Analisa Performansi Web Load Balancing via LVS-DR dengan Algoritma Penjadwalan Never Queue (NQ)

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
RAGA YUSTIA
NIM: 115060807113044

TEKNIK INFORMATIKA
PROGRAM TEKNOLOGI INFORMASI DAN ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2015

`

PENGESAHAN

ANALISA PERFORMANSI WEB LOAD BALANCING VIA LVS-DR DENGAN ALGORITMA PENJADWALAN NEVER QUEUE (NQ)

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
RAGA YUSTIA
NIM: 115060807113044

Skripsi ini telah diuji dan dinyatakan lulus pada
26 November 2015
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Sabriansyah Rizqika Akbar, ST., M.Eng	Adhitya Bhawiyuga, S.Kom., M.S.
NIP: 19820809 201212 1 004	NIK: 201405 890720 1 1 001

Mengetahui
Ketua Program Studi Informatika/Ilmu Komputer

Drs. Marji., M.T.
NIP: 19670801 199203 1 001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 26 November 2015

RAGA YUSTIA
NIM: 115060807113044

KATA PENGANTAR

Puji syukur penulis panjatkan kehadirat Allah SWT atas limpahan rahmat dan karunia-Nya sehingga penulis dapat menyelesaikan proposal skripsi yang berjudul "Analisa Performansi *Web Load Balancing* via *LVS-DR* dengan Algoritma Penjadwalan *Never Queue (NQ)"* dengan baik.

Penulisan skripsi ini diajukan untuk syarat memenuhi gelar sarjana komputer, dan merupakan salah satu syarat yang harus ditempuh oleh setiap mahasiswa Teknik Informatika Universitas Brawijaya. Penyusunan proposal skripsi ini dapat terlaksana dengan baik karena adanya bantuan secara langsung maupun tidak langsung dari pihak tertentu, diantaranya :

1. Bapak **Sabriansyah R. Akbar, ST,. M.Eng.** sebagai dosen pembimbing I yang telah membantu berbagi ilmu dan memberi saran untuk penyelesaian penelitian ini.
2. Bapak **Adhitya Bhawiyuga, S.Kom., M.S.** selaku dosen pembimbing II yang juga memberikan saran serta masukkan untuk pengerjaan penelitian ini.
3. **Bapak** dan **Ibu** saya yang selalu memberi motivasi serta memfasilitasi studi saya selama ini.
4. **Ratna Puspaningtyas** yang terus memberikan motivasi dan dorongan untuk lebih semangat mengerjakan penelitian ini.
5. Teman – teman yang selalu mengajak penulis untuk refreshing dengan main game bersama saat penulis suntuk dalam pengerjaan penelitian ini.

Penulis menyadari bahwa skripsi ini masih banyak kekurangan dan jauh dari sempurna, karena keterbatasan materi serta pengetahuan. Dan akhir kata semoga hasil penelitian skripsi ini dapat memberikan manfaat dan berguna bagi pembaca.

Malang, November 2015

Penulis

ABSTRAK

Perkembangan *website* pada saat ini menuntut peningkatan kinerja layanan *web*. *Website* berarsitektur *single web server* sudah tidak lagi efisien, penerapan teknologi *load balancing* yang diharapkan dapat menangani beban *request* yang besar dengan cara mendistribusikan beban kerja dari beberapa komputer atau *cluster.* Pada *linux virtual server* dapat diterapkan beberapa konsep *routing* dan algoritma penjadwalan untuk perancangan sistem *web load balancing*. Salah satunya menggunakan konsep *direct routing* dengan menggunakan algoritma penjadwalan *never queue*.

Algoritma *Never Queue* (NQ) akan meneruskan koneksi yang datang dari *client* ke *server* yang sedang berstatus *idle* (menganggur). Namun jika tidak ada *server* yang sedang *idle*, maka algoritma *never queue* akan menghitung harapan *delay* koneksi terpendek dengan menggunakan algoritma *Shortest Expected Delay* (*SED*). Konsep *direct routing* memungkinkan *client* bisa langsung berkomunikasi dengan *real server* setelah terjadi pembagian *request* pada *director*/*load balancer*, sehingga lebih meringankan beban dari *load balancer*.

Algoritma penjadwalan *never queue* mampu bekerja dengan baik pada hasil pengujian dengan perulangan 30 kali untuk setiap skenario pengujian dengan interval 50 *request*/, 80 *request*/detik, 120 *request*/detik, dan 170 *request*/detik. Pada pengujian dengan perulangan 10 kali, *never queue* bekerja lebih baik pada permintaan koneksi sejumlah 170 *request*/detik. Untuk pengujian dengan perulangan 60 kali, algoritma *never queue* mampu bekerja dengan baik pada jumlah permintaan koneksi 50 *request*/detik dan 120 *request*/detik. Hasil pengujian pada implementasi sistem *web load balancing, web load balancing* via *LVS-DR* dengan algoritma penjadwalan *Never Queue* (*NQ*) mampu memproses *request* secara optimal pada pengujian dengan 50 *request*/detik dengan *throughput, response time,* dan *request rate* yang terus meningkat pada setiap perulangannya mencapai 237,02 *KB/s* untuk *throughput* yang dihasilkan, rata-rata waktu respon sejumlah 1126 *ms* dan *request rate* yang mampu di proses sebesar 43,5 *request*/detik pada pengujian dengan perulangan 60 kali.

Kata Kunci: *Load balancing*, *Linux Virtual Server*, *Direct Routing*, *never queue*

ABSTRACT

The development of the website is now demanding increased performance web services. Websites with a single web server architecture is no longer efficient, application load balancing technology that is expected to handle a large load of requests by distributing the workload from multiple computers or clusters. In the virtual linux server can apply some of the concepts of routing and scheduling algorithms for load balancing web system design. One of them uses the concept of direct routing using never queue scheduling algorithm.

Never Queue (NQ) schedulling algorithm will forward incoming connections from the client to the server being idle status. However, if no server is idle, then the algorithm will calculate the shortest expectations of connection delay using Shortest Expected Delay (SED) algorithm. The concept of direct routing allows the client can directly communicate with the real server after a request to the division director / load balancer, so that further lighten the load of the load balancer.

Never queue scheduling algorithm is able to work well on the test results with a looping 30 times for each scenario testing at intervals of 50 requests/sec, 80 requests/sec, 120 requests/sec, and 170 requests/sec. On testing with a looping 10 times, never queue work better on a number of connection requests 170 requests/sec. For testing with a looping 60 times, never queue algorithm works better on a number of connection requests 50 requests/sec and 120 requests/sec. The test results on the implementation of the system is web load balancing, web load balancing via LVS-DR with the Never Queue (NQ) scheduling algorithm is able to process the request optimally on testing 50 requests/sec with the amount of throughput, response time, and request rate continues to increase in each recurrence reached 237.02 KB/s of throughput generated, the average number of response time 1126 ms and request rate capable in the process of 43.5 requests/sec on testing with a looping 60 times.

Keyword: Load balancing, Linux Virtual Server, Direct Routing, never queue

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB 1 PENDAHULUAN

1.1 Latar belakang

Website merupakan media penyebaran informasi yang bersifat global. Saat ini *website* semakin berkembang pesat serta menjadi suatu infrastruktur yang penting pada suatu instansi pemerintahan, organisasi maupun perusahaan untuk menyediakan layanan ataupun informasi secara publik.

Website berarsitektur *single web server* yang banyak diterapkan ternyata memiliki banyak kekurangan, disaat *website* mengalami peningkatan akses *request* dari pengguna hingga terjadi *overload* untuk merespon, maka *server* bisa mengalami *down* total sehingga tidak ada lagi yang akan melayani seluruh *request* dari pengguna. Menurut pendapat G. Teodoro dan kawan-kawan, jika dilakukan peningkatan terhadap kapasitas pemrosesan pada *server* secara invidu tersebut dianggap tidaklah efektif (Teodoro, Tavares, Coutinho, Meira, & Guedes, 2003).

Maka dari itu perlu diterapkannya teknologi *load balancing* yang diharapkan dapat menangani beban *request* yang besar dengan cara mendistribusikan beban kerja dari beberapa komputer atau *cluster*. Salah satu jenis sistem *network load balancing* yaitu Linux Virtual Server (LVS) (Haris, 2011). *Real server* yang bertindak sebagai *server* penyedia konten *website* dan salah satu *server* sebagai *load balancer* untuk mendistribusik an beban *request* ke beberapa *real server*.

Terdapat beberapa algoritma penjadwalan untuk *load balancing*, seperti *round robin*, *least connection*, *weighted round robin* dan *weighted least connection*. Berdasarkan penelitian Yogi Kurniawan (Kurniawan, 2013), dalam penelitiannya melakukan perbandingan performa dari empat algortima penjadwalan yang ada pada *LVS* dengan konsep *Network Address Translation* (*NAT*), keempat algoritma tersebut adalah *round robin*, *least connection*, *weighted round robin* dan *weighted least connection*. Namun metode pendistribusian beban *request* dari algoritma penjadwalan *round robin* dan *least connection* hanya mengacu pada tetapan beban setiap *server* dan tidak memperhatikan dari kondisi waktu respon dan sumber daya setiap *real server* sehingga ketika salah satu *server* dalam keadaan load yang tinggi maka *load balancer* akan tetap mengarahkan koneksi ke *server* tersebut sehingga rawan terjadi down terhadap *server* saat dalam keadaan trafik tinggi. Menurut Wensong Zhang (Zhang, 2010), dari hasil penelitiannya yang membahas konsep kerja *Linux Virtual Server* via *NAT*, *Tunneling*, dan *Direct Routing* menyimpulkan bahwa skalabilitas dari *LVS* via *NAT* sangatlah tidak baik dikarenakan *load balancer* rawan mengalami *bottleneck* atau macetnya proses aliran data jika jumlah *real server* di dalamnya semakin banyak karena konsep *LVS NAT* mengharuskan *load balancer* menuliskan ulang paket *request* yang masuk dan paket balasan untuk dikembalikan ke *client*. Sehingga lebih disarankan menggunakan konsep *Tunneling* atau *Direct Routing* jika pada *load balancer* mulai terjadi *bottleneck*.

Algoritma *Never Queue* dapat menjadi satu pilihan algoritma yang diharapkan mampu bekerja secara optimal yang akan diterapkan pada sistem *load balancing* dengan *Linux Virtual Server* via *Direct Routing* karena alur kerja algoritma *Never Queue* juga memanfaatkan algoritma penjadwalan lainnnya yaitu algoritma *Shortest Expected Delay* (*SED*). Cara kerja algoritma *Never Queue* (*NQ*) adalah pertama-tama *Load Balancer* akan mengecek *server* mana yang sedang *idle* (menganggur), jika ada yang menganggur maka *request* yang datang akan dikirimkan ke *server* tersebut. Namun jika tidak ada *server* yang sedang *idle*, maka algoritma *Shortest Expected Delay* (SED) menangani *request* dengan mengirimkan ke *server* yang memiliki estimasi *delay* koneksi terpendek (GNU Free Doc License, 2006). Konsep penerapan *Linux Virtual Server* via *Direct Routing* juga diharapkan dapat mengoptimalkan kinerja dari load balancing ketika melayani requet dalam skala yang besar.

Pada penelitian kali ini akan melakukan implementasi *web Load Balancing* dengan algoritma *never queue* dengan konsep *direct routing* untuk dilakukan analisa terhadap proses pembagian beban kerja dari algoritma *never queue*, *throughput*, *response time* dan *request rate* yang dihasilkan dari hasil pengujian.

1.2 Rumusan masalah

Berdasarkan latar belakang yang telah dijabarkan, dapat dirumuskan permasalahan yang akan diselesaikan dalam penelitian ini adalah :

1. Bagaimana langkah-langkah implementasi sistem *web load balancing* via *LVS-DR* dengan Algoritma Penjadwalan *Never Queue* (*NQ*)?
2. Bagaimana cara kerja algoritma penjadwalan *Never Queue* (*NQ*) dalam membagi beban kerja pada sistem *web load balancing*?
3. Bagaimana melakukan analisa terhadap proses pengujian pembagian beban kerja dengan melakukan pengamatan terhadap *throughput*, *response time*, dan *request rate* yang dihasilkan?

1.3 Tujuan

Tujuan dibuatnya skripsi ini adalah :

1. Menjelaskan tentang langkah-langkah implementasi sistem *web load balancing* via *LVS-DR* dengan algoritma penjadwalan *Never Queue* (*NQ*).
2. Untuk mengetahui konsep cara kerja dari algoritma penjadwalan *Never Queue* (*NQ*) balam menentukan pembagian beban kerja dalam sistem *web load balancing*.
3. Mejelaskan proses analisa dari hasil pengujian terhadap sistem *web load balancing* yang diimplementasikan dengan melakukan proses pengamatan terhadap pembagian beban kerja, *throughput*, *response time*, dan *request rate*.

1.4 Manfaat

Manfaat dari penulisan skripsi ini adalah sebagai berikut :

1. Umum

Memberikan referensi konsep dan cara kerja routing dari *direct routing* dan algoritma penjadwalan *Never Queue* (NQ) untuk diterapkan pada *server web load balancing.*

2. Khusus
    a. Mahasiswa
        i. Sebagai salah satu sarana referensi untuk pengerjaan tugas mata kuliah dengan topik seputar *load balancing*.
        ii. Untuk acuan bagi mahasiswa lain yang akan melaksanakan tugas akhir dengan topik yang sama.
    b. Penulis
        i. Sebagai salah satu sarana untuk menerapkan hasil studi selama empat tahun di jenjang S1 Informatika/Ilmu Komputer Universitas Brawijaya.
        ii. Dapat menerapkan pengetahuan yang diperoleh selama kuliah serta menambah pengetahuan dan pengalaman baru dalam penerapan *load balancing* pada sebuah *web server*.

1.5 Batasan masalah

Batasan masalah pada skripsi ini adalah sebagai berikut :

1. Diimplementasikan pada pengalamatan jaringan *IPv4*.
2. Perancangan sistem dan pengujian algoritma dilakukan pada protokol *HTTP* (*port* 80).
3. Tidak membahas segi keamanan jaringan.
4. Tidak berfokus pada pembahasan replikasi *database* dan *session database* pada sistem *web load balancing.*
5. Pengujian dilakukan berdasarkan pada perancangan pengujian.

1.6 Sistematika pembahasan

Penyusunan pada skripsi ini menggunakan kerangka penulisan sebagai berikut :

BAB 1 PENDAHULUAN

Berisi latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, serta sistematika penulisan dari topik dan judul skripsi yang diajukan.

BAB 2 LANDASAN KEPUSTAKAAN

Pada bab ini berisi teori-teori dan penelitian yang pernah ada sebelumnya untuk menunjang penyelesaian masalah tentang arsitektur *web load balancing* yang diterapkan.

BAB 3 METODE PENELITIAN DAN PERANCANGAN

Membahas tentang metode yang digunakan dalam penelitian berupa perancangan sistem, proses pembuatan, pengujian dan analisa sistem sampai dengan penarikan kesimpulan dan saran serta berisi tentang analisis perencanaan kebutuhan dan perancangan sistem *load balancing* pada *web server*.

BAB 4 IMPLEMENTASI

Membahas tentang implementasi dari sistem yang akan dibangun dari hasil perancangan.

BAB 5 PENGUJIAN DAN ANALISA

Membahas tentang pengujian sistem yang telah diimplementasikan dan analisa pada hasil data pengujian.

BAB 6 PENUTUP

Memuat kesimpulan serta saran yang diperoleh dari hasil analisa uji coba yang dapat digunakan untuk pengembangan selanjutnya.

BAB 2 LANDASAN KEPUSTAKAAN

2.1 Penelitian Terkait

Penelitian tentang analisa algoritma penjadwalan sebuah *cluster web server* yang sebelumnya dilakukan oleh Yogi Kurniawan pada tahun 2013 yang berjudul "Analisis Kinerja Algoritma *Load Balancer* dan Implementasi pada Layanan *Web*" tentang perbandingan empat algoritma penjadwalan yaitu algoritma *round robin*, *least connection*, *weighted round robin* dan *weighted least connection* yang diterapkan pada sebuah *cluster web server* diperoleh kesimpulan bahwa *load balancer* yang menggunakan algoritma *least connection* memiliki performa yang lebih baik pada 3000 *request* dan 6000 *request*. Tetapi pada 9000 *request* memiliki performa setara dengan *round robin* pada layanan *web* dinamis, sedangkan untuk *weighted least connection* memiliki peforma yang setara dengan *least connection*. (Kurniawan, 2013)

2.2 *Web server cluster*

Cluster pada *web server* adalah sekumpulan *web server* independen yang beroperasi serta bekerja secara erat dan terlihat oleh *client* seolah-olah *server* tersebut adalah satu buah unit server. Sehingga *cluster server* ini mempunyai kemampuan komputasi yang relatif baik.

Kemudian kelebihannya lagi dibanding dengan *server* biasa pemroses dalam hal ini prosesor pada *cluster* dapat terus bertambah sesuai dengan jumlah prosesor yang di-*cluster*-kan sehingga dapat dipastikan bahwa *server* yang sudah di-*cluster*-kan mempunyai kemampuan yang relatif lebih baik dibandingkan dengan *server* biasa. (Dwi, 2012)

Clustering terbagi kedalam beberapa jenis, sebagai berikut :

1. ***High Availability***. *High Availability cluster* atau yang juga sering disebut *Failover Cluster* pada umumnya diimplementasikan untuk tujuan meningkatkan ketersediaan layanan yang disediakan oleh *cluster* tersebut. Elemen *cluster* akan bekerja dengan memiliki *node* redundan, yang kemudian digunakan untuk menyediakan layanan saat salah satu elemen *cluster* mengalami kegagalan. Ukuran yang paling umum dari kategori ini adalah dua *node*, yang merupakan syarat minimum untuk melakukan redudansi. Dimana implementasi *cluster* jenis ini akan mencoba untuk menggunakan redudansi komponen *cluster* untuk menghilangkan kegagalan disuatu tititk (*Single Point of Failure*).
2. ***Load Balancing Cluster***. *Cluster* jenis ini beroperasi dengan mendistribusikan beberapa pekerjaan secara merata melalui beberapa *node* yang bekerja dibelakang (*Back end node*). Proses *load balancing* sebenarnya merupakan proses fleksibel yang dapat diciptakan dengan berbagai cara dan metode. Proses ini tidak dapat dilakukan oleh sebuah perangkat tertentu atau sebuah *software* khusus saja. Cukup banyak cara dan pilihan untuk mendapatkan jaringan yang dilengkapi dengan sistem *load balancing*. Cara kerja dan prosesnya pun berbeda-beda satu dengan yang lainnya. Namun, cara yang paling umum dan banyak digunakan adalah dengan mengandalkan konsep

Virtual server atau *Virtual IP*. Istilah *Virtual server* atau *Virtual IP* sebenarnya merupakan istilah bebas, karena mungkin saja sistem lain menggunakan konsep yang sama namun dengan istilah yang berbeda. Secara umum, konsep dari *Virtual server* atau *Virtual IP* ini adalah sebuah alamat *IP*, sebuah nama, atau bisa juga dikatakan sekelompok alamat *IP* yang bertugas sebagai jembatan penghubung antara pengakses dari luar dengan sekelompok *server* atau perangkat jaringan yang berada dibelakangnya. Tujuan dibuatnya sistem perwakilan tersebut adalah agar ketika nama atau alamat *IP* tersebut diakses dari luar, yang dapat melayani permintaan tersebut tidak terbatas hanya satu perangkat *server* saja. Sekelompok *server* atau perangkat jaringan yang diwakilinya memiliki kemampuan untuk menjawab permintaan-permintaan tersebut. Sebagai hasilnya, permintaan-permintaan tersebut terdistribusi ke beberapa *server* sehingga beban proses kerja *server-server* tersebut tidak terlalu berat. Hal ini membuat servis dan layanan yang diberikan *server* tersebut ke si pengguna dapat berjalan lebih baik dan berkualitas. Sistem *load balancing* yang sederhana memang hanya mampu membuat sebuah perwakilan nama atau alamat *IP* untuk mewakili beberapa *IP* dari *server-server* dibelakangnya, namun perangkat yang memang dikhususkan menangani sistem *load balancing* kompleks dapat melakukan perwakilan hanya terhadap servis-servis yang dibuka oleh *server* dibelakangnya. Dalam sistem *load balancing*, proses pembagian bebannya memiliki teknik dan algoritma tersendiri. Pada perangkat load balancing yang kompleks biasanya disediakan bermacam- macam algoritma pembagian beban ini. Tujuannya adalah untuk menyesuaikan pembagian beban dengan karakteristik dari *server-server* yang ada di belakangnya.

