ANALISIS KINERJA *WEB SERVER* MENGGUNAKAN METODE *LOAD BALANCING AS A SERVICE* PADA LINGKUNGAN VIRTUAL OPENSTACK

SKRIPSI

Dibuat untuk Melengkapi Syarat-Syarat yang Diperlukan untuk Memperoleh Gelar Sarjana Terapan

Rizki Amalia

4816050313

KONSENTRASI KEAMANAN SISTEM INFORMASI

PROGRAM STUDI TEKNIK MULTIMEDIA DAN JARINGAN

JURUSAN TEKNIK INFORMATIKA DAN KOMPUTER

POLITEKNIK NEGERI JAKARTA

2020

HALAMAN PERNYATAAN ORISINALITAS

Skripsi ini adalah hasil karya saya sendiri, dan semua sumber baik yang dikutip maupun dirujuk telah saya nyatakan dengan benar.

Nama : Rizki Amalia

NIM : 4816050313

Tanggal : Kamis, 30 Juli 2020

Tanda Tangan :

HALAMAN PENGESAHAN

Skripsi diajukan oleh:

Nama	:	Rizki Amalia
NIM	:	4816050313
Program Studi	:	Teknik Multimedia dan Jaringan
Judul Skripsi	:	Analisis Kinerja *Web Server* Menggunakan Metode *Load Balancing as a Service* pada Lingkungan Virtual Openstack

Telah diuji oleh tim penguji dalam Sidang Skripsi pada hari Rabu, Tanggal 15, Bulan Juli Tahun 2020 Dan dinyatakan **LULUS.**

Disahkan oleh

Pembimbing I	: Drs. Refirman, M.Kom.	( )
Penguji I	: Ayu Rosida Zain, S.ST, M.T.	( )
Penguji II	: Dr. Yohan Suryanto, S.T.,M.T.	( )
Penguji III	: Asep Kurniawan, S.Pd., M.Kom.	( )

Mengetahui:

Ketua Jurusan Teknik Informatika dan Komputer

Mauldy Laya, S.Kom., M.Kom.

NIP. 197802112009121003

KATA PENGANTAR

Puji Syukur saya panjatkan kepada Tuhan Yang Maha Esa, karena atas berkat dan rahmat-Nya, penulis dapat menyelesaikan laporan skripsi ini. Penulisan laporan skripsi ini dilakukan dalam rangka memenuhi salah satu syarat untuk mencapai gelar Sarjana Terapan Politeknik. Penulis menyadari bahwa, tanpa bantuan dan bimbingan dari berbagai pihak, dari masa perkuliahan sampai pada penyusunan laporan skripsi, sangatlah sulit bagi penulis untuk menyelesaikan skripsi ini. Oleh karena itu, penulis mengucapkan terima kasih kepada:

a. Drs. Refirman, M.Kom, selaku dosen pembimbing yang telah menyediakan waktu, tenaga, dan pikiran untuk mengarahkan penulis dalam penyusunan laporan skripsi ini;
b. Hata Maulana, S.Si, M.TI, selaku kepala laboratorium jurusan Teknik Informatia dan Komputer yang telah memberikan izin dalam menggunakan perangkat laptop dan mikrotik untuk melakukan kegiatan skripsi ini;
c. Orang tua dan keluarga penulis yang telah memberikan bantuan dukungan moral dan material; dan
d. Sahabat dan teman-teman yang telah banyak membantu penulis dalam menyelesaikan laporan skripsi ini.

Akhir kata, penulis berharap Allah SWT berkenan membalas segala kebaikan semua pihak yang telah membantu. Semoga laporan skripsi ini membawa manfaat bagi pengembangan ilmu.

Depok, 02 Juli 2020

Penulis

HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI SKRIPSI UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademik Politeknik Negeri Jakarta, saya yang bertanda tangan dibawah ini :

Nama	: Rizki Amalia
NIM	: 4816050313
Program Studi	: Teknik Multimedia dan Jaringan
Jurusan	: Teknik Informatika dan Komputer
Jenis Karya	: Skripsi

Demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Politeknik Negeri Jakarta **Hak Bebas Royalti Noneksklusif (*Non-exclusive Royalty-Free Right*)** atas karya ilmiah saya yang berjudul :

Analisis Kinerja *Web Server* Menggunakan Metode *Load Balancing as a Service* pada Lingkungan Virtual Openstack.

Dengan Hak Bebas Royalti Noneksklusif ini Politeknik Negeri Jakarta berhak menyimpan, mengalih media/format-kan, mengelola dalam bentuk pangkalan data (*database)*, merawat, dan mempublikasikan skripsi saya selama tetap mencamtumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyataan ini saya buat dengan sebenarnya.

Dibuat di : Depok Pada tanggal : 02 Juli 2020

Yang Menyatakan

(Rizki Amalia)

Analisis Kinerja *Web Server* Menggunakan Metode *Load Balancing as a Service* pada Lingkungan Virtual Openstack

Abstrak

Openstack merupakan sistem operasi *cloud* yang salah satu layanannya menyediakan sistem *load balancing* didalamnya yaitu LBaaS (*Load Balancing as a Service*). LBaaS sendiri merupakan penerapan *load balancer* yang didalam *cloud* sehingga prosesnya menjadi lebih efektif karena langsung dalam satu *server* yang sama. Layanan *load balancer* dalam Openstack menyediakan beberapa algoritma *load balancing* yaitu algoritma *round robin, least connection,* dan *source IP*. Data *real* dari kinerja LBaaS pada penelitian ini berdasarkan enam parameter QoS yaitu *throughput, elapsed time, data transferred, responds time*, *transaction rate* dan *load cpu*. Pada penelitian ini dilakukan analisis perbandingan *web server* dengan menggunakan layanan *load balancer as a service* pada Openstack. Dari data yang didapat dan hasil analisis perbandingan kinerja server yang menggunakan *load balancing* lebih balik dibandingkan dengan *single server*. Hal ini ditunjukan pada pengujian layanan *web* dengan apache, menunjukan peningkatan nilai *throughput* sebesar 49.23 % dari *single server*, penurunan nilai elapsed time sebesar 99.58 % dari *single server*, peningkatan nilai *data transferred* 89.17 % sebesar dari *single server*, penurunan nilai *responds time* sebesar dari 95.82 % *single server*, peningkatan *transaction rate* sebesar 77.85 % dari nilai *single server* dan penurunan *load cpu* sebesar 20.40 % dari nilai *single server*. Sedangkan pada pengujian layanan *web* dengan nginx, sistem *load balancing* menunjukan hasil berupa *throughput, elapsed time, data transferred, responds time, transaction rate* dan *load cpu* juga lebih unggul dibandingkan dengan sistem *single server*. Ditunjukan dengan peningkatan nilai *throughput* sebesar 49.09 % dari *single server*, penurunan nilai *elapsed time* sebesar 97.96% dari *single server*, peningkatan nilai *data transferred* 76.50 % sebesar dari *single server*, penurunan nilai *responds time* sebesar dari 93.89 % *single server*, peningkatan *transaction rate* sebesar 84.82 % dari nilai *single server* dan penurunan *load cpu* sebesar 19.48 % dari nilai *single server*.

Kata Kunci : *cloud computing*, lbaas, *load balancer*, Openstack, *web server*

Analisis Kinerja *Web Server* Menggunakan Metode *Load Balancing as a Service* pada Lingkungan Virtual Openstack

Abstrack

Openstack is a cloud operating system where one of the services provides a load balancing system in it, namely LBaaS (Load Balancing as a Service). LBaaS itself is an application of load balancer that is in the cloud so that the process becomes more effective because it is directly in the same server. Load balancer services in Openstack provide several load balancing algorithms, namely round robin algorithm, least connection, and source IP. Real data from LBaaS performance in this study is based on six QoS parameters, namely throughput, elapsed time, data transferred, respond time, transaction rate and cpu load. In this study a comparative analysis of web servers was carried out using load balancer as a service on OpenStack. From the data obtained and the results of a comparative analysis of server performance that uses load balancing more backward than the single server. This is indicated by testing web services with Apache, showing an increase in throughput value of 49.23% from a single server, decreasing the value of elapsed time by 99.58% from a single server, increasing the value of data transferred 89.17% by a single server, decreasing the value of responds time by 95.82 % single server, an increase in transaction rate of 77.85% of the value of a single server and a decrease in cpu load by 20.40% of the value of a single server. Whereas in testing web services with Nginx, load balancing systems show results in the form of throughput, elapsed time, data transferred, responds time, transaction rate and cpu load are also superior to single server systems. Shown by an increase in throughput of 49.09% from a single server, a decrease in the value of elapsed time by 97.96% from a single server, an increase in the value of data transferred by 76.50% from a single server, a decrease in the value of responds time by 93.89% for a single server, an increase in transaction rates of 84.82 % of single server value and cpu load decrease of 19.48% of single server value.

Keywords : *cloud computing, lbaas, load balancer, Openstack, web server*

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR LAMPIRAN

GLOSARIUM

ISTILAH	KETERANGAN
All-in-one single machine	Menyediakan sekumpulan skrip modular yang dapat dijalankan untuk menggunakan *cloud* Openstack dapat digunakan sebagai demo atau lingkungan pengujian. Skrip tersebut berjalan pada satu node yang ada di mesin virtual dan skrip tersebut dapat dikonfigurasikan untuk beberapa node.
Associate	Proses mengaitkan alamat IP mengambang (*floating*) *Compute* dengan alamat IP tetap.
Address pool	Sekelompok alamat IP tetap dan / atau mengambang yang ditugaskan untuk sebuah proyek dan dapat digunakan oleh atau ditugaskan ke *instance* VM dalam proyek.
Allocate	Proses mengambil alamat *floating* IP dari *pool address* sehingga dapat dikaitkan dengan IP tetap pada tamu (*guest*) VM *instance*.
Bandwidth	Jumlah data yang tersedia digunakan oleh sumber daya komunikasi, seperti internet. Merupakan jumlah data yang digunakan untuk men-download hal-hal atau jumlah data yang tersedia untuk men-download.
Client	Komputer yang meminta (*request*) satu layanan tertentu ke suatu server.
Cloud computing	Sebuah model yang mengaktifkan akses ke kolam bersama (shared pool) sumber daya komputasi yang dikonfigurasi, seperti jaringan, server, penyimpanan, aplikasi, dan layanan, yang dapat dengan cepat ditetapkan dan dirilis dengan upaya manajemen yang minimal atau interaksi penyedia layanan.
Cloud Controller	Kumpulan komponen Compute yang mewakili keadaan global cloud; pembicaraan untuk layanan, seperti otentikasi identitas, *Object Storage*, dan pekerja node/storage melalui antrian.
Cloud Controller node	Sebuah node yang menjalankan layanan jaringan, volume, API, scheduler, dan image. Setiap layanan dapat dipecah menjadi node terpisah untuk skalabilitas dan ketersediaan.
Cluster	Dua atau lebih instace yang bekerja sama untuk tujuan komputasi.

Compute node	Sebuah node yang menjalankan daemon nova-compute yang mengelola VM instance yang menyediakan berbagai layanan, seperti aplikasi web dan analisis.
Concurrent users	Jumlah user yang bisa mengaskes *software* secara bersamaan pada saat yang sama.
Credentials	Data yang hanya diketahui atau diakses oleh pengguna dan digunakan untuk memverifikasi bahwa pengguna adalah orang yang katanya dia. Kredensial disajikan ke server selama otentikasi. Contohnya termasuk password, kunci rahasia (secret key), sertifikat digital (digital certificate), dan sidik jari (fingerprint).
Daemon	Sebuah proses yang berjalan di latar belakang dan menunggu permintaan. Mungkin atau mungkin tidak mendengarkan pada TCP atau port UDP. Jangan rancu dengan pekerja (worker).
Dashboard (horizon)	Proyek Openstack yang menyediakan *user interface extensible*, terpadu, berbasis web untuk semua layanan OpenStack.
Flavor	Istilah alternatif untuk VM jenis instance (VM instance type).
Floating IP address	Sebuah alamat IP yang proyek dapat mengaitkan dengan VM sehingga instance memiliki alamat IP publik yang sama setiap kali boot. Anda membuat kolam dari alamat IP mengambang dan menetapkan mereka untuk instance seperti yang diluncurkan untuk menjaga alamat IP yang konsisten untuk mempertahankan DNS tugas.
Firewall	Digunakan untuk membatasi komunikasi antara host dan/atau node, dilaksanakan di Compute menggunakan iptables, arptables, ip6tables, dan iptables.
Health monitor	Menentukan apakah anggota back-end dari kolam VIP dapat memproses permintaan. Sebuah kolam (*pool*) dapat memiliki beberapa monitor kesehatan yang berhubungan dengan itu. Ketika kolam memiliki beberapa monitor yang terkait dengan itu, semua monitor memeriksa setiap anggota dari kolam. Semua monitor harus menyatakan anggota untuk menjadi sehat untuk itu untuk tetap aktif.
High availability (HA)	Sebuah ketersediaan tinggi pendekatan desain sistem dan implementasi layanan yang terkait memastikan bahwa tingkat pengaturan sebelumnya (*prearranged level*) kinerja operasional akan terpenuhi selama periode persetujuan kontrak

	(*contractual measurement*). Sistem ketersediaan tinggi berusaha untuk meminimalkan *downtime* sistem dan kehilangan data.
***Hypertext Transfer Protokol* (HTTP)**	Protokol aplikasi untuk sistem terdistribusikan, kolaboratif, informasi hypermedia. Ini adalah dasar dari komunikasi data untuk *World Wide Web*. *Hypertext* adalah teks terstruktur yang menggunakan link logis (*hyperlink*) antar node yang berisi teks. HTTP adalah protokol untuk menukar atau mentransfer *hypertext*.
***Hypertext Transfer Protocol Secure* (HTTPS)**	Protokol komunikasi terenkripsi untuk komunikasi yang aman melalui jaringan komputer, dengan penyebaran terutama luas di Internet. Secara teknis, itu bukan protokol di dan dari dirinya sendiri; bukan, itu adalah hasil dari hanya layering *Hypertext Transfer Protocol* (HTTP) di atas TLS atau SSL protokol, sehingga menambah kemampuan keamanan TLS atau SSL untuk komunikasi HTTP standar. Kebanyakan API endpoint Openstack dan banyak komunikasi antar-komponen mendukung HTTPS komunikasi.
Identity (keystone)	Proyek yang memfasilitasi otentikasi klien API, penemuan layanan, otorisasi multi-proyek terdistribusi, dan audit. Ini menyediakan direktori pusat pengguna yang dipetakan ke layanan Openstack yang dapat mereka akses. Ini juga mencatat endpoint untuk layanan Openstack dan bertindak sebagai sistem otentikasi yang umum.
Image	Sebuah kumpulan file untuk sistem operasi tertentu (OS) yang Anda gunakan untuk membuat atau membangun kembali server. Openstack memberikan *image pre-built*. Anda juga dapat membuat image kustom, atau snapshot, dari server yang telah Anda diluncurkan. Image kustom dapat digunakan untuk backup data atau sebagai "gold" image untuk server tambahan.
Instances	Sebuah *running* VM, atau VM dalam keadaan terkenal seperti ditangguhkan (*suspended*), *instance* dapat digunakan seperti *server hardware*.
IP *Address*	Deretan angka biner yang dipakai sebagai alamat identifikasi untuk tiap komputer dalam internet.

Least Connection	Algoritma *Least-connection* melakukan pengiriman request pada server anggota cluster, berdasarkan pada server mana yang memiliki *fewest connections* (koneksi paling sedikit).
Load balancer	Sebuah penyeimbang beban (*load balancer*) adalah perangkat logis milik akun *cloud*. Hal ini digunakan untuk mendistribusikan beban kerja antara beberapa sistem back-end atau layanan, berdasarkan kriteria yang ditetapkan sebagai bagian dari konfigurasi.
Load balancing	Proses penyebaran permintaan klien antara dua atau lebih node untuk meningkatkan kinerja dan ketersediaan.
Load Balancer as a service (LBaaS)	Mengaktifkan *Networking* untuk mendistribusikan permintaan yang masuk secara merata antara *instance* yang ditunjuk.
Octavia	Proyek yang bertujuan untuk memberikan *scalable, on demand*, akses *self service* ke layanan *load-balancer*, dengan cara teknologi-agnostik.
Monitor (LBaaS)	Fitur LBaaS yang menyediakan monitoring ketersediaan menggunakan perintah **ping**, TCP, dan HTTP / HTTPS GET.
Node	Sebuah instance VM yang berjalan pada host.
Open source	Sistem pengembangan yang tidak dimiliki oleh suatu individu / lembaga pusat, tetapi oleh para pelaku yang bekerja sama dengan memanfaatkan kode sumber (source-code) yang tersebar dan tersedia bebas (biasanya menggunakan fasilitas komunikasi internet) atau gratis.
Pool	Satu set logis dari perangkat, seperti server web, bahwa kelompok Anda bersama-sama untuk menerima dan lalu lintas proses. Fungsi *load balancing* memilih dimana anggota dari kolam (pool) menangani permintaan baru atau koneksi yang diterima pada alamat VIP. Setiap VIP memiliki satu kolam renang.
Pool member	Sebuah aplikasi yang berjalan pada server back-end dalam sistem *load-balancing*.
Pool details	Berfungsi sebagai pemilihan algoritma yang akan digunakan dalam percobaan.
Port	Sebuah port jaringan virtual dalam Networking; VIF/vNIC terhubung ke port.

Round Robin	Algoritma round robin ditugaskan untuk setiap proses dengan porsi yang sama dan dalam urutan melingkar, menjalankan semua proses tanpa prioritas.
Source IP	*Load balancer* akan memilih salah satu server mana yang akan digunakan berdasarkan hash dari IP sumber, seperti alamat IP client.
Security group	Satu set lalu lintas jaringan aturan penyaringan yang diterapkan untuk *instance Compute*.
Server	Komputer yang menyediakan layanan eksplisit untuk perangkat lunak klien yang berjalan pada sistem itu, sering mengelola berbagai operasi komputer. Server adalah sebuah instance VM dalam sistem Compute. Flavor dan image adalah elemen yang diperlukan saat membuat server.
Storage	Penyimpan, tempat penyimpanan, media yang digunakan untuk menyimpan data yang diolah oleh computer.
Upstream	Adalah versi paling terbaru atau yang masih belum rilis.

BAB I
PENDAHULUAN

1.1 Latar Belakang

Sekarang ini perkembangan teknologi informasi berjalan sangat pesat terutama dalam bidang *cloud computing*. *Cloud computing* merupakan teknologi virtualisasi yang berfungsi sebagai sarana untuk perbandingan luas layanan *cloud* dan strategi penempatan kerja, dan memungkinkan akses jaringan dimana-mana seperti penyediaan *server*, *storage*, *applications* dan *services*, yang dapat dengan cepat disediakan dan dirilis dengan upaya manajemen minimal atau interaksi penyedia layanan.

Seiring berkembangnya bisnis *platform cloud* saat ini, memberikan layanan yang lebih cepat dan lebih handal telah menjadi masalah yang perlu diselsaikan segera. Meningkatnya permintaan akan kebutuhan informasi dalam internet menyebabkan traffik dalam internet juga semakin padat yang mengakibatkan beban kerja pada suatu penyedia *web server* akan mengalami kelebihan beban, sehingga dapat menyebabkan *server* tersebut tidak bisa diakses. *Web server* sendiri memberikan layanan berbasis data dengan menggunakan protokol HTTP atau HTTPS sebagai media dalam mengakses berkas-berkas di dalam suatu halaman *website* dengan menggunakan *web browser* untuk *request* data dan *server* akan mengirim data dalam bentuk halaman *web* dan pada umumnya berbentuk dokumen HTML (Hidayah 2019).

Sebuah jaringan yang memiliki distribusi *load* yang merata akan membantu optimasi terhadap *resource* yang tersedia untuk dapat memaksimalkan *throughput*, meminimalisasi *response time*, dan mencegah terjadinya *overload* pada jaringan. Jadi dapat dikatakan bahwa *load balancer* memiliki fungsi utama untuk membagi jumlah pekerjaan yang harus dilakukan pada sebuah komputer menjadi dua atau lebih sehingga banyak pekerjaan dapat dilakukan secara bersamaan dan secara umum semua pengguna dapat dilayani lebih cepat.

Openstack menyediakan layanan *Load Balancing as a Service* (LbaaS) yang bisa dimanfaatkan untuk menangani permasalahan *overload* terhadap *request* tanpa

mengalami kelebihan beban, memaksimalkan *throughput*, menghindari *overload*, mengurangi konsumsi energi, dan meminimalkan *response time*.

Sebelumnya telah dilakukan penelitian tentang implementasi *Load Blancing as a Service* (LbaaS) berbasis Openstack dan menganalisa perbandingan antara *single server* dan multi server yang menggunakan *load balancing* dengan metode *round robin*, *least connection*, dan *source ip* dengan parameter pengukuran *load balancing* berupa *Quality of Service* seperti, *throughput*, *elapses time*, *response time*, *transaction rate*, dan *data transferred* dengan layanan *web server*. Berdasarkan referensi penelitian tersebut, penelitian ini akan melakukan evaluasi terhadap dua jenis *web server* pada lingkungan virtual Openstack dengan arsitektur *single server* dan *multi server* menerapkan *load balancing* menggunakan parameter pelayanan pada penelitian sebelumnya.

1.2 Perumusan Masalah

Dalam mencapai tujuan dari penelitian ini, terdapat beberapa permasalahan yang dirumuskan sebagai berikut :

1. Bagaimana mewujudkan kinerja *web server* dengan *load balancing* untuk menghindari *overload* ?
2. Bagaimana kinerja dua jenis *web server* saat diterapkan *load balancing* ?
3. Bagaimana memperoleh nilai pengukuran parameter layanan *Quality of Service* pada *web server* yang menggunakan *load balancing* pada Openstack sebelum dan sesudah diterapkan *load balancing* ?

1.3 Batasan Masalah

Pada penelitian ini, ruang lingkup penelitian ini meliputi :

1. Analisis kinerja dua jenis *web server* pada lingkungan virtual Openstack dengan menggunakan *load balancing* pada tiga *instance server*.
2. Algoritma penjadwalan pada *load balancing* yang digunakan hanya *Round Robin*, *Least Connection* dan *Source IP*.
3. Pengujian dilakukan dengan parameter *Quality of Service* penelitian sebelumnya.
4. Tidak melibatkan optimasi untuk performa dari *server* yang digunakan.

1.4 Tujuan dan Manfaat

1.4.1 Tujuan

1. Untuk mewujudkan kinerja *web server* dengan *load balancing* dapat dilakukan dengan menggunakan LBaaS pada lingkungan Openstack.
2. Untuk mengetahui kinerja dua jenis *web server* yaitu apache dan ngix saat diterapkan *load balancing* dapat dilakukan dengan menganalisa perbandingan hasil kinerja dua jenis *web server d*engan LBaaS pada Openstack.
3. Untuk memperoleh nilai pegukuran QoS dengan menggunakan siege pada *web server* dengan menggunakan *load balancing* berupa *throughput, elapsed time, data transferred, responds time, transaction rate* dan *load cpu* yang terdapat pada Openstack.

1.4.2 Manfaat

1. Mengetahui kinerja *web server* yang ada pada Openstack *cloud*.
2. Mengetahui kinerja metode *load balancing* dengan metode *round robin, least connection* dan *source ip* pada Openstack.
3. Mengetahui evaluasi *web server* menggunakan *load balancing* pada lingkungan virtual Openstack adalah mengetahui kinerja dua jenis *web server* sebelum dan sesudah menggunakan *load balancing*.

1.5 Metode Pelaksanaan

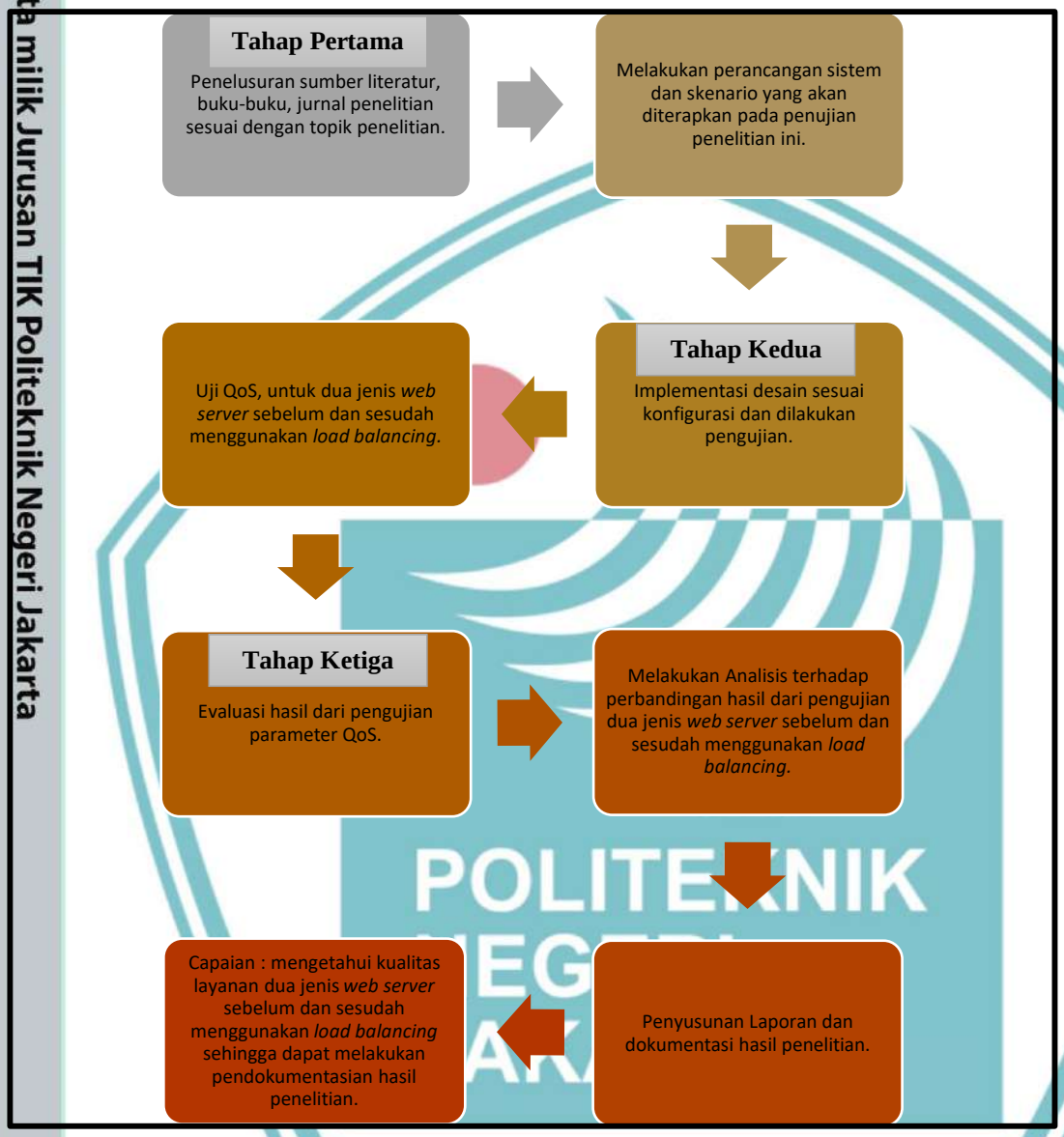
Penelitian ini dilakukan dengan metode sebagai berikut :

Tabel 1.1 Metode Pelaksanaan

Tahapan Penelitian	Pekerjaan Penetilan	Hasil
Tahap Pertama	Penelusuran sumber literatur, buku-buku, jurnal penelitian sesuai dengan topik penelitian.	Capaian : Desain perancangan sistem dan desain skenario untuk Analisis Kinerja *Web Server* Menggunakan Metode *Load Balancing as a Service* pada Lingkungan Virtual Openstack
	Melakukan perancangan sistem dan skenario yang akan diterapkan pada penujian penelitian ini.	
Tahap Kedua	Implementasi desain sesuai konfigurasi dan dilakukan pengujian.	Tahap ini Pengujian menggunakan aplikasi Siege
	Uji QoS, untuk dua jenis *web server* sebelum dan sesudah	Capaian : mengetahui kualiatas layanan dua jenis *web server* sebelum dan

	menggunakan *load balancing*.	sesudah menggunakan *load balancing*.
Tahap Ketiga	Evaluasi hasil dari pengujian parameter QoS.	Tahap ini Menganalisis hasil dari penelitian yang di ujikan. Capaian : mengetahui kualitas layanan dua jenis *web server* sebelum dan sesudah menggunakan *load balancing* sehingga dapat melakukan pendokumentasian hasil penelitian.
	Melakukan Analisis terhadap perbandingan hasil dari pengujian dua jenis *web server* sebelum dan sesudah menggunakan *load balancing*.	
	Penyusunan Laporan dan dokumentasi hasil penelitian.	

Pada tahapan metode pelaksanaan diatas ini diuraikan dalam tiga tahapan. Pelaksanaan tahapan dikerjakan secara berurutan, seperti tahapan-tahapan pekerjaan berikut : tahapan pertama melakukan penelusuran sumber literatur, buku-buku, jurnal penelitian sesuai dengan topik penelitian dan melakukan perancangan sistem dan skenario yang akan diterapkan pada penujian penelitian ini, tahapan kedua melakukan implementasi desain sesuai konfigurasi dan dilakukan pengujian dan uji QoS, untuk dua jenis *web server* sebelum dan sesudah menggunakan *load balancing*, tahapan ketiga melakukan evaluasi hasil dari pengujian parameter QoS, melakukan analisis terhadap perbandingan hasil dari pengujian dua jenis *web server* sebelum dan sesudah menggunakan *load balancing*, dan penyusunan laporan dan dokumentasi hasil penelitian sehingga mendapatkan capaian mengetahui kualitas layanan dua jenis *web server* sebelum dan sesudah menggunakan *load balancing* sehingga dapat melakukan pendokumentasian hasil penelitian.



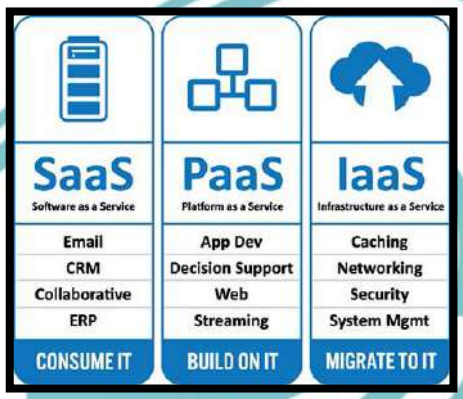
Gambar 1.2 Metode Pelaksanaan

Pada tahapan metode pelaksanaan diatas ini diuraikan dalam tiga tahapan. Pelaksanaan tahapan dikerjakan secara berurutan, seperti tahapan-tahapan pekerjaan berikut : tahapan pertama melakukan penelusuran sumber literatur, buku-buku, jurnal penelitian sesuai dengan topik penelitian dan melakukan perancangan sistem dan skenario yang akan diterapkan pada penujian penelitian ini, tahapan kedua melakukan implementasi desain sesuai konfigurasi dan dilakukan pengujian dan uji QoS, untuk dua jenis *web server* sebelum dan sesudah menggunakan *load balancing*, tahapan ketiga melakukan

evaluasi hasil dari pengujian parameter QoS, melakukan analisis terhadap perbandingan hasil dari pengujian dua jenis *web server* sebelum dan sesudah menggunakan *load balancing*, dan penyusunan laporan dan dokumentasi hasil penelitian sehingga mendapatkan capaian mengetahui kualitas layanan dua jenis *web server* sebelum dan sesudah menggunakan *load balancing* sehingga dapat melakukan pendokumentasian hasil penelitian.

BAB II
TINJAUAN PUSTAKA

2.1 *Cloud Computing*



Gambar 2.1 *Cloud Computing Service*

(Sumber : http://robicomp.com/wp-content/uploads/2017/11/Layanan-Cloud-Storage.jpg**)**

Cloud computing adalah model unutk memungkinkan akses jaringan dimana-mana, sesuai permintaan *resources* komputasi (misalnya, jaringan, *server*, penyimpanan, aplikasi, dan *services*) yang dapat dengan cepat disediakan dan dirilis dengan upaya manajemen minimal atau interkasi penyedia layanan (Kurniawan 2015).

Cloud computing memiliki tiga model layanan, yaitu :

a. *Software as a Service* (SaaS)

   Software as a Service adalah sebuah layanan dimana pengguna hanya diberikan layanan aplikasi. Konsumen tidak mengelola atau mengontrol infrastruktur *cloud* yang mendasarinya termasuk jaringan, server, sistem operasi, penyimpanan, dengan pengecualian pengaturan konfigurasi aplikasi spesifik pengguna yang terbatas. Contohnya adalah google apps, Microsoft dynamics, Email, CRM, dan ERP.

b. *Platform as a Service* (PaaS)

   Platform as a Service adalah jenis layanan diatas SaaS dimana pengguna diberikan hak untuk mengembangkan aplikasinya melalui internet. Konsumen tidak mengelola atau mengontrol infrastruktur cloud yang mendasarinya termasuk

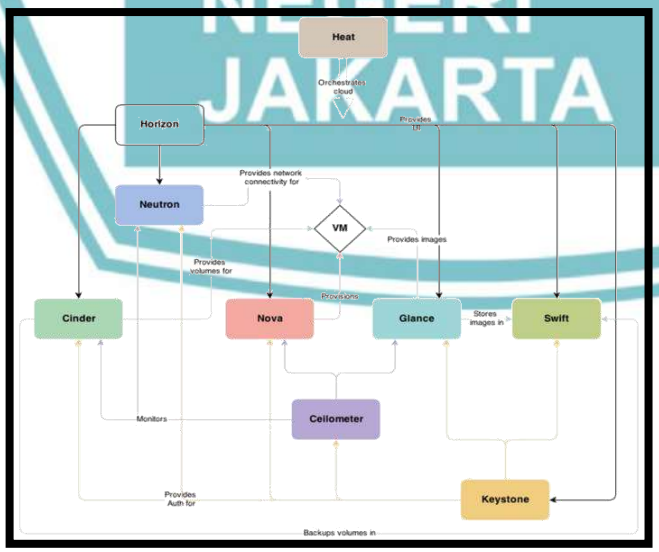
jaringan, server, sistem operasi, atau penyimpanan, tetapi memiliki kontrol atas aplikasi yang digunakan dan mungkin konfigurasi lingkungan hosting aplikasi. Contohnya adalah Windows Azure, dan force.com.

Infrastructure as a Service (IaaS)

Infrastructure as a Service adalah sebuah layanan dimana pengguna diberikan hak untuk menyewa layanan berupa sumber daya atau infrastruktur secara penuh seperti *proccessor, memory, storage*, dan *bandwidth*. Konsumen tidak mengelola atau mengendalikan infrastruktur *cloud* yang mendasarinya tetapi memiliki kendali atas sistem operasi, penyimpanan, aplikasi yang disebarkan, dan kemungkinan kendali terbatas untuk komponen jaringan tertentu. Contohnya adalah Amazon EC2, Rackspace, dan VMware.

2.2 Openstack

Openstack adalah sistem operasi *cloud* yang mengelola sumber daya antara lain komputasi, *storage*, dan jaringan, yang tersedia pada infrastruktur fisik seperti dalam sebuah fasilitas *data center*. Admin atau pengguna dapat mengendalikan dan menyediakan layanan melalui *interface web*. Di luar fungsionalitas IaaS, menyediakan komponen tambahan seperti *fault management* dan *service management* untuk memastikan *high availability* aplikasi pengguna (Openstack 2020).



Gambar 2.2 *Arsitektur Openstack*

(Sumber : https://eueung.gitbooks.io/buku-komunitas-sdn-rg/content/pengantar_openstack/assets/os02.jpg**)**

2.1 Arsitektur Openstack

a) Nova (*Compute Service*)

Merupakan komponen utama dari sistem IaaS, karena *nova compute* yang mengatur proses dan alokasi CPU untuk setiap VM yang dibuat di Openstack. Hal ini membuat nova memegang penuh kendali dan mengelola sumber daya komputasi, jaringan, otorisasi, dan kebutuhan skalabilitas dari Openstack *cloud* ini.

b) Neutron (*Networking Service)*

Fungsi utama Neutron adalah menyediakan layanan *Network as a service*. Neutron merupakan sistem untuk melakukan *provisioning* jaringan yang melibatkan entitas *virtual machine* (VM) yaitu mengatur jaringan/*subnet*, router, *load-balancer*, dan *floating IP*.

c) Horizon (*Dashboard)*

Merupakan suatu layanan *user interface* dalam infrastruktur Openstack yang memberikan akses visualisasi bagi *user* dalam menciptakan *cloud*.

d) Glance (*Image Service)*

Merupakan salah satu produk Openstack yang digunakan sebagai *virtual disk images.*

e) Keystone (*Identity Service*)

Menyediakan layanan identitas dan akses kebijakan untuk semua komponen Openstack. 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.

f) Cinder (*Block Storage Service)*

Menyediakan layanan penyimpanan blok untuk digunakan oleh *compute instances.* Cinder di desain untuk bekerjasama dengan komponen Openstack, terutama *compute* dan *dashboard*. Cinder mengatur kebutuhan terhadap media penyimpanan dan dapat digunakan untuk skenario pemakain *sensitive* atau membutuhkan kinerja tinggi seperti, penyimpanan database, akses raw pada

penyimpanan blok, *snapshot management* dan untuk *backup*.

Placement

Placement merupakan sekumpulan REST API dan model data yang digunakan unutk melacak inventaris dan penggunaan penyedia *resource*, bersama dengan *resource* kelas yang berbeda. Contohnya, penyedia *resource* dapat berupa node komputasi, kumpulan penyimpanan bersama, atau kumpulan alokasi IP.

Heat *(Orchestration*)

Heat adalah sebuat *services* yang digunakan untuk menyusun/mengkolaborasi multiple composite aplikasi *cloud* menggunakan AWS menggunakan REST (*Representational State Transfer*) API dan *Cloud Formation-Compatible* Query API. Software ini mengintegrasikan komponen lainnya dari Openstack ke dalam sebuah *one-file template*. Dari *template* tersebut memungkinkan pembuatan hampir semua openstack resource type seperti : *Instances, Floating, IPs, Volumes, Security Groups, Users, dan juga advanced fungsionality* seperti *High Availability*, *instance autoscaling*, dan *nested stacks*.

i) Swift (*Object Storage*)

Swift, menawarkan perangkat lunak penyimpanan cloud sehingga dapat menyimpan dan mengambil banyak data dengan API. Swift berguna untuk menyimpan data yang tidak terstruktur dan tanpa terikat.

j) Ceilometer/Telemetry

Ceilometer menyediakan data penggunaan bagi pengguna unutk cloud berbasis Openstack, yang dapat digunakan unutk penagihan pelanggan, pemantauan sistem, atau peringatan.

2.2.2 Struktur Openstack

a) *Controller*

Node pengontrol menjalankan layanan *identity, image, placement, compute, networking*, berbagai agen *networking*, dan *dashboar*d. Ini juga termasuk layanan pendukung seperti database SQL, *message qu*eue, dan *Network Time Protokol* (NTP). Secara opsional, *controller node* menjalankan bagian dari layanan b*lock*

storage, object storage, orchestration, dan telemetry. *Controller node* membutuhkan minimal dua *interface* jaringan. Kapasitas minimum yang diperlukan pada umumnya untuk membangun *controller* node adalah 1 processor, 2 GB *memory*, dan 5 GB *storage* (Openstack 2020).

b) *Compute*

Node komputasi menjalankan bagian hypervisor dari *compute*, yang beroperasi pada tenant *virtual machine* atau *instances*. Secara *default compute* menggunakan *Kernel-Based Virtual Machine* (KVM) sebagai hypervisor. Compute node tersebut juga menjalankan *netwoking plug-in* dan layer 2 agen yang beroperasi didalam *network tenant* dan menerapkan *security group*, dan dalam menjalankan lebih dari satu compute node. *Service optional* dari *compute node* adalah menjalankan *telemetry agent*. Komponen ini memberikan fitur tambahan berupa grafik untuk 8 cakupan ke para pengguna. Kapasitas minimum yang diperlukan untuk membangun *compute node* adalah 1 *processor*, 2 GB *memory*, dan 10 GB *storage* (Openstack 2020).

c) *Network*

Network node menjalankan *network plug-in*, *layer* 2 *agent*, dan beberapa layer 3 *agent* yang menyediakan dan mengoperasikan n*etwork tenant*. *Service layer* 2 meliputi penyediaan *virtual network* dan *tunnels*. *Service layer* 3 meliputi *routing*, *Network Address Translation* (NAT), dan *Dynamic Host Configurarion Protocol* (DHCP). *Node* ini juga menangani konektivitas eksternal (internet) untuk *tenant virtual machine* atau *instances* . Kapasitas minimum untuk membangun *network node* adalah 1 *processor*, 512 MB *memory*, dan 5 GB *storage* (Openstack 2020).

2.2.3 Devstack (*All in one single machine*)

Gambar 2.3 Logo Devstack

(Sumber : https://docs.openstack.org/devstack/latest/_images/logo-blue.png**)**

Devstack merupakan *all in one single machine* yang menyediakan sekumpulan skrip modular yang dapat dijalankan untuk menggunakan *cloud* Openstack dapat

digunakan sebagai demo atau lingkungan pengujian. Skrip tersebut berjalan pada satu node yang ada di mesin virtual dan skrip tersebut dapat dikonfigurasikan untuk beberapa node. Secara *default*, layanan inti pada Openstack akan diinstal tetapi pengguna dapat menambahkan layanan melalui mengkonfigurasikan layanan tambahan tersebut. Semua layanan devstack bersumber dari git master (Openstack, 2020).

2.2.4 Openstack Train

Gambar 2.4 Logo Openstack Train

(Sumber : https://www.openstack.org/software/train/**)**

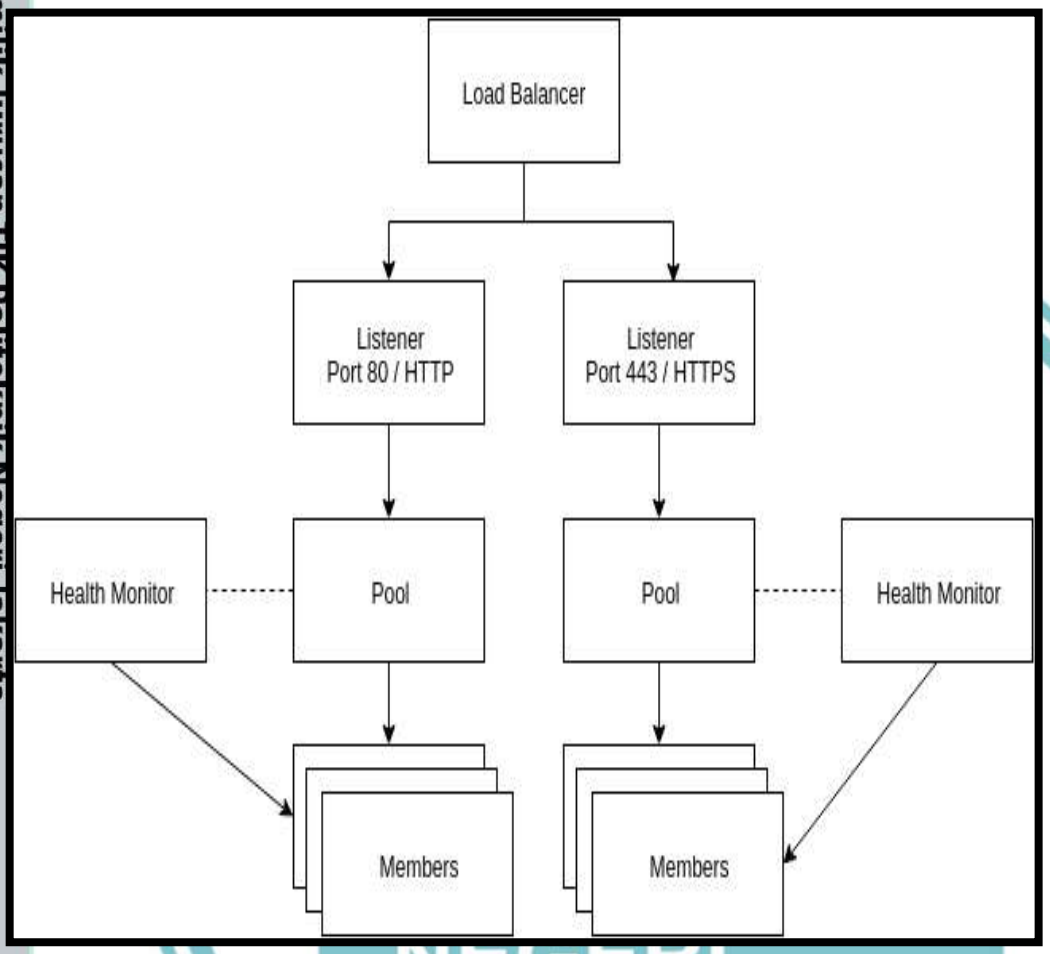
Openstack Train merupakan versi terbaru dari Openstack yang rilis pada 16 Oktober 2019. Keunggulan versi Train ini melingkupi keamanan dan perlindungan data yang sudah ditingkatkan dari versi sebelumnya, kemajuan untuk *artificial intelligence* (AI) dan *machine learning* (ML) untuk kasus pengelolaan dan pelacakan *resource* yang ditingkatkan, serta keamanan dan perlindungan data yang lebih ditingkatkan dari versi sebelumnya (Openstack, 2019).

