Analisis Performansi Metode *Round robin* dan *Least connection* Pada Openstack *Load Balance As A Service*(LBaaS)

Rosyaid[1)], Irawan[2)], Dharma Aryani[3)]
[1]Teknik Elektro, Politeknik Negeri Ujung Pandang
rosyaid.chai@gmail.com
[2]Teknik Elektro, Politeknik Negeri Ujung Pandang
irawan@poliupg.ac.id
[3]Teknik Elektro, Politeknik Negeri Ujung Pandang
dharma.aryani@poliupg.ac.id

Abstract

Openstack adalah sebuah *platform cloud computing* berbasis *open source* untuk menyediakan *cloud* IaaS (Infrastructure as a *Service*). Di era sekarang kebutuhan akan mengakses layanan informasi yang disediakan oleh *server* semakin banyak sehingga hal ini mengakibatkan server menjadi *over load*.Untuk mengatasi terjadinya *over load* pada sebuah *web server* dibutuhkan *load balancer*. *Load balancer as a service* merupakan salah satu layanan dari openStack yang memanfaatkan HAProxy. Metode *load balancing* yang digunakan pada openstack yaitu metode *round robin* dan metode *least connection* . Implementasi *load balancer as a service* pada openstack newton dengan 2 node yaitu *controller* dan *compute*.Skenario pengujian untuk melakukan analisis performansi terdapat 3 skenario. Pengujian *load balancer* menggunakan *tools* httperf dengan 10 kali percobaan tiap skenario untuk kedua metode *load balancing*. Ada 3 parameter untuk menguji perfomansi dari *load balancer* yaitu CPU *Usage*, *Throughput* dan *Response time*. Rata penggunaan CPU pada metode *least connection* pada skenario 1 sebesar 60,72% sedangkan pada metode *round robin* pada skenario 1 sebesar 61,55%. Pada *response time* pada metode *least connection* rata-rata hasil yang didapatkan pada skenario 1 sebesar 2,07 ms sedangkan pada metode *round robin* sebesar 2,17%.Berdasarkan hasil pengujian metode *least connection* lebih unggul dibandingkan metode *Round robin* pada parameter CPU *Usage*, *Throughput*, *Response time*.

Keywords : Openstack, *Load balancer*, Httperf, *Round robin*, *Least connection*.

I. PENDAHULUAN

Seiring dengan berkembangnya teknologi informasi yang semakin pesat, penggunaan *web* untuk mengakses berbagai informasi semakin banyak digunakan karena dengan mudah pengguna mendapatkan informasi yang dibutuhkan dalam waktu yang relatif singkat. Hal ini menyebabkan peningkatan permintaan pada *web server* penyedia layanan informasi oleh *client*. Sehingga mengakibatkan *web server* menjadi *overload*, lambat dan akhirnya *web server* menjadi down. Solusi pada permasalahan ini dapat diatasi dengan menggunakan *load balancing*.

Load balancing adalah suatu teknik mendistribusikan beban kerja (*workload*) secara merata antara dua atau lebih komputer, link jaringan, CPU (Central Processing Unit), hard drive, atau sumber daya lain, dalam rangka untuk mendapatkan pemanfaatan sumber daya yang optimal, memaksimalkan *Throughput*, meminimalkan waktu tanggap, dan menghindari *overload* [1].

Openstack merupakan sistem operasi *cloud* yang berbasis *open source* yang dapat bersifat *public* maupun *private*. Openstack berfungsi untuk mengontrol lingkungan komputasi yang besar seperti *CPU*, *Memory*, *Storage*, *Network* di seluruh datacenter dan semuanya dikelola melalui *dashboard* yang memberikan kontrol ke *administrator* sekaligus pengguna ke sumberdaya penyediaan antarmuka *web* [2]. *Load balancer as a service* (LBaaS) merupakan *service* yang disediakan oleh openstack sebagai balancer (penyeimbang) untuk protokol http pada *instance* (VM).