3. ***Grid Computing***. *Grid Computing* biasa disebut *computer cluster*, tapi difokuskan pada *throughput* seperti utilitas perhitungan ketimbang menjalankan pekerjaan–pekerjaan yang sangat erat yang biasanya dilakukan oleh super komputer. *Grid Computing* dioptimalkan untuk beban pekerjaan yang mencakup banyak pekerjaan independen.

2.3 *DBMS* (*Database Management System*)

Basis data (atau *database*) adalah kumpulan informasi yang disimpan di dalam komputer secara sistematik sehingga dapat diperiksa menggunakan suatu program komputer untuk memperoleh informasi dari basis data tersebut. *Database* digunakan untuk menyimpan informasi atau data yang terintegrasi dengan baik di dalam komputer.

Untuk mengelola *database* diperlukan suatu perangkat lunak yang disebut *DBMS* (*Database Management System*). *DBMS* merupakan suatu sistem perangkat lunak yang memungkinkan *user* (pengguna) untuk membuat, memelihara, mengontrol, dan mengakses *database* secara praktis dan efisien. Dengan *DBMS*, *user* akan lebih mudah mengontrol dan memanipulasi data yang ada. (Solichin, 2010)

2.3.1 *MySQL*

MySQL adalah sebuah perangkat lunak sistem manajemen basis data *SQL* (bahasa Inggris: *database management system*) atau *DBMS* yang *multithread*, *multi-user*, dengan sekitar 6 juta instalasi di seluruh dunia. *MySQL* tersedia sebagai perangkat lunak gratis di bawah lisensi *GNU General Public License* (*GPL*), tetapi mereka juga menjual dibawah lisensi komersial untuk kasus-kasus dimana penggunaannya tidak cocok dengan penggunaan *GPL*.

Tidak seperti *PHP* atau *Apache* yang merupakan software yang dikembangkan oleh komunitas umum, dan hak cipta untuk kode sumber dimiliki oleh penulisnya masing-masing, *MySQL* dimiliki dan disponsori oleh sebuah perusahaan komersial Swedia yaitu *MySQL AB*. *MySQL AB* memegang penuh hak cipta hampir atas semua kode sumbernya. Kedua orang Swedia dan satu orang Finlandia yang mendirikan *MySQL AB* adalah: David Axmark, Allan Larsson, dan Michael "Monty" Widenius. (Solichin, 2010)

Fitur-fitur MySQL antara lain :

1. *Relational Database System*

Seperti halnya *software database* lain yang ada di pasaran, *MySQL* termasuk *RDBMS*.

2. Arsitektur *Client-Server*.

MySQL memiliki arsitektur *client-server* dimana *server database MySQL* terinstal di *server*. *Client MySQL* dapat berada di komputer yang sama dengan *server*, dan dapat juga di komputer lain yang berkomunikasi dengan *server* melalui jaringan bahkan internet.

3. Mengenal perintah *SQL* standar

MYSQL (*Structured Query Language*) merupakan suatu bahasa standar yang berlaku di hampir semua *software database*. *MySQL* mendukung *SQL* versi *SQL*:2003.

4. Mendukung *Triggers*.

MySQL mendukung *trigger* pada versi 5.0 namun masih terbatas. Pengembang *MySQL* berjanji akan meningkatkan kemampuan *trigger* pada versi 5.1.

5. Mendukung *replication backup*.

Backup di MySQL sebenarnya ada 2 jenis, yaitu secara otomatis dan manual. Secara otomatis kita dapat menggunakan konsep *replication* , dimana server database kita secara *real-time* di-backup dengan server lain. Jika terdapat perubahan di server utama kita, maka secara otomatis perubahannya akan di-*replikasi* ke server kedua. Selain itu, kita juga dapat melakukan backup otomatis dengan bantuan software tertentu. Biasanya kita dapat membuat *schedule-backup.* Backup akan dijalankan oleh software secara otomatis setiap periode waktu tertentu.

6. *Free* (bebas didownload) Stabil

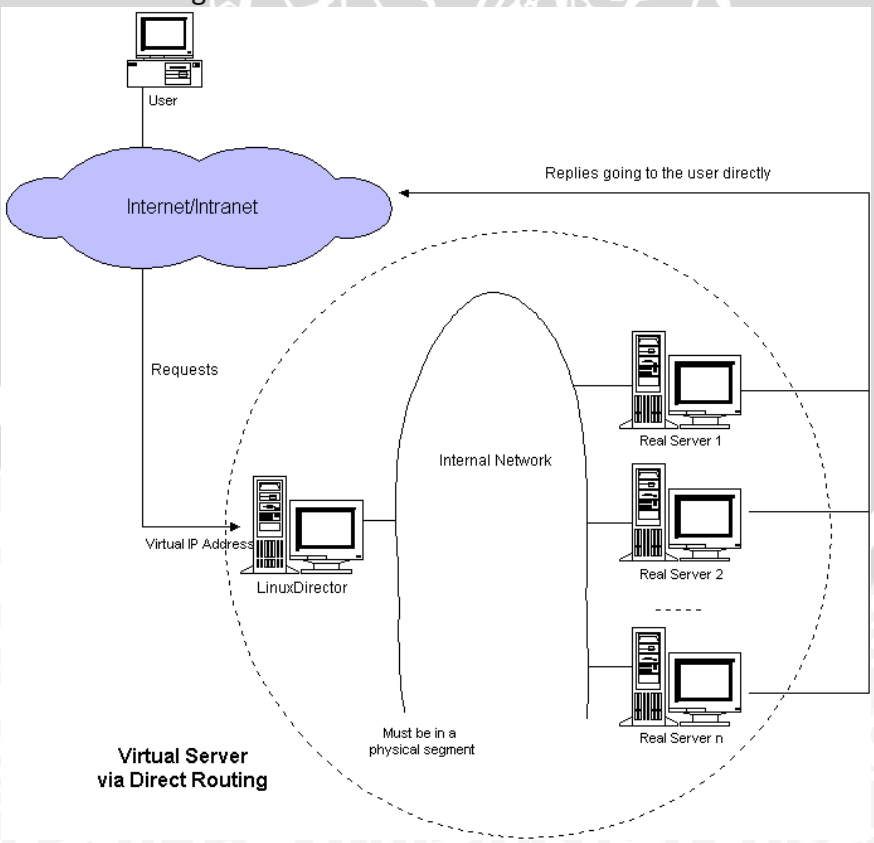
Tangguh Fleksibel dengan berbagai pemrograman *Security* yang baik Dukungan dari banyak komunitas Perkembangan *software* yang cukup cepat.

2.4 *LVS* (*Linux Virtual Server*)

Linux virtual server adalah perangkat lunak yang mengarahkan koneksi jaringan ke beberapa *server* yang memiliki beban kerja masing-masing yang dapat digunakan untuk membangun layanan berskala tinggi dan sangat tersedia. *Linux Virtual Server* mengarahkan koneksi ke *server* yang berbeda sesuai dengan algoritma penjadwalan dan membuat layanan *paralel cluster* untuk muncul sebagai layanan *virtual* pada satu alamat *IP*. (Wensong, 2011)

2.4.1 *LVS* via *Direct Routing*

Konsep *direct routing* pada *LVS* adalah *load balancer* dan *real server* berada dalam satu jaringan. *Load balancer* mempunyai *virtual ip address*, yang digunakan untuk menerima paket, dan langsung mengarahkan paket ke *server* yang terpilih. Semua *real server* mempunyai *interface non-arp alias* yang dikonfigurasi dengan *virtual ip address* atau mengalihkan paket yang ditujukan untuk *virtual ip address* untuk soket lokal, oleh karena itu *real server* dapat memproses paket secara lokal (Zhang, 2011). Arsitektur dari *LVS* via *direct routing* adalah sesuai gambar 2.1 :



Gambar 2.1 Arsitektur *LVS* via *direct routing*

Sumber: Wensong Zhang (2011)

Ketika *client* mengakses layanan yang diberikan oleh *server cluster*, paket yang ditujukan untuk *virtual IP* address tiba. *Load balancer* memeriksa alamat dan *port* paket tujuan. Jika mereka ditemukan untuk *virtual service*, maka sebuah *real server* dipilih dari *cluster sesuai* dengan algoritma penjadwalan. Kemudian *load balancer* secara langsung mem-*forward* ke *server* yang terpilih. Ketika *server* menerima paket lalu *server* akan memproses *request* dan akhirnya mengembalikan hasil secara langsung kepada *client*.

2.5 Algoritma Penjadwalan

Beberapa jenis algoritma penjadwalan yang dapat diterapkan pada sistem *Linux Virtual Server* pada proses distribusi *request* kepada *real server*, antara lain yaitu:

1. *Round Robin* (*rr*), yaitu algoritma penjadwalan yang memperlakukan semua *real server* sama menurut jumlah koneksi atau waktu respon.
2. *Weighted Round Robin* (*wrr*), penjadwalan ini memperlakukan *real server* dengan kapasitas proses yang berbeda. Masing-masing *real server* dapat diberi bobot bilangan *integer* yang menunjukkan kapasitas proses, dimana bobot awal adalah 1.
3. *Least Connection* (*lc*), merupakan algoritma penjadwalan yang mengarahkan koneksi jaringan pada *server* aktif dengan jumlah koneksi yang paling sedikit. Penjadwalan ini termasuk salah satu algoritma penjadwalan dinamik, karena memerlukan perhitungan koneksi aktif untuk masing-masing *real server* secara dinamik. Metode penjadwalan ini baik digunakan untuk melancarkan pendistribusian ketika *request* yang datang banyak.
4. *Weighted Least Connection* (*wlc*), merupakan sekumpulan penjadwalan *least connection* dimana dapat ditentukan bobot kinerja pada masing-masing *real server*. Server dengan nilai bobot yang lebih tinggi akan menerima persentase yang lebih besar dari koneksi-koneksi aktif pada satu waktu. Bobot pada masing-masing *real server* dapat ditentukan dan koneksi jaringan dijadwalkan pada masing-masing *real server* dengan persentase jumlah koneksi aktif untuk masing-masing server sesuai dengan perbandingan bobotnya (bobot awal adalah 1).
5. *Locality Based Least Connection* (*lblc*), metode penjadwalan yang akan mendistribusikan lebih banyak *request* kepada *real server* yang memiliki koneksi kurang aktif. Algoritma ini akan meneruskan semua *request* kepada *real server* yang memiliki koneksi kurang aktif tersebut sampai kapasitasnya terpenuhi.
6. *Destination Hashing* (*dh*), merupakan algortima penjadwalan statik yang dapat meneruskan *request* dari *client* kepada satu *real server* tertentu sesuai dengan layanan yang diminta. Terdapat suatu tabel *hash* berisi alamat tujuan dari masing-masing *real server* beserta layanan yang tersedia pada setiap *real server*.

7. *Source Hashing* (*sh*), hampir sama dengan metode *destination hashing* tetapi pada metode ini tabel berisi mengenai informasi alamat asal paket yang dikirimkan oleh *client*.

2.5.1 *Never Queue*

Algoritma *never Queue* mengadopsi 2 konsep penjadwalan yaitu :

1. Ketika ada *server* yang berstatus *idle* / menganggur, *request* yang datang akan disalurkan ke *server* yang menganggur tersebut.
2. Tapi jika tidak ada *server* yang menganggur, maka *request* akan dikirimkan ke *server* yang memiliki *delay* koneksi terkecil dengan menggunakan algoritma *Shortest Expected Delay*.

Konsep algoritma *Shortest Expected Delay* adalah memberikan koneksi kepada *server* yang diharapkan memiliki delay koneksi terpendek. Berikut adalah rumus dasar pada algoritma *Shortest Expected Delay* : (GNU Free Doc License, 2006)

SED = $(C_i + 1) / U_i$

i = inisialisasi *server*

C_i = jumlah koneksi pada i *sever*

U_i = tetapan beban yang diberikan pada i

2.6 *Apache Jmeter*

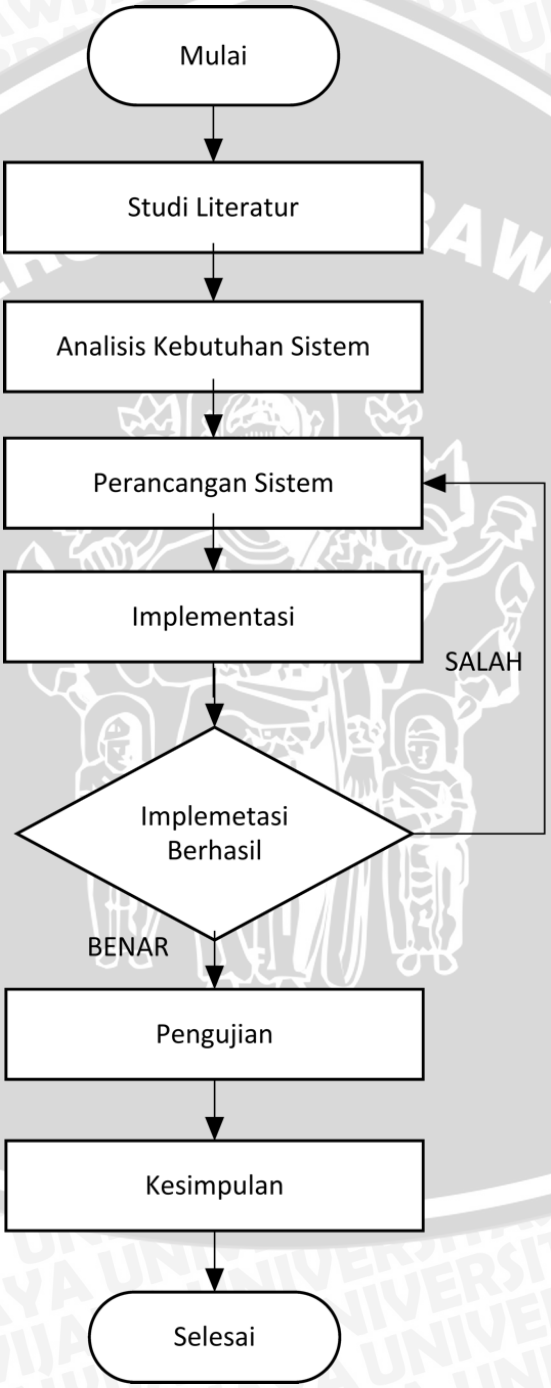
Apache JMeter adalah 100% aplikasi murni berbasis *Java* dirancang untuk Menguji *server* (seperti *web server*). Ini dapat digunakan untuk menguji kinerja baik pada *resources* statis dan dinamis seperti file statis, *Java Servlets, ASP.NET, PHP, CGI script, database, server FTP*, dan banyak lagi. *JMeter* dapat digunakan untuk mensimulasikan pengiriman beban pada server, jaringan atau objek untuk menguji kekuatan atau untuk menganalisa kinerja secara keseluruhan di bawah jenis beban yang berbeda. (Apache Fondation Software, 2014)

Selain itu, *JMeter* dapat membantu untuk menguji aplikasi dengan membiarkanskrip pengujian dengan pernyataan untuk memvalidasi bahwa aplikasi dapat mengembalikan hasil yang diharapkan. Untuk fleksibilitas maksimum, *JMeter* memungkinkan Anda membuat pernyataan ini menggunakan ekspresi reguler.

Stefano Mazzocchi dari *Apache Software Foundation* adalah pengembang asli dari *JMeter*. Alasan dia menulis itu terutama untuk menguji kinerja *Apache JServ* (proyek yang sejak itu telah digantikan oleh proyek *Apache Tomcat*). Dilakukan desain ulang *JMeter* untuk meningkatkan *GUI* dan untuk menambahkan kemampuan fungsional pengujian.*JMeter* menjadi proyek *Top Level Apache* pada bulan November 2011, yang berarti ia memiliki Manajemen Komite Proyek dan situs yang didedikasikan.

BAB 3 METODOLOGI PENELITIAN DAN PERANCANGAN

Pada bab ini membahas langkah-langkah untuk melakukan perancangan, implementasi, pengujian dan analisis algoritma *load balancing* pada sebuah sebuah *web server*. Berikut adalah diagram alur dari pelaksanaan penelitian dijelaskan pada gambar 3.1 :



Gambar 3.1 Diagram alur penelitian

3.1 Studi Literatur

Studi literatur merupakan dasar teori yang mendukung dalam perancangan dan implementasi untuk melakukan analisa kinerja algoritma penjadwalan *load balancer* pada layanan *web*. Dasar teori pendukung adalah sebagai berikut :

a. Penelitian Terkait

b. Web Server Cluster

c. *DBMS* (*Database Management System*)

i. MySQl

d. *LVS* (*Linux Virtual Server*)

i. *LVS* via *Direct Routing*

e. Algoritma Penjadwalan

i. Algoritma *Never Queue*

f. Apache Jmeter

3.2 Analisis Kebutuhan

Kegiatan analisis kebutuhan bertujuan untuk memperoleh semua kebutuhan yang diperlukan untuk tujuan penelitian. Yang dibutuhkan untuk membuat arsitektur *cluster web server* dengan menggunakan *load balancing* yaitu terdapat 3 buah *web server* / *real server* sebagai penyedia layanan *website* dan *database*, lalu dibutuhkan 1 *server director* yang berperan sebagai *load balancer* untuk membagi koneksi yang datang dari pengguna untuk disalurkan ke 3 buah *real server* di jaringannya. Lalu untuk pengujian dibutuhkan sebuah *client* yang akan bertindak untuk melakukan *stress testing* ke *cluster server* tersebut.

Dari analisis kebutuhan diatas, perlu dilakukan identifikasi terhadap perangkat lunak, toplogi jaringan dan perangkat keras yang akan digunakan dalam perancangan, implementasi dan analisa komparasi algoritma penjadwalan. Dengan demikian diharapkan penelitian dapat berjalan dengan mudah dan berjalan lancar.

3.2.1 Perangkat Keras yang digunakan

Arsitektur *cluster web server* dengan *load balancing* menggunakan 3 mesin sebagai *real server* serta *database server* dan 1 buah mesin sebagai *load balancer* dengan spesifikasi sebagai berikut :

1. Spesifikasi mesin *director* :
   a. Prosesor : Intel Core 2 Duo 2.2 GHz
   b. *Core* : 2 core
   c. *RAM* : 512 MB
   d. *Hardisk* : 30 GB

2. Spesifikasi mesin *real server* :
    a. Prosesor : Intel Core 2 Duo 2.2 GHz
    b. *Core* : 2 core
    c. *RAM* : 512 MB
    d. *Hardisk* : 40 GB

3.2.2 Perangkat Lunak yang Digunakan

Sistem *web server* dalam penelitian ini membutuhkan beberapa perangkat lunak diantaranya sebagai berikut :

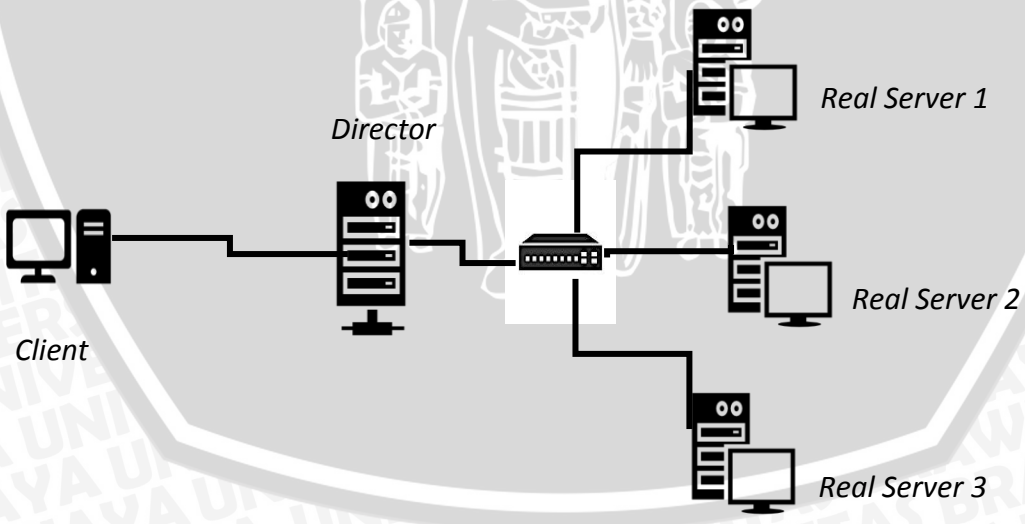
1. Sistem Operasi : *Centos* 5.5 *64-Bit*
2. *Web Server* : *Apache* dan *CMS Codeigniter*
3. *Load Balancing* : *Linux Virtual Server* dengan *ipvsadm*
4. *DBMS* : *MySQL Server*

3.3 Perancangan

Kegiatan perancangan dimulai ketika seluruh kebutuhan perangkat keras maupun perangkat lunak yang dibutuhkan sudah didapatkan dari hasil analisa kebutuhan. Penelitian dan analisis sistem dilakukan dalam jaringan lokal dan menggunakan 1 *client*, 1 *director*, dan 3 *real server*.