2.2.5 Octavia

Octavia merupakan salah satu turunan dari layanan Neutron LBaaS yang dirancang sebagai penyeimbang atau *load balancing* beban kerja yang ada pada Openstack. Octavia bertugas untuk menyelesaikan pengiriman beban pada *load balancing* dengan mengelola mesin virtual, dan server sebagai wadah yang dilakukan secara

kolektif yang disebut amphorae yang bekerja sesuai dengan permintaan (Openstack, 2017).

2.3 Load Balancing



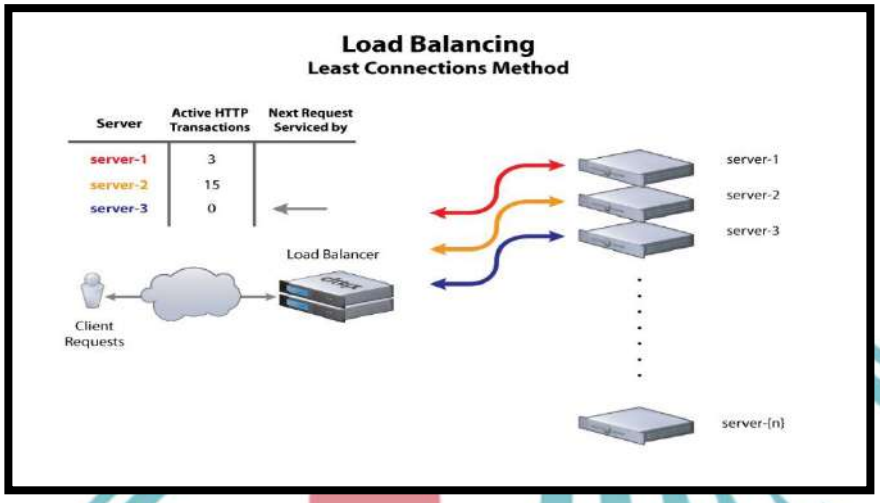
Gambar 2.5 Konsep LBaaS Openstack

(Sumber : https://docs.openstack.org/newton/id/networking-guide/_images/lbaasv2-diagram.png**)**

Load balancing adalah proses pendistribusian beban terhadap sebuah servis yang ada pada sekumpulan *server* atau perangkat jaringan. Ketika ada permintaan dari pemakai ketika banyak permintaan dari pemakai maka *server* tersebut akan terbebani karena harus melakukan proses pelayanan terhadap permintaan pemakai. Solusi yang cukup bermanfaat adalah dengan membagi-bagi beban yang datang ke beberapa *server*, jadi tidak berpusat ke salah satu perangkat jaringan saja. Teknologi itulah yang disebut Teknologi *Load Balancing*. Teknologi *load balancing* dapat diperoleh keuntungan seperti menjamin reabilitas servis, availabilitas dan skalabilitas suatu jaringan (Irsyad 2017).

Terdapat beberapa algoritma *load balancing* :

a. *Least Connection*

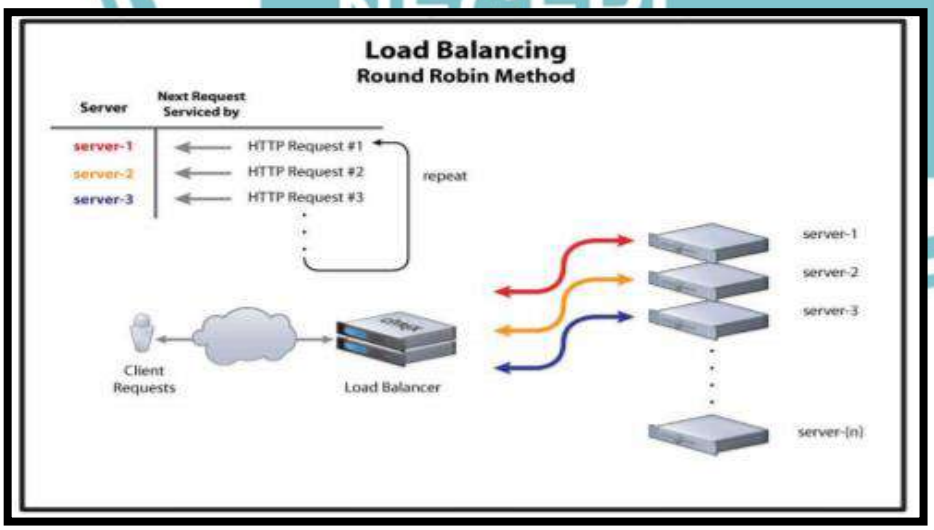


Gambar 2.6 Proses Kerja Algoritma *Least connection*

(**Sumber :** https://livecodetech.co.ke/images/service/load.jpg)

Algoritma *least connection* mengarahkan koneksi jaringan koneksi jaringan ke *server* dengan sedikitnya jumlah koneksi yang ada pada sistem. Ini adalah salah satu algoritma penjadwalan dinamis, karena itu perlu menghitung jumlah koneksi untuk setiap *server* secara dinamis untuk memperkirakan bebannya. Penyeimbang beban mencatat jumlah koneksi dari setiap *server*, meningkatkan jumlah koneksi dari *server* ketika koneksi baru dikirim untuk itu, dan menurunkan jumlah koneksi dari *server* ketika *finish* sambungan atau *timeout* (Ramadhani 2019).

b. *Round Robin*



Gambar 2.7 Proses Kerja Algoritma *Round Robin*

(**Sumber :** https://www.citrix.com/blogs/wp-content/uploads/2010/09/LB_LeastConns_1008311.jpg)

Algoritma *round robin* dirancang untuk sistem *time sharing*. Algoritma ini mirip dengan penjadwalan FCFS, namun preemption ditambahkan untuk switch antara proses. Antrian *ready* dan mengalokasikan masing-masing proses untuk interval waktu tertentu sampai satu time slice l quantum.

Algoritma *round robin*, memiliki prinsip dasar, yaitu semua sumber antrian dianggap sama sehingga diberi waktu yang disebut time quantum. Jika time quantum habis atau proses selesai, maka proses berlanjut ke antrian berikutnya. Penjadwalan ini cukup adil karena tidak ada antrian yang diprioritaskan,semua mendapat jatah waktu yang sama (Ramadhani 2019).

Source IP

Pada algoritma *source ip*, untuk memilih *server* mana yang akan mengirim permintaan didasarkan pada alamat IP sumber sebagai kunci *hash* dari tabel *hash*. Jika *server* tersedia, permintaan akan dikirimkan ke *server*, atau akan dikembalikan bahwa *server* tidak dapat diakses.

2.4 *Web Server*

Web server adalah sebuah *software* yang memberikan layanan berbasis data dengan menggunakan protokol HTTP atau HTTPS dari sebagai media dalam mengakses berkas-berkas di dalam suatu halaman *website* dengan menggunakan *web browser* untuk *request* data dan *server* akan mengirim data dalam bentuk halaman *web* dan pada umumnya berbentuk dokumen HTML. Halaman *web* yang diminta bisa terdiri dari berkas teks, video, gambar, file dan banyak lagi (Rahmatulloh 2017).

High Availability web server adalah sistem *web server* yang menjaga dan menjamin ketersediaan data dan informasi yang disajikan agar secara terus menerus dapat diakses oleh pengguna dengan tujuan untuk memenuhi kebutuhan data dan informasi yang diperlukan oleh pengguna.

2.5 Apache

Apache HTTP *server* adalah perangkat lunak dengan platform *operating system* (OS) yang mendukung multi-tasking, dan menyediakan layanan untuk aplikasi lain yang terhubung ke dalamnya, seperti *web browser*. Apache pertama kali dikembangkan untuk bekerja dengan sistem operasi Linux/Unix, tetapi kemudian diadaptasi untuk bekerja di bawah sistem lain, termasuk Windows dan Mac (Aziz 2015).

2.6 Nginx

Nginx adalah software *open-source* yang memiliki kinerja tinggi sebagai *server* HTTP dan *reverse proxy*. Nginx dengan cepat memberikan konten statis dengan penggunaan efisien sumber daya sistem. Hal ini dapat menyebarkan dinamis HTTP konten di jaringan menggunakan FastCGI *handler* untuk *script*, dan dapat berfungsi sebagai perangkat lunak yang sangat mampu penyeimbang beban. Nginx dibangun secara modular dan dengan demikian mampu mendukung berbagai fitur seperti *load Balancing* dan *Reverse Proxying*, *Virtual hosts* berbasis nama dan IP, Fast CGI, akses langsung ke cache, SSL, Flash *Video Streaming* dan sejumlah fitur-fitur standar lainnya. Nginx dapat dijalankan dan tersedia untuk platform Unix, Linux, varian dari BSD, MacOS X, Solaris, dan Microsoft Windows (Aziz 2015).

2.7 Siege

Siege adalah *tools benchmarking* yang digunakan untuk menguji http/https. Siege didesain agar seorang *web developers* dapat mengukur performansi *code* mereka dibawah tekanan, untuk melihat bagaimana *web* tersebut dapat bertahan di internet. Siege memungkinkan pengguna untuk melakukan permintaan *request* pada *web server* dengan jumlah *user* simultan yang dapat dikonfigurasi. Durasi pada siege diukur dalam bentuk transaksi, jumlah *user* yang disimulasikan dan berapa kali setiap pengguna yang disimulasikan mengulangi proses *hit* pada *server*. Performa yang diukur meliputi durasi keseluruhan tes (*elapsed time*), jumlah data yang di transfer, *response time* dari server, *transaction rate*, *throughput*, jumlah *user* yang simultan (*concurrency*), dan jumlah transaksi yang sukses. Hasil dari pengukuran ditampilkan pada setiap akhir tes setelah tes selesai dijalankan. Siege pada dasarnya memiliki 3 mode operasi, yaitu *regression*, *internet simulation*, dan *brute force*. Siege dapat membaca dengan jumlah URL yang banyak yang terdapat didalam *configuration file* dan menjalankannya secara bertahap (*regression*) atau secara random (*internet simulation*). Atau pengguna dapat melakukan *hit* pada satu URL dengan menjalankannya pada *command line* (Hidayah 2019).

2.8 Ubuntu

Ubuntu merupakan salah satu distro Linux yang berbasis Debian. Ubuntu memfasitasi penggunanya dengan memberikan perbaikan keamanan, mengeluarkan perbaikan bug kritis, tampilan desktop yang konsisten. Ubuntu untuk

tampilan GUI memakai desktop Gnome. Ubuntu juga banyak dikembangkan menjjadi distro Linux yang lainnya, seperti Linux Mint, Klikit Linux, dan Blankon2 (Wirasasmita 2015).

2.9 *Quality of Service* (QoS)

Quality of service (QoS) merupakan metode pengukuran tentang seberapa baik jaringan dan merupakan suatu usaha untuk mendefinisikan karakteristik dan sifat dari satu servis. QoS digunakan untuk mengukur sekumpulan atribut kinerja yang telah dispesifikasikan dan diasosiasikan dengan suatu servis (Wulandari 2016). Parameter QoS yang digunakan antara lain :

a) *Throughput*

Throughput adalah jumlah rata-rata *byte* yang ditransfer setiap detik dari *server* ke semua pengguna yang disimulasikan.

b) *Elapsed Time*

Elapsed Time adalah durasi seluruh tes siege. Ini diukur dari saat pengguna menggunakan siege hingga pengguna yang disimulasikan terakhir menyelesaikan transaksinya.

c) *Response Time*

Response Time adalah waktu rata-rata yang diperlukan untuk menanggapi permintaan setiap pengguna yang disimulasikan.

d) *Transaction Rate*

Transaction Rate adalah jumlah rata-rata transaksi yang dapat ditangani server per detik, singkatnya: transaksi dibagi dengan waktu yang berlalu.

e) *Data Transffered*

Data Transffered adalah jumlah data yang ditransfer ke setiap pengguna simulasi siege. Ini termasuk informasi tajuk serta konten. Karena itu termasuk informasi *header*, jumlah yang dilaporkan oleh siege akan lebih besar daripada jumlah yang dilaporkan oleh *server*.

f) *Load* CPU

Load CPU adalah jumlah proses yang sedang dieksekusi oleh CPU atau menunggu untuk dieksekusi oleh CPU. Jadi rata-rata *load* CPU adalah jumlah rata-rata proses yang sedang atau menunggu dieksekusi. Rata-rata *load* CPU tinggi mengartikan bahwa CPU kelebihan beban dengan terlalu banyak proses.

2.10 Virtual Box

Sebuah program komputer *under* Windows yang merupakan sebuah *tool* atau program virtualisasi yang dapat digunakan untuk mengeksekusi sistem operasi tambahan" di dalam sistem operasi "utama", dengan kata lain sebuah sistem operasi di dalam sistem operasi dinamakan Virtual Box. Virtual Box pada awalnya dirancang oleh sebuah perusahaan Jerman Innotek GmbH pada tahun 2007 dengan vesi 1.0 Virtual Box OSE (*Open Source Edition*) dengan lisensi GNU (*General ublic License*). Pada tahun 2010 di akuisisi oleh Oracle Corporation dan ertanggung jawab melanjutkan proyek Virtual Box hingga kini sampai versi erbarunya yang mampu bekerja di *operating system* Windows, Linux, dan Solaris (Assegaf 2019).

2.11 Htop

Htop merupakan sistem untuk memonitor proses pada program yang berjalan secara nteraktif, htop memberikan informasi visual tentang prosesor, CPU dan memori yang sedang berjalan. Htop dirancang sebagai alternatif program Unix yang memberikan daftar lengkap proses yang sedang berjalan (Haritsah 2019).

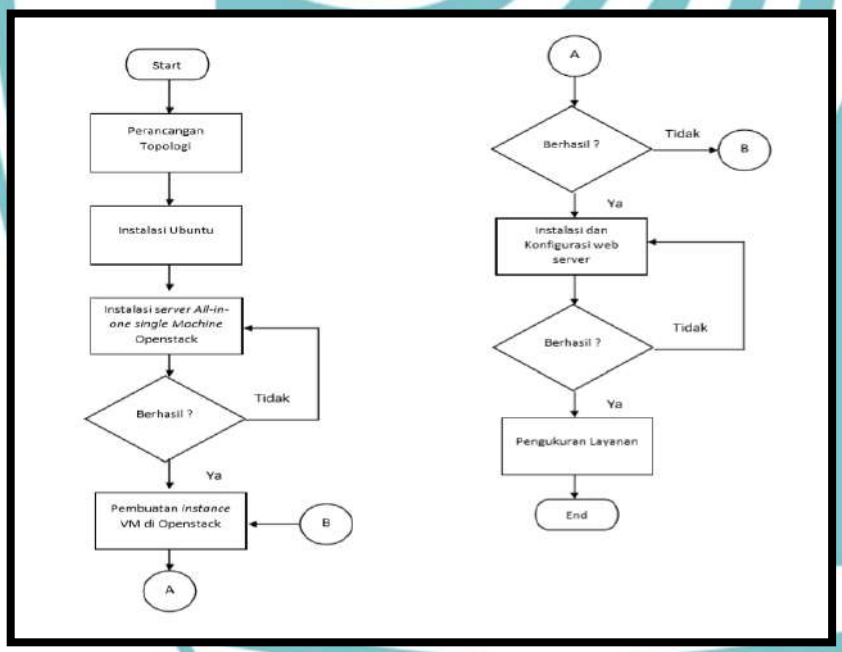
BAB III
PERANCANGAN DAN REALISASI

Pada bab ini akan dijelaskan mengenai perancangan sistem yang diterapkan pada penelitian. Penelitian ini menggunakan layanan *load balancing* pada Openstack yaitu LBaaS (*Load Balancing as a Service*) untuk menganalis kinerja dua jenis *web server*, yaitu nginx dan apache. Berikut penjelasan mengenai perancangan sistem dari penelitian ini.

3.1 Perancangan Sistem

Pada tahap ini akan dijelaskan gambaran mengenai perancangan sistem, dimulai pembuatan *flowchart*, desain topologi jaringan, spesifikasi perangkat dan *software/tools*.

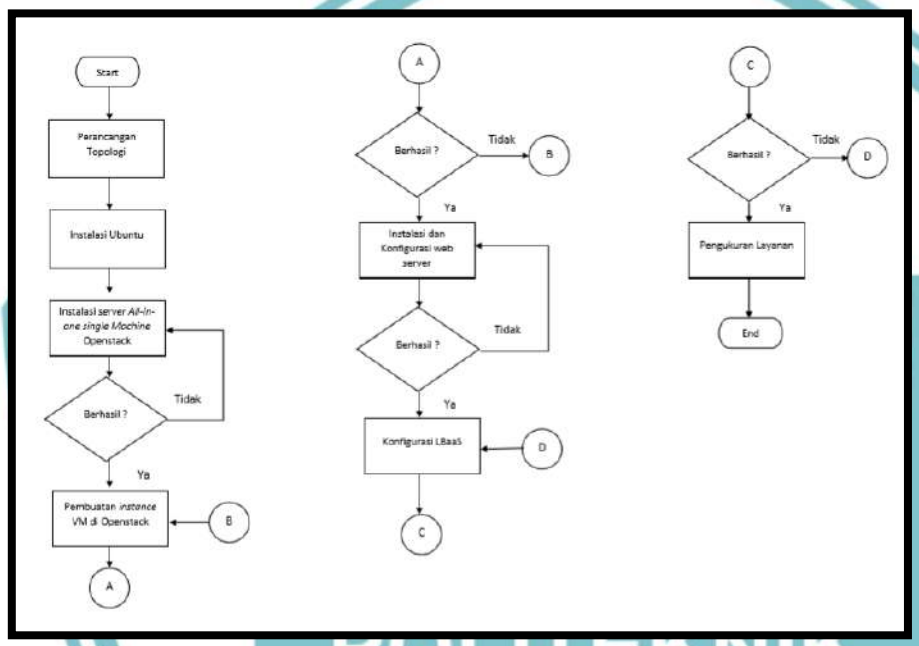
3.1.1 *Flowchart* Pengerjaan



Gambar 3.1 *Flowchart* Pengujian *Single Server*

Gambar 3.1 diatas merupakan *flowchart* untuk pengerjaan dan pengujian terhadap satu *instance* pada platform Openstack yang akan dipasangkan *web server* dan kemudian dilakukan pengujian terhadap *load instance* tersebut. Platform Openstack akan diinstall pada laptop yang telah dilakukan pemasangan sistem operasi Ubuntu. Laptop ini akan bertindak sebagai *All-in-one server* Openstack (*controller* dan *compute node* Openstack dalam satu mesin). Setelah instalasi Openstack selesai

dilakukan, langkah berikutnya adalah dengan melakukan pembuatan *instance* didalam platform Openstack. *Instance* dibuat dengan menggunakan *image cloud* sistem operasi ubuntu. Setelah *instance* selesai dibuat, *instance* akan diinstall kedua aplikasi *web server* dan kemudian dilakukan pengambilan sampel data secara bergantian untuk masing-masing aplikasi *web server*, nginx dan apache sebagai perbandingan dengan skenario penggunaan *load balancer*.



Gambar 3.2 *Flowchart* Pengujian *Multi Server*

Gambar 3.2 diatas merupakan *flowchart* untuk pengerjaan dan pengujian terhadap skenario penggunaan *load balancer* pada *cluster* 2 *instance* dan *cluster* 3 *instance* di platform Openstack yang akan dipasangkan *web server* dan kemudian dilakukan pengujian terhadap *load instance* tersebut. Platform Openstack akan diinstall pada laptop yang telah dilakukan pemasangan sistem operasi Ubuntu. Laptop ini akan bertindak sebagai *All-in-one server* Openstack (*controller* dan *compute node* openstack dalam satu mesin). Setelah instalasi Openstack selesai dilakukan, langkah berikutnya adalah dengan melakukan pembuatan *instance* didalam platform Openstack. *Instance* dibuat dengan menggunakan *image cloud* sistem operasi ubuntu. Setelah *instance* selesai dibuat, *instance* akan diinstall kedua aplikasi *web server* dan kemudian dilakukan pengambilan sampel data secara bergantian untuk masing-masing aplikasi *web server*, nginx dan apache, dan untuk

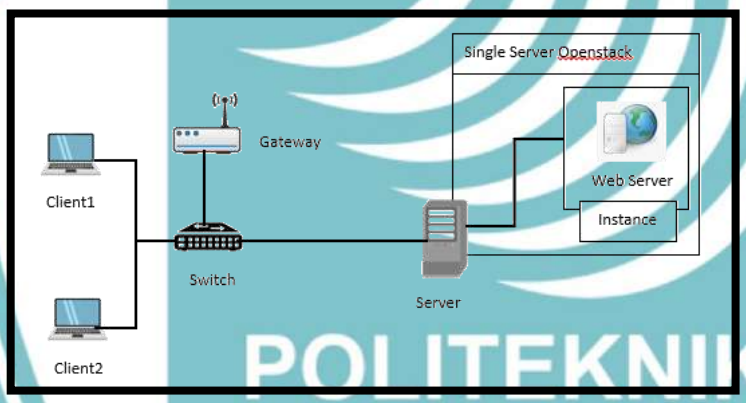
masing-masing skenario *cluster*. Kemudian *cluster instance* akan dialokasikan pada *pool* dari *load balancer* yang dibuat pada platform Openstack sebagai layanan LBaaS. Data yang didapat akan dibandingkan dan dianalisis untuk tiap-tiap skenario *cluster load balancer* dan data dari *single server*.

3.1.2 Desain Topologi Jaringan

Jaringan sistem akan dibangun berdasarkan topologi sebagai berikut :

1) Jaringan *single server* tanpa menggunakan *Load Balancer*.

Dalam topologi jaringan pada penelitian ini menggunakan 1 (satu) buah laptop yang telah terinstall sistem operasi ubuntu yang akan bertindak sebagai *server* Openstack, 2 (dua) buah laptop untuk *client* yang telah diinstall aplikasi siege sebagai *load testing* untuk *web server* dan satu buah switch sebagai penghubung antar laptop.



Gambar 3.3 Desain topologi *single server*

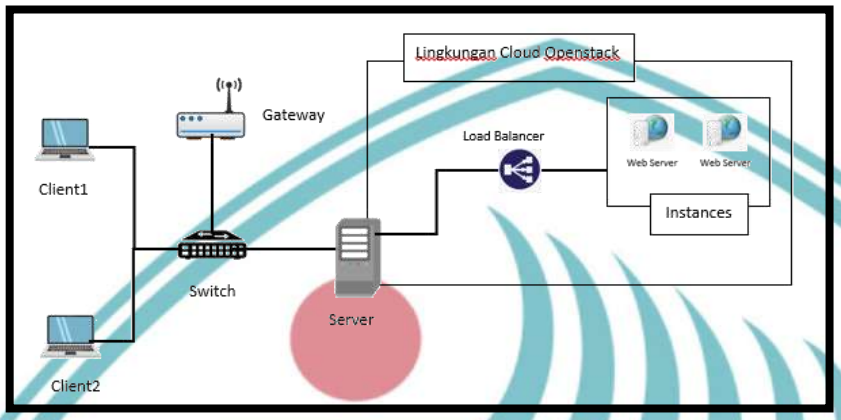
Keterangan :

1. *Server* merupakan satu kesatuan server Openstack yang terdiri dari *server* openstack *All in one single machine* (*controller* dan *compute node* dalam satu mesin) *Server* tersebut diinstall pada sistem operasi ubuntu yang berjalan pada laptop secara langsung.
2. *Client* 1 dan *Client* 2 bertindak sebagai *load tester* yang telah terinstall aplikasi siege di dalam sistem operasi linux mint yang berjalan diatas *virtual machine* VirtualBox.

2) Jaringan *multi server* dengan menggunakan *Load Balancer* pada *cluster* 2 *instance*.

Dalam topologi jaringan ini dilakukan implementasi LBaaS dari layanan platform Openstack, pada penelitian ini menggunakan 1 (satu) buah laptop

yang telah terinstall sistem operasi ubuntu sebagai *server* Openstack, 2 (dua) buah laptop untuk *client* yang telah diinstall aplikasi siege sebagai *load testing* untuk *web server* dan satu buah switch sebagai penghubung antar laptop.



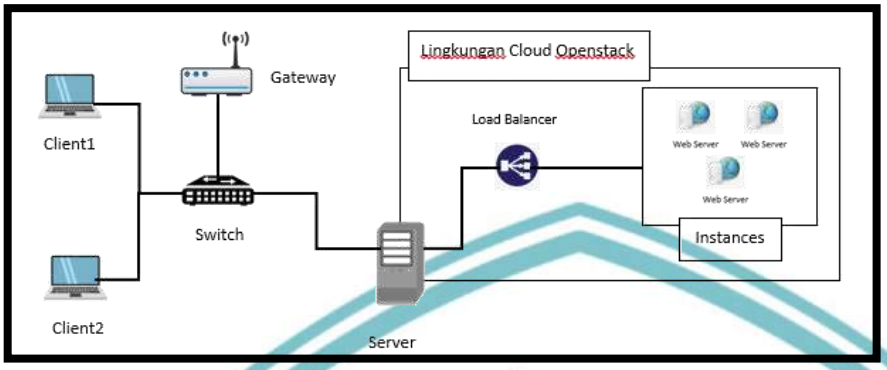
Gambar 3.4 Desain Topologi *Multi Server*

Keterangan :

1. *Server* merupakan satu kesatuan *server* Openstack yang terdiri *server* openstack *All in one single machine* (*controller* dan *compute node* dalam satu mesin). *Server* tersebut diinstall pada sistem operasi ubuntu yang berjalan pada laptop secara langsung.
2. *Client* 1 dan *Client* 2 bertindak sebagai *load tester* yang telah terinstall aplikasi siege di dalam sistem operasi linux mint yang berjalan diatas *virtual machine* VirtualBox.
3. Menggunakan 2 buah *intances* atau *virtual machine* yang telah diinstall layanan *web server* yang berada pada 1 *cluster* lingkungan Openstack.

3) Jaringan *multi server* dengan menggunakan *Load Balancer* pada *cluster* 3 *instance*.

Dalam topologi jaringan ini dilakukan implementasi LBaaS dari layanan platform Openstack, pada penelitian ini menggunakan 1 (satu) buah laptop yang telah terinstall sistem operasi ubuntu sebagai *server* Openstack, 2 (dua) buah laptop untuk *client* yang telah diinstall aplikasi siege sebagai *load testing* untuk *web server* dan satu buah switch sebagai penghubung antar laptop.



Gambar 3.5 Desain topologi menggunakan *Load Balancer*

Keterangan :

1. Server merupakan satu kesatuan server Openstack yang terdiri dari server openstack *All in one single machine* (*controlle*r dan *compute node* dalam satu mesin). Server tersebut diinstall pada sistem operasi ubuntu yang berjalan pada laptop secara langsung.
2. *Client* 1 dan *Client* 2 bertindak sebagai *load tester* yang telah terinstall aplikasi siege di dalam sistem operasi Ubuntu yang berjalan diatas *virtual machine* VMWare.
3. Menggunakan 3 buah *intances* atau *virtual machine* yang telah diinstall layanan *web server* yang berada pada 1 *cluster* lingkungan Openstack.

3.1.3 Spesifikasi Perangkat dan *Software/Tools*

Beberapa perangkat keras yang dibutuhkan dalam melakukan implementasi penelitian adalah sebagai berikut :

a. 1 buah Laptop sebagai *server* Openstack.

- *Prosesor* : Intel(R) Core(TM) i7
- *Memory* : 16 GB
- *Hardisk Drive* : 1 TB
- NIC : *Fast Ethernet*

b. 2 buah Laptop sebagai *client*.

- *Prosesor* : Intel(R) Core(TM) i3
- *Memory* : 8 GB
- *Hardisk Drive* : 500 GB
- NIC : *Fast Ethernet*

c. 1 Buah Switch Ethernet Mikrotik

d. Kabel UTP 4 buah.

Beberapa perangkat lunak yang dibutuhkan dalam melakukan implementasi penelitian adalah:

a. VirtualBox : *virtual machine*

b. Ubuntu 18.04 : sistem operasi *server* Openstack dan sistem operasi *web server*

c. Openstack (Versi Train) : platform *cloud computing*

d. Apache *Web Server* : aplikasi *web server* (versi Apache/2.4.29)

e. Nginx *Web Sever* : aplikasi *web server* (versi nginx/1.14.0)

f. Windows 10 : sistem operasi *client*

g. Siege : *load testing*

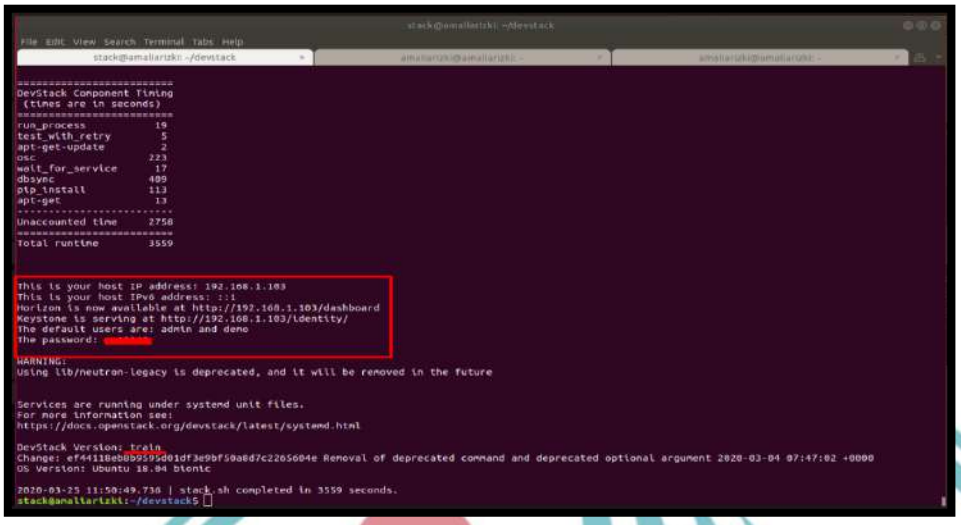
h. Htop : *load* cpu

3.2 Realisasi Sistem

Pada tahap ini akan dijelaskan gambaran mengenai instalasi dan konfigurasi sistem, dimulai dari instalasi dan konfigurasi platform Openstack, pembuatan *instance* atau *virtual machine* pada platform openstack dan juga implementasi layanan LBaaS pada Openstack.

3.2.1 Instalasi Openstack pada Ubuntu

Openstack yang akan digunakan adalah versi train yang dipasang dengan menggunakan devstack. Devstack merupakan serangkaian *script* yang dapat digunakan untuk memasang lingkungan Openstack baru ataupun memperluas lingkungan Openstack yang telah ada dengan cepat. Secara *default*, devstack akan memasang versi *upstream* (versi paling baru) Openstack melalui website git *source* mereka. Maka dari itu untuk *download script* devstack ini diperlukan git menuju *source* yang tersedia, yaitu “https://opendev.org/openstack/devstack”. Dengan menggunakan devstack seluruh servis yang ada di Openstack akan langsung terpasang.

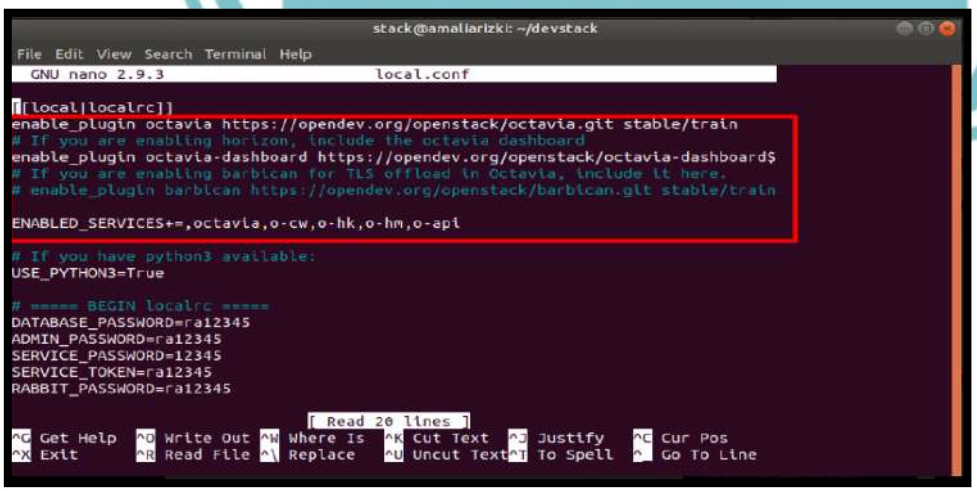


Gambar 3.6 Hasil akhir instalasi Openstack

Layanan LBaaS pada Openstack, yang bernama Octavia, dipasang dengan memasukkan baris yang ada di tabel 3.1 di berkas konfigurasi instalasi devstack bernama local.conf. Selain itu, berkas local.conf ini juga dapat digunakan untuk mengatur konfigurasi instalasi devstack, seperti penggunaan versi python dan *default password* untuk tiap layanan Openstack. Isi dari berkas local.conf dapat dilihat pada gambar 3.7.

Tabel 3.1 Konsfigurasi berkas local.conf

enable_plugin octavia https://opendev.org/openstack/octavia.git stable/train enable_plugin octavia-dashboard https://opendev.org/openstack/octavia-dashboard.git stable/train ENABLED_SERVICES+=,octavia,o-cw,o-hk,o-hm,o-api

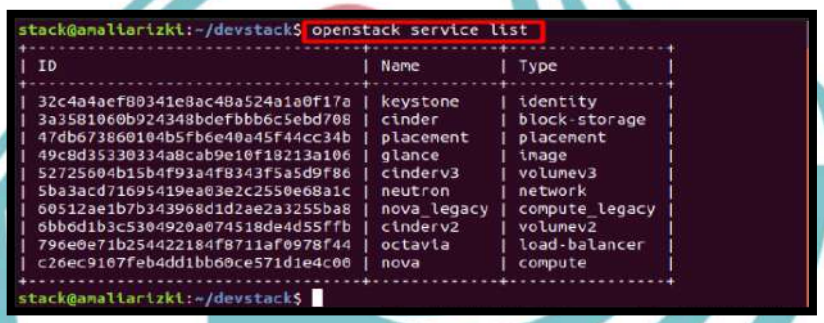


Gambar 3.7 Pendistribusian Octavia LBaaS pada Openstack

Setelah instalasi Openstack menggunakan devstack selesai. Langkah selanjutnya adalah dengan memverifikasi layanan openstack yang berhasil terpasang. Verifikasi layanan dapat dilakukan dengan me-*load* berkas yang bernama demo-openrc, kemudian menggunakan sintaks CLI Openstack sebagai berikut:

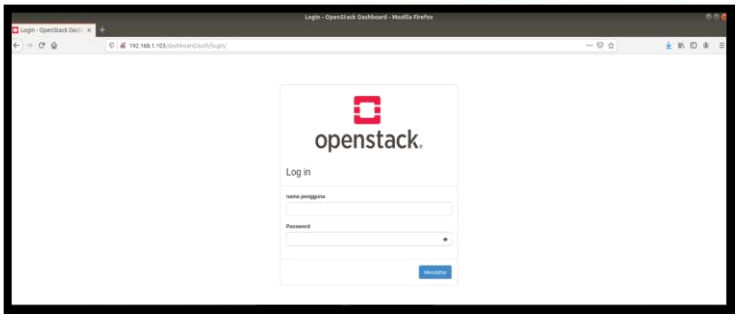
- *source demo-openrc*
- *openstack service list*

Hasil yang dimunculkan sintaks tersebut adalah list layanan Openstack yang telah terpasang, layanan tersebut adalah keystone, glance, nova, placement, cinder, neutron, horizon dan octavia.



Gambar 3.8 Daftar layanan openstack yang terpasang di mesin

Pengoperasian Openstack dapat dilakukan dengan dua acara. Cara yang pertama adalah melalui terminal dengan menggunakan *command* Openstack ataupun *command* dari masing masing layanan Openstack seperti *command* neutron, nova, cinder, dan lainnya. Cara yang kedua dapat dilakukan melalui *web dashboard* yang disajikan oleh layanan Openstack horizon. Horizon menyediakan tampilan *web* yang dapat berinteraksi dengan layanan Openstack lainnya. *Dashboard* Openstack dapat diakses melalui *web browser* dengan memasukan alamat horizon yang ditampilkan saat tahap instalasi selesai pada penelitian ini, yaitu "http://192.168.1.103/dashboard". IP *address* 192.168.1.103 merupakan alamat IP yang sudah dikonfigurasikan pada ubuntu sebelumnya.

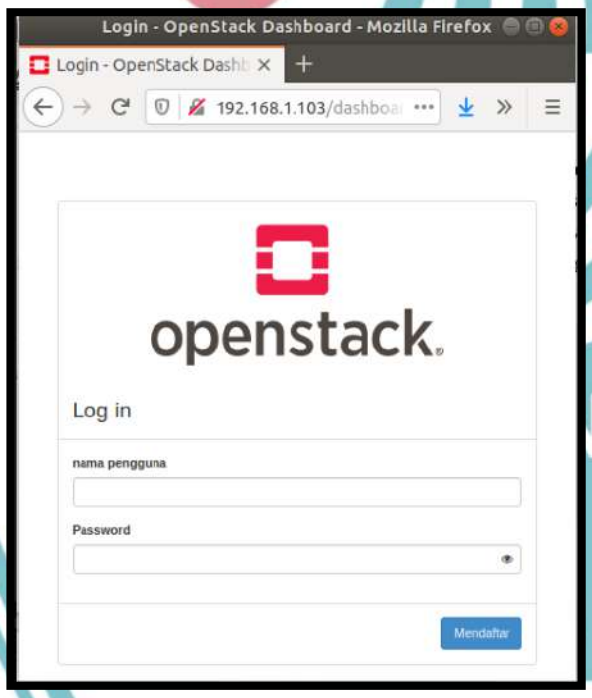


Gambar 3.9 *Login Dashboard* Openstack

3.2.2 Pembuatan *Virtual Machine* di Openstack

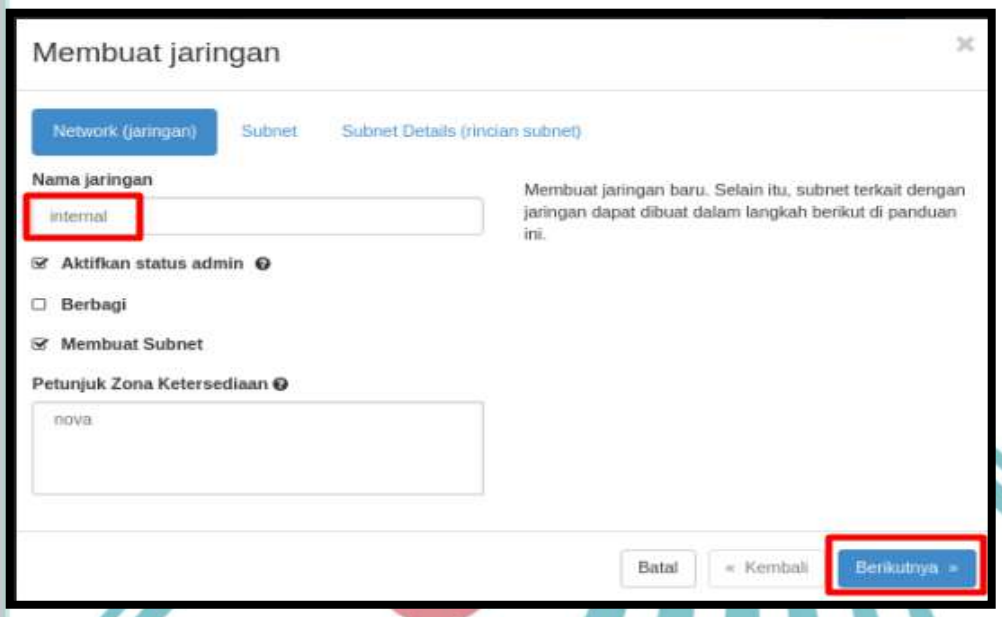
Setelah instalasi Openstack selesai. Langkah berikutnya adalah dengan membuat *instance* atau *virtual machine* menggunakan platform Openstack. Pembuatan dapat dilakukan melalui CLI ataupun melalui *dashboard web* horizon. Berikut adalah langkah yang dilakukan untuk melakukan pembuatan *virtual machine* didalam Openstack.

1. *Dashboard* Openstack diakses menggunakan *web browser* dengan alamat "http://192.168.1.103/dashboard". Kemudian login menggunakan *credential* yang dikonfigurasikan sebelumnya pada berkas local.conf dengan *username* admin.



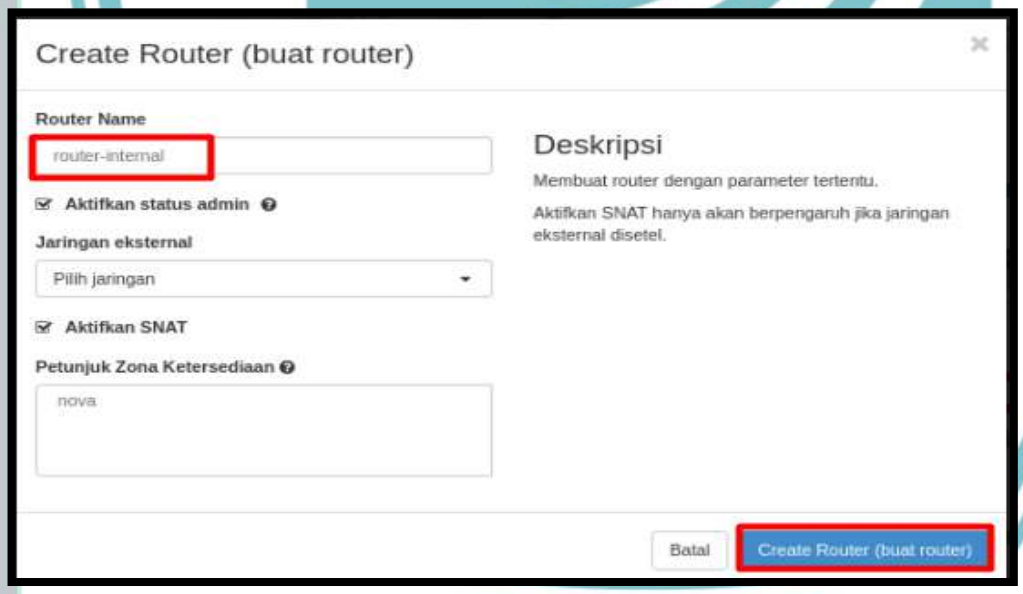
Gambar 3.10 *Login Dashboard* Openstack

2. Saat *login* berhasil, hal selanjutnya adalah melakukan konfigurasi jaringan internal sebagai *private network* sebagai jaringan lokal *instance* dan sebagai penghubung ke jaringan publik Openstack. Openstack sudah memiliki jaringan publik yang dibuat pada tahap instalasi Openstack. Jaringan internal dapat dibuat pada menu "Project" > "Network/Jaringan" > "Networks" dan subnet ditambahkan pada *network* yang telah di buat.



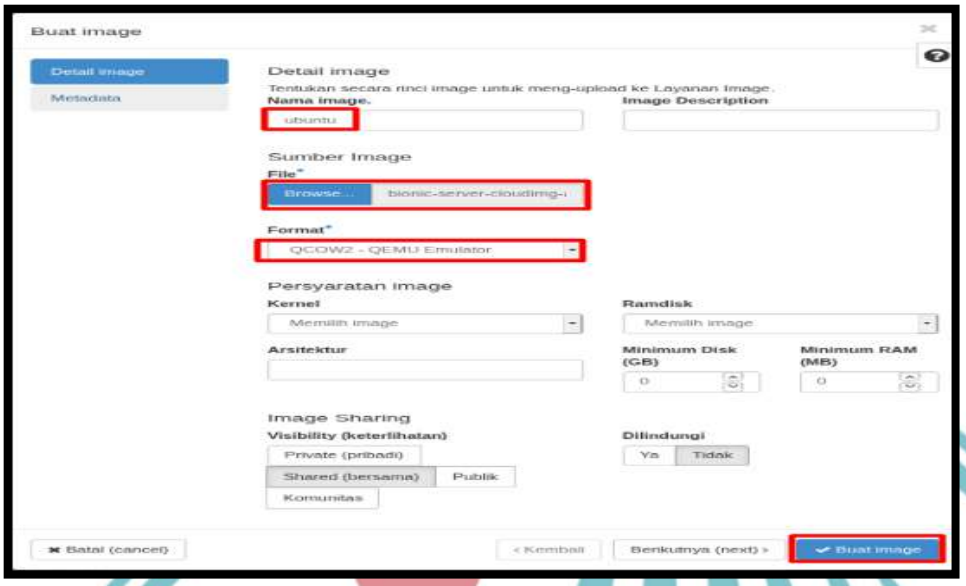
Gambar 3.11 Pembuatan jaringan internal di Openstack

3. Setelah jaringan internal dibuat, selanjutnya adalah membuat virtual router untuk menghubungkan jaringan internal *instance* dengan jaringa publik Openstack. Virtual router dapat dibuat pada menu "Project" > "Network/Jaringan" > "Router".



