PERANCANGAN DAN ANALISIS LOAD BALANCING AS A SERVICE MENGGUNAKAN *OPENSTACK* UNTUK DATABASE GUNUNG API

Reza Nuryati[1], Ir. Ahmad Tri Hanuranto M.T. [2], Ratna Mayasari S.T., M.T.[3]

[1,2,3] Prodi S1 Teknik Telekomunikasi, Fakultas Teknik Elektro, Universitas Telkom
Jln. Telekomunikasi No.1 Terusan Buah Batu Bandung 40257 Indonesia
[1]rezanuryati97@gmail.com, [2]Athanuranto@gmail.com, [3]Ratnamayasari07@yahoo.com

ABSTRAK

Pembangunan infrastruktur jaringan pada sistem database untuk hasil pengamatan gunung api diperlukan karena Indonesia termasuk negara *ring of fire* yang memiliki 127 gunung api aktif maupun tidak. Pengamatan gunung api terbagi menjadi dua metode secara visual dan instrumentasi menggunakan sensor analog dan digital. Data hasil metode visual dan instrumentasi menggunakan sensor analog di *input* melalui web aplikasi. Data hasil metode intrumentasi sensor digital masuk secara otomatis kedalam *server* database. Semua data hasil pengamatan menggunakan dua metode tersebut masuk ke *server* database secara bersamaan sehingga menyebabkan *down* (tidak aktif) pada web aplikasi *input* data. Hal ini dapat menyebabkan kinerja *server* database kurang maksimal.

Mekanisme sistem antrian dengan menditribusikan beban trafik data ke *server* dengan teknologi *Load Balancing* dapat menjadi solusi. Pada penelitian ini dilakukan implementasi *Load Balancing* pada *web server* guna mendistribusikan trafik data hasil pengamatan yang masuk kedalam *server* database. Infrastruktur jaringan sistem database menggunakan *platform cloud computing* yaitu *OpenStack* dengan implementasi *Load Balancing as a Service*. Penelitian ini membandingkan dua algoritma *Load Balancing* yaitu *Round Robin* dan *Least Connection*.

Least Connection memiliki performansi yang lebih baik dari *Round Robin* untuk segi uji parameter *Respon Time*, *Throughput*, *Transaction Rate* dan *Failed Transaction* pada *web server*. Hasil pengujian algoritma *Least Connection* jika dibandingkan dengan algoritma *Round Robin* dan *Single Web Server* berdasarkan skenario pengujian memiliki nilai rata-rata *Respon Time* lebih kecil 8,49% dan 71,37%, *Throughput* lebih besar 5,07% dan 93,72%, *Transaction Rate* lebih besar 5,26% dan 50,16% serta untuk *Failed Transaction* 0%. Algoritma *Least Connection* yang diimplementasikan pada *web server* aplikasi *input* data hasil pengamatan gunung api dapat mendukung kinerja *server database* yang kurang maksimal. Kinerja *server database* yang maksimal dapat mempercepat proses keputusan status level gunung api sebelum terjadi bencana.

Kata kunci : Gunung Api, *Cloud Computing, OpenStack, Load Balancing as a Service*

ABSTRACT

Construction of network infrastructure in the database *system for the observation of volcano is necessary because Indonesia is a ring of fire country which has 127 volcanoes. Volcano observation is divided into two methods visually and instrumentation using analog and digital sensors. The visual and instrumentation result data uses analog sensors in input using web applications. The data result of intrumentation method of digital sensor entered automatically into* database *server. All the observed data using these two methods into the* database *server simultaneously causing down on the web application data input. It can make the performance of the* database *server less than the maximum.*

The mechanism of the queuing system by distributing the data traffic load to the server with Load Balancing technology can be a solution. In this research is done Load Balancing implementation on web server to distribute data traffic of observation result into database *server. Infrastructure* database *system network using cloud computing platform is OpenStack with Load Balancing as a Service implementation. This study compares two Load Balancing algorithms namely Round Robin and Least Connection. Least Connection has a better performance than Round Robin for the test aspects of Response Time, Throughput, Transaction Rate and Failed Transaction parameters on the web server. Least Connection algorithm test result compared to test scenario has average value of Response Time more smaller 8,49% and 71,37%, Throughput more*

