

**ANALISIS DAN IMPLEMENTASI LOAD BALANCING PADA WEB
SERVER DENGAN ALGORITMA LEAST CONNECTION**

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer



Disusun oleh :

WIDHI YAHYA

NIM. 0710683043

KEMENTERIAN PENDIDIKAN DAN KEBUDAYAAN

PROGRAM STUDI TEKNIK INFORMATIKA

PROGRAM TEKNOLOGI INFORMASI DAN ILMU KOMPUTER

UNIVERSITAS BRAWIJAYA

MALANG

2012

LEMBAR PERSETUJUAN

**ANALISIS DAN IMPLEMENTASI LOAD BALANCING PADA WEB
SERVER DENGAN ALGORITMA LEAST CONNECTION**

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer



Disusun oleh :

WIDHI YAHYA

NIM. 0710683043

Telah diperiksa dan disetujui oleh :

Dosen Pembimbing I

Dosen Pembimbing II

LEMBAR PENGESAHAN

**ANALISIS DAN IMPLEMENTASI LOAD BALANCING PADA WEB
SERVER DENGAN ALGORITMA LEAST CONNECTION**

SKRIPSI

KONSENTRASI KOMPUTASI BERBASIS JARINGAN

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh :

WIDHI YAHYA

NIM. 0710683043

Skripsi ini telah diuji dan dinyatakan lulus pada
tanggal 23 Juli 2012

Penguji I

Penguji II

R. Arief Setvawan, ST., MT

Aswin Suharsono, ST., MT.

Penguji III

Ir. Heru Nurwasito, M.Kom.

Mengetahui
Ketua Program Studi Teknik Informatika

Drs. Marji, MT.

NIP. 19670801 199203 1 001

KATA PENGANTAR

Syukur Alhamdulillah penulis ucapkan kehadiran Allah SWT atas limpahan rahmat dan karunia-Nya, sehingga penulis dapat menyelesaikan skripsi yang berjudul “Analisis dan Implementasi *Load Balancing* pada *Web Server* dengan Algoritma *Least Connection*”.

Skripsi ini merupakan tugas akhir yang diajukan untuk memenuhi syarat dalam memperoleh gelar Sarjana Komputer pada Program Teknologi Informasi dan Ilmu Komputer (PTIIK) Universitas Brawijaya Malang.

Penulis menyadari bahwa penyusunan skripsi ini tidak akan terwujud tanpa adanya bantuan dan dorongan dari berbagai pihak. Oleh karena itu pada kesempatan ini penulis menyampaikan ucapan terima kasih kepada yang terhormat :

1. Bapak dan Ibuku tercinta serta adikku tersayang, terima kasih atas segala doa, usaha dan dukungan serta kesabarannya.
2. Bapak Ir. Sutrisno, MT selaku Ketua Program Teknologi Informasi dan Ilmu Komputer Universitas Brawijaya Malang.
3. Bapak Sabriansyah R. A, ST., M.Eng selaku Dosen Pembimbing I yang telah menyempatkan waktu memberikan bimbingan dan arahan selama proses penyusunan skripsi ini.
4. Bapak Barlian Henryranu P, ST, MT selaku Dosen Pembimbing II yang telah menyempatkan waktu memberikan bimbingan dan arahan selama proses penyusunan skripsi ini.
5. Bapak dan Ibu Dosen Program Teknologi Informasi dan Ilmu Komputer Universitas Brawijaya Malang yang telah memberikan ilmu dan pengalaman berharga selama proses perkuliahan.
6. Kekasih penulis, Ekky Retno Stefany, yang selalu menjadi motivasi penulis, memberi doa, dan semangat agar skripsi ini bisa cepat selesai dan lulus untuk segera bekerja.

7. Teman-teman TPL, TIF dan, Program Studi Teknologi Informasi khususnya angkatan 2007.

Hanya doa yang bisa penulis berikan dan semoga Allah SWT memberikan pahala serta balasan kebaikan yang berlipat. Amin.

Penulis menyadari bahwa skripsi ini masih banyak kekurangan. Untuk itu, saran dan kritik yang membangun sangat penulis harapkan. Dan semoga skripsi ini membawa manfaat bagi penulis sendiri maupun pihak lain yang akan menggunakannya.

Malang, 19 Juni 2012

Penulis



ABSTRAK

Widhi Yahya, 2011. Analisis dan Implementasi Load Balancing pada Web Server dengan Algoritma Least Connection. Informatika, Program Teknologi Informasi dan Ilmu Komputer (PTI IK), Universitas Brawijaya, Malang. Dosen Pembimbing: Sabriansyah R. A, ST., M.Eng dan Barlian Henryranu P, ST, MT.

Network load balancing merupakan teknologi yang diharapkan mampu menangani beban (*request*) yang sangat besar dengan kemungkinan kegagalan yang kecil. *Network load balancing* dapat diterapkan pada sistem Linux Virtual Server (LVS). Penelitian ini membahas dan mengimplementasikan *load balancing* pada *web server*. Algoritma yang digunakan adalah *least connection*. Pengujian terhadap *web server* dilakukan untuk mengetahui tingkat ketersediaan yang tinggi serta mengetahui karakteristik dari *web server* yang mengimplementasikan *load balancing* dengan algoritma *least connection*. Parameter yang diuji untuk mengetahui performa *web server* antara lain waktu respon, *throughput* dan CPU *utilization*. Pada pengujian dengan koneksi 60000 dan *rate* 6000 koneksi/detik yang menunjukkan waktu respon untuk sistem *load balancing* pada *web server* 0,9 ms sedangkan *single web server* 5,42 ms, *throughput* sistem *load balancing* 7602,82 KBps sedangkan *single web server* 7358,04 KBps dan CPU *utilization* (CPU *Idle*) pada sistem *load balancing* 94,08% sedangkan *single web server* 84,76%. Hasil pengujian performa menunjukkan bahwa sistem *load balancing* lebih handal daripada *single web server*. Tingkat ketersediaan yang tinggi diperoleh pada saat memanipulasi *real server* yang terhubung dalam sistem.

Kata Kunci : Load Balancing, Algoritma Penjadwalan, Server.

DAFTAR ISI

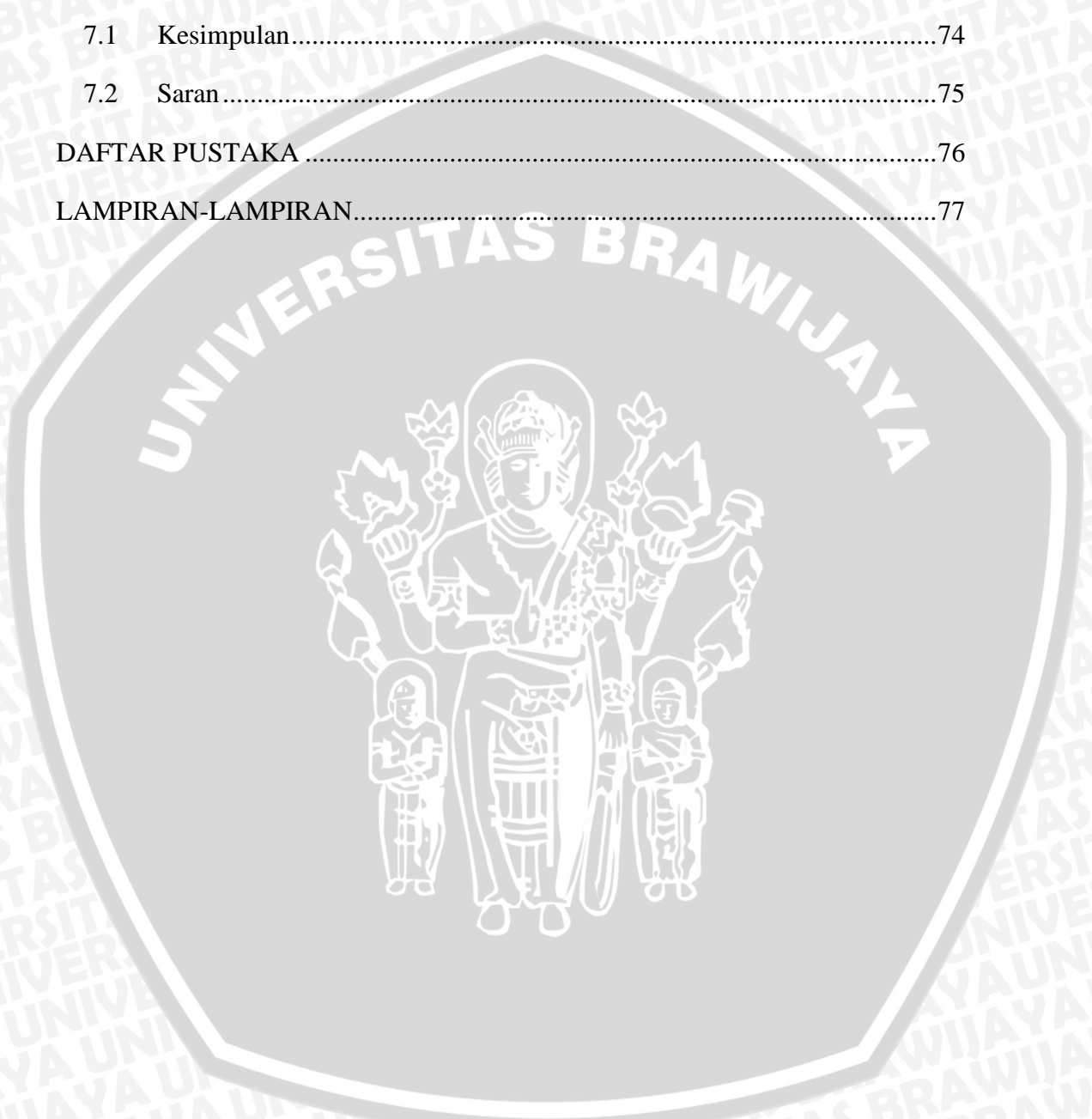
KATA PENGANTAR	i
ABSTRAK	vi
DAFTAR GAMBAR	x
DAFTAR TABEL	xii
BAB I PENDAHULUAN	1
1.1 Latar Belakang Permasalahan	1
1.2 Identifikasi dan Pembatasan Masalah	2
1.3 Rumusan Masalah	2
1.4 Tujuan	2
1.5 Manfaat	3
1.6 Sistematika Pembahasan	3
BAB II DASAR TEORI	5
2.1 Jaringan Komputer	5
2.1.1 Pengertian	5
2.1.2 Model Hubungan pada Jaringan Lokal (LAN)	6
2.2 Protokol	8
2.2.1 Internet Protokol	9
2.2.2 TCP (<i>Transmission Control Protocol</i>)	9
2.2.3 HTTP	10
2.3 Web Server	12
2.4 Linux Virtual Server (LVS)	13
2.5 Load Balancing	16
2.6 Algoritma Penjadwalan	19



2.7	Parameter yang Dianalisis	22
BAB III METODOLOGI PENELITIAN		23
3.1	Studi Literatur	24
3.2	Analisis Kebutuhan	24
3.3	Perancangan	25
3.4	Implementasi	25
3.4.1	Instalasi dan Konfigurasi Perangkat Keras	26
3.4.2	Instalasi dan Konfigurasi Perangkat Lunak	27
3.5	Pengujian dan Analisis	27
3.5.1	Pengujian Kinerja (<i>High Availability</i>)	28
3.5.2	Pengujian Performa	29
3.6	Pengambilan Kesimpulan	31
BAB IV PERANCANGAN		32
4.1	Analisis Kebutuhan	32
4.1.1	<i>System Requirements</i>	32
4.2	Algoritma Penjadwalan <i>Least Connection</i>	36
BAB V IMPLEMENTASI		39
5.1	Proses Instalasi pada <i>Real Server</i>	39
5.2	Proses Instalasi GlusterFS	40
5.2.1	Proses Instalasi Server GlusterFS	40
5.2.2	Proses Instalasi Client GlusterFS	41
5.3	Proses Instalasi <i>Director (Virtual Server)</i>	44
BAB VI PENGUJIAN DAN ANALISIS		55
6.1	Pengujian Kinerja	55
6.2	Pengujian Performa	62
6.2.1	Single Web Server	64



6.2.2	Load Balancing dengan 2 <i>Real Server</i>	66
6.2.3	Load Balancing dengan 3 <i>Real Server</i>	68
BAB VII PENUTUP		74
7.1	Kesimpulan	74
7.2	Saran	75
DAFTAR PUSTAKA		76
LAMPIRAN-LAMPIRAN		77