Gambar 3.12 Pembuatan router di Openstack

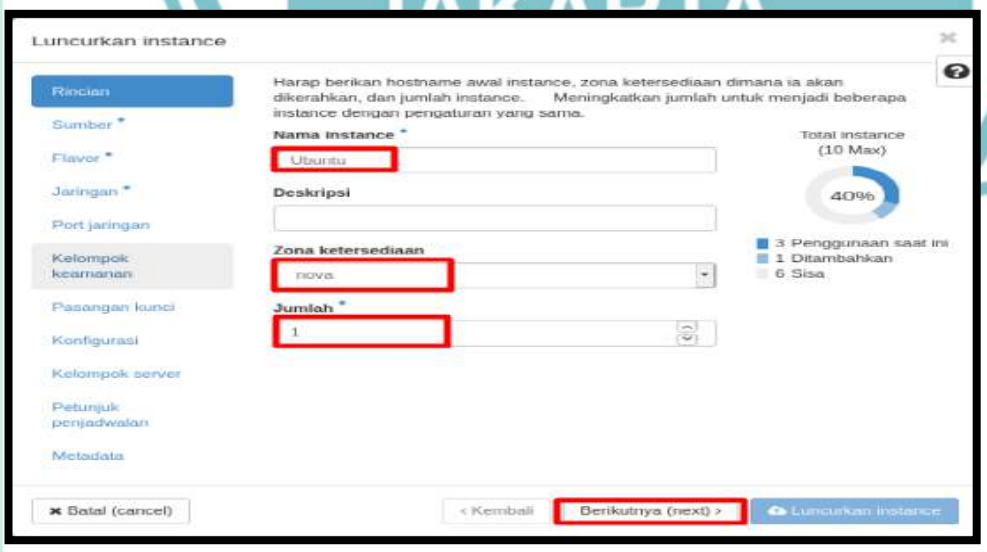
4. Selanjutnya adalah dengan menambahkan *image cloud* dari sistem operasi yang diinginkan ke dalam platform Openstack. Platform Openstack dapat menerima *image* file dengan format iso, raw, ataupun qcow2. *Image* yang ditambakan dapat digunakan sebagai dasar dari *instance* dan dapat diakses pada menu "Project" > "Compute" > "Images".



Gambar 3.13 Pembuatan *image* di Openstack

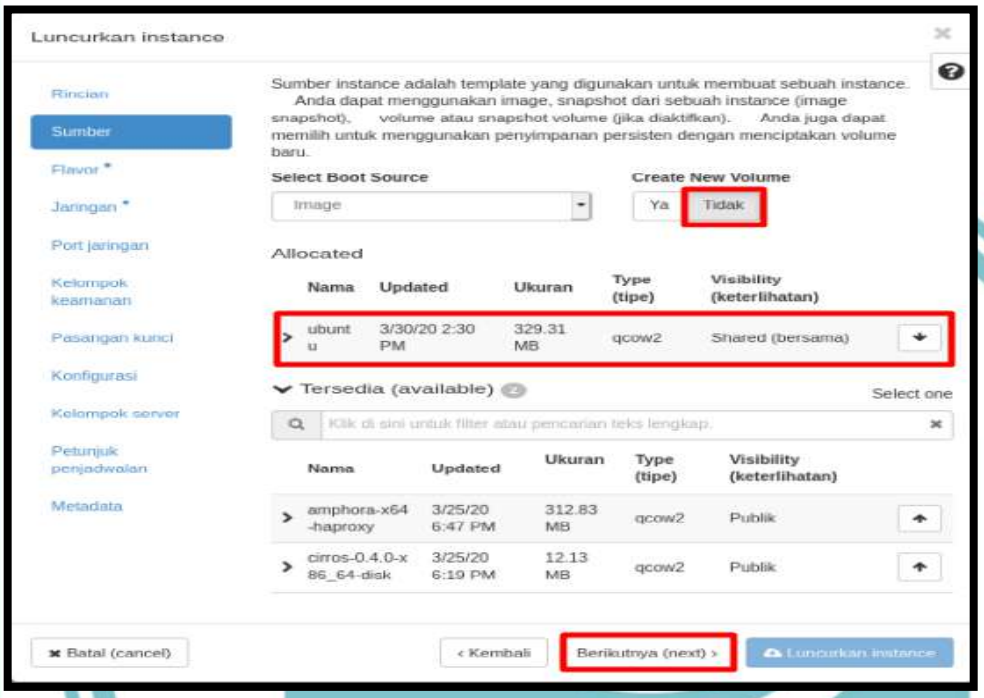
b. *Instance* pada Openstack dapat dibuat pada menu "Project" > "Compute" > "Instances". Sebelum *instance* dibuat, ada beberapa *resource* pada Openstack yang harus dibuat terlebih dahulu seperti jaringan internal dimana *instance* akan ditempatkan dan *image* yang digunakan sebagai dasar dari *instance*. Dalam pembuatan *intances,* langkah-langkah berikut dilakukan untuk membuat *instance virtual server* yang baru.

a. Nama *instance* dan banyaknya *instance* yang akan dibuat dimasukkan pada kolom yang tersedia. *Availibility zone* yang dipilih adalah nova. Karena nova merupakan layanan *compute* pada Openstack yang digunakan untuk mengatur *instance*.



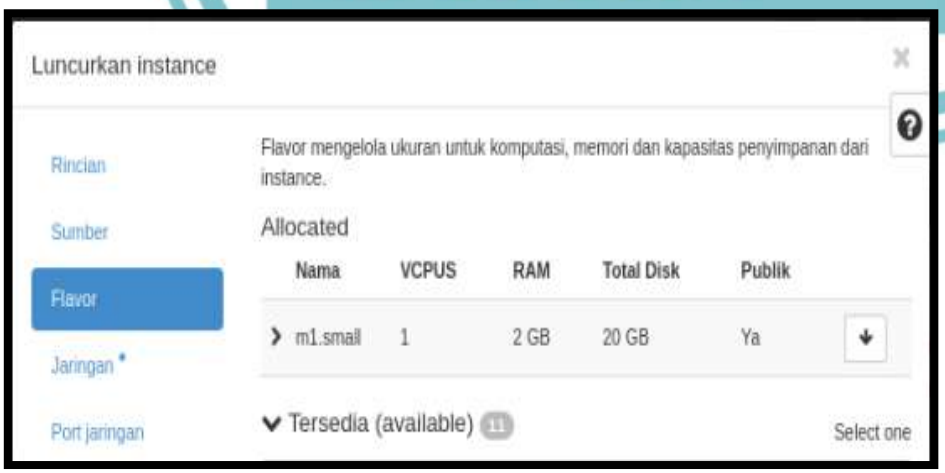
Gambar 3.14 Rincian pembuatan *instance*

b. Pemilihan sumber *boot* yang digunakan untuk membuat *instance*. Sumber yang dipilih adalah dari *image* dan *image* yang sebelumnya telah dibuat dipilih serta pilihan "tidak" dipilih untuk pembuatan *volume* baru.



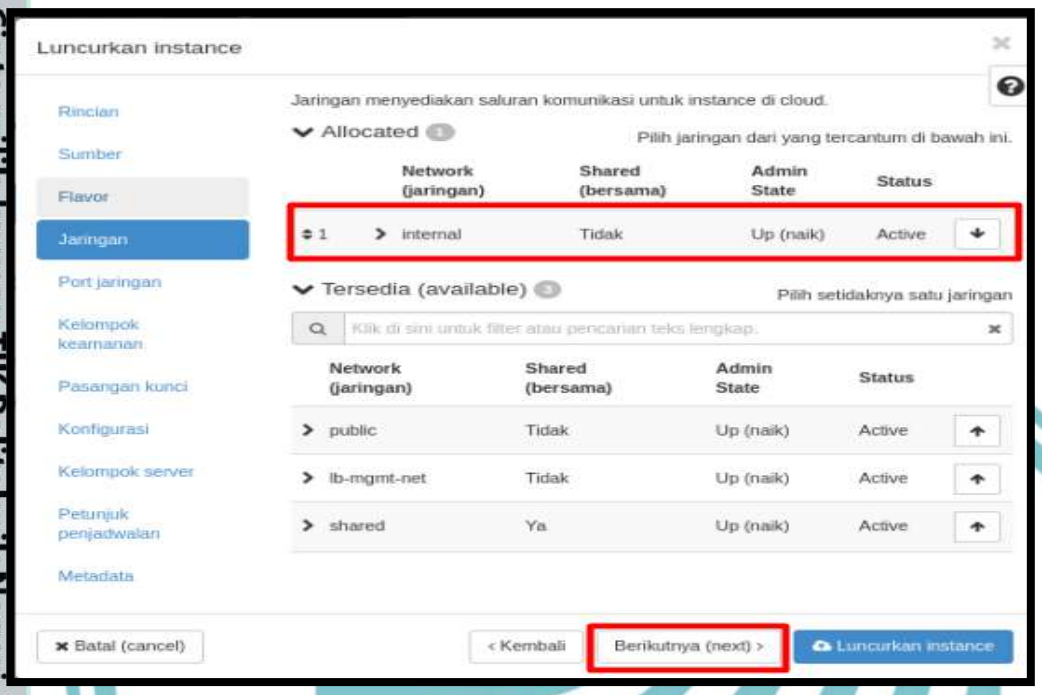
Gambar 3.15 Pemilihan sumber *instance* dalam pembuatan *instance*

c. Pemilihan *flavor* yang digunakan untuk mengatur ukuran atau spesifikasi dari *instance* yang dibuat. Pada penelitian ini, *flavor* yang dipilih adalah bawaan dari openstack yaitu m1.small dengan spesifikasi yang dapat dilihat pada gambar 3.16.



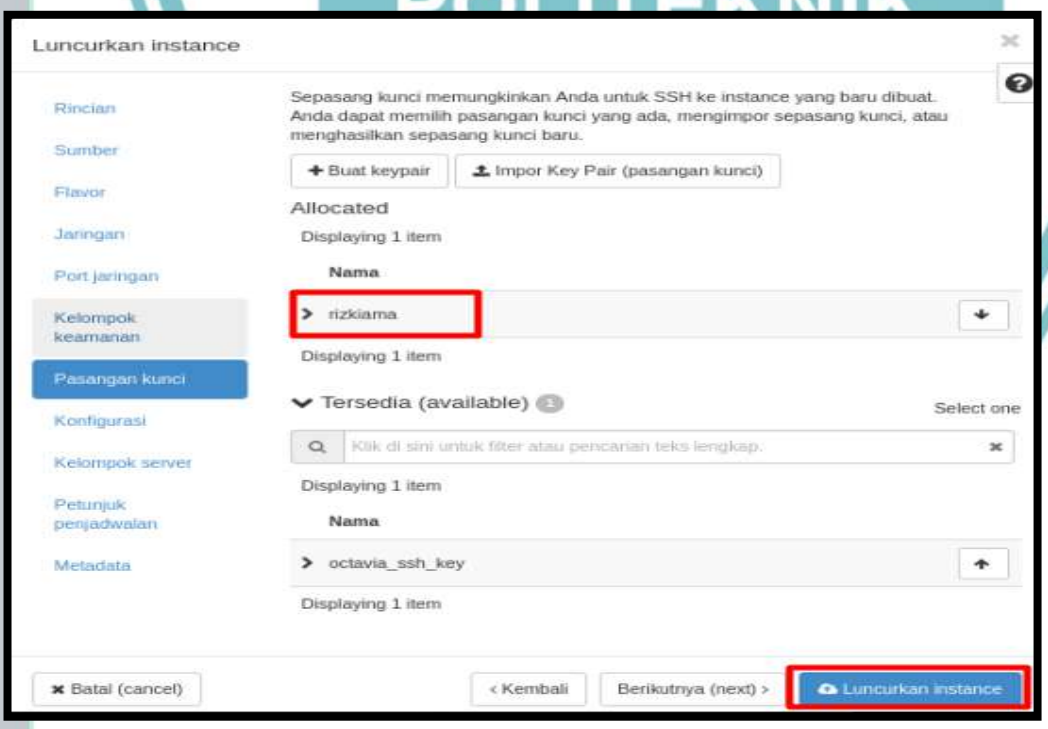
Gambar 3.16 Pemilihan *flavor* dalam pembuatan *instance*

f. Pemilihan jaringan internal/jaringan *private* yang akan digunakan oleh *instance*.



Gambar 3.17 Pemilihan jaringan dalam pembuatan *instance*

g. Penambahan dan pemilihan *key pair* yang dibuat dengan mengimport ssh key dari ubuntu yang sebelumnya dibuat dengan *command* "ssh-keygen" yang dapat digunakan sebagai akses *login* SSH menuju *instance*. Jika sudah selesai *instance* dapat diluncurkan dan *instance* baru tersebut sudah selesai dibuat.

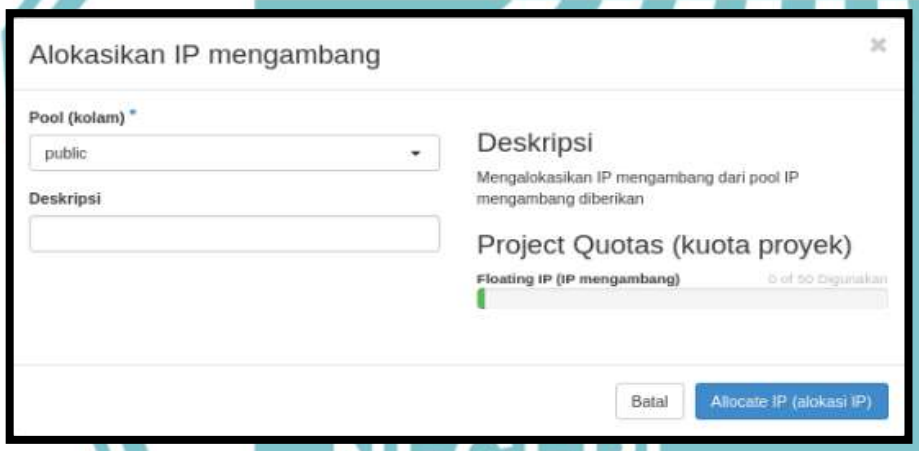


Gambar 3.18 Penambahan *key pair* dalam pembuatan *instance*

3.2.3 Pembuatan *Web Server*

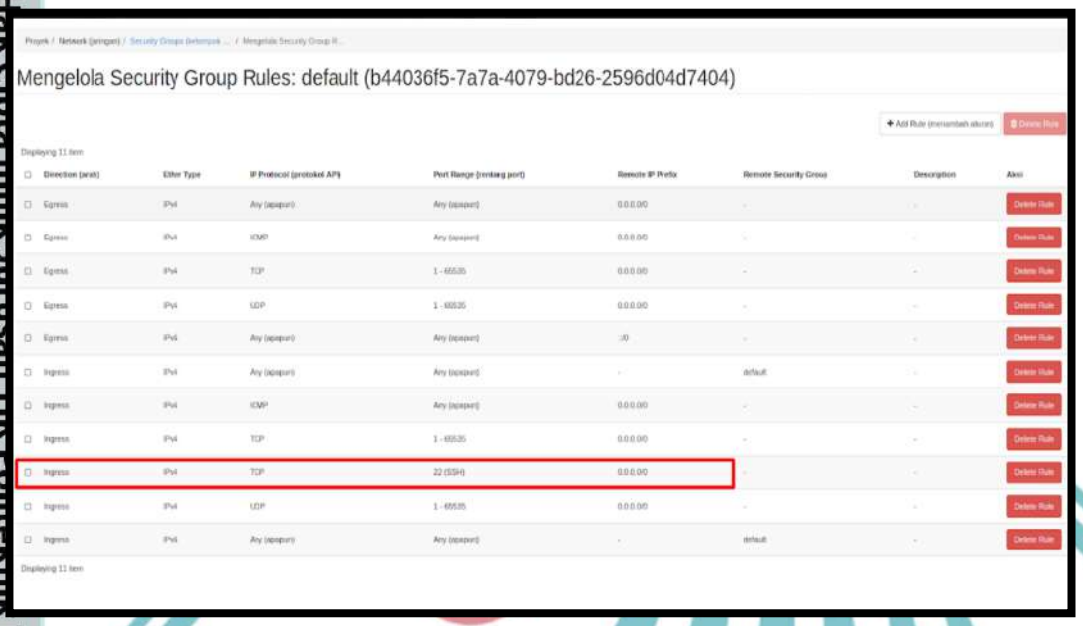
Setelah *instance* berhasil dibuat dan diluncurkan. Instalasi *web server* didalam *instance* menjadi Langkah selanjutnya. Berikut adalah langkah yang dilakukan dalam pembuatan *web server* pada *instance* Openstack.

a. Langkah pertama sebelum *web server* dipasang pada *instance* yaitu dengan melakukan *floating* IP, *floating* IP berfungsi agar *instance-instance* yang mempunyai IP *address* internal menjadi dapat diakses oleh jaringan luar (jaringan mesin server dan Internet). *Instance* akan mempunyai dua buah IP *address*. IP *address* pertama adalah IP *internal* yang didapat dari jaringan *private* yang dibuat di Openstack, dan yang kedua adalah IP *external* hasil dari *floating* IP yang diambil dari jaringan publik Openstack. Akses *floating* IP dilakukan pada menu “Project” > “Network” > “Floating IPs”.



Gambar 3.19 Pembuatan *Floating* IP

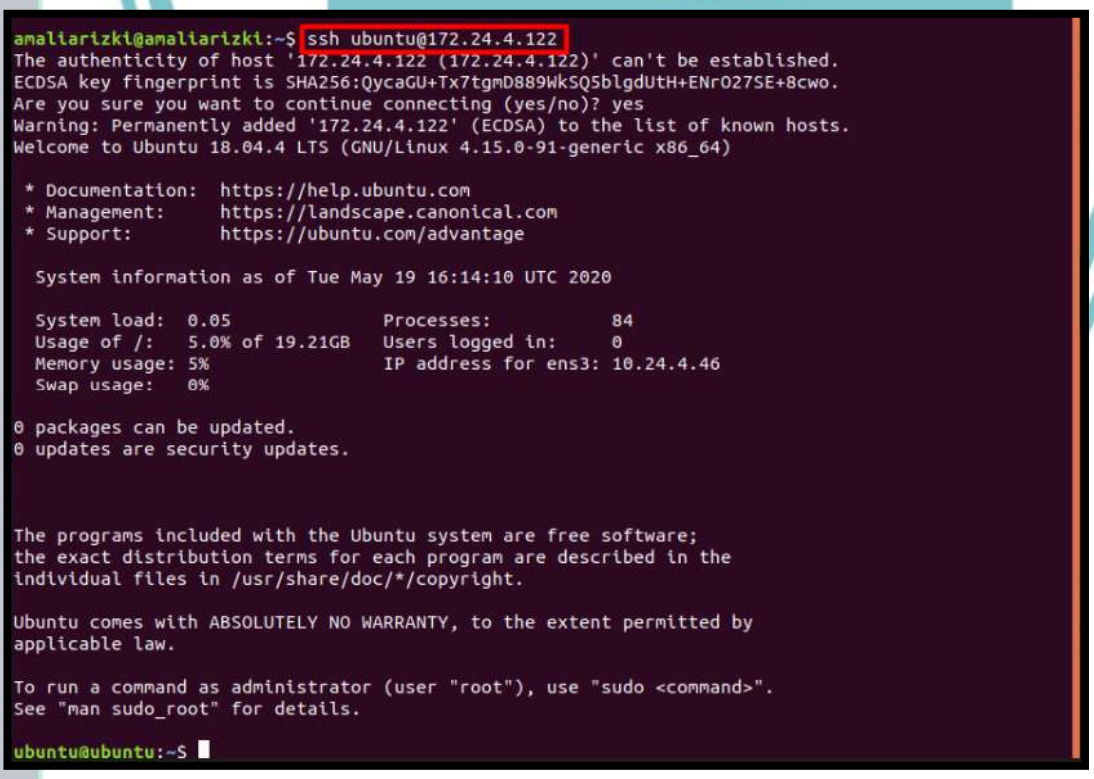
b. Langkah selanjutnya adalah dengan merubah isi *default security group* dari Openstack. *Security Group* di Openstack bertindak layaknya *firewall* pada sebuah sistem operasi, namun *security group* akan berlaku secara langsung saat *instance* baru dibuat tanpa harus melakukan konfigurasi *firewall* didalam *instance*. Perubahan *rules* dari *security group* akan berdampak lansung walaupun *instance* sudah diluncurkan dan dibuat sebelumnya dan hal ini dilakukan tanpa harus melakukan konfigurasi firewall didalam *instance* secara manual. Untuk penelitian ini, *rules* pada *security group* yang harus ditambahkan agar *instance* dan *web server* dapat diakses adalah dengan menambahkan *rule* TCP *ingress* pada port 22 sebagai akses *ssh instance* dan port 80 sebagai akses ke *web server*.



Gambar 3.20 Perubahan di *Security Group*

4. *Floating* IP sudah terbuat akan dapat di *associate* dengan *instance* yang sebelumnya sudah dibuat dan dijalankan. IP *address* yang didapatkan dari hasil *floating* IP dapat digunakan untuk akses ssh ke dalam *instance*. *Instance* dapat diremote dengan ssh dengan *command* berikut:

- ssh ubuntu@<*floating* IP *address*>



Gambar 3.21 Proses ssh ke *instance* Openstack melalui terminal

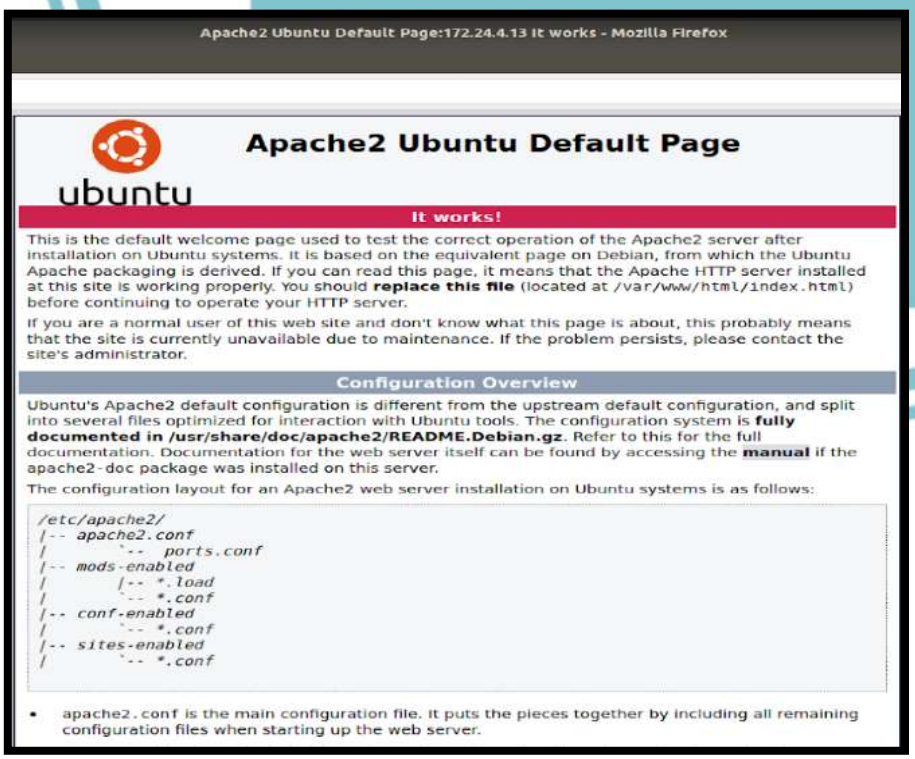
d. Setelah ssh dapat diakses dilanjutkan dengan meng-*update, upgrade* dan menginstal *web server* nginx atau apache lalu tunggu hingga prosesnya selesai.

```
ubuntu@ubuntu:~$ sudo apt update && sudo apt upgrade -y && sudo apt install nginx
```

Gambar 3.22 *Command update, upgrade,* dan *install web server*

e. Setelah proses instalasi selesai, *web server* dapat diakses di dalam *web browser* dengan menuliskan alamat *floating* IP *address instance*, jika berhasil akan menampilkan tampilan seperti gambar 3.23 untuk *web server* nginx dan gambar 3.24 untuk *web server* apache yang menandakan *web server* sudah bisa digunakan.

Gambar 3.23 Proses akses ke *web server* ngnix

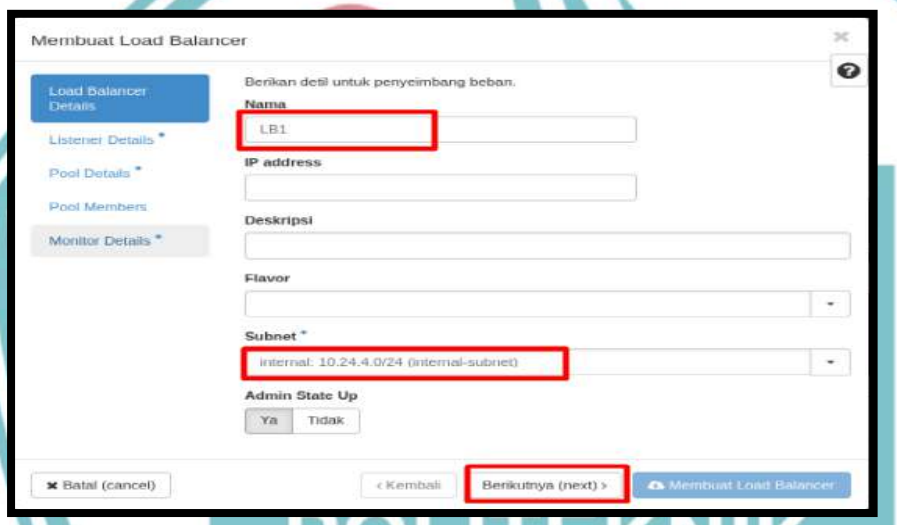


Gambar 3. 24 Proses akses ke *web server* apache

3.2.4 Pembuatan *Service Load Balancer* di Openstack

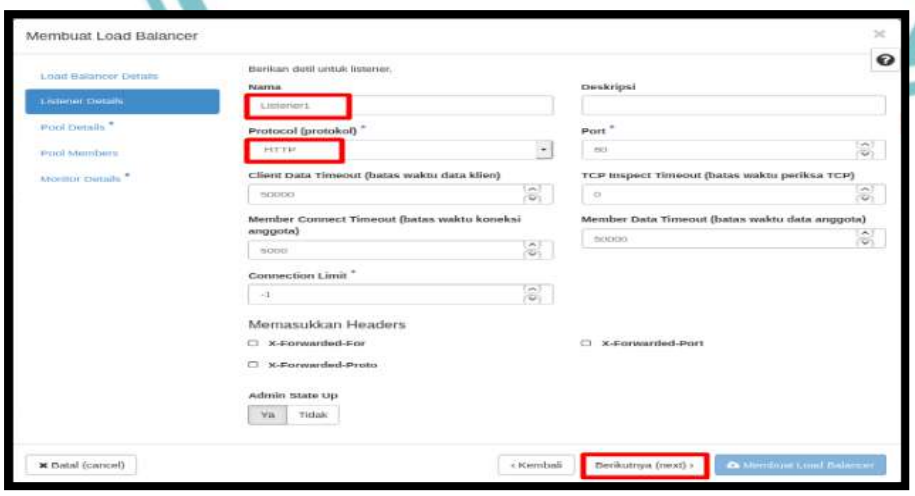
Setelah proses pembuatan *instance* dan instalasi *web server* berhasil. Pembuatan layanan *load balancer* dapat dilakukan untuk *cluster* 2 *instance* dan *cluster* 3 *instance*. Berikut adalah langkah yang dilakukan untuk melakukan pembuatan *load balancer* didalam Openstack.

a. Pembuatan *Load Balancing as a Service* (LbaaS) dapat dilakukan pada menu "Project" > "Network" > "Load Balancer". Pada tahap ini nama serta subnet harus di isi. Jaringan yang dipilih adalah jaringan *internal* dimana *cluster instance* berada pada pengisian subnet.



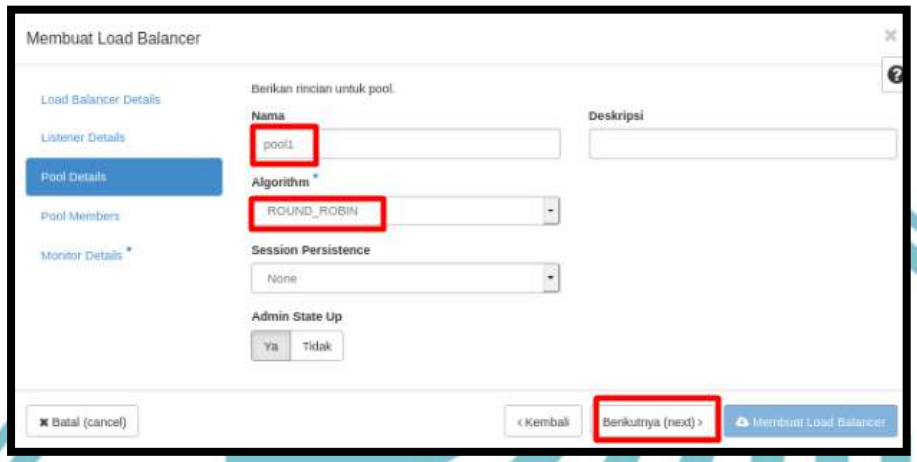
Gambar 3.25 Pembuatan *load balancer*

b. Pada bagian *Listener Details*, nama dimasukkan dan *protocol* HTTP dipilih untuk melakukan pengujian terhadap *web server. Listener* berfungsi sebagai pendengar dan penangaanan permintaan pada beberapa *port* dari *load balancer*. Dan masing-masing *port* ditentukan oleh *listener*.



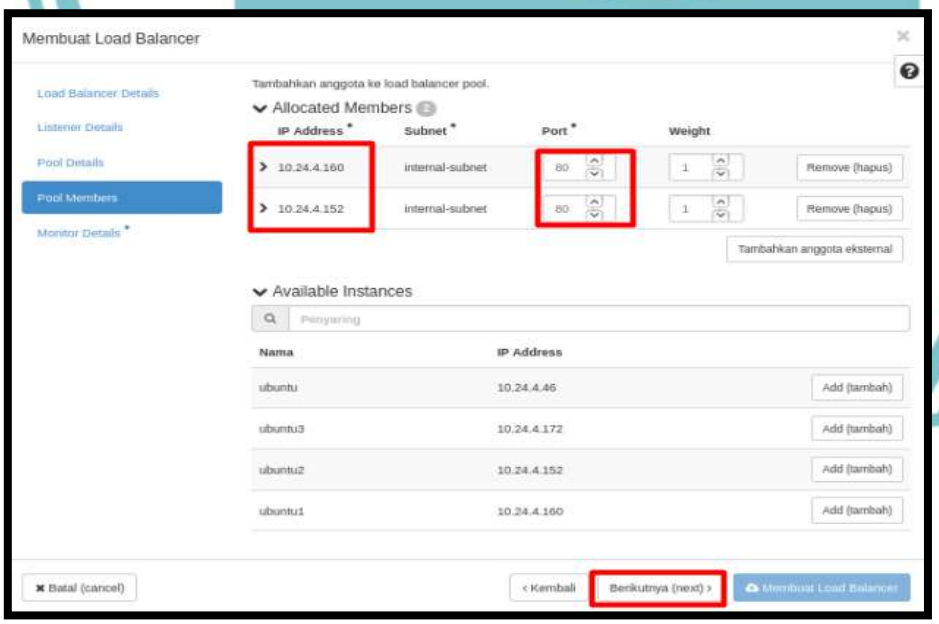
Gambar 3.26 *Listener details load balancer*

c. *Pool details* berfungsi sebagai pemilihan algoritma yang akan digunakan dalam percobaan. Terdapat 3 (tiga) algoritma yang bisa dipilih yaitu *round robin, least connection, dan source IP.*



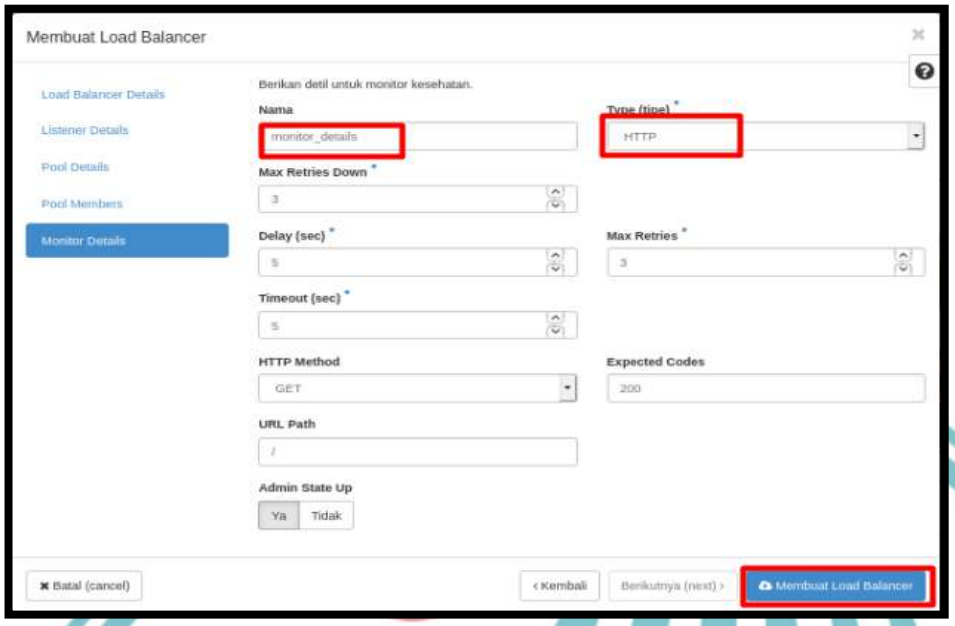
Gambar 3.27 *Pool details load balancer*

d. Selanjutnya adalah pemilihan member dari *pool*, *Pool member* merupakan daftar *instance* yang dijadikan anggota atau *pool* untuk kebutuhan *load balancing*. Dan *member* itu sendiri adalah *server* yang melayani *traffic* di belakang *load balancer*.



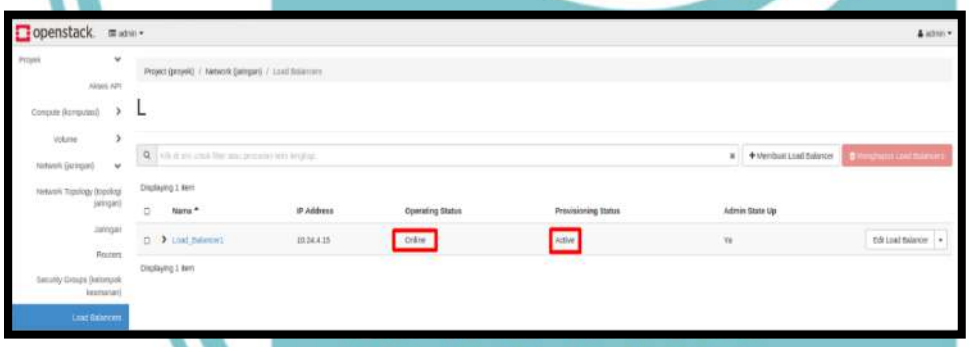
Gambar 3.28 *Pool member load balancer*

e. Pada bagian *Monitor Details*, nama dimasukkan dan tipe *protocol* HTTP dipilih dan selanjutnya *Load Balancer* dapat dibuat.



Gambar 3.29 *Monitor details load balancer*

f. Setelah selesai diluncurkan dan *load balancer* perlu beberapa saat sebelum statusnya menjadi aktif dan bisa di gunakan. Jika sudah aktif *load balancer* dapat di *associate* dengan *floating* IP agar dapat diakses oleh *client* dan dapat dikirimkan trafik dari masing-masing komputer *client*.



Gambar 3.30 *Load balancer* berhasil dibuat

3.3 *Troubleshoot* Openstack

Proses *troubleshoot* dilakukan ketika laptop dimatikan atau di *restart*. Saat laptop *server* Openstack dimatikan atau di *restart*, ada beberapa *service* dan *interface* Openstack harus di konfigurasi ulang agar Openstack bekerja dengan semestinya. *Command* ifconfig memiliki fungsi untuk mengaktifkan kembali *interface* supaya laptop bisa terhubung lagi ke dalam jaringan Openstack. Sedangkan *command* iptables berfungsi untuk membuka *firewall* dan akan meneruskan paket jaringan dari luar (jaringan router/wifi dan Internet) ke jaringan internal Openstack dan *instance* didalamnya.

```
stack@amaliarizki:~/devstack$ sudo ifconfig br-ex 172.24.4.1 netmask 255.255.255.0 up
stack@amaliarizki:~/devstack$ sudo ifconfig o-hm0 192.168.0.174 netmask 255.255.255.0 up
stack@amaliarizki:~/devstack$ sudo iptables -t nat -A POSTROUTING -o wlp2s0 -j MASQUERADE
```

Gambar 3.31 Proses *troubleshoot interface* Openstack

Tabel 3.2 *Command troubleshoot interface openstack*

sudo ifconfig br-ex 172.24.4.1 netmask 255.255.255.0 up sudo ifconfig o-hm0 192.168.0.174 netmask 255.255.255.0 up sudo iptables –t nat –A POSTROUTING –o wlp2s0/enp3s0f1 –j MASQUERADE

Untuk 3 (tiga) *command* iptables berikut berfungsi untuk membuka *port* dari *server* Openstack agar berkomunikasi ke *agent load balancer* Octavia. *Agent* Octavia ini diibaratkan sebagai *software monitoring* untuk memonitoring *load balancer* Octavia, yang memastikan apakah *agent* Octavia berhasil atau tidaknya mengaktifkan *load balancer* yang berada di Openstack. *Command* ip link set dev o-hm0 address *mac address* berfungsi untuk mengganti *mac address* yang telah terganti akibat proses *restart* menjadi *mac address* yang sama dengan agent *load balancer health monitor* di Openstack. *Coammand* systemctl restart devstack@* digunakan untuk memuat ulang seluruh *service* dan *daemon* Openstack devstack yang ada pada sistem, agar perubahan konfigurasi dapat diterapkan.

```
stack@amaliarizki:~/devstack$ sudo ip link set dev o-hm0 address fa:16:3e:06:39:44
stack@amaliarizki:~/devstack$ sudo systemctl restart devstack@*
stack@amaliarizki:~/devstack$ sudo iptables -I INPUT -i o-hm0 -p udp --dport 5555 -j ACCEPT
stack@amaliarizki:~/devstack$ sudo iptables -I INPUT -i o-hm0 -p udp --dport 10514 -j ACCEPT
stack@amaliarizki:~/devstack$ sudo iptables -I INPUT -i o-hm0 -p udp --dport 20514 -j ACCEPT
```

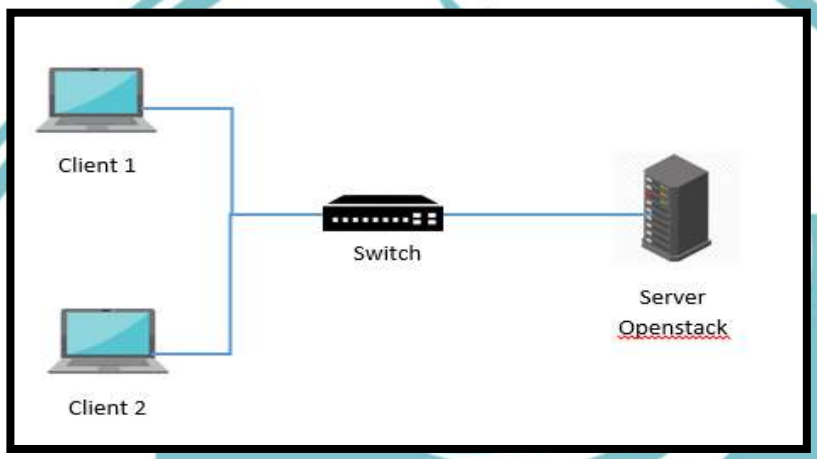
Gambar 3.32 *Proses troubleshoot firewall* Openstack

Tabel 3.3 *Command troubleshoot firewall* Openstack

sudo ip link set dev o-hm0 address fa:16:3e:06:39:44 sudo systemctl restart devstack@* sudo iptables –I INPUT –i o-hm0 –p udp - -dport 5555 –j ACCEPT sudo iptables –I INPUT –i o-hm0 –p udp - -dport 10514 –j ACCEPT sudo iptables –I INPUT –i o-hm0 –p udp - -dport 20514 –j ACCEPT

3.4 Skenario Pengujian

Pada tahap pengujian, hal yang dilakukan adalah menguji untuk memecahkan kendala-kendala agar mendapatkan nilai parameter yang diinginkan. Skenario pengujian ini bertujuan untuk mengukur kinerja layanan *Load Balancer* pada Openstack dengan layanan kedua jenis *web server* sehingga mendapatkan nilai parameter pengukuran. Parameter-parameter pengukuran pada penelitian ini mengacu kepada penelitian sebelumnya adalah seperti, *Throughput*, *Elapsed Time*, *Response Time*, *Transaction Rate*, *Data Transffered* dan *Load CPU*.



Gambar 3.33 Skenario Pengujian

Untuk mendapatkan nilai seperti *throughput*, *elapsed time*, *response time*, *transaction rate*, *data transffered* dan *load cpu* penelitian dilakukan dengan percobaan berupa 2 *client* membangkitkan HTTP *request* sebesar 50, 100, 200, 400, dan 500 *concurrent user* dan akan dilakukan sebanyak 26 (dua puluh enam) kali dari aplikasi siege kepada *web server* yang kemudian hasil yang didapat dari keluaran aplikasi siege dicatat dan dilakukan analisa perbandingan. Kemudian untuk mengetahui parameter *Load* CPU, penulis menggunakan aplikasi Htop yang diinstal pada masing-maasing *client* untuk melihat presentasi penggunaan CPU ketika dibangkitkan HTTP *request* oleh *client* melalui aplikasi siege.

BAB IV

HASIL DAN PEMBAHASAN

4.1 Pengujian

Pengujian ini dilakukan untuk mengevaluasi kinerja layanan *load balancing* pada Openstack yaitu LBaaS (*Load Balancing as a Service*) pada kinerja dua jenis *web server,* yaitu nginx dan apache. Pengujian ini mengukur dan memiliki hasil akhir berupa QoS.

4.2 Deskripsi Pengujian

Parameter-parameter pengujian QoS berikut digunakan sebagai parameter yang diamati dalam penelitian ini merujuk pada penelitian sebelumnya yang dilakukan oleh Hidayah, 2019. Parameter QoS tersebut adalah *elapsed time, data transferred, responds time, transaction rate, throughput* dan *load cpu*. Pengujian dilakukan dengan menggunakan 3 (tiga) skenario yaitu *single server,* dua *multi server* dengan menggunakan *load balancer,* dan tiga *multi server* dengan menggunakan *load balancer*. *Web server* yang digunakan yaitu nginx dan apache sebagai perbandingan dengan skenario kinerja penggunaan *load balancer*. Untuk mendapatkan nilai QoS, *clients* membangkitkan HTTP *request* sebesar 50, 100, 200, 400, dan 500 *concurrent user* dari aplikasi siege kepada *web server* yang kemudian hasilnya dapat dilihat dari keluaran aplikasi siege. *Tools* Siege ini dapat digunakan untuk menguji http/https. Pengujian pada tiap skenario dilakukan berulang sebanyak 26 (dua puluh enam) kali untuk mendapatkan hasil yang presisi. Pada skenario pengujian *multi-server* menggunakan *load balancer* menggunakan 3 (tiga) algoritma yang disediakan dari LBaaS Openstack yaitu *round robin, least connection* dan *source IP*.

4.3 Prosedur Pengujian

Pengujian dilakukan dengan membagi 1 laptop sebagai *server* dan 2 laptop sebagai *clients* untuk menguji *server* Openstack. Laptop yang berfungsi sebagai *server* telah diinstall Openstack dengan devstack dan menjadi *All-in-one server* dimana *controller* dan *compute node* Openstack dalam satu mesin tersebut. Setelah instalasi Openstack selesai dilakukan, langkah berikutnya adalah dengan melakukan pembuatan *instance* didalam platform Openstack. *Instance* dibuat dengan

menggunakan *image cloud* dari sistem operasi ubuntu. Setelah *instance* selesai dibuat, *instance* akan diinstall kedua aplikasi *web server* dan kemudian dilakukan pengambilan sampel data secara bergantian untuk masing-masing aplikasi *web server*, nginx dan apache sebagai perbandingan dengan skenario penggunaan *load balancer*. Diseluruh *clients* telah terdapat *tools* siege yang digunakaan untuk pengujian terhadap kinerja *web server*. *Tools* siege sudah dalam bentuk *executable* sehingga dapat langsung digunakan melalui *shell* atau *terminal* di sistem operasi Linux.

