IMPLEMENTASI LOAD BALANCING WEB SERVER MENGGUNAKAN HAPROXY

Syaqia Azizah
D3 Manajemen Informatika, Fakultas Teknik, Universitas Negeri Surabaya, syaqiaazizah@gmail.com

Asmunin
Jurusan Teknik Informatika, Fakultas Teknik, Universitas Negeri Surabaya, asmunin@unesa.ac.id

Abstrak

Ketika ribuan pengguna mengakses situs atau aplikasi web pada waktu bersamaan. Maka akan membuat web server kelebihan beban (*overload*) karena tidak dapat menampung *request* yang diterima. Permasalahan tersebut umumnya terjadi ketika web server bermesin tunggal atau *single* web server. Namun dengan teknik *load balancing* menggunakan Haproxy, dapat membantu permasalahan tersebut dalam mengatur pembagian beban. Implementasi yang dilakukan yaitu dengan bantuan *webserver stress* dalam mensimulasi jumlah beban 250 *user*, 500 *user*, 750 *user*, 1000 *user*, 1250 *user* dan web statis untuk pemberian beban pada pengujian pemerataan beban, hingga melihat pengaruh CPU dengan beban *user* yang disimulasikan. Hasil yang dicapai adalah implementasi *load balancing* menggunakan haproxy dapat berjalan secara fungsional dan telah dikerjakan sesuai skenario pengujian. Untuk menjadi solusi cepat memperbaiki kinerja web server dalam melayani *request* dari pengguna, terbukti dengan pengujian menggunakan *apache bench* dalam menghitung *throughput* dan *time per request* antara *single* server dan haproxy. Dengan hasil bahwa *throughput* yang digunakan lebih besar haproxy dari *single* server yang berarti menggunakan haproxy lebih baik dari *single* server dan hasil *time per request* lebih besar *single* server dari haproxy yang berarti waktu yang dibutuhkan menangani *request* pada *single* server lebih besar dari haproxy dan terbukti bahwa haproxy lebih cepat dalam menangani *request*.
Kata kunci : Load Balancing, Haproxy, Web Server

Abstract

When thousands of users access the site or web app at the same time. So it will make the web server overloaded (overload) because it can not accommodate the received request. Such problems generally occur when a single-engine or single web server web server. However, with load balancing techniques using Haproxy, can help the problem in managing the distribution of the load. Implementation is done with the help of stress webserver in simulating the load number of 250 users, 500 users, 750 users, 1000 users, 1250 users and static web for load loading on load equity testing, to see the effect of CPU with simulated user load. The results achieved are the implementation of load balancing using haproxy can run functionally and has been done according to the test scenario. To be a quick solution to improve web server performance in serving requests from users, it is proven by testing using apache bench in calculating throughput and time per request between single server and haproxy. With the result that greater throughput is used haproxy than single server which means using haproxy better than single server and result of time per request bigger single server from haproxy which mean time required handle request on single server larger than haproxy and proved that haproxy faster in handling requests.
Keywords: Load Balancing, Haproxy, Web Server

PENDAHULUAN

Kini internet sudah menjadi salah satu kebutuhan yang penting. Semakin banyak orang yang mengakses informasi melalui internet. Dan semakin banyak *website* yang dikunjungi pengguna internet dalam mencari berbagai informasi yang dibutuhkan. *Website* sebagai komponen situs *online*, pasti sering dikunjungi oleh *user* (pengguna) dan akan terus bertambah. Ini membuat *web server* memiliki peran penting dalam menangani *request* dari para *user*, tetapi *web server* memiliki kemampuan berbeda dalam menampung *request* dari para *user*. Kebanyakan *website* dan aplikasi *web*, berjalan lancar

selama hanya tiga *user* atau hanya beberapa *user* mengunjungi pada waktu tertentu. Tapi, ketika yang terjadi ribuan pengguna mengakses situs atau aplikasi *web* pada saat waktu yang sama. Karena semakin banyak *user* yang melakukan *request* pada sebuah *web server*, maka *web server* akan kelebihan beban (*overload*) dan akan *down* ketika *web server* tidak dapat menampung *request* tersebut.

Permasalahan tersebut umumnya terjadi ketika *web server* bermesin tunggal atau *single web server*. Salah satu solusinya yaitu dengan menggunakan *load balancer*. *Load balancer* adalah perangkat lunak yang berguna untuk membagi beban agar menjadi seimbang dalam melayani suatu *request* dari para *user*. Pembagian beban permintaan atau akses dari *user* akan merata ke semua *server* yang ada, kinerja *server* juga menjadi lebih baik. *Haproxy* sebagai *load balancer*, digunakan untuk mengatur pembagian beban tersebut. Selain itu *haproxy* termasuk *software* gratis yang dapat diandalkan dalam hal *high availibility*, *load balancing*, dan *proxy* untuk TCP dan HTTP. Tujuan dari penelitian tugas akhir ini adalah mengimplementasi *load balancing web server* menggunakan haproxy; sebagai solusi yang dapat menangani kelebihan beban bagi *web server*. Manfaat dari penelitian tugas akhir ini adalah engan adanya *haproxy* sebagai *load balancer web server*, dapat membantu dalam meminimalkan terjadinya *overload* pada *web server*; dengan menggunakan teknik *load balancing* dengan algoritma *round robin*, pembagian kerja *web server* akan secara adil dan merata.