DAFTAR GAMBAR

Gambar 2.1 Time-Shared Computer System	5
Gambar 2.2 Model Hubungan <i>Peer to Peer</i>	6
Gambar 2.3 Model Hubungan <i>Client Server</i>	7
Gambar 2.4 Perintah <i>Get</i> dari Internet Explorer Sistem Operasi Machintos ke <i>Server</i>	11
Gambar 2.5 Perintah <i>Get</i> dari Netscape Sistem Operasi Windows ke <i>Server</i>	11
Gambar 2.6 Respon <i>Get</i> HTTP yang Dikirim dari <i>Server</i> ke <i>Browser</i>	12
Gambar 2.7 Struktur Dasar Linux Virtual Server dengan 3 buah <i>real server</i>	15
Gambar 2.8 Konsep Dasar <i>Server Load Balancing</i>	17
Gambar 2.9 Contoh Implementasi <i>Two-Ways Layer-3 Forwarding</i>	18
Gambar 2.10 <i>Three-Ways Handshake</i> pada NAT (<i>Layer 3 Forwarding</i>)	19
Gambar 3.2 Topologi LVS.....	26
Gambar 3.3 Diagram Alir Pengujian Kinerja (<i>High Availibility</i>)	28
Gambar 3.4 Diagram Alir Pengujian Performa.....	30
Gambar 4.1 Diagram Topologi LVS.....	33
Gambar 4.2 Mekanisme Algoritma <i>Least Connection</i>	37
Gambar 5.1 Isi Glusterfsd.vol.	41
Gambar 5.2 Isi Glusterfs.vol.	42
Gambar 5.3 Hasil <i>Mount</i> GlusterFS ke <i>/var/www</i>	43
Gambar 5.4 Glusterfs.vol Telah Ter- <i>mount</i> pada <i>var/www</i>	43
Gambar 5.5 Glusterfs.vol Telah Ter- <i>mount</i> pada <i>var/www</i>	44
Gambar 5.6 Sistem GlusterFS	44
Gambar 5.7 <i>Enable IP Packet Forwarding</i>	45

Gambar 5.8 <i>Masquerade iptables</i>	46
Gambar 5.9 Tabel LVS	46
Gambar 5.10 <i>Virtual Service</i>	47
Gambar 5.11 Halaman <i>Monitoring Piranha</i>	47
Gambar 5.12 Halaman <i>Global Setting Piranha</i>	48
Gambar 5.13 Halaman <i>Virtual Server Piranha</i>	49
Gambar 5.14 Halaman <i>Edit Virtual Server Piranha</i>	50
Gambar 5.15 Halaman <i>Edit Real Server</i>	52
Gambar 5.16 <i>Edit Real server</i>	53
Gambar 5.17 <i>Edit Monitoring Scripts</i>	54
Gambar 6.1 <i>Capture Request HTTP Client ke Server dengan Wireshark</i>	56
Gambar 6.2 <i>Capture Request HTTP pada eth0 director (IP public)</i>	56
Gambar 6.3 <i>Capture Request HTTP pada eth1 director (IP NAT ke real-server)</i>	57
Gambar 6.4 <i>Capture Request HTTP pada Real Server dengan tcpdump</i>	57
Gambar 6.5 <i>Screenshot Halaman Web yang Diminta oleh Client</i>	58
Gambar 6.6 <i>Screenshot Halaman Web yang Diminta oleh Client</i>	59
Gambar 6.7 LVS Routing Table dengan 0 Koneksi.....	60
Gambar 6.8 LVS Routing Table dengan 5 Koneksi.....	60
Gambar 6.9 LVS Routing Table dengan 6 Koneksi.....	61
Gambar 6.10 LVS Routing Table dengan 5 Koneksi.....	62
Gambar 6.11 LVS Routing Table dengan 6 Koneksi.....	62
Gambar 6.12 Grafik Waktu Respon (ms) Hasil Pengujian Performa.....	71
Gambar 6.13 Grafik CPU <i>Utilization (%)</i> Hasil Pengujian Performa.....	73



DAFTAR TABEL

Tabel 2.1 Tipe *Forwarding*18

Tabel 4.1 Distribusi Koneksi.....38

Tabel 6.1 Tabel Koneksi Single Web Server yang Menerima 10000 Koneksi dengan Rate 1000 Koneksi/Detik64

Tabel 6.2 Tabel Koneksi *Single Web Server* yang Menerima 30000 Koneksi dengan *Rate* 3000 Koneksi/Detik.65

Tabel 6.3 Tabel Koneksi *Single Web Server* yang Menerima 360000 Koneksi dengan *Rate* 6000 Koneksi/Detik.66

Tabel 6.4 Tabel Koneksi *Load Balancing* dengan 2 *Real Server* yang Menerima 10000 Koneksi dengan *Rate* 1000 Koneksi/Detik67

Tabel 6.5 Tabel Koneksi *Load Balancing* dengan 2 *Real Server* yang Menerima 30.000 Koneksi dengan *Rate* 3000 Koneksi/Detik67

Tabel 6.6 Tabel Koneksi *Load Balancing* dengan 2 *Real Server* yang Menerima 60.000 Koneksi dengan *Rate* 6000 Koneksi/Detik68

Tabel 6.7 Tabel Koneksi *Load Balancing* dengan 3 *Real Server* yang Menerima 60.000 Koneksi dengan *Rate* 6000 Koneksi/Detik68

Tabel 6.8 Tabel Koneksi *Load Balancing* dengan 3 *Real Server* yang Menerima 60.000 Koneksi dengan *Rate* 6000 Koneksi/Detik68

Tabel 6.9 Tabel Koneksi *Load Balancing* dengan 3 *Real Server* yang Menerima 60.000 Koneksi dengan *Rate* 6000 Koneksi/Detik70

Tabel 6.10 Standar Deviasi *Throughput*72



BAB I

PENDAHULUAN

1.1 Latar Belakang Permasalahan

Pendistribusian informasi jarak jauh dapat dilakukan melalui jaringan komputer yang telah berkembang semakin cepat dan menghubungkan banyak komputer di dunia. Salah satu media pada *layer* aplikasi sebagai pendukung *interface* adalah *website*. *Website* digunakan dalam berbagai bidang seperti pendidikan, bisnis, pemerintahan serta banyak bidang yang lain.

Single web server banyak digunakan oleh berbagai organisasi maupun perusahaan memiliki kekurangan yang sangat serius. Ketika *website* memperoleh *request* yang sangat banyak dalam waktu tertentu sehingga *server* tidak dapat menanganinya, dapat memungkinkan terjadi *overload* dan *crash*. Untuk mengatasi masalah tersebut dapat menggunakan sistem *load balancing* [BOU-01].

Network load balancing merupakan teknologi yang diharapkan mampu menangani beban (*request*) yang sangat besar dengan kemungkinan kegagalan yang kecil. Contoh *network load balancing* adalah Linux Virtual Server (LVS).

Load balancing adalah metodologi dalam jaringan komputer untuk mendistribusikan beban kerja dari beberapa komputer *server* atau *cluster* komputer [BOU-01]. Ada beberapa algoritma dalam *load balancing* antara lain, *Round Robin*, *Middle Manager*, *Ratio*, *Fastest*, *Least Connection*, dan masih banyak yang lainnya. Pemilihan algoritma *load balancing* yang tepat dapat memaksimalkan kinerja *server*.

Sesuai latar belakang mengenai pentingnya penggunaan dan pemilihan algoritma *load balancing* penulis mengambil judul “ Analisis dan Implementasi *Load Balancing* pada *Web Server* dengan Algoritma *Least Connection*” diharapkan dapat membuat *web server* yang lebih *realible* dan *high availability* daripada *single web server*.

1.2 Identifikasi dan Pembatasan Masalah

Identifikasi masalah pada pembahasan ini terkait dengan hal-hal seputar perancangan *server* (LVS), teori yang mendukung, penjelasan tentang teknologi *load balancing*, teknis implementasi algoritma *least connection* pada Linux Virtual Server (LVS) dan Piranha.

Agar mendapatkan hasil pembahasan yang sesuai dengan apa diharapkan, maka perlu diberikan pembatasan masalah pada pengembangan sistem ini, yaitu :

1. Hanya pada implementasi dan analisis penggunaan algoritma *least connection* yang diterapkan pada sistem Linux Virtual Server (LVS).
2. Test performasi algoritma hanya pada protokol http (port 80).
3. Diimplementasikan pada pengalamatan jaringan pada Ipv4.
4. Pengujian menggunakan 1, 2 dan 3 *server*.
5. Tidak membahas secara rinci *mirroring data*.

1.3 Rumusan Masalah

Berdasarkan permasalahan yang diangkat pada bagian latar belakang, maka rumusan masalah dikhususkan pada :

1. Implementasi algoritma *least connection* pada *load balancing* LVS.
2. Mengetahui karakteristik *web server* yang menggunakan sistem *load balancing* dengan algoritma *least connection*.

1.4 Tujuan

Berdasarkan rumusan masalah yang ditulis diatas, tujuan dari penulisan skripsi ini adalah memahami dan berhasil mengimplementasikan sistem Linux Virtual Server (LVS), membuat *web server* yang lebih *realible* dan *high availibility* dengan teknologi *load balancing* serta megetahui dan memahami karakteristik dari *web server* yang mengimplementasikan *load balancing* dengan algoritma *least connection*.

1.5 Manfaat

Sesuai dengan rumusan dan latar belakang, manfaat dari analisis dan implementasi penggunaan algoritma *least connection* pada *web server* adalah sebagai pertimbangan pada saat pemilihan algoritma *load balancing* dalam pembangunan sebuah *web server*.

1.6 Sistematika Pembahasan

Sistematika pembahasan dari penyusunan proyek akhir ini direncanakan sebagai berikut :

BAB I PENDAHULUAN

Pendahuluan terdiri dari latar belakang, identifikasi dan pembatasan masalah, rumusan masalah, tujuan dan manfaat serta sistematika pembahasan dari proyek akhir ini.

BAB II DASAR TEORI

Bab ini membahas mengenai teori-teori yang berkaitan dan menunjang dalam penyelesaian proyek akhir ini.

BAB III METODOLOGI PENELITIAN

Langkah-langkah dalam merancang *web server* dengan *load balancing* yang menggunakan algoritma *least connection* akan dijelaskan pada bab metodologi penelitian. Pada bab ini juga dijelaskan langkah-langkah implementasi, analisis, dan pengujian sistem yang dirancang.

BAB IV PERANCANGAN

Pada bab perancangan membahas analisis kebutuhan sistem seperti, sistem operasi, perangkat lunak, dan perangkat keras yang digunakan dalam implementasi sistem serta akan dijelaskan topologi jaringan yang akan digunakan dalam implementasi sistem.

BAB V IMPLEMENTASI

Pada bab implementasi akan dijelaskan pembuatan *web server* dengan sistem Linux Virtual Server (LVS) dan penggunaan algoritma *least connection* untuk *load balancing* pada *web server* yang dibuat.

BAB VI PENGUJIAN DAN ANALISIS

Bab VI berisi pengujian dan analisis dari perancangan *web server* dengan *load balancing* yang menggunakan algoritma *least connection* terhadap beberapa parameter meliputi *throughput*, *CPU utilization*, dan waktu respon.

BAB VII PENUTUP

Kesimpulan dari penelitian ini dibuat setelah dilakukan pengujian pada sistem yang terangkum pada bab penutup. Untuk meningkatkan hasil dari kinerja sistem yang telah dibuat dalam penelitian ini maka diberikan saran-saran untuk perbaikan dan penyempurnaan sistem yang telah dibuat.



BAB II

DASAR TEORI

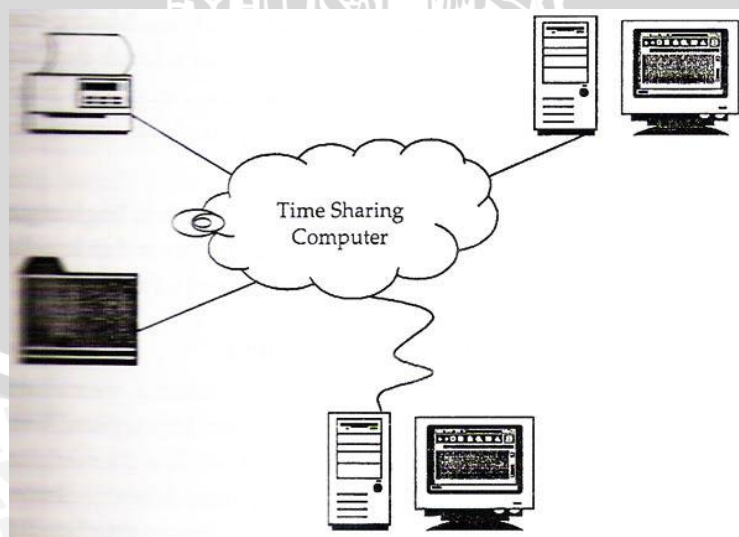
2.1 Jaringan Komputer

Pada sub-bab ini membahas pengertian jaringan komputer dan model hubungan pada jaringan lokal.