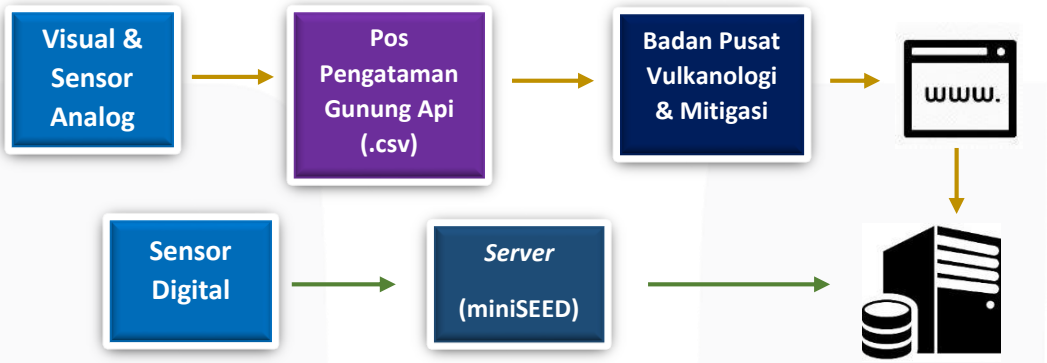
greater 5,07% and 93,72%, Transaction Rate more greater 5,26% and 50,16% and then for Failed Transaction 0%. Least Connection algorithm that is implemented on web server application data input of observation of volcano can support database *server performance less than maximum. The performance of volcano* database *server can speed up the process of deciding the status of the volcano level before a disaster occurs.*

Keywords: *Volcano, Cloud Computing, OpenStack, Load Balancing as a Service*

1. Pendahuluan

1.1 Latar Belakang Masalah

Bencana yang dapat sering muncul di Indonesia diantara nya adalah bencana gunung api karena indonesia memiliki 127 gunung api aktif maupun tidak. Proses pengamatan gunung api terbagi menjadi dua secara visual dan instrumentasi menggunakan sensor seperti pada Gambar 1.1. Pengamatan secara visual dilakukan oleh petugas pos pengamatan gunung api. Data hasil pengamatan berbentuk file dokumen dengan format .csv di input melalui sebuah aplikasi web oleh petugas ke *server* database gunung api. Pengamatan Gunung api secara instrumentasi terbagi menjadi dua yaitu menggunakan sensor analog dan digital. Data hasil pengamatan menggunakan sensor analog juga di input melalui aplikasi web input data dengan format data yang sama. Berbeda dengan sensor digital dimana data hasil pengamatan dengan format data *miniseed* diolah dan diubah secara otomatis oleh sistem dan disimpan kedalam *server* database [1].



Gambar 1.1 Alur Pemantauan Sistem Gunung Api

Namun pengamatan secara instrumentasi menggunakan sensor digital belum terealisasi pada semua gunung api, hal ini dikarenakan keterbatasan alat yang tersedia. Pengamatan fokus dilakukan secara visual dan menggunakan sensor analog maka ketika semua data tersebut masuk secara bersamaan kedalam *server* database saat proses pengamatan berlangsung dapat menyebabkan *down* (tidak aktif) pada aplikasi web *input* data yang digunakan. Hal ini akan mempengaruhi kinerja *server* database dalam memberikan keputusan status level gunung api. Oleh karena itu, untuk mengatasi masalah tersebut diperlukan suatu mekanisme yang dapat menimalisir kepadatan beban trafik yang terjadi saat petugas menginput data hasil pengataman. Implementasi mekanisme *Load Balancing* dengan teori sistem antrian dapat menjadi solusi dari masalah tersebut. Mekanisme *Load Balancing* bekerja dengan mendistribusikan beban trafik ke beberapa *server* sehingga dapat meringankan beban kerja *server*. Pada penelitian ini mekanisme Load Balancing diimplementasikan di *cloud computing*. Salah satu *platform cloud computing* yang menyediakan layanan *Load Balancing* adalah *OpenStack* yang disebut *Load Balancing as a Service*. *OpenStack* digunakan untuk membangun infrastruktur komputasi jaringan dan *Load Balancing as a Service* pada aplikasi web *input* data gunung api karena sifatnya yang *open source* sehingga lebih efisien dan performansi nya yang lebih baik dibandingkan *platform cloud computing* lainnya [2]. Pada penelitian tugas akhir ini akan dirancang sebuah mekanisme *Load Balancing* yang diimplementasikan didalam infrastruktur jaringan komputasi awan menggunakan *platform OpenStack*.

