IMPLEMENTASI HIGH AVAILABILITY SERVER MENGGUNAKAN METODE LOAD BALANCING DAN FAILOVER PADA VIRTUAL WEB SERVER CLUSTER

IMPLEMENTATION OF HIGH AVAILABILITY SERVER USING LOAD BALANCING AND FAILOVER METHOD ON VIRTUAL WEB SERVER CLUSTER

Maya Rosalia[1], Dr. Rendy Munadi, Ir., M. T.[2], Ratna Mayasari, S. T., M. T.[3]

[1,2,3]Prodi S1 Teknik Telekomunikasi, Fakultas Teknik Elektro, Universitas Telkom

[1]**rosalia@students.telkomuniversity.ac.id**, [2]**rendymunadi@telkomuniveristy.ac.id**, [3]**ratnamayasari@telkomuniversity.ac.id**

Abstrak

Saat ini *Web* menjadi media berorientasi bisnis dan antarmuka yang lebih disukai untuk sistem informasi terbaru . Semakin banyak yang mengakses melalui suatu situs web akan menyebabkan beban kerja suatu penyedia layanan yaitu web *server* menjadi lebih berat dan kurang optimal. Jika kondisi tersebut terus berlanjut, besar kemungkinan akan terjadi *overload* dan *server* menjadi *down* sehingga *request* tidak dapat lagi dilayani.

Server Clustering merupakan salah satu solusi yang bisa diterapkan untuk mengatasi permasalahan tersebut, yaitu suatu teknologi yang menggabungkan beberapa *server* yang bekerja bersama-sama yang seolah-olah merupakan satu sistem tunggal. Terdapat beberapa metode pada sistem *clustering*, yaitu *load balancing* dan *failover*. Dengan *load balancing*, sistem akan dapat melayani beban pengaksesan yang besar dan meminimalisir kegagalan dalam melayani *request* dari *user* karena *load balancing* bekerja dengan mendistribusikan secara merata beban trafik ke beberapa *server* lain yang terkluster. *Failover* berfungsi untuk meningkatkan ketersediaan yang tinggi. Jika terjadi kegagalan sistem pada *server* utama, *server* cadangan akan langsung menggantikan *server* utama untuk tetap memberikan layanan.

Pada tugas akhir ini telah diimplementasikan *load balancing* dan *failover* pada *virtual server cluster*. Dari hasil penelitian yang dilakukan diketahui bahwa kinerja *server* dengan menggunakan *load balancing* jauh lebih baik dibandingkan *single server*, dengan jumlah *request* per-detik maksimal yaitu sebesar 2352.937 *request* dan *throughput* sebesar 3.53 MB/s pada Haproxy penjadwalan *least connection*. Adanya pembagian beban ke tiga buah *server* memberikan penurunan terhadap nilai CPU *utilization* sebesar 21%. Untuk ketersediaan *server* pada skenario *failover* didapatkan nilai *downtime* rata-rata sebesar 1992.8 ms. Pada penelitian ini juga diketahui bahwa *load balancing* dengan menggunakan Haproxy memiliki performansi yang lebih baik dibandingkan dengan Nginx.

Kata Kunci*: Cluster, Load Balancing, Failover, Virtual, Haproxy, Nginx*

Abstract

Currently, the site has become a business-oriented media and the preferred interface for system updates. he more that access through a web site, will lead to the workload of a service provider that is a web server becomes heavier and less than optimal. If the condition persists, it is likely there will be overloaded and the server will be down so that the request can no longer be served.

Server Clustering is one solution that can be implemented to resolve the problem, which is a technology that combines multiple servers working together as if a single system. There are several methods on the clustering system, the load balancing and failover. With load balancing, the system will be able to serve a large load access and minimize failures in the service request from the user, since load balancing works evenly distribute the traffic load to some other server that clustering. Failover serves to increase the high availability. If a system failure occurs on the main server, a backup server will directly replace the main server to continue to provide services.

In this final project has been implemented load balancing and failover on a virtual server cluster. From this research, it is known that the performance of the server by using load balancing is much better than single server, with the number of requests per second maximum in the amount of 2352.937 requests and throughput at 3.53 MB/s with the least connection scheduling HAProxy. The distribution of the load to three servers provide a decrease in the value of the CPU utilization of 21%. For availability server on failover scenario, obtained the value of downtime by an average of 1992.8 ms. In this research also note that load balancing by using HAProxy has better performance compared with Nginx.

