IMPLEMENTASI *LOAD BALANCER* DENGAN *LIGHTWEIGHT VIRTUALIZATION* MENGGUNAKAN DOCKER UNTUK LAYANAN *VIDEO ON DEMAND*

IMPLEMENTATION OF LOAD BALANCER IN LIGHTWEIGHT VIRTUALIZATION USING DOCKER FOR VIDEO ON DEMAND SERVICE

Novia Indrawati[1], Dr. Ir. Rendy Munadi, M.T.[2], Danu Dwi Sanjoyo, S.T., M.T.[3]
[1,2,3] Prodi S1 Teknik Telekomunikasi, Fakultas Teknik Elektro, Universitas Telkom
[1]noviaindrawati16@gmail.com, [2]rendy_munadi@yahoo.co.id, [3]danudwj@teklomuniversity.ac.id

Abstrak

Pada saat ini minat masyarakat Indonesia akan *streaming video* di *internet* berupa *Video On Demand* sangatlah meningkat, dari mulai *video* hiburan hingga edukasi. Dengan kebutuhan akan VOD yang *massive* ini, *provider* layanan VOD harus mempersiapkan *server* handal untuk mengatasi kenaikan trafik dan beban kerja pada *server*. Maka peran *load balancer* dibutuhkan dalam kondisi tersebut untuk mendistribusikan beban trafik ke beberapa *server cluster* secara seimbang sehingga *server* tidak mengalami kelebihan trafik (*overload*) atau bahkan *down*.

Penerapan teknologi virtualisasi berbasis *container* digunakan dalam pengimplementasian *load balancer* ini. *Container* adalah teknologi virtualisasi pada *level* sistem operasi yang memungkinkan setiap proses atau aplikasi dapat berjalan pada tiap *container* dengan berbagi kernel sistem operasi yang sama. Berbeda hal nya dengan *Virtual Machine* (VM) yang merupakan teknologi virtualisasi pada *level hardware*, sehingga memerlukan sistem operasi secara keseluruhan untuk membangun satu VM. Hal ini lah yang membuat *container* dikenal sebagai teknologi *lightweight virtualization*.

Tugas Akhir ini akan mengimplementasikan *load balancer* yang dijalankan di atas teknologi *container*. *Container* yang digunakan adalah docker. Penelitian ini bertujuan untuk mengetahui kinerja dari *load balancer* pada layanan VOD. Dari hasil penelitian yang dilakukan diketahui bahwa kinerja *server* dengan menggunakan *load balancing* lebih baik dibandingkan dengan *single server*, karena beban kerja dan beban trafik tidak lagi dilayani oleh satu *server* lagi melainkan beban dibagi ke tiga buah *server*. Pada penelitian ini juga diketahui, algoritma yang paling baik digunakan untuk *load balancing* adalah *least connectiion*, karena dapat terdapat penurunan CPU *Utilization* sebesar 5.17 %.

Kata Kunci : *Load Balancer, Video On Demand, Container, Lightweight Virtualization, Kubernetes*

Abstract

Nowadays, streaming video such as Video On Demand on internet become one of the highest consumptions in cyber world. Education and even entertainment is provided. Because of massive demand on video streaming, provider must prepare relliable server to resolve overcome the increase in traffic and workload on the server. Then in these conditions the role of load balacer is needed to distribute traffic load to several cluster servers in a balanced manner so that the server does not experience excess traffic (overload) or even down.

The application of container-based virtualization technology is used in implementing this load balancer. Container is a virtualization technology at the operating system level that allows each process or application to run on each container by sharing the same operating system kernel. It's different between container and Virtual Machine. Virtual Machine (VM) which is a virtualization technology at the hardware level, so it requires the whole operating system to build a VM. This is what makes containers known as lightweight virtualization technology.

This Final Assignment will implement load balancers that run on container technology. The container used is the docker. This study aims to determine the performance of load balancers on VOD services. From the results of the research conducted, it is known that server performance using load balancing is better than a single server, because the workload and traffic load are no longer served by one server but rather the burden is divided into three servers. In this study it is also known, the best algorithm for load balancing is least connectivity because there can be a decrease in CPU Utilization of 5.17%.

Keywords : Load Balancer, Video On Demand, Container, Lightweight Virtualization, Kubernetes

1. Pendahuluan

Layanan *Video On Demand* (VOD) telah menjadi kebutuhan baru di Indonesia dan memperoleh popularitas yang sangat tinggi akhir-akhir ini, *video* tersebut meliputi film, acara televisi, video edukasi, *video-video* yang

dibuat dan dibagikan oleh orang awam (vlog) dan ditonton oleh jutaan orang tiap hari. Hal tersebut dapat mengakibatkan trafik pada sebuah *website* meningkat dan beban kerja pada *server* bertambah. Ketika sebuah sistem mengalami kenaikan jumlah *request* sampai ribuan per hari, hal ini dapat menyebabkan kinerja sistem menjadi sangat lambat dan menimbulkan banyak permasalahan, salah satunya adalah keluhan dari sisi pengguna [1].