Pada penelitian sebelumnya tentang *Load balancer as a service*(LBaaS) pada openstack menggunakan metode round robin membandingkan dengan *server cloud* tanpa *load balancing* mempunyai hasil kesimpulan bahwa dengan menggunakan metode *Round robin* dapat memaksimalkan kinerja *server cloud* daripada tanpa *load balancing* [3]. Selain metode *Round robin* pada *load balancing* metode *least connection* merupakan metode yang sering digunakan pada *load balancing* dimana pada penelitian yang membandingkan performa kedua metode *round robin* dan *least connection* pada *Software Defined Network* (SDN) mempunyai kesimpulan bahwa Performa algoritma *Round robin* pada pengujian *Throughput* lebih unggul dibandingkan algoritma *least connection* pada koneksi yang kecil. Sedangkan algoritma *least connection* lebih unggul pada koneksi yang besar. Pada pengujian *Response time* menujukan keunggulan algoritma *Round robin* dibandingkan dengan algoritma *least connection* [4].

Tujuan penelitian ini adalah mengimplementasikan LBaaS pada openstack dan mengalisis performansi *load balancing* dengan menggunakan metode *round robin* dan *least connection*. Kontribusi dalam penelitian ini untuk memberikan gambaran mengenai sistem kerja *Load balancer* pada openstack dan mengetahui perbandingan metode *Round robin* dan *least connection* pada LBaaS Openstack.

II. KAJIAN LITERATUR

A. Openstack

Openstack adalah sebuah *platform cloud computing* berbasis *open source* untuk menyediakan *cloud* IaaS (Infrastructure as a *Service*),baik untuk pribadi maupun perusahan berupa sumber daya untuk komputasi dan penyimpanan data dalam bentuk mesin *virtual* [5].

Openstack tersusun dari beberapa komponen. Adapun komponen-komponen tersebut adalah sebagai berikut:

1. Nova (*Compute Service*). Semua kegiatan yang diperlukan untuk mendukung siklus hidup dari *instance* dalam OpenStack *cloud* yang ditangani oleh Nova. Hal ini membuat Nova sebagai *Platform* Manajemen yang mengelola sumber daya komputasi, jaringan, otorisasi, dan kebutuhan skalabilitas dari OpenStack *cloud*.
2. Glance (*Image Service*) OpenStack Imaging *Service* adalah salah satu produk dari OpenStack yang digunakan untuk layanan virtual disk images.
3. Keystone (*Identity Service*) menyediakan layanan identitas dan akses kebijakan untuk semua komponen dalam keluarga OpenStack. Keystone menerapkan itu di REST-nya sendiri yang berbasis API (Identity API). Keystone menyediakan otentikasi dan otorisasi untuk semua komponen OpenStack. Otorisasi akan memverifikasi apakah pengguna yang terotentikasi memiliki akses ke layanannya yang dia minta atau tidak.
4. Neutron (*Networking Service*) adalah salah satu komponen openstack yang menyediakan layanan *cloud* Network as A *Service*. Neutron menyediakan API yang memungkinkan Anda menentukan konektivitas jaringan dan menangani di awan.
5. Keystone (*Identity Service*) menyediakan layanan identitas dan akses kebijakan untuk semua komponen dalam keluarga OpenStack. Keystone menerapkan itu di REST-nya sendiri yang berbasis API (Identity API). Keystone menyediakan otentikasi dan otorisasi untuk semua komponen OpenStack. Otorisasi akan memverifikasi apakah pengguna yang terotentikasi memiliki akses ke layanannya yang dia minta atau tidak.
6. Cinder (*Block Storage Service*) adalah komponen penyimpanan blok yang lebih analog dengan gagasan tradisional komputer yang dapat mengakses lokasi tertentu pada disk drive serta menyediakan perangkat penyimpanan untuk digunakan dengan *instance*s pada OpenStack.
7. Horizon (*User Interface Service*) merupakan suatu layanan user interface dalam infrastruktur Openstack yang memberikan akses visusalisasi bagi user dalam menciptakan *cloud*.

B. *Load balancing*

Load balancing merupakan suatu kemampuan untuk mengurangi beban dari proses untuk sebuah aplikasi kepada beberapa sistem yang berbeda sehingga untuk meningkatkan kemampuan pemrosesan pada permintaan yang datang. *Load balancing* akan mengirimkan beberapa pemrosesan dari permintaan ke sebuah sistem kepada sistem lainnya yang akan ditangani secara bersamaan [6].

Load balancing mempunyai beberapa metode yang sering digunakan dalam penerepannya. Berikut ini metode *load balancing* :

1. *Round robin*

   Algoritma *Round robin* merupakan algoritma yang paling sederhana dan paling banyak digunakan oleh perangkat *load balancing*. Algoritma *Round robin* bekerja dengan cara membagi beban secara bergiliran dan berurutan dari satu *server* ke *server* lainnya. Konsep dasar dari algoritma *Round robin* ini adalah dengan menggunakan *time sharing*, pada intinya algoritma ini memproses antrian secara bergiliran [7].
2. *Least connection*.