Keyword: Cluster, Load Balancing, Failover, Virtual, Haproxy, Nginx

1. Pendahuluan

Perkembangan teknologi memberikan pengaruh besar bagi kehidupan salah satunya dalam penyampaian suatu informasi. Kebutuhan akan informasi dan komunikasi menjadi salah satu kebutuhan pokok dalam kehidupan sehari-hari. Kebutuhan akan koneksi internet semakin meningkat seiring berlimpahnya perangkat yang bisa dipakai untuk menjelajahi dunia maya. Sejak memasuki tahun 2000-an, *Web* menjadi media berorientasi bisnis dan antarmuka yang lebih disukai untuk sistem informasi terbaru[5]. Semakin banyak yang mengakses melalui suatu situs web membuat beban kerja yang lebih pada suatu penyedia layanan yang disebut *web server* dan menjadi kurang optimal. Suatu *single server* bisa mengalami kegagalan yang disebabkan oleh meningkatnya jumlah *request* yang mencapai ribuan bahkan jutaan pada waktu yang bersamaan atau disebut dengan *overload*. Hal ini akan merugikan pihak yang mempercayakan situsnya pada suatu *web server*, karena situs tersebut tidak dapat diakses untuk waktu tertentu.

Server Clustering merupakan salah satu solusi yang bisa diterapkan untuk mengatasi permasalahan tersebut, yaitu suatu teknologi yang menggabungkan beberapa *server* yang bekerja bersama-sama yang seolah-olah merupakan satu sistem tunggal[1]. Dengan didukung teknik *load balancing* yang diharapkan dapat menangani beban yang sangat berat dengan mendistribusikannya kepada server lain yang ter*cluster*, dan juga *failover* untuk mengantisipasi kegagalan atau kerusakan pada komputer *server* sehingga ketika suatu *server* utama mati, maka *server* lain yang berperan sebagai cadangan akan mengambil alih untuk terus memberikan layanan.

Pada tugas akhir ini akan dibuat implementasi *clustering* untuk *virtual web server* dan *high availability server* menggunakan metode *load balancing* dan *failover* untuk meningkatkan kehandalan dan ketersediaan layanan. Kemudian akan dibandingkan performansi dua *software load balancing* dan *failover* dengan parameter yang akan diuji meliputi *throughput, request* per-detik, *request loss, cpu utilization* dan *downtime.*

2. Dasar Teori

2.1 Cluster Computing

Komputer kluster adalah sekumpulan komputer (umumnya server jaringan) independen yang bekerja secara bersama sebagai sumber daya komputasi tunggal yang terintegrasi [1] dan terlihat oleh klien seolah-olah komputer-komputer tersebut adalah satu buah unit komputer. Proses menghubungkan beberapa komputer agar dapat bekerja seperti itu dinamakan dengan *Clustering.* Komponen *cluster* biasanya saling terhubung melalui sebuah interkoneksi yang sangat cepat, atau bisa juga melalui jaringan lokal (LAN) [2].

Dalam menjalankan fungsinya *Cluster Computing* diklasifikasikan menjadi tiga, yaitu:

1. *High-availability Clusters*, yang juga disebut *failover cluster* pada umumnya diimplementasikan untuk tujuan meningkatan ketersediaan layanan yang disediakan oleh kluster. Elemen kluster memiliki *node-node* redundan yang akan digunakan untuk menyediakan layanan ketika salah satu komponen mengalami kegagalan. Dibutuhkan dua buah *node* sebagai syarat minimum suatu kluster untuk dapat melakukan redudansi[1].

   Macam-macam Failover:

   a. Active/Passive Failover

   Pada mode ini terdapat dua komponen, yang satu menjadi komponen atau *node* aktif dan yang lainnya pasif. *Node* aktif bertugas untuk melakukan eksekusi terhadap aplikasi atau tugas tertentu, sedangkan *node* pasif berstatus stand by dengan tidak melakukan tugas apapun sampai mendeteksi bahwa terdapat masalah pada *node* utama/aktif. Pada saat *node* utama mengalami kegagalan, *node* pasif akan mengambil alih tugas yang tadinya dilakukan oleh *node* utama[18].

   b. Active/Active Failover

   Pada mode ini, semua node berstatus aktif dengan menjalankan masing-masing aplikasi atau proses yang merupakan beban kerja masing-masing node. Apabila satu node aktif mengalami kegagalan, node aktif lain akan mengambil alih beban kerja node yang gagal tesebut sehingga node aktif yang masih berfungsi menjalankan seluruh aplikasi dan proses yang ada[18]. Tujuan dari *active/active failover* ini adalah untuk mencapai *load balancing*. Yang membedakan mode ini dengan *load balancing cluster* yaitu adanya konfigurasi untuk redudancy diantara kedua node yang aktif.

2. *Load-balancing Clusters*, kluster kategori ini beroperasi dengan mendistribusikan beban kerja secara merata kepada beberapa *node* yang bekerja dibelakang (*back-end node*) sehingga beban kerja disisi *server* menjadi lebih ringan. Tujuan dari *load balancing* adalah mempersingkat waktu rata-rata pengerjaan tugas pada *server* dan ketersediaan layanan yang tinggi[1].

3. *High Performance Cluster, cluster* yang bertujuan untuk meningkatkan kemampuan komputasi dengan memanfaatkan utilitas perangkat secara maksimal[1].