Penggunaan *server* dengan sistem terdistribusi membutuhkan suatu metode agar dapat membagi beban dengan merata disetiap *server,* maka diterapkan lah metode *load balancing* dengan menggunakan Nginx. *Load balancer* akan diterapkan dalam *container*. *Container* mulai populer dengan docker dan kubernetes karena menjadikan seluruh proses menjadi efisien. Docker digunakan untuk membuat Dockerimage dari aplikasi dengan menggunakan dockerfile. Dockerfile memiliki semua instruksi bagaimana membangun dan mendistribusikan sebuah aplikasi, hasil dari dockerfile tersebut adalah berupa *image* yang nantinya akan digunakan oleh kubernetes untuk *deployment* aplikasi secara langsung.

Berbagai penelitian telah dilakukan untuk menerapkan metode *load balancing*. Pada penelitian yang ditulis pada jurnal [2] membahas mengenai implementasi *load balancer* pada *web server* dengan dua algoritma yang digunakan, yaitu *least connection* dan *round robin* dan membandingkan keduanya dengan parameter *response time, network throughput* dan *request loss*. Lalu pada jurnal [3] ini membahas mengenai virtualisasi ringan yang di dalamnya terdapat aplikasi *web server* dan *load balancer* namun docker yang digunakan adalah docker yang telah ter-*install* pada sistem operasi Hypriot untuk raspberry pi 2. Pada penelitian [4] membahas mengenai perbandingan docker, lxc dan lxd pada *load balancer* dengan *service* yang digunakan adalah *web server* dan menggunakan *orchestration tool* docker-compose. Sedangkan pada penelitian [5] membahas pula mengenai perbandingan antara docker, lxc dan lxd, namun *service* yang dijadikan penelitian adalah FTP *server*, *web server* dan *email server* tanpa menggunakan *orchestration tool*. Adapun pada penelitian [6] membahas mengenai pengujian *load balancer* pada sistem docker dan *orchestration tool* docker swarm dengan *service* yang digunakan adalah *video on demand*. Dan yang terakhir, pada penelitian [7] membahas mengenai implementasi aplikasi huginn (aplikasi *monitoring*) pada docker sebagai *container-based virtualization* dan kubernetes sebagai *orchestration tool.*

Tugas Akhir ini adalah mengimplementasikan *load balancer* menggunakan Nginx dengan algoritma *round robin* dan *least connection* pada layanan *Video on Demand server* yang berbasiskan virtualisasi *container* dengan menggunakan docker dan kubernetes. Parameter yang akan diuji untuk melihat performansi dari *load balancer* tersebut meliputi *throughput, response time, request per second*, *request loss* dan CPU *Utilization*

2. Dasar Teori

2.1. Virtualisasi

Virtualisasi adalah sebuah teknologi yang memungkinkan untuk melakukan abstraksi pada aplikasi dan komponen-komponen lain yang *support* menjadi sesuatu yang bersifat *virtual* [8]. Hal ini dapat menciptakan tampilan buatan bahwa akan terdapat banyak komputer namun yang terlihat hanyalah satu buah komputer fisik. Hingga saat ini sumber daya yang telah dapat divirtualisasikan antara lain adalah perangkat keras komputer (*hardware*), media penyimpanan data (*storage*), *Operating System* (OS), layanan jaringan (*networking*) dan daftar ini akan terus bertambah [8].

Terdapat beberapa keuntungan menerapkan teknologi ini yaitu meningkatkan efisiensi penggunaan *server* dengan memaksimalkan kinerja perangkat tersebut tanpa adanya *resource* yang tidak terpakai, sehingga ketergantungan akan *physical hardware* dari *server* sistem operasi secara perlahan akan berkurang lalu kebutuhan akan perawatan *hardware* dan konsumsi *power* akan berkurang juga [9].

2.2. Virtualisasi Container

Virtualisasi *container* ini memiliki tingkat abstraksi yang berbeda dalam hal virtualisasi dan isolasi jika dibandingkan dengan virtualisasi berbasis *hypervisors*. Virtualisasi *container* ini dapat disebut sebagai *lightweight virtualization* dibandinkan dengan *hypervisors. Hypervisor* mengabstraksi perangkat keras yang menghasilkan *overhead* dalam hal virtualisasi perangkat keras dan *virtual device hardware*. Sistem operasi digunakan secara menyulurh (misalnya linux) yang berjalan di atas laptop atau pc pada setiap *virtual machine* (VM). Sebaliknya dengan *container* yang mengimplementasikan isolasi nya pada *level* sistem operasi, sehingga menghindari *overhead* tersebut. *Container* ini berjalan di atas kernel sistem operasi berbagi yang sama dari mesin *host*, dan satu atau beberapa proses dapat dijankan dalam setiap *container* [10].

