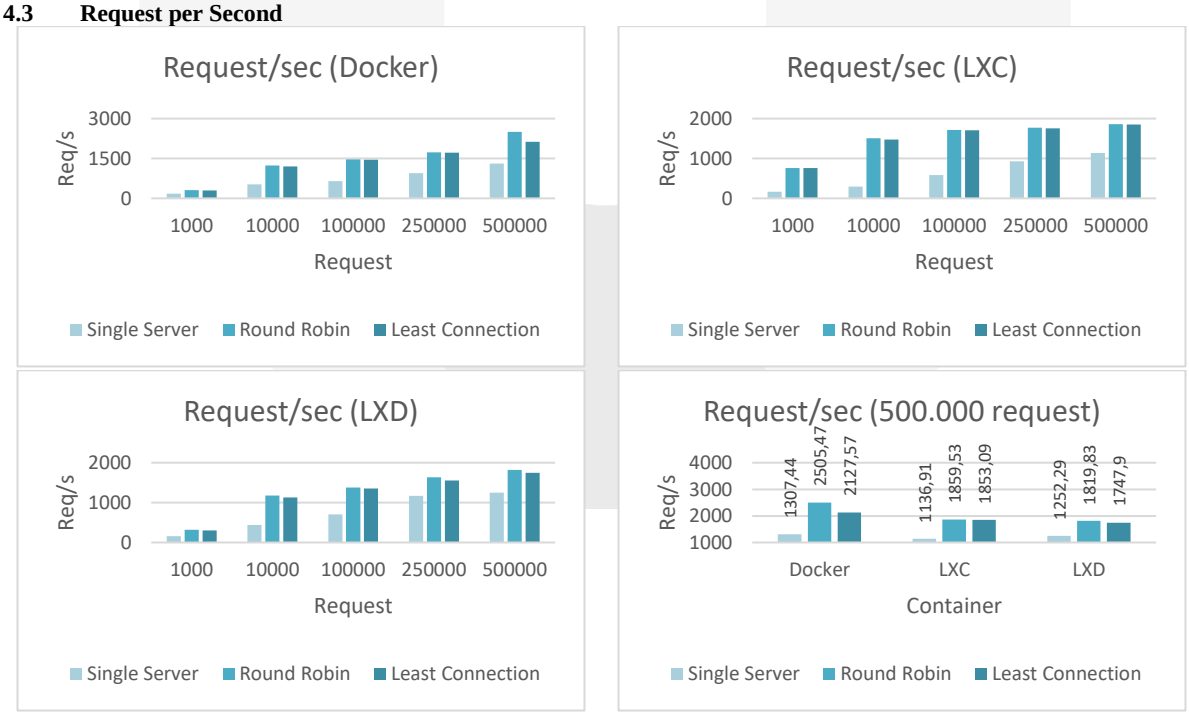
4.2 Response Time



Gambar 4.2 Grafik Hasil Pengujian *Response Time* pada Docker, LXC, dan LXD

Hasil pengukuran pada ketiga *container* secara keseluruhan dapat dilihat bahwa *Single Server* memiliki nilai *response time* yang lebih besar dibandingkan sistem yang menggunakan *load balancing*, hal tersebut dikarenakan pada *Single Server* hanya menggunakan satu buah *web server* saja oleh sebab itu waktu yang dibutuhkan oleh *Single Server* lebih lama untuk menyelesaikan *request* dari *client*. Dari hasil pengujian *response time* yang dilakukan, nilai rata-rata *response time* yang didapat cenderung menurun seiring dengan naiknya nilai *throughput* yang didapat, semakin besar nilai *throughput* yang didapat pada saat pengujian maka waktu yang dibutuhkan oleh sebuah *web server* untuk mengirimkan http *response* ke *client* akan semakin cepat. Didapatkan nilai *response time* minimum yaitu 0,139 *second* pada Docker yang menggunakan algoritma *Round Robin*.