2. Dasar Teori

2.1 *Cloud Computing*

Definisi *cloud computing* atau komputasi awan secara umum merupakan gabungan dari pemanfaatan komputasi atau teknologi komputer dengan pengembangan berbasis awan sebutan lain untuk Internet yang biasanya di wujudkan dalam bentuk layanan atau aplikasi yang dapat di akses melalui internet. *Cloud computing* mempunyai 3 tingkatan layanan utama yang diberikan kepada pengguna, sebagai berikut [3].

2.1.1 *Software as a Service*

Layanan ini pada *cloud computing* hanya menyediakan *software* sebagai *service* kepada pengguna, dimana pengguna dapat langsung menggunakan *software* atau aplikasi yang telah disediakan oleh *provider cloud*. Contohnya seperti pada layanan email yaitu *gmail.com* kita sebagai pengguna tidak perlu memikirkan tentang keamanan dari aplikasi tersebut.

2.1.2 *Platform as a Service*

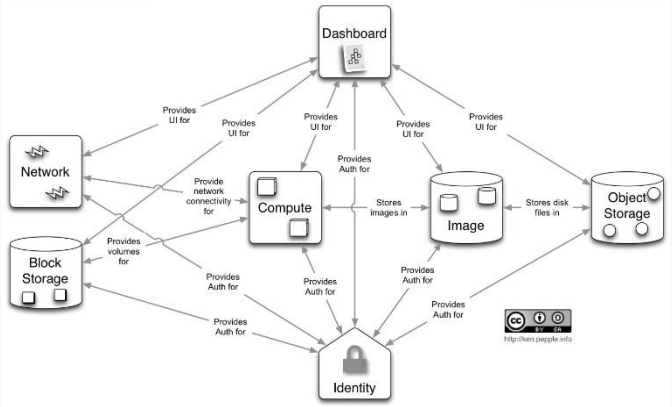
Layanan ini menyediakan *platform* untuk aplikasi yang pengguna jalankan pada *cloud computing*. Layanan ini dapat di analogikan dengan penyewaan sebuah "rumah" beserta lingkungan nya (sistem operasi, *network*, database *engine*, *framework aplikasi*, dll) untuk aplikasi yang akan di jalankan oleh pengguna.

2.1.3 *Infrastructure as a Service*

Layanan ini menyediakan infrastruktur jaringan teknologi informasi seperti komputasi, *storage*, *memory*, *network*, dll. Contoh penyedia layanan ini diantaranya adalah *Amazon EC2*, *TelkomCloud*, *BizNetCloud*.

2.2 *OpenStack*

OpenStack adalah sebuah *software open source* yang digunakan untuk pengembangan layanan *Infrastructure as a Service* pada *cloud computing*. *OpenStack* dirancang dengan tiga komponen layanan utama yaitu *networking* sebagai jaringan dari *cloud*, *compute* sebagai komputasi pada *cloud* dan *storage* sebagai media atau pun tempat penyimpanan pada *cloud* yang dapat digunakan sebagai media virtualisasi *server*. Pada perancangan ini digunakan *OpenStack* jenis Mitaka dengan instalasi secara *all in one*. Berikut adalah arsitektur serta komponen yang ada di dalam *OpenStack* secara umum.