KAJIAN PUSTAKA

Load balancing

Load Balance dalam jaringan komputer adalah teknik untuk membagi beban (*load*) ke dalam beberapa jalur (*link*). Tujuan dari load balance ini agar tidak ada link yang mendapatkan beban yang lebih besar dari link yang lain. Diharapkan dengan membagi beban ke dalam beberapa link tersebut, maka akan tercapai keseimbangan (*balance*) penggunaan *link-link* tersebut (Towidjojo, 2013).

Load balancing adalah membagi jumlah pekerjaan yang harus dilakukan komputer antara dua atau lebih komputer sehingga setiap komputer melakukan pekerjaan dalam jumlah dan waktu yang sama serta secara umum semua pengguna mendapatkan akses cepat. Load balancing bertujuan untuk mengoptimalkan sumber daya, memaksimalkan throughput, meminimalkan waktu respon dan menghindari kelebihan beban dari sumber daya tunggal (Lakhe, Shinde, Sukhthankar, & Reddy, 2016)

Haproxy

HAproxy merupakan solusi gratis, sangat cepat dan dapat diandalkan dalam high availability, load balancing, dan proxy untuk TCP dan HTTP berbasis aplikasi. Sangat cocok untuk lalu lintas situs website yang sangat tinggi dan kekuatan cukup banyak yang paling sering dikunjungi di dunia. Selama bertahun-tahun telah menjadi de-facto *opensource* standart beban penyeimbang, kini dikirimkan dengan sebagian besar distribusi Linux utama dan sering digunakan secara default di *platform cloud*. Karena tidak mengiklankan dirinya, hanya bisa tahu itu digunakan ketika admin melaporkannya. Modus operasinya membuat integrasi ke dalam arsitektur yang ada sangat mudah dan tanpa resiko.

Versi 1.5, dirilis pada tahun 2014 Versi ini memperluas 1.4 lagi dengan 4 tahun kerja keras: dukungan SSL asli di kedua sisi dengan stapel SNI / NPN / ALPN dan OCSP, soket IPv6 dan UNIX didukung di mana-mana, HTTP penuh tetap bertahan untuk mendapatkan dukungan yang lebih baik. dari NTLM dan peningkatan efisiensi di *static farms*, kompresi HTTP / 1.1 (deflate, gzip) untuk menghemat *bandwidth*, protokol PROXY versi 1 dan 2 di kedua sisi, pengambilan data pada segala hal yang sesuai permintaan atau tanggapan, termasuk muatan, ACL dapat menggunakan metode yang sesuai. dengan peta masukan masukan dan ACL yang dinamis yang dapat diupdate dari tabel *stick* CLI-*stick*, *support counter* untuk melacak aktivitas pada setiap format kustom sampel masukan untuk *log*, *unique-id*, penulisan ulang *header*, dan pengalihan, pemeriksaan kesehatan yang lebih baik (SSL, *scripted* TCP, *check agent*, ...), konfigurasi yang lebih terukur mendukung ratusan ribu *backends* dan *certificates without sweating* (Haproxy, 2017).

Web server

Web server adalah sebuah komputer yang mana kandungan web disimpan. Pada dasarnya web server yang digunakan untuk meng-host situs web tetapi ada server web lain juga seperti game, penyimpanan, FTP, e-mail dll. Situs web adalah koleksi halaman web whileweb server adalah sebuah software yang menanggapi permintaan sumber daya web (Tutorials Point Originated, 2017).

Apache

Apache adalah web server paling populer di internet. Digunakan untuk melayani lebih dari setengah dari semua situs web yang aktif. Meskipun ada banyak web server layak yang akan melayani

konten, ini sangat membantu untuk memahami bagaimana Apache bekerja karena ubiquity nya (Ellingwood, 2013).

Webserver stress tools

Web stress merupakan alat yang dapat digunakan untuk mensimulasikan berbagai pola beban web server dalam menemukan masalah di server web. Dapat mengetahui berapa banyak beban server yang dapat ditangani sebelum muncul masalah serius.

Webserver stres mensimulasikan sejumlah besar pengguna mengakses situs web melalui HTTP/HTTPS. Perangkat lunak dapat mensimulasikan hingga 10.000 pengguna yang independen klik jalan mereka melalui set URL. Pola URL sederhana didukung serta pola URL yang kompleks (melalui Script file).

Webserver stres mensimulasikan independen pengguna melangkah melalui set URL atau URL yang dapat ditentukan menggunakan VB script. Berdasarkan parameter yang ditentukan, aplikasi tidak hanya meminta HTML URL tetapi juga frame, gambar, berkas flash, dll yang meniru tingkah laku yang sama dengan web browser yang akan menunjukkan saat mengakses situs web.

Setiap pengguna disimulasikan oleh thread terpisah dengan informasi sesi sendiri (yaitu, cookie untuk setiap simulasi pengguna disimpan secara terpisah) dan "surfs" URL secara independen dari pengguna lain seperti di dunia nyata penggunaan Web.

URL dapat diberi parameter untuk setiap pengguna dan urutan URL dapat bervariasi. Posting dan mendapatkan permintaan didukung otentikasi HTTP dasar dan beberapa pengaturan lainnya. Dengan fungsi skrip baru, bahkan dapat membuat pola URL yang sangat kompleks untuk aplikasi web berskala besar.