Pengujian terhadap *single server* dilakukan dengan mengirimkan HTTP *request* dari dua *clients* sebagai pengirim menuju *instance server* sebagai penerima untuk menguji *web* instance *server* apache dan nginx yang ada terpasang pada *instance* tersebut. HTTP *request* yang melalui aplikasi siege dikirimkan sebesar 50, 100, 200, 400, dan 500 *concurrent user* kepada *web server*. Percobaan pengiriman HTTP *request* melalui siege ini dilakukan sebanyak 26 (dua puluh enam) kali untuk mendapatkan hasil yang lebih presisi.

Pengujian terhadap 2 *multi server* dengan menggunakan *load balancer* dilakukan dengan menggunakan tiga algoritma dari LBaaS Openstack yaitu *round robin, least connection* dan *source* IP dengan mengirimkan HTTP *request* dari dua *clients* sebagai pengirim menuju *instance web server* sebagai penerima. HTTP *request* yang dikirimkan melalui aplikasi siege sebesar 50, 100, 200, 400, dan 500 kepada *web server concurrent user*. Percobaan pengiriman HTTP *request* melalui siege ini dilakukan sebanyak 26 (dua puluh enam) kali untuk mendapatkan hasil yang lebih presisi.

Sama dengan skenario pengujian 2 *multi server*. Pengujian terhadap 3 *multi server* dengan menggunakan *load balancer* juga dilakukan dengan menggunakan tiga algoritma LBaaS Openstack yaitu *round robin, least connection* dan *source* IP dengan mengirimkan HTTP *request* dari dua *clients* sebagai pengirim menuju *instance web server* sebagai penerima. HTTP *request* yang dikirimkan melalui aplikasi siege sebesar 50, 100, 200, 400, dan 500 kepada *web server concurrent user*. Percobaan pengiriman HTTP *request* melalui siege ini dilakukan sebanyak 26 (dua puluh enam) kali untuk mendapatkan hasil yang lebih presisi.

4.4 Data Hasil Pengujian

Data hasil pengujian terhadap layanan LBaaS pada *server* Openstack yang dibangun sesuai parameter QoS dan algoritma yang disebutkan pada sub-bab sebelumnya beserta prosedur pengujian.

4.4.1 Data *Throughput*

Data *throughput* yang didapat dari hasil percobaan pada *instance web* instance server apache menunjukan peningkatan nilai *throughput* pada seluruh skenario. Selain itu, nilai dari *single server* dan algoritma *source* IP memiliki nilai yang lebih rendah dibandingkan *round robin* dan *least connection*. Data tersebut dapat dilihat di tabel 4.1.

Tabel 4.1 *Rata-rata nilai* throughput *pada instance* server *apache*

RATA-RATA *THROUGHPUT SERVER* APACHE (MB/seconds)							
Jumlah *Concurrent User*	*Single Server*	*Load Balancer 2 Server Round Robin*	*Load Balancer 2 Server Least Connection*	*Load Balancer 2 Server Source IP*	*Load Balancer 3 Server Round Robin*	*Load Balancer 3 Server Least Connection*	*Load Balancer 3 Server Source IP*
50	0,571	0,723	0,704	0,574	0,671	0,665	0,583
100	0,645	0,7996	0,803	0,648	0,81	0,714	0,674
200	0,589	0,935	0,941	0,599	0,927	0,908	0,564
400	0,548	1,071	1,063	0,554	1,098	1,068	0,537
500	0,545	1,082	1,081	0,534	1,107	1,102	0,436

Data berikutnya ditunjukan untuk *throughput* pada *instance* dengan *web server* nginx, sama seperti instance *server* apache dimana terjadi peningkatan pada algoritma *round robin* dan *least connection* sedangkan nilai dari *single server* dan algoritma *source* IP memiliki nilai yang lebih rendah. Data ini dapat dilihat di tabel 4.2.

Tabel 4.2 *Rata-rata nilai* throughput *pada* server *nginx*

RATA-RATA *THROUGHPUT SERVER* NGINX (MB/seconds)							
Jumlah *Concurrent User*	*Single Server*	*Load Balancer 2 VM Round Robin*	*Load Balancer 2 VM Least Connection*	*Load Balancer 2 VM Source IP*	*Load Balancer 3 VM Round Robin*	*Load Balancer 3 VM Least Connection*	*Load Balancer 3 VM Source IP*
50	0,091	0,101	0,110	0,091	0,102	0,103	0,092
100	0,108	0,123	0,118	0,113	0,126	0,1302	0,109
200	0,099	0,149	0,138	0,097	0,130	0,132	0,095
400	0,089	0,163	0,159	0,085	0,153	0,152	0,083
500	0,081	0,168	0,160	0,08	0,165	0,166	0,079

4.4.2 Data *Elapsed Time*

Data *elapsed time* yang didapat dari semua hasil percobaan pada *instance web instance server* apache yang dirata-ratakan dalam tiap skenario percobaannya menunjukan tidak adanya perbedaan yang signifikan dari seluruh skenario. Sedangkan peningkatan nilai *concurrent user* menunjukkan pula peningkatan nilai *elapsed time*. Data tersebut dapat dilihat pada tabel 4.3.

Tabel 4.3 Rata-rata nilai *elapsed time pada instance server apache*

RATA-RATA *ELAPSED TIME SERVER* APACHE (*seconds*)							
Jumlah *Concurrent User*	*Single Server*	*Load Balancer 2 Server Round Robin*	*Load Balancer 2 Server Least Connection*	*Load Balancer 2 Server Source IP*	*Load Balancer 3 Server Round Robin*	*Load Balancer 3 Server Least Connection*	*Load Balancer 3 Server Source IP*
50	59,443	59,438	59,432	59,442	59,451	59,465	59,469
100	59,623	59,577	59,612	59,63	59,57	59,582	59,619
200	59,706	59,473	59,524	59,705	59,492	59,505	59,625
400	59,715	59,489	59,533	59,7202	59,447	59,572	59,7302
500	59,744	59,499	59,638	59,742	59,495	59,688	59,747

Data *elapsed time* yang didapat dari percobaan pada *instance web server* nginx yang telah dirata-ratakan dalam setiap skenario percobaanya menunjukan peningkatan nilai *elapsed time* untuk tiap peningkatan jumlah *concurrent user* dan rata-rata hasil yang didapatkan tidak menunjukkan adanya perbedaan yang signifikan. Data tersebut dapat dilihat di tabel 4.4.

Tabel 4.4 Rata-rata nilai *elapsed time pada* server *nginx*

RATA-RATA ELAPSED TIME SERVER NGINX (seconds)							
Jumlah *Concurrent User*	*Single Server*	*Load Balancer 2 Server Round Robin*	*Load Balancer 2 Server Least Connection*	*Load Balancer 2 Server Source IP*	*Load Balancer 3 Server Round Robin*	*Load Balancer 3 Server Least Connection*	*Load Balancer 3 Server Source IP*
50	59,398	59,345	59,38	59,399	59,373	59,388	59,399
100	59,689	59,479	59,536	59,6002	59,589	59,552	59,664
200	59,98	59,549	59,578	59,915	59,551	59,575	59,913
400	60,789	59,56	59,589	59,945	59,571	59,587	59,975
500	60,823	59,582	59,594	60,648	59,585	59,604	60,672

4.4.3 Data *Data Transferred*

Untuk parameter *data transferred* pada *instance web* instance *server* apache menunjukan peningkatan nilai *data transferred* untuk tiap peningkatan nilai jumlah *concurrent user*. Selain itu nilai *data transferred* dari algoritma *round robin* dan algoritma *least connection* menunjukkan nilai yang lebih besar dibandingkan *single server* dan algoritma *source* IP. Ditambah lagi terdapat kenaikan secara signifikan ketika jumlah *concurrent user* dari 200 ke 400. Data ini dapat dilihat di tabel 4.5.

Tabel 4.5 Rata-rata nilai *data transferred* pada *instance server apache*

RATA-RATA *DATA TRANSFERED SERVER* APACHE (MB)							
Jumlah *Concurrent User*	*Single Server*	*Load Balancer 2 Server Round Robin*	*Load Balancer 2 Server Least Connection*	*Load Balancer 2 Server Source IP*	*Load Balancer 3 Server Round Robin*	*Load Balancer 3 Server Least Connection*	*Load Balancer 3 Server Source IP*
50	48,445	48,717	48,749	48,467	48,873	48,89	48,48
100	48,489	48,79	48,633	48,471	48,822	48,946	48,592
200	49,859	55,751	56,024	49,605	55,147	54,207	49,584
400	55,564	63,775	63,203	58,786	65,441	63,701	56,699
500	58,454	64,358	64,4	58,899	65,551	65,793	58,146

Data transferred pada *instance web server* nginx menunjukan peningkatan nilai *data transferred* ketika nilai dari jumlah *concurrent user* meningkat, hal ini terjadi pada seluruh skenario. Nilai *data transferred* dari algoritma *round robin* dan algoritma *least connection* memiliki nilai yang lebih besar dibandingkan dengan *single server* dan algoritma *source* IP untuk setiap nilai jumlah *concurrent user*. Data ini dapat dilihat di tabel 4.6.

Tabel 4.6 Rata-rata nilai *data transferred* pada *serve*r nginx

RATA-RATA *DATA TRASFERED SERVER* NGINX (MB)							
Jumlah *Concurrent User*	*Single Server*	*Load Balancer 2 Server Round Robin*	*Load Balancer 2 Server Least Connection*	*Load Balancer 2 Server Source IP*	*Load Balancer 3 Server Round Robin*	*Load Balancer 3 Server Least Connection*	*Load Balancer 3 Server Source IP*
50	10,542	17,492	16,422	10,679	17,738	16,162	10,476
100	12,854	18,394	16,963	12,782	17,794	17,759	12,493
200	13,875	19,709	18,264	13,969	18,886	17,783	13,562
400	14,865	20,049	19,498	14,746	20,15	19,054	14,903
500	15,789	20,503	19,575	15,098	20,638	19,918	15,602

4.4.4 Data *Responds Time*

Data untuk parameter *responds time* pada *instance web* server apache yang telah didapat secara umum menunjukan peningkatan nilai saat jumlah *concurrent user* juga meningkat untuk setiap skenario. Selain itu, algoritma *round robin* dan algoritma *least connection* menunjukkan memiliki nilai *responds time* yang lebih rendah dibandingkan dengan *single server* dan algoritma *source* IP. Data ini dapat dilihat di tabel 4.7.

Tabel 4.7 Rata-rata nilai *responds time* pada *instance server* apache

RATA-RATA *RESPONDS TIME SERVER* APACHE (*seconds*)							
Jumlah *Concurrent User*	*Single Server*	*Load Balancer 2 Server Round Robin*	*Load Balancer 2 Server Least Connection*	*Load Balancer 2 Server Source IP*	*Load Balancer 3 Server Round Robin*	*Load Balancer 3 Server Least Connection*	*Load Balancer 3 Server Source IP*
50	0,115	0,105	0,106	0,118	0,116	0,117	0,126
100	0,205	0,191	0,199	0,207	0,189	0,198	0,206
200	0,33	0,319	0,320	0,332	0,324	0,329	0,333
400	0,571	0,541	0,550	0,570	0,534	0,543	0,577
500	0,680	0,654	0,665	0,689	0,652	0,652	0,686

Data berikutnya ditunjukkan untuk data *responds time* pada *instance web server* nginx juga menunjukan peningkatan nilai *responds time* pada tiap skenario percobaan saat jumlah *concurrent user* juga meningkat. Selain itu, nilai *responds time* yang didapat dari algorima *round robin* dan algoritma *least connection* menunjukkan nilai yang lebih kecil dibandingkan dengan *single server* dan algoritma *source* IP pada setiap nilai jumlah *concurrent user*. Data ini dapat dilihat di tabel 4.8.

Tabel 4.8 Rata-rata nilai *responds time* pada *server* nginx

RATA-RATA *RESPONDS TIME SERVER* NGINX (*seconds*)							
Jumlah *Concurrent User*	*Single Server*	*Load Balancer 2 Server Round Robin*	*Load Balancer 2 Server Least Connection*	*Load Balancer 2 Server Source IP*	*Load Balancer 3 Server Round Robin*	*Load Balancer 3 Server Least Connection*	*Load Balancer 3 Server Source IP*
50	0,105	0,084	0,091	0,106	0,097	0,098	0,111
100	0,173	0,146	0,151	0,178	0,156	0,152	0,181
200	0,334	0,263	0,284	0,352	0,291	0,297	0,386
400	0,505	0,449	0,476	0,511	0,491	0,489	0,521
500	0,606	0,574	0,583	0,612	0,569	0,564	0,617

4.4.5 Data *Transaction Rate*

Data untuk parameter *transaction rate* pada *instance web instance server* apache menunjukan peningkatan nilai *transaction rate* pada setiap skenario percobaan untuk tiap meningkatnya jumlah *concurrent user*. Nilai *transaction rate* dari algoritma *round robin* dan algoritma *least connection* menunjukkan nilai yang lebih besar dibandingkan dengan *single server* dan algoritma *source* IP. Data ini dapat dilihat di tabel 4.9.

Tabel 4.9 Rata-rata nilai *transaction rate* pada *instance server* apache

RATA-RATA *TRANSACTION RATE SERVER* APACHE (*trans/sec*)							
Jumlah *Concurrent User*	*Single Server*	*Load Balancer 2 Server Round Robin*	*Load Balancer 2 Server Least Connection*	*Load Balancer 2 Server Source IP*	*Load Balancer 3 Server Round Robin*	*Load Balancer 3 Server Least Connection*	*Load Balancer 3 Server Source IP*
50	223,525	224,75	224,901	223,397	224,783	225,844	222,587
100	235,456	265,625	266,527	258,967	269,53	235,318	224,192
200	297,456	310,894	312,835	298,255	307,907	304,468	299,922
400	313,74	362,337	348,0504	317,91	365,362	355,223	320,091
500	319,25	413,461	389,102	329,093	410,079	399,269	325,355

Data berikutnya ditunjukan untuk *transaction rate* pada *instance web server* nginx juga menunjukan peningkatan untuk setiap skenario percobaan saat jumlah *concurrent user* meningkat. Selain itu, sama seperti apache algoritma *round robin* dan algoritma *least connection* menunjukkan nilai yang lebih besar dibandingkan dengan *single server* dan algoritma *source* IP. Data ini dapat dilihat di tabel 4.10.

Tabel 4.10 Rata-rata nilai *transaction rate* pada *server* nginx

RATA-RATA *TRANSACTION RATE SERVER* NGINX (*trans/sec*)							
Jumlah *Concurrent User*	*Single Server*	*Load Balancer 2 Server Round Robin*	*Load Balancer 2 Server Least Connection*	*Load Balancer 2 Server Source IP*	*Load Balancer 3 Server Round Robin*	*Load Balancer 3 Server Least Connection*	*Load Balancer 3 Server Source IP*
50	249,804	312,904	353,865	251,001	320,716	371,776	251,103
100	296,043	370,711	308,279	295,182	371,847	335,84	230,86
200	245,986	392,926	365,379	248,76	395,723	341,53	250,185
400	307,24	425,431	420,003	325,853	428,863	397,473	335,825
500	383,046	443,043	442,388	391,494	451,614	436,217	399,62

4.4.6 Data *Load* CPU

Data untuk parameter *load cpu* pada *client* saat akses *server* apache menunjukan peningkatan nilai *load cpu* pada setiap skenario percobaan untuk tiap meningkatnya jumlah *concurrent user*. Nilai *load cpu* dari algoritma *round robin* dan algoritma *least connection* menunjukkan nilai yang lebih kecil dibandingkan dengan *single server* dan algoritma *source* IP. Data ini dapat dilihat di tabel 4.11.

Tabel 4.11 Rata-rata nilai *load cpu* pada *client* saat akses *server* apache

RATA-RATA *LOAD CPU* PADA *CLIENT* SAAT AKSES *SERVER* APACHE (*load/minute*)							
Jumlah *Concurrent User*	*Single Server*	*Load Balancer 2 Server Round Robin*	*Load Balancer 2 Server Least Connection*	*Load Balancer 2 Server Source IP*	*Load Balancer 3 Server Round Robin*	*Load Balancer 3 Server Least Connection*	*Load Balancer 3 Server Source IP*
50	1,966	0,487	0,694	1,431	0,492	0,711	1,466
100	3,964	0,682	0,829	3,759	0,697	0,847	3,907
200	6,472	1,12	2,265	5,857	1,219	2,523	5,924
400	9,253	1,554	3,334	9,141	1,721	3,565	9,283
500	12,525	2,407	4,371	12,372	2,555	4,665	12,483

Data berikutnya ditunjukan untuk *load cpu* pada *client* saat akses *server* nginx juga menunjukan peningkatan untuk setiap skenario percobaan saat jumlah *concurrent user* meningkat. Selain itu, sama seperti apache algoritma *round robin* dan algoritma *least connection* menunjukkan nilai yang lebih kecil dibandingkan dengan *single server* dan algoritma *source* IP. Data ini dapat dilihat di tabel 4.12.

Tabel 4.12 Rata-rata nilai *load cpu* pada *client* saat akses *server* nginx

RATA-RATA *LOAD CPU* PADA *CLIENT* SAAT AKSES *SERVER* NGINX (*load/minute*)							
Jumlah *Concurrent User*	*Single Server*	*Load Balancer 2 Server Round Robin*	*Load Balancer 2 Server Least Connection*	*Load Balancer 2 Server Source IP*	*Load Balancer 3 Server Round Robin*	*Load Balancer 3 Server Least Connection*	*Load Balancer 3 Server Source IP*
50	1,573	0,433	0,698	1,451	0,429	0,714	1,478
100	3,925	0,679	0,834	3,774	0,668	0,848	3,798
200	5,967	1,118	2,353	5,882	1,108	2,423	5,923
400	9,322	1,567	3,453	9,177	1,548	3,479	9,256
500	12,513	2,484	4,515	12,446	2,437	4,573	12,562

4.5 Analisis Data

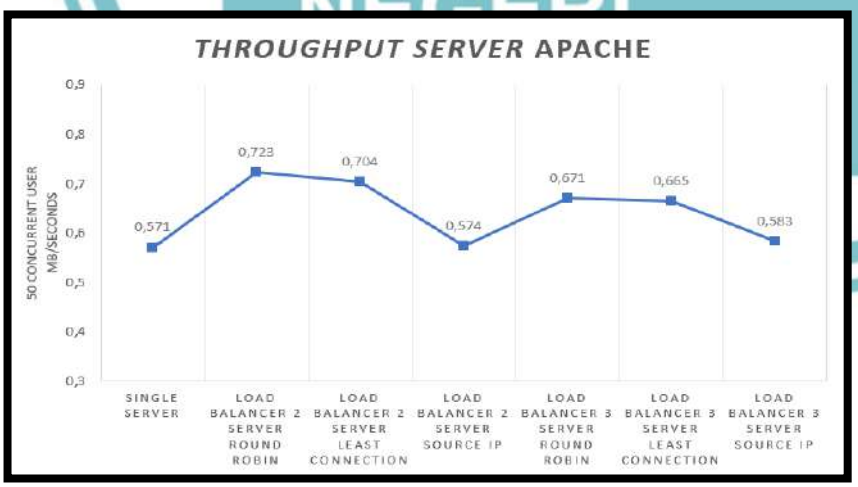
Data dari pengujian *single server* dan *load balancing* dianalisis dengan membandingkan data sebelum menggunakan *load balancer* dan setelah menggunakan *load balancer* dengan menggunakan skenario dan prosedur pengujian yang sama.

4.5.1 Analisis Data *Web Instance server* Apache

Data dari pengujian *single server* dan *multi server* dengan *load balancing* dianalisis dengan membandingkan hasil *throughput, elapsed time, data transferred, responds time, transaction rate* dan *load cpu* dari *single server* dan *load balancing* dengan menggunakan 2 *instances* dan 3 *instances* pada *instance server* apache.

4.5.1.1 Analisis *Throughput* Apache

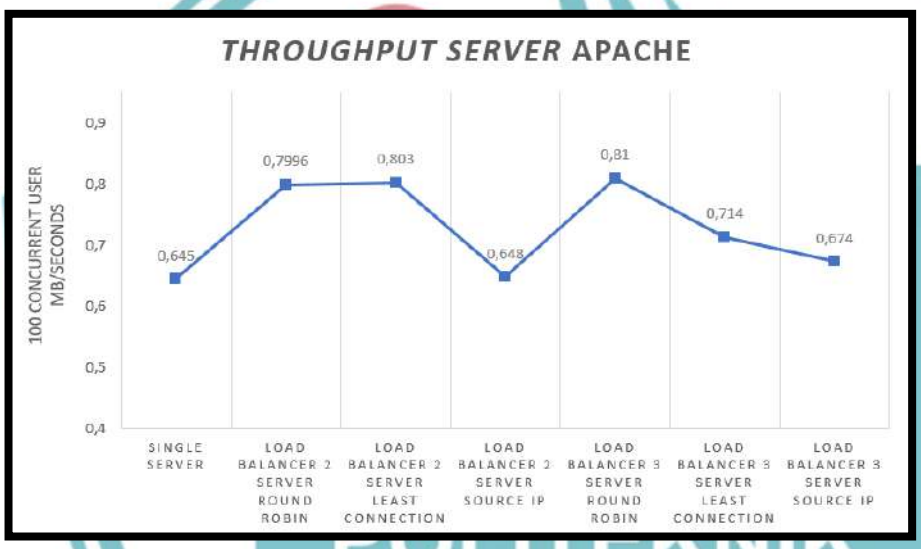
Dari grafik dibawah ini menunjukan perbandingan *throughput* pada *instance server* apache dengan HTTP *request* dari siege sebesar 50 *concurrent users*. Data yang didapat dari tiga pengujian skenario menunjukan bahwa algoritma *round robin* dan algoritma *least connection* memiliki nilai yang tidak berbeda jauh, akan tetapi jika dibandingkan dengan *single server* dan *source* IP memiliki perbedaan nilai yang cukup signifikan. Dari ketiga skenario yang diuji algoritma *round robin* menunjukan nilai yang paling tinggi dibandingkan dengan algoritma lain dan *single server*. Semakin tinggi nilai *throughput* maka performansi dari *web server* dinyatakan bagus. Data perbandingan dapat dilihat pada grafik di gambar 4.1.



Gambar 4.1 Grafik perbandingan *throughput* apache 50 *concurrent users*

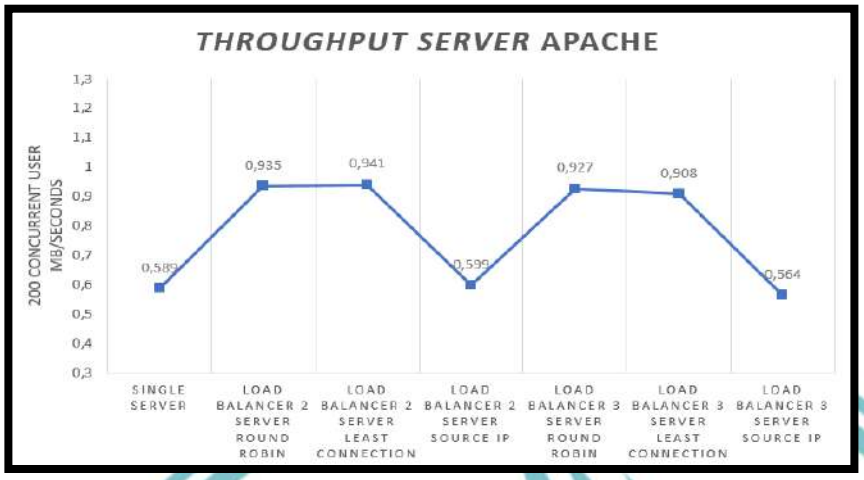
Data berikutnya menunjukan perbandingan *throughput* pada *instance server* apache dengan HTTP *request* dari siege sebesar 100 *concurrent users*. Data yang didapat

dari hasil pengujian tiga skenario menunjukan bahwa *single server* dan *source* IP memiliki nilai yang tidak berbeda jauh dan cenderung dibawah algoritma *round robin* dan *least connection*. Algoritma *least connection* menunjukan penurunan disaat skenario 3 instance dan algoritma *round robin* menunjukan peningkatan antara 2 *server* ke 3 *server*. Dari ketiga skenario yang diuji algoritma *round robin* menunjukan nilai yang paling tinggi dibandingkan dengan algoritma lain dan *single server*. Semakin tinggi nilai *throughput* maka performansi dari *web serv*er dinyatakan bagus. Data perbandingan dapat dilihat pada grafik di gambar 4.2.



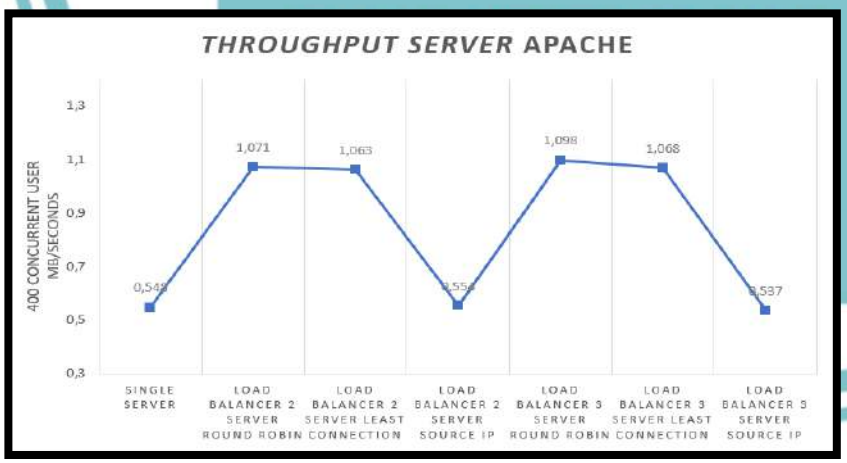
Gambar 4.2 Grafik perbandingan *throughput* apache 100 *concurrent users*

Grafik berikutnya menunjukan perbandingan *throughput* pada *instance server* apache dengan HTTP *request* dari siege sebesar 200 *concurrent users*. Algoritma *round robin* dan *least connection* menunjukan penurunan nilai tapi penurunan yang terjadi pada *round robin* tidak terlalu jauh berbeda dibandingkan dengan penurunan yang terjadi pada algoritma *least connection* menunjukan penurunan yang cukup signifikan. Sedangkan untuk *single server* dan *source* IP berada jauh dibawah algoritma *round robin* dan *least connection*. Dari ketiga skenario yang diuji algoritma *round robin* menunjukan nilai yang paling tinggi dibandingkan dengan algoritma lain dan *single server*. Semakin tinggi nilai *throughput* maka performansi dari *web server* dinyatakan bagus. Data perbandingan ini dapat dilihat pada grafik di gambar 4.3.



Gambar 4.3 Grafik perbandingan *throughput* apache 200 *concurrent users*

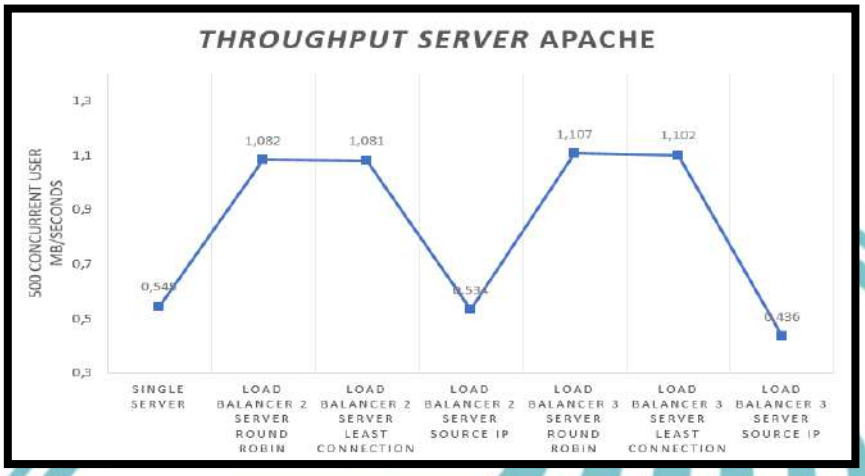
Grafik berikutnya menunjukan perbandingan *throughput* pada *instance server* apache dengan HTTP request dari siege sebesar 400 *concurrent users*. Algoritma *round robin* dan *least connection* menunjukan peningkatan antara pengujian 2 *server* ke 3 *server* sedangkan *single server* dan *source* IP menunjukan penurunan dari grafik sebelumnya. Dari ketiga skenario yang diuji algoritma *round robin* menunjukan nilai yang paling tinggi dibandingkan dengan algoritma lain dan *single server*. Semakin tinggi nilai *throughput* maka performansi dari *web server* dinyatakan bagus.



Gambar 4.4 Grafik perbandingan *throughput* apache 400 *concurrent users*

Grafik berikutnya menunjukan perbandingan *throughput* pada *instance server* apache dengan HTTP request dari siege sebesar 500 *concurrent users*. Semakin meningkatnya nilai *concurrent users*, *single server* maupun *source* IP tidak mampu untuk menerima beban *request* dari *client* karena tidak ada pembagian *request* kepada *server* yang lainnya. Sebaliknya algoritma *round robin* dan *least connection*

menunjukan peningkatan dikarenakan dapat membagi jumlah beban *request* dari *client* sehingga menunjukan *load balancing* lebih unggul dibandingkan *single server* dan *source* IP. Data perbandingan ini dapat dilihat pada grafik di gambar 4.5



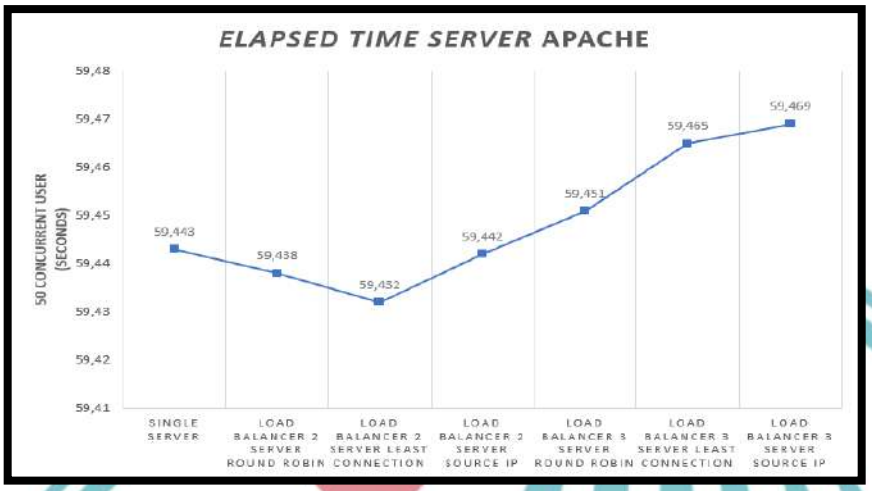
Gambar 4.5 Grafik perbandingan *throughput* apache 500 *concurrent users*

Dari data keseluruhan grafik pengujian *throughput* pada *instance server* apache diatas menunjukan perbandingan antara *single server* dengan *load balancing* dimana grafik dari *single server* dan *source* IP mulai menunjukan penurunan yang cukup signifikan ketika menerima beban *request* sebanyak 200 *request*. Terjadinya penurunan diakibatkan *single server* maupun *source* IP tidak mampu menangani beban dikarenakan hanya memiliki satu *server* dan tidak bisa membagi beban *request* ke *server* lainnya. Performansi dikatakan bagus jika nilai *throughput* menunjukan peningkatan. Algoritma *round robin* dan *least connection* menunjukan peningkatan dikarenakan dapat membagi jumlah beban *request* kepada *web server* lainnya.

4.5.1.2 Analisis *Elapsed Time* Apache

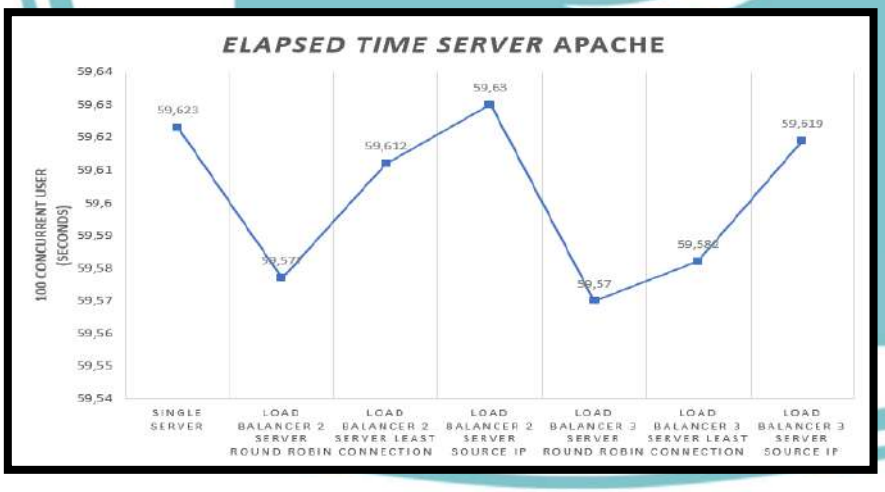
Elapsed time merupakan durasi keseluruhan tes yang dijalankan oleh siege dari *client*. Dari grafik dibawah ini menunjukan perbandingan *elapsed time instance server* apache dengan HTTP *request* dari siege dengan besar 50 *concurrent users*. Data yang didapat dari hasil pengujian pada *single server*, algoritma *round robin, least connection* dan *source IP* menunjukan perbedaan nilai *elapsed time* yang tidak terlalu jauh. Data dari algoritma *round robin, least connection* dan *source* IP menunjukan adanya kenaikan nilai *elapsed time* pada beberapa skenario dengan

menggunakan 2 *instances* dan 3 *instances*. Data perbandingan ditunjukkan pada grafik di gambar 4.6.



Gambar 4.6 Grafik perbandingan *elapsed time* apache 50 *concurrent users*

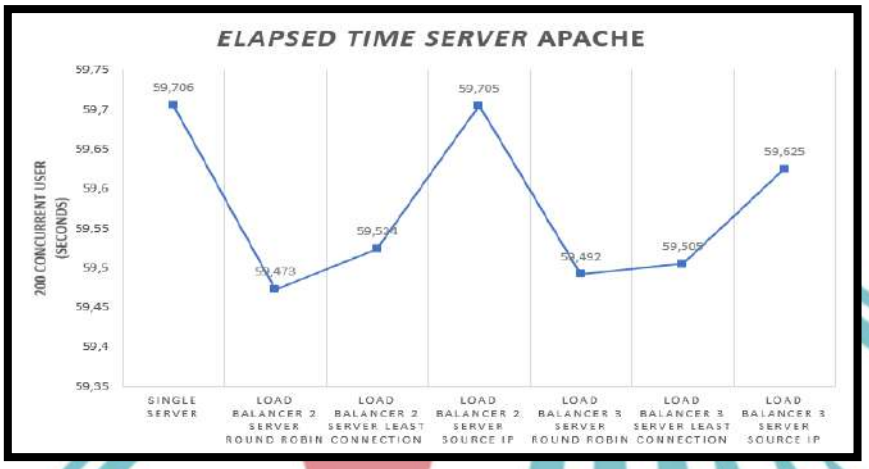
Data pada gambar 4.7 menunjukan perbandingan *elapsed time instance server* apache dengan HTTP *request* dari siege dengan besar 100 *concurrent users*. Data yang didapat dari hasil pengujian *single server* dan algoritma *least connection* dan *source* IP menunjukan peningkatan nilai sedangkan algoritma *round robin* menunjukan nilai yang stabil pada kedua percobaan antara 2 dan 3 *instances*.



Gambar 4.7 Grafik perbandingan *elapsed time* apache 100 *concurrent users*

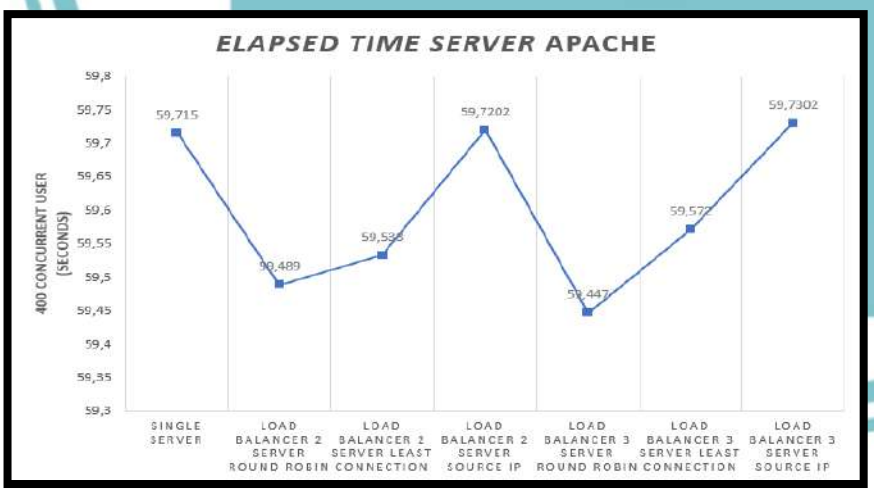
Data pada gambar 4.8 menunjukan perbandingan *elapsed time* untuk *instance server* apache dengan HTTP *request* dari siege dengan besar 200 *concurrent users*. Dilihat dari grafik bahwa *single server* dan algoritma *source* IP mengalami peningkatan dan memiliki nilai yang tidak terlalu jauh, tapi jika dibandingkan dengan algoritma *round robin* dan *least connection* yang menunjukan penurunan

dan menunjukan perbedaan yang signifikan ketika diberi beban *request* 200 *concurrent users*.



Gambar 4.8 Grafik perbandingan *elapsed time* apache 200 *concurrent users*

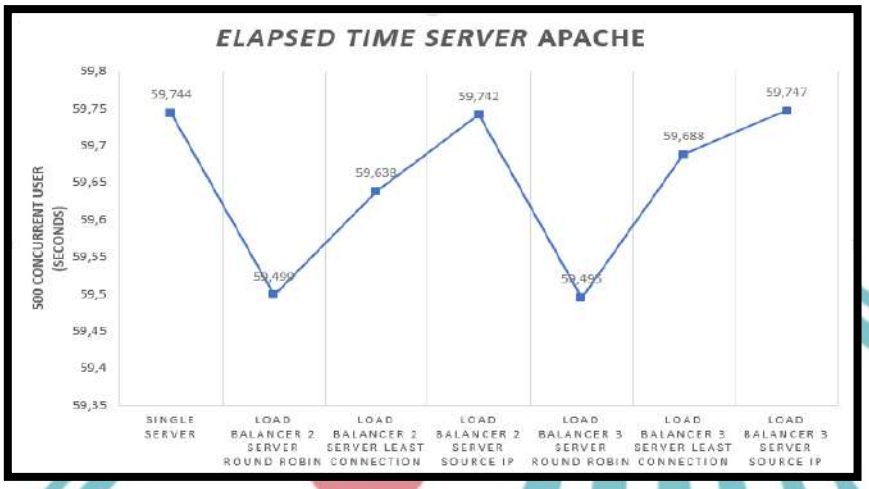
Gambar 4.9 dibawah ini menunjukan perbandingan dari data *elapsed time instance server* apache dengan HTTP *request* dari siege dengan besar 400 *concurrent users*. Grafik dari algoritma *round robin* dan *least connection* menunjukan nilai yang tidak berbeda jauh sedangkan grafik *single server* dan algoritma *source* IP menunjukan peningkatan yang signifikan sehingga berbeda jauh dengan algoritma *round robin* dan *least connection*.



Gambar 4.9 Grafik perbandingan *elapsed time* apache 400 *concurrent users*

Gambar 4.10 dibawah ini menunjukan data *elapsed instance server* apache dengan HTTP *request* dari siege dengan besar 500 *concurrent users*. Dilihat dari grafik bahwa *single server* dan algoritma *source* IP mengalami peningkatan dan memiliki nilai yang tidak terlalu jauh, tapi jika dibandingkan dengan algoritma *round robin*

dan *least connection* yang menunjukan penurunan dan menunjukan perbedaan yang signifikan ketika diberi beban *request* 500 *concurrent users*.



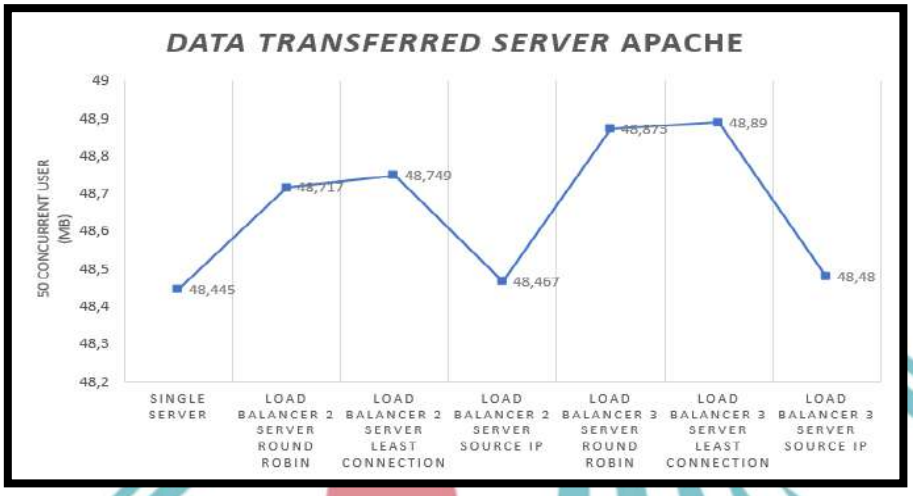
Gambar 4.10 Grafik perbandingan *elapsed time* apache 500 *concurrent users*

Dari data keseluruhan grafik pengujian *elapsed time* pada *instance server* apache diatas menunjukan nilai perbandingannya pada setiap pengujian yang dilakukan. Performa *elapsed time* dikatakan bagus jika nilainya semakin rendah. Pengujian *elapsed time* terhadap *single server* dan *source* IP pada *instance server* apache memiliki nilai yang cenderung sama tinggi. Kemudian untuk perbandingan pengujian *single server* dengan *load balancing* pada *instance server* apache dengan algoritma *round robin* dan *least connection* menunjukan perbedaan yang cukup signifikan ketika menerima beban mulai dari 200 *request*. Terjadinya peningkatan jumlah beban yang diterima oleh *single server* dan *source* IP dikarenakan kedua sistem tersebut hanya bisa melayani dengan satu *server*. Sedangkan algortima *round robin* dan *least connection* dapat membagi jumalah beban *request* tersebut kepada kedua dan ketiga *instances* yang tersedia.

4.5.1.3 Analisis *Data Transferred* Apache

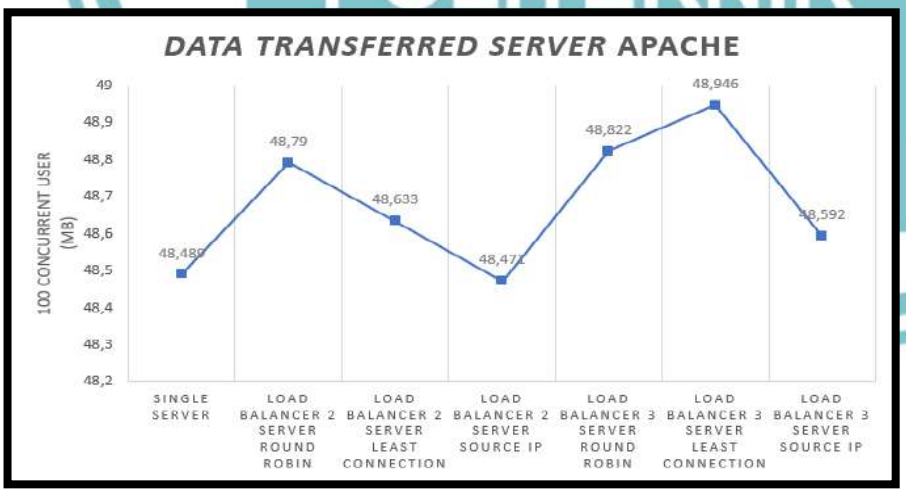
Dari grafik dibawah ini menunjukan perbandingan *data transferred instance* server apache dengan HTTP *request* dari siege sebesar 50 *concurrent users*. Data yang didapat dari hasil pengujian pada *single server*, algoritma *round robin*, *least connection* dan *source IP* menunjukan perbedaan nilai *data transferred*. Data dari algoritma *round robin* dan *least connection* menunjukan peningkatan dan memiliki nilai yang tidak terlalu jauh sedangkan untuk *single server* dan *source IP* menunjukan bahwa nilainya dibawah dari algoritma *round robin* dan *least*

connection. Data perbandingan dapat dilihat pada grafik di gambar 4.11.