2.2 Virtualisasi Server

Virtualisasi *server* merupakan konsep baru pada perkembangan teknologi. Hal ini dikarenakan, virtualisasi *server* memungkinkan penggunaan satu perangkat keras untuk menjalankan beberapa sistem operasi secara independent dengan layanan yang berbeda pada waktu bersamaan[6]. Virtualisasi *server* sebagai paradigma baru memberikan efisiensi yang lebih dibanding dengan sistem *server* yang ada dengan meminimalkan jumlah perangkat fisik, tempat, biaya pemeliharaan sistem, dll[8].

2.2.1 Hypervisor[6]

Pengelola virtualisasi biasa disebut sebagai *hypervisor*. *Hypervisor* merupakan jenis perangkat lunak yang menciptakan mesin virtual, dimana beroperasi secara terpisah dari satu sama lain. *Hypervisor* memungkinkan sistem operasi yang berbeda untuk dijalankan secara terpisah dari satu sama lain meskipun masing-masing sistem ini menggunakan daya komputasi dan kemampuan penyimpanan pada komputer yang sama. *Hypervisor* biasa juga dikenal dengan nama *virtual machine monitor* (VMM) .

Pada umumnya *hypervisor* dikelompokkan dalam dua jenis, yaitu :

1. *Hypervisor* tipe-1
   Hypervisor tipe 1 disebut dengan *native / bare metal hypervisor*, yaitu *hypervisor* yang dapat langsung di *install* pada *hardware server* yang kosong (*baremetal)* yang belum berisi sistem operasi apapun. Artinya *hypervisor* ini telah menjadi satu paket dengan sistem operasi.
2. *Hypervisor* tipe-2
   Hypervisor tipe 2 disebut dengan *hosted hypervisor*, yaitu *hypervisor* yang berjalan diatas sistem operasi sehingga membutuhkan sistem operasi untuk dapat menjalankan *hypervisor* tersebut.

2.3 Proxmox[17]

Proxmox merupakan *software open source Virtualization Platform* untuk menjalankan *Virtual Appliance* dan *Virtual Machine*. Proxmox VE adalah distro khusus yang didedikasikan secara khusus sebagai mesin *host* virtualisasi sistem dan memuat 2 teknologi virtualisasi, yaitu KVM dan OpenVZ.

2.4 Web Server[7]

Web server merupakan perangkat lunak yang menyediakan layanan akses ke suatu berkas, berkas tersebut dapat berupa *Hypertext Markup Language (HTML),* berkas *Javascript,* dan berkas *Perl.* Komunikasi antara *client (web browser)* dan *server* menggunakan protocol yang disebut *Hypertext Transfer Protocol (Http).*

2.4.1 Apache Web Server

Apache merupakan web *server* unggul daripada banyak *server* web berbasis unix lainnya dalam hal fungsi, efisiensi dan kecepatan. Apache juga merupakan sebuah aplikasi *multi-platform, responsive*, dan juga *open-source*. Hampir 50% dari penyedia layanan web dalam sepuluh tahun terakhir ini menggunakan apache [9].

2.5 Haproxy

HAProxy adalah produk *open source* yang menyediakan solusi untuk menciptakan sistem *load balancing* dan *failover* dari aplikasi yang berbasis TCP dan HTTP. Perangkat lunak ini sangat cocok digunakan untuk *website* yang trafik hariannya tinggi sementara itu diperlukan keteguhan dan kekuatan dari pemrosesan pada *layer* 7. *HAProxy* dipasang pada *server front-end. Fron-end server* umumnya adalah *server* yang memiliki IP statis teregistrasi dengan DNS[12].

Berikut ini beberapa algoritma penjadwalan yang didukung Haproxy[11] :

- Round robin
- Weighted round robin
- Static round robin
- Least connection
- Source
- URI
- URL parameter

2.6 Keepalived

Keepalived merupakan *software routing* yang ditulis dalam bahasa C. Tujuan utama dari *software* ini untuk menyediakan fasilitas sederhana berkemampuan load balancing dan *high availability* untuk sistem linux dan infrastruktur berbasis linux. *Keepalived* mengimplementasikan bagian dari pengecekan secara dinamis dan adaptif, menjaga dan mengatur *load balanced* menurut keadaan koneksi mereka. Di sisi lain *high availability* dicapai dengan protokol VRRP. VRRP adalah dasar untuk *failover router*[15].