Jumlah beban yang dapat dihasilkan oleh alat web stress telah berhasil dalam pengujian dengan lebih dari 1 Gigabit/s jaringan throughput lebih dari 1.000.000 tampilan halaman per jam dan hingga 10.000 simultan pengguna (AG, 2017).

AB (apache http server benchmarking tool)

Ab adalah alat untuk membandingkan server Apache Hypertext Transfer Protocol (HTTP). Ini dirancang untuk memberi kesan bagaimana kinerja instalasi Apache saat ini. Terutama menunjukkan berapa banyak permintaan per detik instalasi Apache mampu melayani (The Apache Software Foundation (ASF), 2017).

Overload

Banyak hal yang menyebabkan *website* menjadi lambat ketika diakses padahal sebelumnya sangat cepat. Salah satu hal yang menyebabkan hal ini karena *website* menggunakan resource (CPU, MEMORI, VIRTUAL MEMORI dan limit lainnya) yang telah disediakan untuk akun. Itu dapat selalu di cek status penggunaan resource di akun melalui menu ***Resource Usage*** pada *cpanel* dengan *thema X3* dan pada *Paper Latern*, cek pada CPU and *Concurrent Connection Usage*.

Penggunaan *plugin* juga yang memberatkan (contoh: *plugin Jet pack* jika menggunakan *wordpress*) atau *script* atau *thema* dari sumber yang tidak dapat dipercaya. *Script* atau *plugin* atau *theme* dari sumber yang tidak dapat dipercaya seringkali menjadi sasaran *malware* sehingga sering digunakan untuk tindakan kejahatan dan menyebabkan akun *overload* (The Apache Software Foundation (ASF), 2017).

METODE REKAYASA

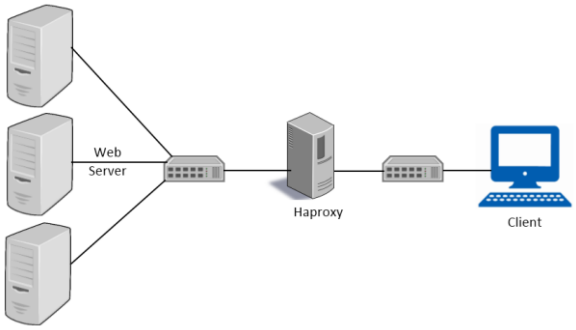
Analisa Sistem

Load balance merupakan teknik pembagian beban agar menjadi seimbang. Teknik *load balance* ini akan di implementasikan pada *web server* menggunakan *haproxy* sebagai *load balancer* dengan bantuan *webserver stress tools* 8 sebagai alat pengujian yang nantinya akan dipasang di *client*. Versi haproxy yang digunakan yaitu 1.5.8 yang dirilis pada 31 Oktober 2014, itu dapat dilihat di laporan statistik haproxy. Haproxy merupakan *open source* yang mendukung *high availability*, *load balancing*, dan *proxy* untuk TCP dan HTTP, beserta tampilan statiknya. Untuk sistem operasi nya menggunakan *debian* atau *ubuntu*, jika menggunakan debian 7 harus menambahkan *package* melalui *backport* yang harus di simpan di file /etc/apt/sources.list.d/ pada sistem operasi debian 7 agar *haproxy* dapat terpasang.

Web servernya menggunakan *apache*. Dan *web* yang akan digunakan yaitu *web* statis sederhana dan hanya HTTP saja. Nantinya ketika *client* mengakses *web* tersebut melalui *browser*, akan muncul tulisan bahwa *client* sedang dilayani oleh *web server* berapa beserta IP nya di *web* tersebut. Untuk *webserver stress tools* akan membantu pengujian simulasi jumlah beban *user*, jumlah beban *user* yang disimulasikan yaitu 250 *user*, 500 *user*, 750 *user*, 1000 *user*, dan 1250 *user*. Pengujiannya dari *client* yang nanti akan diterima Haproxy. Dan *haproxy* sendiri yang membagi beban secara merata dari semua permintaan *client* ke *web server*, agar ketiga *web server* menerima permintaan

dengan banyak beban yang adil dan merata. Pemerataan bebannya bisa diliat di laporan statistik *haproxy*. Untuk penambahan *apache bench* digunakan dalam menghitung *thoughput* dan *time per request* pada penggunaan *haproxy* dan *single server*. Agar dapat menunjukkan hasil perbandingan penggunaan antara menggunakan *haproxy* dan *single server*.