2.3. Docker

Docker adalah sebuah *project open source* yang ditujukan untuk developer atau *sysadmin* untuk membangun, mengemas dan menjalankan aplikasi dimana pun di dalam sebuah *container*. Docker berfungsi sebagai virtualisasi sebuah sistem operasi atau sebuah *server* atau sebuah *web server* atau bahkan sebuah database server, dimana dengan menggunakan virtualisasi ini, diharapkan *developer* dapat mengembangkan aplikasi sesuai dengan spesifikasi *server* atau dengan kata lain, jika kita mengembangkan sebuah aplikasi lalu kita jalankan pada komputer kita sendiri maka secara otomatis aplikasi akan berjalan dengan baik [11].

2.4. Kubernetes

Kubernetes adalah platform *open-source* yang dapat mengatur penempatan (*scheduling*) dan menjalankan aplikasi *container* di dalam dan di seluruh *cluster* komputer. Kubernetes merupakan *orchestration tool container* terkemuka di bidang *containerization*. Dikembangkan dari berbagai pengalaman mengenai pengaturan *container* yang diakumulasikan oleh Google dan ide-ide terbaik dari komunitas. kubernetes menggunakan docker *images* sebagai dasar untuk membangun aplikasi diatas *container*. Dengan kubernetes, *container* dapat dengan mudah ditingkatkan jumlahnya (*scaled-up*), dihancurkan dan dibuat kembali. Dibandingkan dengan mesin virtual normal, kubernetes ini dapat membangun aplikasi lebih cepat, lebih efisien dan handal [12].

2.5. *Video Streaming*

Teknologi *streaming* merupakan salah satu bentuk teknologi yang memperkenankan *file* digunakan secara langsung tanpa menunggu selesainya unggahan (*download*) dan berlangsung secara kontinu tanpa interupsi. Aplikasi layanan *streaming* dibagi menjadi dua, yaitu *on-demand* dan *live video streaming*. Layanan *streaming on-demand* seperti halnya musik atau *video* yang dijalankan langsung dari media penyimpanan *server* yang diakses melalui *client*. Sedangkan *live streaming* seperti halnya *video* atau *audio* yang disiarkan secara *broadcast* dan langsung pada saat itu juga atau *real time*.

2.6. *Load Balancing*

Load balancing adalah teknik yang digunakan untuk mendistribusikan trafik yang masuk ke *server* yang tersedia sehingga permintaan dapat ditangani dengan cepat. Sistem jaringan komputasi yang kompleks melakukan perutean jutaan paket data setiap detiknya. Trafik data yang begitu besar harus didistribusikan secara efisien di antara *server* yang tersedia sehingga *server* dapat menanganinya dengan lebih baik dan cepat.

High availability merupakan salah satu faktor pendorong *load balancing*. Keuntungan utama *load balancing* sebagai berikut [13]:

1. Membantu dalam mengendalikan dan melacak trafik.
2. Meningkatkan pemanfaatan sumber daya.
3. Menyeimbangkan beban jaringan.
4. Meningkatkan ketersediaan sumber daya.
5. Mengurangi *overprovision* pada infrastruktur.