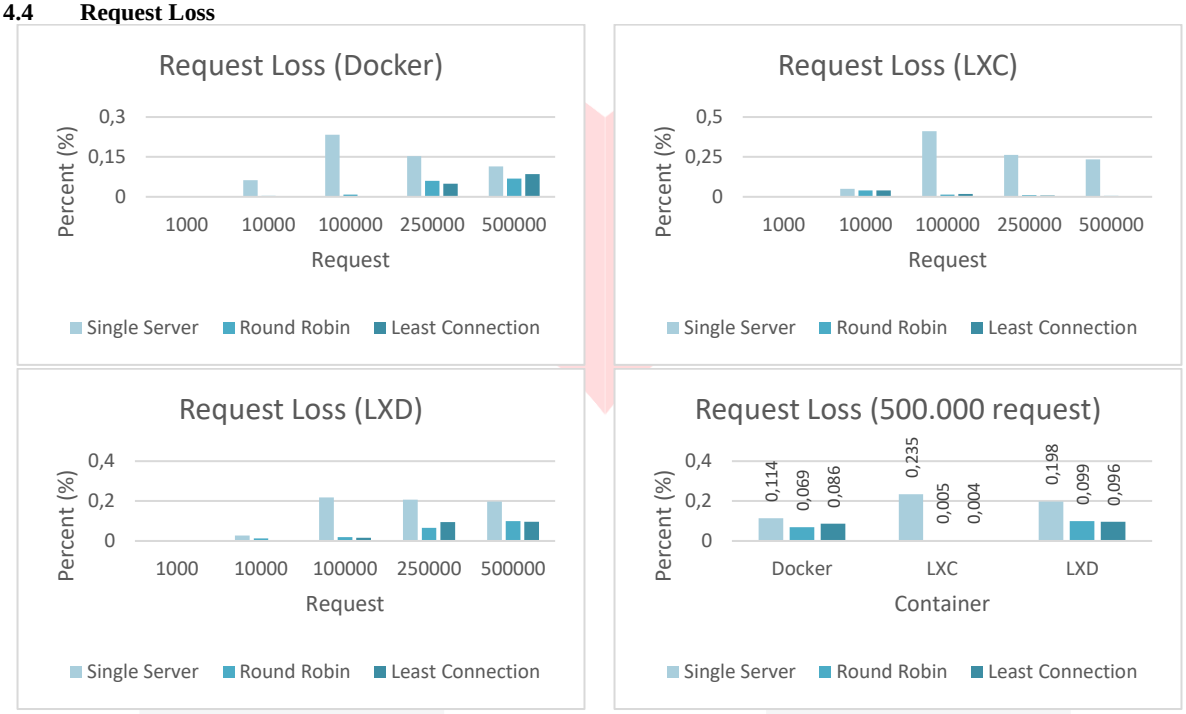
4.3 Request per Second



Gambar 4.3 Grafik Hasil Pengujian *Request per Second* pada Docker, LXC, dan LXD

Terlihat bahwa jumlah *request* yang dapat dilayani per satuan detik pada sistem yang menggunakan *load balancing* jauh lebih besar dibanding dengan *Single Server*. Hal ini dikarenakan pada sistem yang menggunakan *load balancing* baik dengan algoritma *Round Robin* maupun *Least Connection* permintaan (*request*) yang datang dilayani oleh tiga buah *web server cluster* yang saling bekerja sama sehingga jumlah *request* yang dapat dilayani meningkat dibanding pada *Single Server* yang hanya menggunakan satu buah *web server*. Didapatkan nilai maksimal *request per second* pada Docker sebesar 2505,47 *request/sec* dengan algoritma *Round Robin*. Dengan demikian dapat diambil kesimpulan bahwa nilai *throughput* yang didapat yaitu berbanding lurus dengan nilai *request per second* yang didapat.

4.4 Request Loss



Gambar 4.4 Grafik Hasil Pengujian *Request Loss* pada Docker, LXC, dan LXD

Dari hasil pengukuran tersebut, nilai *request loss* baik pada sistem yang menggunakan *load balancing* dan tanpa *load balancing* mulai terlihat ketika *server* menerima beban *request* lebih besar sama dengan 10.000 *request*. Pada sistem yang menggunakan *load balancing* juga terdapat nilai *request loss* akan tetapi nilai tersebut lebih kecil dibandingkan *Single Server* karena semua *request* yang datang didistribusikan kepada lebih dari satu *web server* secara merata, artinya tidak dibebankan pada satu buah *server* saja. Nilai *request loss* minimum terdapat pada LXC yaitu 0,004% dengan algoritma *Least Connection*.