Gambar 2.2 Arsitektur *OpenStack* [4]

Komponen pada *OpenStack* memiliki fungsi nya masing-masing, beberapa diantaranya sebagai berikut [4] :

2.2.1 *Networking* (Neutron)

OpenStack Networking yaitu Neutron merupakan sistem yang melakukan *provisioning* jaringan seperti mengatur jaringan atau subnet, *router*, *load balancing*, *gateway* dan atau *floating* IP yang melibatkan entitas pada *virtual machine*. Neutron juga berfungsi sebagai penyedia *Network as a Service* pada *cloud computing*.

2.2.2 *Compute* (Nova)

OpenStack Compute adalah otak dari *cloud* dan dapat mengelola jaringan dengan skala besar secara virtual. Nova merupakan bagian utama dari sistem IaaS yang memungkinkan

pengguna untuk membuat dan mengelola *server* secara virtual [4]. Nova juga dapat mengatur fungsi proses dan alokasi CPU untuk setiap virtual *machine*.

2.2.3 *Block Stroge* (Cinder)

Cinder menyediakan layanan penyimpanan blok untuk digunakan oleh *compute instance.* Cinder memungkinkan pengguna untuk mengatur kebutuhan media penyimpanan dan dapat digunakan untuk penyimpanan *database*, *expandable file system*, akses pada penyimpanan blok, *snapshot management.*

2.2.4 *Object Storage* (Swift)

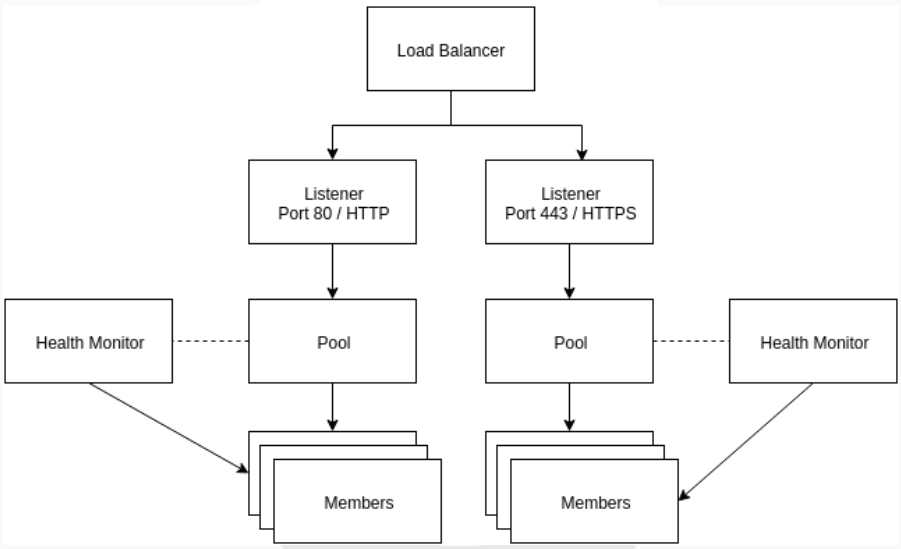
Swift termasuk kedalam jenis komponen *OpenStack Object stroge*. Swift menyediakan tempat penyimpanan untuk menyimpan objek, file atau data dalam jumlah besar. Swift dapat menyimpan dan menerima data yang sangat banyak dan dapat di-*scale* dengan mudah. Swift ideal untuk menyimpan data tidak terstruktur yang dapat berkembang tanpa batas.

2.2.5 *Dashboard* (Horizon)

Horizon merupakan tampilan *dashboard* untuk *OpenStack*. Horizon menyediakan antarmuka *web* untuk semua komponen dalam *OpenStack*.

2.3 *Load Balancing as a Service*

Load Balancing 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 subnet. *Listener* merupakan *Load balancers* yang dapat berfungsi mendengarkan permintaan pada beberapa port contohnya *port* 80 untuk HTTP dan *port* 443 untuk HTTPS.



Gambar 2.3 *Load Balancing as a Service*[5]

3. Pembahasan

3.1 Konfigurasi Sistem

Masing-masing *port* tersebut ditentukan oleh *Listener*. *Pool* sebagai manajemen daftar anggota yang menyajikan konten memalui penyeimbang beban. *Members* atau anggota adalah *server* yang melayani lalu lintas di belakang penyeimbang beban. Setiap anggota ditentukan oleh alamat IP dan port yang digunakannya untuk melayani lalu lintas. Anggota dapat mengalami offline dari waktu ke waktu dan *Health Monitor* mengalihkan lalu lintas dari anggota yang tidak merespons dengan benar. Health Monitor terhubung dengan *Pool*. Berikut adalah beberapa macam algoritma *Load Balancing as a Service* [5].