Gambar 4.11 Grafik perbandingan *data transferred* apache 50 *concurrent users*

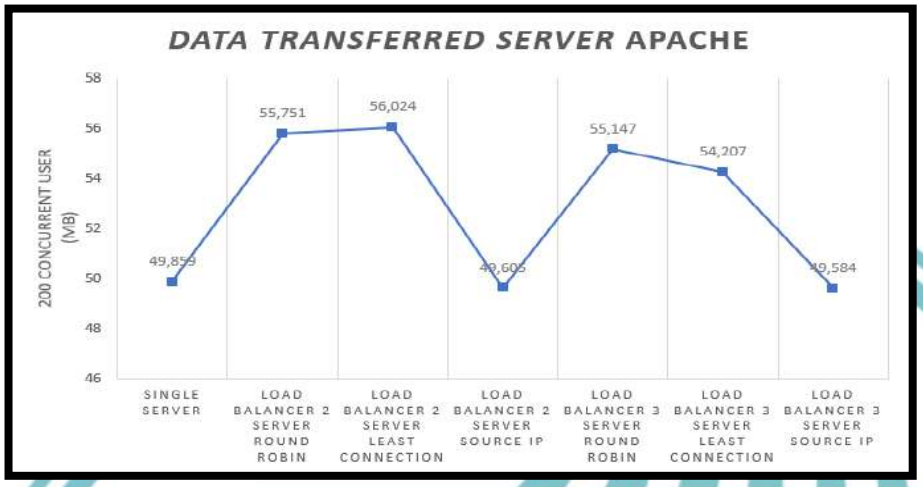
Data berikutnya merupakan perbandingan *data transferred instance server* apache dengan HTTP *request* siege sebesar 100 *concurrent users*. Data grafik dibawah ini semua algoritma *round robin, least connection* dan *source IP* mengalami peningkatan dari pengujian antara 2 *instances* dan 3 *instances*. Grafik dari gambar 4.12 dibawah ini menunjukkan nilai yang tidak berbeda jauh antara *single server* dan *source* IP, sedangkan jika dibandingkan dengan grafik pada algoritma *round robin* ataupun *least connection* keduanya mengalami kenaikan yang jauh terutama algoritman *least connection.*



Gambar 4.12 Grafik perbandingan *data transferred* apache 100 *concurrent users*

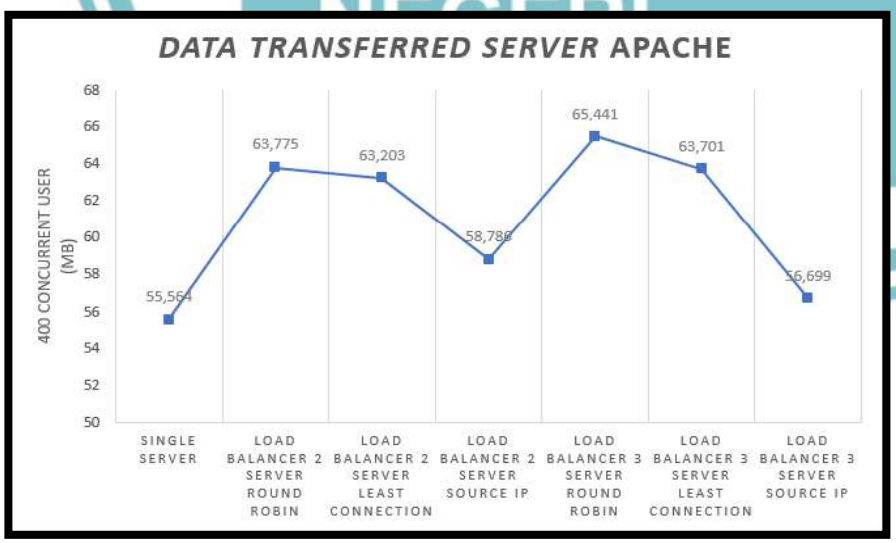
Data grafik dibawah ini menunjukan *data transferred* pada *instance server* apache dengan HTTP *request* dari siege sebesar 200 *concurrent users*. Grafik dari algoritma *round robin* dan *least connection* menunjukan nilai yang tidak berbeda

jauh, sedangkan *single server* dan *source IP* menunjukan perbedaan yang cukup signifikan dengan algoritma *round robin* ataupun *least connection*. Data perbandingan ini dapat dilihat pada gambar 4.13



Gambar 4.13 Grafik perbandingan *data transferred* apache 200 *concurrent users*

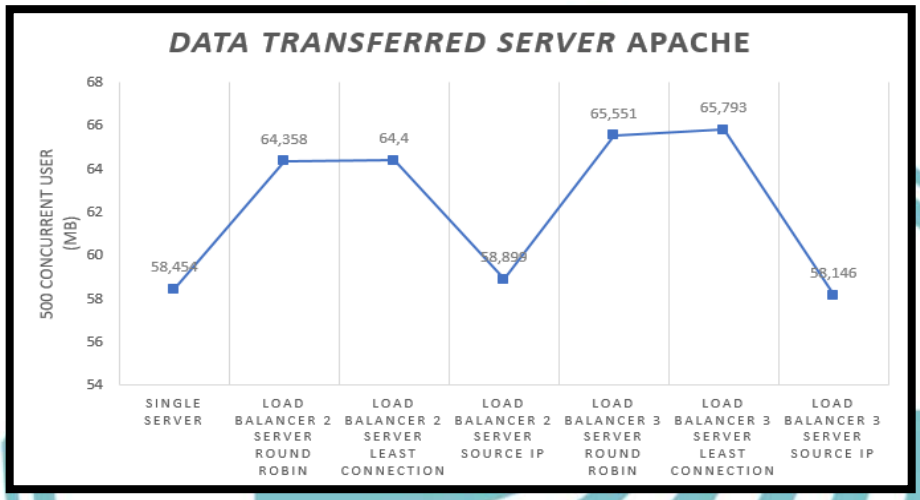
Data grafik dibawah ini menunjukan *data transferred* pada *instance server* apache dengan HTTP *request* dari siege sebesar 400 *concurrent users*. Grafik pada algoritma *round robin* dan *least connection* menunjukan pengingkatan dan tidak memiliki nilai yang berbeda jauh untuk scenario 2 *instances*. Lalu, jika dibandingkan dengan *single server* dan *source IP* menunjukan perbedaan yang cukup singnifikan dari nilai yang di dapatkan oleh algoritma *round robin* dan *least connection*. Data dari perbandingan ini dapat dilihat pada gambar 4.14.



Gambar 4.14 Grafik perbandingan *data transferred* apache 400 *concurrent users*

Grafik dibawah ini menunjukan perbandingan *data transferred* pada *instance server* apache dengan jumlah HTTP request dari siege sebesar 500 *concurrent*

users. Grafik dari *single server* dan *source* IP nilainya berbeda jauh jika dibandingkan dengan grafik dari algoritma *round robin* dan *least connection*. Namun, seluruh skenario mengalami peningkatan nilai yang tidak terlalu signifikan. Data dari perbandingan ini dapat dilihat pada gambar 4.15.



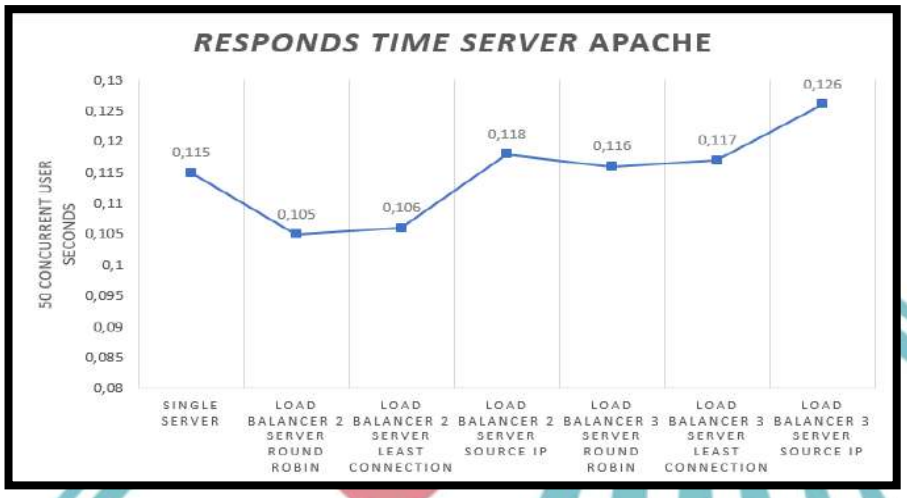
Gambar 4.15 Grafik perbandingan *data transferred* apache 500 *concurrent users*

Dari data keseluruhan grafik pengujian *data transferred* pada *instance server* apache diatas menunjukan nilai perbandingannya pada setiap pengujian yang dilakukan. Jika *server* bisa menangani *request* dengan cepat, jadi semakin tinggi nilai *data transferred* bisa dikatakan bahwa performanya bagus. Dari seluruh grafik diatas data grafik dari algorima *round robin* dan *least connection* menunjukan nilai yang lebih tinggi jika dibandingkan dengan grafik dari *single server* dan *source IP*. Mengapa *single server* dan *source IP* tidak bisa menangani beban *request* dengan baik dikarenakan *single server* maupun *source IP* hanya memiliki satu *sever* sehingga tidak bisa menerima beban *request* yang tinggi.

4.5.1.4 Analisis *Responds Time* Apache

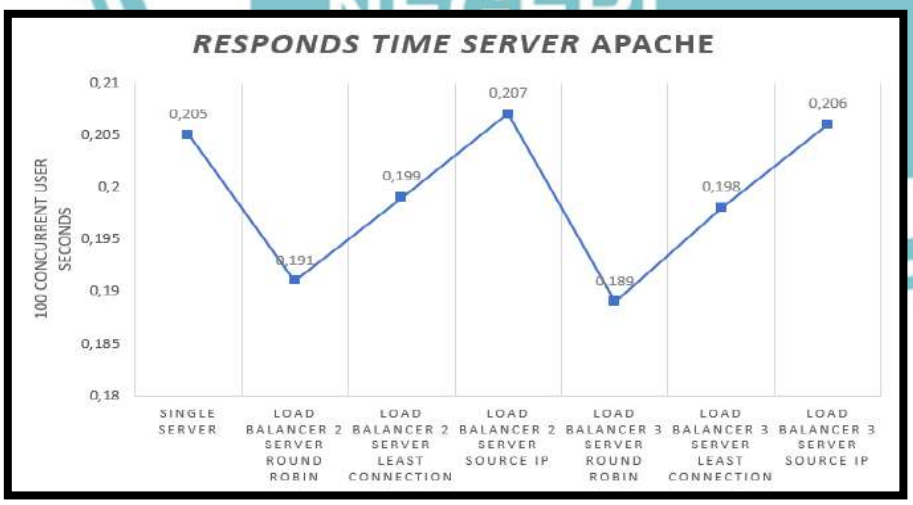
Dari grafik dibawah ini menunjukan bahwa perbandingan *responds time* pada *instance server* apache dengan HTTP *request* dari siege sebesar 50 *concurrent users*. Data yang didapat dari hasil pengujian pada menunjukan peningkatan pada tiap skenario dari hasil *single server*. Tetapi nilai pada skenario 3 *server* pada algoritma *round robin* dan *least connection* tidak berbeda jauh dengan nilai *single server*. Sedangkan pada scenario 2 *server* peforma algoritma *round robin* dan *least connection* lebih rendah dari *single server*. Performa *responds time* dikatakan bagus

jika nilainya menjadi semakin kecil yang diberikan dari *server* tersebut. Data perbandingan dapat dilihat pada gambar 4.16.



Gambar 4.16 Grafik perbandingan *responds time* apache 50 *concurrent users*

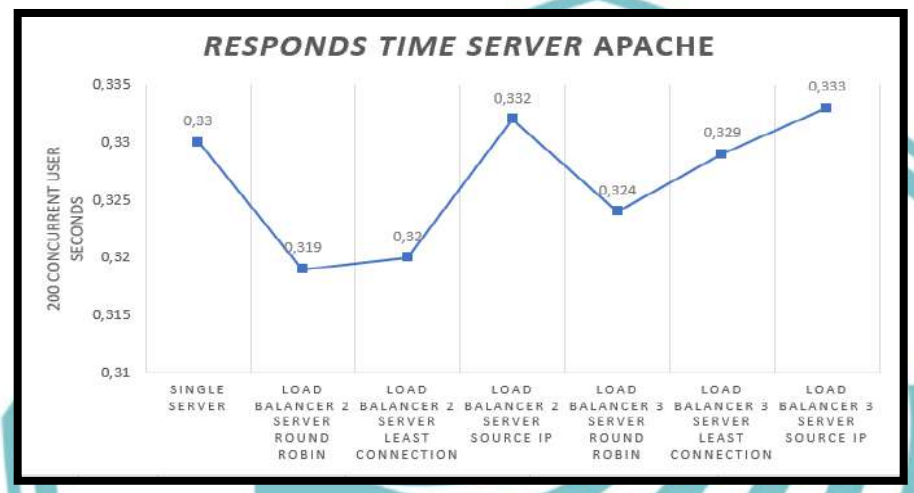
Data berikutnya menunjukan *responds time* pada *instance server* apache dengan HTTP *request* dari siege sebesar 100 *concurrent users*. Grafik *source* IP menunjukan peningkatan dari grafik sebelumnya dan memiliki nilai yang lebih tinggi daripada *single server* walau tidak terlalu jauh. Sedangkan untuk algoritma *round robin* dan *least connection* menunjukan nilai yang cukup berbeda dengan *single server* dan *source* IP. Selain itu, nilai algoritma *round robin* dan *least connection* mengalami penurunan dari skenario 2 *server* ke skenario 3 *server*. Data perbandingan dapat dilihat pada gambar 4.17.



Gambar 4.17 Grafik perbandingan *responds time* apache 100 *concurrent users*

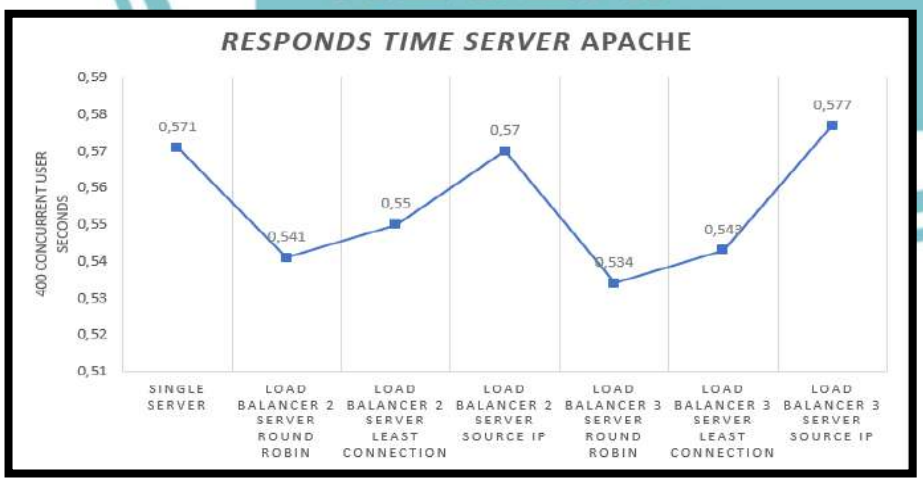
Grafik dibawah ini menunjukan perbandingan *responds time* pada *instance server* apache dengan HTTP *request* dari siege sebesar 200 *concurrent users*. Algoritma

round robin dan least connection memiliki nilai yang lebih kecil dibandingkan dengan *single server* dan *source* IP. Hal tersebut dikarenakan algoritma *round robin* dan *least connection* memiliki 2 *server* dan 3 *server* oleh sebab itu waktu yang dibutuhkan relatif lebih sedikit untuk menyelesaikan *reques*t dari *client*. Data perbandingan dapat dilihat pada gambar 4.18.



Gambar 4.18 Grafik perbandingan *responds time* apache 200 *concurrent users*

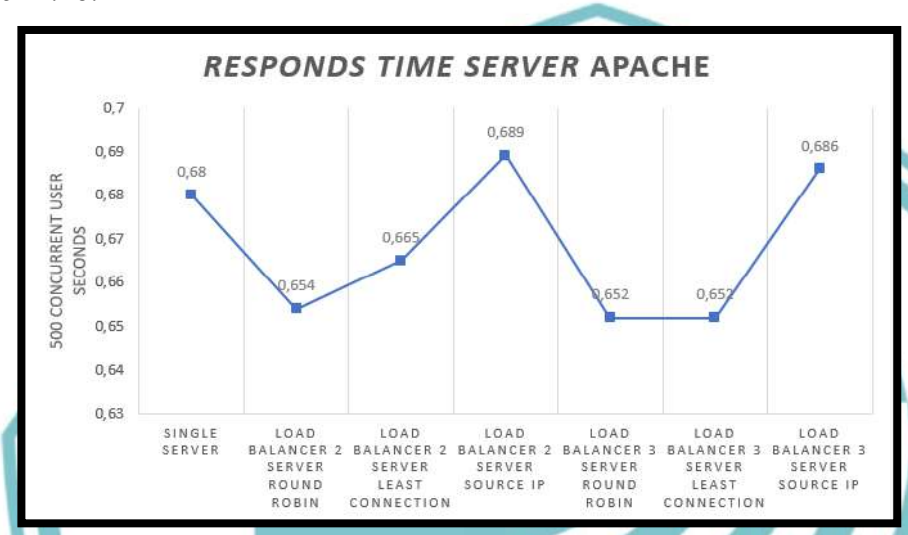
Grafik selanjutnya menunjukan perbandingan *responds time* pada *instance server* apache dengan HTTP *request* dari siege sebesar 400 *concurrent users*. Grafik ini menunjukan peningkatan pada *single server* dan *source* IP dan memiliki nilai yang tidak terlalu jauh. Dibandingkan dengan algoritma *round robin* dan *least connection* menunjukkan penurunan nilai dan perbedaan nilai yang cukup signifikan dengan *single server* dan *source* IP. Data perbandingan dapat dilihat pada gambar 4.19.



Gambar 4.19 *Grafik perbandingan* responds time *apache 400* concurrent users

Grafik dibawah ini menunjukan perbandingan *responds time* pada *instance server* apache dengan HTTP *request* dari siege sebesar 500 *concurrent users*. Algoritma

round robin dan *least connection* menunjukan penurunan dan memiliki nilai yang tidak berbeda jauh satu sama lain. Sedangkan untuk *single server* dan *source* IP menunjukan nilai yang lebih tinggi dan cukup signifikan perbedaannya dengan algoritma *round robin* dan *least connection*. Data perbandingan dapat dilihat pada gambar 4.20.



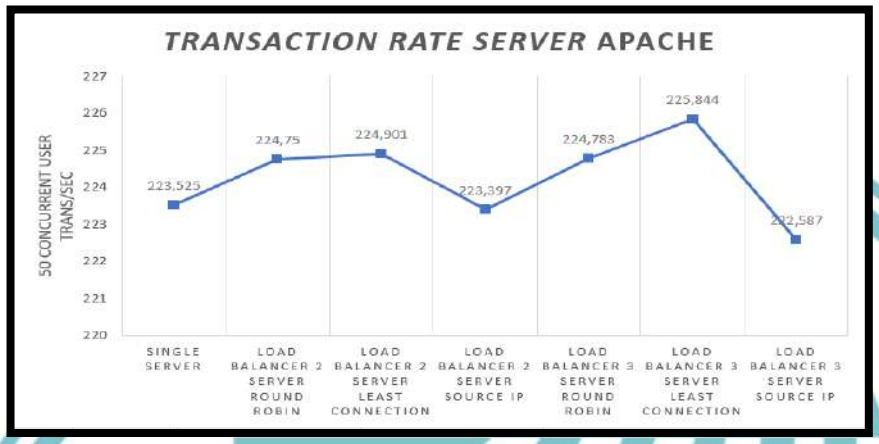
Gambar 4.20 Grafik perbandingan *responds time* apache 500 *concurrent users*

Dari data keseluruhan grafik pengujian *responds time* pada *instance server* apache diatas menunjukan nilai perbandingannya pada setiap pengujian yang dilakukan. Hasil dari pengujian keseluruhan menunjukan bahwa semakin kecil nilai *responds time* yang didapat maka akan semakin bagus performa yang diberikan *server* tersebut. Grafik *single server* dan *source* IP menunjukan peningkatan yang cukup drastis saat jumlah HTTP *request* sebanyak 200 *concurrent users*. Peningkatan nilai dialami karena dalam penerimaan beban, *single server* dan *source* IP hanya memiliki satu *server* sehingga tidak dapat membagi beban kepada *server* lainnya dan membutuhkan waktu yang lebih lama untuk menyelesaikan *request* dari *client.* Berbeda dengan algoritma *round robin* dan *least connection* yang memiliki waktu yang relatif lebih kecil jika dibandingkan dengan *single server* maupun *source* IP dikarenakan memiliki 2 dan 3 *server* untuk membagi jumlah beban sehingga dapat menyelesaikan tugas dengan cepat.

4.5.1.5 Analisis *Transaction Rate* Apache

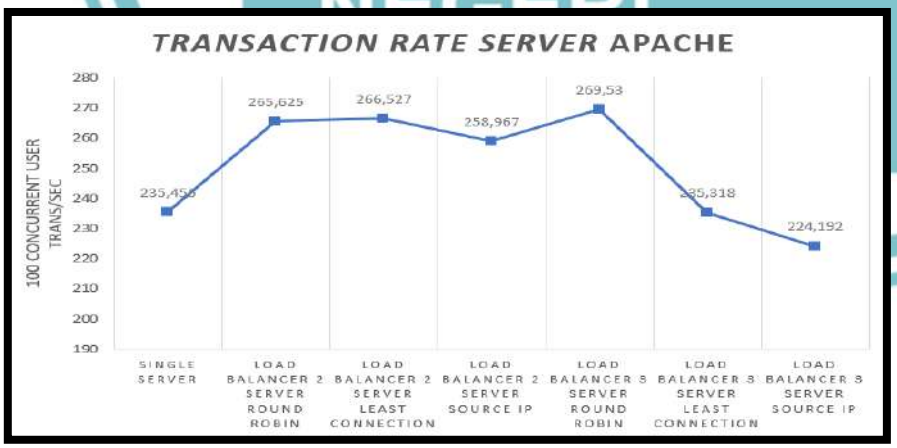
Dari grafik dibawah ini menunjukan bahwa perbandingan *transaction rate* pada *instance server* apache dengan HTTP *request* dari siege sebesar 50 *concurrent users*. Grafik pada algoritma *round robin* dan *least connection* menunjukan

peningkatan nilai sedangkan algoritma *source* IP menunjukan penurunan nilai. Pada pengujian dengan jumlah HTTP request sebesar 50 *concurrent users* tidak menunjukan nilai yang jauh berbeda antara *single server*, *round robin*, *least connection* dan *source* IP.



Gambar 4.21 Grafik perbandingan *transaction rate* apache 50 *concurrent users*

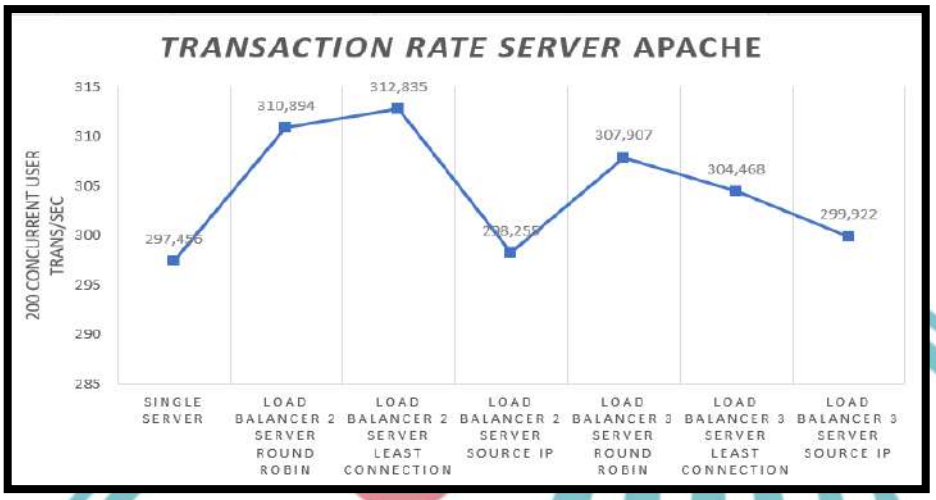
Data perbandingan selanjutnya menunjukan data *transaction rate* pada *instance server* apache dengan HTTP *request* dari siege sebesar 100 *concurrent users*. Algoritma *round robin* menunjukan peningkatan nilai antara pengujian 2 *server* ke 3 *server*, sedangkan algoritma *least connection* dan *source* IP menunjukan penurunan. Pada pengujian dengan jumlah HTTP *request* sebanyak 100 *concurrent users* tidak menunjukan nilai yang jauh berbeda antara *single server*, *round robin*, *least connection* dan *source* IP.



Gambar 4.22 Grafik perbandingan *transaction rate* apache 100 *concurrent users*

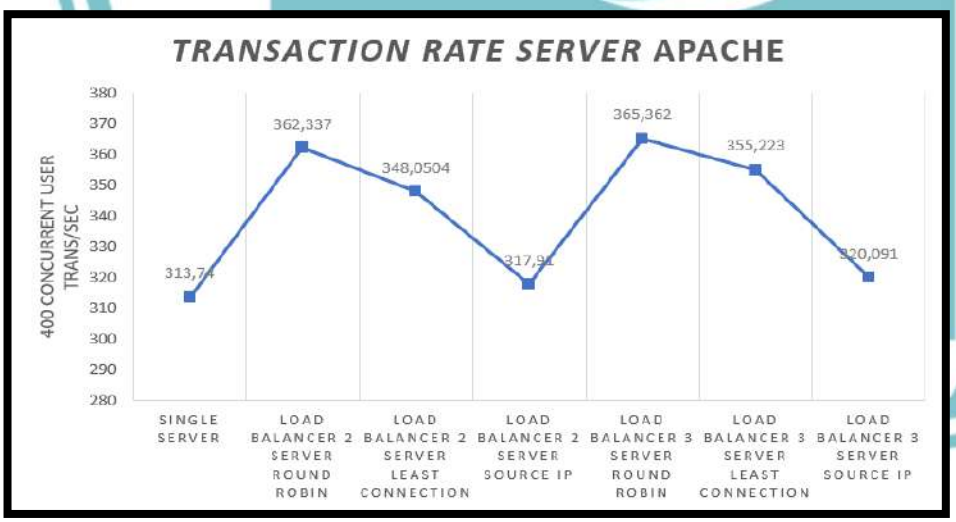
Grafik dibawah ini menunjukan perbandingan *transaction rate* pada *instance server* apache dengan HTTP *request* dari siege sebesar 200 *concurrent users*. Pada grafik kali ini *single server* dan *source* IP menunjukan nilai yang cukup rendah

dibandingan dengan algoritma *round robin* dan *least connection*. Semakin rendah nilai yang didapat oleh *server* tersebut maka semakin buruk performanya.



Gambar 4.23 Grafik perbandingan *transaction rate* apache 200 *concurrent users*

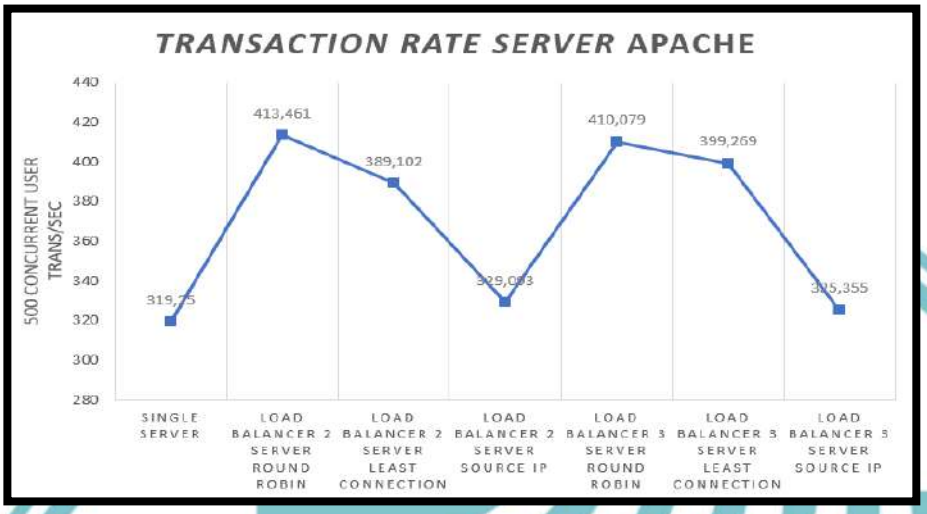
Data selanjutnya merupakan perbandingan *transaction rate* pada *instance server* apache dengan HTTP *request* dari siege sebesar 400 *concurrent users*. Grafik pada algoritma *round robin* dan *least connection* menunjukan peningkatan nilai pada pengujian 2 *server* dan 3 *server*. Selain itu, *single server* dan *source* IP menunjukan nilai lebih rendah dari algoritma *round robin* dan *least connection*.



Gambar 4.24 Grafik perbandingan *transaction rate* apache 400 *concurrent users*

Data selanjutnya merupakan perbandingan *transaction rate* pada *instance server* apache dengan HTTP *request* dari siege sebesar 500 *concurrent users*. Grafik pada algoritma *round robin* dan *least connection* menunjukan peningkatan nilai pada pengujian 2 *server* dan 3 *server*. Pada grafik ini *single server* dan *source* IP menunjukan nilai yang cukup rendah dan tidak terlalu jauh dibandingan dengan

grafik sebelumnya terhadap algoritma *round robin* dan *least connection*. Semakin tinggi nilai *transaction rate* berbanding lurus dengan semakin bagus performa yang didapat.



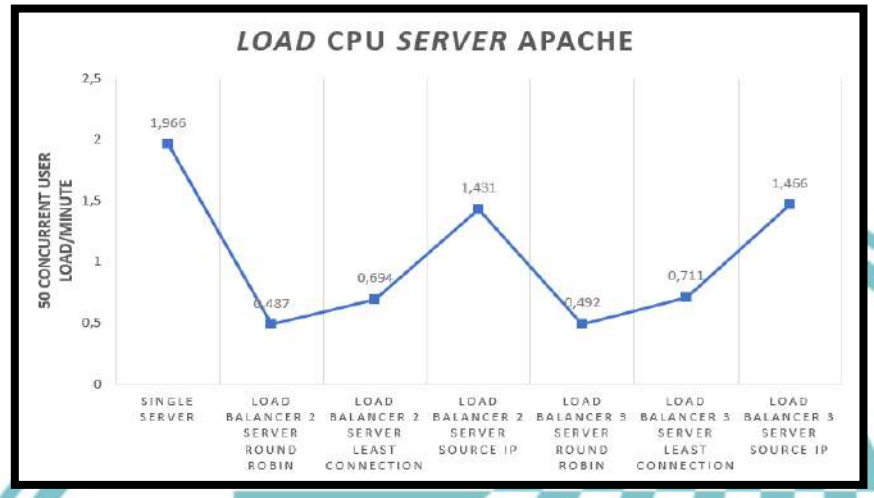
Gambar 4.25 Grafik perbandingan *transaction rate* apache 500 *concurrent users*

Dari data keseluruhan grafik pengujian *transaction rate* pada *instance server apache* diatas menunjukan nilai perbandingannya pada setiap pengujian yang dilakukan. Performansi *transaction rate* dikatakan bagus berbanding lurus dengan semakin tinggi nilai yang didapat. Grafik *transaction rate* pada algoritma *round robin* dan *least connection* dengan menggunakan *load balancing* menunjukan peningkatan. Hal ini dikarenakan dengan *load balancing, server* dapat membagi sejumlah *request* kepada *server* lainnya sehingga mampu mengoptimalkan layanan *server*. Dimulai saat *server* diberikan *request* sebanyak 200 *concurrent users*, *single server* maupun *source* IP menunjukan penurunan menyebabkan *server* tersebut tidak bekerja secara optimal. Pada sistem yang menggunakan *load balancing* dengan algoritma *round robin* dan *least connection* dapat melayani jumlah *request* yang lebih besar dibandingkan dengan sistem *single server* dan algoritma *source* IP.

4.5.1.6 Analisis *Load* CPU Apache

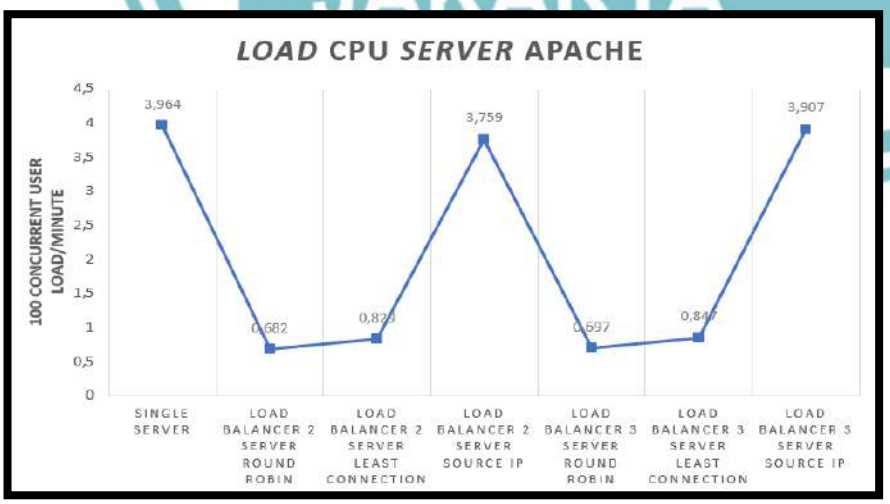
Untuk mengukur seberapa besar penggunaan *resource* dari segi CPU *client* pada saat menjalankan sistem dengan *load balancing* dan *single server* merupakan tujuan dari pengujian ini. Dari grafik dibawah ini menunjukan bahwa perbandingan *load cpu* pada *server* apache dengan HTTP *request* dari siege sebesar 50 *concurrent users*. Data yang didapat dari hasil pengujian menunjukan bahwa algoritma *round*

robin dan algoritma *least connection* memiliki nilai yang tidak terlalu jauh jika dibandingkan dengan *single server* dan algoritma *source* IP. Performa CPU dikatakan bagus berbanding lurus dengan nilai yang dihasilkan semakin kecil.



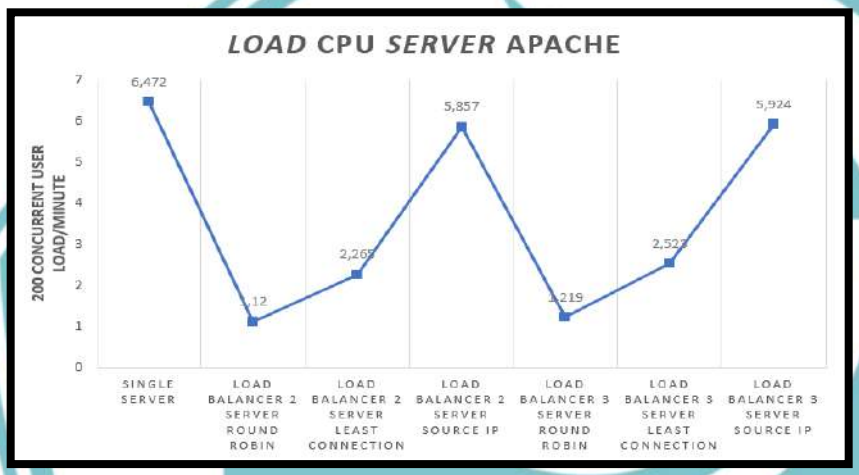
Gambar 4.26 Grafik perbandingan *load cpu* apache 50 *concurrent users*

Data perbandingan selanjutnya menunjukan data *load cpu* pada *server* apache dengan HTTP *request* dari siege sebesar 100 *concurrent users*. Data yang didapat dari hasil pengujian menunjukan bahwa algoritma *source* IP dan *single server* memiliki nilai yang sama-sama tinggi dibangdingkan dengan algoritma *round robin* dan algoritma *least connection* dikarenakan setiap *server* hanya memliki satu vcpu saja, berbeda dengan algoritma *round robin* dan *least connection* yang memiliki dua vcpu dan tiga vcpu berdasarkan skenario yang dilakukan pada setiap percobaan sehingga mampu mengoptimalkan beban secara merata dan menghasilkan pengujian yang lebih optimal.



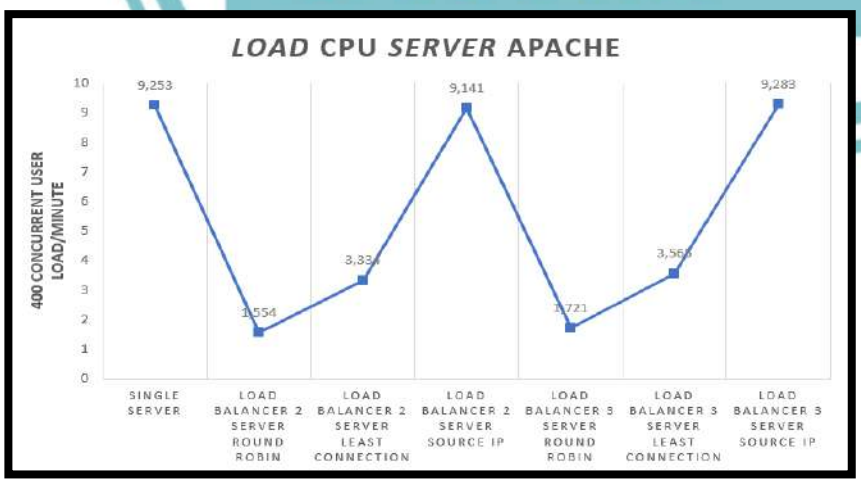
Gambar 4.27 Grafik perbandingan *load cpu* apache 100 *concurrent users*

Data selanjutnya menunjukan data *load cpu* pada *server* apache dengan HTTP *request* dari siege sebesar 200 *concurrent users*. Data yang didapat dari hasil pengujian menunjukan peningkatan yang signifikan dari *single server* dan algoritma *source* IP dibandingkan dengan algoritma *round robin* dan algoritma *least connection*. Dikarenakan *single server* dan algoritma *source* IP tidak bisa membagi beban sehingga kinerja vcpu menjadi lebih berat dan tidak berjalan secara optimal. Performa CPU dikatakan bagus berbanding lurus dengan nilai yang dihasilkan semakin kecil.



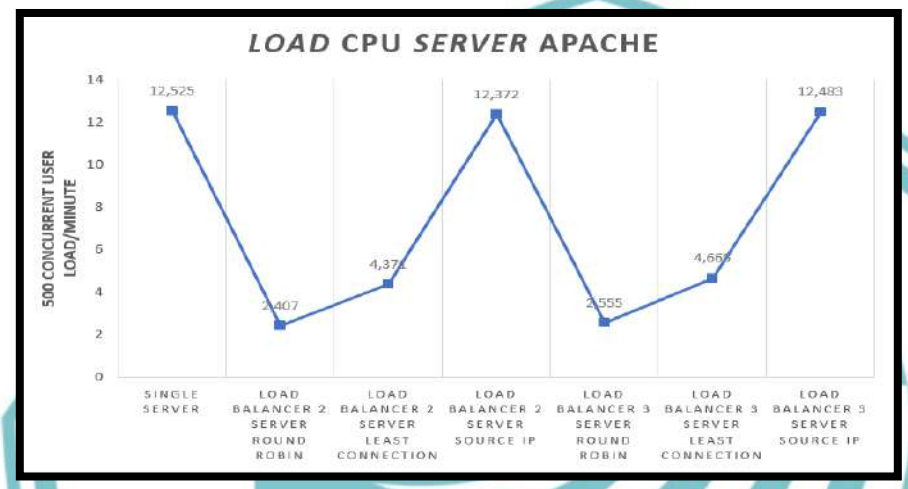
Gambar 4.28 Grafik perbandingan *load cpu* apache 200 *concurrent users*

Data selanjutnya merupakan perbandingan *load cpu* pada *server* apache dengan HTTP *request* dari siege sebesar 400 *concurrent users*. Grafik pada algoritma *round robin* dan *least connection* menunjukan peningkatan nilai pada pengujian 2 *server* dan 3 *server*. Selain itu, *single server* dan *source* IP menunjukan nilai lebih tinggi dari algoritma *round robin* dan *least connection*.



Gambar 4.29 Grafik perbandingan *load cpu* apache 400 *concurrent users*

Grafik dibawah ini menunjukan perbandingan *load cpu* pada *server* apache dengan HTTP *request* dari siege sebesar 500 *concurrent users*. Pada grafik kali ini *single server* dan *source* IP menunjukan nilai yang sanagt tinggi dibandingan dengan algoritma *round robin* dan *least connection*. Semakin tinggi nilai yang didapat oleh *server* tersebut maka semakin buruk performanya.



Gambar 4.30 Grafik perbandingan *load cpu* apache 500 *concurrent users*

Dari data keseluruhan grafik pengujian *load cpu* pada *server apache* diatas menunjukan nilai perbandingannya pada setiap pengujian yang dilakukan. Performansi *load cpu* dikatakan bagus berbanding lurus dengan semakin rendah nilai yang didapat. Grafik *load cpu* pada algoritma *round robin* dan *least connection* dengan menggunakan *load balancing* menunjukan nilai yang rendah dikarenakan dapat membagi beban dari *client* dibagi kepada dua dan tiga *server* yang berarti dua vcpu dan tiga vcpu sehingga kinerja masing-masing vcpu menjadi lebih ringan dan hasil pengujian pun menjadi lebih optimal.

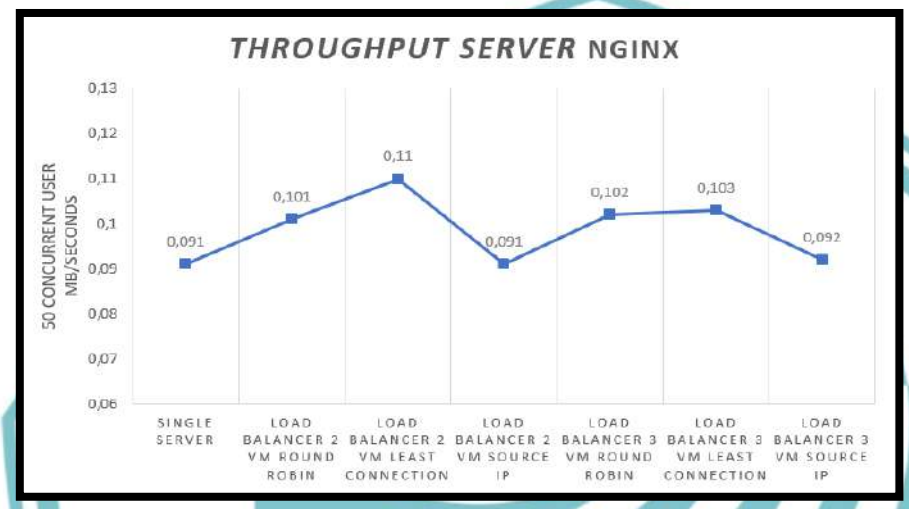
4.5.2 Analisis Data *Web Server* Nginx

Data dari pengujian *single server* dan *load balancing* dianalisis dengan membandingkan hasil *throughput, elapsed time, data transferred, responds time*, *transaction rate* dan *load cpu* dengan menggunakan 1 *instance*, 2 *instances* dan 3 *instances* dengan *web server* nginx.

4.5.2.1 Analisis *Througput* Nginx

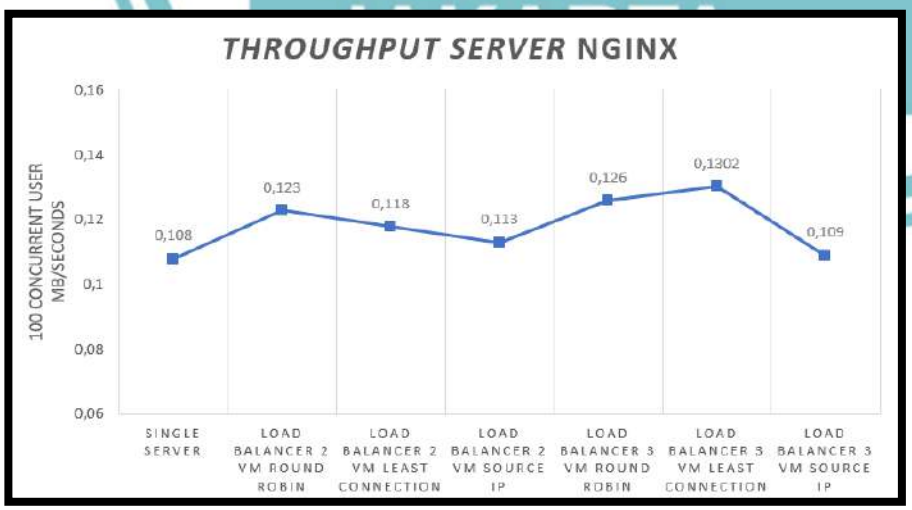
Dari grafik dibawah ini menunjukan perbandingan *throughput* pada *server* nginx dengan HTTP *request* dari siege sebesar 50 *concurrent users*. Data yang didapat dari tiga pengujian skenario menunjukan bahwa algoritma *round robin* dan

algoritma *least connection* memiliki nilai yang tidak berbeda jauh. Tetapi jika dibandingkan dengan *single server* dan *source* IP memiliki nilai yang cukup signifikan. Dari ketiga skenario yang diuji algoritma *round robin* menunjukan nilai yang paling tinggi dibandingkan dengan algoritma lain dan *single server*. Semakin tinggi nilai *throughput* maka performansi dari *web server* dinyatakan bagus.