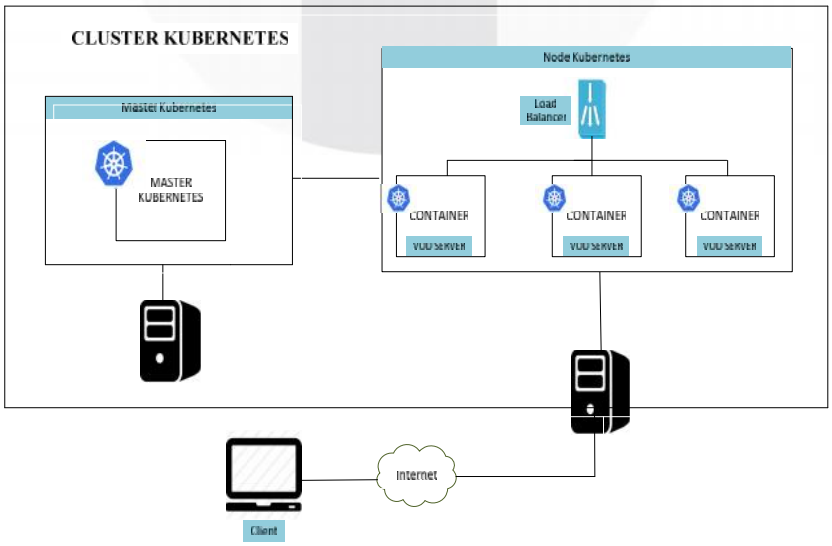
2.7. Nginx

Nginx adalah *software open-source* untuk *web serving*, *reverse proxying*, *caching*, *load balancing*, *media streaming* dan masih banyak lagi. Salah satu yang membuat nginx menjadi sangat cepat adalah jenis arsitektur nginx itu sendiri. Jika di bandingkan dengan apache yang *process based*, nginx menjadi jauh lebih unggul karena *event-based* nya. Sehingga mampu memanfaatkan seminimal mungkin *thread* untuk memproses *request* dari *user*, sehingga akhirnya memori yang terpakai oleh Nginx menjadi minimal. Karena memori yang dipakai sangat kecil, maka hasilnya *server* menjadi ringan dan jauh lebih responsif (memiliki respon sangat cepat [14].

3. Perancangan Sistem

3.1 Desain Model Sistem

Secara keseluruhan, Desain model pada sistem yang digunakan untuk Tugas Akhir ini dapat dilihat pada Gambar 3.1.



Gambar 3.1 Diagram Model Sistem

Seperti terlihat pada Gambar 3.1, *client* akan mengkases langsung *service* yang telah disediakan oleh *server* melalui jaringan *internet*, karena *server* yang digunakan pada tugas akhir ini adalah berupa *Virtual Private Server* (VPS) dari *Digital Ocean* maka IP *address* yang disediakan telah berupa IP *Public*.

Sistem pada *server* ini menggunakan Nginx untuk dua buah fungsi yaitu sebagai *tools* yang dapat melakukan proses *load balancing* dan tempat untuk menampilkan VOD *server*, selanjutnya docker sebagai teknologi untuk menjalankan *container* pada *server* dan kubernetes sebagai *orchestration tools* yang bertugas untuk mengatur seluruh *container* yang terdapat pada *server*. Untuk menampilkan VOD *server* pada sistem, dibutuhkan *dockerfile* yang berfungsi untuk mencatat semua perintah-perintah berupa script dalam membangun VOD *server*, *output* dari *dockerfile* adalah *image* sehingga dapat di *pull* oleh kubernetes. Sedangkan pada sisi *client* akan dibangkitkan sejumlah *request* terhadap *server* menggunakan *software* untuk melakukan *load testing* yaitu menggunakan siege. *Load balancer* dan VOD *server* akan di implementasikan pada *container*. Lalu akan dilakukan perbandingan sejauh mana peningkatan kinerja *server* pada masing-masing algoritma, yaitu *round robin* dan *least connection* pada Nginx *load balancer*.

3.2 Parameter Pengujian

Untuk melakukan pengujian pada sistem *load balancing* ini, parameter yang diukur meliputi *throughput, response time, request per second, request loss,* CPU *utilization* dan *fairness index*. Berikut penjelasan untuk masing-masing parameter:

1. *Throughput*
   Throughput adalah *bandwith* aktual yang terukur pada suatu ukuran waktu tertentu dalam suatu jaringan. Pengujian *throughput* merupakan parameter variabel dari QOS (*Quality Of Sevice*) dengan tujuan untuk melihat performansi jaringan dari segi kecepatan pengiriman paket yang dapat dikirim dengan memanfaatkan *bandwith* yang tersedia. Pengujian *throughput* dilakukan dengan menggunakan aplikasi siege dengan cara membebani *server* dengan mengirimkan http *request*. Besar nilai *throughput* pada penelitian ini dapat dilihat dari hasil pengujian pada aplikasi siege dengan satuan MB/s.
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 *pod* dapat menerima *request* dari *client*. Hasil dari pengujian ini dapat dilihat menggunakan aplikasi siege dengan satuan *seconds* (s).
3. *Request per Second*
   Pengujian *request per second* dilakukan dengan cara melakukan sejumlah layanan per satuan detik dari *client* yang akan dikirimkan ke vod *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 di atas *container* (pods) pengujian parameter ini dilakukan dengan menggunakan aplikasi siege.
4. *Request loss*
   Pada saat melakukan pengiriman paket, paket yang dikirim memiliki kemungkinan untuk dibuang karena *resources* menampung lebih dari kapasitas maksimum. Proses pembangunan 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. *CPU Utilization*
   CPU Utilization adalah jumlah sumber daya yang dibutuhkan untuk melakukan suatu proses komputerisasi, dalam hal ini *load balancing*. Pengujian ini bertujuan untuk melakukan peninjauan terhadap penggunaan *resources* oleh pod dari segi penggunaan CPU (*processor*) untuk melihat apakah terjadi perbedaan penggunaan CPU pada masing-masing pod saat menjalankan *load balancing*.