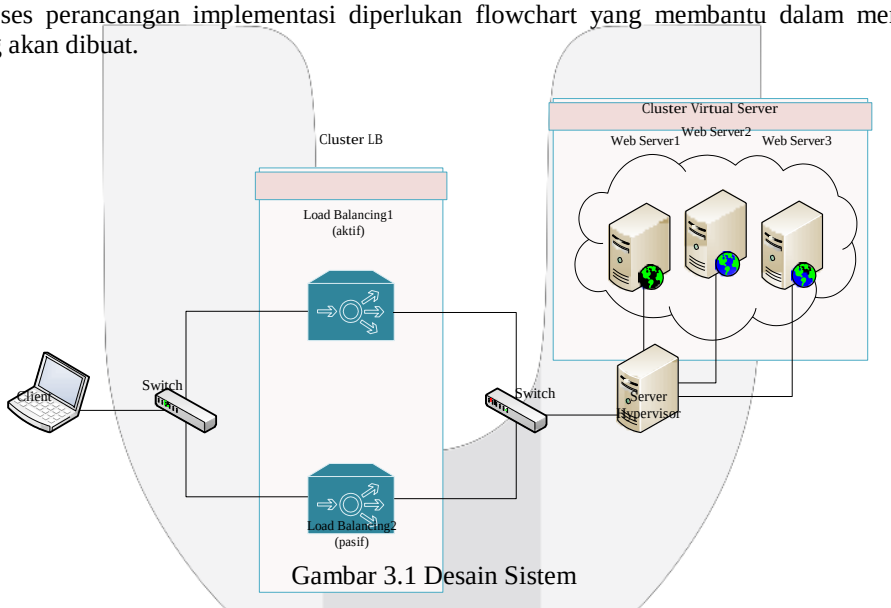
2.8 Availability

Berdasarkan dokumen ISO 2382-14, 1997 *availability* dapat didefinisikan sebagai: "Kemampuan sebuah alat untuk berada dalam kondisi siap pakai sesuai fungsi yang diinginkan pada waktu tertentu atau kapanpun dalam interval waktu tertentu, diasumsikan bahwa sumber eksternalnya bila diperlukan, adalah tersedia". Secara garis besar *availability* merupakan nilai persentase jumlah waktu suatu jaringan mampu memberikan layanan dibandingkan dengan jumlah waktu yang diharapkan. Sedangkan rata-rata waktu suatu sistem atau jaringan dalam kondisi *down* atau tidak mampu memberikan layanan disebut *downtime*. Sedangkan *uptime* didefinisikan sebagai waktu rata-rata suatu sistem atau jaringan dalam keadaan operasional[10]. Nilai Availability dapat dihitung dengan menggunakan persamaan 2.1 [14].

$$Availability = \frac{uptime}{(uptime+downtime)} x\ 100\% \quad (2.1)$$

3. Perancangan dan Implementasi

Dalam proses perancangan sebuah sistem, diperlukan sebuah skenario yang terstruktur dengan baik. Untuk memudahkan proses perancangan implementasi diperlukan flowchart yang membantu dalam memahami proses perancangan yang akan dibuat.



Gambar 3.1 Desain Sistem

Dalam implementasi, *load balancer* akan dikonfigurasi menggunakan Haproxy dan Nginx sebagai media *load balancing* untuk layanan http. Pada sisi *client* akan dibangkitkan *request* secara simultan yang nantinya akan didistribusikan *request* tersebut dengan algoritma penjadwalan *round robin* dan *least connection* kepada tiga buah *virtual web server*. Untuk meningkatkan ketersediaan *server* dalam melayani permintaan akan dilakukan skenario *failover* pada kedua *load balancer* kemudian dilihat nilai *downtime* nya. Dan sebagai pembuat lingkungan *virtual* serta sistem *clustering* di sisi *web server* menggunakan Proxmox.

3.1 Skenario Pengujian

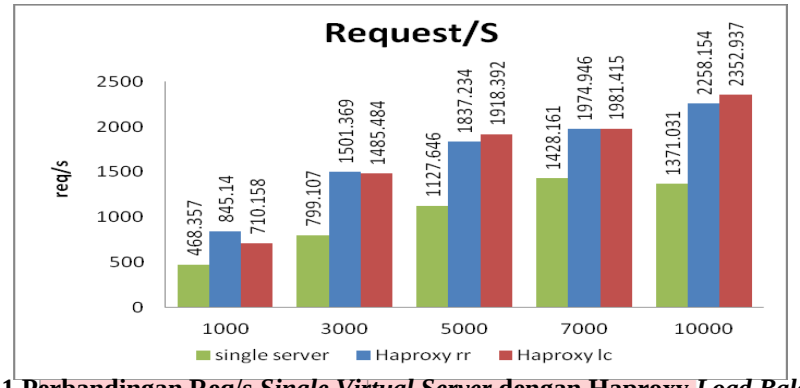
Berikut skenario yang telah dilakukan pada tugas akhir ini dengan parameter yang diukur adalah jumlah *request* per-detik, *throughput*, CPU *utilization* dan *request loss*:

a. Skenario pada *single virtual web server*.
b. Skenario penjadwalan *round robin* dan *least connection* menggunakan Haproxy.
c. Skenario *failover* pada *load balancer cluster*, untuk melihat nilai *downtime* yang terjadi.
d. Perbandingan kinerja Haproxy dengan Nginx menggunakan penjadwalan *round robin*.