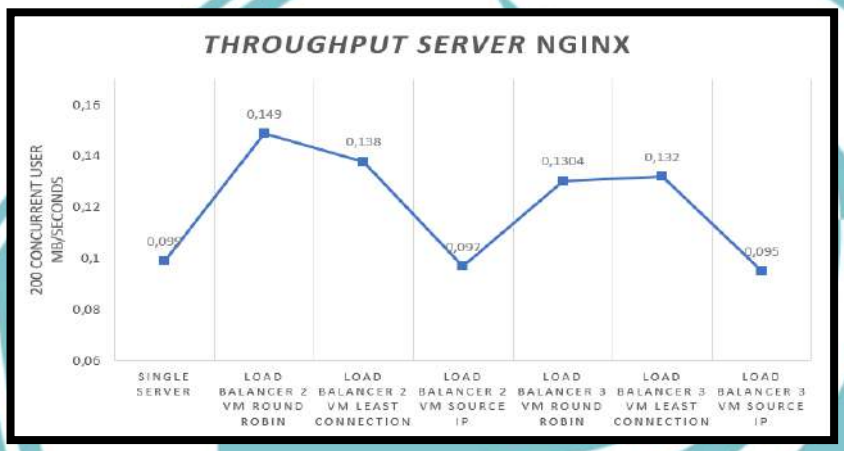
Gambar 4.31 Grafik perbandingan *throughput* nginx 50 *concurrent users*

Grafik berikutnya menunjukan perbandingan *throughput* pada *server* nginx dengan HTTP request dari siege sebesar 100 *concurrent users*. Algoritma *round robin* dan *least connection* menunjukan peningkatan antara pengujian 2 *instances* ke 3 *instances* sedangkan pada *single server* dan *source* IP menunjukkan nilai lebih rendah dibandinganken algoritma *round robin* dan *least connection*. Semakin tinggi nilai *throughput* maka semakin bagus performa *server* tersebut.



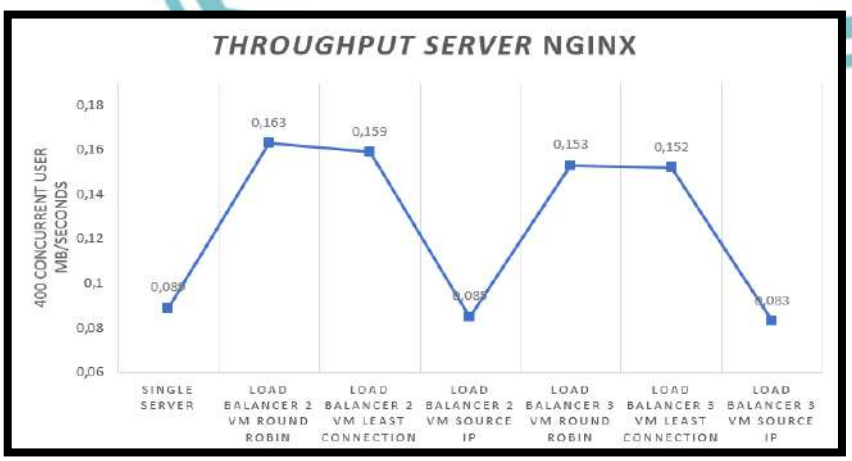
Gambar 4.32 Grafik perbandingan *throughput* nginx 100 *concurrent users*

Grafik dibawah ini menunjukan perbandingan *throughput* pada *server* nginx dengan HTTP request dari siege sebesar 200 *concurrent users*. Grafik *single server* dan algoritma *source* IP menunjukan penurunan dibandingkan dengan nilai grafik sebelumnya. Sedangkan untuk algoritma *round robin* dan *least connection* mengalami peningkatan dari grafik sebelumnya dikarenakan algoritma yang menggunakan *load balancing* dapat membagi sejumlah beban kepada *server* lainnya sedangkan untuk *single server* dan *source* IP hanya memiliki satu *server* sehingga tidak bisa menangani beban secara optimal.



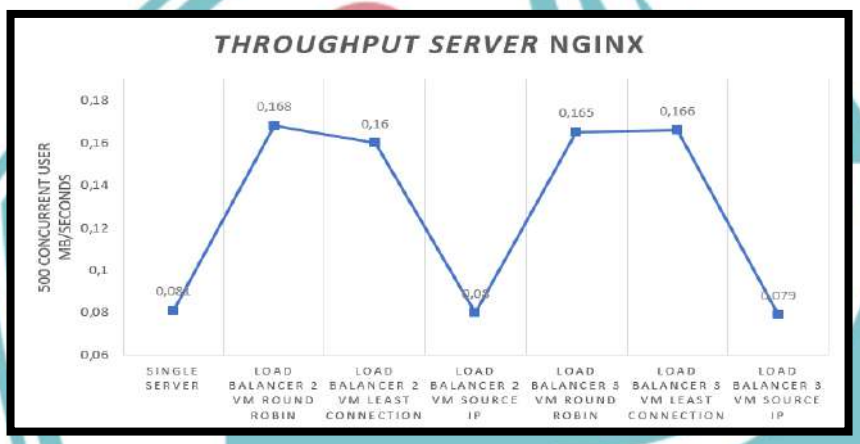
Gambar 4.33 Grafik perbandingan *throughput* nginx 200 *concurrent users*

Grafik berikutnya menunjukan perbandingan *throughput* pada *server* nginx dengan HTTP request dari siege sebesar 400 *concurrent users*. Seluruh skenario mengalami penurunan nilai dari scenario 2 *server* ke 3 *server*, namun untuk algoritma *round robin* dan *least connection*, nilai *throuhgputnya* lebih besar dibandingkan dengan *single server* sedangkan algoritma *source* IP memiliki nilai yang lebih rendah daribandingkan *single server*. Penuruan terjadi akibat *server* tidak mampu menerima beban lagi dan tidak mampu bekerja secara optimal.



Gambar 4.34 Grafik perbandingan *throughput* nginx 400 *concurrent users*

Grafik dibawah ini menunjukan perbandingan *throughput* pada *server* nginx dengan HTTP request dari siege sebesar 500 *concurrent users*. Sama seperti pengujian pada *instance server* apache bahwa semakin banyaknya *request single server* maupun *source* IP tidak mampu untuk menerima beban *request* dari *client* karena tidak ada pembagian *request* kepada *server* yang lainnya. Sebaliknya algoritma *round robin* dan *least connection* menunjukan peningkatan dikarenakan dapat membagi jumlah beban *request* dari *client* dan menunjukan dengan *load balancing* lebih unggul dibandingkan *single server* dan *source* IP.



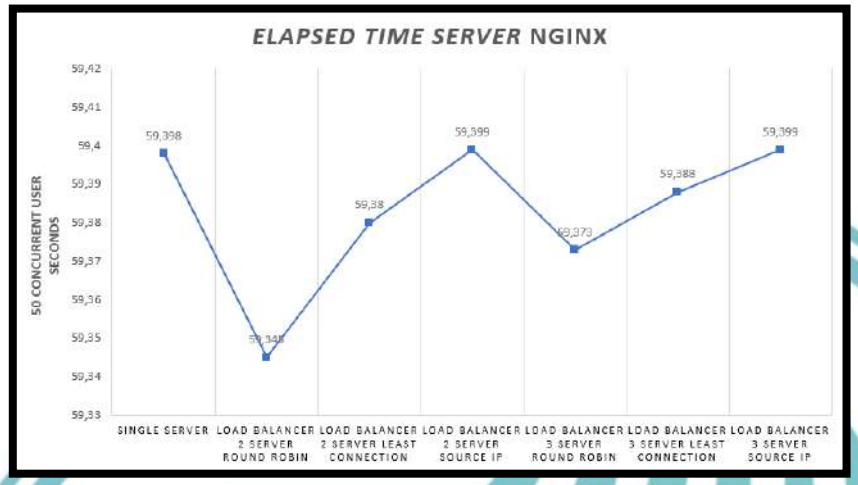
Gambar 4.35 Grafik perbandingan *throughput* nginx 500 *concurrent users*

Dari data keseluruhan grafik pengujian *throughput* pada *server* nginx diatas menunjukan perbandingan antara *single server* dengan *load balancing* dimana grafik dari *single server* dan *source* IP mulai menunjukan penurunan yang cukup signifikan ketika menerima beban *request* sebanyak 200 *concurrent users*. Terjadinya penurunan diakibatkan *single server* maupun *source* IP tidak mampu menangani beban dikarenakan hanya memiliki satu *server* dan tidak bisa membagi beban *request* ke *server* lainnya. Performansi dikatakan bagus jika nilai *throughput* menunjukan peningkatan. Algoritma *round robin* dan *least connection* menunjukan peningkatan dikarenakan dapat membagi jumlah beban *request* kepada *instance web server* lainnya.

4.5.2.2 Analisis *Elapsed Time* Nginx

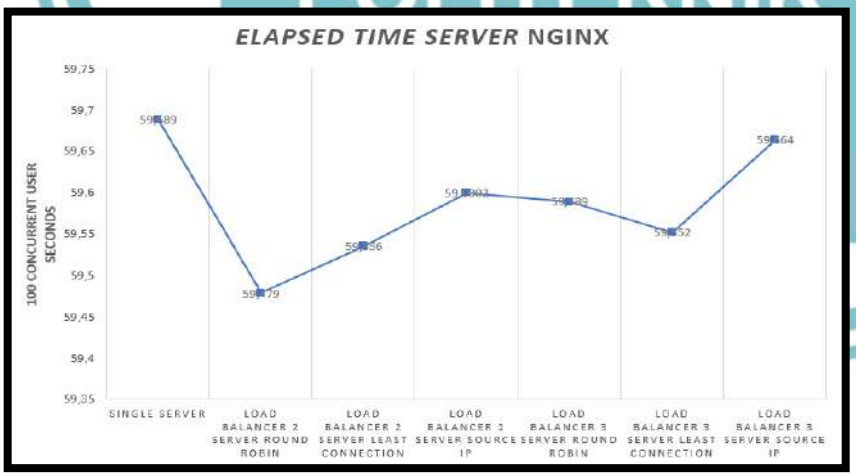
Grafik dibawah ini menunjukan data *elapsed time server* nginx dengan HTTP *request* dari siege sebesar 50 *concurrent users*. Data yang didapat dari hasil pengujian pada *single server*, algoritma *round robin, least connection* dan *source*

IP menunjukan perbedaan nilai *elapsed time* yang tidak terlalu jauh. Data dari algoritma *round robin, least connection* dan *source* IP menunjukan adanya kenaikan nilai *elapsed time* pada skenario dengan menggunakan 2 dan 3 *instances*.



Gambar 4.36 Grafik perbandingan *elapsed time* nginx 50 *concurrent users*

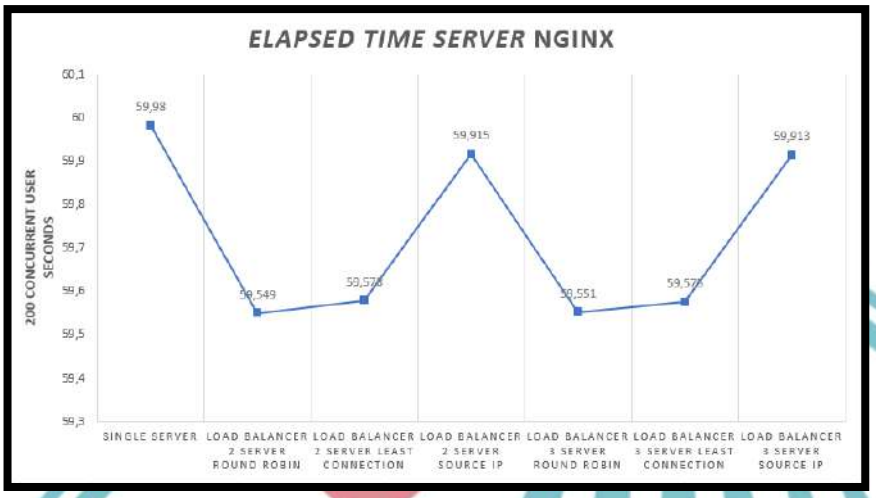
Data berikutnya ditunjukkan untuk *elapsed time server* nginx dengan HTTP *request* dari siege sebesar 200 *concurrent users*. Data yang didapat dari hasil pengujian pada *single server*, algoritma *round robin, least connection* dan *source IP* menunjukan perbedaan nilai *elapsed time* yang tidak terlalu jauh. Data dari algoritma *round robin, least connection* dan *source* IP menunjukan adanya kenaikan nilai *elapsed time* pada skenario dengan menggunakan 2 dan 3 *instances*.



Gambar 4.37 Grafik perbandingan *elapsed time* nginx 100 *concurrent users*

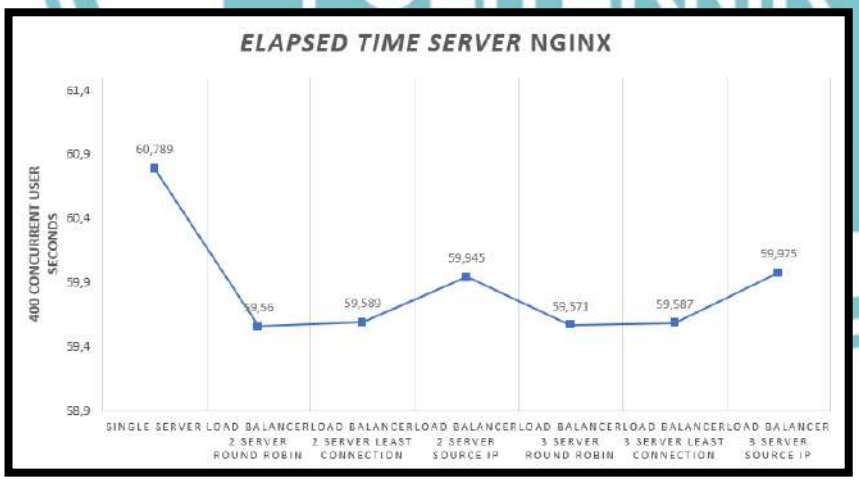
Data berikutnya, menunjukan *elapsed time server* nginx dengan HTTP *request* dari siege sebesar 200 *concurrent users*. Tidak berbeda jauh dari data *instance server* apache, *server* nginx menunjukan peningkatan yang signifikan dari *single server*

dan algoritma *source* IP sehingga terdapat perbedaan jauh dengan algoritma *round robin* dan *least connection*.



Gambar 4.38 Grafik perbandingan *elapsed time* nginx 200 *concurrent users*

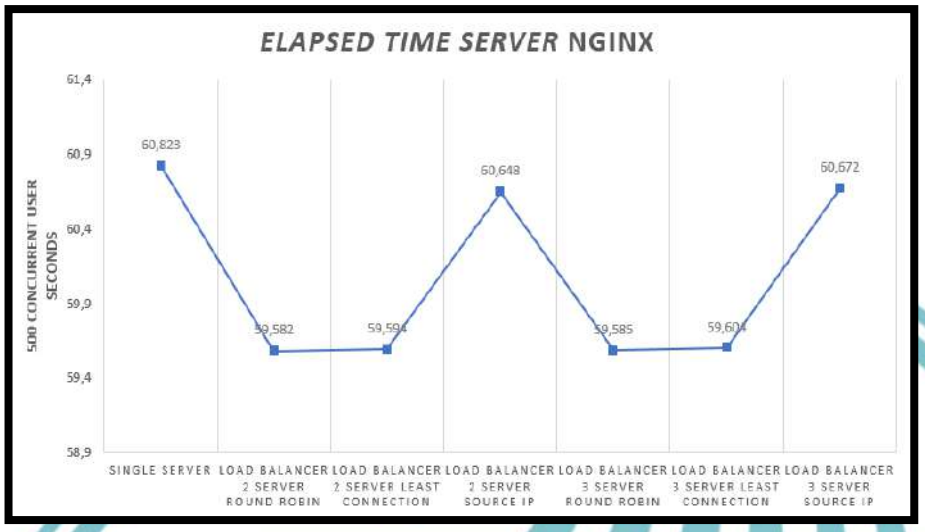
Gambar berikutnya menunjukan data perbandingan *elapsed time server* nginx dengan HTTP *reques*t dari siege sebesar 400 *concurrent users*. Grafik dari *single server* memiliki nilai yang paling tinggi dibandingkan dengan algoritma lain. Terlebih dengan algoritma *source* IP menunjukan nilai yang tinggi setelah grafik *single server*. Sementara itu nilai dari algoritma *round robin* dan *least connection* tidak berbeda jauh, jika dibandingkan dengan *single server* dan *source* IP maka menunjukan peningkatan sehingga berbeda jauh nilainya.



Gambar 4.39 Grafik perbandingan *elapsed time* nginx 400 *concurrent users*

Data berikutnya, menunjukan *elapsed time server* nginx dengan HTTP *request* dari siege sebesar 500 *concurrent users*. Tidak berbeda jauh dari data instance server apache, *server* nginx menunjukan peningkatan yang signifikan dari *single serve*

dan algoritma *source* IP sehingga terdapat perbedaan jauh dengan algoritma *round robin* dan *least connection*.



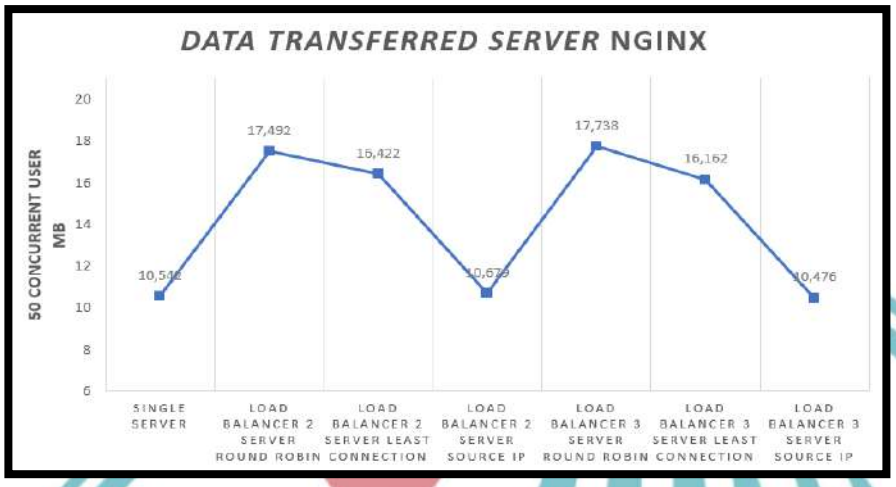
Gambar 4.40 Grafik perbandingan *elapsed time* nginx 500 *concurrent users*

Dari data keseluruhan grafik pengujian *elapsed time* pada *server* nginx diatas menunjukan nilai perbandingannya pada setiap pengujian yang dilakukan. Performa *elapsed time* dikatakan bagus jika nilainya semakin rendah. Pengujian *elapsed time* terhadap *single server* dan *source* IP pada *server* nginx memiliki nilai yang cenderung sama tinggi. Kemudian untuk perbandingan pengujian *single server* dengan *load balancing* pada *instance* server apache maupun nginx dengan algoritma *round robin* dan *least connection* menunjukan perbedaan yang cukup signifikan ketika menerima beban mulai dari 200 *concurrent users*. Terjadinya peningkatan jumlah beban yang diterima oleh *single server* dan *source* IP dikarenakan kedua sistem tersebut hanya bisa melayani dengan satu *server*. Sedangkan algortima *round robin* dan *least connection* dapat membagi jumalah beban *request* tersebut kepada kedua dan ketiga *instances* yang tersedia.

4.5.2.3 Analisis *Data Transferred* Nginx

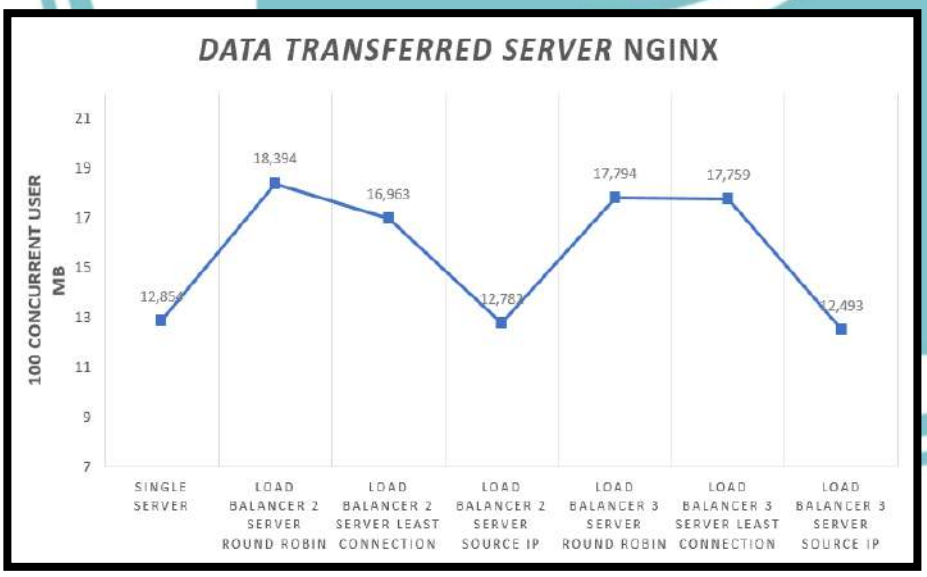
Dari grafik dibawah ini menunjukan perbandingan *data transferred server* nginx dengan HTTP *request* dari siege sebesar 50 *concurrent users*. Data yang didapat dari hasil pengujian pada *single server*, algoritma *round robin, least connection* dan *source IP* menunjukan perbedaan nilai *data transferred*. Data dari algoritma *round robin* dan *least connection* menunjukan peningkatan dan memiliki nilai yang tidak terlalu jauh satu sama lain. Sedangkan untuk *single server* dan *source IP*

menunjukan bahwa nilainya dibawah dari algoritma *round robin* dan *least connection*.



Gambar 4.41 Grafik perbandingan *data transferred* nginx 50 *concurrent users*

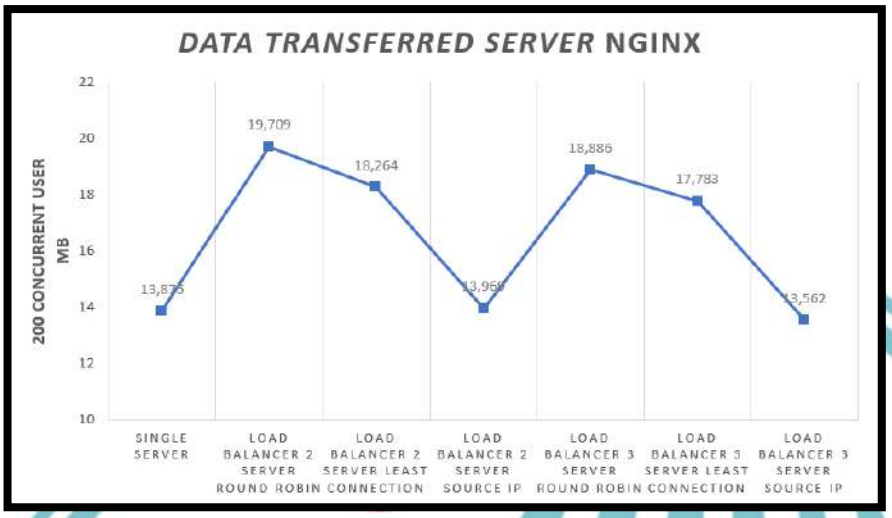
Grafik dibawah ini menunjukan data pengujian data transferred dari *server* nginx dengan HTTP *request* dari siege sebesar 100 *concurrent users*. Grafik dari *single server* dan *source* IP menunjukan nilai yang tidak berbeda jauh satu sama lain, namun berbeda jauh jika dibandingkan dengan nilai pada algoritma *round robin* ataupun *least connection.*



Gambar 4.42 Grafik perbandingan *data transferred* nginx 100 *concurrent users*

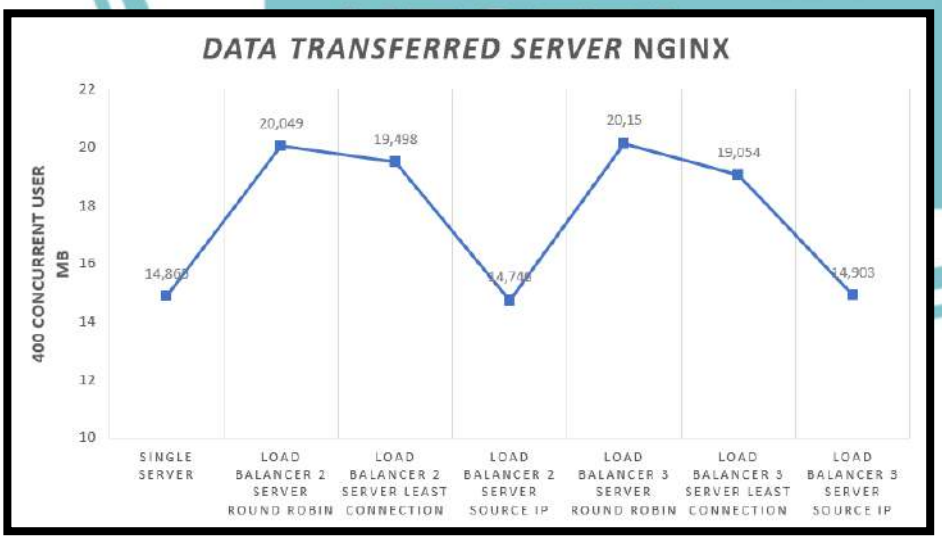
Data berikutnya menjukan *data transferred* pada *server* nginx dengan HTTP *request* dari siege sebesar 200 *concurrent users*. Sama seperti pengujian terhadap *instance server* apache bahwa grafik dari algoritma *round robin* dan *least connection* menunjukan nilai yang tidak berbeda jauh satu sama lain, sedangkan

single server dan *source* IP menunjukan perbedaan yang cukup signifikan dengan algoritma *round robin* ataupun *least connection*.



Gambar 4.43 Grafik perbandingan *data transferred* nginx 200 *concurrent users*

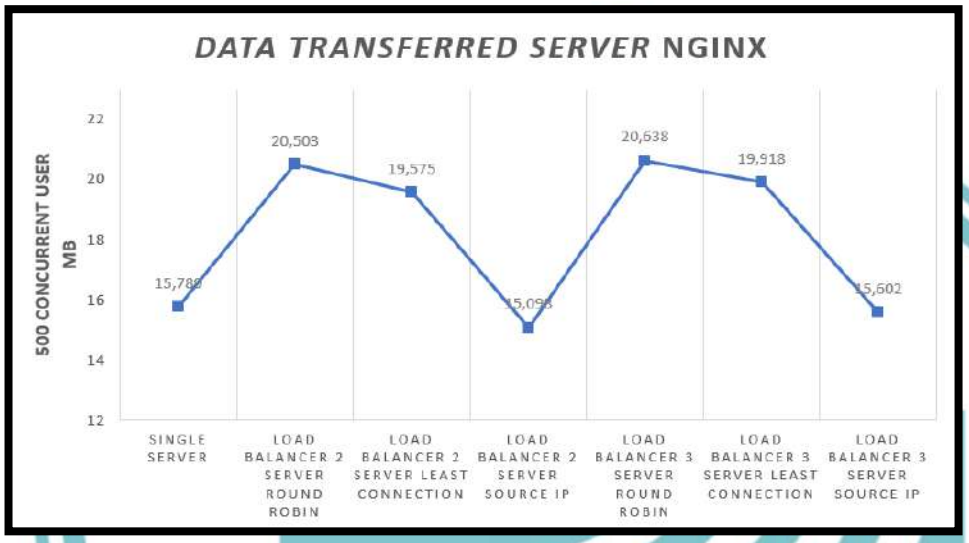
Data berikutnya menunjukan perbandingan *data transferred* pada *server* nginx dengan HTTP *request* dari siege sebesar 400 *concurrent users*. Sama seperti pengujian pada *instance server* apache bahwa grafik pada algoritma *round robin* dan *least connection* menunjukan pengingkatan dan tidak memiliki nilai yang berbeda jauh satu sama lain. Sedangkan jika dibandingkan dengan *single server* dan *source IP* menunjukan perbedaan yang cukup singnifikan berbeda dari nilai yang di dapatkan oleh algoritma *round robin* dan *least connection*.



Gambar 4.44 Grafik perbandingan *data transferred* nginx 400 *concurrent users*

Grafik selanjutnya menunjukan perbandingan *data transferred* pada *server* nginx dengan HTTP request dari siege sebesar 500 *concurrent users*. Sama seperti

pengujian yang dilakukan pada *instance server* apache bahwa grafik dari *single server* dan *source IP* tidak berbeda jauh satu sama lain. Namun jika dibandingkan dengan nilai dari algoritma *round robin* dan *least connection* yang menunjukan data yang cukup berbeda.

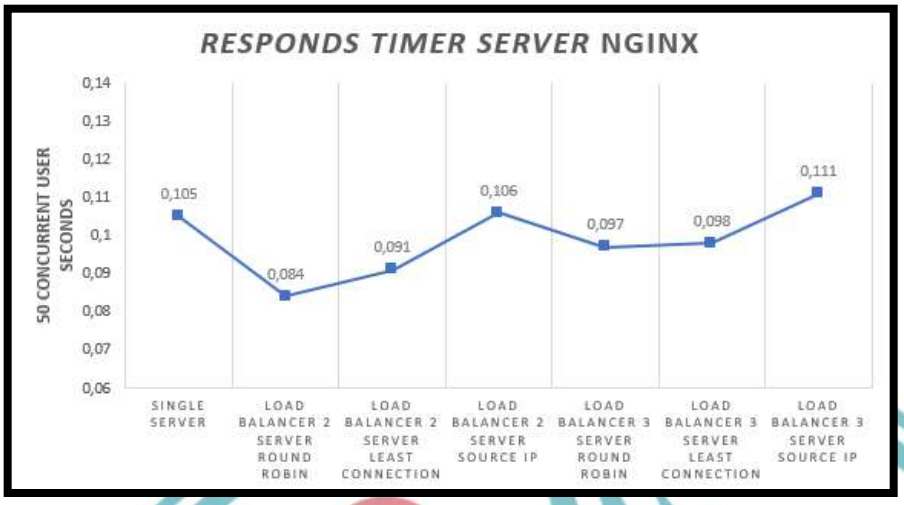


Gambar 4.45 Grafik perbandingan *data transferred* nginx *500 concurrent users*

Dari data keseluruhan grafik pengujian *data transferred* pada *server* nginx diatas menunjukan nilai perbandingannya pada setiap pengujian yang dilakukan. Jika *server* bisa menangani *request* dengan cepat, jadi semakin tinggi nilai *data transferred* bisa dikatakan bahwa performanya bagus. Dari seluruh grafik diatas data grafik dari algorima *round robin* dan *least connection* menunjukan nilai yang lebih tinggi jika dibandingkan dengan grafik dari *single server* dan *source IP*. *Single server* dan *source IP* tidak bisa menangani beban *request* dengan baik dikarenakan *single server* maupun *source IP* hanya memiliki satu *server* untuk menangani *request* sehingga tidak bisa menerima beban *request* yang tinggi.

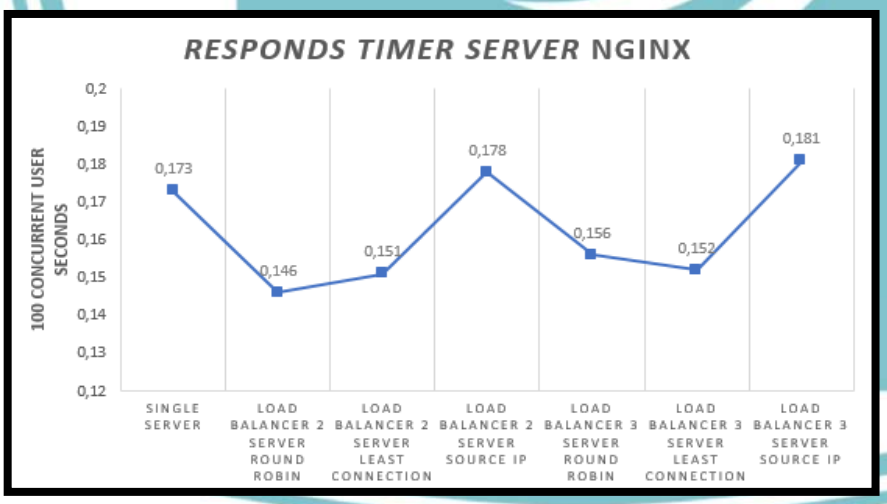
4.5.2.4 Analisis *Responds Time* Nginx

Dari grafik dibawah ini menunjukan bahwa perbandingan *responds time* pada *server* nginx dengan HTTP *request* dari siege sebesar 50 *concurrent users*. Data yang didapat dari hasil pengujian pada *single server* dan source IP menunjukan peningkatan dan nilainya pun lebih tinggi dengan algoritma *round robin* dan *least connection*. Performa *responds time* dikatakan bagus jika nilainya menjadi semakin kecil yang diberikan dari *server* tersebut.



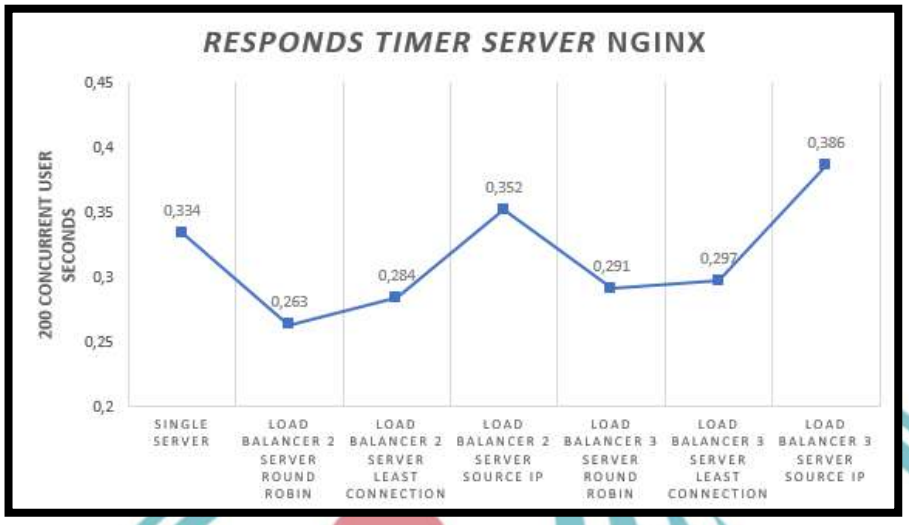
Gambar 4.46 Grafik perbandingan *responds time* nginx 50 *concurrent users*

Grafik berikutnya menunjukan perbandingan *responds time* pada *server* nginx dengan HTTP *request* dari siege sebesar 100 *concurrent users*. Pada grafik *single server* dan *source* IP menunjukan peningatan waktu lebih besar jika dibandingkan dengan algoritma *round robin* dan *least connection* menunjukan waktu yang lebih kecil. Peningkatan waktu terjadi akibat *server* tidak mampu menangani beban sehingga membutuhkan waktu yang lebih lama untuk menyelesaikan *request*.



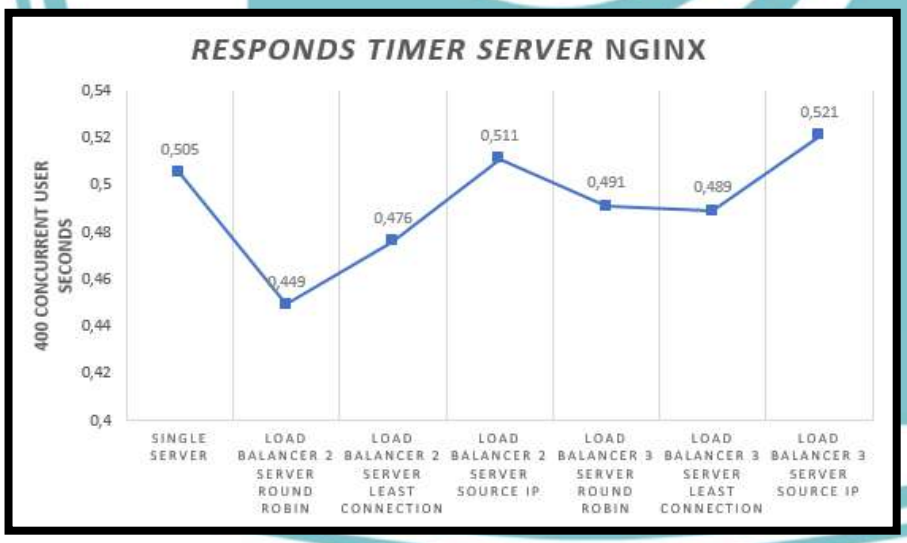
Gambar 4.47 Grafik perbandingan *responds time* nginx 100 *concurrent users*

Grafik dibawah ini menunjukan perbandingan *responds time* pada *server* nginx dengan HTTP *request* dari siege sebesar 200 *concurrent users*. Algoritma *round robin* dan *least connection* memiliki nilai yang lebih kecil dibandingkan dengan *single server* dan source IP. Hal tersebut dikarenakan algoritma *round robin* dan *least connection* memiliki 2 *server* dan 3 *server* oleh sebab itu waktu yang dibutuhkan relatif lebih sedikit untuk menyelesaikan *request* dari *client*.



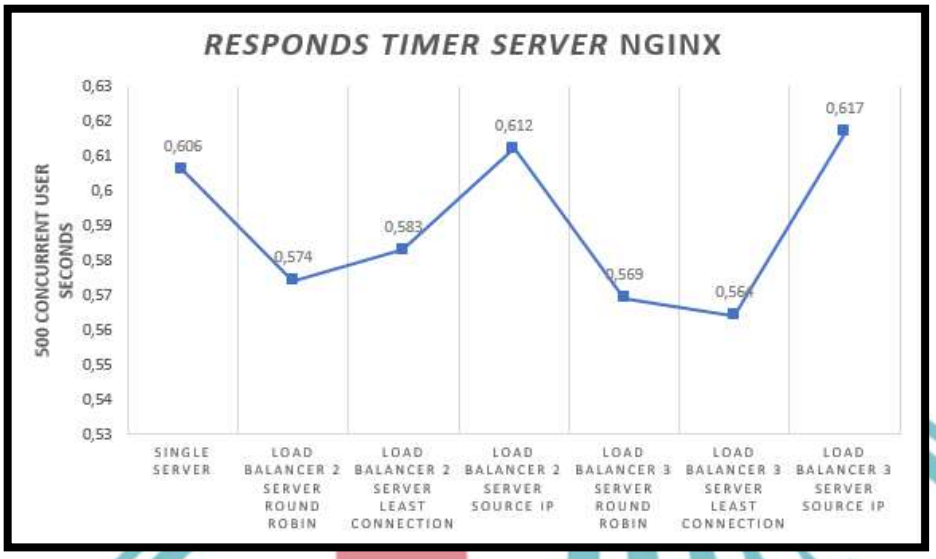
Gambar 4.48 Grafik perbandingan *responds time* nginx 200 *concurrent users*

Grafik selanjutnya menunjukan perbandingan *responds time* pada *server* nginx dengan HTTP *request* dari siege sebesar 400 *concurrent users*. Grafik ini menunjukan peningkatan pada *single server* dan *source* IP dan memiliki nilai yang tidak terlalu jauh. Dibandingkan dengan algoritma *round robin* dan *least connection* memiliki nilai yang cukup signifikan berbeda.



Gambar 4.49 Grafik perbandingan *responds time* nginx 400 *concurrent users*

Grafik dibawah ini menunjukan perbandingan *responds time* pada *server* nginx dengan HTTP *request* dari siege sebesar 500 *concurrent users*. Algoritma *round robin* dan *least connection* menunjukan penurunan dan memiliki nilai yang tidak berbeda jauh. Sedangkan untuk *single server* dan *source* IP menunjukan nilai yang lebih tinggi dan cukup signifikan perbedaannya dengan algoritma *round robin* dan *least connection*.

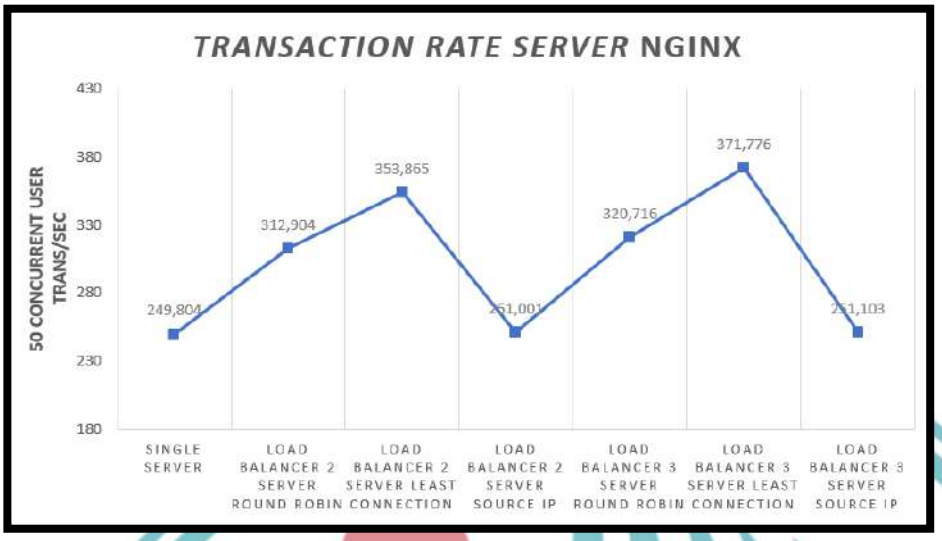


Gambar 4.50 Grafik perbandingan *responds time* nginx 500 *concurrent users*

Dari data keseluruhan grafik pengujian *responds time* pada *server* nginx diatas menunjukan nilai perbandingannya pada setiap pengujian yang dilakukan. Hasil dari pengujian keseluruhan menunjukan bahwa semakin kecil nilai *responds time* yang didapat maka akan semakin bagus performa yang diberikan *server* tersebut. Grafik *single server* dan *source* IP menunjukan peningkatan yang cukup drastis saat jumlah HTTP *request* sebanyak 200 *concurrent users*. Peningkatan nilai dialami karena *single server* dan *source* IP hanya memiliki satu *server* sehingga tidak dapat membagi beban kepada *server* lainnya sehingga membutuhkan waktu yang lebih lama untuk menyelesaikan *request* dari client. Berbeda dengan algoritma *round robin* dan *least connection* yang memiliki waktu yang relatif lebih kecil jika dibandingkan dengan *single server* maupun *source* IP dikarenakan memiliki 2 dan 3 *server* untuk membagi jumlah beban sehingga dapat menyelesaikan *request* dengan cepat.

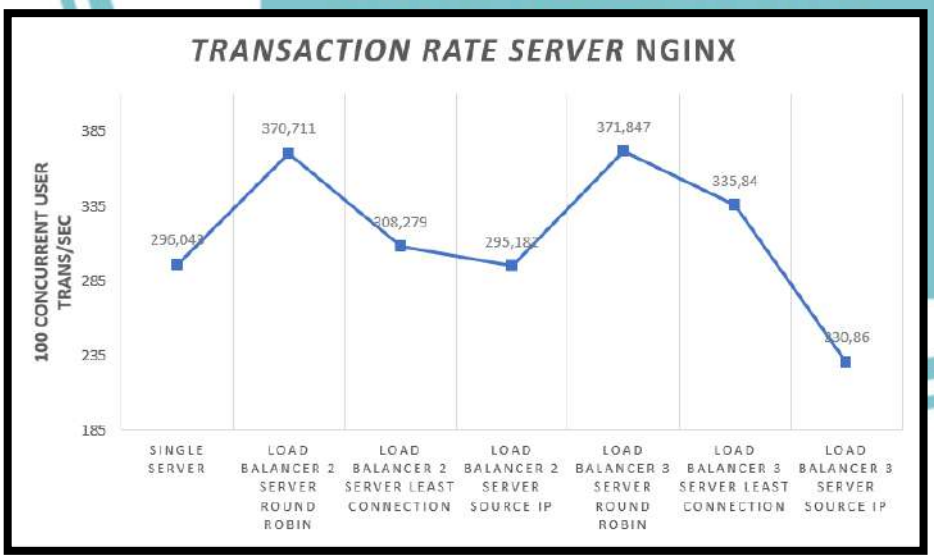
4.5.2.5 Analisis *Transaction Rate* Nginx

Dari grafik dibawah ini menunjukan bahwa perbandingan *transaction rate* pada *server* nginx dengan HTTP *request* dari siege sebesar 50 *concurrent users*. Grafik pada algoritma *round robin* dan *least connection* menunjukan peningkatan nilai sedangkan algoritma *source* IP menunjukan penurunan nilai. Pada pengujian dengan jumlah HTTP *request* sebanyak 50 *concurrent users* tidak menunjukan nilai yang jauh berbeda antara *single server*, *round robin*, *least connection* dan *source* IP.