   Algoritma *Least connection* melakukan pembagian beban berdasarkan banyaknya koneksi yang sedang dilayani oleh sebuah *server*. *Server* dengan koneksi yang paling sedikit akan diberikan beban berikutnya, begitu pula *server* dengan koneksi banyak akan dialihkan bebannya ke *server* lain yang bebannya lebih rendah [7].

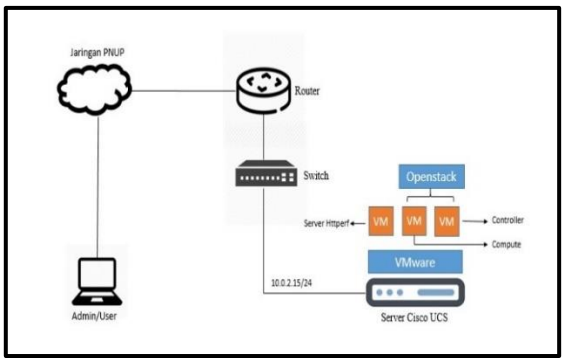
C. *Load balancer* As a Servcie (LBaaS)

Load balancer as a service merupakan salah satu layanan dari openStack yang memanfaatkan HAProxy. HAProxy merupakan salah satu *open soure* yang menyediakan *Load balancer* berupa *software-based* dan merupakan mesin penyeimbang beban bawaan (*balancer*) yang ada di openStack. Penyeimbang beban berbasis HAProxy ini memiliki akses jaringan ke *client* dengan mengirim dan menerima *request* pada neutron menggunakan alamat IP yang disebut VIP (*Virtual* IP). *Load balancer* menempati *port* jaringan neutron dan memiliki alamat IP yang ditetapkan dari *subne*t. *Listener* merupakan *Load balancer*s yang dapat berfungsi mendengarkan permintaan pada beberapa port contohnya *port* 80 untuk HTTP dan *port* 443 untuk HTTPS [8].

III. METODOLOGI PENELITIAN

A. Perancangan Sistem

Tahapan ini merupakan tahapan untuk membangun sistem yang digunakan dalam penelitian ini. Berikut ini adalah topologi yang digunakan sebagai berikut:

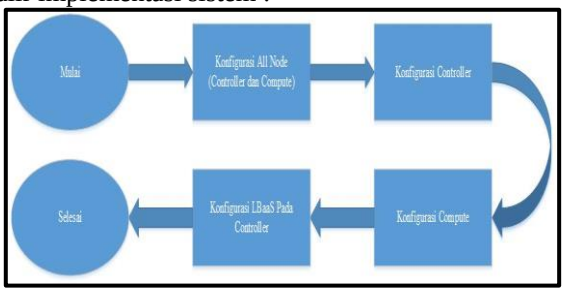


Gambar 2 Topologi Openstack LBaaS

Pada Gambar 2 merupakan topologi yang digunakan dalam membangun sistem *cloud computing* pada penelitian ini. Dalam infrastruktur *cloud* yang dibangun terdiri dari 2(dua) node yaitu *controller* node dan *compute* node yang dibangun di Vmware vcenter pada *server* cisco UCS. Sedangkan untuk pengujian pada penelitian ini menggunakan *server* Httperf yang dibangun juga pada *server* UCS.

B. Implementasi

Pada tahap ini akan dijelaskan implementasi sistem yang akan dibuat. Berikut ini adalah diagram alir implementasi sistem :