3.1.1 *Round Robin*

algoritma ini yang paling umum digunakan pada Load Balancing dengan cara mengatur permintaan dengan mekanisme memutar permintaan yang masuk ke dalam *server* secara merata dan atau berulang.

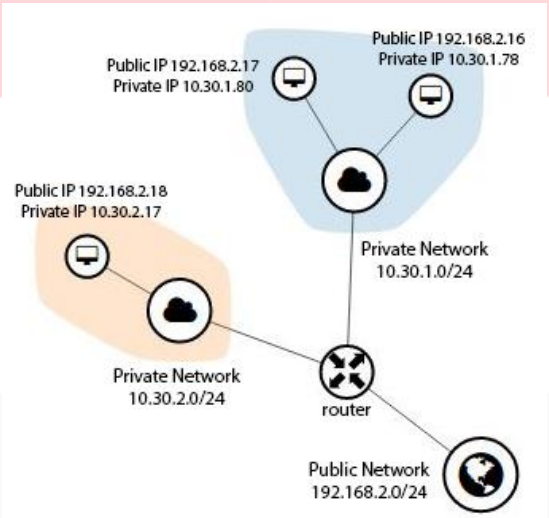
3.1.2 *Least Connection*

yaitu algoritma ini bekerja dengan cara mengalokasikan permintaan ke *server* yang memeiliki jumlah koneksi aktif yang lebih sedikit.

3.1.3 *Source IP*

Berbeda dengan algoritma lainnya, cara kerja algoritma ini adalah dengan cara mengatur permintaan dari alamat IP unik secara konsisten untuk diarahkan ke *server* yang sama.

Load Balancing as a Service diimplementasikan pada model perancangan topologi yang dibuat menggunakan *OpenStack* sebagai berikut.



Gambar 2.4 Topologi Sistem

3.2 Hasil dan Pembahasan

Data hasil pengujian diperoleh dengan mengirimkan request sebanyak 100, 300, 500, 700 dan 900 ke *web server* menggunakan *tools* untuk mengukur performansi *server* yaitu *siege*.

Tabel 3.1 Hasil Pengujian Skenario Single Web Server

Jumlah Request	Respon Time (Sec) (Second)	Throughput (Mb/Secs)	Transaction Rate (Trans/Secs)	Failed Transaction (Transaction)
100	0.099	0.501	1286.7	0.0
300	0.180	0.534	1317.2	1.2
500	0.316	0.556	1409.5	1.3
700	0.381	0.611	1505.0	1.6
900	0.453	0.933	1667.3	1.9

Tabel 3.2 Hasil Pengujian Skenario Algoritma Least Connection

Jumlah Request	Respon Time (Sec) (Second)	Throughput (Mb/Secs)	Transaction Rate (Trans/Secs)	Failed Transaction (Transaction)
100	0.060	0.763	1520.5	0.0
300	0.088	1.253	2249.9	0.0

500	0.174	1.270	2272.0	0.0
700	0.241	1.292	2343.2	0.0
900	0.304	1.326	2417.1	0.0

Tabel 3.2 Hasil Pengujian Skenario Algoritma Round Robin

Jumlah Request	**Respon Time (Sec)** (Second)	**Throughput** (Mb/Secs)	**Transaction Rate** (Trans/Secs)	**Failed Transaction** (Transaction)
100	0.067	0.759	1370.7	0.0
300	0.101	1.221	2187.9	0.0
500	0.180	1.244	2235.3	0.0
700	0.251	1.255	2252.2	0.0
900	0.330	1.262	2261.4	0.0