Gambar 4.51 Grafik perbandingan *transaction rate* nginx 50 *concurrent users*

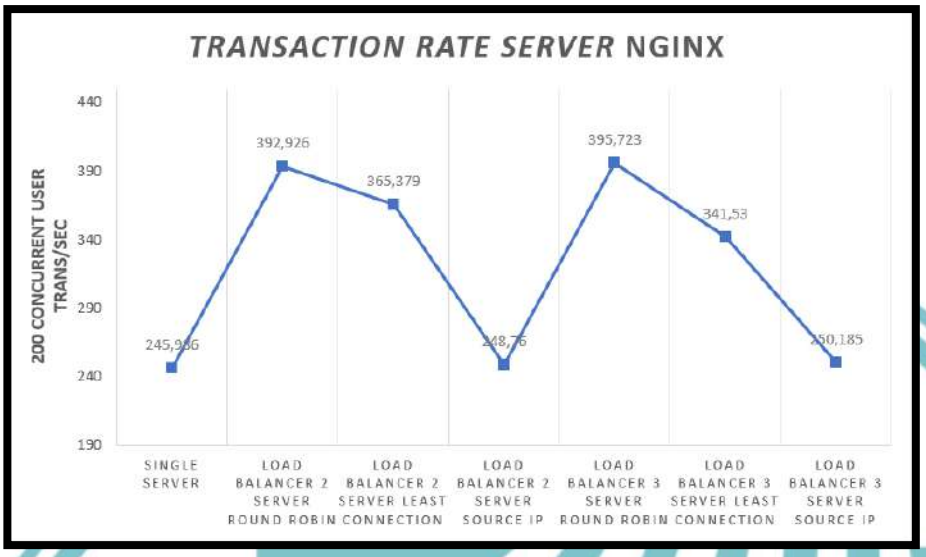
Data perbandingan selanjutnya menunjukan data *transaction rate* pada *server* nginx dengan HTTP *request* dari siege sebesar 100 *concurrent users*. Algoritma *round robin* menunjukan peningkatan nilai antara pengujian 2 *server* ke 3 *server*, Sedangkan algoritma *least connection* dan *source* IP menunjukan penurunan. Pada pengujian dengan jumlah HTTP request sebanyak 100 *concurrent users* tidak menunjukan nilai yang jauh berbeda antara *single server, round robin, least connection* dan *source* IP.



Gambar 4.52 Grafik perbandingan *transaction rate* nginx 100 *concurrent users*

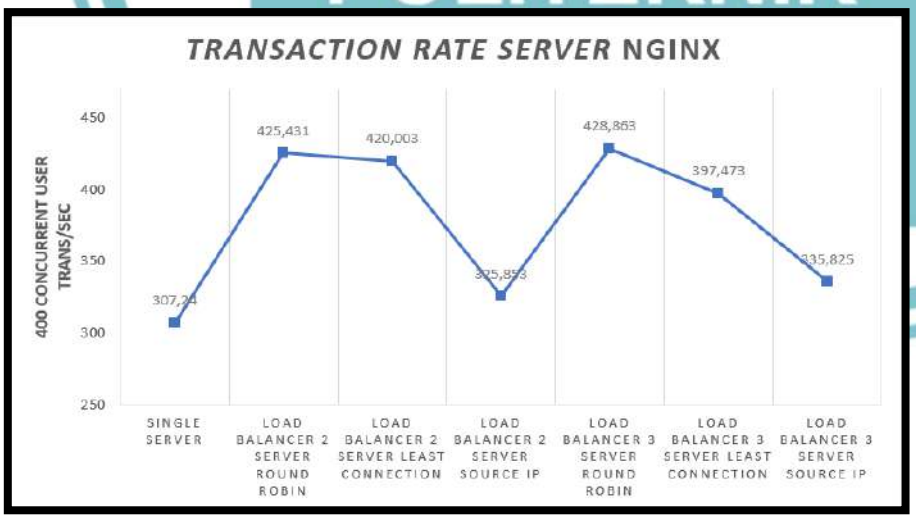
Grafik dibawah ini menunjukan perbandingan *transaction rate* pada *server* nginx dengan HTTP *request* dari siege sebesar 200 *concurrent users*. Pada grafik kali ini *single sever* dan *sourc*e IP menunjukan nilai yang cukup rendah dibandingan

dengan algoritma *round robin* dan *least connection*. Semakin rendah nilai yang didapat oleh *server* tersebut maka semakin buruk performanya.



Gambar 4.53 Grafik perbandingan *transaction rate* nginx 200 *concurrent users*

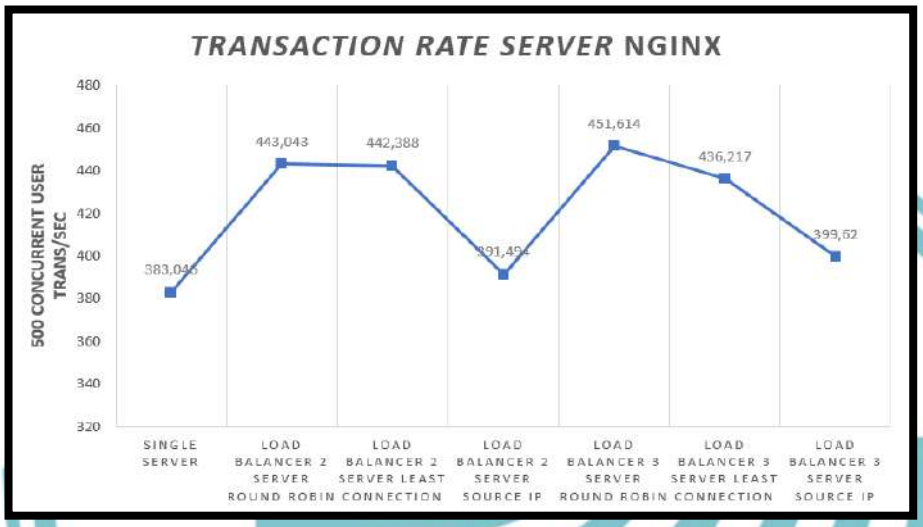
Data selanjutnya merupakan perbandingan *transaction rate* pada *server* nginx dengan HTTP *request* dari siege sebesar 400 *concurrent users*. Grafik pada algoritma *round robin* dan *least connection* menunjukan peningkatan nilai pada pengujian 2 *server* dan 3 *server*. Data sebaliknya ditunjukan oleh *single server* dan *source* IP yang menunjukan nilai dibawah dari algoritma *round robin* dan *least connection*.



Gambar 4.54 Grafik perbandingan *transaction rate* nginx 400 *concurrent users*

Data selanjutnya merupakan perbandingan *transaction rate* pada *server* nginx dengan HTTP *request* dari siege sebesar 500 *concurrent users*. Grafik pada algorit*ma round robin* dan *least connection* menunjukan peningkatan nilai pada

pengujian 2 *server* dan 3 *serve*r. Pada grafik kali ini *single server* dan *source* IP menunjukan nilai yang cukup rendah dibandingan dengan algoritma *round robin* dan *least connection*. Semakin tinggi nilai *transaction rate* berbanding lurus dengan semakin bagus performa yang akan didapat.



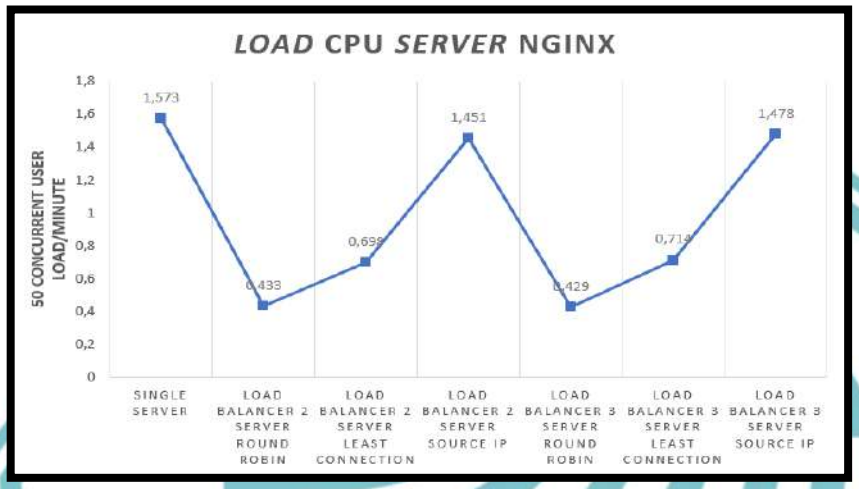
Gambar 4.55 Grafik perbandingan *transaction rate* nginx 500 *concurrent users*

Dari data keseluruhan grafik pengujian *transaction rate* pada *server* nginx diatas menunjukan nilai perbandingannya pada setiap pengujian yang dilakukan. Performansi *transaction rate* dikatakan bagus berbanding lurus dengan semakin tinggi nilai yang didapat. Grafik *transaction rate* pada algoritma *round robin* dan *least connection* dengan menggunakan load balancing menunjukan peningkatan, dikarenakan dapat membagi sejumlah *request* kepada *server* lainnya sehingga mampu mengoptimalkan *server* tersebut. Saat *server* diberikan *request* sebanyak 200 *concurrent users*, *single server* maupun *source* IP menunjukan penurunan menyebabkan *server* tersebut tidak bekerja secara optimal. Pada sistem yang menggunakan *load balancing* dengan algoritma *round robin* dan *least connection* dapat melayani jumlah *request* yang lebih besar dibandingkan dengan sistem *single server* dan algoritma *source* IP.

4.5.2.6 Analisis *Load* CPU Nginx

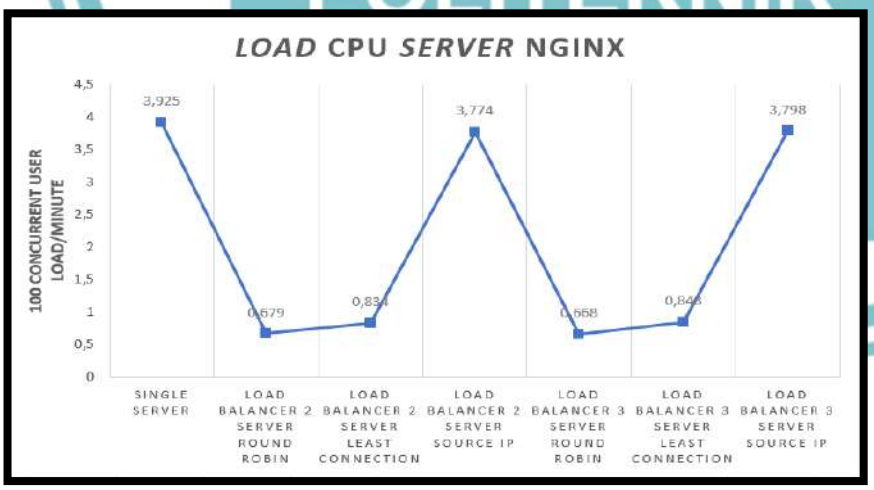
Untuk mengukur seberapa besar penggunaan *resource* dari segi CPU *client* pada saat menjalankan sistem dengan *load balancing* dan *single server* merupakan tujuan dari pengujian ini. Dari grafik dibawah ini menunjukan bahwa perbandingan *load cpu* pada *server* nginx dengan HTTP *request* dari siege sebesar 50 *concurrent users*. Data yang didapat dari hasil sama seperti pada *server* apache dimana pengujian

menunjukan bahwa algoritma *round robin* dan algoritma *least connection* memiliki nilai yang tidak terlalu jauh jika dibandingkan dengan *single server* dan algoritma *source* IP. Performa CPU dikatakan bagus berbanding lurus dengan nilai yang dihasilkan semakin kecil.



Gambar 4.56 Grafik perbandingan *load cpu* nginx 50 *concurrent users*

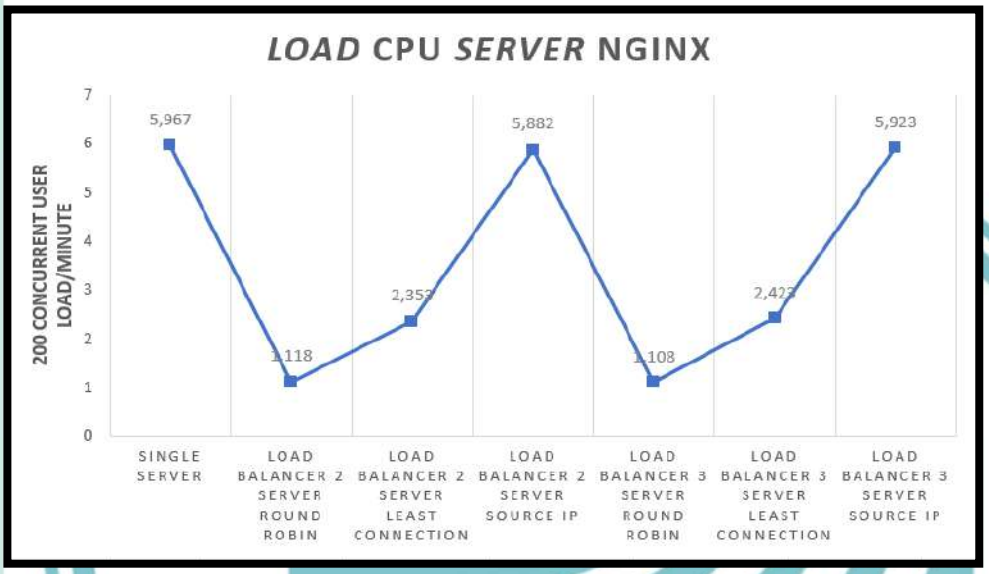
Data perbandingan selanjutnya menunjukan data *load cpu* pada *server* nginx dengan HTTP *request* dari siege sebesar 100 *concurrent users*. Data yang didapat dari hasil pengujian menunjukan bahwa algoritma *source* IP dan *single server* memiliki nilai yang sama-sama tinggi dibangdingkan dengan algoritma *round robin* dan algoritma *least connection.*



Gambar 4.57 Grafik perbandingan *load cpu* nginx 100 *concurrent users*

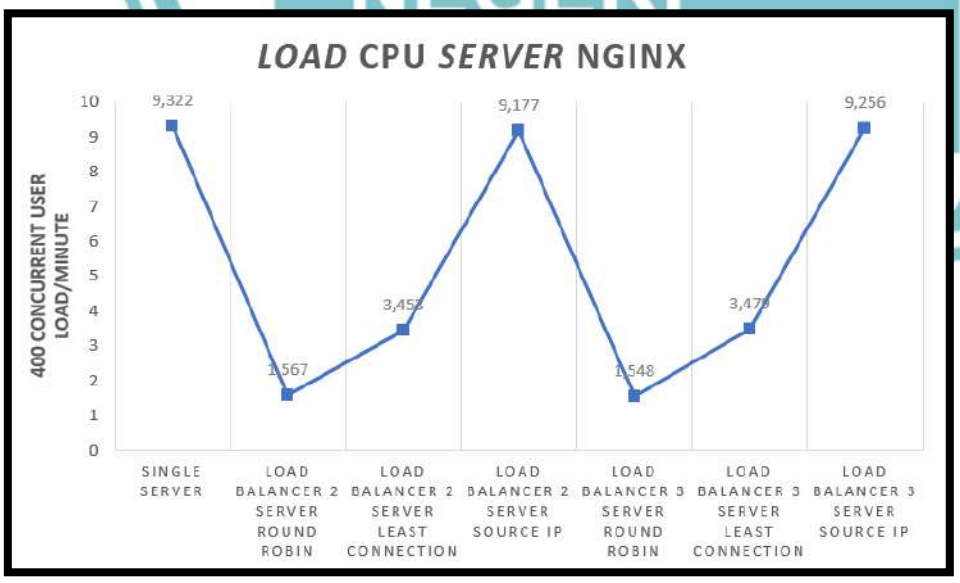
Data selanjutnya menunjukan data *load cpu* pada *server* nginx dengan HTTP *request* dari siege sebesar 200 *concurrent users*. Data yang didapat dari hasil pengujian menunjukan peningkatan yang signifikan dari *single server* dan

algoritma *source* IP dibandingkan dengan algoritma *round robin* dan algoritma *least connection*. Dikarenakan *single server* dan algoritma *source* IP tidak bisa membagi beban sehingga kinerja vcpu menjadi lebih berat dan tidak berjalan secara optimal.



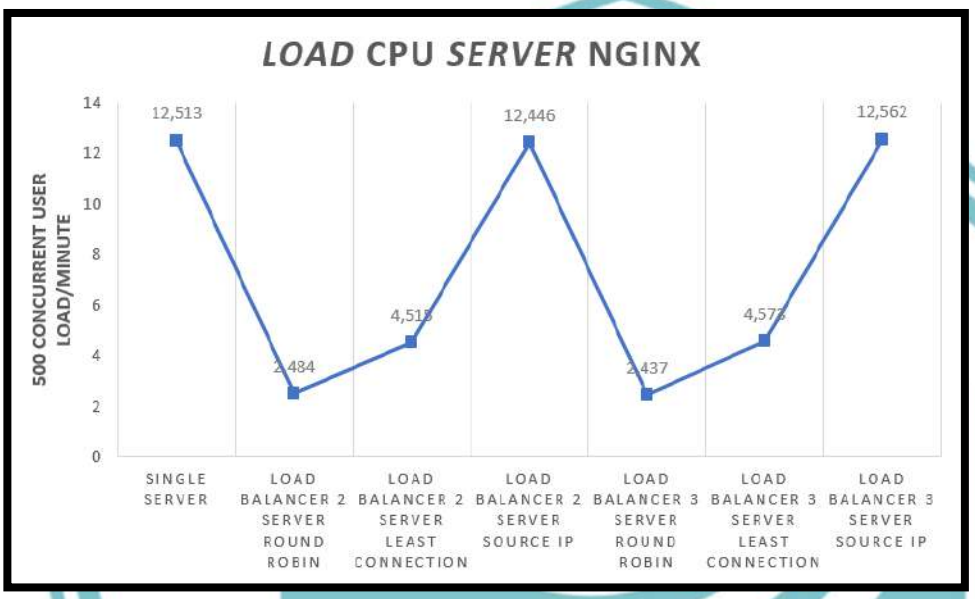
Gambar 4.58 Grafik perbandingan *load cpu* nginx 200 *concurrent users*

Data selanjutnya merupakan perbandingan *load cpu* pada *server* nginx dengan HTTP *request* dari siege sebesar 400 *concurrent users*. Grafik pada algoritma *round robin* dan *least connection* menunjukan peningkatan nilai pada pengujian 2 *server* dan 3 *server*. Selain itu, *single server* dan *source* IP menunjukan nilai lebih tinggi dari algoritma *round robin* dan *least connection*.



Gambar 4.59 Grafik perbandingan *load cpu* apache 400 *concurrent users*

Grafik dibawah ini menunjukan perbandingan *load cpu* pada *server* nginx dengan HTTP *request* dari siege sebesar 500 *concurrent users*. Pada grafik kali ini *single server* dan *source* IP menunjukan nilai yang sangat tinggi dibandingan dengan algoritma *round robin* dan *least connection*. Semakin tinggi nilai yang didapat oleh *server* tersebut maka semakin buruk performanya.



Gambar 4.60 Grafik perbandingan *load cpu* nginx 500 *concurrent users*

Dari data keseluruhan grafik pengujian *load cpu* pada *server apache* diatas menunjukan nilai perbandingannya pada setiap pengujian yang dilakukan. Performansi *load cpu* dikatakan bagus berbanding lurus dengan semakin rendah nilai yang didapat. Grafik *load cpu* pada algoritma *round robin* dan *least connection* dengan menggunakan *load balancing* menunjukan nilai yang rendah dikarenakan dapat membagi beban dari *client* dibagi kepada dua dan tiga *server* yang berarti dua vcpu dan tiga vcpu sehingga kinerja masing-masing vcpu menjadi lebih ringan dan hasil pengujian pun menjadi lebih optimal. Sedangkan untuk grafik *single server* dan algoritma *source* IP menunjukan nilai yang tingi hal tersebut dikarenakan setiap servernya hanya memiliki satu vcpu, sehingga tidak bisa mengoptimalkan kinerja.

4.5.3 Analisis Perbandingan Apache dan Nginx

Data dari pengujian kedua *web server* di Openstack dianalisis untuk membandingkan perfomansi *web server* mana yang lebih baik antara apache dan nginx.

4.5.3.1 Analisis Perbandingan 2 *Server*

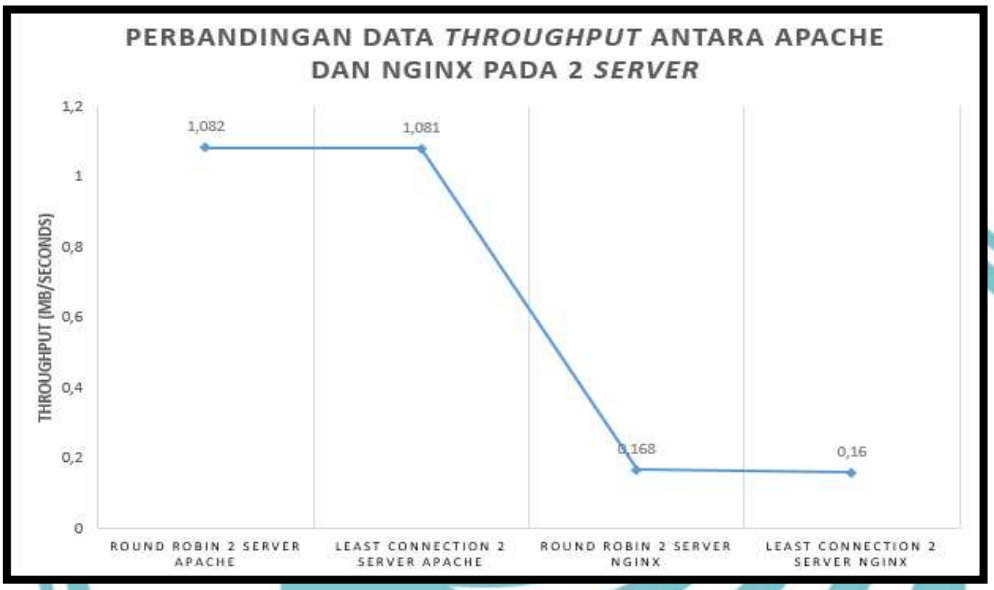
Data dibawah ini menunjukan perbandingan antara *instance server* apache dan nginx pada pengujian 2 *server* dengan variabel *throughput, elapsed time, data transferred, responds time, transaction rate* dan *load cpu* dengan menggunakan algoritma *round robin* dan *least connection*. Data ini dapat dilihat pada tabel 4.11.

Tabel 4.13 Perbandingan Data Statik Antara server apache dan Nginx pada 2 Server

Perbandingan Data Statik Antara *Server* Apache dan Nginx pada 2 *Server*				
Kategori	***Round Robin 2 Server Apache***	***Round Robin 2 Server Nginx***	***Least Connection 2 Server Apache***	***Least Connection 2 Server Nginx***
Throughput (MB/*seconds*)	1,082	0,168	1,081	0,16
Elapsed Time (*seconds*)	59,499	59,582	59,638	59,594
Data Transferred (MB)	64,358	20,503	64,4	19,575
Responds Time (seconds)	0,654	0,574	0,665	0,583
Transacion Rate (trans/sec)	413,461	443,043	389,102	442,388
Load CPU (*load/minute*)	2,407	2,484	4,371	4,515

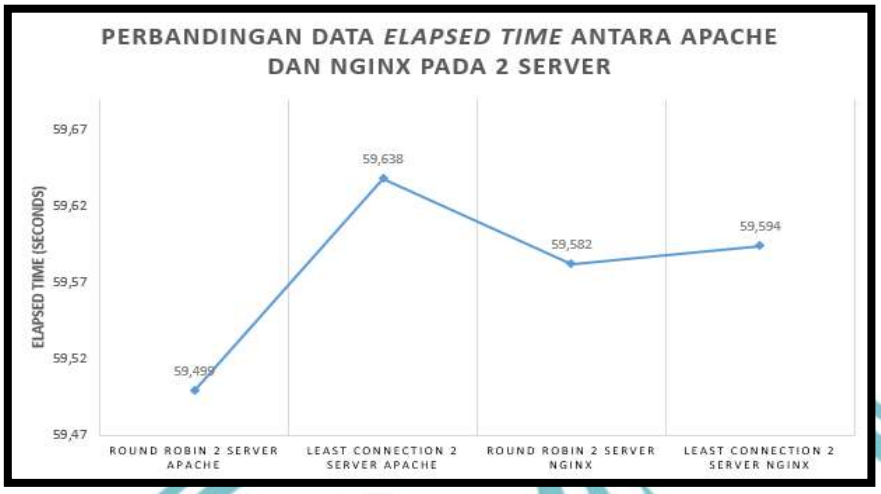
Grafik dibawah ini menunjukan perbandingan *throughput* antara *instance server* apache dan nginx pada pengujian 2 *server* dengan menggunakan algoritma *round robin* dan *least connection*. Grafik *throughput* dari *server* nginx menunjukan nilai yang jauh lebih kecil dibandingkankan dengan grafik dari *instance server* apache dikarenakan *server* nginx disetting *default* dan tidak memanfaatkan *bandwidth* secara optimal, sedangkan *server* apache hanya disetting agar dapat menerima *request* lebih dari 500 *concurrent users* sesuai dengan rekomendasi aplikasi siege. Lalu untuk kinerja *instance server* apache menunjukan nilai *throughput* yang bagus dengan nilai 1.082 MB/seconds pada algoritma *round robin* dan 1.081 MB/seconds pada algoritma *least connection*. *Throughput* dinyatakan bagus performanya jika

nilai *throughput* cenderung naik seiring dengan bertambahnya jumlah *request* yang diberikan. Pada pengujian *throughput instance server* apache lebih unggul dari *server* nginx.



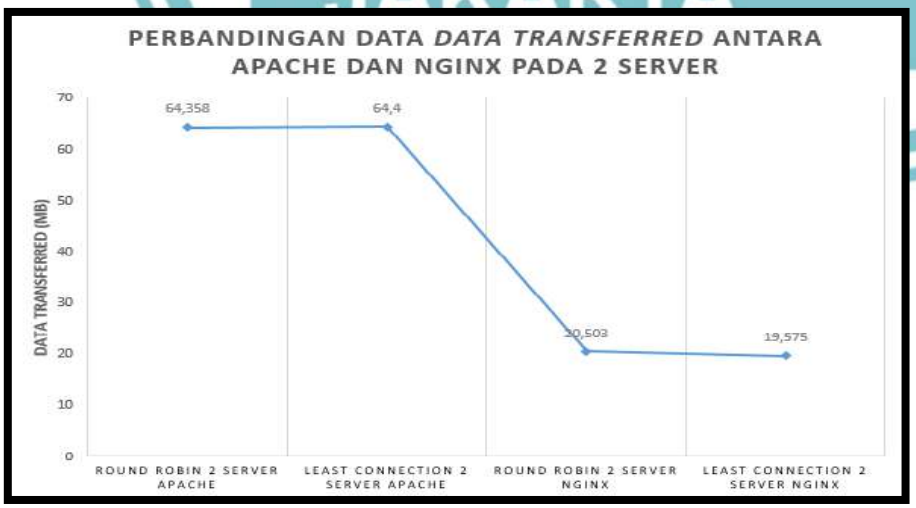
Gambar 4.61 Grafik perbandingan data *throughput* antara apache dan nginx pada 2 *server*

Grafik berikutnya menunjukan perbandingan *elapsed time* antara *instance server* apache dan nginx pada pengujian 2 *server* dengan menggunakan algoritma *round robin* dan *least connection*. Grafik pada *instance server* apache menunjukan hasil yang lebih rendah pada algoritma *round robin* dengan nilai 59.499 seconds jika dibandingkan dengan *server* nginx pada algoritma *round robin* dengan nilai 59.582 seconds. Sedangkan untuk nilai *elapsed time* pada *instance server* apache pada algoritma *least connection* menunjukan peningkatan yang cukup signifikan dengan nilai 59.638 seconds jika dibandingkan dengan server nginx pada algoritma *least connection* dengan nilai 59.594 seconds. *Elapsed time* merupakan durasi keseluruhan tes yang dijalankan oleh siege dari *client*, performanya dikatakan bagus jika nilainya semakin rendah karena mampu menyelesaikan *request* dengan cepat. Pada pengujan *elapsed time instance server* apache lebih unggul dengan rata- rata nilai 59.568 seconds dari pada *server* nginx dengan rata-rata nilai 59.588 seconds.



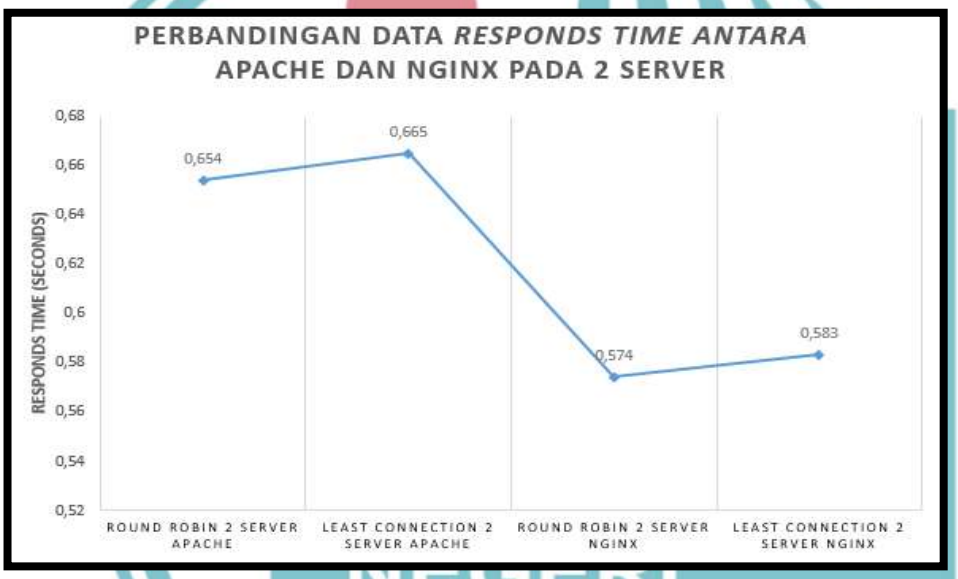
Gambar 4.62 Grafik perbandingan data *elapsed time* antara apache dan nginx pada 2 *server*

Grafik berikutnya menunjukan perbandingan *data transferred* antara *instance server* apache dan nginx pada pengujian 2 *server* dengan menggunakan algoritma *round robin* dan *least con*nection. Grafik pada *server* nginx menunjukan penurunan pada algoritma *round robin* dengan nilai 20.503 MB dan pada algoritma *least connection* menunjukan nilai 19.575 MB. Sedangkan grafik pada *instance server* apache menunjukan adanya peningkatan pada algoritma *round robin* dengan nilai 64.358 MB dan pada algoritma *least connection* dengan nilai 64.4 MB. Performansi *data transferred* dikatakan bagus jika nilainya semakin tinggi artinya *server* menangani *request* yang datang lebih cepat sehingga data yang ditransfer besarnya bisa lebih besar karena *server* bisa memproses datanya dengan lebih cepat. Pada pengujian *data transferred instance server* apache lebih unggul dibandingkan dengan *server* nginx.



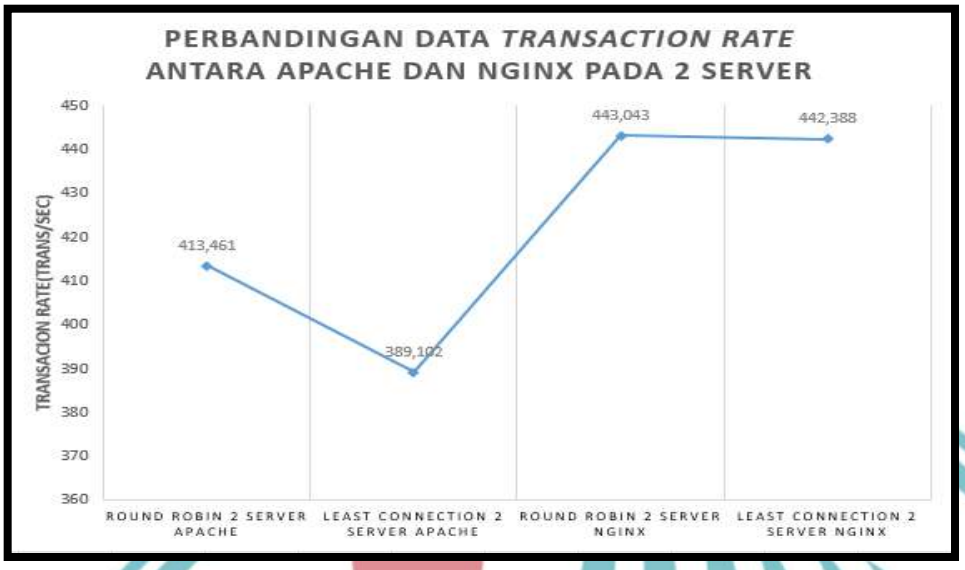
Gambar 4.63 Grafik perbandingan *data transferred* antara apache dan nginx pada 2 *server*

Grafik berikutnya menunjukan perbandingan *responds time* antara *instance server* apache dan nginx pada pengujian 2 *server* dengan menggunakan algoritma *round robin* dan *least connection*. Grafik pada *instance server* apache dengan menggunakan algoritma *round robin* memiliki nilai 0.654 seconds dan pada *algoritma least connection* memiliki nilai 0.665 seconds. Sedangkan grafik pada *server* nginx dengan menggunakan algoritma *round robin* memiliki nilai 0.574 seconds dan pada algoritma *least connection* memiliki nilai 0.583 seconds. Performa *responds time* dikatakan bagus jika semakin kecil nilainya karena dapat menyelesaikan *request* dari *client* dengan cepat. Pada pengujian *responds time* kali ini *server* nginx lebih unggul dibandingkan dengan *instance server* apache.



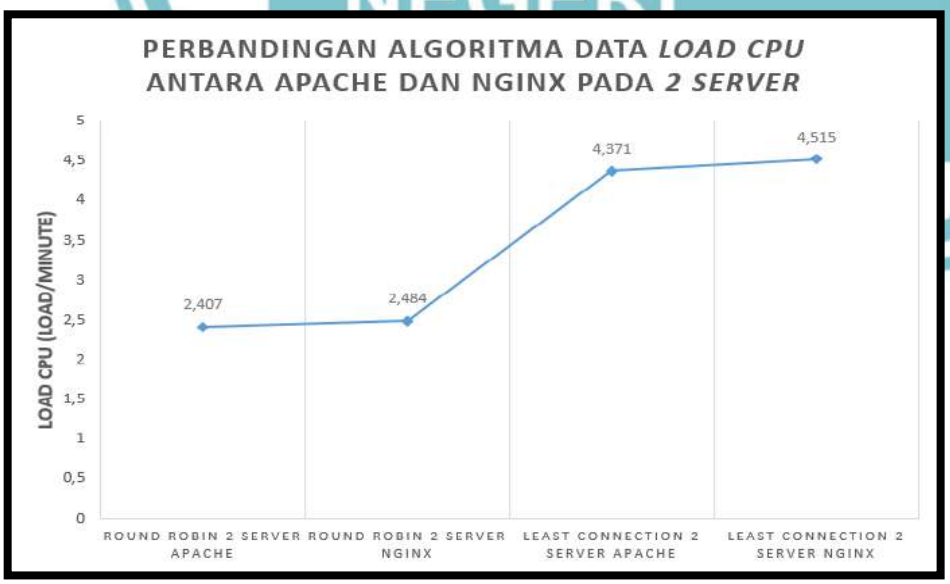
Gambar 4.64 Grafik perbandingan data *responds ti*me antara apache dan nginx pada 2 *server*

Grafik berikutnya menunjukan perbandingan *transaction rate* antara *instance server* apache dan nginx pada pengujian 2 *server* dengan menggunakan algoritma *round robin* dan *least connection*. Grafik pada *instance server* apache menunjukan penurunan dengan menggunakan algoritma *round robin* memiliki nilai 413.461 trans/sec dan pada algoritma *least connection* memiliki nilai 389.102 trans/sec . Sedangkan grafik pada *server* nginx dengan menggunakan algoritma *round robin* memiliki nilai 443.043 trans/sec dan pada algoritma *least connection* memiliki nilai 442.388 trans/sec. Performa *transaction rate* dikatakan bagus jika memiliki nilai yang cenderung naik sehingga dapat menjalankan *server* dengan optimal. Pada pengujian *transaction rate server* nginx lebih unggul dari pada *instance server* apache.



Gambar 4.65 Grafik perbandingan data *transaction rate* antara apache dan nginx pada 2 *server*

Grafik berikutnya menunjukan perbandingan *load cpu* antara *server* apache dan nginx pada pengujian 2 *server* dengan menggunakan algoritma *round robin* dan *least connection*. Grafik pada *server* apache menunjukan penurunan dengan menggunakan algoritma *round robin* memiliki nilai 2.407 *load/minute* dan pada algoritma *least connection* memiliki nilai 4.371 *load/minute*. Sedangkan grafik pada *server* nginx dengan menggunakan algoritma *round robin* memiliki nilai 2.484 *load/minute* dan pada algoritma *least connection* memiliki nilai 4.515 *load/minute*. Performa *load cpu* dikatakan bagus jika memiliki nilai yang cenderung menurun sehingga *load cp*u *server* apache lebih unggul dari pada *server* nginx.



Gambar 4.66 Grafik perbandingan data *load cpu* antara apache dan nginx pada 2 *server*

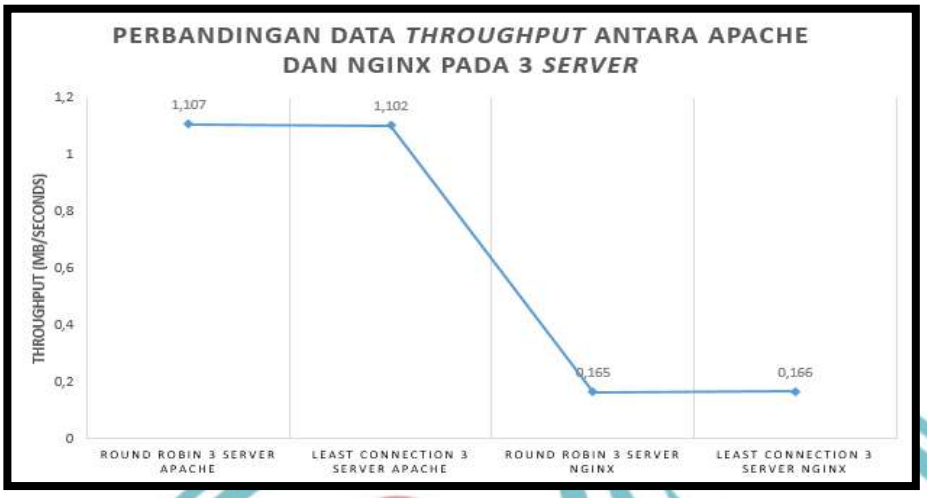
4.5.3.2 Analisis Perbandingan 3 *Server*

Data dibawah ini menunjukan perbandingan antara *instance server* apache dan nginx pada pengujian 3 *server* dengan variabel *throughput, elapsed time, data transferred, responds time*, *transaction rate* dan *load cpu* dengan menggunakan algoritma *round robin* dan *least connection*. Data ini dapat dilihat pada tabel 4.12.

Tabel 4.14 Perbandingan Data Statik Antara server apache dan Nginx pada 3 Server

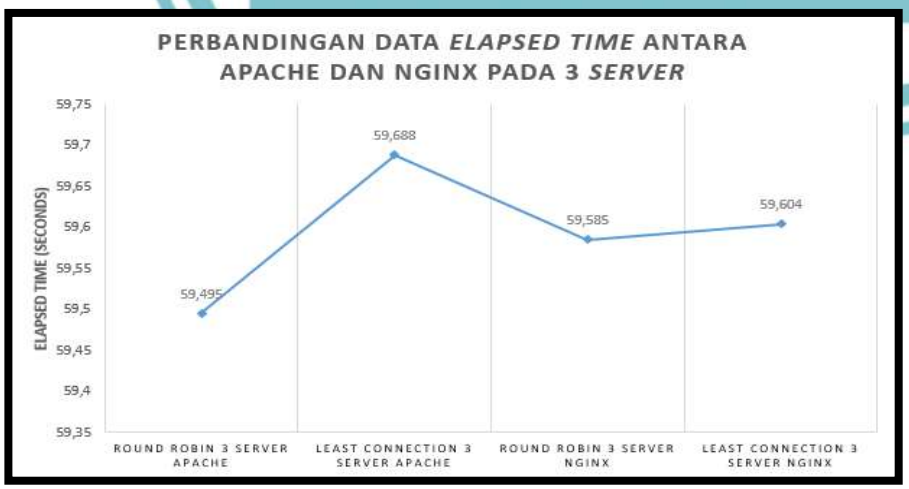
Perbandingan Data Statik Antara *Server* Apache dan Nginx pada 3 *Server*				
Kategori	***Round Robin 3 Server Apache***	***Round Robin 3 Server Nginx***	***Least Connection 3 Server Apache***	***Least Connection 3 Server Nginx***
Throughput (MB/*seconds*)	1,107	0,165	1,102	0,166
Elapsed Time (*seconds*)	59,495	59,585	59,688	59,604
Data Transferred (MB)	65,551	20,638	65,793	19,918
Responds Time (*seconds*)	0,652	0,569	0,652	0,564
Transacion Rate (trans/sec)	410,079	451,614	399,269	436,217
Load CPU (*load/minute*)	2,555	2,437	4,665	4,573

Grafik dibawah ini menunjukan perbandingan *throughput* antara *instance server* apache dan nginx pada pengujian 3 *server* dengan menggunakan algoritma *round robin* dan *least connection*. Grafik *throughput* dari *server* nginx menunjukan nilai yang jauh lebih kecil dibandingkan dengan grafik dari *instance server* apache dikarenakan *server* nginx disetting *default* dan tidak memanfaatkan *bandwidth* secara optimal sedangkan *server* apache hanya disetting agar dapat menerima lebih dari 500 *concurrent user* sesuai dengan rekomendasi aplikasi siege. Selanjutnya untuk kinerja *instance server* apache menunjukan peningkatan dari pengujian 2 *server* ke 3 *server* dan memiliki nilai *throughput* yang bagus dengan nilai 1.107 MB/seconds pada algoritma *round rob*in dan 1.102 MB/seconds pada algoritma *least connection*. *Throughput* dinyatakan bagus performanya jika nilai *throughput* cenderung naik seiring dengan bertambahnya besar *concurrent users*. Pada pengujian *throughput instance server* apache lebih unggul dari *serve*r nginx.



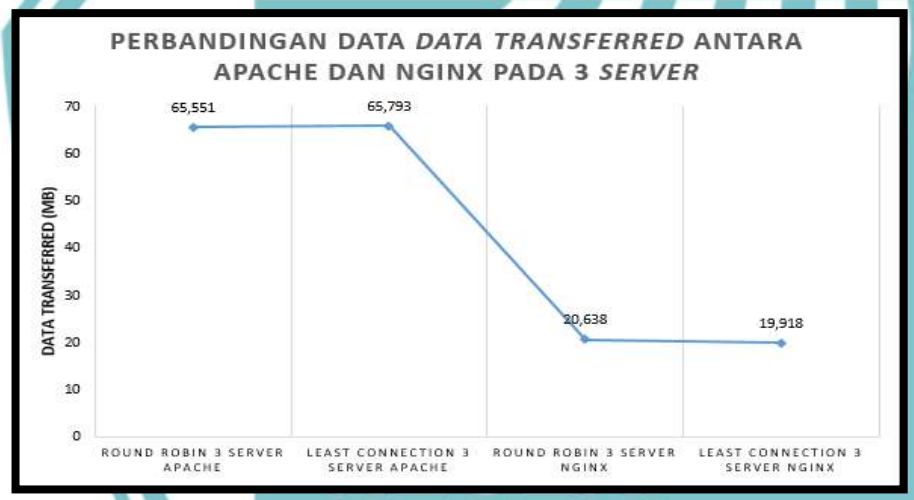
Gambar 4.67 Grafik perbandingan data *throughput* antara apache dan nginx pada 3 *server*

Grafik berikutnya menunjukan perbandingan *elapsed time* antara *instance server* apache dan nginx pada pengujian 3 *server* dengan menggunakan algoritma *round robin* dan *least connection*. Grafik pada *instance server* apache menunjukan hasil yang lebih rendah pada algoritma *round robin* dengan nilai 59.495 seconds jika dibandingkan dengan *server* nginx pada algoritma *round robin* dengan nilai 59.585 seconds. Sedangkan untuk nilai *elapsed time* pada *instance server* apache pada algoritma *least connection* menunjukan peningkatan yang cukup signifikan dengan nilai 59.688 seconds jika dibandingkan dengan *server* nginx pada algoritma *least connection* dengan nilai 59.604 seconds. *Elapsed time* merupakan durasi keseluruhan tes yang dijalankan oleh siege dari *client*, performanya dikatakan bagus jika nilainya semakin rendah karena mampu menyelesaikan *request* dengan cepat. Pada pengujan *elapsed time instance server* apache lebih unggul dengan rata- rata nilai 59.591 seconds dari pada *server* nginx dengan rata-rata nilai 59.595 seconds.