3.3.1 Perancangan Jaringan

Konsep topologi serta arsitektur jaringan sistem *web load balancing* yang akan dibangun adalah seperti pada gambar 3.2 :



Gambar 3.2 Perancangan arsitektur *web load balancing*

Director bertugas sebagai pintu masuk *request* dari *client* dan meneruskan koneksi yang datang dari *client* kepada *web server*. Lalu *web server* bertugas memproses *request* yang datang dari *client*. Pada *real server* terdapat *database* serta terjadi proses *sinkronisasi* data dan *session* antar *real server* untuk distribusi data serta pembagian *session*.

3.3.2 Perancangan Perangkat Lunak dan Perangkat Keras

Penelitian ini membutuhkan beberapa perangkat keras dan perangkat lunak. Perangkat keras dibuthkan untuk membangun arsitektur / topologi jaringan *cluster web server*, lalu untuk perangkat lunak yang dibutuhkan yaitu menggunakan perangkat lunak yang memiliki lisensi *open source* seperti sistem operasi *Centos* 5.5 *64-Bit*, *web server Apache*, *CMS Codeigniter*, *DBMS MySQL* dan *LVS* dengan *tool ipvsadm*. Berikut adalah penjelasan kegunaan perangkat lunak tersebut :

1. *Apache* : merupakan *web server* yang dibutuhkan untuk
2. *MySQL* : *database* yang dibutuhkan untuk penyimpanan data serta *sinkronisasi session* antar *real server* dari pengguna.
3. *Ipvsadm* : merupakan *tool* untuk *director* untuk membagi / *routing request* dari *client* kepada *web server* dengan algoritma yang terdapat didalam modulnya.

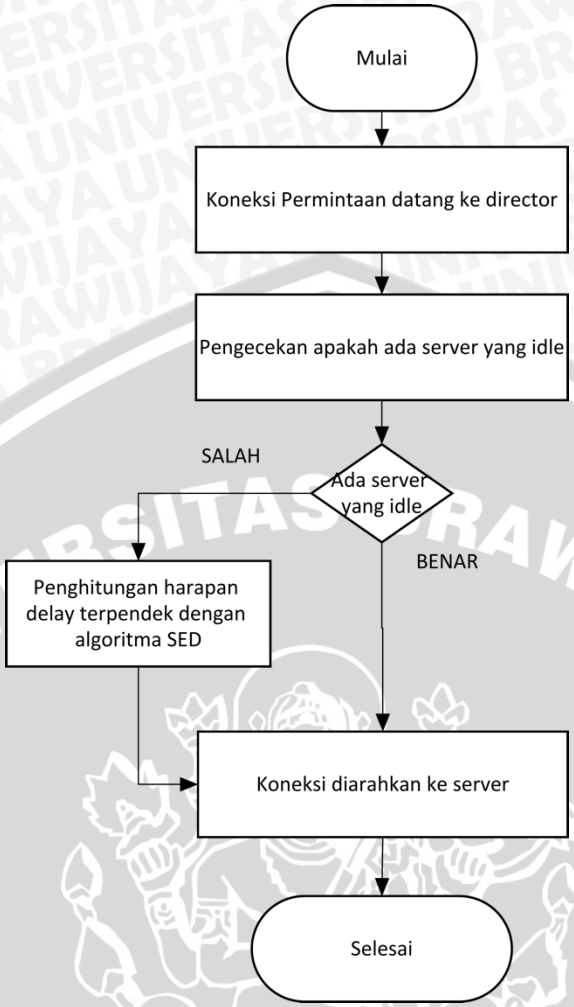
Dan untuk perangkat keras yang dibutuhkan untuk perancangan sistem *web load balancing* dengan menggunakan PC yang terdiri dari 3 buah mesin sebagai *real server*, dan 1 buah mesin sebagai *director*.

3.4 Algoritma penjadwalan *Never Queue* (*NQ*)

Algoritma penjadwalan *Never Queue* (*NQ*) merupakan algoritma yang digunakan serta di analisa untuk penelitian ini. Algoritma penjadwalan *never queue* akan diterapkan pada *director* yang bertugas membagi koneksi ke *server* yang ada di dalamnya. Algoritma *never queue* akan bertugas menentukan pembagian koneksi yang datang dari *client* saat mengakses alamat *IP director*.

Pada gambar 3.3 dijelaskan tentang proses kerja dari algoritma penjadwalan *never queue*. Algoritma *never queue* melakukan tugasnya sebagai pembagi koneksi dari *client* dengan menggunakan 2 konsep penjadwalan yaitu :

1. Ketika ada *server* yang berstatus *idle* / menganggur, *request* yang datang akan disalurkan ke *server* yang menganggur tersebut.
2. Tapi jika tidak ada *server* yang menganggur, maka *request* akan dikirimkan ke *server* yang memiliki *delay* koneksi terkecil dengan menggunakan algoritma *Shortest Expected Delay*.



Gambar 3.3 Diagram alur algoritma penjadwalan *Never Queue* (*NQ*)

Pada dasarnya algoritma penjadwalan *never queue* akan menentukan koneksi diarahkan ke *server* yang mana *server* tersebut masih berstatus *idle*. Jika pada saat penentuan pembagian koneksi tidak ada *server* yang sedang dalam kondisi *idle*, maka koneksi akan diarahkan ke *server* yang diharapkan memiliki *delay* koneksi terpendek dengan cara melakukan penghitungan dengan konsep algoritma *Shortest Expected Delay*. Berikut adalah rumus dasar pada algoritma *Shortest Expected Delay* :

SED = (C_i + 1) / U_i

i = inisialisasi *server*
C_i = jumlah koneksi pada i *sever*
U_i = tetapan beban yang diberikan pada i

Pada rumus penentuan harapan *delay* koneksi terpendek atau *Shortest Expected Delay*, nilai tetapan beban yang diberikan kepada *server* adalah dari konfigurasi yang ditetapkan menentukan beban koneksi yang akan diterima untuk setiap *server*.

Berikut adalah contoh proses kerja dari algoritma penjadwalan *never queue* dalam menentukan pembagian beban koneksi :

- Untuk tiap *server* diberika tetapan beban yang sama, konfigurasi tetapan beban *server* yaitu :
    - *Server* 1 pada alamat *IP* 10.0.2.11 memiliki tetapan beban sebesar 1
    - *Server* 2 pada alamat *IP* 10.0.2.12 memiliki tetapan beban sebesar 1
- Jumlah koneksi = 5 koneksi
- Rata – rata waktu respon tiap koneksi = 313 ms

Pada koneksi yang datang sejumlah 5 koneksi dan rata-rata untuk waktu respon untuk setiap koneksi dari *director* yaitu sebesar 1023 *ms*, berikut adalah contoh hasil dari pengamatan waktu koneksi yang datang pada table 3.1:

Tabel 3.1 Contoh waktu pengiriman untuk setiap koneksi

Nomor Koneksi	Waktu Pengiriman Koneksi
1	08:34:20.144
2	08:34:20.226
3	08:34:20.325
4	08:34:20.540
5	08:34:20.660

Maka untuk setiap koneksi yang datang penghitungannya adalah sebagai berikut :

1. Koneksi 1

   Server semua dalam keadaan *idle*, maka koneksi akan diarahkan ke *server* yang berada pada urutan konfigurasi pertama yaitu pada *server* 1 dengan waktu respon sebesar 313 *ms*, artinya *server* 1 akan bertatus *idle* kembali pada waktu 08:34:20.457 dan *server* 2 masih berstatus *idle*.
2. Koneksi 2

   Pengiriman koneksi kedua terjadi pada waktu 08:34:20.226 yang artinya *server* 1 masih belum dalam status *idle*, maka koneksi akan diarahkan ke *server* 2 yang masih berstatus *idle*. Pada waktu respon sebesar 313 *ms* maka *server* 2 akan berstatus *idle* kembali pada 08:34:20.539
3. Koneksi 3

   Untuk koneksi ketiga yang datang pada waktu 08:34:20.325, diperoleh keadaan bahwa *server* 1 dan *server* 2 masih belum dalam keadaan *idle*. Maka selanjutnya untuk menentukan pembagian koneksi dilakukan penghitungan dengan algoritma *Shortest Expected Delay* untuk menentukan *server* mana yang memiliki harapan *delay* koneksi terpendek, untuk *server* 1 dan *server* 2 pada waktu ini masih menangani koneksi dari koneksi pertama dan kedua, maka penghitungannya sebagai berikut :

   - *Server* 1
     SED = (1+1)/1 = 2
     Nilai harapan *delay* pada *server* 1 sebesar = 2
   - *Server* 2

SED = (1+1)/1 = 2

Nilai harapan *delay* pada *server* 2 sebesar = 2

Terlihat diantara kedua *server* memiliki harapan *delay* koneksi yang sama sebesar 2, artinya koneksi akan diarahkan ke *server* yang berada pada urutan pertama yaitu *server* 1. Pada saat memproses koneksi didapatkan waktu respon sebesar 313 *ms* artinya *server* 1 akan kembali berstatus *idle* pada waktu 08:34:20.638.

4. Koneksi 4

   Pengiriman koneksi keempat terjadi pada waktu 08:34:20.540. Pada waktu ini artinya *server* 1 masih bekerja memproses dari koneksi ketiga, dan *server* 2 sudah dalam keadaan *idle*. Maka koneksi akan langsung diarahkan ke *server* 2 yang artinya *server* 2 akan kembali *idle* pada waktu 08:34:20.853 dengan waktu respon sebesar 313 *ms*

5. Koneksi 5

   Koneksi kelima dikirimkan pada waktu 08:34:20.660 yang dalam kondisi *server* 1 sudah dalam keadaan *idle* setelah memproses dari koneksi ketiga, dan *server* 2 yang masih bekerja memproses koneksi keempat. Maka koneksi akan diarahkan ke *server* 1 yang sudah dalam keadaan *idle*.

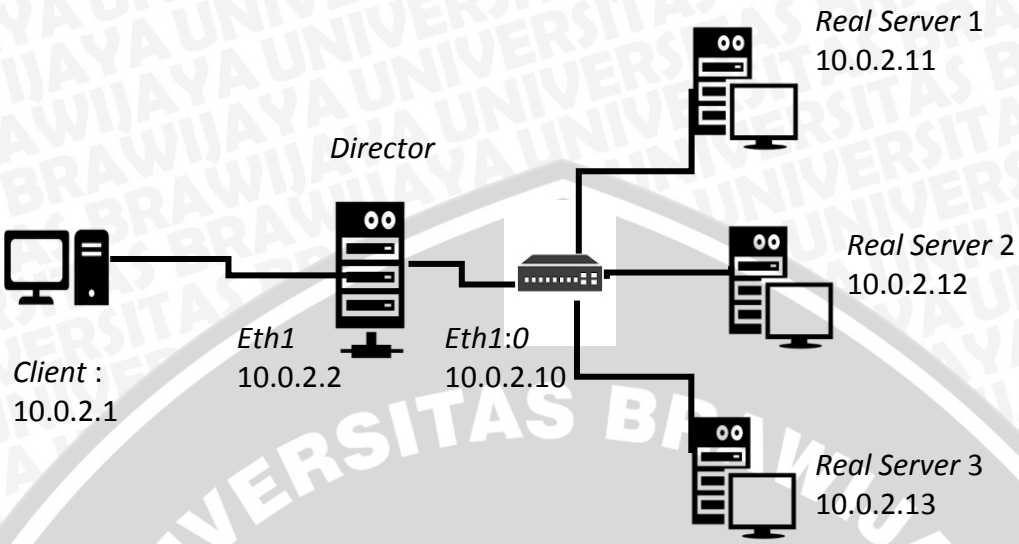
3.5 Implementasi

Kegiatan implementasi dilakukan berdasarkan hasil dari perancangan. Pada penelitian ini kegiatan implementasi terdiri dari implementasi jaringan dan pengalamatan *IP* serta instalasi dan konfigurasi perangkat lunak pada *real server* dan *director*.

3.5.1 Implementasi jaringan dan pengalamatan IP

Konfigurasi jaringan antara *client* dengan *web load balancing* dilakukan dengan konsep *LVS direct routing* dimana antara *client*, *director*, dan *real server* memiliki alamat *IP* dalam satu jaringan. Karena nantinya setelah terjadi pembagian *request* oleh *director*, maka *real server* akan langsung berkomunikasi dengan *client* tanpa harus melalui *director* lagi.

Untuk pengalamatan IP dari *client*, *director*, dan *real server* semuanya berada pada satu jaringan. Karena pada konsep *direct routing* nantinya *real server* bias membuat jalur koneksi sendiri dengan *client* tanpa harus melewati *director*. *Director* memiliki 2 alamat *IP* dimana *eth1* sebagai alamat *IP* dari *interface network* dan *eth1:0* sebagai *VIP* dimana nantinya *client* mengakses alamat *VIP* untuk melakukan *request* dan *director* akan meneruskan *request* tersebut ke *real server* lalu *real server* akan merespon *request* dan mengembalikan langsung ke *client* tanpa melalui *director* karena *real server* juga berada pada satu *netwrok* dengan *client*. Pada gambar 3.3 berikut ini adalah konfigurasi jaringan :



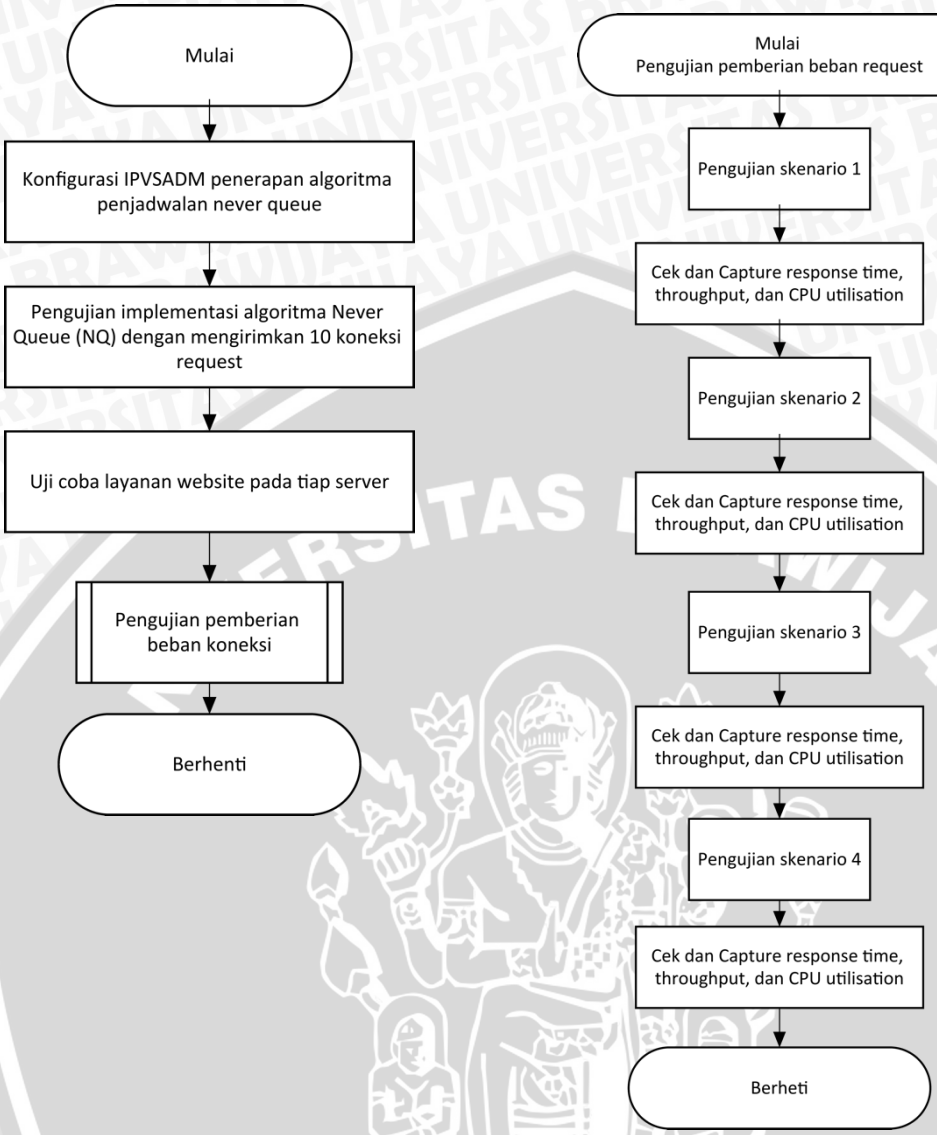
Gambar 3.4 Pengalamatan *IP web load balancing*

3.5.2 Instalasi dan konfigurasi perangkat lunak

Director dan *real server* menggunakan sistem operasi *Centos* 5.5 *64-Bit*. Pada *director* dilakukan konfigurasi terhadap *tool ipvsadm* serta *forwarding*nya, dan untuk *real server* dilakukan konfigurasi terhadap *web server*, *database* serta *gateway* terhadap *director*.

3.6 Pengujian

Pada tahap ini akan dilakukan pengujian terhadap sistem *web load balancing* untuk membuktikan apakah algoritma penjadwalan *never queue* mampu membagi koneksi secara seimbang ke 3 *real server* yan berada di dalamnya. Akan dilakukan juga pengujian untuk membuktikan proses kerja dari algoritma penjadwalan *never queue* dengan mengirimkan 10 koneksi kepada sistem *web load balancing.* Pengujian selanjutnya adalah memastikan apakah ketiga real server benar-benar mampu memberikan layanan *web* ketika *client* melakukan *request* ke *web load balancing*. Setelah itu pada pengujian pemberian beban *request* akan dilakukan pengujian yang terdiri dari 4 skenario dimana setiap skenario dilakukan pengujian dengan memberikan *request* dalam jumlah koneksi yang berbeda. Pengujian pemberian *request* dilakukan dengan kondisi konfigurasi antar *real server* memiliki spesifikasi dan beban yang sama. Alur pengujian pemberian beban request dapat dilihat pada gambar 3.4 berikut :



Gambar 3.5 Diagram alur pengujian

Pada setiap skenario dilakukan pengujian pemberian beban *request* kepada *web load balancing* dengan pengiriman *request* sebesar 50 *request*/detik, pada skenario kedua dilakukan pengiriman *request* sebesar 80 *req*uest/detik, skenario ketiga dilakukan pengiriman request sebesat 120 *request*/detik dan skenario keempat pengiriman *request* sebesar 170 *request*/detik. Lalu pada setiap pemberian beban *request* dilakukan pengecekan dan *capture* terhadap hasil *response time*, *throughput*, *request rate* dan pembagian *request* pada ketiga *real server*. Untuk setiap skenario dilakukan pengujian dengan interval pengulangan sebanyak 10 kali, 30 kali, dan 60 kali.

3.7 Pengambilan kesimpulan dan saran

Kegiatan pengambilan kesimpulan dilakukan setelah mendapatkan data dari hasil pengujian dan analisis yang telah dilakukan. Hasil kesimpulan diperoleh berdasarkan pada data yang diperoleh dari pengujian. Isi dari kesimpulan

diharapkan performa algoritma *never queue* dapat bekerja optimal pada *web load balancing* dengan menggunakan arsitektur *Linux Virtual Server* via *direct routing*. Pada akhir penulisan terdapat saran untuk penyempurnaan penelitian ini selanjutnya.