Desain Sistem

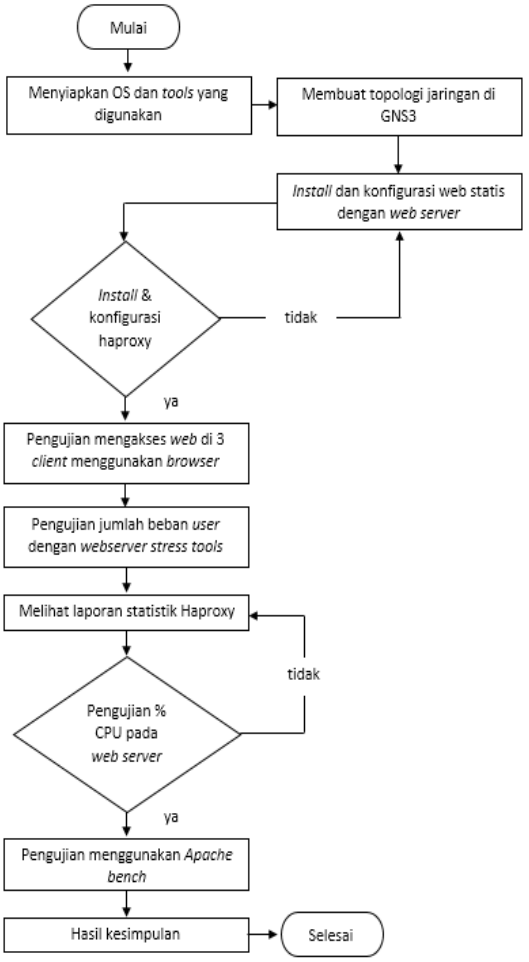


Gambar 1. Topologi yang akan digunakan

Pada gambar 1 menunjukkan jalannya sistem memakai haproxy sebagai *load balancer*. Terdapat 3 server debian 7 yang telah di beri web statis, lalu ada *haproxy* sebagai *load balancing* dengan OS debian 7 yang ditaruh didepan *web* server, lalu tersambung ke *client* yang menggunakan OS windows xp. Nantinya *client* bisa ditambah lagi hingga 3 *client*. Cara kerjanya yaitu ketika *user* atau *client* mengunjungi *web* yang telah dibuat, maka permintaan akan masuk ke *haproxy* terlebih dulu sebagai penengah atau pembagi beban yang masuk, selanjutnya akan diteruskan mengatur jalannya *user* ke *web server* mana *user* akan diarahkan. Pengaturan jalannya *user* tersebut dilakukan oleh *haproxy*. Karena sudah terdapat 3 *web server*, maka semua *user* yang mengakses tidak akan ditimpakan hanya ke *web server1* saja. Melainkan akan dibagi ke *web server2* dan *web server3* agar ketiga *web server* menangani beban yang sama. Lalu *web* yang dikunjungi *user* akan menampilkan tulisan berisi *web server* berapa & IP nya yang sedang melayani *user*. Untuk membuat seakan-akan *web server* dikunjungi oleh banyak *user* atau *client*, maka digunakanlah *webserver stress tools* 8 yang dipasang di *client* sebagai alat pengujinya.

Perancangan Pengujian

Alur perencanaan pengujian merupakan alur panduan yang nantinya dilakukan untuk tugas akhir ini. Terdapat beberapa tahapan yang dilakukan yaitu mulai menyiapkan OS dan *tool* yang digunakan agar mengetahui kebutuhan sistem yang akan dikerjakan hingga hasil kesimpulan *load balancing* yang dilakukan untuk mengetahui hasil implementasinya. Pengujian pertama meliputi mengakses web di 3 *client* menggunakan *browser*, dengan *url* yang dibuat yaitu *www.tasyaqia.com*. Pengujian kedua adalah uji jumlah beban *user* dengan *clicks test* pada *webserver stress* dan jumlah yang digunakan 250 user, 500 user, 750 user, 1000 user, dan 1250 user. Pengujian ketiga adalah % CPU web *server* dengan memberi jumlah beban *user* menggunakan *ramp test* 1 menit dengan beban yang di berikan sama dengan pengujian jumlah beban *user*, lalu mengamatinya di web server dengan *system monitor*. Dan uji terakhir yaitu uji *throughput* serta *time per request* menggunakan *apache bench* pada web *server*. Urutan langkah yang dilakukan seperti pada gambar 2.



Gambar 2. Perancangan Sistem

Skenario Pengujian

Skenario pengujian *haproxy* sebagai *load balancer* yang akan dilakukan sebagai tahap menganalisa hasil dari uji coba. Tahapan pengujiannya sebagai berikut:

1. Pengujian koneksi antara *web server apache*

dengan *web* statis yang dimasukkan ke *web server*. Pada pengujian ini memperlihatkan tersambungnya web yang dibuat dengan di akses *client* melalui *browser*. Lalu untuk pengujian *haproxy* sebagai *load balancer* pada *web server* yaitu dengan mematikan salah satu server. Ketika salah satu server dimatikan maka seharusnya *haproxy* melakukan pembagian beban terhadap *server* yang masih aktif dan *web* masih bisa berjalan. Yang berarti *client* yang mengakses web akan dialihkan ke server lain yang aktif.