4. Pengujian dan Analisis

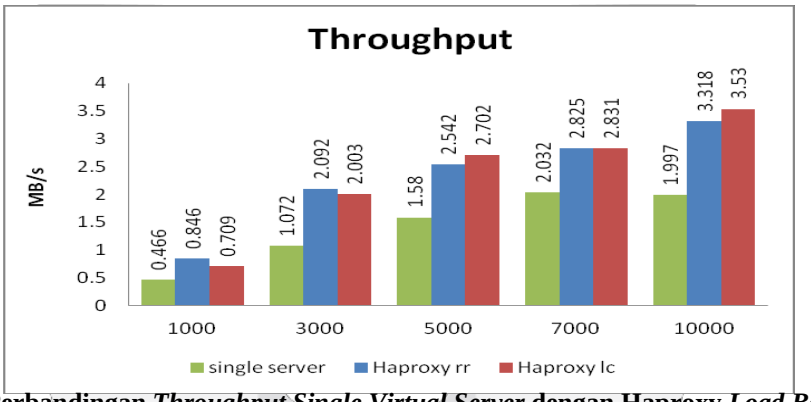
Berikut ini hasil pengujian dari tiga puluh kali pengamatan data:

4.1 Perbandingan Single Virtual Server dengan Haproxy Load Balancing



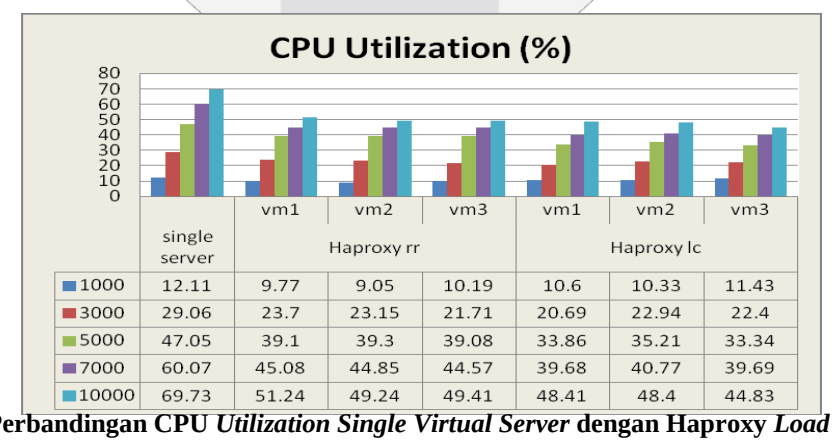
Gambar 4.1 Perbandingan Req/s *Single Virtual Server* dengan Haproxy *Load Balancing*

Jumlah *request* per-detik pada *single server* jauh lebih rendah dibandingkan dengan sistem *load balancing*. Hal ini dikarenakan pada sistem *load balancing* permintaan *request* yang datang dilayani oleh tiga buah *virtual server* yang saling bekerja sama sehingga jumlah *request* yang dapat dilayani meningkat. Didapatkan nilai maksimal jumlah *request* per-detik sebesar 2352.937 *request* per-detik. Nilai tersebut dapat mempengaruhi nilai *throughput* pada *load balancer*, semakin besar jumlah *request* per-detik maka semakin besar *throughput* yang didapat.



Gambar 4.2 Perbandingan *Throughput Single Virtual Server* dengan Haproxy *Load Balancing*

Hasil pengukuran secara keseluruhan menunjukkan bahwa nilai *throughput* lebih besar saat sistem menggunakan *load balancing* dibandingkan dengan nilai *throughput* pada *single virtual server*. Hal ini dikarenakan terdapat lebih dari satu *server* yang melayani permintaan yang datang. Didapatkan nilai maksimal *throughput* sebesar 3.53 MB/s saat *server* menangani 10000 *request* dengan sistem *load balancing*.