4.5 Fairness Index

Tabel 4.1 Hasil *Fairness Index* pada Docker

Jumlah *Request*	*Fairness Index* (Docker)	
	Round Robin	*Least Connection*
1000	0.999998	0.999506
10000	0.99999998	0.999938064
100000	1	0.999996147
250000	1	0.999999835
500000	1	0.999999191

Tabel 4.2 Hasil *Fairness Index* pada LXC

Jumlah *Request*	*Fairness Index* (LXC)	
	Round Robin	*Least Connection*
1000	0.999998	0.997550017
10000	0.99999998	0.999891872
100000	1	0.999968695
250000	1	0.999998006
500000	1	0.999998753

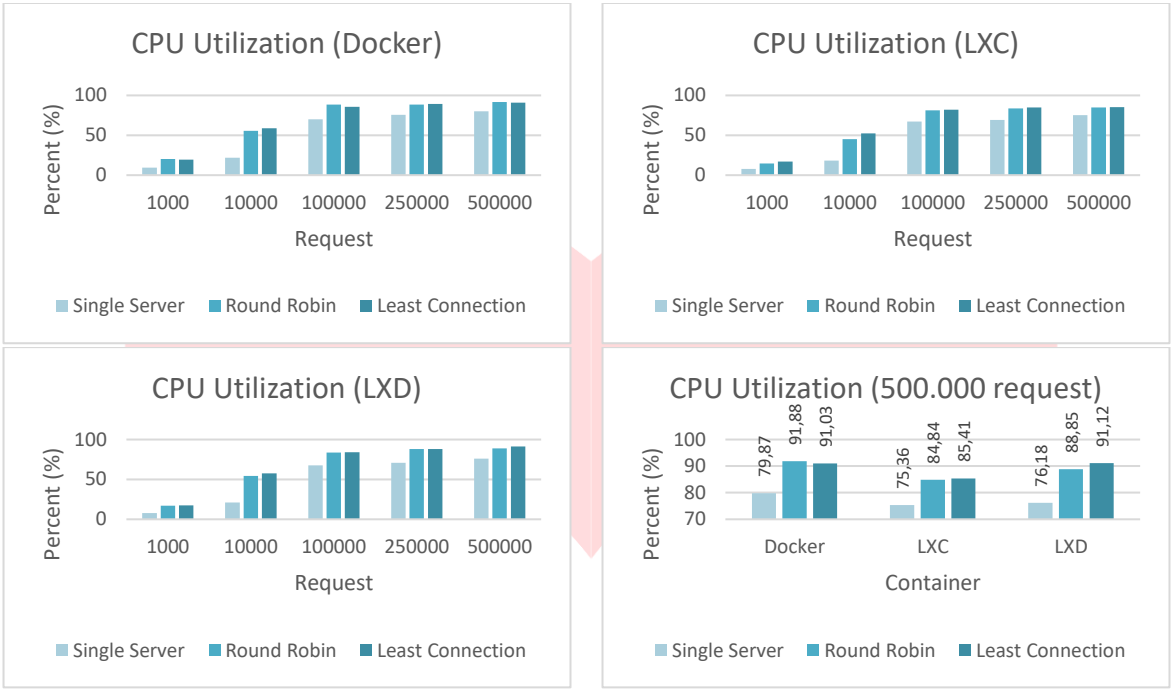
Tabel 4.3 Hasil *Fairness Index* pada LXD

Jumlah *Request*	*Fairness Index* (LXD)	
	Round Robin	*Least Connection*
1000	0.999998	0.999782048
10000	0.99999998	0.999508701
100000	1	0.999560707
250000	1	0.999617841
500000	1	0.999366706

Nilai *fairness index* yang dihasilkan oleh algoritma *Round Robin* hampir mendekati angka 1 untuk jumlah *request* 1000 dan 10.000, sedangkan untuk jumlah *request* 100.000, 250.000, dan 500.000 mendapat angka *fairness index* yaitu 1. Hasil

pengukuran secara keseluruhan menunjukkan bahwa *Round Robin* memiliki nilai *fairness index* yang lebih unggul dibanding *Least Connection*, artinya sistem yang menggunakan algoritma *Round Robin* lebih *fair* dalam pendistribusian beban.