2.1.1 Pengertian

Jaringan komputer merupakan sekumpulan komputer yang terhubung bersama dan dapat berbagi sumber daya yang dimilikinya, seperti printer, CDROM, pertukaran file, dan komunikasi secara elektronik antar komputer. Hubungan antarkomputer dalam jaringan dapat mempergunakan media kabel, telepon, gelombang radio, satelit, atau sinar infra merah (*infrared*) [HID-04].

Prosesor berkomunikasi dengan periferalnya melalui peralatan *input/output* (I/O) pada jarak yang pendek dan bekerja pada kecepatan yang sangat rendah. Tahun 1960-an lahir konsep *timesharing*, dimana pengguna dihubungkan ke komputer melalui suatu *dumb terminal*. Konsep *timesharing* dapat dilihat pada Gambar 2.1.



Gambar 2.1 Time-Shared Computer System

Sumber : [IRA-05:5]

Setelah berkembangnya teknologi IC (*integrated circuit*) dan mikro-prosesor yang memungkinkan munculnya komputer pribadi secara drastis mengubah cara pandang orang terhadap komputer. Munculnya teknologi jaringan lokal (*Local Area Network-LAN*) dalam tahun 1980-an melengkapi komputer dengan kemampuan berkomunikasi dengan komputer lainnya. Kondisi ini menyebabkan terjadinya migrasi dari konsep pemrosesan secara tersentralisasi (*centralized computing*) menjadi konsep pemrosesan secara terdistribusi (*distribution computing*).

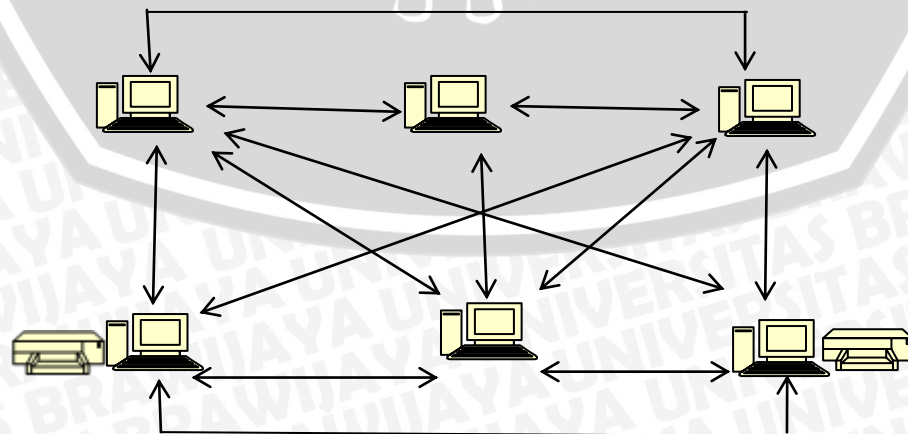
Pada *centralized computing*, sistem atau prosesor utama diletakkan secara terpusat. Semua komputer yang letaknya berjauhan dihubungkan dengan menggunakan link secara langsung ke prosesor utama tersebut. Semua informasi (*database*) terletak di sistem pusat.

Pada kasus *distributed computing*, prosesor atau komputer utama didistribusikan pada lokasi-lokasi yang berbeda. Masing-masing komputer tersebut mempunyai sebagian atau seluruh duplikat dari data yang ada di komputer pusat. Pengguna mengakses informasi dari prosesor yang terdekat yang secara periodik meng-*update* data pada *database*-nya [IRA-05].

2.1.2 Model Hubungan pada Jaringan Lokal (LAN)

Ada dua model hubungan jaringan lokal yaitu *Peer-to-Peer* dan *Client/Server*.

2.1.2.1 Peer-to-Peer

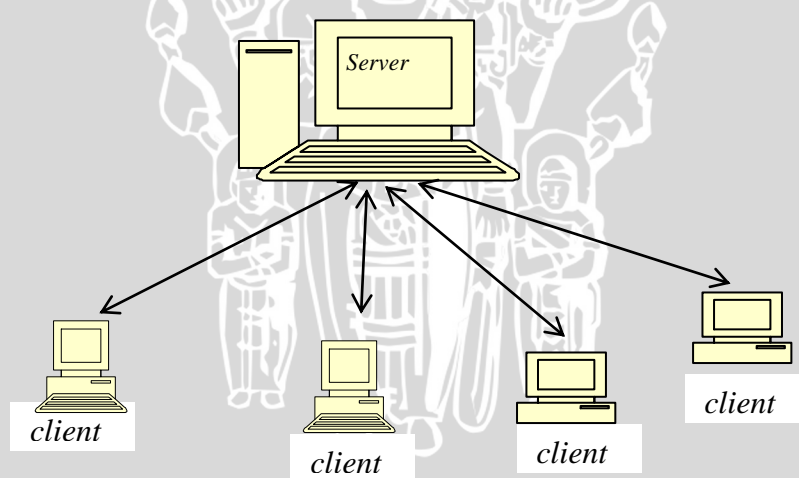


Gambar 2.2 Model Hubungan *Peer to Peer*

Model hubungan *peer to peer* pada Gambar 2.2 memungkinkan *user* berbagi sumber daya yang ada dikomputernya baik itu berupa *file*, layanan *printer* dan lain-lain serta mengakses sumber daya yang terdapat pada komputer lain. Namun model ini tidak mempunyai sebuah *file server* atau sumber daya yang terpusat. Didalam model hubungan *peer to peer* ini, seluruh komputer adalah sama, yang mana mempunyai kemampuan yang sama untuk memakai sumber daya yang tersedia didalam jaringan. Model ini didisain untuk jaringan berskala kecil dan menengah.

Kelebihan model hubungan *peer to peer* adalah tidak terlalu mahal karena tidak membutuhkan *server*, mudah dalam konfigurasinya. Kekurangan model *peer to peer* diantaranya tidak terpusat untuk penyimpanan *file* dan aplikasi [IRA-05:29].

2.1.2.2 Client/Server



Gambar 2.3 Model Hubungan *Client Server*

Model hubungan *client/server* pada Gambar 2.3 memungkinkan jaringan untuk mensentralisasi fungsi dan aplikasi kepada satu atau dua *dedicate file server*. Sebuah *file server* menjadi jantung dari keseluruhan sistem, memungkinkan untuk mengakses sumber daya, dan menyediakan keamanan. *Workstation* yang berdiri sendiri dapat mengambil sumber daya yang ada pada *file*

server. Mode hubungan ini menyediakan mekanisme untuk mengintegrasikan seluruh komponen yang ada di jaringan dan memungkinkan banyak pengguna secara bersama-sama memakai sumber daya pada *file server* [IRA-05:30].

2.2 Protokol

Protokol merupakan sekumpulan aturan yang mendefinisikan beberapa fungsi seperti pembuatan hubungan; mengirim pesan, data, informasi atau *file*; yang harus dipenuhi oleh pengirim atau penerima agar suatu sesi komunikasi data dapat berlangsung dengan baik dan benar. Di samping itu, protokol juga merupakan sekumpulan aturan untuk memecahkan masalah-masalah khusus yang terjadi antara alat-alat komunikasi agar transmisi data dapat berjalan dengan baik dan benar.

Faktor-faktor yang diperhatikan dalam protokol antara lain :

a. *Syntax*

Syntax merupakan format data dan cara pengkodean yang digunakan untuk mengkodekan sinyal atau tegangan.

b. Sematik

Sematik digunakan untuk mengetahui maksud dan mengoreksi informasi yang dikirim.

c. *Timing*

Timing merupakan pewaktuan yang digunakan untuk mengetahui transmisi kecepatan data.

Secara umum protokol berfungsi untuk membuat hubungan antara pengirim dan penerima serta menyalurkan informasi dengan keandalan yang tinggi. Secara rinci fungsi protokol adalah sebagai berikut :

- a. Fragmentasi dan *reassembly*, yaitu membagi-bagi berita dalam bentuk paket-paket data saat komputer mengirim data dan menggabungkan lagi setelah paket-paket tersebut diterima oleh penerima.
- b. *Encapsulation*, melengkapi paket-paket dengan *address*, kode koreksi, dll.

- c. *Connection control*, yaitu membangun hubungan komunikasi antara pengirim dan penerima. Tahapan dalam membangun komunikasi tersebut meliputi membangun hubungan, melakukan transmisi data dan mengakhiri hubungan (*connection termination*).
- d. *Flow control*, protokol berfungsi mengatur perjalanan data.
- e. *Error control*, protokol juga berfungsi untuk mengontrol terjadinya kesalahan dalam proses komunikasi data.
- f. *Transmission service*, protokol berfungsi untuk memberi pelayanan komunikasi data khususnya dan berkaitan dengan prioritas dan keamanan atau perlindungan [OET-03].

2.2.1 Internet Protokol

Internet protokol atau IP adalah mekanisme transmisi yang digunakan oleh TCP/IP yang sifatnya *unreliable* dan *connectionless*. Internet protokol disebut *best effort deliver* yang artinya IP menyediakan *no error checking* atau *tracking*. Jika diperlukan reliabilitas maka IP dipasangkan dengan protokol yang reliabel, misalnya TCP. Contoh alamat dari IP adalah, kantor pos mengirimkan surat tapi tidak selalu sukses dikirimkan. Jika alamat surat tersebut tidak lengkap maka terserah pengirim ingin mengantarkannya atau tidak. Kantor pos juga tidak pernah menjejaki kemana surat-surat yang jumlahnya jutaan itu terkirim [IRA-05].

2.2.2 TCP (*TransmissionControl Protocol*)

Protokol TCP menjamin keutuhan data yang ditransfer. Paket data yang dikirim berorientasi pada hubungan komunikasi visual yang sebelumnya harus terlebih dahulu dibentuk. Teknik ini disebut dengan *connection oriented* dan paket data yang dikirim dinamakan *reliable data stream* atau *segment*.

Lapisan *transport* melakukan 2 fungsi, antara lain:

- a. *Flowcontrol* melalui *sliding window*.
- b. Transmisi yang realibel melalui nomor urut (*sequence number*) dan *acknowledgement*.

TCP mengirimkan *header* dengan informasi yang lebih lengkap dari UDP, yaitu :

- a. Port asal – port tujuan

- b. Seperti juga UDP, port adalah nomor yang harus dikenal pada mesin *remote* dan dijadikan sebagai identitas layanan aplikasi. *Source port* berfungsi karena penerima harus mengirim jawaban dengan menyertakan *source port* tersebut.
- c. Nomor urut pengiriman – Nomor *Acknowledge*.
- d. Nomor urut pengiriman diperlukan untuk menyusun segment yang sampai dan meminta pengulangan bila segmen yang diterima tidak baik. Nomor *acknowledge* adalah nomor urut yang diberikan mesin *remote* untuk menyatakan bahwa segmen diterima dengan status kondisi baik atau buruk.
- e. *Checksum*

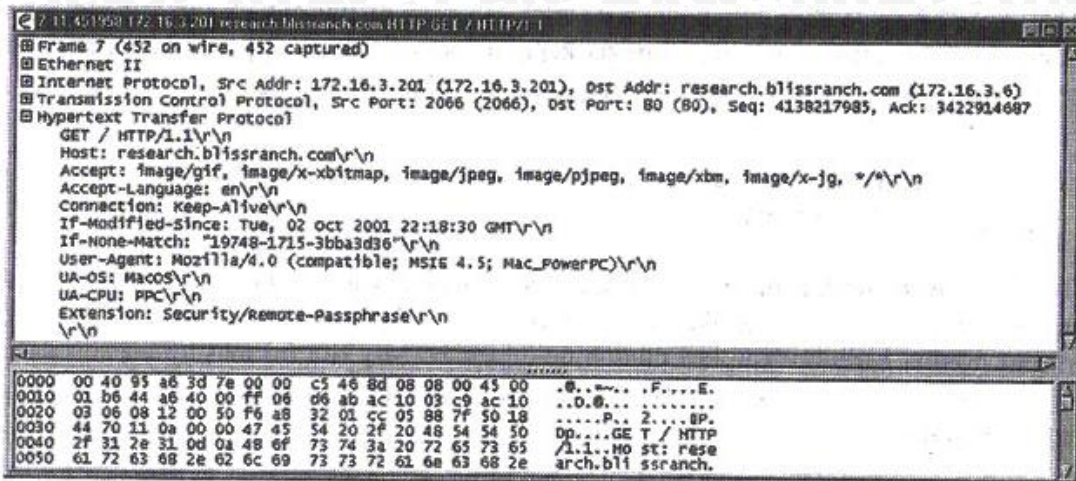
Checksum adalah mekanisme pemeriksaan dan verifikasi TCP untuk mendeteksi *error* pada pengiriman data.