4. Analisis

4.1 Perbandingan Parameter *Respon Time Single Server* dan *Load Balancing*

Pada Gambar 3.1 tersebut implementasi *Load Balancing* pada *web server* dengan algoritma *Least Connection* memiliki nilai *respon time* lebih kecil 71.37% dari *single web server* dan 8.49% lebih kecil dari algoritma *Round Robin* rendah. Hal tersebut disebabkan oleh mekanisme algoritma *Least Connection* yaitu mendistribusikan beban trafik dengan memilih jalur yang memiliki kepadatan beban trafik yang lebih Hal ini menyebabkan nilai *respon time* dari *server* lebih kecil, karena tidak membutuhkan waktu yang lama pada *server* untuk menangani *request*. Sehingga dapat dikatakan bahwa implementasi algoritma *Least Connection* pada *web server* dari sisi parameter *Respon Time* lebih baik dibandingkan *single web server* dan algoritma *Round Robin* .

4.2 Perbandingan Parameter *Throughput Single Server* dan *Load Balancing*

Dapat dilihat pada Gambar 3.2 perbandingan dari hasil pengujian parameter *Throughput* dari skenario yang dilakukan. Implementasi *Load Balancing* pada web *server* menggunakan algoritma *Least Connection* memiliki persentase rata-rata *Throughput* yang lebih besar yaitu 93,72% jika dibandingkan dengan persentase *Single Server* dan lebih besar 2,63%. Hal ini disebabkan karena pada algoritma *Least Connection* digunakan dua *server* untuk menangani request yang masuk ke *server*. Oleh sebab itu, ketika jumlah request banyak cenderung konsumsi *bandwidth* pun meningkat sehingga menyebabkan nilai *Throughput* tinggi. Tetapi mengindikasikan bahwa *server* berhasil mentransfer banyak data ke *client*. Parameter *Throughput* ini juga berkaitan dengan parameter *Respon Time* dimana semakin kecil nilai *respon time* maka akan semakin besar nilai *throughput* yang dapat menyebabkan konsumsi *bandwidth* juga semakin meningkat.

4.3 Perbandingan Parameter *Transaction Rate Single Server* dan *Load Balancing*

Pada Gambar 3.3 dapat dilihat, *Load Balancing* dengan algoritma *Least Connection* memiliki persentase rata-rata *Transaction Rate* yang lebih besar yaitu 50,16% jika dibandingkan dengan persentase *Single Web server* dan jika dibandingkan dengan algoritma *Round Robin* 5,26%. Hal ini disebabkan karena request klien saat mengirimkan SYN pada *server* untuk proses pembangunan koneksi TCP IP dilakukan dengan dua *server*. Implementasi algoritma *Least Connection* dilakukan ketika klien akan melakukan request ke salah satu *server* maka jalur yang dipilih adalah jalur dengan kepadatan trafik data yang rendah. Hal tersebut mengakibatkan jumlah transaksi yang gagal berkurang dan kecepatan laju transaksi data yang berhasil direspon *server* semakin besar. Oleh karena itu, untuk parameter *Transaction Rate* semakin besar nilai *Transaction Rate* maka semakin cepat respon *server* dalam melayani *request.*

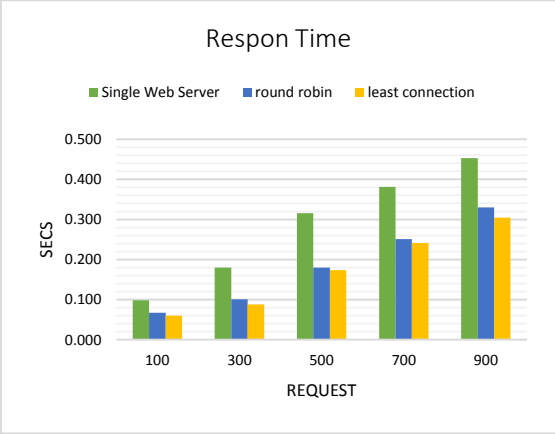
4.4 Perbandingan Parameter *Failed Transaction Single Server* dan *Load Balancing*