BAB 4 IMPLEMENTASI

Bab ini akan dilaksanakan langka-langkah untuk mengimplementasikan *web load balancing* via *LVS-DR* dengan menggunakan algoritma penjadwalan *never queue* dengan merujuk pada perancangan yang telah dibuat, lalu layanan *web* akan diuji dengan langkah-langkah yang telah dituliskan pada bab metodologi penelitian dan perancangan.

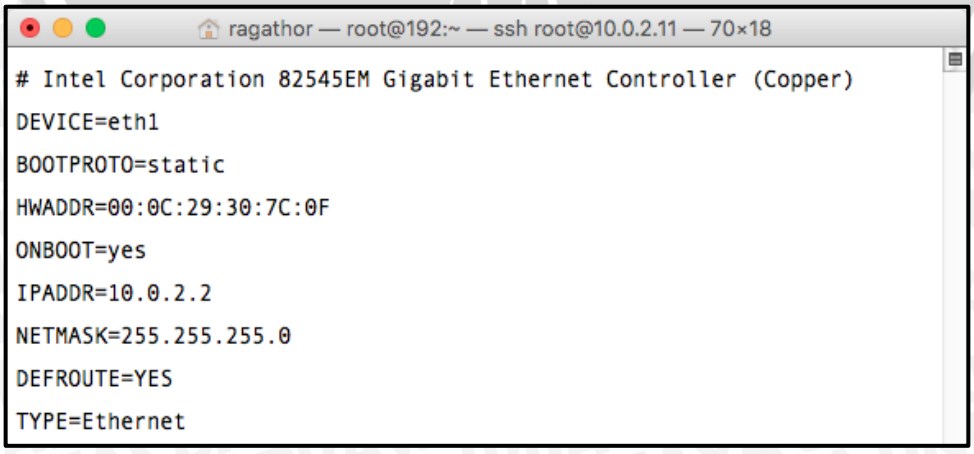
Tahap implementasi adalah proses yang dilakukan penulis untuk mengimplementasikan konsep dari perancangan. Langkah – langkah yang dilakukan antara lain berupa instalasi dan konfigurasi pada *web load balancing*. Instalasi dan konfigurasi akan dilakukan pada bagian perangkat lunak dan perangkat keras yang digunakan.

4.1 Implementasi load balancer

Langkah pertama yang dilakukan dalam merancang sistem *web load balancing* yaitu konfigurasi serta instalasi pada *load balancer* atau *director*. Pada *director* diinstall *system* operasi *Centos* 5.5 *64-bit* dengan versi minimalis dan dilanjutkan dengan instalasi *IPVSADM*.

Setelah instalasi perangkat lunak sudah terinstall, langkah selanjutnya adalah melakukan konfigurasi pada sistem operasi dan perangkat lunak. Pada sistem operasi akan dilakukan konfigurasi *IP network interface* yang ada pada *director* agar *director* memiliki *IP* statis untuk komunikasi antara *client* dan *real server*. Pada system operasi *centos*, konfigurasi *network interface* terletak pada *script file ifcfg-eth1*. Pada *script* ini, penulis merubah beberapa baris diantaranya pada *BOOTPROTO*, *IPADDR* dan *NETMASK*. Berikut langkah-langkah dan konfigurasi *script ifcfg-eth1* :

Konfigurasi *network interface eth1*
`#cd /etc/sysconfig/network-scripts/` `#vi ifcfg-eth1`



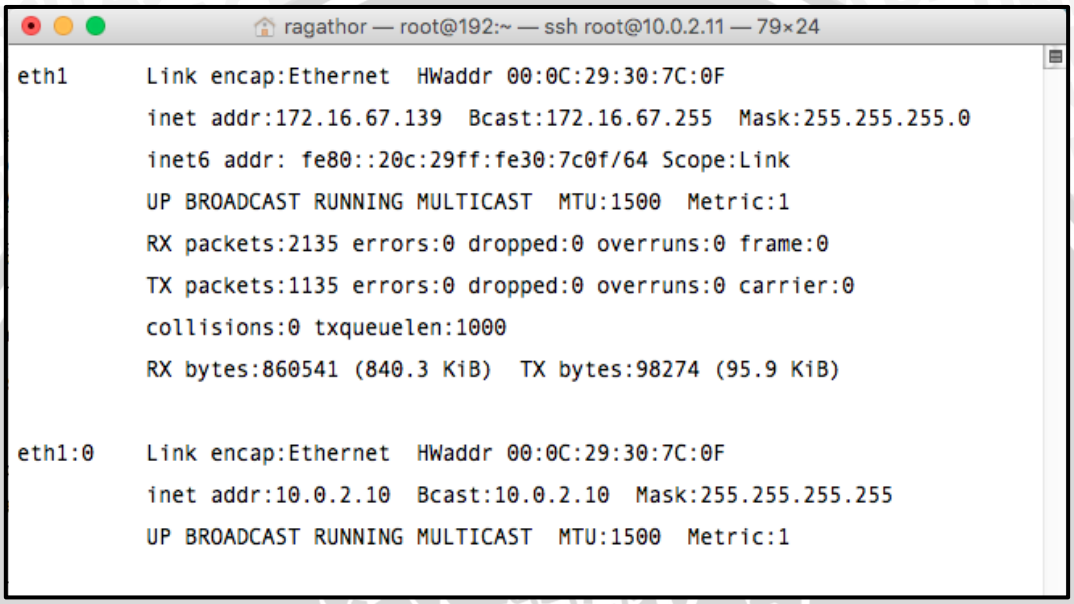
Gambar 4.1 Konfigurasi *eth1* pada *load balancer*

Pada gambar 4.1 Dilakukan perubahan *script ifcfg-eth1* pada baris *BOOTPROTO* dirubah dengan *static*, lalu pada *IPADDR* diberi alamat *IP* 10.0.2.2 dan pada *NETMASK* 255.255.255.0.

Langkah selanjutnya yaitu ditambahkan virtual *IP* / *VIP* pada *network interface eth1* sebagai alamat akses untuk *client* dan komunikasi antar *real server* dengan konfigurasi *eth1*:0. Berikut adalah konfigurasinya :

Konfigurasi *VIP eth1*:0

```
#ifconfig eth1:0 10.0.2.10 netmask 255.255.255.255 up
```



Gambar 4.2 Konfigurasi *VIP eth1*:0 pada *load balancer*

Pada gambar 4.2 diatas adalah hasil konfigurasi pada *network interface eth1* dengan alamat *IP* 10.0.2.2 *netmask* 255.255.255.0 dan konfigurasi *VIP* 10.0.2.10 *netmask* 255.255.255.255 dengan nama *eth1*:0 sebagai alamat jalur akses *client* kepada *load balancer* dan alamat *gateway* dari setiap *real server* untuk mengarahkan koneksi dari *client* ke *real server*.

Langkah selanjutnya adalah konfigurasi pada *IP forwarding director* agar *director* / *load balancer* bisa langsung mem-*forward*-kan paket data yang dikirim dari *client* kepada *real server*. Konfigurasi yang dilakukan adalah sebagai berikut :

Konfigurasi *IP forwarding*

```
#echo "1" > /proc/sys/net/ipv4/ip_forward

#sysctl -p
```

Pada gambar berikut adalah hasil konfigurasi. Dilakukan perubahan pada *script sysctl.conf* dengan merubah pada baris *net.ipv4.ip_forward* = 1 yang artinya angka 1 adalah *true* dimana *director* akan bekerja melakukan *IP forwarding* jika ada paket yang datang.

Gambar 4.3 Konfigurasi *sysctl* pada *load balancer*

Selanjutnya adalah konfigurasi pada *IPVSADM* agar *director* bekerja sebagai pembagi paket data yang masuk ke 3 *real server* di bawahnya. Karena konsep *LVS* pada penelitian ini menggunakan *direct routing*, maka paket yang datang langsung diarahkan ke *real server* dan setelah itu *real server* akan langsung berkomunikasi dengan *client*, karena konsep *direct routing* yaitu *director* dan *real server* sama-sama memiliki alamat *IP* publik. Konfigurasi awal *IPVSADM* adalah memasukkan alamat *IP director* yang bertindak sebagai penerima paket dari luar dan pemberian konfigurasi algoritma penjadwalan *never queue*, berikut adalah perintah yang dilakukan :

Pendaftaran *VIP* pada *IPVSADM*
`#ipvsadm -C` `#ipvsadm -A -t 10.0.2.10:80 -s nq`

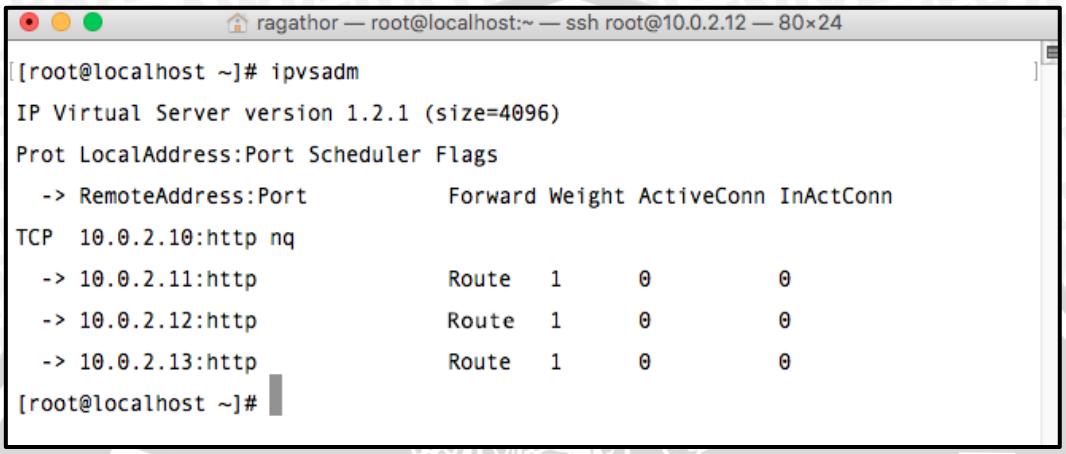
Pada perintah diatas *ipvsadm -c* artinya melakukan penghapusan pada konfigurasi default *ipvsadm*. Lalu ditambahkan alamat *IP* 10.0.2.10 dimana alamat *IP* tersebut digunakan sebagai alamat penerima paket dari *client* melalui *port* 80 (*http*) yang akan diteruskan ke *real server*. Konfigurasi selanjutnya adalah perintah *–s nq*, perintah ini bermaksud memberikan konfigurasi agar *ipvsadm* menggunakan algoritma penjadwalan *never queue* yang sudah ada pada modul *IPVSADM*.

Konfigurasi selanjutnya dalah mendaftarkan alamat *IP* dari *real server* pada *ipvsadm*. Ketiga *server* memiliki alamat *IP* 10.0.2.11, 10.0.2.12, dan 10.0.2.13. berikut perintah yang dilakukan :

Pendaftaran *IP real server* pada *IPVSADM*

`#ipvsadm -a -t 10.0.2.10:80 -r 10.0.2.11:80 -g -w 1`
`#ipvsadm -a -t 10.0.2.10:80 -r 10.0.2.12:80 -g -w 1`
`#ipvsadm -a -t 10.0.2.10:80 -r 10.0.2.13:80 -g -w 1`

Pada perintah diatas alamat *IP director* 10.0.2.10 pada *port* 80 akan meneruskan paket pada ketiga *real server* pada alamat *IP* 10.0.2.11, 10.0.2.12, dan 10.0.2.13 yang didaftarkan. Diteruskan dengan penambahan -w 1 yang artinya tiap *real server* diberi beban sama dengan batas paket yang dikirim sebanyak 1. Dari kedua konfigurasi *IPVSADM* diatas, maka berikut adalah hasil aturan yang telah dikonfigurasikan pada *ipvsadm* :



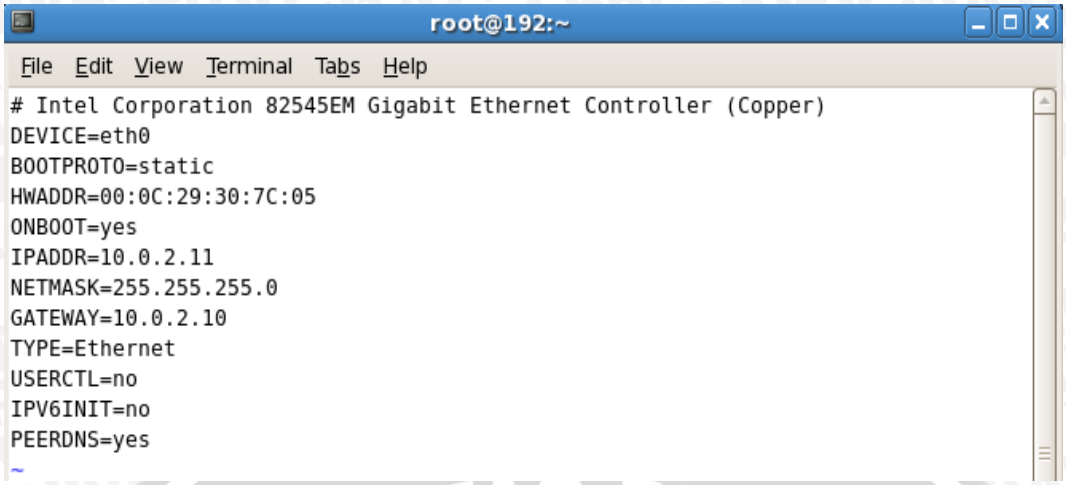
Gambar 4.4 Hasil konfigurasi *ipvsadm*

4.2 Implementasi *real server*

Implementasi selanjutnya adalah instalasi dan konfigurasi pada ketiga *real server*. Langkah awal adalah instalasi sistem operasi pada *real server*. Untuk kali ini sistem operasi yang digunakan adalah *centos* 5.5 *64-bit* dengan *mode desktop* agar dapat secara langsung *monitoring* tampilan website yang disediakan. Lalu pada tiap *real server* diinstall *web server* dengan menggunakan *apache web server*, *php*, serta *database* menggunakan *mysql*.

Setelah selesai proses instalasi, selanjutnya aakan dilakukan konfigurasi beberapa *script* pada sistem operasi. Seperti *director*, hal yang pertama dilakukan adalah konfigurasi alamat *IP* pada tiap *network interface real server*. Konfigurasi *network interface* terletak pada *script* file *ifcfg-eth0*. Pada *script* ini, penulis merubah beberapa baris diantaranya pada *BOOTPROTO*, *IPADDR*, *NETMASK*, dan *GATEWAY*. Berikut adalah konfigurasi *script ifcfg-eth0* :

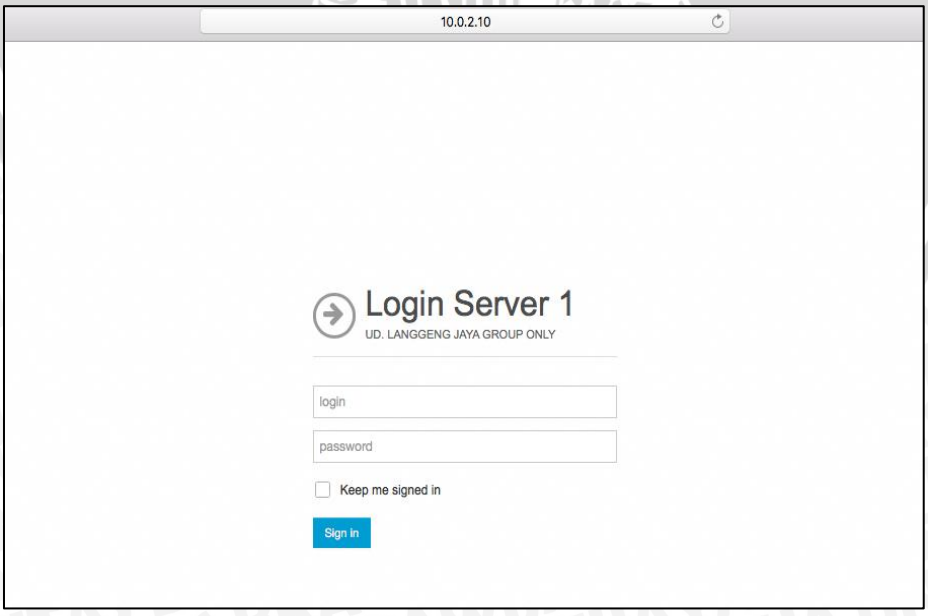
Konfigurasi *network interface eth0*
`#cd /etc/sysconfig/network-scripts/` `#vi ifcfg-eth0`



Gambar 4.5 Konfigurasi *eth0* pada *real server*

Pada gambar 4.5 *script ifcfg-eth0* diatas terjadi perubahan pada baris *BOOTPROTO* menjadi *static*, *IPADDR* pada ketiga *server* yaitu 10.0.2.11, 10.0.2.12, 10.0.2.3, *NETMASK* 255.255.255.0, dan *GATEWAY* diarahkan ke alamat *IP director* yaitu 10.0.2.10. Alamat *IP real server* terletak pada satu *network* dengan *client* dan *director* dikarenakan konsep *LVS-DR* adalah *real server* dan *director* menggunakan alamat *IP public* sehingga bisa langsung berkomunikasi dengan *client* sehingga ketika paket datang dari *director* ke *real server*, maka *real server* akan langsung mengembalikan paket ke *client* tanpa harus melewati *director* lagi.

Selanjutnya dilakukan konfigurasi untuk aplikasi *web* yang akan digunakan untuk pengujian. *Website* dibuat dengan berbasis *MVC* (*Model*, View, *Controller*) menggunakan *CMS* (*Conten Management System*) *codeigniter* yang berupa aplikasi *web login*. Pada gambar 4.6 dibawah ini adalah contoh tampilan utama *web* aplikasi yang digunakan untuk pengujian :



Gambar 4.6 Tampilan aplikasi *web*

Pada aplikasi *web* dilakukan beberapa konfigurasi untuk mengarahkan alamat *database* serta menambahkan *session* yang awalnya berupa *text* menjadi berbasis pada *session database* agar setiap *web server* dapat saling melakukan *sinkronisasi session* ketika ada pengguna yang melakukan *login*. Berikut pada gambar 4.7 adalah konfigurasi *database* dan pada gambar 4.8 yaitu konfigurasi *session database* pada *codeigniter* :

```
$active_group = 'default';
$active_record = TRUE;

$db['default']['hostname'] = 'localhost';
$db['default']['username'] = 'root';
$db['default']['password'] = 'raga';
$db['default']['database'] = 'ci_blog';
$db['default']['dbdriver'] = 'mysql';
$db['default']['dbprefix'] = '';
$db['default']['pconnect'] = TRUE;
$db['default']['db_debug'] = TRUE;
$db['default']['cache_on'] = FALSE;
$db['default']['cachedir'] = '';
$db['default']['char_set'] = 'utf8';
$db['default']['dbcollat'] = 'utf8_general_ci';
$db['default']['swap_pre'] = '';
$db['default']['autoinit'] = TRUE;
$db['default']['stricton'] = FALSE;
```