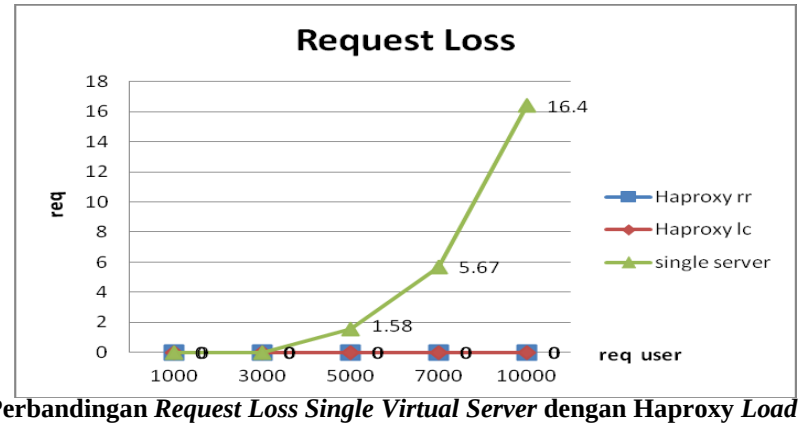


	single server	Haproxy rr			Haproxy lc		
		vm1	vm2	vm3	vm1	vm2	vm3
1000	12.11	9.77	9.05	10.19	10.6	10.33	11.43
3000	29.06	23.7	23.15	21.71	20.69	22.94	22.4
5000	47.05	39.1	39.3	39.08	33.86	35.21	33.34
7000	60.07	45.08	44.85	44.57	39.68	40.77	39.69
10000	69.73	51.24	49.24	49.41	48.41	48.4	44.83

Gambar 4.3 Perbandingan CPU *Utilization Single Virtual Server* dengan Haproxy *Load Balancing*

Seiring meningkatnya jumlah permintaan dari *user* maka akan memberi peningkatan pada nilai CPU. Pada *single virtual server* didapat beban kerja yang jauh lebih tinggi dibandingkan dengan sistem *load balancing* dengan algoritma *round robin* maupun *least connection*. Terjadi penurunan sebesar 18-21% pada pengujian *load balancing* dikarenakan beban tidak terpusat pada satu *server*. Pada penjadwalan round robin nilai CPU tidak akan sama persis

antara VM1, VM2, dan VM3 walaupun pembagian dilakukan secara merata. Hal ini dikarenakan pada salah satu atau dua server akan mendapat beban lebih satu dibanding lainnya, dan response time untuk setiap request yang diproses berbeda-beda. Semakin besar nilai response time akan membuat antrian sehingga nilai CPU akan semakin tinggi. Namun nilai CPU yang didapat pada penjadwalan round robin tidak memililki perbedaan yang signifikan.

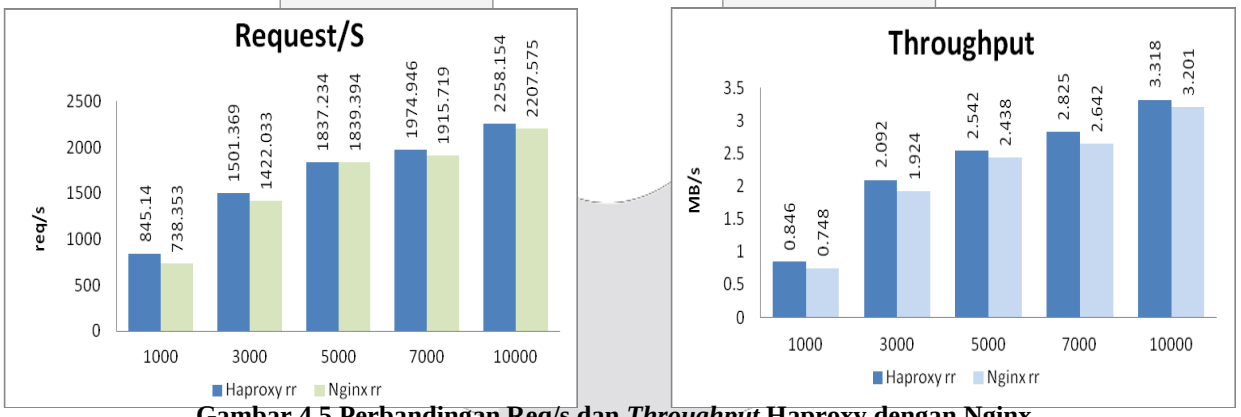


Gambar 4.4 Perbandingan *Request Loss Single Virtual Server* dengan Haproxy *Load Balancing*

Nilai *request loss* pada *single server* mengalami kenaikan dimulai saat *server* mendapat 5000 *request* secara bersamaan. Single virtual server tidak mampu melayani seluruh *request* yang datang sehingga nilai *request loss* semakin naik seiring meningkatnya *request* yang datang. Pada sistem *load balancing* dengan algoritma penjadwalan *round robin* maupun *least connection* tidak terdapat *request loss.*