Pada pengujian parameter ini, terlihat bahwa implementasi *Load Balancing* pada *web server* tidak memiliki nilai *Failed Transaction*. Pada *Single Web server* terdapat nilai rata-rata *Failed Transaction* sebesar 1,2 trans selama 30 detik waktu simulasi pengujian dimana semakin banyak request yang diujikan maka semakin besar nilai *Failed Transaction* yang menandakan kinerja *server*

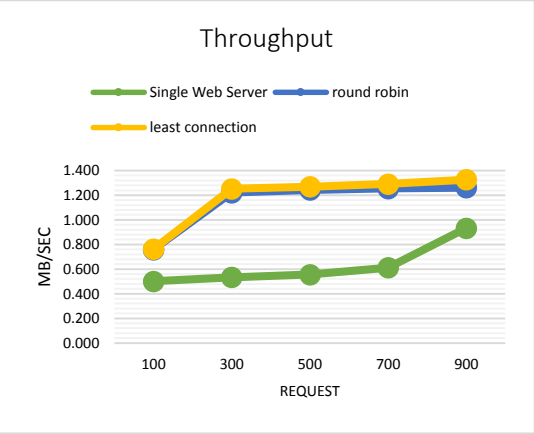
tidak maksimal. Hal ini dapat menyebabkan *request* yang gagal akibat tidak mendapat respon dari *server*.

4.5 Perbandingan Parameter CPU *Utilization Single Server* dan *Load Balancing*

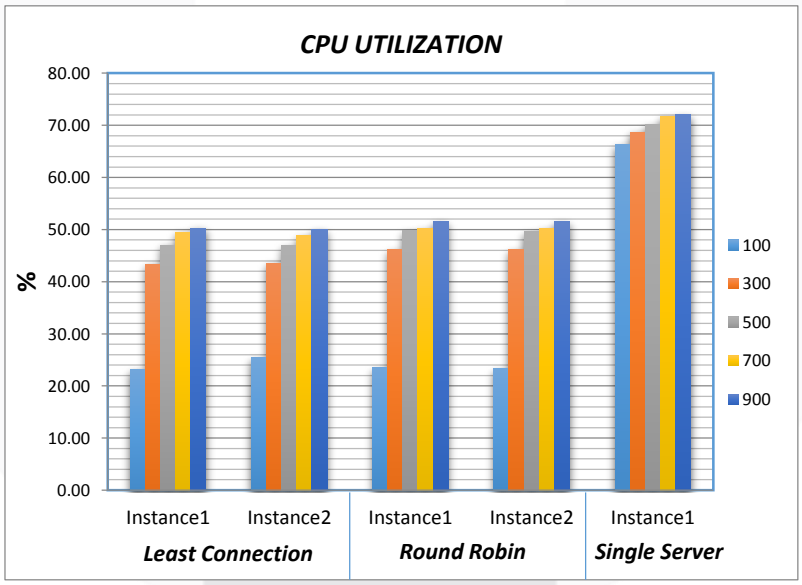
Pada Gambar 3.5 yaitu perbandingan rata-rata persentase dari pengukuran CPU *Utilization* pada setiap *Instance* atau *server* yang tidak menggunakan *Load Balancing*. Kemudian dua *Instance* yang digunakan untuk implementasi dengan dua algoritma *Load Balancing*. Implementasi algoritma *Least Connection* untuk parameter CPU *Utilization* memiliki nilai rata-rata persentase yang lebih rendah yaitu 42,79%. Untuk algoritma *Round Robin* memiliki nilai persentase 44,18% dan single web *server* 69,75%. Oleh karena itu nilai CPU *Utilization* dari algoritma *Least Connection* yang paling kecil dibandingkan yang lain, hal ini disebabkan karena pada algoritma ini *request* dikirim dengan memilikih jalur ke salah satu *server* yang memiliki kepadatan trafik rendah. Sehingga tidak membutuhkan proses respon yang lebih lama untuk mengeksekusi *request*. Lain halnya dengan algoritma *Round Robin* memiliki persentase nilai yang lebih kecil dari *Least Connection*. Hal ini disebabkan karena, jika *client* mengirim *request* yang dikirim ke *server* pertama tidak mendapat respon dari *server* maka *request* otomatis menuju ke *server* dua. Tetapi hal ini mengakibatkan lamanya waktu tunggu respon *server* pertama dari *request* lebih lama jika request tidak mendapat respon dari *server*.