Gambar 3 Diagram Alir Implementasi Sistem

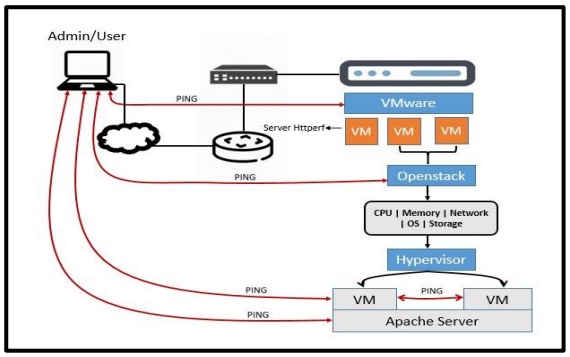
1. Konfigurasi Pada Semua Node
   Konfirgurasi yang dilakukan pada semua node adalah melakukan konfigurasi ip address yaitu 10.2.15.111/24 untuk *controller* node dan 10.2.15.112/24 untuk *compute* node dan instalasi NTP (Chrony), repository openstack dan aplikasi openstack.
2. Konfigurasi Pada *Controller* Node
   Instalasi dan konfigurasi yang dilakukan pada *controller* node yaitu MariaDB, RabbitMQ, Memcached, Keystone, Glance, Nova, Neutron dan Horizon,
3. Konfigurasi Pada *Compute* Node
   Instalasi dan konfigurasi yang dilakukan pada *compute* node yaitu Nova-*compute*, KVM Hypervisor dan Neutron.
4. Konfigurasi LBaaS pada Neutron di *controller* node.

C. Pengujian dan Analisis

Tujuan pengujian ini adalah untuk mengetahui perbandingan performa dari metode *Round robin* dan *least connection* untuk *load balancing* pada sistem *cloud* Openstack. Metode yang digunakan adalah metode Komparatif yaitu penelitian yang membandingkan dan menganalisa dua gejala atau lebih, Pengujian *Load balancing* dengan kedua metode menggunakan software yang bernama Httperf. Httperf menampilkan nilai sesuai parameter. Parameter yang digunakan adalah sebagai berikut :

1. CPU *Usage*
   Parameter CPU *Usage* digunakan untuk mengetahui sumber daya yang diperlukan dalam komputerisasi. Pada parameter CPU *Usage* dilakukan pada sisi *client*. Nilai yang akan diukur merupakan nilai dari kinerja CPU *load balancing server*. CPU *Usage* dihitung dalam bentuk satuan persen (%). Sehingga semakin kecil nilai persentasenya, maka akan semakin sedikit source yang digunakan dalam proses *load balancing*.
2. *Throughput*
   Parameter *Throughput* mengukur berapa besar bandwidth yang di dapatkan pada saat mengakses *web server*. Parameter ini dihitung dalam satuan Kb/second. Semakin besar nilai yang didapatkan maka semakin baik kinerja dari web *service* tersebut.
3. *Response time*,
   Parameter *Response time* mengukur kecepatan *web server* dalam menanggapi request dari *client*. Parameter ini dihitung dengan satuan milisecond. Semakin kecil nilai dari parameter ini, maka semakin cepat sebuah *web server* dalam menanggapi request dari *client*.

Dalam melakukan pengujian melalui admin/user terdapat alur komunikasi dari satu perangkat ke perangkat yang lain. Berikut dibawah ini arsitektur *Load balancer as a service*(LBaaS) :



Gambar 4 Arsitektur Load Balance as a *Service* (LBaaS)

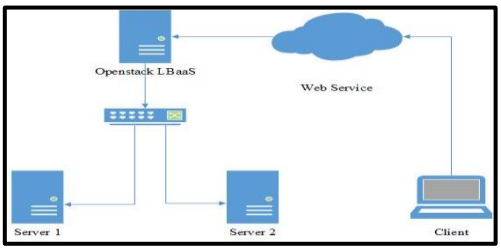
Pada gambar 4 dapat dilihat arsitektur *load balancing* dimana admin/user dapat berkomunikasi

ke semua mulai dari mesin Vmware , openstack , VM (*instance*) sampai ke apache *server* yang diinstall pada *instance*.

Dalam pengujian pada penelitian ini menggunakan tiga (3) skenario sebagai berikut :

1. Skenario 1

Pada skenario ini *load balancing* dilakukan pada 2 buah *instance* , berikut ini adalah topologi pada skenario 1:

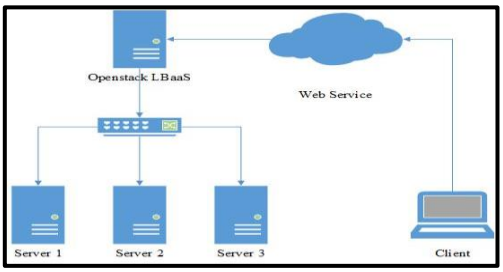


Gambar 5 Topologi skenario 1

Pada gambar 5 skenario pengujian *load balancing* yang dilakukan dimana *client* akan melakukan request ke *web server* yang akan dilayani oleh *server* 1 dan *server* 2 melalui *load balancing* as a *service* (LBaaS) melalui IP floating.