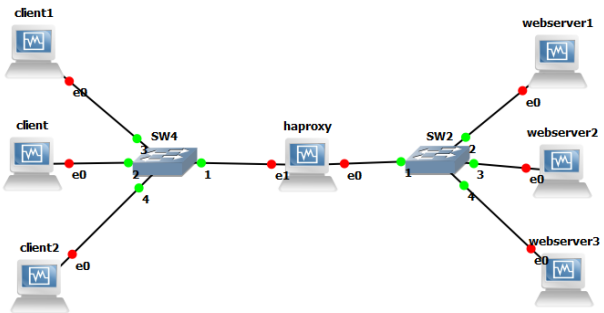
2. Pengujian *web server* menggunakan *webserver stress tools* yang dipasang pada *client*. Untuk pengujian jumlah beban *user* menggunakan jenis pengujian yang ada di *webserver stres* yaitu *click test* karena hanya menguji jumlah *user* yang mengakses dengan 2x klik per *user*. Pengujian ini berguna untuk membantu pengujian dalam mensimulasikan banyaknya pengguna yang mengakses *web* dan untuk melihat *limit* haproxy dalam menangani jumlah *user* yang di uji kan dapat dilihat di laporan statistiknya haproxy sendiri.
3. Pengujian mengamati kinerja RAM pada web server (*worker nodes*) dengan *system monitor* di web server untuk pengujiannya lalu dibuatkan tabel dan grafiknya. Dikarenakan pada pengujian yang telah dilakukan tidak terlalu berpengaruh pada RAM, maka akan diganti mengamati kinerja CPU. Pengamatan CPU dilakukan agar terlihat *load balancing* yang terjadi, namun dengan bantuan laporan statistik haproxy juga yang nantinya akan dilihat pembagian bebannya.

Hasil Implementasi

Tugas akhir ini bertujuan untuk mengimplementasikan *load balancing web* server menggunakan *haproxy*. Proses *load balancing* dilakukan untuk membantu server menangani permintaan pengguna yang terlalu banyak secara bersamaan hingga membuat server kelebihan beban (*overload*) dan kewalahan dalam menanganinya. Dikatakan *overload* yaitu suatu keadaan server yang menangani banyak pengguna hingga kinerja server melambat dalam menanganinya dan waktu tunggu pengguna menjadi semakin lama.

Load balancer bertindak sebagai penengah diatara layanan utama dan pengguna, dimana layanan utama merupakan sekumpulan server/mesin yang siap melayani banyak pengguna. Disaat *load balancer* menerima permintaan layanan dari *user*, maka permintaan tersebut akan diteruskan ke server utama. Biasanya *load balancer* dengan pintar dapat menentukan server mana yang memiliki *load* yang lebih rendah dan *respons* yang lebih cepat. Bahkan bisa menghentikan akses ke server yang sedang mengalami masalah dan hanya meneruskannya ke server yang dapat memberikan layanan. Hal ini salah satu kelebihan yang umumnya dimiliki *load balancer*, sehingga layanan seolah olah tidak ada gangguan di mata pengguna.

Dalam proses pengerjaan tugas akhir ini terdapat beberapa pengaturan dan pengamatan yang dilakukan dan dijelaskan lebih detail pada bab ini. Berikut adalah arsitektur implementasi *load balancing* menggunakan *haproxy* di GNS3 terdiri dari 3 *client* dengan OS *windows xp*, 2 *switch*, 4 OS debian 7 yang 1 buah untuk haproxy dan 3 buah untuk web server. *Load balancer* (perangkat *load balancing*) menggunakan beberapa peralatan yang sama untuk menjalankan tugas yang sama. Hal ini memungkinkan pekerjaan dilakukan dengan lebih cepat dibandingkan apabila dikerjakan oleh hanya 1 peralatan saja dan dapat meringankan beban kerja peralatan, serta mempercepat waktu *respons*. Dihubungkan dengan LAN tanpa internet, tampilannya ditunjukkan pada gambar 3.



Gambar 3. Tologi *Load balancing*

Dari gambar 3 penjelasan alamat IP statik yang dibuat pada struktur *load balancing* tersebut seperti berikut.

1. IP 192.168.137.19 yaitu eth0 yang dimiliki oleh haproxy menuju ke web server.
2. IP 192.168.137.16 adalah eth0 web server1
3. IP 192.168.137.17 adalah eth0 web server2
4. IP 192.168.137.18 adalah eth0 web server3
5. IP 192.168.138.20 yaitu eth1 yang dimiliki oleh haproxy menuju ke *client*.
6. IP 192.168.138.21 adalah eth0 *client*
7. IP 192.168.138.22 adalah eth0 *client1*
8. IP 192.168.138.23 adalah eth0 *client*2

Berikut adalah hasil konfigurasi yang dilakukan :

1. *Install* dan konfigurasi haproxy

a) Konfigurasi *listen* di file haproxy.cfg
b) Konfigurasi IP *localhost* & *url*

2. *Install* dan konfigurasi apache sebagai web server
   a) Konfigurasi IP statik
   b) Konfigurasi IP *localhost* & *url*
3. Konfigurasi *client*
   a) Konfigurasi IP statik
   b) Konfigurasi IP *localhost* & *url*

Hasil Implementasi dan Pembahasan

Pengujian yang dilakukan yaitu sesuai dengan skenario pengujian yang sudah dibuat pada tahap sebelumnya yaitu sesuai berikut:

1. Pengujian jumlah beban *user* menggunakan *webserver stress tools.*

Tabel 1. Hasil *result per url*

Jumlah User	Clicks	Error	Error (%)	Time Spent(ms)	Avg. Click Time (ms)
250	500	0	0,00	341.429	638
	500	0	0,00	761.915	1.524
	500	0	0,00	140.106	280
500	991	0	0,00	1.858.219	1.875
	972	1	0,10	3.383.689	3.485
	1000	0	0,00	756.178	756
750	1.499	6	0,40	9.559.512	6.403
	1.497	11	0,73	10.924.009	7.351
	1.499	6	0,40	6.608.113	4.426
1000	1.997	58	2,90	20.187.858	10.411
	1.998	87	4,35	20.777.446	10.873
	1.999	46	2,30	13.101.882	6.709
1250	2.499	200	8,00	31.192.375	13.568
	2.498	443	17,73	32.227.057	15.682
	2.497	259	10,37	19.261.739	8.607