Gambar 4.7 Konfigurasi *database* pada *real server*

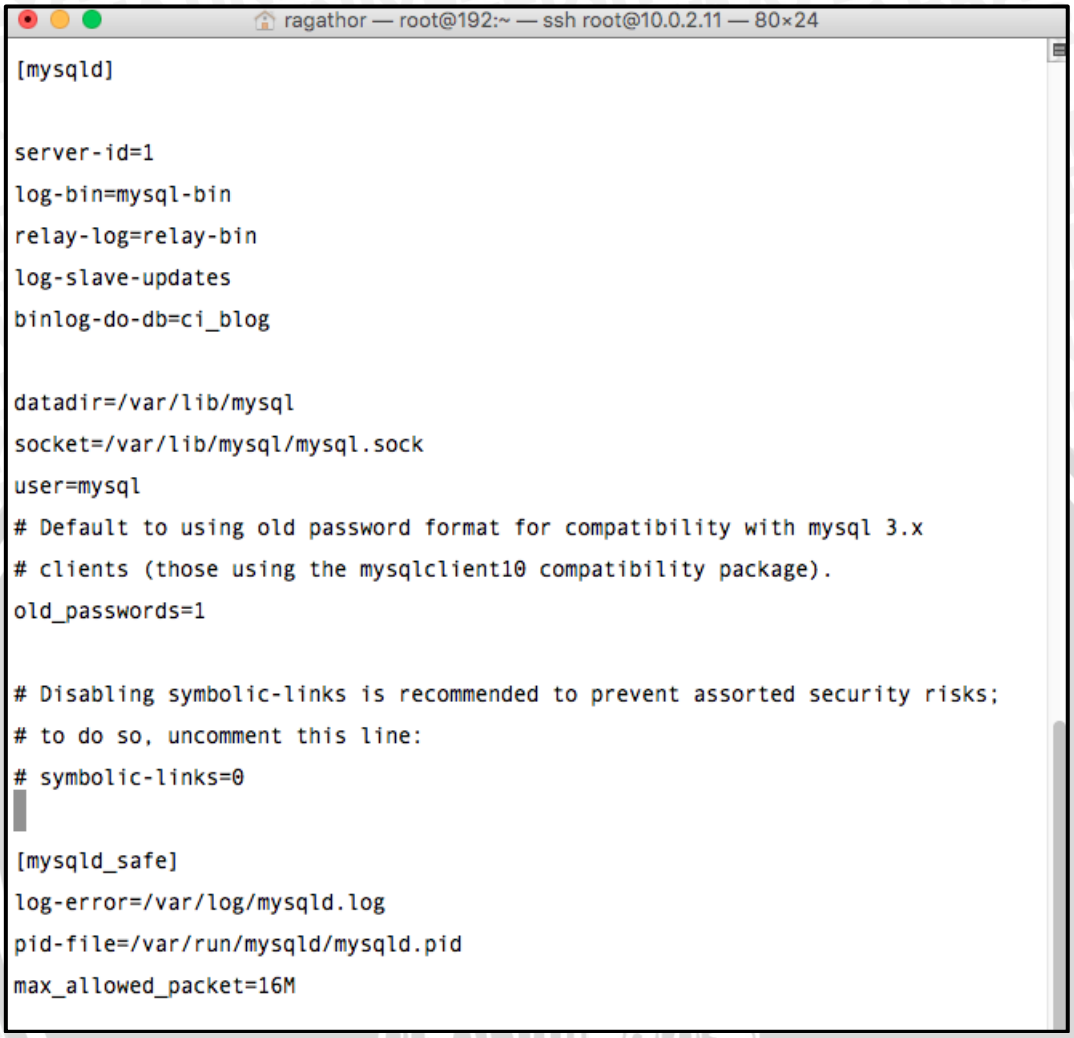
```
$active_group = 'default';
$active_record = TRUE;

$db['default']['hostname'] = 'localhost';
$db['default']['username'] = 'root';
$db['default']['password'] = 'raga';
$db['default']['database'] = 'ci_blog';
$db['default']['dbdriver'] = 'mysql';
$db['default']['dbprefix'] = '';
$db['default']['pconnect'] = TRUE;
$db['default']['db_debug'] = TRUE;
$db['default']['cache_on'] = FALSE;
$db['default']['cachedir'] = '';
$db['default']['char_set'] = 'utf8';
$db['default']['dbcollat'] = 'utf8_general_ci';
$db['default']['swap_pre'] = '';
$db['default']['autoinit'] = TRUE;
$db['default']['stricton'] = FALSE;
```

Gambar 4.8 Konfigurasi *session database* pada *real server*

Setelah konfigurasi pada *website*, lalu dilakukan konfigurasi terhadap masing-masing *database* pada *real server* agar antar *database* terjadi *sinkronisasi* data pada *database*. Pada *MYSQL* konsep *sinkronisasi database* menggunakan metode *replikasi* dimana nantinya dari ketiga *database* pada tiap *server* terjadi

proses *circular master slave*. Berikut adalah konfigurasi *replication* pada *database*:



Gambar 4.9 Konfigurasi *my.cnf*

Pertama-tama dilakukan konfigurasi pada *script* */etc/my.cnf* dengan menambahkan *server-id* sesuai dengan urutan *real sever*, lalu *log-bin=mysql-bin* adalah menunjukkan *log* mana yang akan *direplikasi* lalu pada *binlog-do-db=ci_blog* adalah *database* yang dituju untuk *replikasi*. Setelah itu dilakukan inistialisasi dengan menambahkan *user* untuk *slave* pada *MYSQL*, lalu konfigurasi untuk *MASTER_HOST* dimana pada *server* 1 *MASTER SLAVE*-nya adalah *database* pada *server* 3, *server* 2 mengarah pada *server* 1, dan pada *server* 3 *MASTER SLAVE* mengarah ke *server* 2. Maka dari itu akan terjadi proses perputaran *master to master* untuk proses *slave replication*. Berikut adalah contoh hasih konfigurasi replikasi pada gambar 4.9 adalah hasil konfigurasi pada *file my.cnf*, gambar 4.10 yaitu menambahkan *user privileges* untuk *replikasi*, dan gambar 4.11 adalah *monitor* dari hasil konfigurasi *slave replication* :

```
ragathor — root@192:~ — ssh root@10.0.2.11 — 84×8
mysql> CHANGE MASTER TO MASTER_HOST='10.0.2.12', MASTER_USER='repl',MASTER_PASSWORD=
'r3plic4tor',MASTER_LOG_FILE='mysql-bin.000025',MASTER_LOG_POS=29722;
Query OK, 0 rows affected (0.05 sec)

mysql> start slave;
Query OK, 0 rows affected (0.04 sec)

mysql>
```

Gambar 4.10 Penambahan *user slave*

```
ragathor — root@192:~ — ssh root@10.0.2.11 — 80×19
mysql> show slave status\G
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: 10.0.2.12
                  Master_User: repl
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: mysql-bin.000025
          Read_Master_Log_Pos: 29722
               Relay_Log_File: relay-bin.000013
                Relay_Log_Pos: 235
        Relay_Master_Log_File: mysql-bin.000025
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
```

Gambar 4.11 Hasil konfigurasi *slave*

BAB 5 PENGUJIAN DAN ANALISA

Pengujian awal yaitu melakukan pengujian untuk cara kerja dari algoritma penjadwalan *never queue*. Selanjutnya dilakukan pengujian terhadap *web load balancing* via *LVS-DR* dengan algoritma penjadwalan *never queue* yaitu membuktikan apakah *director* mampu membagi koneksi ke 3 *real server* di dalamnya. Setelah itu akan dilakukan pengujian dengan memberikan *request* dalam jumlah koneksi yang berbeda. Pengujian dilakukan dengan pemberian *request* dengan konfigurasi antar *real server* memiliki spesifikasi dan beban yang sama.

Pada pengujian pemberian beban *request*, Pada setiap skenario dilakukan pengujian pemberian beban *request* kepada *web load balancing* dengan pengiriman request sebesar 50 request/detik, pada skenario kedua dilakukan pengiriman request sebesar 80 *req*uest/detik, scenario ketiga dilakukan pengiriman request sebesat 120 request/detik dan scenario keempat pengiriman request sebesar 170 request/detik. Lalu pada setiap pemberian beban *request* dilakukan pengecekan dan *capture* terhadap hasil *response time*, *throughput*, *request rate* dan pembagian request pada ketiga *real server*. Untuk setiap skenario dilakukan pengujian dengan interval pengulangan sebanyak 10 kali, 30 kali, dan 60 kali.

5.1 Pengujian implementasi algoritma *Never Queue* (*NQ*)

Pengujian ini dilakukan untuk mengamati alur kerja dari algoritma penjadwalan *never queue* dalam membagi koneksi yang datang untuk setiap *server*. Pengujian dilakukan pada sistem *web load balancing* yang telah diimplementasikan dengan terdiri dari 1 *director* dan 3 *real server* dengan tetapan beban masing-masing *server* sebesar 1.

5.1.1 Perangkat pengujian

Pada pengujian untuk analisa alur kerja dari algoritma penjadwalan *never queue* digunakan *tool Apache Jmeter*. Konfigurasi yang perlu diperhatikan adalah pada 3 buah komponen dari *Apache Jmeter* berupa *Group Thread*, *Controller http sample*, dan *listener* di dalam *http sampler*. *Group thread* akan akan dikonfigurasi dengan jumlah *thread* sebesar 10, *ramp up* sebesar 1 dan perulangan sebesar 1, artinya akan dilakukan pengiriman *request* sebanyak 10 kali dalam 1 kali pengiriman. Untuk *controller* berupa *http sampler* dikonfigurasikan untuk mengarahkan koneksi ke alamat *IP director*. Lalu pada *listener* ditambahkan *view result in table* untuk mendapatkan data waktu pengiriman dan waktu respon untuk setiap koneksi.

5.1.2 Hasil Pengujian

Dari pengujian dengan mengirimkan koneksi sebanyak 10 kali, data yang diperoleh dari *listener* berupa *result in table* pada perangkat lunak *apache Jmeter* adalah pada tabel 5.1 dan tabel 5.2 berikut:

Tabel 5.1 Pembagian koneksi pada setiap *server*

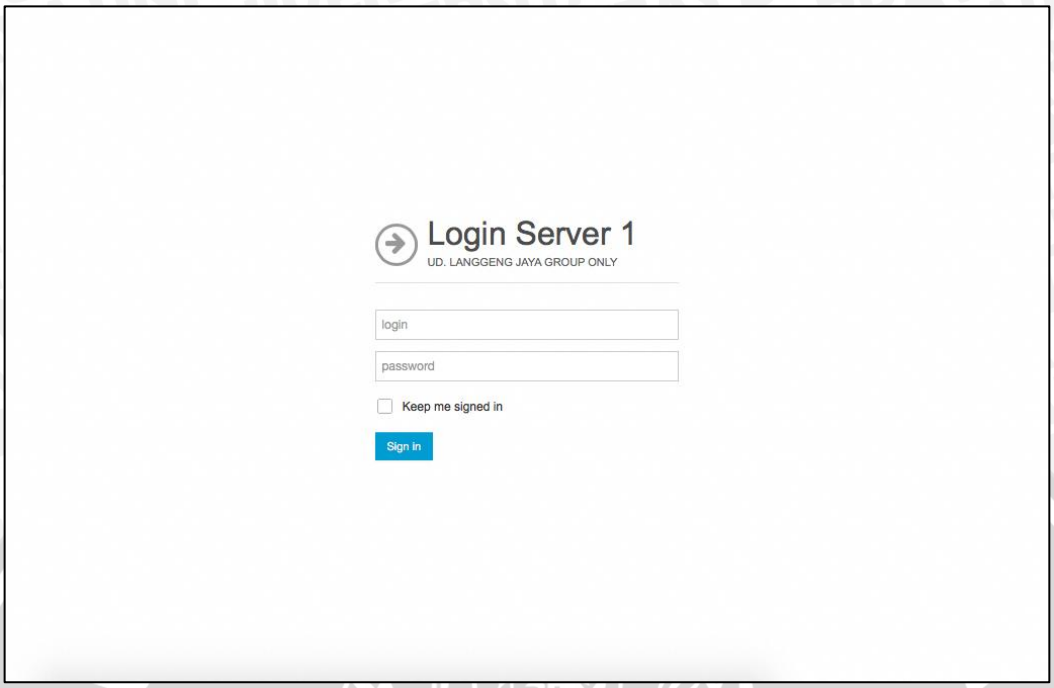
Server	Jumlah koneksi
1	5
2	3
3	2

Tabel 5.2 Waktu pengiriman dan *response time* dari setiap koneksi

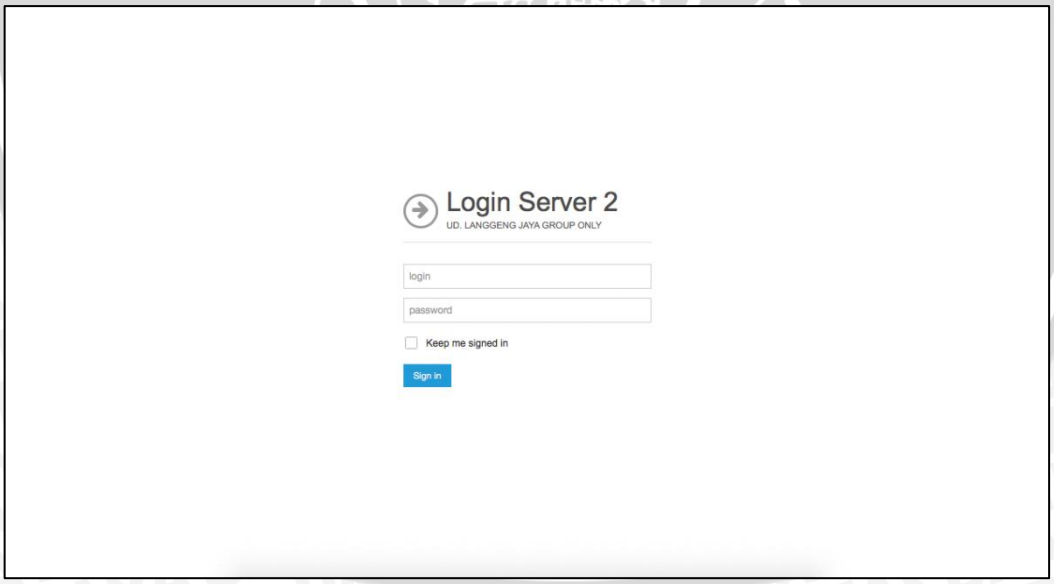
Nomor	Waktu Pengiriman	*Response Time*
1	20:34:40.144	107
2	20:34:40.226	258
3	20:34:40.325	378
4	20:34:40.426	278
5	20:34:40.530	226
6	20:34:40.627	245
7	20:34:40.734	351
8	20:34:40.832	268
9	20:34:40.932	373
10	20:34:41.037	475

5.2 Pengujian halaman *website*

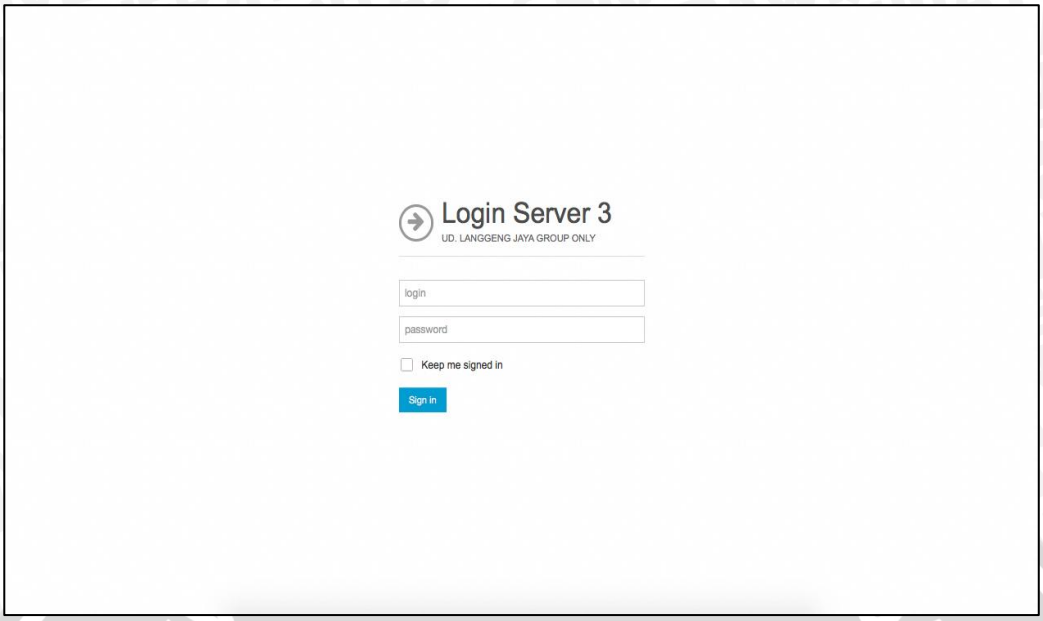
Pengujian halaman *website* adalah untuk membuktikan bahwa ketiga *real server* mampu memberikan respon layanan *website* ketika ada *request* yang datang, untuk halamn utama dari halaman *website* adalah berupa halaman *login* dengan memasukkan *username* dan *password*, untuk ketiga *real server* diberi tampilan pembeda agar bisa diketahui *server* mana yang sedang melayani *request*. Berikut adalah tampilan *website* dari ketiga *real server* dimana pada gambar 5.1 adalah tampilan *website* ketika *real server* 1 pada alamat *IP* 10.0.2.11 yang melayani *request*. Lalu pada gambar 5.2 diatas adalah tampilan *website* ketika *real server* 2 pada alamat *IP* 10.0.2.12 yang melayani *request*. Dan juga pada Gambar 5.3 diatas adalah tampilan *website* ketika *real server* 3 pada alamat *IP* 10.0.2.13 yang melayani *request.*



Gambar 5.1 Tampilan *web real server* 1



Gambar 5.2 Tampilan *web real server* 2



Gambar 5.3 Tampilan *web real server* 3

Dari pengujian diatas, maka ketiga *real server* mampu memberikan layanan *website* ketika ada *request* yang datang dari *client*. Dan halaman utama *website* tersebut dibedakan *server* mana yang menangani dengan pemberian header "*Login Server* 1" untuk *real server* 1, "*Login Server* 2" untuk *real server* 2, dan "*login Real Server* 3" untuk *real server* 3.

5.3 Pengujian pemberian *request*

Setelah dilakukan pengujian untuk membuktikan bahwa ketiga *real server* mampu melayani *request* dari *client*. Maka pengujian kali ini dilakukan dengan memberikan *request* dalam jumlah koneksi yang berbeda. Pengujian dilakukan pemberian *request* dengan konfigurasi antar *real server* memiliki spesifikasi dan beban yang sama. Pada tiap pengujian dilakukan *stress load* dengan pemberian beban *request* kepada *web load balancing*, untuk setiap skenario dilakukan pengujian pemberian beban *request* kepada *web load balancing* dengan pengiriman *request* sebesar 50 *request*/detik untuk skenario pertama, pada skenario kedua dilakukan pengiriman *request* sebesar 80 *request*/detik, skenario ketiga dilakukan pengiriman *request* sebesar 120 *request*/detik dan skenario keempat pengiriman *request* sebesar 170 *request*/detik. Lalu pada setiap pemberian beban *request* dilakukan pengecekan dan *capture* terhadap hasil *response time*, *throughput*, *request rate* dan pembagian *request* pada ketiga *real server*. Untuk setiap skenario dilakukan pengujian dengan interval pengulangan sebanyak 10 kali, 30 kali, dan 60 kali.

Pada setiap pengujian sebanyak 4 skenario. Proses dan alur pelaksanaan pengujian sesuai dengan tahap yang telah dituliskan pada bab metodologi penelitian dan perancangan. Data yang diambil dari hasil pengujian adalah data untuk pembagian koneksi pada setiap *web server*, *throughput*, *response time*, dan *request rate* yang diperoleh dari data perangkat pengujian.

5.3.1 Perangkat pengujian

Pada pengujian dengan pemberian beban *request*, penulis menggunakan *tool Apache Jmeter*. *Apache Jmeter* adalah perangkat lunak berbasis *Java* yang digunakan sebagai alat pengujian suatu sistem dengan memberikan beban pada fungsional yang konfigurasikan serta untuk mengukur peforma pada suatu *web server*. *Apache Jmeter* akan memberikan simulasi beban kepada sebuah *sever* untuk diukur tingkat kemampuan dan peformanya.

Konfigurasi yang perlu diperhatikan untuk melakukan pengujian *web load balancing* dengan menggunakan *Apache Jmeter* adalah terletak pada 3 buah komponen dari *Apache Jmeter* berupa *Group Thread*, *Controller http sample*, dan *listener* di dalam *http sampler*. *Group thread* adalah taha awal dari konfigurasi dikarenakan *controller* dan *listener* akan berada di dalam sebuah *group thread*. Pada *group thread* akan dikonfigurasikan tentang jumlah *user thread* yang akan dibuat, *ramp up* untuk mengatur waktu pengiriman *group thread*, dan perulangan pengiriman *group thread* yang dilakukan.