2. Skenario 2

Pada skenario ini *load balancing* dilakukan pada 3 buah *instance* , berikut ini adalah topologi pada skenario 2:

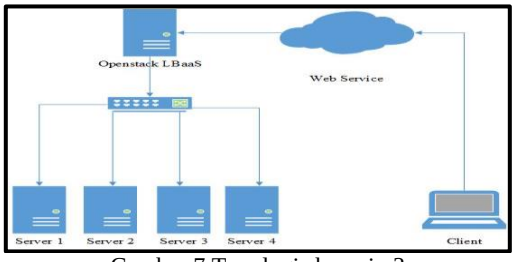


Gambar 6 Topologi skenario 2

Pada gambar 6 skenario pengujian *load balancing* yang dilakukan dimana *client* akan melakukan request ke *web server* yang akan dilayani oleh 3 *server* yaitu *server* 1, *server* 2 dan *server* 3 melalui *load balancing* as a *service* (LBaaS) melalui IP floating.

3. Skenario 3

Pada skenario ini *load balancing* dilakukan pada 4 buah *instance* , berikut ini adalah topologi pada skenario 3:



Gambar 7 Topologi skenario 3

Pada gambar 7 skenario pengujian *load balancing* yang dilakukan dimana *client* akan melakukan request ke *web server* yang akan dilayani oleh 4 *server* yaitu *server* 1, *server* 2, *server* 3 dan *server* 4 melalui *load balancing* as a *service* (LBaaS) melalui IP floating.

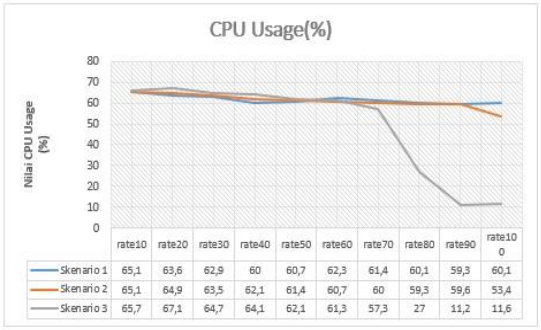
Untuk melakukan pengujian pada *Load balancer* digunakan *tools* httperf yang di install pada VM test, dimana fungsi dari *tools* httperf akan melakukan request ke *web server* sesuai dengan parameter yang diinginkan

Pengujian yang akan dilakukan untuk ketiga skenario pada metode *Round robin* dan *least connection* dengan menggunakan *tools* httperf, dimana setiap skenario untuk tiap metode dilakukan 10 kali pengujian dengan menambahkan request tiap detiknya mulai dari *rate* 10 sampai *rate* 100.

IV. HASIL DAN PEMBAHASAN

A. Hasil Pengujian Penggunaan CPU

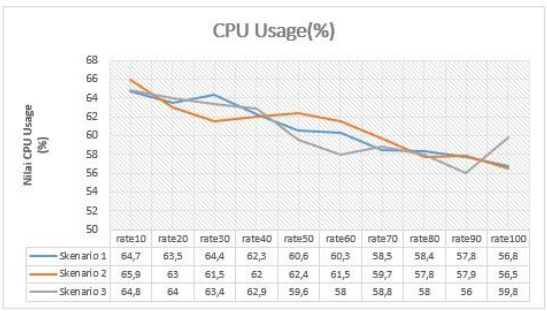
Dari hasil pengujian yang dilakukan pada 3 skenario yang telah ditentukan maka hasil yang didapatkan untuk parameter cpu *Usage* sebagai berikut :



	rate10	rate20	rate30	rate40	rate50	rate60	rate70	rate80	rate90	rate100
Skenario 1	65,1	63,6	62,9	60	60,7	62,3	61,4	60,1	59,3	60,1
Skenario 2	65,1	64,9	63,5	62,1	61,4	60,7	60	59,3	59,6	53,4
Skenario 3	65,7	67,1	64,7	64,1	62,1	61,3	57,3	27	11,2	11,6

Gambar 8 Grafik CPU *usage* dengan menggunakan metode *Round robin*

Pada pengujian ini dapat dilihat grafik CPU *Usage* pada Gambar 8 dengan menggunakan metode *Round robin* menunjukkan besarnya penggunaan daya pada CPU pada skenario 1 dan skenario 2 cenderung lebih stabil meskipun mengalami peningkatan permintaan user dari *rate* 10 sampai *rate* 100 bahkan penggunaan daya CPU cenderung menurun dan untuk skenario 3 pada penggunaan daya CPU stabil dari *rate* 10 sampai *rate* 60 akan tetapi dari *rate* 80 mengalami penurunan yang signifikan pada penggunaan daya CPU dari 57,3% ke 27% dan pada *rate* 90 juga *rate* 100 penggunaan daya turun menjadi rata-rata 11%