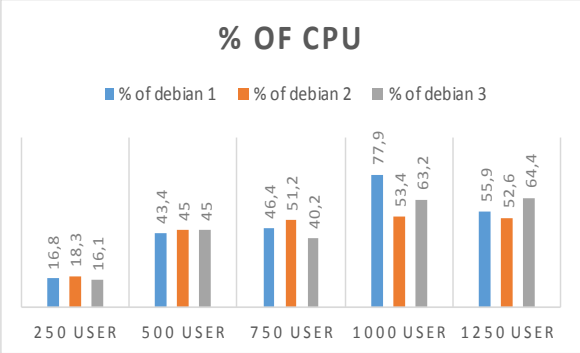
Dari 5 pengujian jumlah *user* yang telah dilakukan, dibuatlah 1 tabel secara ringkasnya yaitu pada tabel 1 yang memperlihatkan bahwa waktu rata-rata klik yang didapat dipengaruhi oleh jumlah *user* dengan bukti pada 1250 *user* memiliki waktu (*time spent*) yang lebih banyak dari jumlah *user* yang dibawahnya. Namun pada hasil *clicks* yang dilayani yaitu pada jumlah 250 *user* teratasi semua dan tidak terdapat eror. Pada 500 *user* hanya 1 yang terlayani 1000 *clicks*, yang 2 hanya 991 & 972 clicks dengan 1 eror. Karena seharusnya menghasilkan 1000 klik di tiap. Pada 750 *user* yang seharusnya menghasilkan 1500 klik per klien, pada hasil yang didapat hanya 1.499, 1.497, 1.499 klik dengan jumlah eror 3 *client* 23. Pada 1000 user yang dihasilkan seharusnya 2000 per klien, tapi hanya menghasilkan 1.997, 1.998, & 1.999 dengan jumlah eror 191. Dan 1250 user menghasilkan 2.499, 2.498, & 2.497 dengan eror yang sangat melonjak yaitu 902. Yang seharusnya menghasilkan 2500 per klien.

2. Pengujian dari kinerja RAM web server yang digunakan

Tabel 2. Pengujian *system monitor* debian

Jumlah User	Nama server	% of CPU	% of Memory
250	Debian 1	16,8	75,3
	Debian 2	18,3	81,7
	Debian 3	16,1	79,4
500	Debian 1	43,4	74,8
	Debian 2	45,0	77,0
	Debian 3	45,0	77,0
750	Debian 1	46,4	76,1
	Debian 2	51,2	75,1
	Debian 3	40,2	76,5
1000	Debian 1	77,9	74,3
	Debian 2	53,4	74,0
	Debian 3	63,2	71,3
1250	Debian 1	55,9	74,0
	Debian 2	52,6	67,5
	Debian 3	64,4	71,7

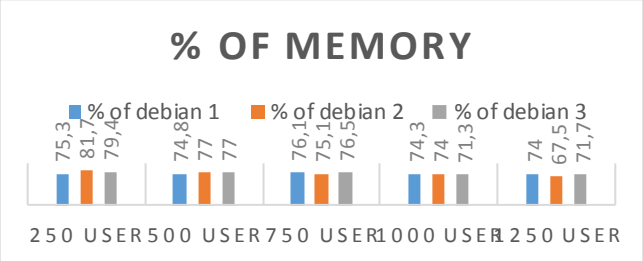
Hasil dari pengujian % of CPU yang telah dilakukan dan di ringkaslah berupa tabel agar lebih mudah membacanya. Pembagian beban pada 3 debian yang tersedia dilakukan oleh haproxy selaku *load balancing* secara fungsional. Pada pengujian 250 *user*, 3 server menangani beban yang hampir setara karena hasil nya 16-18%. Di 500 *user* menghasilkan 43-45%. Di 750 tidak terlalu melonjak jauh karena masih antara 40-51%. Di 1000 *user* memiliki ketidak seimbangan beban paling menonjol, debian 1 melonjak hingga 77,9 % & selisih 24,5% dengan debian 2, untuk debian 2 & 3 selisih 10%. Debian 2 nya terendah antara 53,4%. Dan di 1250 *user* masih mendekati setara pada 3 debian, karena yang dihasilkan debian 1 55,9% & debian 2 52,6% masih setara, debian 3 64,4% hanya selisih 10% dengan debian 1 & selisih 12% dengan 2.



Gambar 4. CPU dari 5 hasil pengujian *system monitor* debian dengan tipe tes *ramp*

Grafik hasil dari tabel 2 dan terlihat bahwa kinerja CPU dalam proses pengujian ini dari 250 *user* & 500 *user* masih seimbang dalam pembagian bebaannya. Namun untuk 750 *user* & 1250 *user* masih terlihat kurang seimbang pada grafik, meskipun demikian tidak terlalu jauh juga selisihnya. Dan terlihat dari grafik bahwa pada 1000 *user* terlihat sekali tidak seimbang pembagian bebannya. Debian 1 yang terlalu melonjak hingga selisih 24,5% dengan debian 2 & selisih 14,7% dengan debian 3.