Jika dibandingkan UDP, TCP termasuk lebih handal. Namun, *overhead* yang diperlukan untuk TCP lebih banyak dibanding UDP. TCP memerlukan *handshake* tiga babak untuk transmisi data. Pertama, *initial request* (permintaan untuk mengirim); kedua, jawaban dari *host* yang jauh tersebut; ketiga, *acknowledgement* atau jawaban kembali atas *reply* tersebut.

Dengan demikian, unjuk kerja dari UDP lebih cepat daripada TCP. Analogi TCP adalah seperti hubungan telepon, sedangkan UDP menyerupai pengiriman surat lewat pos [MAD-03].

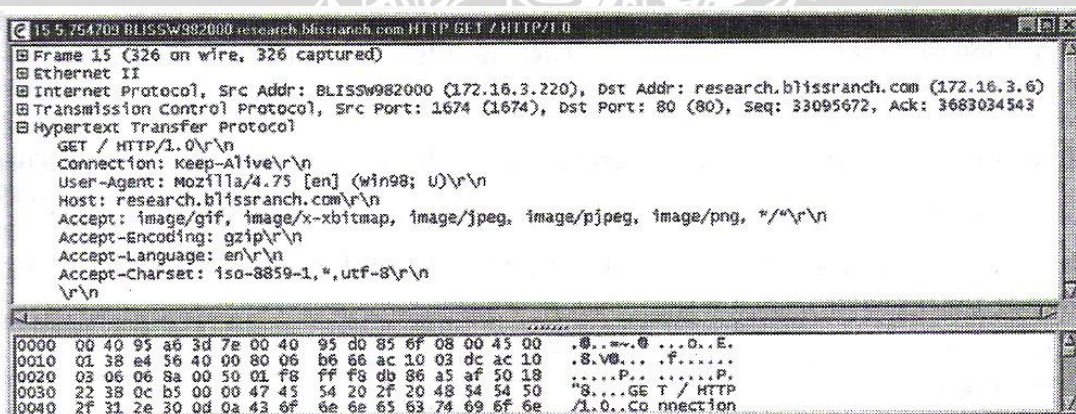
2.2.3 HTTP

Protokol *transfer Hypertext (hypertext transfer protocol-http)* memeriksa aturan-aturan untuk komunikasi antara *browser* dan *web server*. Permintaan http ini dikirim sebagai *text ASCII* dan terdapat beberapa kata kunci yang memungkinkan tipe tindakan yang berlainan. Perintah *get* digunakan untuk digunakan untuk meminta satu dokumen atau hal dari *server* bersangkutan. Gambar 2.4 dan 2.5 menunjukkan contoh-contoh dua sistem operasi berbeda menggunakan *browser* yang berbeda, mengakses informasi yang sama pada *web server* yang sama. Perhatikan informasi yang dikirim ke *server* dari *browser* bersangkutan ketika perintah *get HTTP* dikirim.



Gambar 2.4 Perintah *Get* dari Internet Explorer Sistem Operasi Machintos ke *Server*
Sumber : [TIT-04:210]

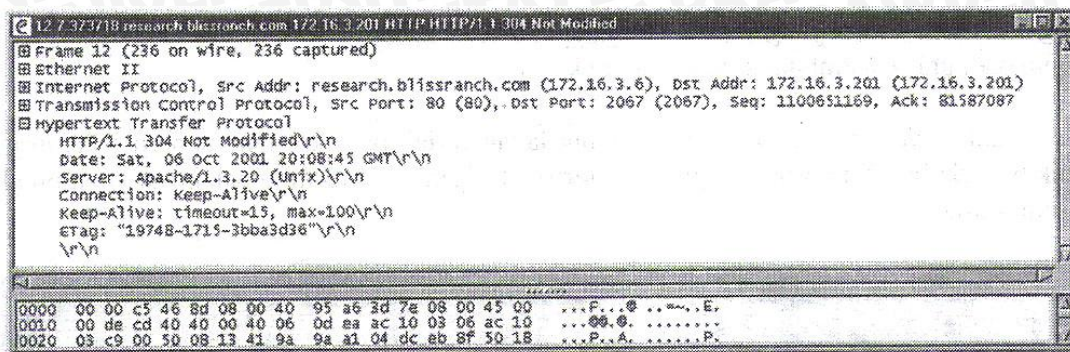
Informasi yang dikembalikan dari perintah *get* oleh *server* mencakup *header* yang berisi informasi status, baris kosong, dan kemudian butir itu sendiri. Gambar 2.5 merupakan contoh informasi yang dikirim kembali ke *browser* untuk menjawab satu perintah *get*.



Gambar 2.5 Perintah *Get* dari Netscape Sistem Operasi Windows ke *Server*
Sumber : [TIT-04:211]

Di bawah judul “*Hypertext Transfer Protocol*” dalam Gambar 2.6, kode 304 memperlihatkan bahwa page yang diminta tidak dimodifikasi. *Web server* ini mengirim page tersebut ke *browser* karena page tersebut telah dimodifikasi.





Gambar 2.6 Respon *Get* HTTP yang Dikirim dari *Server* ke *Browser*
Sumber : [TIT-04:211]

Perintah ini meliputi *post*, *put*, dan *head*. Perintah HTTP *post* menggunakan informasi yang dimasukkan ke dalam *browser* ke basis data. Perintah *put* mengirim informasi ke *web server* yang memodifikasi *file*. Perintah ini digunakan oleh sebagian program yang menerbitkan *web* untuk mendapatkan *web page* yang dikembangkan dalam program penyusunan *web* dalam *web server*. Perintah *head* digunakan untuk meminta status suatu hal tanpa mengembalikan hal tersebut ke *browser*.

Salah satu yang menjadi perhatian khusus untuk HTTP ini ialah kurangnya keamanan. Untuk memperoleh transmisi *page* secara aman melalui jaringan yang tidak aman, seperti internet, protokol https biasanya digunakan. Protokol https ini menyediakan sertifikat koneksi dan sesi serta enkripsi data. *Web server* bersangkutan harus diatur agar dapat menangani HTTPS, dan transfer informasi membutuhkan waktu yang sedikit lebih lama disebabkan oleh proses tambahan (*overhead*) yang dilakukan oleh protokol keamanan ini [TIT-04:210].

2.3 Web Server

World Wide Web (WWW) atau sering dikenal dengan nama *web* adalah aplikasi pertama yang berbasis internet. www dikembangkan oleh Tim Berner Lee. *Web* adalah aplikasi *multiplatform* atau berbasis grafis atau sering disebut aplikasi berbasis GUI (*Graphic User Interface*). Dengan menggunakan bahasa pemrograman yang disebut dengan markup language, bahasa paling populer dan disupport oleh *web server* adalah *Hypertext Markup Language* (HTML), *web server* menggunakan port 80 sebagai jalur komunikasi, *web server* mempunyai kemampuan untuk transmisi data berupa *text*, *database*, suara, gambar, dan video

secara *realtime*. Kecepatan transmisi bergantung pada kecepatan koneksi yang kita miliki.

Dalam perkembangannya, jenis pemrograman berbasis *web* mengalami perkembangan pesat, yaitu mulai memasuki pengolahan multimedia dan *database* [MAD-03].

Banyak aplikasi yang digunakan untuk membuat suatu *web server*. Akan tetapi dalam dunia linux, apache adalah satu-satunya aplikasi yang digunakan untuk *web server*, yang disebabkan karena beberapa alasan yaitu :

- a. Kecepatan yang dimiliki lebih baik jika dibandingkan dengan aplikasi-aplikasi lain yang digunakan untuk *web server*.
- b. *Performance* yang sangat baik.
- c. Dapat diperoleh dengan gratis.

Apache merupakan turunan dari *web server* yang dikembangkan oleh NCSA (*National Center for Super Computing Application*) sekitar tahun 1995 yang dikenal dengan NCSA HTTP Daemon (NCSA HTTPd) yang pada RedHat Linux 7.1 telah digunakan patch untuk mengganti NCSA HHTTPd tersebut. Pada RedHat Linux 7.1 Apache digunakan adalah versi 1.3.19 dengan release 5, atau lebih dikenal dengan paket `apache-1.3.19-5.i386.rpm`.

Apache *web server* memiliki program pendukung cukup banyak yang dapat memberikan layanan yang cukup bagi penggunaannya. Berikut ini adalah beberapa program pendukung dari Apache *webServer* tersebut, antaranya :

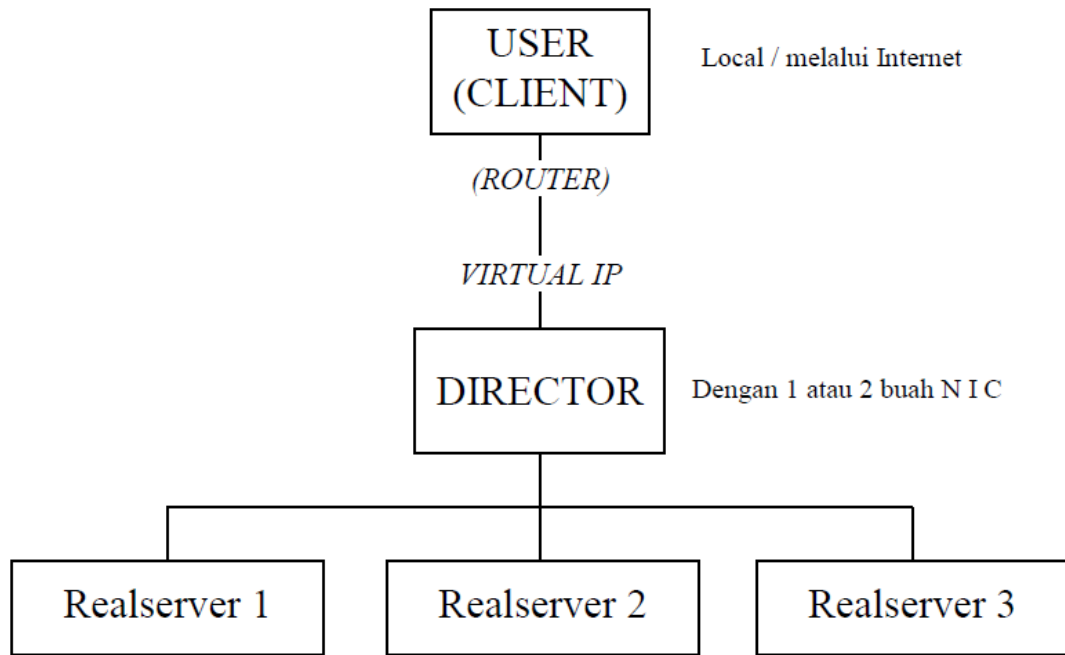
- a. Kontrol akses.
- b. *Common gateway interface* (CGI) yang paling terkenal dan sangat sering digunakan adalah PERL (*Practical Extraction and Report Language*).
- c. PHP (*Personal Home Page*).
- d. SSI (*Server Side Include*) [SYA-03].

2.4 Linux Virtual Server (LVS)

Linux adalah sistem operasi yang bersifat *multiuser*, *multitasking* yang berbasis UNIX, namun bersifat *freeware*, yaitu *software* yang bebas didistribusikan dan dikembangkan oleh semua orang [MAD-03].

Seperti halnya UNIX, sistem operasi Linux bergantung pada apa yang disebut dengan kernel yang berisi seluruh informasi *hardware* dan *interface* yang ada pada PC. Kernel Linux pada awalnya dikembangkan untuk CPU berbasis 80386. 80386 pada awalnya didesain untuk *multitasking*, namun sistem operasi yang masih ada bersifat *single-task*, yaitu MS-DOS. Untuk itu Linux memanfaatkan seluruh fasilitas yang ada pada 80386, terutama manajemen memori dan *floating emulation* yang membuat Linux bisa berfungsi pada prosesor yang tidak memiliki co-prosesor atau prosesor pengolah data matematis [MAD-03].