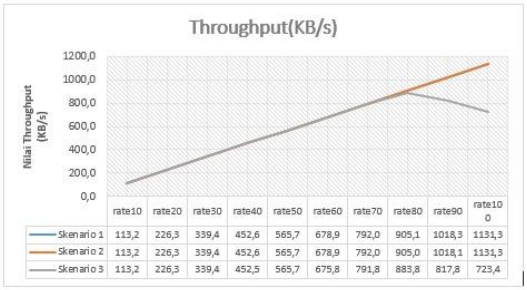
	rate10	rate20	rate30	rate40	rate50	rate60	rate70	rate80	rate90	rate100
Skenario 1	64,7	63,5	64,4	62,3	60,6	60,3	58,5	58,4	57,8	56,8
Skenario 2	65,9	63	61,5	62	62,4	61,5	59,7	57,8	57,9	56,5
Skenario 3	64,8	64	63,4	62,9	59,6	58	58,8	58	56	59,8

Gambar 9 Grafik CPU *usage* dengan menggunakan metode *least connection*

Pada gambar 9 grafik dengan menggunakan metode *least connection* dapat dilihat penggunaan daya CPU pada skenario 1 dan skenario 2 dari *rate* 10 sampai *rate* 100 cenderung lebih stabil dan mengalami penurunan penggunaan daya CPU sedangkan pada skenario 3 dari *rate* 10 sampai *rate* 90 mengalami penurunan dan pada *rate* 100 mengalami kenaikan dari 56% ke 59,8% hal ini di sebabkan karena request yang dikirim *client* ke *server* mengalami penumpukan antrian sehingga penggunaan daya CPU naik.

C. Hasil Pengujian *Throughput* (KB/s)

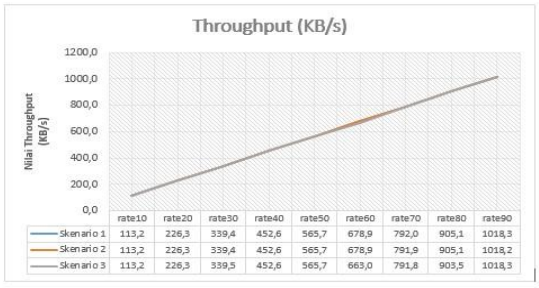
Pada pengujian *Throughput* yang dilakukan pada ketiga skenario yang telah ditentukan maka hasil yang di dapatkan dapat dilihat pada grafik dibawah ini :



	rate10	rate20	rate30	rate40	rate50	rate60	rate70	rate80	rate90	rate100
Skenario 1	113,2	226,3	339,4	452,6	565,7	678,9	792,0	905,1	1018,3	1131,3
Skenario 2	113,2	226,3	339,4	452,6	565,7	678,9	792,0	905,0	1018,1	1131,3
Skenario 3	113,2	226,3	339,4	452,5	565,7	675,8	791,8	883,8	817,8	723,4

Gambar 10 Grafik *throughput* dengan menggunakan metode *Round robin*

Pada pengujian ini dapat dilihat grafik *Throughput* dengan menggunakan metode *Round robin* pada skenario 1 dan skenario 2 mengalami peningkatan setiap *rate* nya sedangkan pada skenario 3 pada *rate* ke 80 sampai *rate* 100 mengalami penurunan.



	rate10	rate20	rate30	rate40	rate50	rate60	rate70	rate80	rate90
Skenario 1	113,2	226,3	339,4	452,6	565,7	678,9	792,0	905,1	1018,3
Skenario 2	113,2	226,3	339,4	452,6	565,7	678,9	791,9	905,1	1018,2
Skenario 3	113,2	226,3	339,5	452,6	565,7	663,0	791,8	903,5	1018,3

Gambar 11 Grafik *throughput* dengan menggunakan metode *least connection*

Pada pengujian ini dapat dilihat grafik *Throughput* dengan menggunakan metode *least connection* pada ketiga skenario mengalami peningkatan setiap *rate* nya dikarenakan user mengirimkan request semakin besar setiap *rate*nya sehingga *Throughput* yang dihasilkan semakin besar pula