4.2 Perbandingan Haproxy denga Nginx

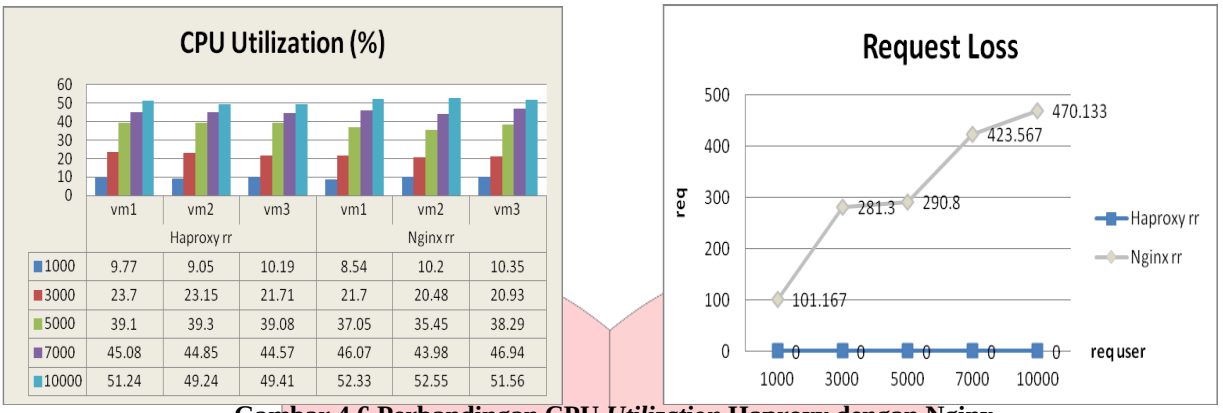
Pada bagian ini akan dipaparkan perbandingan kinerja dari sistem *load balancing* menggunakan Haproxy dan Nginx dengan menggunakan algoritma penjadwalan *round robin.*



Gambar 4.5 Perbandingan Req/s dan *Throughput* Haproxy dengan Nginx

Didapatkan jumlah *request* per-detik dari *load balancing* haproxy lebih besar dibandingkan dengan nginx walaupun tidak berbeda jauh karena kedua *load balancer* sama-sama bekerja pada *layer* 7 dan dengan algoritma penjadwalan yang sama yaitu *round robin*. Nilai jumlah *request* per-detik maksimal haproxy yaitu sebesar 2258.154 req/s sedangkan 2207.575 req/s untuk nginx.

Gambar di atas menunjukkan bahwa kinerja sistem *load balancing* dengan menggunakan Haproxy lebih baik dibandingkan dengan Nginx. Didapat nilai *throughput* maksimal *load balancing* Haproxy yaitu sebesar 3.318 MBps.



Gambar 4.6 Perbandingan CPU *Utilization* Haproxy dengan Nginx

Nilai CPU *Utilization* untuk kedua *load balancer* memiliki perbedaan yang tidak terlalu signifikan. Nilai CPU pada VM1, VM2 dan VM3 cenderung sama untuk *load balancer* Haproxy maupun Nginx karena menggunakan penjadwalan *round robin* yang memberikan bobot sama rata kepada semua *server*.

Sistem *load balancing* dengan Haproxy memiliki kinerja yang jauh lebih baik dalam mendistribusikan beban dan juga meneruskan paket. Pada Haproxy tidak terdapat *request loss* sampai dengan pembangkitan 10000 *request*, sedangkan pada sistem Nginx sudah terdapat *request loss* mulai dari pembangkitan 1000 *request* dan terus naik secara linear seiring meningkatnya jumlah *request*. Nilai *request loss* tertinggi berada saat pembangkitan 10000 *request*, yaitu sebesar 470.133 *request* yang hilang.

4.3 Skenario Failover

Pada skenario ini dilakukan pengukuran menggunakan hrping, dengan mengirimkan paket ICMP dimana diberikan *delay* sebesar 5ms untuk setiap paket yang dikirim. Baik *load balancer master* maupun *slave* mempunyai konfigurasi masing-masing sebagai *load balancing*, akan tetapi karena pada pengujian ini menggunakan mode aktif/pasif hanya pada satu *load balancer* saja yang memliki IP *floating* yaitu pada *load balancer master*. Didapatkan nilai rata-rata *downtime* yang diperoleh dari dua puluh lima kali hasil pengamatan yaitu sebesar 1992.8 ms.

5. Kesimpulan dan Saran

5.1 Kesimpulan

Dari hasil implementasi dan pengujian di atas dapat diambil kesimpulan, yaitu:

a. Kemampuan sistem *load balancing* dengan tiga buah *virtual server* dalam melayani *request* jauh lebih baik dibandingkan *single virtual server*, karena *server* saling bekerja sama dan beban tidak terpusat pada satu *server* saja sehingga tidak memberatkan *server*. Terbukti dengan adanya penurun CPU *Utilization* sebesar 21% .
b. Dengan sistem Haproxy *load balancer* dapat melayani hingga 2352.937 req/s sedangkan *pada single virtual server* hanya sampai 1428.161 req/s.
c. Untuk meningkatkan ketersediaan *server*, sistem *load balancing* Haproxy berhasil melakukan *failover* dengan rata-rata nilai *downtime* yang didapat yaitu sebesar 1992.8 ms. Hal ini disebabkan adanya penyalinan proses dan konfigurasi dari *master* ke *slave*.
d. Pada sistem *load balancing* Haproxy, algoritma penjadwalan *least connection* lebih unggul dibandingkan algoritma *round robin*. Didapatkan nilai *request* per-detik sebesar 2352.937 req/s dan *throughput* sebesar 3. 53 MB/s untuk *least connection*, sedangkan 2258.154 req/s dan 3.318 MB/s untuk *round robin*.
e. Sistem Haproxy *load balancer* terbukti dapat meminimalisir jumlah *request loss*, dimana dalam pengujian ini tidak terdapat *request* yang hilang atau sama dengan nol (0). Pada Nginx sudah terdapat *request loss* mulai dari pembangitan 1000 *requests* yaitu sebesar 101.67 *requests*.
f. Performansi Haproxy *load balancer* lebih baik dibandingkan Nginx, dilihat dari nilai *throughput* yang didapat pada Haproxy sebesar 3.53 MB/s dan 3.318 MB/s untuk Nginx. Nilai *throughput* dipengaruhi oleh jumlah *request* per-detik. Semakin besar jumlah *request* per-detik semakin besar nilai *throughput*. Haproxy mampu melayani sampai 2258.154 req/s sedangkan Nginx 2207.575 req/s.