Linux Virtual Server atau disingkat LVS merupakan suatu teknologi *clustering* yang dapat digunakan untuk membangun suatu *server* dengan menggunakan kumpulan dari beberapa buah *real server*. LVS merupakan implementasi dari komputer *cluster* dengan metode *High Availability*. LVS mengimbangi berbagai bentuk dari *service* jaringan pada banyak mesin dengan memanipulasi paket sebagaimana diproses TCP/IP *stack*. Satu dari banyak peran yang paling umum dari Linux Virtual Server adalah bertindak sebagai *server* yang berada pada garis terdepan dari kelompok *server web*. Seperti ditunjukkan pada Gambar 2.7, Linux virtual server atau LVS ini terdiri dari sebuah *director* dan beberapa *real server* yang bekerja bersama dan memberikan servis terhadap permintaan *user*. Permintaan *user* diterima oleh *director* yang seolah-olah berfungsi sebagai IP *router* yang akan meneruskan paket permintaan *user* tersebut pada *real server* yang siap memberikan servis yang diminta.



Gambar 2.7 Struktur Dasar Linux Virtual Server dengan 3 buah *real server*

Sumber : [ALE-02]

Dengan demikian *virtual server* akan terdiri dari beberapa komputer yang mempunyai image yang sama tetapi ditempatkan pada IP yang berbeda. *User* dapat mengakses *virtual server* tersebut dengan bantuan suatu *director*, yang bertugas untuk melakukan pemetaan IP dari *server* dan komputer lainnya yang berperan sebagai *virtual server*.

LVS *director* adalah modifikasi dari sistem Linux yang bertanggung jawab untuk mendistribusikan permintaan *user/client* terhadap *real server* pada kelompok *server*. *Real server* melakukan pekerjaan untuk memenuhi permintaan serta memberikan atau membuat laporan balik kepada *user/client*.

LVS *director* memelihara rekaman daripada sejumlah permintaan yang telah ditangani oleh masing-masing *real server* dan menggunakan informasi ini ketika memutuskan *server* mana yang akan ditugaskan untuk menangani suatu permintaan berikutnya.

LVS juga dapat memiliki *director* cadangan yang akan menggantikan apabila suatu saat *director* utama mengalami suatu kegagalan sehingga membentuk suatu LVS *failover*.

Masing-masing *real server* dapat bekerja dengan menggunakan berbagai sistem operasi dan aplikasi yang mendukung TCP/IP dan Ethernet. Pembatasan dan pemilihan sistem operasi pada *real server* serta jenis servis yang didukung oleh *real server* dilakukan pada saat proses konfigurasi LVS dijalankan [ALE-02].

Servis yang dapat didukung oleh LVS, antara lain:

- a. Website (HTTP, HTTPS)
- b. FTP (*File Transfer Protocol*)
- c. Email (SMTP, POP3, dan IMAP)
- d. News (NNTP)
- e. DNS (*Domain Name Service*)
- f. LDAP (*Lightweight Directory Access Protocol*)

2.5 Load Balancing

Load balancing adalah suatu proses dan teknologi yang mendistribusikan trafik situs diantara beberapa *server* dengan menggunakan perangkat berbasis jaringan. Proses ini mampu mengurangi beban kerja setiap *server* sehingga tidak ada *server* yang *overload*, memungkinkan ada beberapa *server* dalam satu URL.

Fungsi menggunakan *load balancer* diantaranya adalah:

- a. Merepresentasikan beberapa alamat untuk sebuah *site*.
- b. Membagi beban dan menentukan *server* yang harus menerima *request*.
- c. Memastikan *server* siap menerima *request*, jika tidak *request* dialihkan ke *server* lain yang siap.

Tiga manfaat utama secara langsung pada *website* dengan *traffic* yang sangat tinggi :

- a. *Flexibility*

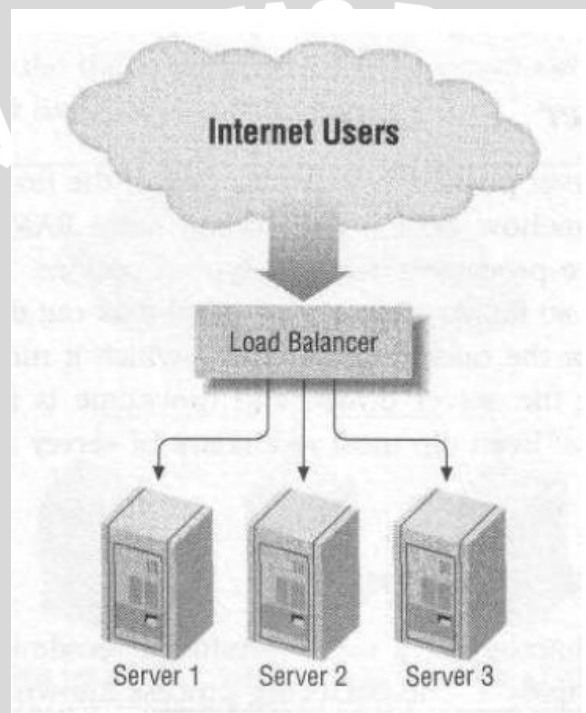
Dengan *load balancer* memungkinkan untuk menambah atau mengurangi *server* tanpa mengganggu *traffic* serta memudahkan untuk perawatan mesin yang dapat dilakukan sewaktu-waktu.

b. *High availability*

Load balancer secara terus-menerus melakukan pemantauan terhadap *server*. Jika terdapat *server* yang mati, maka *load balancer* akan menghentikan *request* ke *server* tersebut dan mengalihkannya ke *server* yang lain.

c. *Scalability*

Dapat mengatasi perubahan kebutuhan terhadap sumberdaya *server* secara efektif [BOU-01].



Gambar 2.8 Konsep Dasar *Server Load Balancing*

Sumber : [BOU-01:3]

Ada beberapa tipe *forwarding* pada *load balancing* sesuai dengan tipe *load balancing* yang digunakan. Pada Tabel 2.1 merupakan tipe *forwarding* pada sistem *load balancing*.

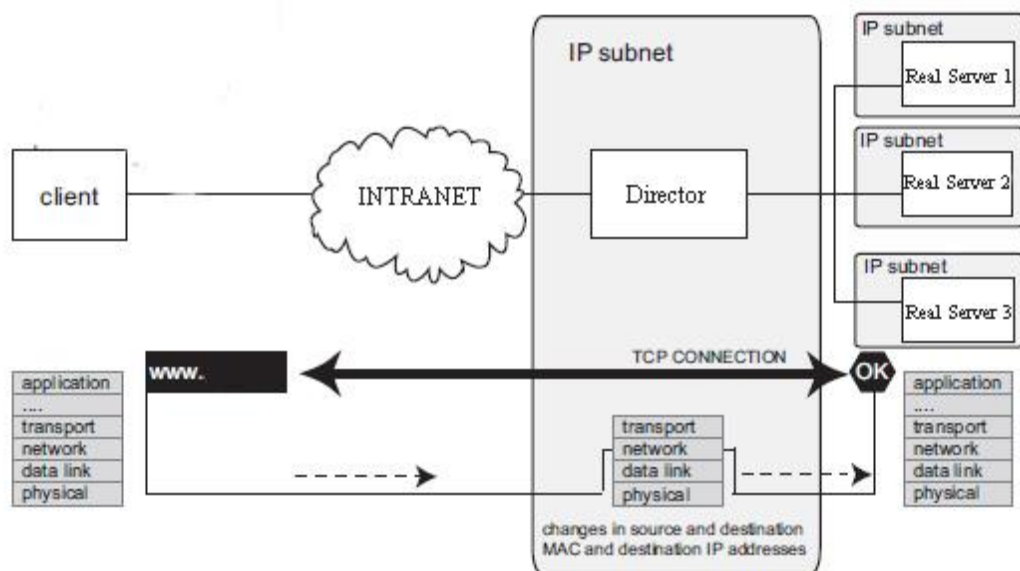
Tabel 2.1 Tipe *Forwarding*

	Layer-2 Forwarding	Layer-3 Forwarding
Two-Ways		Network Address Translation
One-Ways	Direct Routing	IP Tunneling

Sumber : Sierra [2009:14]

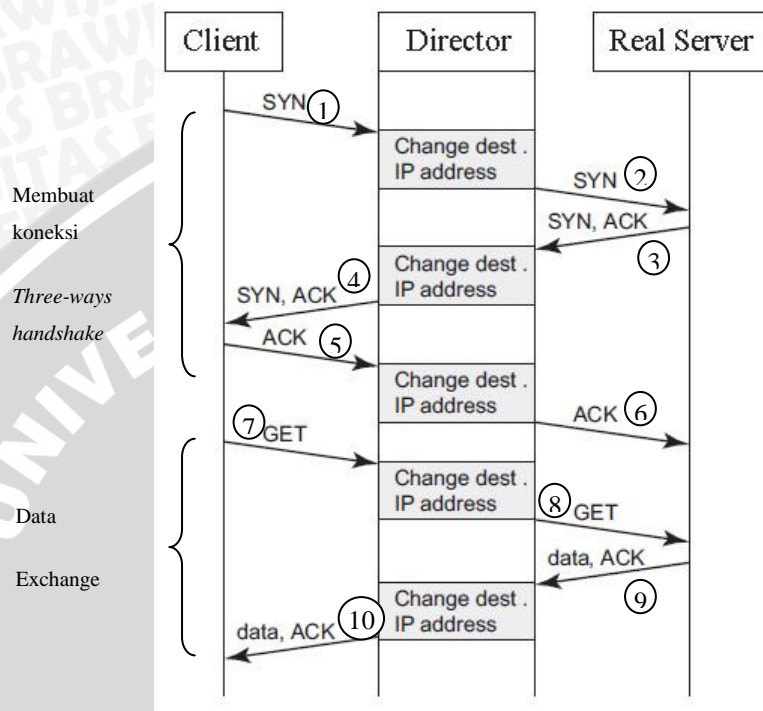
Terdapat arsitektur Two-Ways dan One-Way *forwarding* pada *load balancing*. Arsitektur Two-Ways adalah respon dari *server* ke *client* dapat melalui *load balancer (director)*. Pada arsitektur One-Way respon dari *server* mencari jalur alternatif langsung menuju *client* tanpa melewati *load balancer* [SIE-09].

Layer-2 *forwarding* adalah *forwarding* pada layer 2 OSI (Open Systems Interconnection) yaitu pada data *link layer*. Layer-3 *forwarding* adalah *forwarding* pada layer 3 OSI yaitu pada *network layer*. Sistem *load balancing* yang dibuat menggunakan Layer-3 *forwarding* karena jaringan menggunakan *Network Address Translation (NAT)*. Pada Gambar 2.9 merupakan contoh implementasi *Two-Ways Layer-3 forwarding*.



Gambar 2.9 Contoh Implementasi *Two-Ways Layer-3 Forwarding*
 Sumber : [SIE-09]

Pada Gambar 2.9 dilakukan pertukaran *source* dan *destination* MAC dan IP addresses karena *real server* berada pada NAT. Pada gambar 2.10 merupakan teknik komunikasi pada *load balancing* dengan menggunakan NAT.



Gambar 2.10 Three-Ways Handshake pada NAT (Layer 3 Forwarding)
Sumber : [SIE-09:17]

Pada Gambar 2.10 merupakan three-way handshake untuk membangun sebuah koneksi TCP pada jaringan dengan konfigurasi NAT. Pada langkah 1 sampai dengan 6 adalah untuk membangun sebuah koneksi TCP antara client dengan server dan langkah selanjutnya mulai melakukan pertukaran data.