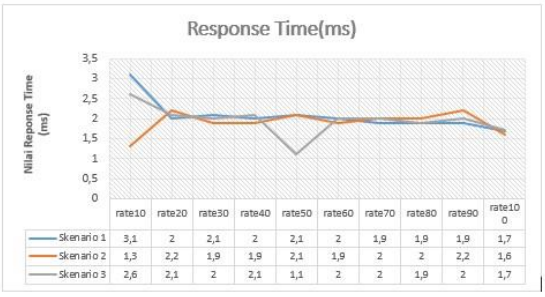
D. Hasil Pengujian *Response time* (ms)

Pada pengujian *Response time* yang dilakukan pada ketiga skenario yang telah ditentukan maka hasil yang di dapatkan dapat dilihat pada grafik dibawah ini :

	rate10	rate20	rate30	rate40	rate50	rate60	rate70	rate80	rate90	rate100
Skenario 1	2,2	2,3	2,1	2	2,7	1,9	2	2,1	2	2,4
Skenario 2	2	2,1	1,9	2,2	0,8	1,9	1,9	2,1	2,2	2,3
Skenario 3	1,9	1,9	1,7	1,9	1	2	1,9	2,2	2,2	2

Gambar 12 Grafik *response time* dengan menggunakan metode *Round robin*

Pada Gambar 12 grafik *Response time* dengan menggunakan metode *Round robin* pada dilihat pada ketiga skenario cenderung stabil pada setiap *rate*nya akan tetapi pada skenario 1 pada *rate* 40 ke *rate* 50 mengalami peningkatan *Response time* dan pada *rate* 60 stabil kembali hal ini disebabkan karena pada *rate* 50 *server* mengalami antrian request dari *client* ke *server* sehingga respone time menjadi tinggi akan tetapi pada skenario 2 dan skenario 3 pada *rate* 40 ke *rate* 50 *Response time* mengalami penurunan dan kembali stabil pada *rate* 60 sampai *rate* 100 hal ini di sebabkan antrian request dari *client* ke *server* lebih cepat di respone oleh *server* sehingga mengalami penururan pada *rate* 40 ke *rate* 50.



	rate10	rate20	rate30	rate40	rate50	rate60	rate70	rate80	rate90	rate100
Skenario 1	3,1	2	2,1	2	2,1	2	1,9	1,9	1,9	1,7
Skenario 2	1,3	2,2	1,9	1,9	2,1	1,9	2	2	2,2	1,6
Skenario 3	2,6	2,1	2	2,1	1,1	2	2	1,9	2	1,7

Gambar 13 Grafik *response time* dengan menggunakan metode *least connection*

Pada Gambar 13 grafik respone time dengan menggunakan *least connection* dapat dilihat pada skenario 1 pada *rate* 10 *Response time* sebesar 3,1ms dan *rate* 20 sampai *rate* 100 cenderung stabil dan mengalami penurunan *Response time* tingginya *Response time* pada *rate* 10 dikarenakan *server* belum stabil saat menerima request dari *client* hal ini juga terjadi pada skenario 3 hanya pada *rate* 10 yang *Response time* yang tinggi dan *rate* 20 sampai *rate* 40 cenderung stabil dan mengalami penurunan signifikan pada *rate* 50 dan *rate* 60 sampai 100 kembali stabil sedangkan pada skenario 2 pada *rate* 10 mempunyai *Response time* yang paling rendah dikarenakan request dari *client* ke *server* lebih sedikit dan *server* cepat merespons permintaan dari *client* dan *rate* 20 sampai *rate* 100 kembali stabil seperti skenario 2 dan skenario 3.

E. Hasil Perbandingan Metode *Round robin* dan *Least connection*

Dari hasil pengujian yang telah dilakukan maka metode *Round robin* dan *least connection* didapatkan perbadingan sebagai berikut :