Thread bertugas mensimulasikan jumlah user untuk mengakses *web server* yang diuji. Setiap *thread* akan berjalan secara independen ketika menjalankan proses pengujian. Lalu pada *ramp up* adalah pengaturan untuk menentukan waktu eksekusi pada setiap *thread* yang dikirimkan. Sebagai contoh pada *thread* yang dikonfigurasikan dengan jumlah 100 dan pada *ramp up* dikonfigurasikan dengan nilai 10 dan perulangan sebanyak 1 kali, artinya pada setiap detik akan ada 10 *thread* yang akan melakukan pengujian

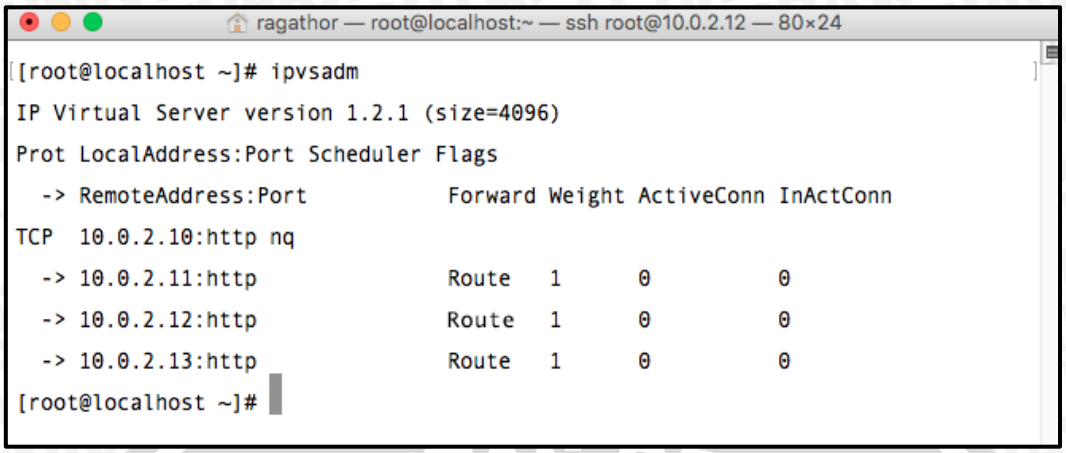
5.3.2 Pengujian skenario 1

Pada pengujian kali ini langkah awal yang dilakukan adalah inisialisasi pada *director* dengan konfigurasi pada *tool IPVSADM*. Pada tabel *routing IPVSADM* dimasukkan *weight* yang sama antara ketiga *real server*. Berikut ini adalah konfigurasi *IPVSADM* untuk pengujian skenario 1 :

Konfigurasi IPVSADM untuk pengujian skenario 1

`#ipvsadm -C`
`#ipvsadm –A -t 10.0.2.10:80 -s nq`
`#ipvsadm –a –t 10.0.2.10:80 –r 10.0.2.11:80 -g -w 1`
`#ipvsadm –a –t 10.0.2.10:80 –r 10.0.2.12:80 -g -w 1`
`#ipvsadm –a –t 10.0.2.10:80 –r 10.0.2.13:80 -g -w 1`

Konfigurasi diatas adalah konfigurasi untuk IPVSADM, pertama dilakukan clear table dari IPVSADM lalu ditambahkan table dengan konfigurasi alamat IP dari load balancer beserta algoritma penjadwalan Never Queue (NQ), lalu penambahan alamat IP untuk real server, pada konfigurasi penambahan real server *–w* 1 artinya setiap *real server* diberikan beban *request* sejumlah 1. Setelah dilakukan konfigurasi pada IPVSADM maka selanjutnya adalah melihat apakah aturan routing yang dikonfigurasikan sudah berjalan dan tersimpan pada table IPVSADM. Berikut adalah tampilan hasil konfigurasi IPVSADM pada gambar 5.4 :



Gambar 5.4 Konfigurasi *IPVSADM* untuk pengujian

Tahap selanjutnya yang dilakukan adalah konfigurasi pada *Apache Jmeter.* Pada *Apache Jmeter* akan dikonfigurasikan dengan memberikan beban *request* sebesar 50 *request*/detik. Pengiriman *request* akan dilakukan pengulangan sebanyak 10, 30, dan 60 kali. Maka dari itu berikut adalah pengaturan dari *group thread* pada *Apache Jmeter* :

- Jumlah *Thread* = 50
- *Ramp Up* (dalam detik) = 1
- Jumlah perulangan = 10

Dengan pengaturan diatas maka *Apache Jmeter* akan mensimulasikan untuk mengirimkan 50 *user* yang melakukan *request http* sebanyak 10 dengan interval 50 *request*/detik yang artinya pada waktu 10 detik akan dikirimkan *request* sebanyak 500 *request*. Lalu untuk pengujian dengan tingkat perulangan yang berbeda maka akan dikonfigurasikan lagi untuk *group thread* pada bagian jumlah perulangan yang akan dilakukan sebanyak 10 kali, 30 kali, dan 60 kali.

Tabel 5.3 Jumlah koneksi setiap *server* pada pengujian skenario 1

Perulangan	Jumlah Koneksi		
	Server 1	*Server* 2	*Server* 3
10	173	164	163
30	499	489	512
60	979	1005	1016

Tabel 5.4 *Throughput*, *response time*, dan *request rate* pengujian skenario 1

	Perulangan		
	10	30	60
Throughput (KB/s)	197,86	215,62	237,02
Response Time (ms)	1285	1200	1126
Request Rate	36,3	39.6	43,5

(req/sec)			

5.3.3 Pengujian skenario 2

Pada pengujian skenario 2 ini, tahap awal yang perlu dilakukan adalah sama seperti pada pengujian skenario 1 yaitu dilakukan inisialisasi pada *director* dengan konfigurasi pada *tool IPVSADM*. Pada tabel *routing IPVSADM* dimasukkan *weight* yang sama antara ketiga *real server*.

Tahap selanjutnya yang dilakukan adalah konfigurasi pada *Apache Jmeter.* Pada *Apache Jmeter* akan dikonfigurasikan dengan memberikan beban *request* sebesar 80 *request*/detik. Pengiriman *request* akan dilakukan pengulangan sebanyak 10, 30, dan 60 kali. Maka dari itu berikut adalah pengaturan dari *group thread* pada *Apache Jmeter* :

- Jumlah *Thread* = 80
- *Ramp U*p (dalam detik) = 1
- Jumlah perulangan = 10

Dengan pengaturan diatas maka *Apache Jmeter* akan mensimulasikan untuk mengirimkan 80 *user* yang melakukan *request http* sebanyak 10 dengan interval 80 *request*/detik yang artinya pada waktu 10 detik akan dikirimkan *request* sebanyak 800 *request*. Lalu untuk pengujian dengan tingkat perulangan yang berbeda maka akan dikonfigurasikan lagi untuk *group thread* pada bagian jumlah perulangan yang akan dilakukan sebanyak 10 kali, 30 kali, dan 60 kali.

Tabel 5.5 Jumlah koneksi setiap *server* pada pengujian skenario 2

Perulangan	Jumlah Koneksi		
	Server 1	*Server* 2	*Server* 3
10	266	276	258
30	796	794	810
60	1572	1569	1659

Tabel 5.6 *Throughput*, *response time*, dan *request rate* pengujian skenario 2

	Perulangan		
	10	30	60
Throughput (*KB/s*)	208,35	231,22	227,46
Response Time (*ms*)	1942	1824	1877

Request Rate (*req/sec*)	38,3	42,5	41,8

5.3.4 Pengujian skenario 3

Selanjutnya pada pengujian skenario 3, tahap awal yang perlu dilakukan adalah sama seperti pada skenario pengujian sebelumnya yaitu dilakukan inisialisasi pada *director* dengan konfigurasi pada *tool IPVSADM*. Pada tabel *routing IPVSADM* dimasukkan *weight* yang sama antara ketiga *real server*.

Tahap selanjutnya yang dilakukan adalah konfigurasi pada *Apache Jmeter.* Pada *Apache Jmeter* akan dikonfigurasikan dengan memberikan beban *request* sebesar 120 *request*/detik. Pengiriman *request* akan dilakukan pengulangan sebanyak 10, 30, dan 60 kali. Maka dari itu berikut adalah pengaturan dari *group thread* pada *Apache Jmeter* :

- Jumlah *Thread* = 120
- *Ramp U*p (dalam detik) = 1
- Jumlah perulangan = 10

Dengan pengaturan diatas maka *Apache Jmeter* akan mensimulasikan untuk mengirimkan 120 *user* yang melakukan *request http* sebanyak 10 dengan interval 120 *request*/detik yang artinya pada waktu 10 detik akan dikirimkan *request* sebanyak 1200 *request*. Lalu untuk pengujian dengan tingkat perulangan yang berbeda maka akan dikonfigurasikan lagi untuk *group thread* pada bagian jumlah perulangan yang akan dilakukan sebanyak 10 kali, 30 kali, dan 60 kali.

Tabel 5.7 Jumlah koneksi setiap *server* pada pengujian skenario 3

Perulangan	Jumlah Koneksi		
	Server 1	*Server* 2	*Server* 3
10	441	374	385
30	1195	1190	1215
60	2385	2405	2410

Tabel 5.8 *Throughput*, *response time*, dan *request rate* pengujian skenario 3

	Perulangan		
	10	30	60
Throughput (*KB/s*)	205,8	210,99	201,68
Response Time (*ms*)	2951	2965	3076

Request Rate (*req/sec*)	37,8	37,4	36,4

5.3.5 Pengujian skenario 4

Dan yang terakhir pada pengujian skenario 4 ini, tahap awal yang perlu dilakukan adalah sama seperti pada pengujian skenario 1, 2, dan 3 yaitu dilakukan inisialisasi pada *director* dengan konfigurasi pada *tool IPVSADM*. Pada tabel *routing IPVSADM* dimasukkan *weight* yang sama antara ketiga *real server*.

Tahap selanjutnya yang dilakukan adalah konfigurasi pada *Apache Jmeter*. Pada *Apache Jmeter* akan dikonfigurasikan dengan memberikan beban *request* sebesar 170 *request*/detik. Pengiriman *request* akan dilakukan pengulangan sebanyak 10, 30, dan 60 kali. Maka dari itu berikut adalah pengaturan dari *group thread* pada *Apache Jmeter* :

- Jumlah *Thread* = 170
- *Ramp U*p (dalam detik) = 1
- Jumlah perulangan = 10

Dengan pengaturan diatas maka *Apache Jmeter* akan mensimulasikan untuk mengirimkan 80 *user* yang melakukan *request http* sebanyak 10 dengan interval 170 *request*/detik yang artinya pada waktu 10 detik akan dikirimkan *request* sebanyak 1700 *request*. Lalu untuk pengujian dengan tingkat perulangan yang berbeda maka akan dikonfigurasikan lagi untuk *group thread* pada bagian jumlah perulangan yang akan dilakukan sebanyak 10 kali, 30 kali, dan 60 kali.

Tabel 5.9 Jumlah koneksi setiap *server* pada pengujian skenario 4

Perulangan	Jumlah Koneksi		
	Server 1	*Server* 2	*Server* 3
10	574	561	565
30	1696	1691	1713
60	3390	3350	3460

Tabel 5.10 *Throughput*, *response time*, dan *request rate* pengujian skenario 4

	Perulangan		
	10	30	60
Throughput (*KB/s*)	215,49	220,51	198,86
Response Time (*ms*)	4301	4125	4539

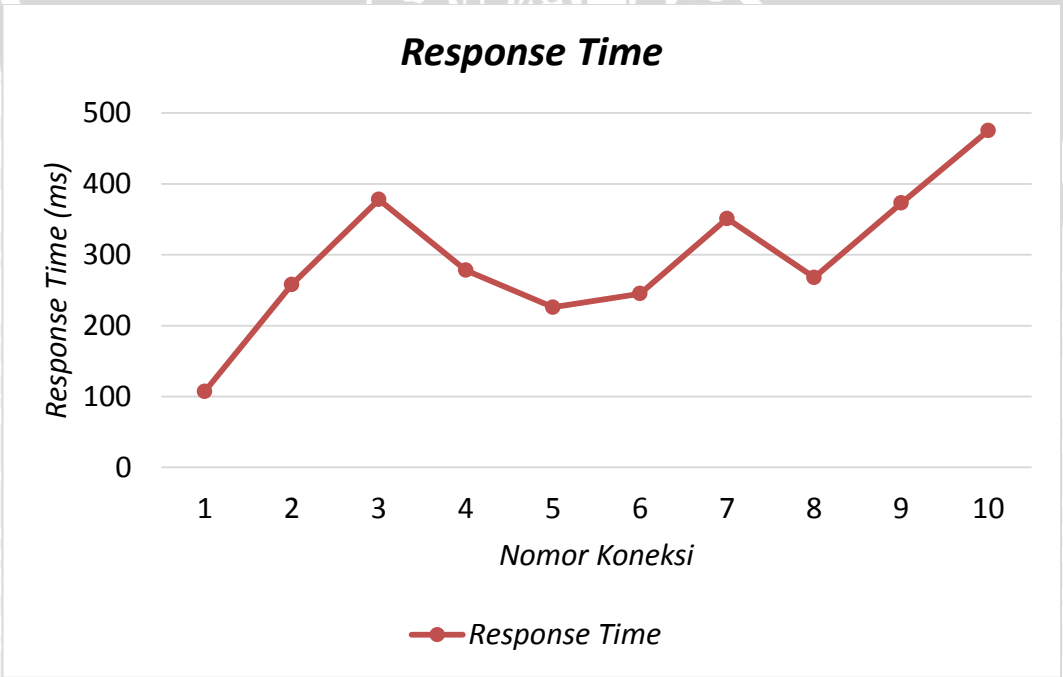
Request Rate (*req/sec*)	37,1	37,8	36,2

5.4 Analisa

Dari hasil pengujian yang dilakukan pada *web load balancing* dengan urutan pengujian yang pertama dalah pengujian untuk layanan *website* pada *real server* dan pengujian yang kedua dengan memberikan beban *request* kepada *web load balancing*. *Real server* mampu memberikan layanan *website* kepada *client* dengan menggunakan alamat *IP director* pada halaman utama sesuai dengan *gateway* yang dikonfigurasikan pada setiap *real server* mengarah pada *director*, lalu setelah login alamat *IP* akan mengarah ke alamat *IP* masing-masing *real server* karena pada konsep *direct routing* mengatur setiap *server* mampu langsung berkomunikasi dengan *client* setelah terjadi pembagian koneksi pada *director*. Lalu pada pengujian dengan pemberian beban *request*, sistem *web load balancing* mampu membagi *request* dari *client* yang dikirim ke *director* untuk diteruskan ke ketiga *real server* sesuai dengan algoritma penjadwalan *never queue*.

5.4.1 Analisa pengujian implementasi algoritma *Never Queue* (*NQ*)

Pada pengujian untuk mengetahui cara kerja dari algoritma *never queue* dilakukan dengan memberikan koneksi sebesar 10 *request*/detik. Diperoleh data pembagian koneksi untuk setiap *server* berupa 5 koneksi mengarah ke *server* 1, 3 koneksi mengarah ke *server* 2 dan 2 koneksi mengarah pada *server* 3. Pada hasil pengujian diperoleh data waktu respon dari setiap koneksi adalah pada gambar 5.5 berikut :



Gambar 5.5 Waktu respon pengujian proses kerja algoritma *Never Queue* (*NQ*)

Dari hasil data waktu pengiriman koneksi dan *response time* untuk setiap koneksi yang diperoleh pada pengujian dengan pengiriman 10 *request* ini, setiap koneksi dapat dianalisa untuk penentuan pembagian koneksi kepada ketiga *real server* pada sistem *web load balancing*. Proses penentuan pemilihan *server* pada setiap koneksi adalah sebagai berikut :

1. Koneksi 1, pengiriman pada waktu 20:34:40.144

   Pada waktu pengiriman *request* pertama, seluruh *server* masih dalam keaadan *idle*. Artinya koneksi akan diarahkan pada *server* 1 yang merupakan *server* dengan urutan yang pertama dengan waktu respon dari *director* sebesar 107 *ms* yang membuat *server* 1 tidak lagi dalam keadaan *idle* pada waktu pengiriman ditambahkan dengan waktu respon yang diperoleh, maka sehingga diperoleh kondisi setiap *server* sebagai berikut :

   - *Server* 1 akan *idle* pada waktu 20:34:40.251
   - *Server* 2 berstatus *idle*
   - *Server* 3 berstatus *idle*

2. Koneksi 2, pengiriman pada waktu 20:34:40.226

   Saat pengiriman koneksi yang kedua, diperoleh status dari setiap *server* adalah *server* 1 akan *idle* pada waktu 20:34:40.251 (masih belum *idle*), *server* 2 berstatus *idle* dan *server* 3 berstatus *idle*. Artinya prioritas pengiriman koneksi akan diarahkan pada *server* yang masih *idle* yaitu pada *server* 2 yang berada pada urutan setelah *server* 1 pada hasil konfigurasi. Dengan waktu respon sebesar 258 *ms*, maka kondisi untuk setiap *server* adalah sebagai berikut:

   - *Server* 1 akan *idle* pada waktu 20:34:40.251 (masih belum *idle*)
   - *Server* 2 akan *idle* pada waktu 20:34:40.484
   - Server 3 berstatus *idle*

3. Koneksi 3, pengiriman pada waktu 20:34:40.325

   Pada waktu pengiriman koneksi ketiga, *server* 1 sudah dalam keadaan *idle* setelah memproses koneksi pertama, pada *server* 2 masih dalam keadaan memproses koneksi kedua, dan *server* 3 dalam keadaan *idle*. Maka koneksi akan diarahkan ke *server* dalam keadaan *idle* yaitu antara *server* 1 dan *server* 3, dalam algoritma *never queue* kedua server ini dianggap statusnya sama meskipun *server* 1 telah memproses 1 koneksi karena konsep pertama dari algoritma *never queue* hanya mencari *server* yang *idle*. Sehingga koneksi ketiga akan diarahkan ke *server* 1 dengan waktu respon sebesar 378 *ms* yang artinya untuk setiap server memiliki status sebagai berikut :

   - *Server* 1 akan *idle* pada waktu 20:34:40.703
   - *Server* 2 akan *idle* pada waktu 20:34:40.484 (masih belum idle)
   - *Server* 3 berstatus *idle*

4. Koneksi 4 pengiriman pada waktu 20:34:40.426

   Untuk koneksi keempat saat waktu pengiriman ternyata *server* 1 belum dalam keadaan *idle* saat memproses koneksi ketiga, *server* 2 masih dalam keadaan memproses koneksi kedua, dan *server* 3 dalam keadaan *idle*. Maka koneksi akan langsung diarahkan ke *server* 3 yang masih dalam keadaan *idle*.