3.3 Skenario Pengujian

Berdasarkan parameter pengujian yang akan diuji pada Tugas Akhir ini, pengujian dilakukan dengan parameter uji meliputi *throughput, response time, request per second, request loss* dan CPU *utilization.* Pengujian parameter tersebut diuji dengan menggunakan tiga skenario pengujian sebagai berikut :

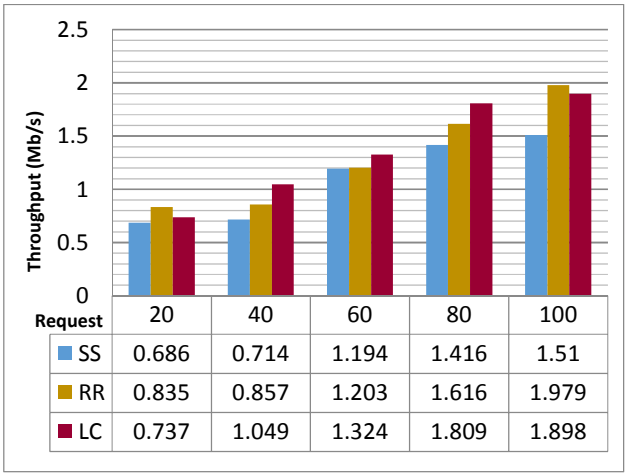
1. Pengujian VOD *server* tanpa menggunakan *load balancing*
2. Pengujian VOD *server* menggunakan sistem *load balancing* dengan algoritma penjadwalan round robin.
3. Pengujian VOD *server* menggunakan sistem *load balancing* dengan algoritma penjadwalan *least connection.*

4. Hasil Pengukuran dan Analisis

Pada bab ini dibahas mengenai hasil pengujian dan analisis dari implementasi sistem *load balancing* yang telah dibangun dengan skenario yang telah dijelaskan pada sub-bab 3.5 mengenai skenario pengujian, yaitu pengujian pada VOD *server* saat menggunakan *load balancer* dengan algoritma round robin, pengujian VOD

server saat menggunakan *load balancer* dengan menggunakan algoritma least connection dan dilakukan pengujian VOD *server* tanpa menggunakan *load balancer*. Dalam melakukan pengujian terhadap ketiga skenario tersebut akan dilakukan tes berdasarkan algoritma penjadwalan yaitu pada saat *server* menggunakan *load balancer* dengan algoritma round robin (RR), saat *server* menggunakan algoritma least connection (LC) dan ketika server tanpa *load balancer (single server)*. Setiap parameter yang diuji dilakukan pengujian dan dihitung rata-ratanya dengan pemberian beban sebesar 20, 40, 60, 80 dan 100 *request*. berikut akan dijabarkan mengenai hasil pengujian beserta analisis dari masing-masing pengujian

4.1 Throughput



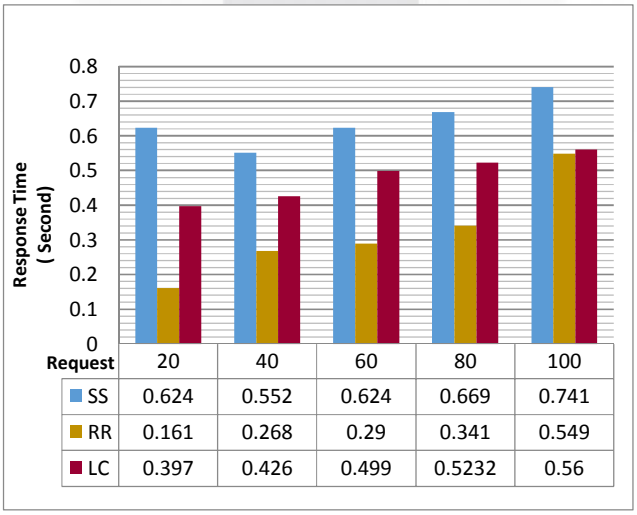
Request	20	40	60	80	100
SS	0.686	0.714	1.194	1.416	1.51
RR	0.835	0.857	1.203	1.616	1.979
LC	0.737	1.049	1.324	1.809	1.898

Gambar 4.1 *Throughput* pada RR, LC dan SS

Dilihat pada Gambar 4.4 dapat disimpulkan bahwa semakin banyak koneksi atau permintaan dari *user* maka semakin besar pula *throughput* yang dihasilkan. Ini menandakan bahwa semakin banyak koneksi *user* semakin baik pula kemampuan *server* dalam melayani permintaan *user*.