5.2 Saran

Berikut ini beberapa saran yang dapat disampaikan guna pengembangan lebih lanjut, antara lain:

a. Penelitian dilakukan dengan menggunakan perangkat server yang sesungguhnya.
b. Menggunakan jenis layanan yang berbeda seperti video streaming atau jenis layanan real time.
c. Menggunakan lebih dari satu macam layanan dalam satu cluster.
d. Menggunakan mode aktif/aktif atau load-sharing pada load balancer.
e. Mengunakan shared storage

Daftar Pustaka:

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

[2] Pribadi, P. T., "IMPLEMENTASI HIGH-AVAILABILITY VPN CLIENT PADA JARINGAN KOMPUTER FAKULTAS HUKUM UNIVERSITAS UDAYANA". Jurnal Ilmu Komputer - Volume 6 - No 1, 2013.

[3] Kaur, K., Rai, A. K., "A Comparative Analysis: Grid, Cluster and Cloud Computing". International Journal of Advanced Research in Computer and Communication Engineering Vol. 3, Issue, 2014.

[4] Microsoft Corporation. "Load-Balanced Cluster", https://msdn.microsoft.com/en-us/ff648960. [Diakses 11 Desember 2015]

[5] Andreolini, M., Casalicchio, E., Colajanni, M., Mambelli, M., "A Cluster-Based Web System Providing Differentiated and Guaranteed Services". Kluwer Academic Publishers, Netherlands, 2004.

[6] Obasuyi, G. C., Sari A., "Security Challenges of Virtualization Hypervisors in Virtualized Hardware Environment". Int. J. Communications, Network and System Sciences, 2015, 8, 260-273.

[7] Herdian, R., "IMPLEMENTASI DAN ANALISIS KINERJA LOAD BALANCING PADA VIRTUAL SERVER MENGGUNAKAN ZEN LOAD BALANCER". Fakultas Elektro dan Komunikasi Universitas Telkom, Bandung, 2015.

[8] Menasche, D. A., "Virtualization: Concepts, Applications, and Performance Modeling". Department of Computer Science George Mason University, 2005.

[9] Apache. Retrieved from https://www.apache.org. (2015, November)

[10] Asterina, D., "IMPLEMENTASI DAN ANALISIS METODE *FAILOVER* PADA SISTEM *REDUNDANT DEDICATED SERVER* DAN *CLOUD SERVER* UNTUK LAYANAN VOIP". Fakultas Teknik Elektro, Bandung, 2015.

[11] Kuba, M., Sapak, T., Charvat, K., Berzins, R., Kepka, M. "Uptake of Open Geographic Information Through Innovative Services Based on Linked Data". D3.2.1 ENABLERS DEPLOYMENT – FIRST RELEASE, SDI4Apps revision no.4, 2014.

[12] Haproxy. Retrieved from https://www.haproxy.org. (2015, November)

[13] Muchtar, A., "IMPLEMENTASI FAILOVER CLUSTERING PADA DUA PLATFORM YANG BERBEDA UNTUK MENGATASI KEGAGALAN FUNGSI SERVER". Fakultas Teknik Universitas Hasanuddin, Makasar.

[14] Lee, J., Lee, K. "SYSNCEYE: An Availability Measurement Tool for Embedded System". Asia-Pasific Software Engineering Conference, 2014.

[15] Keepalived. Retrieved from https://www.keepalived.org. (2015, November)

[16] Nginx. Retrieved from https://www.nginx.com. (2015, November)

[17] Suryono, T., Afif, M. F., "Pembuatan Prototype Virtual Server Menggunakan PROXMOX VE Untuk Optimalisasi Resource Hardware di NOC FKIP UNS". IJNS – Volume 1 Nomor 1, 2012.

[18] Rao, G. P., Brueggemann, E. R., Rodriguez, R. A., "Method for maintaining transaction integrity across multiple remote access servers**".** US 11/626,334, 2010.