2.6 Algoritma Penjadwalan

Dalam *load balancing* terdapat beberapa algoritma penjadwalan diantaranya :

a. Least-Connection

Least-connection mengalokasikan koneksi ke *real-Server* dengan jumlah koneksi paling sedikit [PIE-09].

b. *Weighted Least-Connection (wlc)*

Weighted Least-Connection (wlc) hampir sama dengan algoritma *least-connection*, namun ini algoritma yang mempertimbangkan bobot. Bobot diberikan kepada masing-masing *real server*. Untuk koneksi baru akan dipilih *server* yang paling sedikit kapasitasnya. Kapasitas dihitung dengan membagi bobot suatu *real server* dibagi dengan bobot seluruh *real server* yang terhubung dengan *virtual server* [CIS-07].

c. *Round-Robin (rr)*

Round-Robin (rr) menempatkan semua *real server* pada antrian yang melingkar dan mengalokasikan koneksi bergantian untuk setiap *turn* [PIE-09].

d. *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 [CIS-07].

e. *Locality-Based Least-Connection (lbc)*

Mendistribusikan permintaan lebih ke *server* dengan koneksi yang aktif lebih sedikit dibandingkan dengan IP tujuan mereka. Algoritma ini akan meneruskan semua *request* kepada *real server* yang memiliki koneksi kurang aktif tersebut sampai kapasitasnya terpenuhi.

f. *Locality-Based Least-Connection Scheduling with Replication*

Mendistribusikan permintaan lebih ke *server* dengan koneksi yang aktif lebih sedikit dibandingkan dengan IP tujuan. Algoritma ini juga dirancang untuk digunakan dalam sebuah *cluster server proxy-cache*. Ini berbeda dengan penjadwalan *Locality-Based Least-Connection*, dengan pemetaan alamat IP target untuk subset dari *node real server*. Permintaan ini kemudian diteruskan ke *server* dalam subset dengan jumlah koneksi paling sedikit. Jika semua *node* untuk IP tujuan di atas kapasitas, dilakukan replikasi *server* baru untuk alamat IP tujuan dengan menambahkan *real server* dengan koneksi yang sedikit dari keseluruhan *real server pool* untuk subset dari *real server* sebagai IP tujuan *request*. *Node* yang

bermuatan lebih dikeluarkan dari subset *real server* untuk mencegah over-replikasi [RED-07].

g. *Destination Hash Scheduling*

Menggunakan hash statis dari alamat IP tujuan untuk mengalokasikan koneksi [PIE-09].

h. *Source Hash Scheduling*

Mendistribusikan permintaan ke kumpulan *real server* dengan melihat sumber IP dalam tabel hash statis.

i. *Shortest Expected Delay*

Mengalokasikan koneksi ke *server* yang akan melayani permintaan dengan delay terpendek.

j. *Never Queue*

Mengalokasikan koneksi ke *idle real server* jika ada, yang lain menggunakan algoritma *Shortest Expected Delay* [PIE-09].

Dalam penelitian ini algoritma yang digunakan adalah *least connection* karena sistem yang dibuat menggunakan protokol TCP yang bersifat *connection oriented*. Algoritma *scheduling least connection* membagi beban berdasarkan dengan jumlah koneksi ke masing-masing *real server* dengan kapasitas yang sama sehingga sangat cocok untuk *server cluster* dengan kemampuan yang sama [RED-07].

Pseudocode algoritma *least connection* adalah sebagai berikut :

```
Misal server adalah S (S0, S1, S2, S3, ..., Sn-1).
W(Si) adalah weight untuk server Si, least connection weight =1.
C(Si) adalah koneksi yang sedang ditangani server Si.
CSUM =  $\sum C(S_i)$  (i=0, 1, ..., n-1) adalah jumlah seluruh koneksi
yang sedang ditangani.
Koneksi baru diberikan kepada server j dimana.
 $(C(S_m)/CSUM)/W(S_m) = \min\{(C(S_i)/CSUM)/W(S_i)\}$  (i=0,1,..., n-1)
Dimana W(Si) ≠ 0
CSUM dapat diabaikan karena memiliki nilai yang sama.
 $C(S_m)/W(S_m) = \min\{C(S_i)/W(S_i)\}$  (i=0,1, ..., n-1)
Dimana W(Si) ≠ 0

for (m = 0; m < n; m++) {
  if (W(Sm) > 0) {
    for (i = m+1; i < n; i++) {
      if (C(Sm)*W(Si) > C(Si)*W(Sm))
        m = i;
    }
  }
}
```

```
        return Sm;
    }
}
return NULL;
```

2.7 Parameter yang Dianalisis

Ada beberapa parameter yang akan dianalisis pada sistem *load balancing* dengan algoritma *least connection* diantaranya adalah *throughput*, *CPU utilization*, dan waktu respon.

a. *Throughput*

Throughput, yaitu kecepatan (*rate*) transfer data efektif, yang diukur dalam bps. *Throughput* merupakan jumlah total kedatangan paket yang sukses yang diamati pada *destination* selama interval waktu tertentu dibagi oleh durasi interval waktu tersebut (sama dengan jumlah pengiriman paket IP sukses per *service-second*)[RIZ-10].

b. *CPU utilization*

CPU utilization menunjukkan penggunaan prosesor untuk memproses suatu proses dalam sistem.

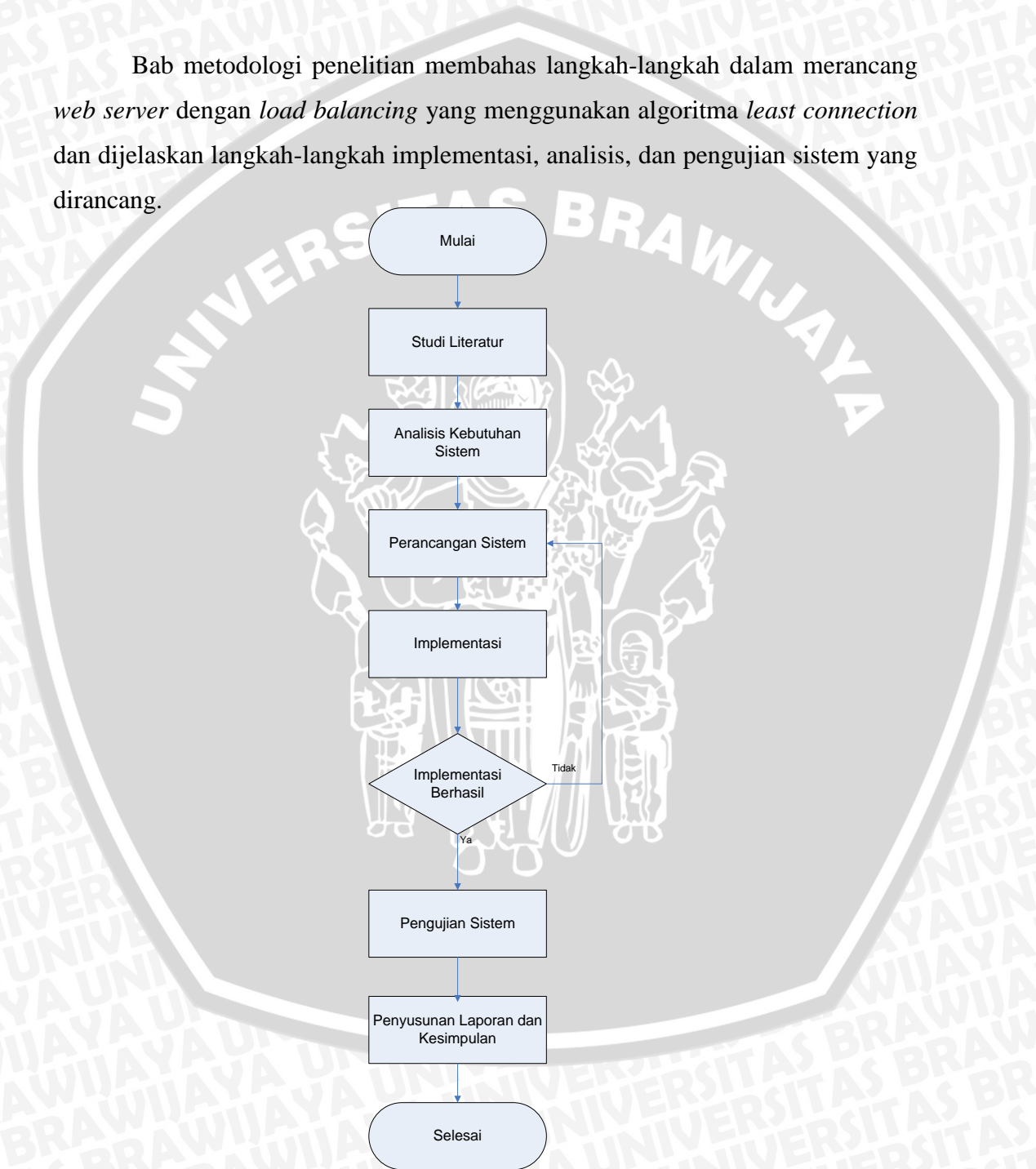
c. Waktu Respon

Waktu Respon menunjukkan durasi waktu suatu permintaan dilayani.

BAB III

METODOLOGI PENELITIAN

Bab metodologi penelitian membahas langkah-langkah dalam merancang *web server* dengan *load balancing* yang menggunakan algoritma *least connection* dan dijelaskan langkah-langkah implementasi, analisis, dan pengujian sistem yang dirancang.



Gambar 3.1 Diagram Alir Penyusunan Laporan Hasil Implementas



3.1 Studi Literatur

Studi literatur menjelaskan dasar teori yang mendukung dalam implementasi dan analisis *load balancing* pada *web server* dengan algoritma *least connection*. Teori-teori pendukung tersebut antara lain :

- a. Jaringan Komputer
 - Model Hubungan pada Jaringan Lokal (LAN)
- b. Protokol
 - Internet Protokol
 - *Tranmission Control Protocol*
 - Protokol HTTP
- c. *WebServer*
 - *Apace Web Server*
- d. Linux
 - Kernel Linux
- e. LVS (linux Virtual Server)
 - Piranha
- f. *Load Balancing*
- g. Algoritma penjadwalan
 - Algoritma *Least-Connection*
- h. Httperf

3.2 Analisis Kebutuhan

Analisis kebutuhan bertujuan untuk mendapatkan semua kebutuhan *sistem load balancing* pada *web server* dengan algoritma *least connection*. Pada analisis kebutuhan tindakan yang dilakukan yaitu mengidentifikasi perangkat lunak, perangkat keras serta topologi yang digunakan dalam perancangan, implementasi dan analisis sistem yang akan dibuat. Dengan demikian diharapkan dapat mempermudah dalam mendesain sistem dan hasil analisis terhadap sistem yang dibuat dapat lebih akurat.

3.2.1 Perangkat Keras yang Digunakan

Sistem *load balancing* dibuat dengan menggunakan tiga buah *real server* dan satu buah *director* dengan spesifikasi yang sama yaitu :

- Prosesor : Intel (R) Core (TM) 2 Quad CPU
8300 @ 2.50GHz
- RAM : 2 GB
- Kapasitas hardisk : 500 GB

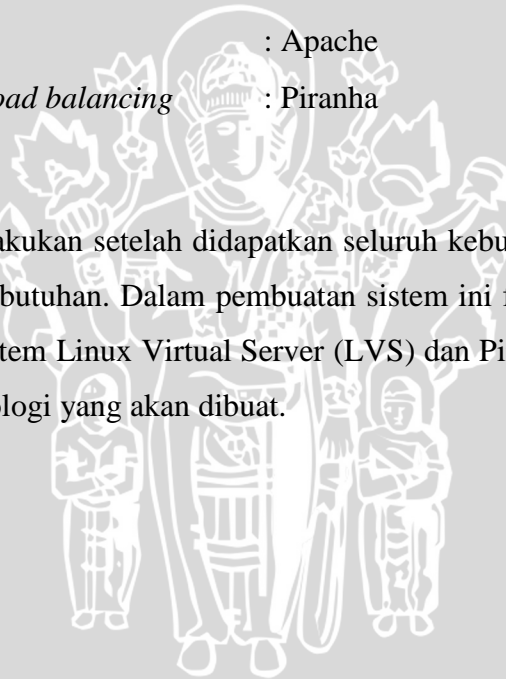
3.2.2 Perangkat Lunak yang digunakan

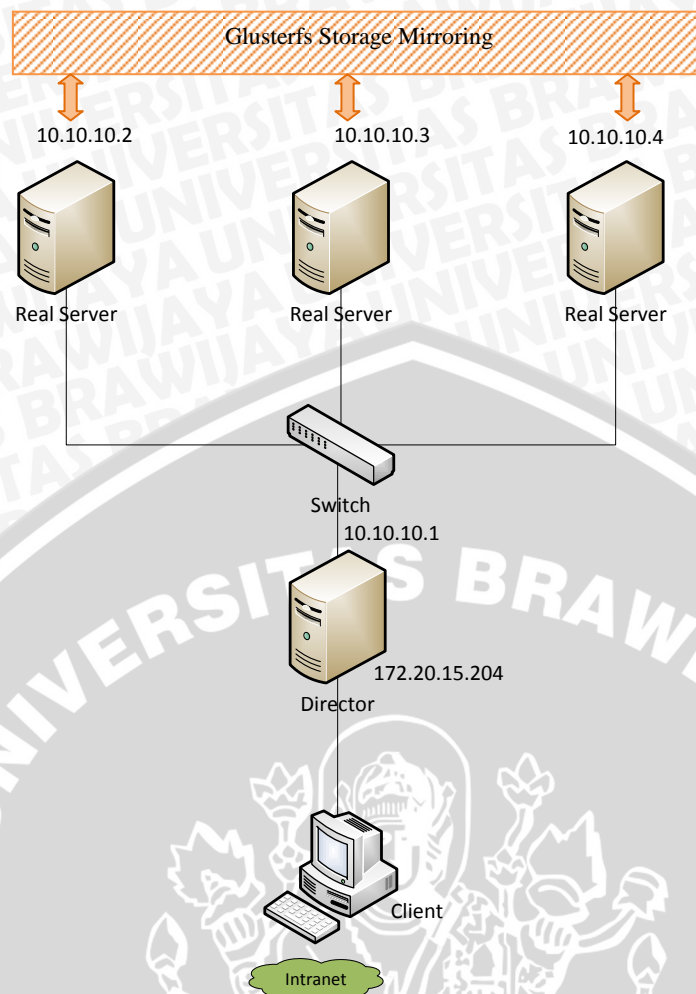
Perangkat lunak dan sistem operasi yang digunakan adalah sebagai berikut:

- Sistem Operasi : Centos 5.7
- Web Server : Apache
- Perangkat *Load balancing* : Piranha

3.3 Perancangan

Perancangan dilakukan setelah didapatkan seluruh kebutuhan sistem yang didapat pada analisis kebutuhan. Dalam pembuatan sistem ini fokus perancangan adalah pada topologi sistem Linux Virtual Server (LVS) dan Piranha. Gambar 3.2 merupakan rencana topologi yang akan dibuat.