Hasil pengukuran yang didapat menyatakan bahwa nilai *throughput* lebih besar saat menggunakan sistem *load balancing* dibandingkan dengan nilai *throughput* yang tidak menggunakan sistem *load balancing*. Pada saat *load balancing* nilai *throughput* mengalami kenaikan sebesar 0.009 – 0.47 Mb/s. Hal tersebut terjadi karena terdapat perbedaan jumlah *server* yang melayani *user*, dimana pada saat penggunaan sistem *load balancing* terdapat tiga *server* yang melayani permintaan *users* dalam waktu yang bersamaan sehingga dapat mengurangi *bottleneck* disisi *server*. Dalam perbandingan *server* yang meggunakan *load balancing*, dapat dilihat bahwa sistem dengan menggunakan algoritma least connection cenderung memiliki nilai yang lebih tinggi dimana proses pembagian beban itu berdasarkan jumlah user yang terdapat pada server. Dengan demikian dapat mengurangi antrian dan dapat melayani *request* lebih banyak sehingga *throughput* yang didapat lebih besar.

4.2 Response Time



Request	20	40	60	80	100
SS	0.624	0.552	0.624	0.669	0.741
RR	0.161	0.268	0.29	0.341	0.549
LC	0.397	0.426	0.499	0.5232	0.56

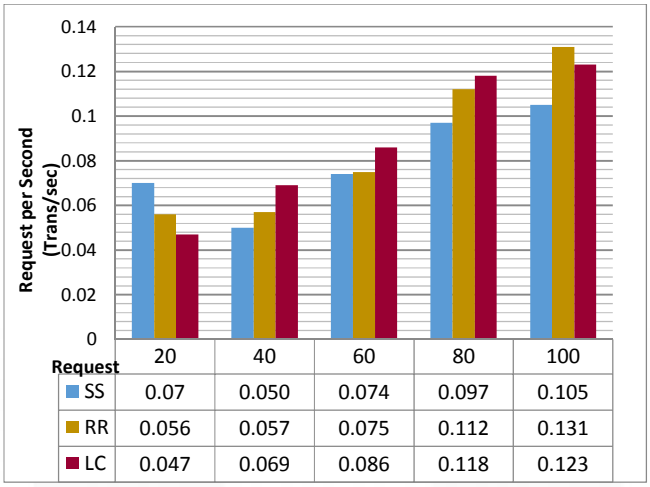
Gambar 4.1 *Response Time* pada RR, LC dan SS

Gambar 4.5 merupakan grafik dari hasil pengujian *response time*, dari ketiga server 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 *server* saja oleh karena itu waktu dang 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 meningkat seiring dengan naiknya nilai *throughput* yang didapat, semakin besar nilai *throughput* yang didapat pada saat pengujian maka waktu yang dibutuhkan oleh sebuah VOD *server* untuk mengirimkan http *response* ke *client* akan semakin lambat dikarenakan request yang datang semakin banyak.

Dari grafik yang ditunjukkan oleh Gambar 4.5 dapat dilihat bahwa nilai *response time* yang paling kecil adalah ketika *server* menggunakaan sistem *load balancing* dengan algoritma round robin, hal ini dikarenakan kecepatan transmisi dan *throughput* yang dihasilkan memiliki hasil yang lebih baik dibandingkan algoritma least connection.

4.3 Request per Second



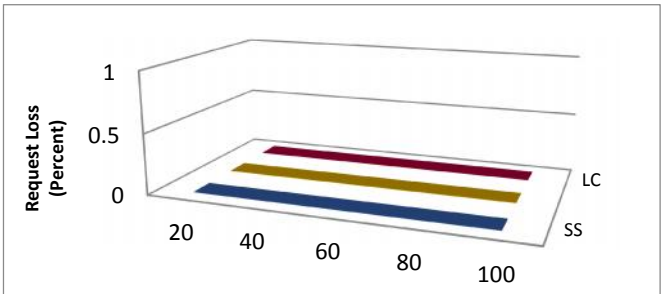
Request	20	40	60	80	100
SS	0.07	0.050	0.074	0.097	0.105
RR	0.056	0.057	0.075	0.112	0.131
LC	0.047	0.069	0.086	0.118	0.123

Gambar 4.3 *Request per Second* pada RR, LC dan SS

Berdasarkan grafik pada Gambar 4.6 menyatakan bahwa semakin banyak *user* yang mengakses *server* maka semakin besar nilai *request per second* yang dihasilkan. Ini menandakan bahwa performansi dari *server* tergolong baik karena semakin banyak *user* yang dilayani ketika permintaan dari *user* meningkat.