Pada pemrosesan koneksi keempat diperoleh waktu respon sebesar 278 *ms*, sehingga diperoleh status pada setiap *server* adalah sebagai berikut :

- *Server* 1 akan *idle* pada waktu 20:34:40.703 (masih belum idle)
- *Server* 2 akan *idle* pada waktu 20:34:40.484 (masih belum idle)
- *Server* 3 akan *idle* pada waktu 20:34:40.704

5. Koneksi 5, pengiriman pada waktu 20:34:40.530

   Untuk koneksi kelima, ternyata *server* 2 sudah dalam keadaan *idle* kembali pada waktu pengiriman koneksi kelima setelah memproses dari koneksi kedua. Sehingga koneksi akan tetap diarahkan ke *server* 2 yang sudah dalam keadaan *idle* dengan waktu respon sebesar 226 *ms*, maka diperoleh untuk status setiap *server* adalah sebagai berikut :

   - *Server* 1 akan *idle* pada waktu 20:34:40.703 (masih belum *idle*)
   - *Server* 2 akan *idle* pada waktu 20:34:40.756
   - *Server* 3 akan *idle* pada waktu 20:34:40.704 (masih belum *idle*)

6. Koneksi 6, pengiriman pada waktu 20:34:40.627

   Pada waktu pengiriman koneksi keenam, pada ketiga *server* masih dalam keadaan memproses koneksi ketiga, keempat, dan kelima. Artinya setiap *server* masih memproses 1 koneksi yang datang, karena tidak ada *server* yang sedang *idle* maka penentuan koneksi dihitung dari harapan *delay* terpendek dengan menggunakan algoritma *Shortest Expected Delay* pada setiap *server*:

   - *Server* 1
     C_1 = 1 koneksi
     U_1 = 1
     SED = $(C_1 + 1) / U_2 = (1 + 1) / 1 = 2$
   - *Server* 2
     C_2 = 1 koneksi
     U_2 = 1
     SED = $(C_2 + 1) / U_2 = (1 + 1) / 1 = 2$
   - *Server* 3
     C_2 = 1 koneksi
     U_2 = 1
     SED = $(C_2 + 1) / U_2 = (1 + 1) / 1 = 2$

   Dari hasil penghitungan harapan *delay* terpendek dengan menggunakan algoritma *Shortest Expected Delay* diperoleh nilai *delay* untuk setiap *server* sebesar 2 karena seluruh *server* dalam keadaan memproses 1 koneksi. Sehingga koneksi keenam diarahkan ke *server* 1 yang berada pada prioritas urutan pertama saat konfigurasi. Dalam memproses koneksi keenam diperoleh waktu respon sebesar 245 *ms*, sehigga kondisi yang diperoleh untuk setiap *server* adalah sebagai berikut :

   - *Server* 1 akan *idle* pada waktu 20:34:40.872
   - *Server* 2 akan *idle* pada waktu 20:34:40.756 (masih belum *idle*)
   - *Server* 3 akan *idle* pada waktu 20:34:40.704 (masih belum *idle*)

7. Koneksi 7, pengiriman pada waktu 20:34:40.734

   Pada saat pengiriman koneksi ketujuh, ternyata *server* 3 sudah dalam kondisi *idle* dan server 1 serta server 3 masih memproses koneksi. Maka dari

itu koneksi ketujuh diarahkan pada server 3. Dalam memproses koneksi ketujuh diperoleh waktu respon sebesar 351 *ms*, sehigga kondisi yang diperoleh untuk setiap *server* adalah sebagai berikut :

- *Server* 1 akan *idle* pada waktu 20:34:40.872 (masih belum *idle*)
- *Server* 2 akan *idle* pada waktu 20:34:40.756 (masih belum *idle*)
- *Server* 3 akan *idle* pada waktu 20:34:41.085

8. Koneksi 8, pengiriman pada waktu 20:34:41.832

   Waktu pengiriman koneksi kedelapan, ternyata *server* 2 sudah dalam keadaan *idle* kembali pada waktu pengiriman koneksi kedelapan setelah memproses dari koneksi kelima. Sehingga koneksi akan tetap diarahkan ke *server* 2 yang sudah dalam keadaan *idle* dengan waktu respon sebesar 268 *ms*, maka diperoleh untuk status setiap *server* adalah sebagai berikut :

   - *Server* 1 akan *idle* pada waktu 20:34:40.872 (masih belum *idle*)
   - *Server* 2 akan *idle* pada waktu 20:34:41.100
   - *Server* 3 akan *idle* pada waktu 20:34:41.085 (masih belum *idle*)

9. Koneksi 9 pengiriman pada waktu 20:34:40.932

   Untuk waktu pengiriman koneksi kesembilan diperoleh data bahwa *server* 1 sudah dalam keadaan *idle* dan *server* 2 serta *server* 3 masih bekerja memproses koneksi. Sehingga secara langsung koneksi diarahkan ke *server* 1 dengan waktu respon sebesar 373 *ms*, dan diperoleh status setiap *server* sebagai berikut:

   - *Server* 1 akan *idle* pada waktu 20:34:41.305
   - *Server* 2 akan *idle* pada waktu 20:34:41.100 (masih belum *idle*)
   - *Server* 3 akan *idle* pada waktu 20:34:41.085 (masih belum *idle*)

10. Koneksi 10 pengiriman pada waktu 20:34:40.037

    Saat pengiriman koneksi kesepuluh, ternyata diantara ketiga *server* tidak dalam keadaan *idle*. Pada *server* 1 masih memproses 1 koneksi yaitu koneksi kesembilan, untuk *server* 2 masih memproses koneksi kedelapan dan *server* 3 sedang memproses koneksi ketujuh. Sehingga dilakukan penghitungan harapan *delay* terpendek sebagai berikut :

    - *Server* 1
      C_1 = 2 koneksi
      U_1 = 1
      SED = $(C_1 + 1) / U_2 = (1 + 1) / 1 = 2$
    - *Server* 2
      C_2 = 1 koneksi
      U_2 = 1
      SED = $(C_2 + 1) / U_2 = (1 + 1) / 1 = 2$
    - *Server* 3
      C_2 = 1 koneksi
      U_2 = 1
      SED = $(C_2 + 1) / U_2 = (1 + 1) / 1 = 2$

    Dari hasil penghitungan dengan algoritma *Shortest Expected Delay* (*SED*) diperoleh nilai harapan *delay* untuk setiap *server* yaitu pada *server* 1 memiliki nilai harapan *delay* sebesar 2, dan untuk *server* 2 serta *server* 3 memiliki nilai

harapan *delay* sebesar 2. Artinya koneksi akan diarahkan ke *server* 1 dengan nilai harapan *delay* terkecil dan dikarenakan *server* 1 berada pada prioritas konfigurasi yang lebih dulu dari pada *server* 2 dan *server* 3.

Dari hasil analisa proses kerja dari algoritma penjadwalan *never queue*, diperoleh data pembagian untuk setiap koneksi adalah pada table 5.11 berikut :

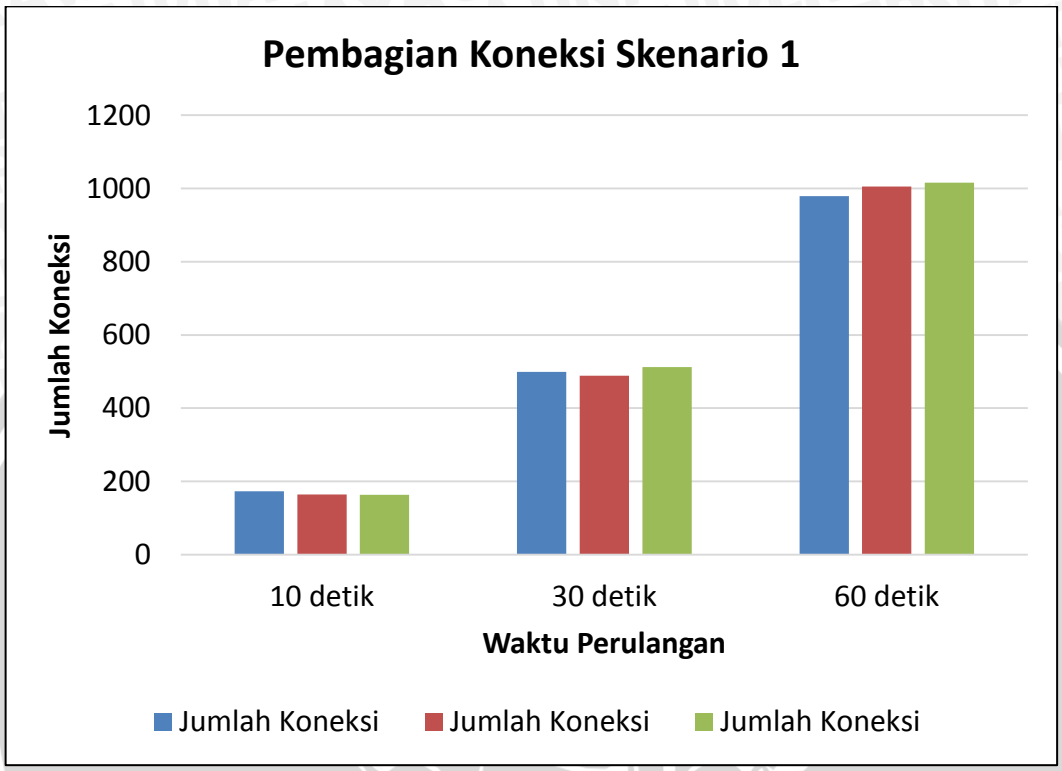
Tabel 5.11 Pembagian koneksi pada setiap *server*

Nomor Koneksi	Diarahkan pada *server*
1	*Server* 1
2	*Server* 2
3	*Server* 1
4	*Server* 3
5	*Server* 2
6	*Server* 1
7	*Server* 3
8	*Server* 2
9	*Server* 1
10	*Server* 1

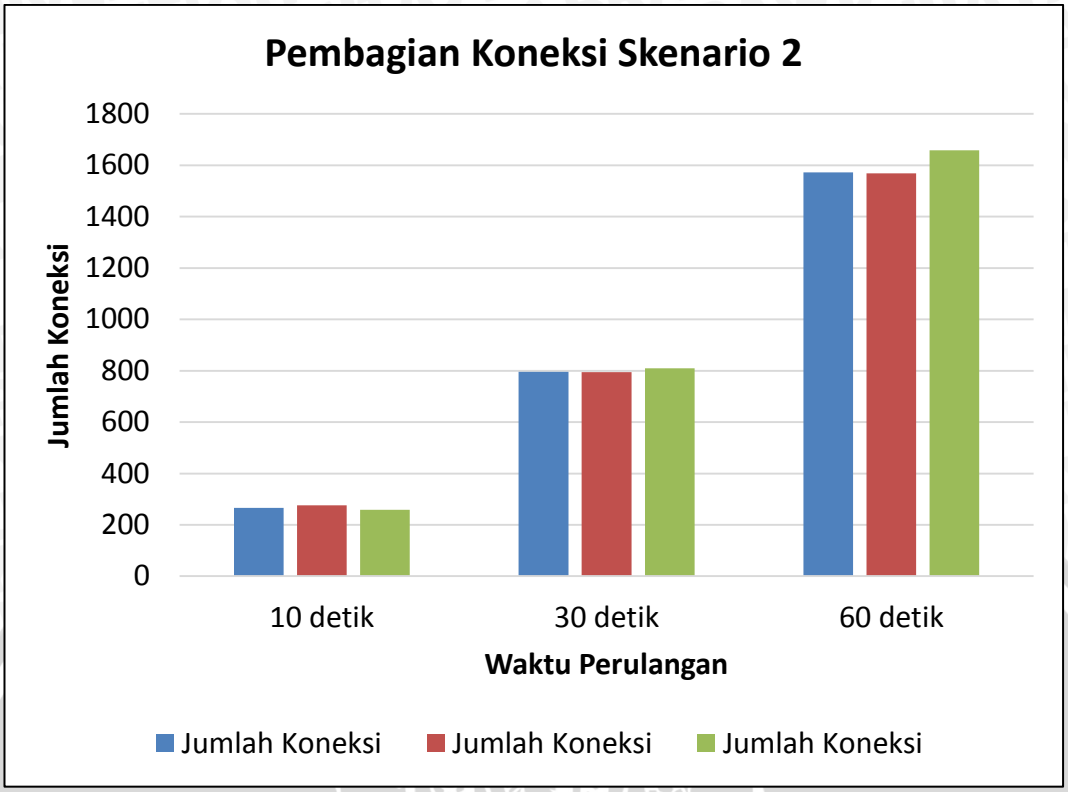
5.4.2 Analisa pengujian pemberian *request*

Pada pengujian dengan pemberian beban *request*, tiap skenario dilakukan pengujian *request* dengan jumlah *request* yang berbeda. Dari hasil pengujian terlihat algoritma penjadwalan *never queue* mampu membagi *request* kepada ketiga *server* dengan seimbang. Algoritma *never queue* membagi permintaan dari pengguna secara dinamis kepada setiap *server*, karena *never queue* akan mencari server manakah yang berstatus *idle* terlebih dahulu, jika tidak ada *server* yang *idle* maka akan dipilih server yang memiliki harapan delay koneksi terpendek dengan algoritma *Shortest Expected Delay.* Pada pengujian skenario kesatu, pembagian *request* terlihat seimbang untuk setiap *server* pada waktu perulangan 10, 30, dan 60 kali. Namun pada pengujian skenario kedua, pada waktu perulangan 60 kali, tidak seimbang pembagian *request* pada setiap server, dan pada perulangan 10 dan 30 kali pembagian *request* masih seimbang. Selanjutnya untuk pengujian skenario ketiga, pada perulangan 10 kali terjadi ketidakseimbangan pembagian *request*, namun ketika pada pengujian dengan perulangan 30 dan 60 pembagian *request* menjadi seimbang. Dan yang terakhir pada pengujian skenario keempat terlihat terjadi keseimbangan pembagian

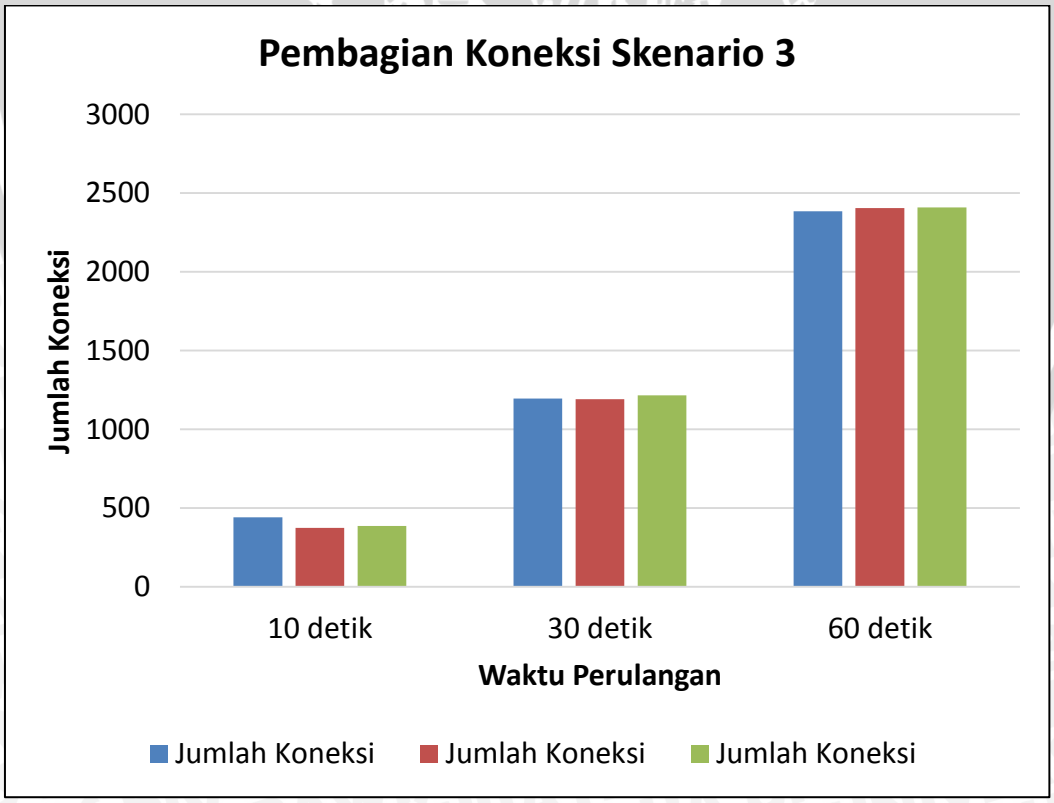
request pada setiap perulangan pengujian. Berikut pada gambar adalah grafik pembagian *request* pada setiap skenario pengujian :



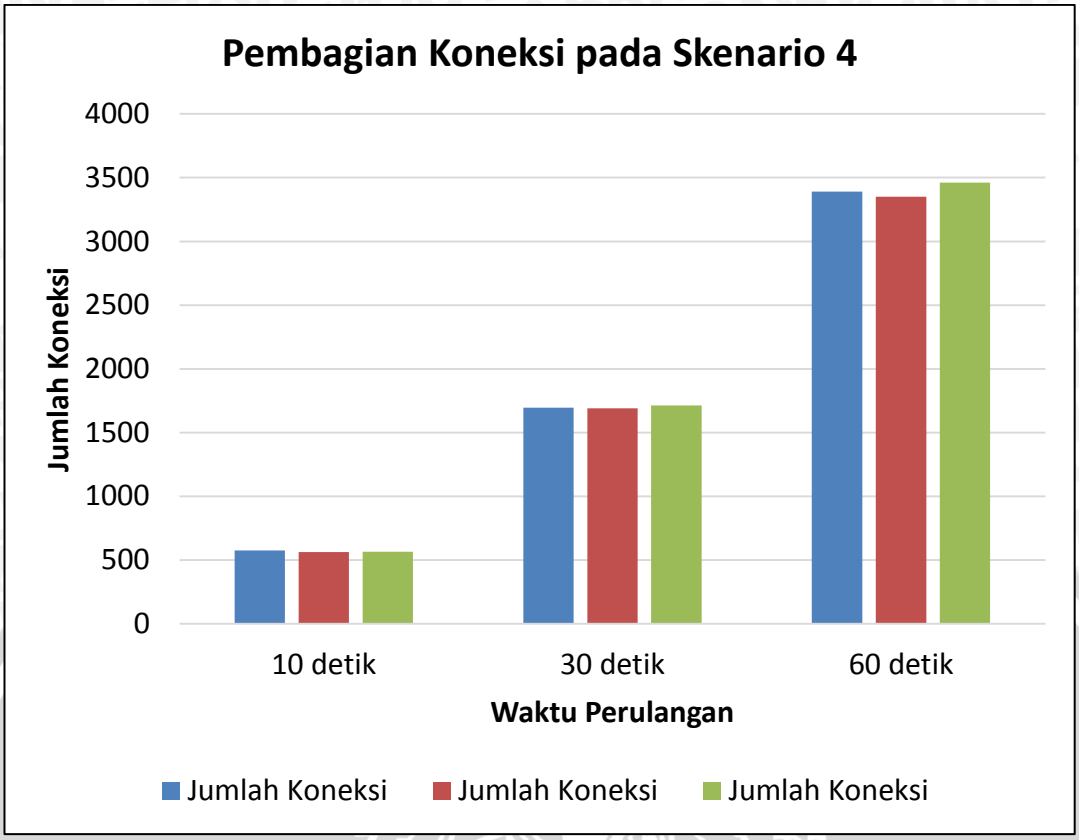
Gambar 5.6 Grafik pembagian *request* pada skenario 1