Gambar 3.2 Topologi LVS

Dalam perancangan akan dijelaskan lebih lanjut tugas serta spesifikasi masing-masing, dengan demikian akan mempermudah konfigurasi masing-masing *server* pada tahap implementasi.

3.4 Implementasi

Implementasi dilakukan mengacu pada topologi dalam perancangan. Pada tahap ini ada beberapa langkah yang harus dilakukan antara lain :

3.4.1 Instalasi dan Konfigurasi Perangkat Keras

Instalasi dan konfigurasi perangkat keras yang dilakukan antara lain *cabling* yang menggunakan kabel UTP dengan metode *cross* (menyilang) dan *straight* (lurus) yang berfungsi untuk menghubungkan *server web* dengan *switch* atau *server director* dengan klien, setelah semua perangkat terhubung dalam sistem dilakukan pengalamatan IP untuk masing-masing perangkat.

3.4.2 Instalasi dan Konfigurasi Perangkat Lunak

Sistem ini menggunakan sistem operasi Centos 5.6. Pada bagian ini ada beberapa konfigurasi yang akan dilakukan antara lain kompilasi *kernel 2.6.x.x*, mengaktifkan fitur-fitur pendukung LVS, menginstal *ipvsadm*, meng-install Piranha. Setelah layanan untuk membangun LVS terinstal, langkah berikutnya adalah instalasi Apache *web server* pada *server web*. *Mirroring* data pada tiap *real server* digunakan *tool* GlusterFS.

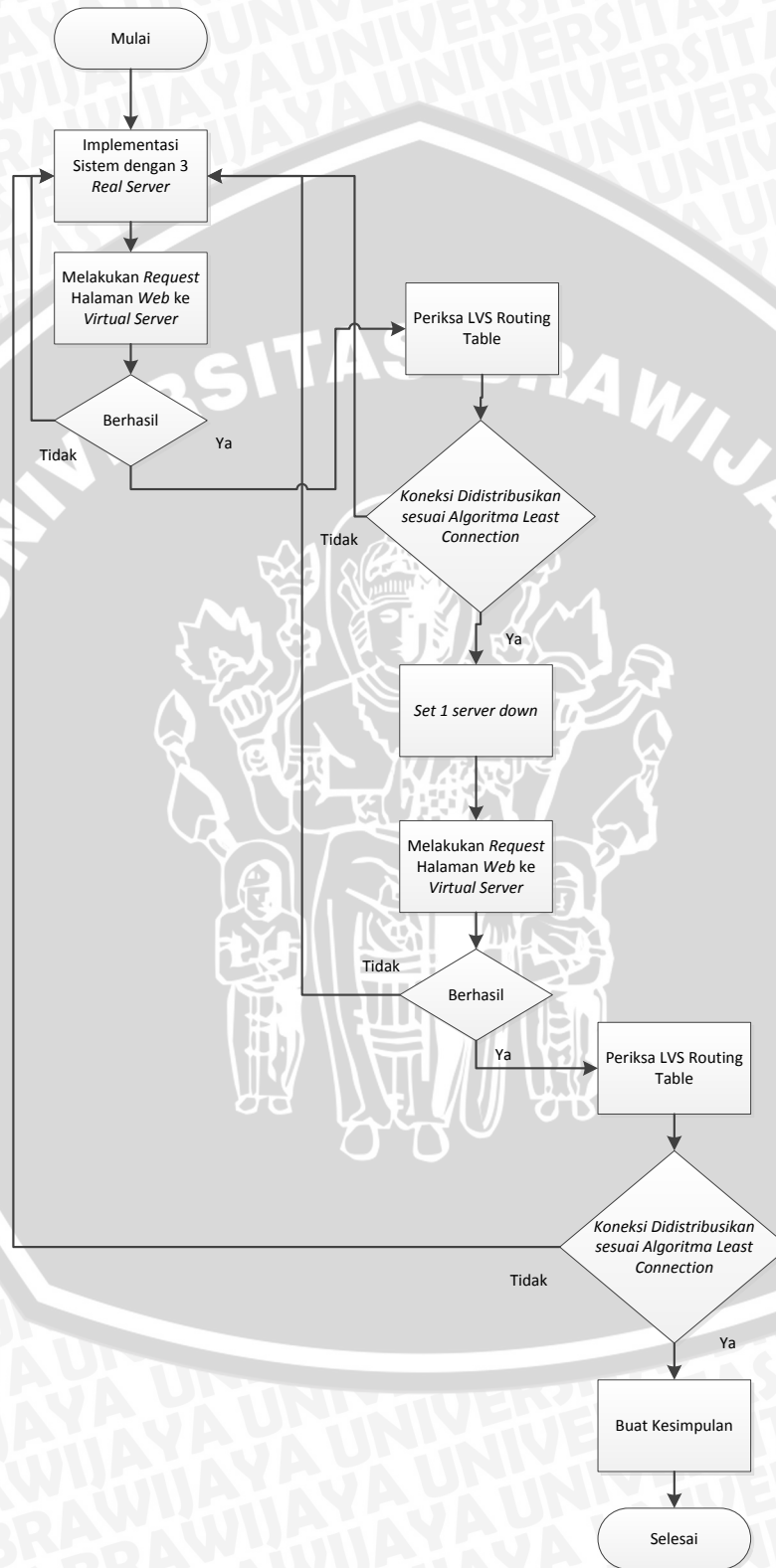
3.5 Pengujian dan Analisis

Pada tahap pengujian dan analisis dilakukan pengujian terhadap sistem dengan jumlah koneksi dan jumlah *real server* yang berbeda-beda serta menganalisis sistem sesuai parameter yang telah ditentukan meliputi *throughput*, *CPU utilization*, dan waktu respon. Pengujian yang dilakukan meliputi pengujian kinerja (*high availability*) dan pengujian performa.



3.5.1 Pengujian Kinerja (*High Availability*)

Pada pengujian kinerja sistem akan diuji sesuai dengan diagram berikut.



Gambar 3.3 Diagram Alir Pengujian Kinerja (*High Availability*)

Keberhasilan pengujian didapat ketika *request* http tetap dilayani walaupun beberapa *server* dalam sistem *load balancing* mengalami kegagalan (*High Availability*). *High Availibility* merupakan salah satu keunggulan sistem dengan *load balancing* dibanding dengan penggunaan *single server*.

3.5.2 Pengujian Performa.

Dalam pengujian performa akan didapatkan data berupa parameter-parameter seperti *throughput*, *CPU utilization*, dan waktu respon. Kemudian data parameter performa dari *single web server* akan dibandingkan dengan sistem *load balancing* yang menggunakan jumlah *real server* yang berbeda.

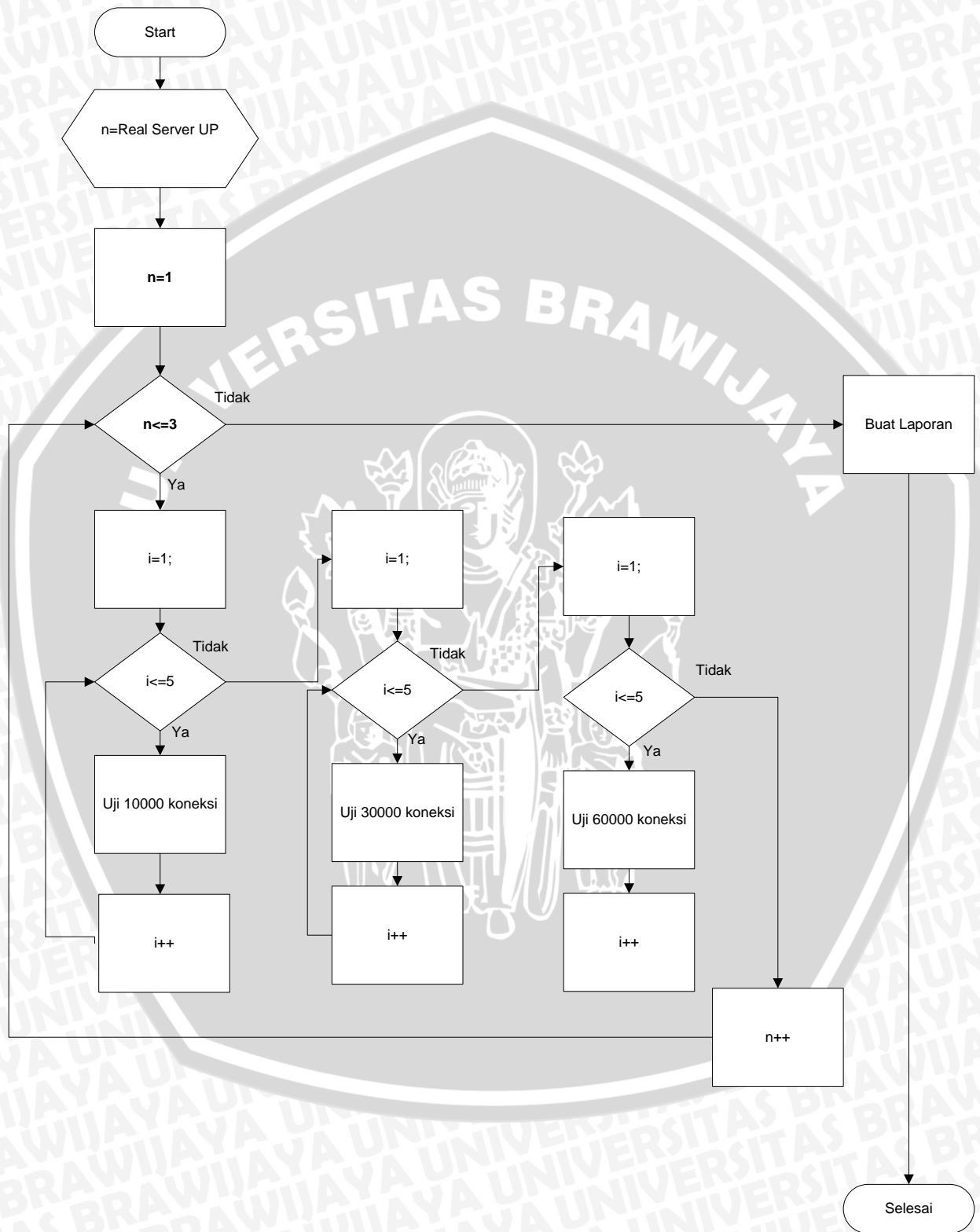
Untuk pengujian performa digunakan perangkat lunak *httperf*. *Httperf* adalah *tool* yang digunakan untuk mengukur kinerja dari sebuah *web server* [MOS-02]. Dengan menggunakan *httperf* dapat menciptakan banyak *request* dan koneksi dalam satu waktu, sehingga dalam pengujian sistem LVS yang dirancang ini tidak membutuhkan banyak komputer.

Berikut ini adalah contoh dari penggunaan *httperf* :

```
httperf --Server hostname --port 80 --uri /test.html --rate 1000 --num-conn 30000 -  
-num-call 1 --timeout 5
```

Pada *command line* diatas *httperf* menggunakan *web server* pada *host* dengan nama IP *host name*, berjalan pada port 80. Halaman *web* yang diambil adalah *"/ test html."*, halaman yang sama diambil beberapa kali dengan jumlah *request* sampai 1000, melibatkan koneksi 30000 TCP dan pada masing-masing koneksi dilakukan pemanggilan HTTP (pemanggilan berisi pengiriman *request* dan menerima balasan dari *request* yang dilakukan). Opsi *timeout* menentukan lamanya klien bersedia menunggu balasan dari *server*. Jika waktu melebihi batas, panggilan dinyatakan gagal [MOS-02].