4.6 CPU Utilization



Tabel 4.4 Grafik Hasil Pengujian CPU *Utilization* pada Docker, LXC, dan LXD

Pada pengujian CPU *utilization*, nilai penggunaan CPU didapat dengan menggunakan aplikasi htop untuk memonitoring *resource* yang terpakai pada *server*, serta digunakan psutil untuk mencatat hasil dari penggunan CPU yang terpakai. Semakin kecil persentase penggunaan CPU yang dihasilkan maka semakin bagus sistem tersebut dan juga semakin ringan sistem tersebut untuk dijalankan pada *server*. Pada grafik terlihat bahwa *Single Server* selalu mendapat nilai CPU *utilization* lebih rendah dibanding sistem yang menggunakan sistem *load balancing*. Hal tersebut disebabkan oleh penggunaan jumlah *container*/*server* yang berbeda, pada *Single Server* hanya menggunakan 2 *container* (1 *container* sebagai *web server* dan 1 *container* sebagai *database server*), sedangkan pada sistem *load balancing* baik dengan algoritma *Round Robin* maupun *Least Connection* menggunakan 5 buah *container* (1 *container* sebagai *load balancer*, 3 *container* sebagai *web server cluster*, dan 1 *container* sebagai *database server*). Dengan kata lain penggunaan 5 *container* pada 1 buah *server* fisik akan memakan *resource* yang cukup banyak dibanding hanya menggunakan 2 buah *container*.

5. Kesimpulan

5.1 Kesimpulan

Setelah dilakukan pengujian dan analisis dari skema yang telah dirancang, maka dapat ditarik kesimpulan.

1. Ditinjau dari segi performansi layanan *web*, kemampuan sistem *load balancing* dengan tiga buah *web server cluster* dalam melayani permintaan *client* (*request*) jauh lebih baik dibandingkan dengan *Single Server*, karena mengacu pada konsep *cluster computing* dimana penggunaan *cluster* dalam sebuah *server* dapat saling bekerja sama dan beban tidak terpusat pada satu *server* saja sehingga tidak memberatkan *server*.
2. Pada pengujian layanan *web*, sistem *load balancing* Docker menunjukkan hasil *throughput*, *response time*, dan *request per second* lebih unggul jika dibandingkan dengan *platform container* lain yang diuji yaitu LXC dan LXD. Hal ini ditunjukkan dengan persentase peningkatan nilai *throughput* sebesar 91,78% dari *Single Server*, penurunan nilai *response time* sebesar 74,9% dari *Single Server*, dan peningkatan nilai *request per second* sebesar 91,6% dari *Single Server*.
3. Sistem *load balancing* terbukti mampu meminimalisir jumlah *request loss* dari banyaknya *request* yang dikirim. Pada pengujian *request loss*, LXC lebih unggul dibanding Docker dan LXD, dengan nilai *request loss* minimum yaitu 0,004%.
4. Ditinjau dari segi *fairness*, algoritma *Round Robin* lebih *fair* dalam pendistribusian beban dibanding *Least Connection* yaitu dengan nilai *fairness index* mencapai 1.
5. Ditinjau dari segi penggunaan CPU (*processor*), LXC merupakan *platform container* yang ringan dibandingkan dengan *platform container* lain yang diuji. Hal ini ditunjukkan dengan penggunaan rata-rata CPU sebesar 84,84% pada sistem yang menggunakan *load balancing* dengan algoritma *Round Robin*, sedangkan pada *Single Server* LXC hanya menggunakan 75,36% dari total CPU yang digunakan pada pengujian yaitu 4.00 GHz.