Gambar 5.7 Grafik pembagian *request* pada skenario 2

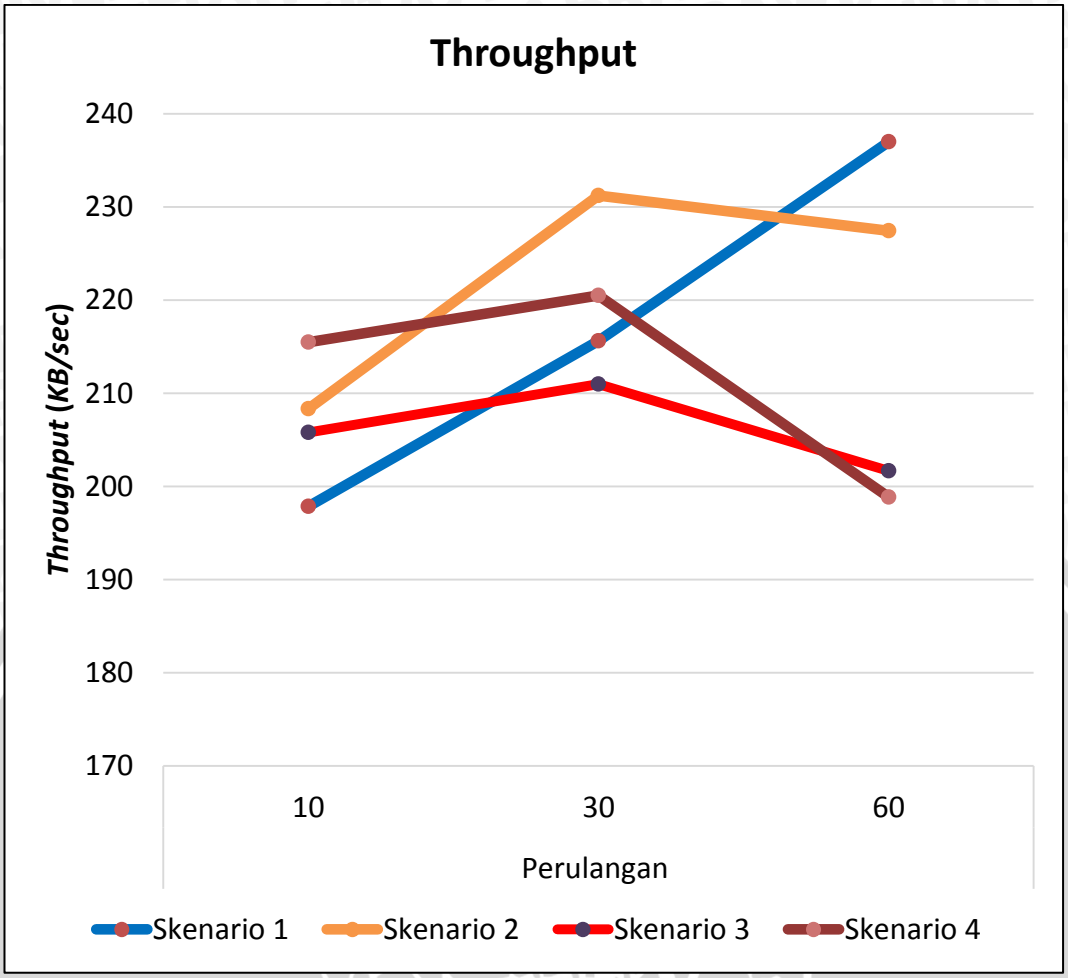


Gambar 5.8 Grafik pembagian *request* pada skenario 3



Gambar 5.9 Grafik pembagian *request* pada skenario 4

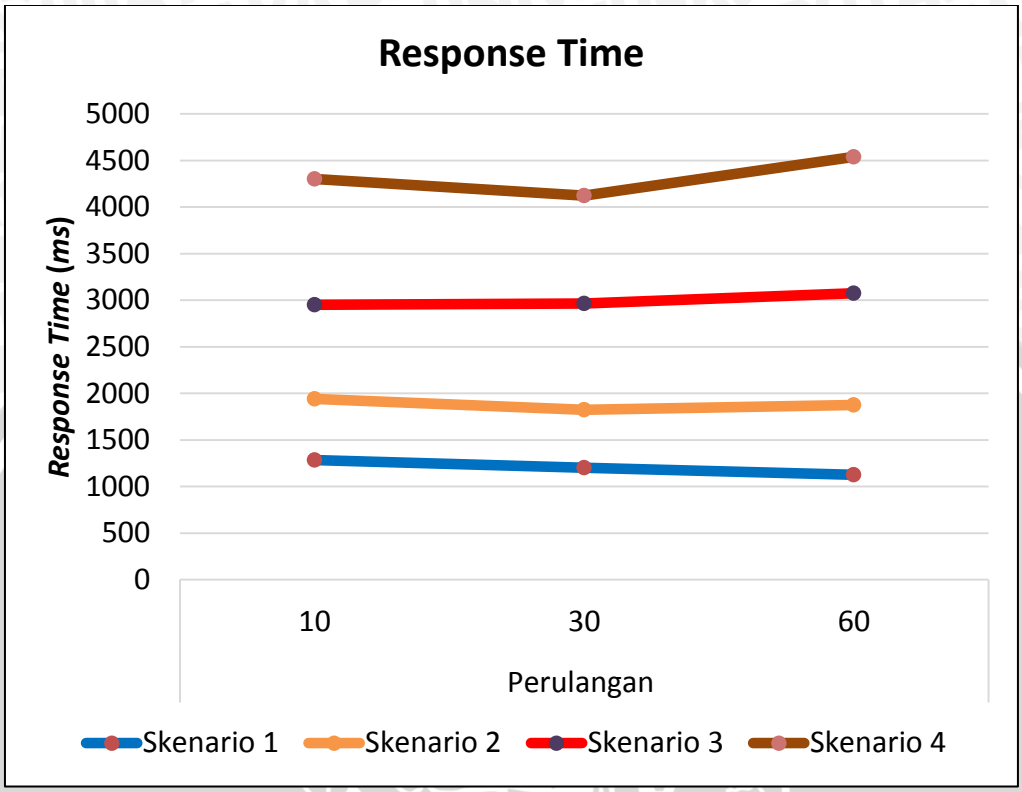
Selanjutnya pada gambar 5.10 adalah grafik dari *throughput* pada setiap skenario pengujian. Terlihat pada skenario pertama terjadi kenaikan *throughput* pada setiap perulangannya dengan 197,86 *KB/sec*, 215,62 *KB/sec*, dan 237,02 *KB/sec* pada perulangan 10, 30, dan 60 kali. lalu pada skenario kedua terjadi kenaikkan *throughput* yang cukup signifikan pada pengujian dengan perulangan 30 kali yaitu dari awalnya 208,35 *KB/sec* pada perulangan 10 kali, lalu menjadi sebanyak 231,22 *KB/sec* pada perulangan 30 kali, Namun pada perulangan 60 kali terjadi penurunan *throughput* dengan 227,46 *KB/sec*. Lalu pada pengujian skenario ketiga, pengujian dengan perulangan 10 kali mendapat *throughput* sebesar 205,8 *KB/sec*, pada perulangan 30 kali terjadi peningkatan *throughput* sebesar 210,99 *KB/sec*, dan pada perulangan 60 kali justru terjadi penurunan *throughput* dengan jumlah 201,68 *KB/sec*. Dan pada pengujian skenario keempat, terjadi kenaikakan *throughput* antara pengujian dengan perulangan 10 dan 30 kali. Pada perulangan 10 kali *throughput* yang dihasilkan sebesar 215,49 *KB/sec*, dan pada perulangan 30 kali sebesar 220,51 *KB/sec*. Namun terjadi penurunan *throughput* secara signifikan pada perulangan 60 kali dengan *throughput* yang dihasilkan sebesar 198,86 *KB/sec*.



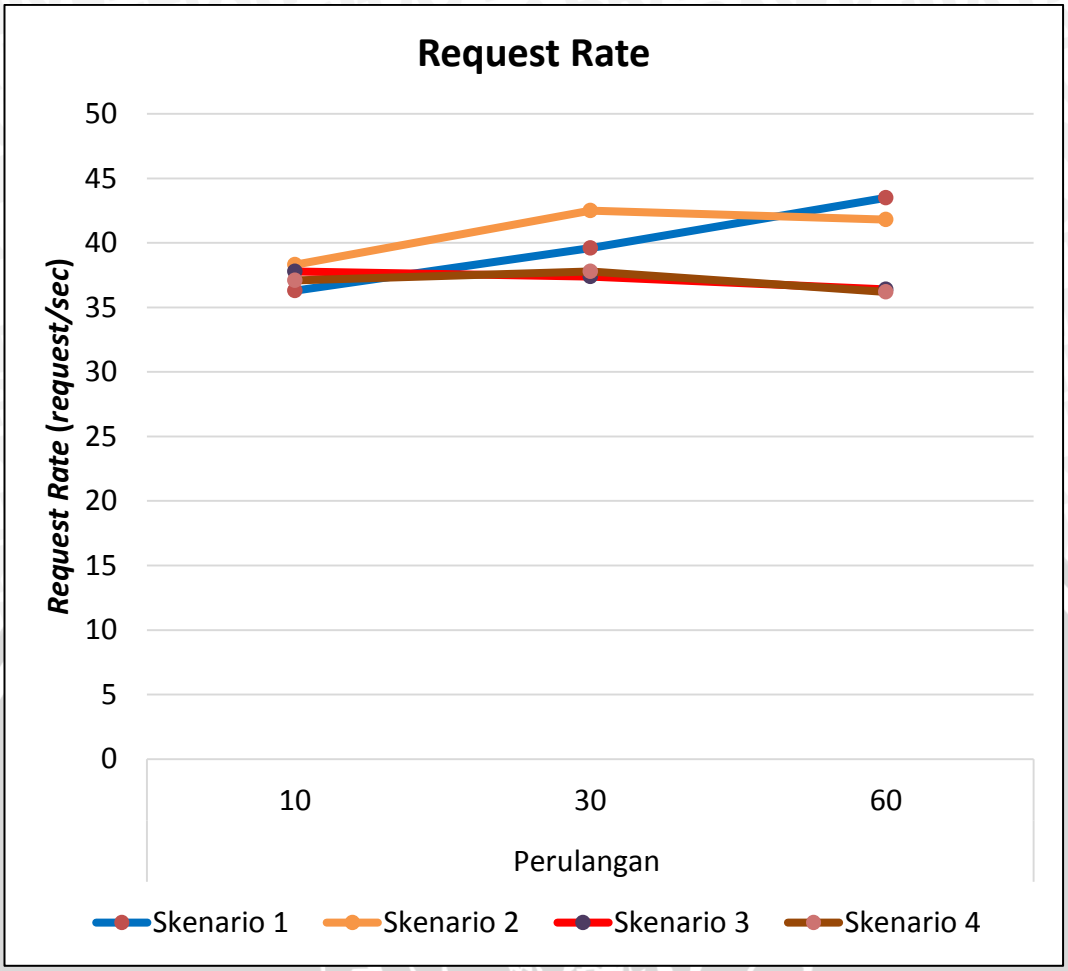
Gambar 5.10 *Throughput* pada setiap skenario pengujian

Pada gambar 5.11 dan 5.12 adalah grafik rata-rata waktu respon dan jumlah request yang mampu diproses pada setiap detiknya untuk setiap skenario pengujian. Pada pengujian skenario satu waktu respon yang dihasilkan stabil dan cenderung semakin baik mulai dari pengujian dengan perulanga 10 kali dengan waktu respon sebesar 1285 *ms* hingga pada pengujian dengan perulangan 60 kali diperoleh waktu respon sebesar 1126 *ms* sehingga mempengaruhi *request rate* yang semakin meningkat mencapai 43,5 *request/sec*. Pada skenario kedua, rata-rata waktu respon yang diperoleh sebesar 1942 *ms/request* untuk *request rate* 38.3 *request/sec* pada perulangan 10 kali, terjadi peningkatan waktu respon pada perulangan 30 kali sebesar 1824 *ms/request* dengan *request rate* 42,5 *request/sec,* dan 1877 *ms/request* dengan *request rate* 41,8 *request/sec* pada perulangan 60 kali. Untuk pengujian skenario ketiga, waktu respon yang dihasilkan sebesar 2951 *ms/request* dengan *request rate* sebesar 37,8 *request/sec* pada perulangan 10 kali, pada perulangan 30 dan 60 kali terjadi peningkatan waktu respon sebesar 2965 *ms/request* dengan request rate sebesar 37,4 *request/sec* dan 3076 *ms/request* dengan 36,4 *request/sec*. Selanjutnya pada skenario pengujian keempat, untuk pengujian dengan perulangan 10 kali diperoleh waktu respon sebesar 4301 *ms/request* dengan *request rate* sebesar 37,1 *request/sec*, terjadi percepatan waktu respon pada

pengujian dengan perulangan 30 kali yaitu sebesar 4125 *ms*/*request* dengan *request rate* sebesar 37,8 *request*/*sec*, namun pada pengujian dengan perulangan 60 kali terjadi peningkatan waktu respon yang signifikan sebanyak 4539 *ms*/*request* dengan *request rate* sebesar 36,2 *request*/*sec.*



Gambar 5.11 Grafik *response time* pada setiap skenario pengujian



Gambar 5.12 Grafik *request rate* pada setiap skenario pengujian

BAB 6 PENUTUP

Bab 6 berisi kesimpulan dari hasil perancangan, implementasi, dan pengujian dari sistem *load balancing* yang menjadi topik penelitian ini serta saran untuk pengembangan penelitian selanjutnya.

6.1 Kesimpulan

Dari hasil data pengujian dan analisa dari sistem *web load balancing* via *LVS direct routing* dengan menggunakan algoritma penjadwalan *never queue* disimpulkan bahwa :

1. Waktu respon dari setiap *real server* sangat mempengaruhi algoritma penjadwalan *Never Queue* (*NQ*) dalam mendistribusikan beban *request* yang datang, dari hasil pengamatan pengujian pemberian beban pada skenario keempat diperoleh data pada perulangan 10 dan 30 kali dengan hasil rata-rata waktu respon yang cenderung membaik sehingga distribusi *request* terbagi secara merata pada setiap *server*, namun pada perulangan 60 kali terjadi peningkatan waktu respon yang signifikan sebesar 4539 ms sehingga berpengaruh pada pendistribusian beban *request* menjadi tidak seimbang pada setiap *server*.
2. Sistem *web load balancing* dengan algoritma penjadwalan *never queue* (*NQ*) dari hasil implementasi mampu bekerja dengan baik pada hasil pengujian dengan perulangan 30 kali untuk setiap skenario pengujian dengan interval 50 *request*/, 80 *request*/detik, 120 *request*/detik, dan 170 *request*/detik. Pada pengujian dengan perulangan 10 kali, *never queue* bekerja lebih baik pada permintaan koneksi sejumlah 170 *request*/detik. Untuk pengujian dengan perulangan 60 kali, algoritma *never queue* mampu bekerja dengan baik pada jumlah permintaan koneksi 50 *request*/detik dan 120 *request*/detik.
3. Hasil pengujian pada implementasi sistem *web load balancing, web load balancing* via *LVS-DR* dengan algoritma penjadwalan *Never Queue* (*NQ*) mampu memproses *request* secara optimal pada pengujian dengan 50 *request*/detik dengan *throughput, response time,* dan *request rate* yang terus meningkat pada setiap perulangannya mencapai 237,02 *KB/s* untuk *throughput* yang dihasilkan, rata-rata waktu respon sejumlah 1126 *ms* dan *request rate* yang mampu di proses sebesar 43,5 *request*/detik pada pengujian dengan perulangan 60 kali.

6.2 Saran

Saran yang dapat disampaikan penulis dari hasil penelitian ini untuk pengembangan penelitian selanjutnya adalah :

1. Perlunya dilakukan penelitian terhadap algoritma penjadwalan lainnya yang untuk sistem *web load balancing* sehingga dapat diketahui karakteristik dan alur pada setiap algoritma penjadwalan yang digunakan pada sistem *web load balancing*.

2. Dalam membagi beban kerja sistem *load balancing* dengan algoritma *Never Queue* (*NQ*) dari hasil implementasi tidak memperhatikan penggunaan sumber daya dan kendala yang terjadi pada *real server* sehingga yang perlu dilakukan untuk penelitian selanjutnya yaitu penerapan konsep untuk *high avaibility* pada sistem *web load balancing* dan juga untuk manajemen *resource*, antara sumber daya dan kendala pada setiap *real server* jika terjadi trafik yang tinggi ataupun terjadi *down*.

DAFTAR PUSTAKA

Apache Fondatioan Sofware.2014.*User Manuals Apache Jmeter*. [online] Apache Fondatioan Sofware. Tersedia di: <http://jmeter.apache.org/usermanual/intro.html> [Diakses 22 September 2015]

Baskoro,A.,2015.[Tools] Load Balancing Server dengan Linux Virtual Server.[online].tersedia di: <http://baskoroadi.web.id/2013/12/tools-load-balancing-server-dengan-linux-virtual-server> [Diakses 10 November 2015]

Daqiqil, I.,2011.Framework Codeigniter Sebuah Pnduan dan Best Practice.[pdf]. Tersedia di:<http://www.koder.web.id> [Diakses 5 September 2015]

Dwi, E.K.,2012.Clustering Computer Server.[pdf] IlmuKomputer.com. Tersedia di:<http://ilmukomputer.org/wp-content/uploads/2012/11/endi-clustering.pdf> [Diakses 1 September 2015]

GNU Free Doc License,2006.*Never Queue Scheduling.*[online] Linux Virtual Server. Tersedia di: <http://kb.linuxvirtualserver.org/wiki/Never_Queue_Scheduling> [Diakses 2 September 2015]

Haris, A.N.,2011.Komparasi Algoritma Penjadwalan pada Layanan Terdistribusi Load Balancing LVS via NAT.S1.Institut Teknologi Sepuluh Nopember. Tersedia di <http://core.ac.uk/download/pdf/12343156.pdf> [Diakses 12 Agustus 2015]

Kurniawan, Y.,2013.Analisis Kinerja Algoritma Load Balancer dan Implementasi pada Layanan Web.S1.Universitas Brawijaya.

Solichin, A.,2010.MYSQL 5 : Dari Pemula Hingga Mahir.[e-book] Achmatim.net. Tersedia di:<http://achmatim.net/2010/01/30/buku-gratis-mysql-5-dari-pemula-hingga-mahir/> [Diakses 1 September 2015]

Teodoro, G., Tavares, T., Coutinho, B., Meira, W.Jr. dan Guedes, D.,2003.Load Balancing on Stateful Clustered Web Servers. Computer Architecture and High Performance Computing, [online] Tersedia di: <http://dl.acm.org/citation.cfm?id=952438> [Diakses 12 Agustus 2015]

Zhang, W.,2010.Creating Linux Virtual Server.[online] Linux Virtual Server. Tersedia di:<http://www.linuxvirtualserver.org/linuxexpo.html> [Diakses 10 November 2015]

Zhang, W.,2011.*Virtual Server via Direct Routing*.[online] Linux Virtual Server. Tersedia di:<http://www.linuxvirtualserver.org/VS-DRouting.html> [Diakses 1 September 2015]

LAMPIRAN A KONFIGURASI SISTEM

A.1 Konfigurasi Jaringan

Client
- IP : 10.0.2.1
- Netmask : 255.255.255.0

Director
- IP 1 eth1 : 10.0.2.2
- Netmask : 255.255.255.0
- IP 2 eth1:0 : 10.0.2.10
- Netmask : 255.255.255.255

Real Server 1
- IP : 10.0.2.11
- Netmask : 255.255.255.0
- Gateway : 10.0.2.10

Real Server 2
- IP : 10.0.2.12
- Netmask : 255.255.255.0
- Gateway : 10.0.2.10

Real Server 3
- IP : 10.0.2.13
- Netmask : 255.255.255.0
- Gateway : 10.0.2.10

A.2 Konfigurasi Sistem

Director :

Install ipvsadm

```
#!/bin/bash
yum install ipvsadm
```

Aktifkan forwarder.sh

```
#!/bin/bash # aktifkan forwarder
echo "1" > /proc/sys/net/ipv4/ip_forward
```

Konfigurasi Ipvsadm

```
#!/bin/bash
echo "
-A -t 202.103.106.5:80 -s nq
-a -t 202.103.106.5:80 -r 172.16.0.2:80 -g
-a -t 202.103.106.5:80 -r 172.16.0.3:80 -g
-a -t 202.103.106.5:80 -r 172.16.0.4:80 -g
" | ipvsadm -R
```

Real Server :

Konfigurasi network interface

Real server 1

```
#ifconfig eth0 10.0.2.11 netmask 255.255.255.255 up
#ifconfig lo:0 10.0.2.10 netmask 255.255.255.255 up
#route add -host 10.0.2.10 dev lo:0
```

Real server 2

```
#ifconfig eth0 10.0.2.12 netmask 255.255.255.255 up
#ifconfig lo:0 10.0.2.10 netmask 255.255.255.255 up
#route add -host 10.0.2.10 dev lo:0
```

Real server 3

```
#ifconfig eth0 10.0.2.13 netmask 255.255.255.255 up
#ifconfig lo:0 10.0.2.10 netmask 255.255.255.255 up
#route add -host 10.0.2.10 dev lo:0
```

Konfigurasi forwarding dan arp_ignore

```
vi /etc/sysctl.conf
net.ipv4.ip_forward = 1
net.ipv4.conf.lo.arp_ignore = 1
net.ipv4.conf.lo.arp_announce = 2
net.ipv4.conf.all.arp_ignore = 1
net.ipv4.conf.all.arp_announce = 2]
```

Konfigurasi replikasi database

[*Database* 1]

```
#vi /etc/my.cnf
#server-id=1
#log-bin=mysql-bin
#relay-log=relay-bin
#log-slave-updates
#binlog-do-db=ci_blog
#mysql -u root -p
#CHANGE MASTER TO
 MASTER_HOST='10.0.2.13',MASTER_USER='repl',
 MASTER_PASSWORD='r3plic4tor',MASTER_LOG_FILE='mysql-
 bin.000013',MASTER_LOG_POS=98;
```

[*Database* 2]

```
#vi /etc/my.cnf
#server-id=2
#log-bin=mysql-bin
#relay-log=relay-bin
#log-slave-updates
#binlog-do-db=ci_blog
#mysql -u root -p
#CHANGE MASTER TO
 MASTER_HOST='10.0.2.11',MASTER_USER='repl',
 MASTER_PASSWORD='r3plic4tor',MASTER_LOG_FILE='mysql-
 bin.000013',MASTER_LOG_POS=98;
```

[*Database* 3]

```
#vi /etc/my.cnf
#server-id=3
#log-bin=mysql-bin
#relay-log=relay-bin
#log-slave-updates
#binlog-do-db=ci_blog
#mysql -u root -p
#CHANGE MASTER TO
 MASTER_HOST='10.0.2.12',MASTER_USER='repl',
 MASTER_PASSWORD='r3plic4tor',MASTER_LOG_FILE='mysql-
 bin.000013',MASTER_LOG_POS=98;
```