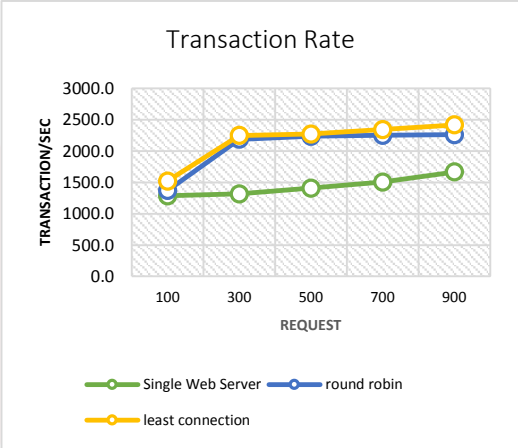
Gambar 3.1 Parameter *Respon Time*



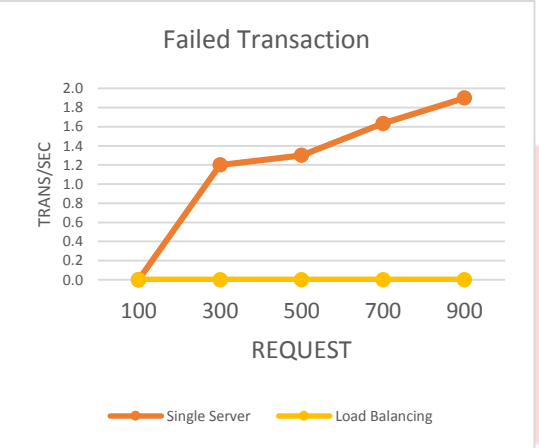
Gambar 3.2 Parameter *Throughput*



Gambar 3.3 CPU *Utilization*



Gambar 3.3 *Transaction Rate*



Gambar 3.4 *Failed Transaction*

Hal ini menyebabkan waktu yang dibutuhkan untuk komunikasi antar *client* dan *server* membutuhkan waktu yang lama. Semakin lama waktu tunggu *server* dalam merespon request maka semakin besar juga jumlah kesibukan yang terjadi.

5. Kesimpulan

Pada hasil pengujian dan analisis yang telah dilakukan, Algoritma *Least Connection* memiliki performansi paling baik diantara algoritma *Round Robin* dan *single server* dari semua segi parameter yang diujikan termasuk *Failed Transaction* dimana pada pengujian algoritma *Least Connection* nilai *Failed Transaction* sama dengan nol. Hal ini menandakan bahwa tidak ada transaksi yang gagal pada saat *server* melayani request. *Load Balancing* dengan algoritma *Least Connection* untuk saat ini adalah cocok diimplementasikan pada web *server* aplikasi input data. Implementasi algoritma tersebut dilakukan menggunakan *OpenStack* yang dirancang untuk menunjang kinerja *server* database gunung api dalam memberi keputusan status level gunung api.

6.

DAFTAR REFERENSI

[1] B. Geologi, "Roadmap 2010 - 2025," 2013. [Online]. Available: http://www.bgl.esdm.go.id. [Accessed September 2016].

[2] S. Ismaeel, M. Ali, C. Dharmendra and R. S. M. Dibaj, "Open Source Cloud Management Platforms: A Review," *IEEE 2nd International Conference On Cyber Security and Cloud Computing*, 2015.

[3] V. G. Joshi and P. M. U. Shankarwar, "An Overview of Open Source Cloud Computing," *International Journal of Advance Research in Computer Science and Managements Studies*, vol. 3, no. 7, pp. 125-130, 2015.

[4] D. U. R, "Cloud Computing with Open Source Tool : *OpenStack*," *American Journal of Engineering Research (AJER)* , vol. 3, no. 9, pp. 233-240, 2014.

[5] O. Comunity, "Load Balancing as a Service," *OpenStack*, [Online]. Available: https://docs.*OpenStack*.org/newton/networking-guide/config-lbaas.html. [Accessed March 2017].