Gambar 4.68 Grafik perbandingan data *elapsed time* antara apache dan nginx pada 3 *server*

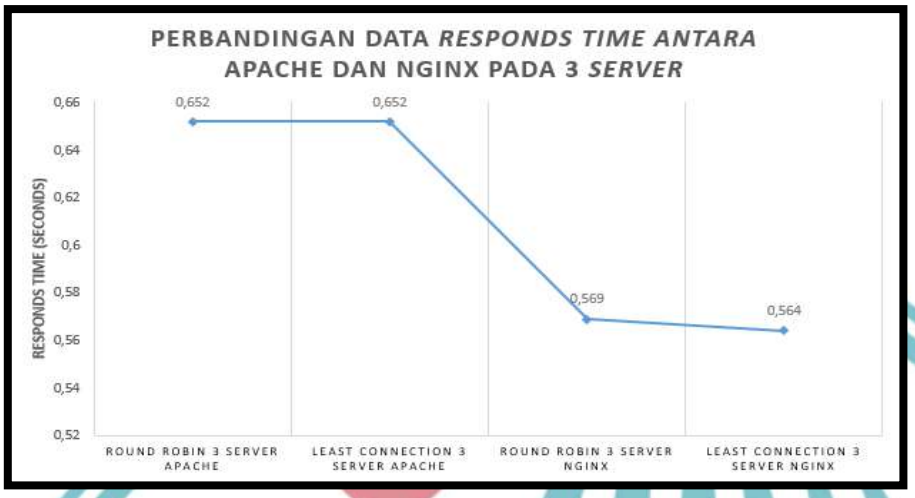
Grafik berikutnya menunjukan perbandingan *data transferred* antara *instance server* apache dan nginx pada pengujian 3 *server* dengan menggunakan algoritma *round robin* dan *least connection*. Grafik pada *server* nginx menunjukan penurunan pada algoritma *round robin* dengan nilai 20.638 MB dan pada algoritma *least connection* menunjukan nilai 19.918 MB. Sedangkan grafik pada *instance server* apache menunjukan adanya peningkatan pada algoritma *round robin* dengan nilai 65.551 MB dan pada algoritma *least connection* dengan nilai 65.793 MB. Performansi *data transferred* dikatakan bagus jika nilainya semakin tinggi artinya *server* menangani *request* yang datang lebih cepat sehingga data yang ditransfer besarnya bisa lebih besar karena *server* bisa memproses datanya dengan lebih cepat. Pada pengujian *data transferred instance server* apache lebih unggul dibandingkan dengan *server* nginx.



Gambar 4.69 Grafik perbandingan *data transferred* antara apache dan nginx pada 3 *server*

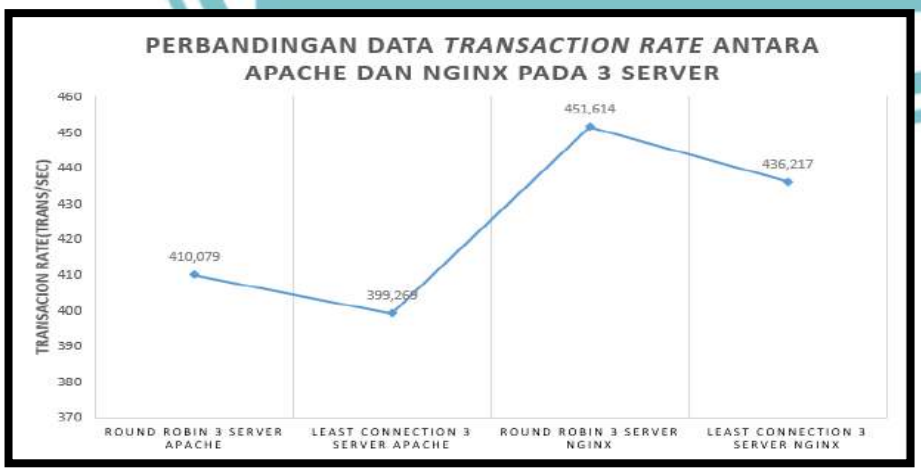
Grafik berikutnya menunjukan perbandingan *responds time* antara *instance server* apache dan nginx pada pengujian 3 *server* dengan menggunakan algoritma *round robin* dan *least connection*. Grafik pada *instance server* apache menunjukan penurunan dari 2 *server* ke pengujian 3 *server* dan memiliki nilai dengan menggunakan algoritma *round robin* memiliki nilai 0.652 seconds dan pada algoritma *least connection* memiliki nilai 0.652 seconds. Sedangkan grafik pada *server* nginx juga menunjukan penurunan dari 2 *server* ke pengujian 3 *server* dan memiliki nilai dengan menggunakan algoritma *round robin* memiliki nilai 0.569 seconds dan pada algoritma *least connection* memiliki nilai 0.564 seconds. Performa *responds time* dikatakan bagus jika semakin kecil nilainya karena dapat

menyelesaikan *request* dari *client* dengan cepat. Pada pengujian *responds time* kali ini *server* nginx lebih unggul dibandingkan dengan *instance server* apache.



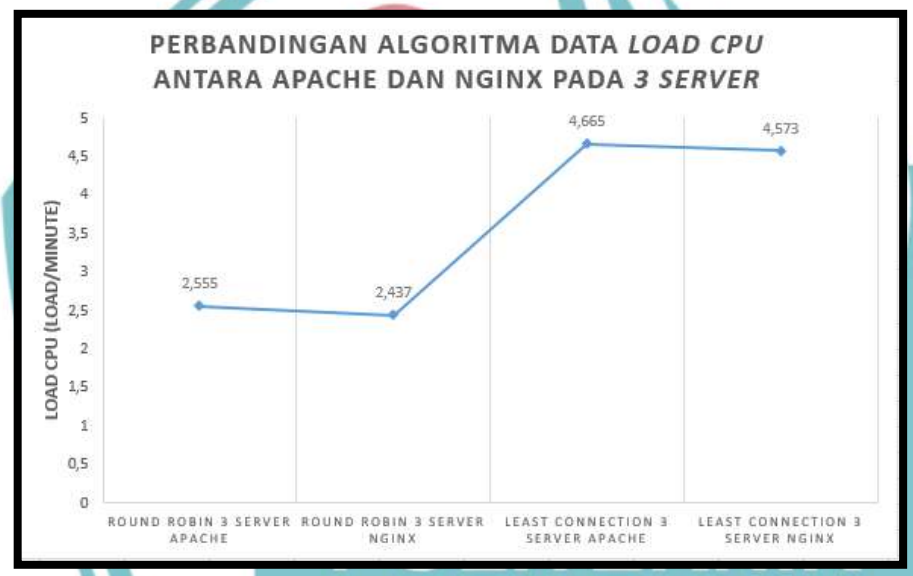
Gambar 4.70 Grafik perbandingan data *responds time* antara apache dan nginx pada 3 *server*

Grafik berikutnya menunjukan perbandingan *transaction rate* antara *instance server* apache dan nginx pada pengujian 3 *server* dengan menggunakan algoritma *round robin* dan *least connection*. Grafik pada *instance server* apache menunjukan penurunan dengan menggunakan algoritma *round robin* memiliki nilai 410.079 trans/sec dan pada algoritma *least connection* memiliki nilai 399.269 trans/sec. Sedangkan grafik pada *server* nginx dengan menggunakan algoritma *round robin* memiliki nilai 451.614 trans/sec dan pada algoritma *least connection* memiliki nilai 436.217 trans/sec. Performa *transaction rate* dikatakan bagus jika memiliki nilai yang cenderung naik sehingga dapat menjalankan *server* dengan optimal. Pada pengujian *transaction rate server* nginx lebih unggul dari pada *instance server* apache.



Gambar 4.71 Grafik perbandingan data *transaction rate* antara apache dan nginx pada 3 *server*

Grafik berikutnya menunjukan perbandingan *load cpu* antara *server* apache dan nginx pada pengujian 3 *server* dengan menggunakan algoritma *round robin* dan *least connection*. Grafik pada *server* apache menunjukan penurunan dengan menggunakan algoritma *round robin* memiliki nilai 2.555 *load/minute* dan pada algoritma *least connection* memiliki nilai 4.665 *load/minute*. Sedangkan grafik pada *server* nginx dengan menggunakan algoritma *round robin* memiliki nilai 2.437 *load/minute* dan pada algoritma *least connection* memiliki nilai 4.573 *load/minute*. Performa *load cpu* dikatakan bagus jika memiliki nilai yang cenderung menurun sehingga *load cp*u *server* apache lebih unggul dari pada *server* nginx.



Gambar 4.72 Grafik perbandingan data *load cpu* antara apache dan nginx pada 3 *server*

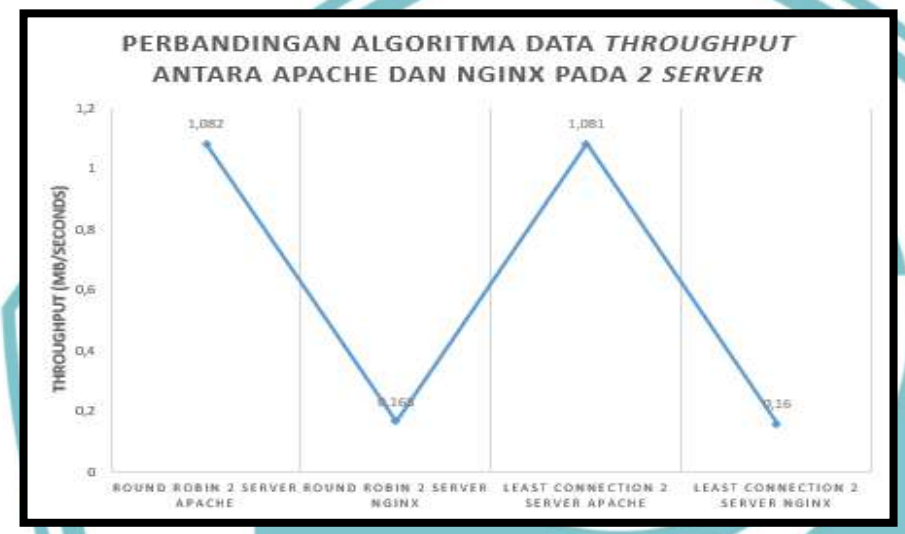
4.5.4 Analisis Algoritma *Load Balancing*

Data yang diambil dari pengujian menggunakan algoritma *round robin* dan algoritma *least connection* pada skenario *load balancing* 2 *server* dan *load balancing* 3 *server* dianalisis untuk menentukan algoritma mana yang terbaik untuk merima *resource* yang adil dan merata.

4.5.4.1 Analisis Perbandingan 2 *Server Load Balancing*

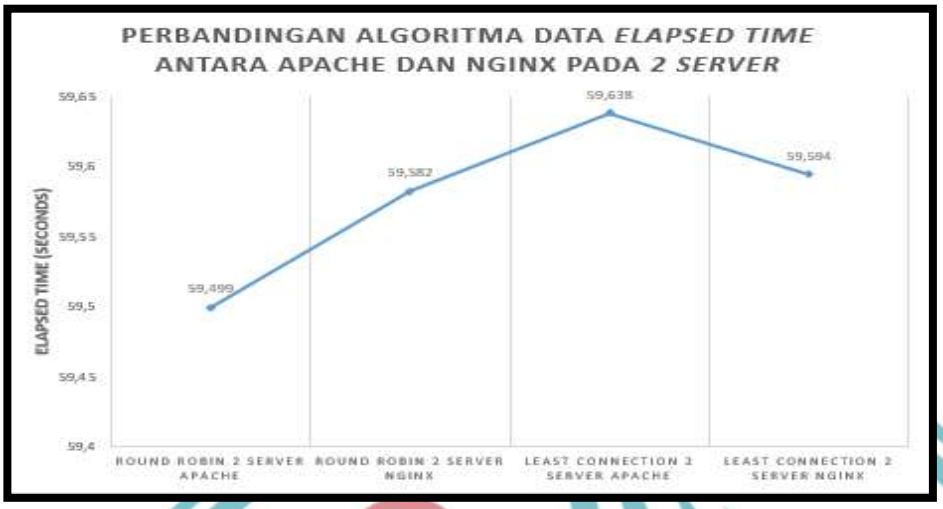
Grafik dibawah ini menunjukan perbandingan *throughput* antara *instance server* apache dan nginx pada pengujian 2 *server* dengan menggunakan algoritma *round robin* dan *least connection*. *Throughput* dinyatakan bagus performanya jika nilai *throughput* cenderung naik seiring dengan bertambahnya jumlah *request* yang diberikan. Grafik dibawah ini menunjukan bahwa *throughput* pada *instance server* apache dengan menggunakan algoritma *round robin* menunjukan nilai yang lebih

tinggi 1.082 MB/seconds dari pada algoritma *least connection* yang memiliki nilai yang lebih rendah sedikit yaitu 1.081 MB/seconds. Begitupun dengan *server* nginx dengan algoritma *round robin* menunjukan nilai 0.168 MB/seconds sedangkan algoritma *least connection* menunjukan nilai 0.160 MB/seconds. Berdasarkan *throughput* yang memiliki nilai yang tinggi memiliki performa yang bagus dapat disimpulkan bahwa algoritma *round robin* lebih unggul dari algoritma *least connection* dalam pengujian *throughput*.



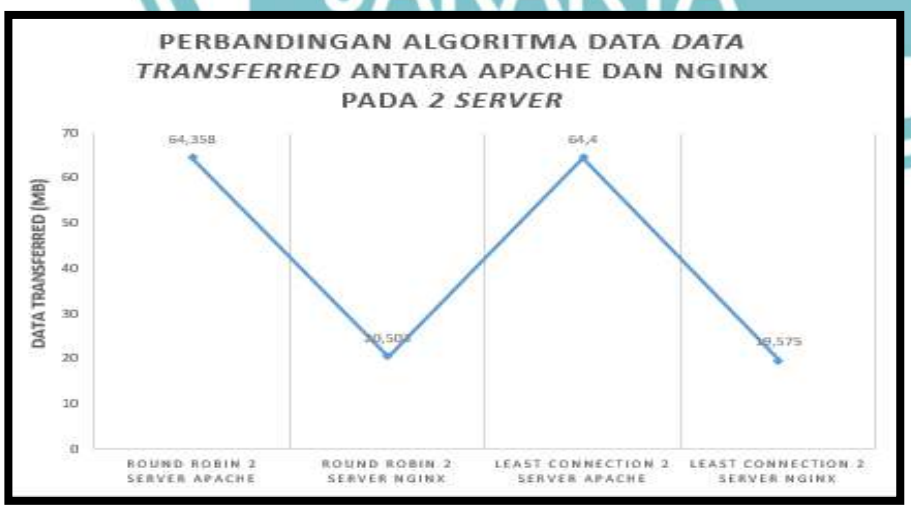
Gambar 4.73 Grafik perbandingan *throughput* antara apache dan nginx pada 2 *server*

Grafik berikutnya menunjukan perbandingan *elapsed time* antara *instance server* apache dan nginx pada pengujian 2 *server* dengan menggunakan algoritma *round robin* dan *least connection*. *Elapsed time* merupakan durasi keseluruhan tes yang dijalankan oleh siege dari *client*, performanya dikatakan bagus jika nilainya semakin rendah karena mampu menyelesaikan request dengan cepat. Grafik dibawah ini menunjukan bahwa *elapsed time* pada *instance server* apache dengan menggunakan algoritma *round robin* menunjukan nilai yang rendah 59.499 seconds dari pada algoritma *least connection* yang memiliki nilai yang lebih tinggi yaitu 59.638 seconds. Begitupun dengan *server* nginx dengan algoritma *round robin* menunjukan nilai 59.582 seconds sedangkan algoritma *least connection* menunjukan nilai 59.594 seconds. Rata-rata nilai dari kedua *instance server* apache dan nginx untuk algoritma *round robin* mendapatkan 59.541 seconds sedangkan untuk algoritma *least connection* mendapatkan 59.616 seconds. Pengujian *elapsed time* menunjukan bahwa algoritma *round robin* lebih unggul dibandingkan dengan *least connection*.



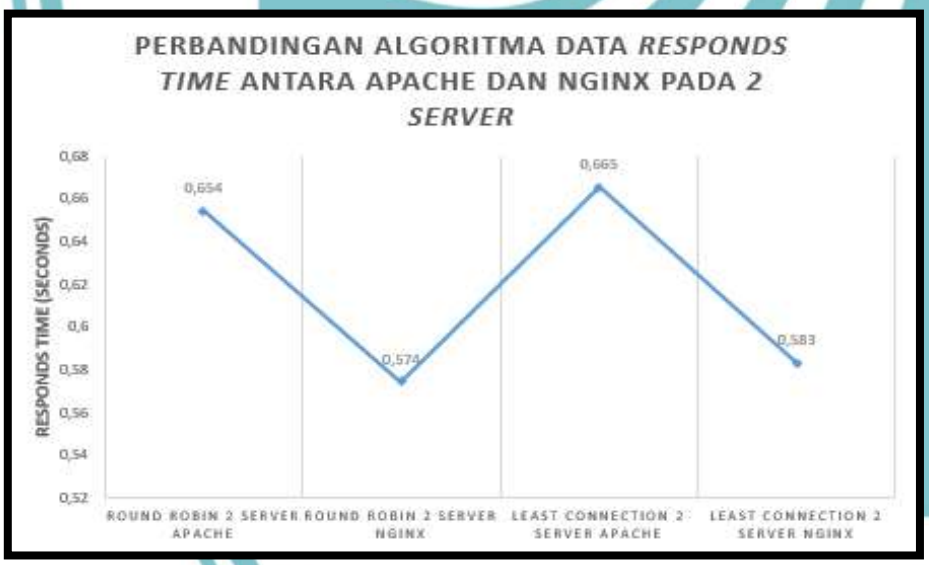
Gambar 4.74 Grafik perbandingan *elapsed time* antara apache dan nginx pada 2 *server*

Grafik berikutnya menunjukan perbandingan *data transferred* antara *instance server* apache dan nginx pada pengujian 2 *server* dengan menggunakan algoritma *round robin* dan *least connection*. Performansi *data transferred* dikatakan bagus jika nilainya semakin tinggi artinya *server* menangani *request* yang datang lebih cepat sehingga data yang ditransfer besarnya bisa lebih besar karena *server* bisa memproses datanya dengan lebih cepat. Kedua grafik *instance server* apache dan nginx dengan menggunakan algoritma *round robin* menunjukan nilai 64.358 MB dan 20.503 MB. Sedangkan untuk algoritma *least connection* menunjukan nilai 64.40 MB dan 19.575 MB. Rata - rata nilai dari algoritma *round robin* yaitu 42.431 MB dan rata-rata nilai dari algoritma *least connection* yaitu 41.988 MB. Berdasarkan pengujian *data transferred* menunjukan bahwa algoritma *round robin* lebih unggul dari pada algoritma *least connection*.



Gambar 4.75 Grafik perbandingan *data transferred* antara apache dan nginx pada 2 *server*

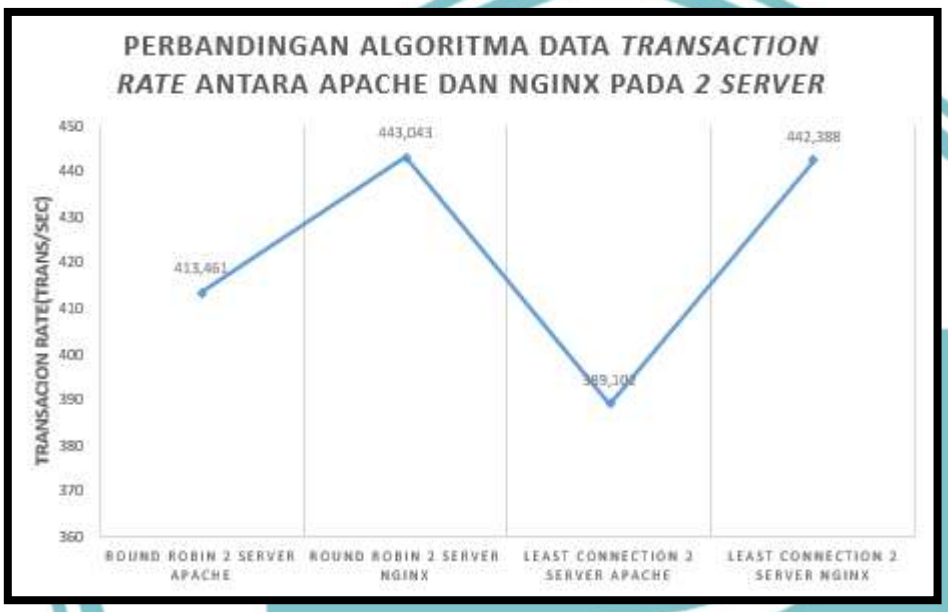
Grafik berikutnya menunjukan perbandingan *responds time* antara *instance server* apache dan nginx pada pengujian 2 *server* dengan menggunakan algoritma *round robin* dan *least connection*. Performa *responds time* dikatakan bagus jika semakin kecil nilainya karena dapat menyelesaikan *request* dari *client* dengan cepat. Berdasarkan pengujian *instance server* apache dengan menggunakan algortima *round robin* mendapatkan nilai 0.654 seconds dan pengujian *server* nginx dengan menggunakan algoritma *round robin* mendapatkan nilai 0.574 seconds. Sedangkan untuk pengujian *instance server* apache dengan menggunakan algoritma *least connection* mendapatkan nilai 0.665 seconds dan untuk *server* nginx dengan menggunakan algoritma *least connection* mendapatkan nilai 0.583 seconds. Berdasarkan rata-rata dari kedua *server* antara apache dan nginx pada algoritma *round robin* 0.614 seconds dan pada algoritma *least connection* 0.624 seconds sehingga dapat dikatakan bahwa algoritma *round robin* lebih unggul dari pada *least connection*.



Gambar 4.76 Grafik perbandingan *responds time* antara apache dan nginx pada 2 *server*

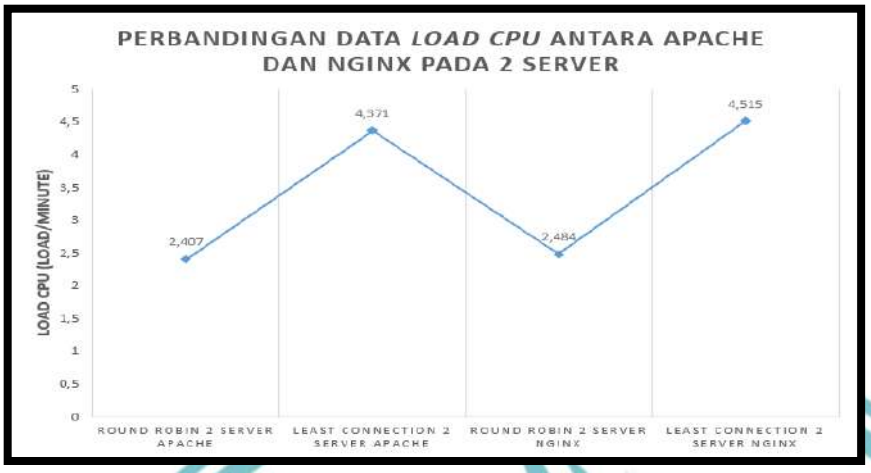
Grafik berikutnya menunjukan perbandingan *transaction rate* antara *instance server* apache dan nginx pada pengujian 2 *server* dengan menggunakan algoritma *round robin* dan *least connection*. Performa *transaction rate* dikatakan bagus jika memiliki nilai yang cenderung naik sehingga dapat menjalankan *server* dengan optimal. Grafik pada *instance server* apache dengan menggunakan algoritma *round robin* mendapatkan nilai 413.461 trans/sec dan pada algoritma *least connection* mendapatkan nilai 389.102 trans/sec. Sedangkan grafik pada server nginx

menunjukan nilai algoritma *round robin* 443.043 trans/sec dan nilai algoritma *least connection* 442.388 trans/sec. berdasarkan rata-rata nilai dari kedua *server* yang menggunakana algoritma *round robin* mendapatkan nilai 428.252 trans/sec sedangkan untuk algoritma *least connection* 415.745 trans/sec dapat dikatakan bahwa algoritma *round robin* lebih unggul dari pada algoritma *least connection*.



Gambar 4.77 Grafik perbandingan *transaction rate* antara apache dan nginx pada 2 *server*

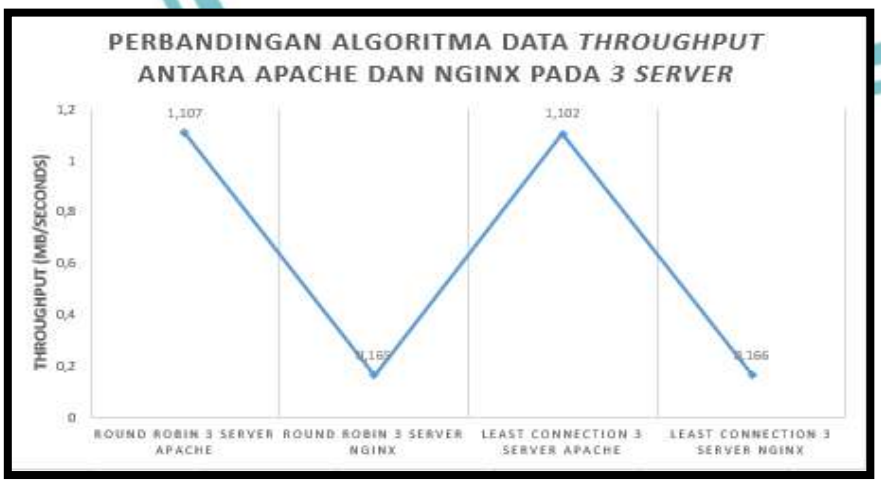
Grafik berikutnya menunjukan perbandingan *load cpu* antara *server* apache dan nginx pada pengujian 2 *server* dengan menggunakan algoritma *round robin* dan *least connection*. Performa *load cpu* dikatakan bagus jika memiliki nilai yang cenderung menurun sehingga dapat menjalankan *server* dengan optimal. Grafik pada *server* apache dengan menggunakan algoritma *round robin* mendapatkan nilai 2.407 *load/minute* dan pada algoritma *least connection* mendapatkan nilai 4.371 *load/minute*. Sedangkan grafik pada server nginx menunjukan nilai algoritma *round robin* 2.484 *load/minute* dan nilai algoritma *least connection* 4.515 *load/minute*. berdasarkan rata-rata nilai dari kedua *server* yang menggunakana algoritma *round robin* mendapatkan nilai 2.445 *load/minute* sedangkan untuk algoritma *least connection* 4.443 *load/minute* dapat dikatakan bahwa algoritma *round robin* lebih unggul dari pada algoritma *least connection*.



Gambar 4.78 Grafik perbandingan *transaction rate* antara apache dan nginx pada 2 *server*

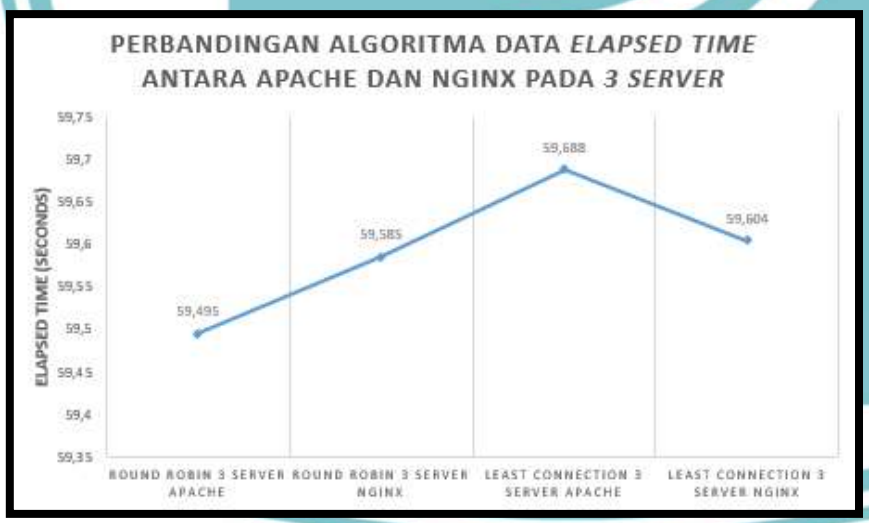
4.5.4.2 Analisis Perbandingan 3 *Server Load Balancing*

Grafik dibawah ini menunjukan perbandingan *throughput* antara *instance server* apache dan nginx pada pengujian 3 server dengan menggunakan algoritma *round robin* dan *least connection*. *Throughput* dinyatakan bagus performanya jika nilai *throughput* cenderung naik seiring dengan bertambahnya jumlah *request* yang diberikan. Grafik dibawah ini menunjukan bahwa *throughput* pada *instance server* apache dengan menggunakan algoritma *round ro*bin menunjukan nilai yang lebih tinggi 1.107 MB/seconds dari pada algoritma *least connection* yang memiliki nilai yang lebih rendah sedikit yaitu 1.102 MB/seconds. Begitupun dengan *server* nginx dengan algoritma *round robin* menunjukan nilai 0.165 MB/seconds sedangkan algoritma *least connection* menunjukan nilai 0.166 MB/seconds. Berdasarkan *throughput* yang memiliki nilai yang tinggi memiliki performa yang bagus dapat disimpulkan bahwa algoritma *round robin* lebih unggul dari algoritma *least connection* dalam pengujian *throughput*.



Gambar 4.79 Grafik perbandingan *throughput* antara apache dan nginx pada 3 *server*

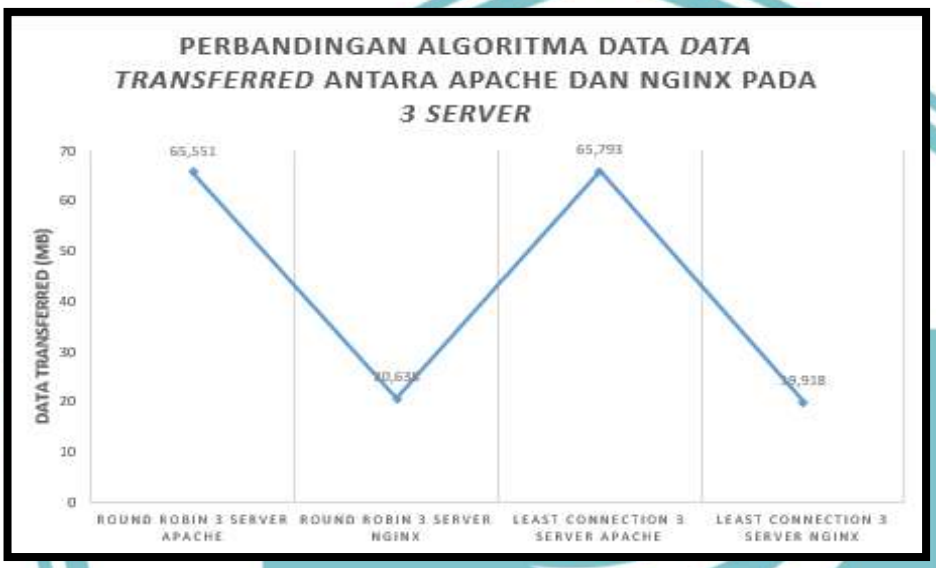
Grafik berikutnya menunjukan perbandingan *elapsed time* antara *instance server* apache dan nginx pada pengujian 2 *server* dengan menggunakan algoritma *round robin* dan *least connection*. *Elapsed time* merupakan durasi keseluruhan tes yang dijalankan oleh siege dari *client*, performanya dikatakan bagus jika nilainya semakin rendah karena mampu menyelesaikan *request* dengan cepat. Grafik dibawah ini menunjukan bahwa *elapsed time* pada *instance ser*ver apache dengan menggunakan algoritma *round robin* menunjukan nilai yang rendah 59.495 seconds dari pada algoritma *least connection* yang memiliki nilai yang lebih tinggi yaitu 59.688 seconds. Begitupun dengan *server* nginx dengan algoritma *round robin* menunjukan nilai 59.585 seconds sedangkan algoritma *least connection* menunjukan nilai 59.604 seconds. Rata-rata nilai dari kedua *instance server* apache dan nginx untuk algoritma *round robin* mendapatkan 59.540 seconds sedangkan untuk algoritma *least connection* mendapatkan 59.646 seconds. Pengujian *elapsed time* menunjukan bahwa algoritma *round robin* lebih unggul dibandingkan dengan *least connection*.



Gambar 4.80 Grafik perbandingan *elapsed time* antara apache dan nginx pada 3 *server*

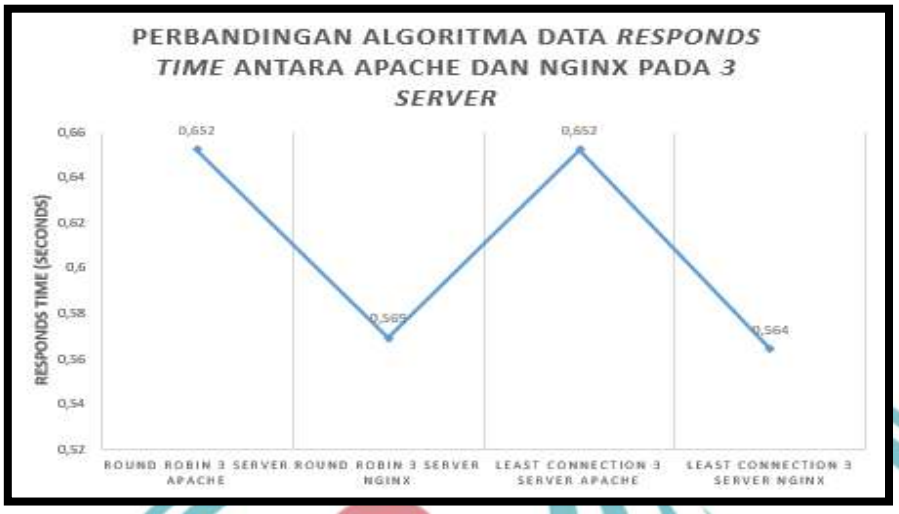
Grafik berikutnya menunjukan perbandingan *data transferred* antara *instance server* apache dan nginx pada pengujian 3 *serve*r dengan menggunakan algoritma *round robin* dan *least connection*. Performansi *data transferred* dikatakan bagus jika nilainya semakin tinggi artinya *server* menangani request yang datang lebih cepat sehingga data yang ditransfer besarnya bisa lebih besar karena *server* bisa memproses datanya dengan lebih cepat. Kedua grafik *instance server* apache dan

nginx dengan menggunakan algoritma *round robin* menunjukan nilai 65.551 MB dan 20.638 MB. Sedangkan untuk algoritma *least connection* menunjukan nilai 65.793 MB dan 19.918 MB. Rata - rata nilai dari algoritma *round robin* yaitu 43.095 MB dan rata-rata nilai dari algoritma *least connectio*n yaitu 42.856 MB. Berdasarkan pengujian *data transferred* menunjukan bahwa algoritma *round robin* lebih unggul dari pada algoritma *least connection*.



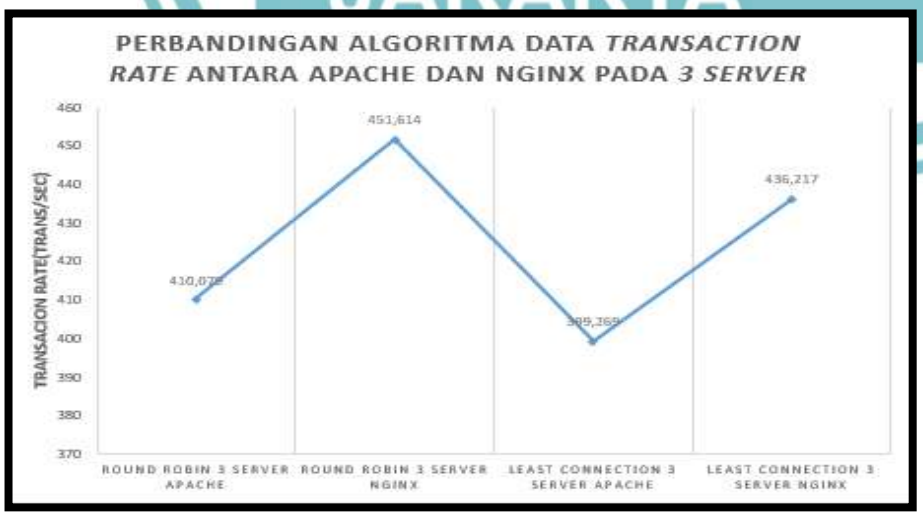
Gambar 4.81 Grafik perbandingan *data transferred* antara apache dan nginx pada 3 *server*

Grafik berikutnya menunjukan perbandingan *responds time* antara *instance server* apache dan nginx pada pengujian 3 *server* dengan menggunakan algoritma *round robin* dan *least connection*. Performa *responds time* dikatakan bagus jika semakin kecil nilainya karena dapat menyelesaikan *request* dari client dengan cepat. Berdasarkan pengujian *instance server* apache dengan menggunakan algortima *round robin* mendapatkan nilai 0.652 seconds dan pengujian *server* nginx dengan menggunakan algoritma *round robin* mendapatkan nilai 0.569 seconds. Sedangkan untuk pengujian *instance server* apache dengan menggunakan algoritma *least connection* mendapatkan nilai 0.652 seconds dan untuk *server* nginx dengan menggunakan algoritma *least connection* mendapatkan nilai 0.564 seconds. Berdasarkan rata-rata dari kedua *server* antara apache dan nginx pada algoritma *round robin* 0.611 seconds dan pada algoritma *least connection* 0.608 seconds sehingga dapat dikatakan bahwa algoritma *round robin* lebih unggul dari pada *least connection*.



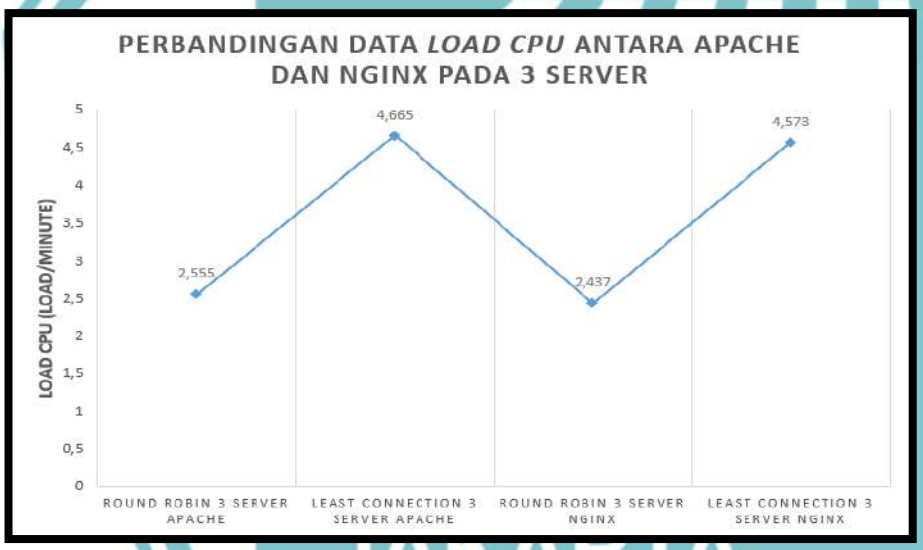
***Gambar 4.82** Grafik perbandingan* responds time *antara apache dan nginx pada 3* server

Grafik berikutnya menunjukan perbandingan *transaction rate* antara *instance server* apache dan nginx pada pengujian 3 *server* dengan menggunakan algoritma *round robin* dan *least connection*. Performa *transaction rate* dikatakan bagus jika memiliki nilai yang cenderung naik sehingga dapat menjalankan *server* dengan optimal. Grafik pada *instance server* apache dengan menggunakan algoritma *round robin* mendapatkan nilai 410.079 trans/sec dan pada algoritma *least connection* mendapatkan nilai 399.269 trans/sec. Sedangkan grafik pada server nginx menunjukan nilai algoritma *round robin* 451.614 trans/sec dan nilai algoritma *least connection* 436.217 trans/sec. Berdasarkan rata-rata nilai dari kedua *server* yang menggunakana algoritma *round robin* mendapatkan nilai 430.847 trans/sec sedangkan untuk algoritma *least connection* 417.743 trans/sec dapat dikatakan bahwa algoritma *round robin* lebih unggul dari pada algoritma *least connection*.



Gambar 4.83 Grafik perbandingan *transaction rate* antara apache dan nginx pada 3 *server*

Grafik berikutnya menunjukan perbandingan *load cpu* antara *server* apache dan nginx pada pengujian 3 *server* dengan menggunakan algoritma *round robin* dan *least connection*. Performa *load cpu* dikatakan bagus jika memiliki nilai yang cenderung menurun sehingga dapat menjalankan *server* dengan optimal. Grafik pada *server* apache dengan menggunakan algoritma *round robin* mendapatkan nilai 2.555 *load/minute* dan pada algoritma *least connection* mendapatkan nilai 4.665 *load/minute*. Sedangkan grafik pada server nginx menunjukan nilai algoritma *round robin* 2.437 *load/minute* dan nilai algoritma *least connection* 4.573*load/minute*. Berdasarkan rata-rata nilai dari kedua *server* yang menggunakana algoritma *round robin* mendapatkan nilai 2.496 *load/minute* sedangkan untuk algoritma *least connection* 4.619 *load/minute* dapat dikatakan bahwa algoritma *round robin* lebih unggul dari pada algoritma *least connection*.



Gambar 4.84 Grafik perbandingan *load cpu* antara apache dan nginx pada 3 *server*

BAB V

PENUTUP

5.1 KESIMPULAN

Berdasarkan hasil dari penelitian yang telah dilakukan oleh penulis, maka dapat ditarik kesimpulan sebagai berikut:

1. Implementasi *load balancer* pada *web server* dapat diwujudkan dengan menerapkan *load balancing* dengan menggunakan LBaaS di lingkungan Openstack serta dengan algoritma *round robin*.
2. Pada pengujian layanan *web server* apache dengan nginx dengan menggunakan *load balancing* menunjukan bahwa apache lebih unggul dibandingkan dengan nginx dari segi *throughput, elapsed time, data transferred* dan *load cpu*.
3. Hasil yang di dapat dari siege dan setelah dilakukan perbandingan pada variabel *throughput, elapsed time, data transferred, responds time, transaction rate* dan *load cpu* menunjukan bahwa hasil dengan menggunakan *load balancer* lebih unggul dibandingkan dengan menggunakan *single server*.

5.2 SARAN

Saran yang dapat diusulkan pada penelitian ini adalah :

1. Menggunakan lebih dari 2 *client* untuk memperbanyak data dan *resourc*e.
2. Menggunakan lebih dari 2 dan 3 *virtual machines* sebagai perbandingan jumlah *web server*.
3. Menggunakan versi Openstack paling baru untuk penelitian selanjutnya.
4. Melakukan penelitian lebih dari 26 kali percobaan pada setiap variabel dan skenario supaya mendapatkan data yang lebih presisi.
5. Membandingkan dengan *load balancing* lain.

DAFTAR PUSTAKA

Assegaf, Setiawan. 2019. "ANALISIS DAN PERANCANGAN JARINGAN VIRTUAL PADA SMK NEGERI 2 KOTA JAMBI."

Aziz, Abdul. 2015. "Analisis Web Server untuk Pengembangan Hosting Server Institusi : Pembanding Kinerja Server Apache dengan Nginx."

Haritsah, Jafar. 2019. "ANALISIS PROSES SISTEM UNTUK IMPLEMENTASI INTERPLANETARY FILE SYSTEM (IPFS) PADA SMART CONTRACT ETHEREUM." 3.

Hidayah, Ivan. 2019. *Implementasi High Availability Web Server Menggunakan Load Blancing As A Service Pada Openstack Cloud.* Bandung: Universitas Telkom.

Irsyad, Hafiz. 2017. "Analisis Kinerja Web Server Menggunakan Algoritma Round Robin dan Least Connection."

Kurniawan, Erick. 2015. "Penerapan Teknologi Cloud Computing Di Universitas Studi Kasus : Fakultas Teknologi UKDW."

Openstack. 2020. "Ikhtisar Struktur Openstack." *https://docs.openstack.org/id/install-guide/overview.html.*

Openstack. 2017. "Load Balancer as a Service (LBaaS)."

Openstack. 2019. "OpenStack Train Release Extends Security and Data Protection, Adds New AI and ML Support."

Openstack. 2020. "Testing Changes dengan DevStack." *https://docs.openstack.org/contributors/id/code-and-documentation/devstack.html.*

Rahmatulloh, F. 2017. "Implementasi Load Balancing Web Server menggunakan HaProxy dan Sinkronisasi File pada Sistem Informasi Akademik Universitas Siliwangi."

Ramadhani, Dian. 2019. "Analisis Perbandingan Algoritma Round Robin dengan Least Connection terhadap Load Balancing pada Load Balancer HAProxy."

Wirasasmita, Rasyid Hardi. 2015. "EFEKTIVITAS PENERAPAN SISTEM OPERASI BERBASIS LINUX UBUNTU HAMZANWADI V.14 UNTUK MENINGKATKAN HASIL BELAJAR MAHASISWA."

LAMPIRAN

DAFTAR RIWAYAT HIDUP PENULIS

Lahir di Kuningan, 20 Juli 1998. Lulus dari SDN 2 Dukuhmaja pada tahun 2010, SMPN 1 Luragung pada tahun 2013, SMAN 1 Luragung pada tahun 2016 dan Diploma II program studi *Network Administrator Professional* di CCIT-FTUI pada tahun 2018. Saat ini sedang menempuh Pendidikan Diploma IV Program Studi Teknik Informatika Jurusan Teknik Informatika dan Komputer di Politeknik Negeri Jakarta.