Dari grafik yang ditunjukkan oleh Gambar 4.6 dapat dilihat bahwa nilai *request per second* yang lebih besar adalah ketika *server* menggunakaan sistem *load balancing* baik ketika menggunakan algoritma round robin maupun least connection, hal ini dikarenakan pada sistem yang menggunakan *load balancing* permintaan atau *request* yang datang dilayani oleh tiga buah *server cluster* yang saling bekerja sama sehingga jumlah *request* yang dapat dilayani meningkat dibandingkan pada *single server* yang hanya menggunaan satu buah *server*. Nilai tersebut sangat dipengaruhi oleh nilai *throughput* yang di dapat, semakin besar nilai *throughput* yang didapatkan maka semakin besar pula jumlah *request* per detik yang dapat dilayani oleh VOD *server* sehingga dapat dikatakan bahwa nilai *throughput* yang didapat yaitu berbanding lurus dengan nilai *request per second* yang didapat.

4.4 Request Loss

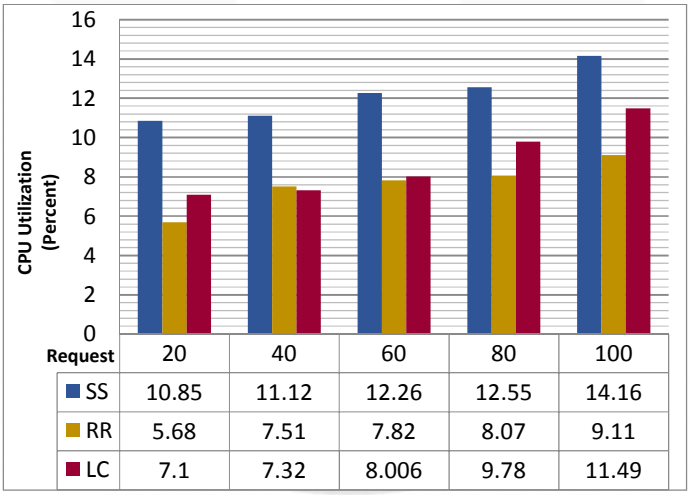


Request	20	40	60	80	100
SS	0	0	0	0	0
RR	0	0	0	0	0
LC	0	0	0	0	0

Gambar 4.4 *Request Loss* pada RR, LC dan SS

Pada pengujian ini semakin kecil nilai *request loss* maka sistem tersebut dapat dikatakan memiliki kehandalan yang baik. Dari grafik yang ditunjukkan oleh Gambar 4.7 dapat dilihat bahwa nilai *request loss* yang dihasikan untuk single server, round robin dan least connection adalah 0. Hal ini dikarenakan kemampuan *server* yang masih bisa menampung beban sebesar 100 *request*. Hal ini menunjukkan bahwa *server* memiliki kemampuan yang baik bahkan dengan hanya menggunakan satu *server* saja dalam menangani jumlah *request* tersebut. Pada pengujian ini di tes pula pada sistem yang menggunakan *load balancing* mulai terjadi *request loss* ketika *server* menerima *request* sebesar 1000. Adanya *request loss* dapat dipengaruhi oleh trafik yang masuk terlalu banyak diluar kemampuan *server* (*congest*).

4.5 CPU Utilization



Request	20	40	60	80	100
SS	10.85	11.12	12.26	12.55	14.16
RR	5.68	7.51	7.82	8.07	9.11
LC	7.1	7.32	8.006	9.78	11.49

Gambar 4.4 CPU Utilization pada RR, LC dan SS

Dari Gambar 4.8 terlihat bahwa beban kerja pada sistem load balancing lebih kecil dibandingkan pada single server, hal ini dikarenakan pembebanan pada Single Server sepenuhnya dilakukan oleh satu VOD server saja sedangkan pada sistema yang meggunakan load balancing, pembebanan request yang datang dapat dibagi ke tiga buah VOD server yang tersedia. Pada load balancing round robin pembagian beban dibagi secara merata kedalam tiga buah server, sedangkan pada load balancing least connection pembagian beban dibagi berdasarkan koneksi aktif yang paling sedikit. Terjadi penurunan hingga 5.17 % pada pengujian round robin dan penurunan 4.53 % pada pengujian least connection. Terlihat dari gambar diatas bahwa sistem load balancing dengan algoritma round robin memiliki kinerja yang lebih baik dalam mendistribusikan beban. Perbedaan nilai CPU pada algoritma least connection dan round robin sebesar 0.64 % – 2.18 %

5 Kesimpulan dan Saran

5.1 Kesimpulan

Berdasarkan hasil implementasi, pengukuran dan pengujian terhadap sistem *load balancing* pada docker dan kubernetes ini, maka dapat diambil kesimpulan sebagai berikut :