1. Penggunaan daya CPU pada metode *least connection* lebih kecil dibandingkan metode *Round robin*, dimana rata-rata penggunaan daya CPU pada metode *least connection* pada skenario 1 sebesar 60,72% ,skenario 2 sebesar 60,82% dan skenario 60,53% sedangkan metode *Round robin* pada skenario 1 sebesar 61,55% , skenario 2 sebesar 61% dan skenario 3 sebesar 49%. Hanya pada skenario 3 metode *Round robin* lebih kecil penggunaan daya CPU dikarenakan penurunan signifikan yang didapatkan pada *rate* 80 sampai *rate* 100 pada saat pengujian.
2. Hasil *Throughput* yang didapatkan pada metode *Round robin* dan *least connection* mendapatkan hasil yang tidak jauh berbeda pada skenario 1 dan skenario 2 ,sedangkan pada skenario 3 metode *Round robin* mengalami penurunan pada *rate* 80 sampai 100 tetapi untuk metode *least connection* mengalami kenaikan seperti pada skenario 1 dan skenario 2 sehingga pada parameter *Throughput* metode *least connection* lebih unggul dibandingkan metode *Round robin* dikarenakan lebih stabil pada bandwith yang dihasilkan.
3. Kecepatan *server* dalam meresponse permintaan dari *client* menggunakan metode *least connection* lebih unggul di bandingkan metode *Round robin*, dilihat dari rata-rata nilai *Response time* dari metode *least connection* pada skenario 1 sebesar 2,07 ms , skenario 2 sebesar 19,1 ms dan skenario 3 sebesar 1,95 ms sedangkan metode *Round robin* mempunyai nilai rata-rata pada skenario 1 sebesar 2,17 ms, skenario 2 sebesar 1,94 ms dan skenario 3 sebesar 1,87 meskipun pada metode least conection hasil *Response time* yang didapatkan besar di *rate* 10 namun dari *rate* 20 sampai 100 mengalami penurunan secara stabil sedangkan pada metode *Round robin* cenderung tidak stabil hasil *Response time* yang didapatkan naik turun.

V. KESIMPULAN

Berdasarkan dari hasil pengujian dan analisis yang dilakukan pada penelitian ini didapatkan kesimpulan sebagai berikut :

1. *Load balancer as a service* (LBaaS) berhasil diimplementasikan dan berfungsi dengan baik pada Openstack *cloud*.
2. Metode *least connection* lebih unggul dibandingkan metode *Round robin* pada skenario 1 dan skenario 2 pada parameter CPU *Usage* ,*Response time* dan *Throughput*. Metode *Round robin* hanya unggul pada skenario 3 pada parameter CPU *Usage* dan *Response time* dikarenakan semakin banyak *server* pada metode ini maka beban kerja pada tiap *server* akan semakin rendah.

REFERENSI

[1] Ariyani Abdullah, S. N. M. P. S. H. R. A., 2010. Implementasi Dan Analisa Load-Balancing Pada Suatu Web-*Server* Lokal. Proceeding Seminar Ilmiah Nasional Kommit, Pp. 1 - 7.

[2] Tegar Winarso Ananda, D. I. R. M. A. M. S. M., 2016. Desain Dan Realisasi Sistem Grid Computing Pada Infrastructure As A *Service* (Iaas) Menggunakan *Cloud Platform* Openstack. E-Proceeding Of Engineering : Vol.3, Pp. 743-748.

[3] I Made Aga Satya Darma, I. G. O. G. A., 2017. Implementasi *Load balancing* Pada Openstack Dengan Metode *Round robin*. Prosiding Seminar Nasional Pendidikan Teknik Informatika, Pp. 115-119.

[4] Agung Nugroho, 2017. Analisis Perbandingan Performa Algoritma *Round robin* Dan *Least connection* Untuk *Load balancing* Pada Software Defined Network. Jurnal Pengembangan Teknologi Informasi Dan Ilmu Komputer Vol.1, Pp. 1568-1577.

[5] Muhammad Fauzan, A. F. F. E. M. A., 2017. Analisis Dan Perancangan Infrastruktur Private *Cloud* Dengan Openstack. Jurnal Pseudocode, Volume Iv Nomor 2,, Pp. 180-189.

[6] Ansharullah, K., 2016. Implementasi Sistem *Load balancing* Dengan Algoritma *Round robin* Untuk Mengatasi Beban*server* Di Smk Negeri 2 Kudus. Pp. 1-136.

[7] Ellrod, C., 2010. *Load balancing – Least connection*. [Online] Available At: Https://Www.Citrix.Com/Blogs/2010/09/02/Load-Balancing-Least-Connections/

[8] Reza Nuryati, I. A. T. H. M. ,. R. M. S. M., 2017. Perancangan Dan Analisis *Load balancing* As A *Service* Menggunakan Openstack Untuk Database Gunung Api. E-Proceeding Of Engineering : Vol.4, No.2, Pp. 1847-1854.