5.2 Saran

Berdasarkan penjabaran pengujian, analisis, dan kesimpulan skema yang telah dirancang, masih terdapat beberapa kesalahan dan kekurangan. Untuk itu, penulis memberi saran untuk penelitian selanjutnya yang bertujuan untuk memperoleh sistem yang lebih baik, yaitu:

1. Menggunakan lebih dari 3 *web server cluster* sebagai perbandingan jumlah *cluster*.

2. Menggunakan lebih dari satu *load balancer* untuk meminimalisir apabila terjadi kegagalan (*down*) pada satu *load balancer*.
3. Menggunakan metode *failover* untuk meningkatkan ketersediaan *server*.
4. Dapat menggunakan parameter pengukuran yang lain untuk menguji kemampuan dan kehandalan suatu sistem *load balancing*.
5. Dapat menggunakan jenis *container* yang lain selain Docker, LXC, dan LXD.

Daftar Pustaka

[1] B. Kahanwal and T. P. Singh, "The Distributed Computing Paradigms: P2P, Grid, Cluster, Cloud, and Jungle," *International Journal of Latest Research in Science and Technology,* vol. 1, no. 2, pp. 183-187, 2012.

[2] M. Rosalia, R. Munadi dan R. Mayasari, "IMPLEMENTASI HIGH AVAILABILITY SERVER MENGGUNAKAN METODE LOAD BALANCING DAN FAILOVER PADA VIRTUAL WEB SERVER CLUSTER," *e-Proceeding of Engineering,* vol. 3, no. 3, 2016.

[3] M. A. Nugroho dan R. Katardi, "ANALISIS KINERJA PENERAPAN CONTAINER UNTUK LOAD BALANCING WEB SERVER PADA RASPBERRY PI," *JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika),* vol. 1, no. 2.

[4] K. Kaur and A. K. Rai, "A Comparative Analysis: Grid, Cluster and Cloud Computing," *International Journal of Advanced Research in Computer and Communication Engineering,* vol. 3, no. 3, 2014.

[5] V. Shinde, A. Shaikh and C. D. D'Souza, "Study of Cluster, Grid and Cloud Computing," *International Journal of Advanced Research in Computer and Communication Engineering,* vol. 4, no. 10, 2015.

[6] E. Mulyana, Buku Komunitas SDN-RG, GitBook.

[7] D. Kusnetzky, Virtualization: A Manager's Guide, O'Reilly Media, Inc., 2011.

[8] Docker, Docker for the Virtualization Admin, 2016.

[9] O. Hane, Build Your Own PaaS with Docker, Birmingham B3 2PB, UK: Packt Publishing Ltd., 2015.

[10] Linux Container, "What's LXC?," [Online]. Available: https://linuxcontainers.org/lxc/introduction/. [Accessed 11 November 2017].

[11] S. Gupta and D. Gera, "A Comparison of LXD, Docker and Virtual Machine," *International Journal of Scientific & Engineering Research,* vol. 7, no. 9, September 2016.

[12] HAProxy, "HAProxy The Reliable, High Performance TCP/HTTP Load Balancer," [Online]. Available: http://www.haproxy.org/. [Accessed 6 November 2017].

[13] T. P. Kusuma, R. Munadi dan D. D. Sanjoyo, "Implementasi dan Analisis Computer Clustering System dengan Menggunakan Virtualisasi Docker," 2017.

[14] B. A. Forouzan, Data Communications and Networking, Fourth Edition, New York: McGraw-Hill, 2007.

[15] Apache, "Apache HTTP Server Project," [Online]. Available: https://httpd.apache.org/. [Accessed 15 November 2017].

[16] Redis, "Introduction to Redis," [Online]. Available: https://redis.io/topics/introduction. [Accessed 20 July 2018].

[17] S. Mulay and S. Jain, "ENHANCED EQUALLY DISTRIBUTED LOAD BALANCING ALGORITHM FOR CLOUD COMPUTING," *International Journal of Research in Engineering and Technology,* vol. 2, no. 6, 2013.

[18] S. Bhatti, M. Bateman dan D. Miras, "A Comparative Performance Evaluation of DCCP," *International Symposium on Performance Evaluation of Computer and Telecommunication Systems,* 2008.

[19] S. Dabkiewicz, "Web Server Performance Analysis," 2010.