1. Kemampuan sistem *load balancing* dalam melayani jumlah *request* lebih baik dibandingkan *single server*, karena beban kerja didistribusikan ke tiga buah *virtual server*. Terbukti dengan adanya penurunan CPU *Utilization* hingga 5.17%
2. Sistem *load balancing* dapat membagi beban trafik secara merata ke tiga buah *virtual server* sehingga dapat mengurangi *bottleneck* disisi *server*.
3. Nilai *throughput* dipengaruhi oleh banyaknya jumlah permintaan, semakin besar jumlah permintaan maka semakin besar nilai *throughput* yang didapat. Terlihat nilai *throughput* tertinggi berada saat *server* menangani 100 *request*, didapatkan nilai *throughput* tertinggi adalah sebesar 1.979 Mbit/s.
4. Nilai *response time* yang paling kecil adalah ketika *server* menggunakaan sistem *load balancing* dengan algoritma round robin, hal ini dikarenakan kecepatan transmisi dan *throughput* yang dihasilkan memiliki hasil yang lebih baik dibandingkan algoritma least connection.
5. Kemampuan server dalam melayani permintaan dari request untuk single server sangatlah baik terbukti dengan tidak adanya request loss. Hal ini menujukkan bahwa untuk 100 request server masih bisa menampung permintaan masuk dengan beban tersebut.

5.2 Saran

Adapun beberapa saran penelitian yang bias dilakukan selanjutnya adalah sebagai berikut :

1. Perlu menggunakan parameter lain untuk menguji sampai mana kehandalan *load balancing* dalam menangani banyak permintaan *user*.
2. Diharapkan pengujian *load balancing* ini dapat menggunakan algoritma lain dan menggunakan lebih dari 3 *server cluster* sebagai perbandingan jumlah *cluster.*
3. Dapat menggunakan *orchestration tool* lain dalam manajemen *container* seperti rancher atau openshift.
4. Dapat menggunakan pembebanan *request* yang lebih besar sehingga dapat diperoleh jumlah maksimum *request* yang masih dilayani oleh *server*.

DAFTAR REFERENSI

[1] Alimuddin, A. Ahmad. Peningkatan Kinerja Siakad Menggunakan Metode *Load Balancing* dan *Fault Tolerance* di Jaringan Kampus Universitas Halu Oleo, IJSS, vol.10, No.1, Januari 2016, pp. 11~12.

[2] M. Rizky, Hafiduddin, S. Aulia. Analisis Performansi Load Balancing dengan Algoritma Round Robin dan Least Connection pada Sebuah Web Server, Jurnal, Telkom University, Bandung, 2015.

[3] M. Agung, R. Katardi. Analisis Kinerja Penerapan Kontainer untuk Load Balancing Web Server pada Raspberry P1, Jurnal Ilmiah, STMIK Akakom, Yogyakarta, 2015. [4] Lo, T., and S. W. Lee, "Antenna Handbook", New York: Van Nostrand Reinhold, 1988.

[4] Pratama, Rivaldy Arif. Implementasi Web Server Cluster Menggunakan Metode Load Balancing pada Container Docker, LXC dan LXD, Jurnal, Telkom University, Bandung, 2018.

[5] Putri, Adinda Riztia. Analisis Performansi FTP Server, Web Server dan Mail Server pada Container Docker, LXC dan LXD, Jurnal, Telkom University, Bandung, 2018.

[6] Perdana, Tanjung. Implementasi dan Analisis Computer Clustering System dengan Menggunakan Virtualisasi Docker, Jurnal, Telkom University, Bandung, 2017.

[7] Moilanen, Miika . Deploying an Application using Docker and Kubernetes, Thesis, Oulu University of Applied Sciences, Isandia, 2018.

[8] Menasce, D. A. (n.d.). Virtualization : Concepts, Applications, and Perfomance Modeling. Virtualization : Concepts, Applications, and Perfomance Modeling.

[9] R. Danielle, R Nelson. (2009). Virtualization a Beginner's Guide: McGraw Hill. Inggris.

[10] Morabito Roberto, Kjallman Jimmy, Komu Miika. (2015). *Hypervisors vs. Lightweight Virtualization: a Performance Comparison*, IEEE International Conference on Cloud Engineering

[11] (2018, August 01). *Retrieved from* https://docs.docker.com/engine/docker-overview/#docker-architecture.

[12] Heidari Parisa, Lemieux Yves, Shami Abdallah. (2016). QoS *Assurance with Light Virtualization* – A survey. IEEE 8th International Conference on Cloud Computing Technology and Science.

[13] Diwi, Anggelina I, M. R Rumani dan Wahidah, Ida. Analisis Kualitas Layanan *Video Live streaming* Pada Jaringan Lokal Universitas Telkom. Jurnal Pusat Penelitian dan Pengembangan Sumber Daya dan Perangkat Pos dan Informatika. Volume 12. 2014. ISSN: 16930991.

[14] Pratama, M. R. (2015). *Load Balancing Performance Analysis With Round Robin And Least Connection Algorithm Based On Web Server*, 10 March 2016.