Untuk keseluruhan hasil keseimbangan beban dari CPU masih bisa disebut seimbang karena hasilnya juga tidak terlalu banyak yang melonjak, hanya pada 1000 *user* saja terlihat melonjaknya terlalu tinggi.



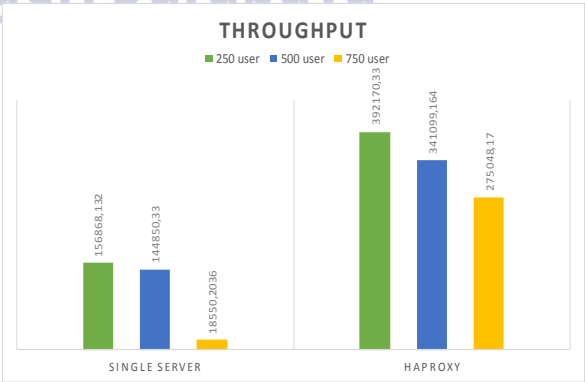
Gambar 5. *Memory* dari 5 hasil pengujian *system monitor* debian dengan tipe tes *ramp*

Grafik hasil dari tabel 2 dan terlihat bahwa kinerja *memory* dalam proses pengujian ini dari 250 *user* sudah mulai naik turun, namun masih antara 75-81,7% dengan selisih tertinggi & terendahnya 6,4%. Di 500 *user* masih seimbang dalam penggunaan bebaannya antara 74-77%. Di 750 *user* juga masih seimbang antara 75-76,5%. Di 1000 *user* juga seimbang antara 71-74,3% penggunaan *memory* & 1250 *user* naik turun lagi hingga selisih tertinggi & terendahnya 6,5%.

Untuk keseluruhan hasil keseimbangan penggunaan *memory* masih bisa disebut seimbang karena hasilnya juga tidak terlalu banyak yang melonjak, bahkan hasil seluruhnya kisaran 71,3% ke atas hingga 81,7%. Hanya debian 2 pada 1250 user bernilai 67,5%.

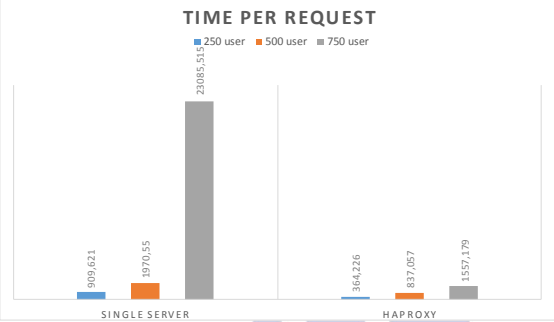
3. Pengujian menggunakan *apache bench*

Apache bench digunakan dalam pengujian ini untuk membantu menghitung *throughput* dan *time per request*. Pada apache bench, *time per request* sudah ditampilkan tetapi *throughput* harus dihitung manual menggunakan data yang telah dihasilkan oleh apache bench tersebut. Karena *throughput* merupakan hasil bagi dari file yang dikirim dibagi dengan waktu yang dibutuhkan, maka menghitung *throughput* menggunakan *total transferred* dibagi dengan *time taken for tes* yang sudah ditampilkan nilainya.



Gambar 6. Grafik membandingkan hasil *throughput single server* dan *haproxy*

Grafik *throughput* hasil perhitungan manual melalui *apache bench*, dari gambar grafik tersebut dapat dijelaskan bahwa menggunakan haproxy berpengaruh pada besarnya *throughput* yang dihasilkan. Hasil yang didapat dengan jumlah *user* yang sama adalah *throughput* yang menggunakan *haproxy* lebih besar dari yang hanya *single server*. Ketika throughput makin besar berarti pelayanan pada *request* akan lebih cepat.



Gambar 7. Grafik membandingkan hasil *time per request single server* dan *haproxy*

Dan pada gambar grafik 5 yaitu grafik *time per request* yang nilainya didapat dari hasil *apache bench*, memperlihatkan bahwa yang menggunakan haproxy lebih cepat dalam melayani *request* dari pada yang *single server*. Waktu yang didapat pada *single server* dalam melayani *request* makin besar berarti membutuhkan waktu yang lebih untuk menangani *request* dengan jumlah *user* yang sama dari pada yang menggunakan haproxy.

PENUTUP

Simpulan

Berikut kesimpulan yang didapat:

1. Secara fungsional, implementasi *load balancing* menggunakan haproxy telah berjalan dengan baik. Proses implementasi yang dilakukan mulai dari membuat topologi menggunakan GNS3 dengan OS debian 7, memasang haproxy versi 1.5.8 pada debian 7 lalu mengedit *file* konfigurasi yang dibutuhkan, terdapat 3 server dengan OS debian 7. Untuk pemasangan haproxy di debian 7 membutuhkan *package* tambahan melaui *backport* karena pada repositori tidak tersedia. Lalu mengubah *file host* di semua OS untuk menguji haproxy dengan membuka web melalui browser menggunakan *url*. Dan pengujian mematikan salah satu server lalu membuka laporan statistik yang dihasilkan haproxy untuk melihat tersambungnya haproxy dengan server yang telah dideklarasikan ke konfigurasi haproxy. Dengan begitu implementasi haproxy telah dinyatakan berjalan dan berfungsi secara baik.
2. Load balancing menggunakan pengujian tambahan dengan *apache bench* yang sudah terpasang di web server apache dengan sendirinya karena sudah bawaan saat memasang apache. Dengan mengambil data *throughput* dan *time per request* yang dihasilkan oleh ab, dapat membandingkan antara *single server* dan haproxy bahwa nilai *throughput* menggunakan haproxy lebih besar dari *single server* yang berarti pelayanan haproxy lebih cepat dari *single server* dan nilai *time per request* haproxy juga lebih sedikit dari *single server* yang berarti respon *request* lebih cepat karena membutuhkan sedikit waktu dalam melayani jumlah *request* yang sama. Itu lah yang membuat *load balancing* dapat menjadi solusi cepat dalam menangai *request* yang berlebih pada web server. Dan didukung dengan 3 pengujian yaitu pemerataan beban yang memakai bantuan *website* sederhana agar terlihat pembagiannya, mensimulasi jumlah beban *user* menggunakan *webserver stress tools*, hingga melihat pengaruh CPU & *memory* dengan beban *user* yang sama. Dari ketiga skenario pengujian yang telah dilakuan, dapat diambil kesimpulan yaitu mensimulasikan jumlah beban *user* dengan menunjukkan rata-rata klik yang didapat pada jumlah 1250 *user* memiliki waktu 15.682 (ms) lebih lama dari jumlah *user* 250, 500, 750, dan 1000. Berarti semakin besar beban *user* yang mengakses maka semakin besar pula kinerja web server yang harus digunakan. Agar sistem tetep berjalan lancar tanpa pelayanan yang melambat, maka penggunan teknik *load balancing* menggunakan haproxy dapat membantu dalam mengatasinya. Terlihat pada grafik % of CPU, pembagian bebannya seimbang, hanya pada 1000 user saja ada lonjakan hingga 77,9% di debian 1. Untuk keseluruhannya terbilang seimbang.

Saran

Untuk mengembangkan tugas akhir ini terdapat beberapa hal yang perlu diteliti lebih lanjut. Berikut beberapa halnya:

1. *Website* yang akan digunakan bisa lebih dikembangkan ke *https*.
2. Menambahkan lagi load balancing untuk *backup*.
3. Bisa menambahkan *database* lalu dikembangkan hingga mensinkronisasi *database*.

4. Memperhatikan cara kerja algoritma *round robin* dalam teknik *load balancing* menggunakan haproxy.

DAFTAR PUSTAKA

AG, P. (2017, Maret 20). *Webserver Stress Tool*. Diambil kembali dari Paessler The Network Monitoring Company: http://www.paessler.com/tools/webstress

Belajar HAProxy dan Konsep Load Balancing Untuk Pemula. (2017, september 13). Diambil kembali dari tutorlinux.com: https://tutorlinux.com/belajar-haproxy-dan-konsep-load-balancing-untuk-pemula.html

Dewannanta, D. (2013, Maret 23). *GNS3, Simulator Jaringan Komputer*. Diambil kembali dari ilmu komputer: http://ilmukomputer.org/2013/01/29.gns3/

Ellingwood, J. (2013, Maret 23). *How To Configure The Apache Web Server On Ubuntu Or Debian VPS*. Diambil kembali dari Digital Ocean: http://www.digitalocean.com/community/tutorials/how-to-configure-the-apache-web-server-on-an-ubuntu-or-debian-vps

Haproxy. (2017, Februari). Diambil kembali dari Haproxy: http://www.haproxy.org/#desc

Jenkov, J. (2014, Oktober 24). *Load Balancing*. Diambil kembali dari Tutorial Jenkov: http://tutorial.jenkov.com/software-architecture/load-balancing.htlm

Lakhe, S., Shinde, A., Sukhthankar, N., & Reddy, C. (2016). Serverload Balancing Using Haproxy. *www.ejurnal.aessangli.in/ASEEJournals/CE166.pdf*, 1-8.

Noviyanto, A. B., N, E. K., & Hamzah, A. (2015). Perencanaan Dan Implementasi Load Balancing Reverse Proxy Menggunakan Haproxy Pada Aplikasi Web. *Jurnal JARKOM Vol.3 No.1 Desember 2015*, 1-11.

PT. Web Media Technology Indonesia. (2017, agustus 7). *website lambat atau lemot ketika diakses (overload)*. Diambil kembali dari niagahoster: https://www.niagahoster.co.id/kb/website-lambat-atau-lemot-ketika-diakses-overload

The Apache Software Foundation (ASF). (2017, agustus 30). *The Apache Software Foundation*. Diambil kembali dari ab - Apache HTTP server benchmarking tool: https://httpd.apache.org/docs/2.4/programs/ab.html

Towidjojo, R. (2013). *Mikrotik Kungfu Kitab 2*. Jakarta: Jasakom.

Tutorials Point Originated. (2017). Diambil kembali dari Tutorialspoint Web Server: http://www.tutorialspoint.com/internet_technologies/web_servers.htm

VirtualBox. (2017, April). Diambil kembali dari VirtualBox: https://www.virtualbox.org/