Gambar 3.4 Merupakan diagram alir pengujian performa terhadap sistem.

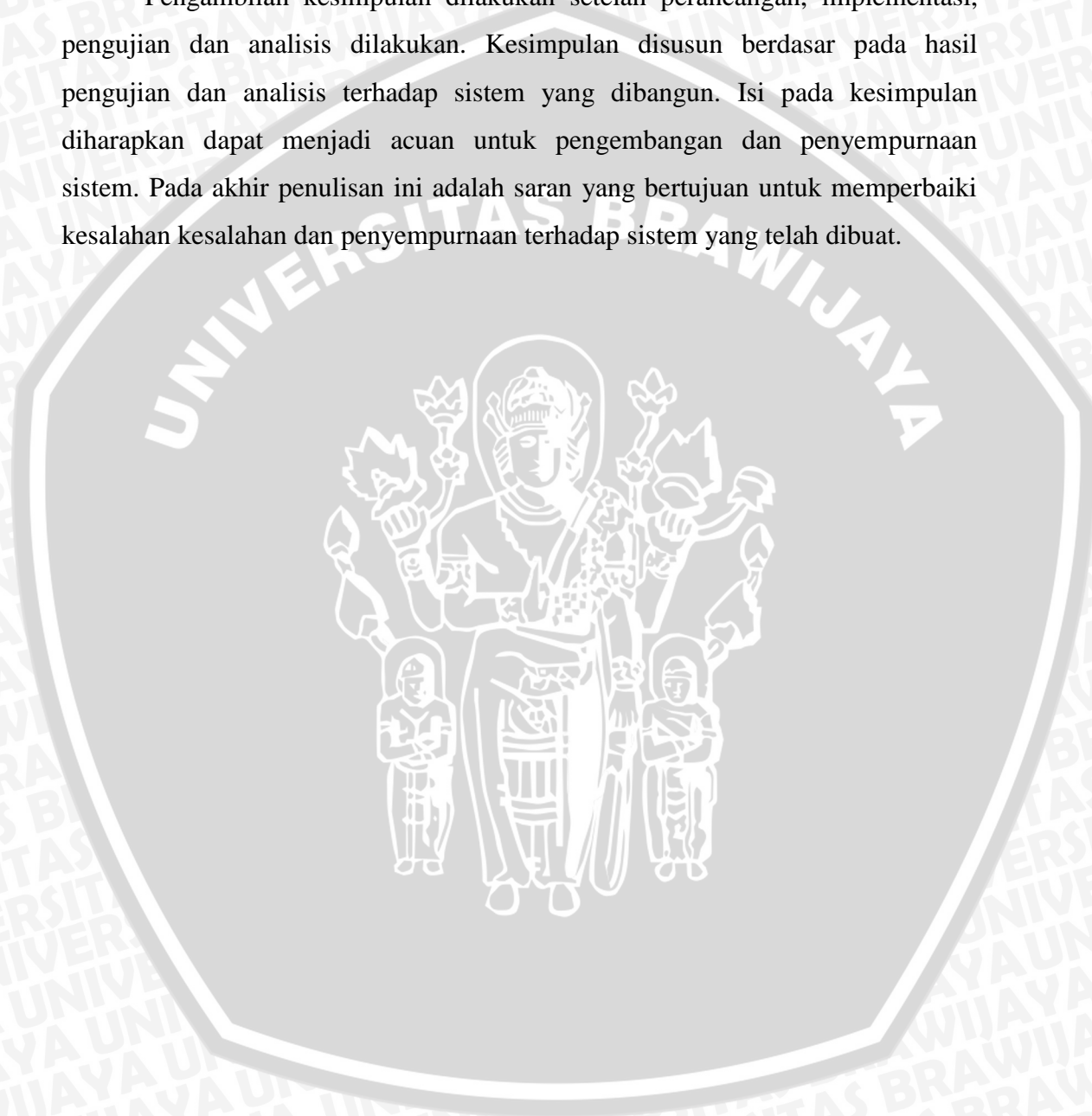


Gambar 3.4 Diagram Alir Pengujian Performa.

Pengujian performa dilakukan dengan memberikan beban koneksi terhadap sistem dengan jumlah koneksi sebesar 10000, 30000, dan 60000 koneksi.

3.6 Pengambilan Kesimpulan

Pengambilan kesimpulan dilakukan setelah perancangan, implementasi, pengujian dan analisis dilakukan. Kesimpulan disusun berdasar pada hasil pengujian dan analisis terhadap sistem yang dibangun. Isi pada kesimpulan diharapkan dapat menjadi acuan untuk pengembangan dan penyempurnaan sistem. Pada akhir penulisan ini adalah saran yang bertujuan untuk memperbaiki kesalahan kesalahan dan penyempurnaan terhadap sistem yang telah dibuat.



BAB IV

PERANCANGAN

Pada bab perancangan ini akan dijelaskan secara rinci rancangan sistem *load balancing* pada *web server* dengan menggunakan algoritma *least connection*. Beberapa hal yang dibahas meliputi analisis kebutuhan, desain dan rancangan sistem.

4.1 Analisis Kebutuhan

Pada analisis kebutuhan dibahas komponen-komponen yang digunakan untuk membangun sistem dan mendeskripsikan fungsi-fungsi dari sistem yang dibuat. Komponen-komponen yang dibahas dalam analisis kebutuhan sistem LVS meliputi *software* dan *hardware* yang digunakan.

4.1.1 System Requirements

Dalam *system requirements* dibahas komponen-komponen yang diimplementasikan pada sistem *load balancing* pada *web server* serta dijelaskan dengan rinci fungsi dan layanan dari sistem *load balancing* pada *web server*.

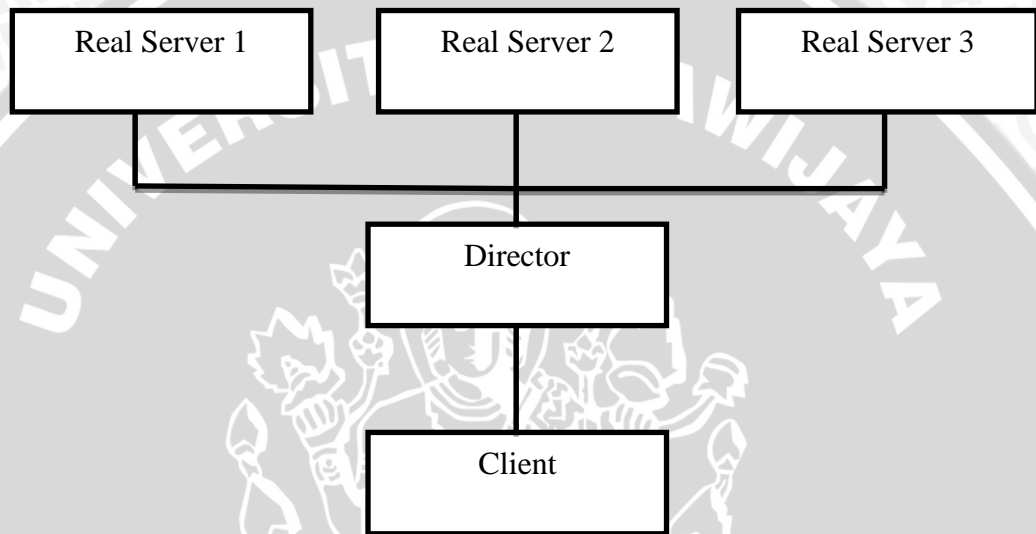
4.1.1.1 Sistem Linux Virtual Server (LVS)

Linux Virtual Server (LVS) merupakan suatu teknologi *clustering* yang dapat digunakan untuk membangun suatu *server* dengan menggunakan kumpulan dari beberapa *real server*. *Load balancing* dengan algoritma *least connection* dapat diterapkan pada sistem Linux Virtual Server. Sistem *load balancing* yang dibuat memiliki karakteristik dan fungsi sebagai berikut :

1. Fungsi sistem *Load Balancing* pada *web server* dengan Algoritma *Least Connection* adalah menciptakan sebuah *web server* dalam sebuah *cluster* komputer. *Web* dapat diakses melalui *IP director*.
2. Sistem *load balancing* dibuat pada sistem jaringan yang menggunakan IPV4.
3. *High availability*, Ketika ada beberapa *real server* yang mati sistem tetap dapat berjalan sesuai fungsinya.

4. *Reliability*, sistem *Load Balancing* pada *web server* dengan Algoritma *Least Connection* akan bekerja lebih baik daripada *single web server* ditinjau dari parameter-parameter seperti, *throughput*, *CPU utilization*, dan waktu respon.

Pada Gambar 4.1, merupakan beberapa *node* dalam sistem LVS. Setiap *node* memiliki fungsi yang berbeda yang akan dijelaskan sebagai berikut :



Gambar 4.1 Diagram Topologi LVS

1. *Client*

Pada sebuah sistem *web server*, *client* adalah *user* yang mengakses *website* dalam *web server* tersebut. *Client* dapat berasal dari *network* dan lokasi yang berbeda. Hal ini menyebabkan *client* sebuah *service web server* sangat banyak. Oleh karena itu dibutuhkan sistem pembagian beban dalam *cluster* komputer agar tidak terjadi *over load* pada *web server*.

2. *Director*

Client yang melakukan *request website* akan mengakses IP address dari *director*. *Director* dalam LVS akan melanjutkan *request* dari *client* ke *real server* yang memiliki data *website* yang diminta oleh *client*. *Director* tidak memiliki data *website* dan berfungsi sama seperti *router* yaitu meneruskan

paket data ke alamat yang dituju. Oleh karena itu *director* disebut dengan *virtual server*.

3. Real Server

Real Server adalah *server* yang mempunyai data *website* yang sebenarnya. *Real Server* terdiri lebih dari satu *machine*. Beban *request* dari *client* akan dibagikan kepada seluruh *real server* yang aktif. Jumlah *real server* yang lebih dari satu menyebabkan sistem lebih *high availability* karena apabila ada satu *real server* yang mati *request* dari *client* tetap dapat dilayani. Pembagian beban kepada *real server* ditentukan dengan penggunaan algoritma penjadwalan.

4.1.2.2 Analisis Kebutuhan Software

Untuk membangun sistem *load balancing* pada *web server* digunakan sistem operasi dan aplikasi sebagai berikut :

1. Sistem Operasi

Sistem operasi yang digunakan dalam pembuatan sistem *load balancing* pada *web server* ini adalah Centos 5.6. Centos 5.6 dengan kernel 2.6.18 digunakan pada *director* dan *real server*. Sistem operasi linux yang mempunyai kernel dibawah 2.4.28 belum mendukung *virtual server* sehingga tidak dapat digunakan dalam membuat sistem *load balancing*.

2. Software Load Balancing (Piranha)

Sistem *load balancing* yang dibuat menggunakan perangkat pendukung Linux Virtual Server (LVS) dari Redhat yaitu Piranha. Piranha adalah perangkat *monitoring cluster* dalam LVS dan dapat pula digunakan untuk konfigurasi *director* serta *real server* pada sistem LVS. Pada Piranha terdapat *ipvsadm* (IP Virtual Server Administration) yang mengatur kerja dari *director*. Pada *ipvsadm* dapat diterapkan fungsi-fungsi yang mengatur kerja *director* termasuk juga algoritma penjadwalan. Ketika *director* mendapat sebuah *request*, *director* akan meneruskan *request* ke *real server* sesuai dengan algoritma yang digunakan.

3. *Software Web Server*

Untuk membuat *web server* digunakan Apache *web server*. Apache *web server* memiliki banyak program pendukung sehingga dapat memberikan layanan yang cukup bagi penggunaannya. Berikut ini adalah beberapa program pendukung dari Apache *web server*, antara lain :

- a. Kontrol akses,
- b. *Common Gateway Interface* (CGI) yang paling terkenal dan sangat sering digunakan adalah Perl (*Practical extraction and report language*),
- c. PHP (*Personal Home Page*), dan
- d. SSI (*Server Side Include*) [SYA-03].

4.1.2.3 Analisis Kebutuhan *Hardware*

Hardware yang digunakan untuk membuat sistem *Load Balancing* pada *Web Server* antara lain, *Central Processing Unit* atau CPU (*director* dan *real server*), *ethernet card*, kabel UTP dan switch.

1. CPU untuk *Director* dan *Real Server*

Untuk spesifikasi satu buah *director* dan tiga buah *real server* yang digunakan adalah sama dengan rincian sebagai berikut :

- Prosesor : Intel (R) Core (TM) 2 Quad CPU 8300 @ 2.50GHz
- Memori : 2 GB
- Kapasitas Hardisk : 500 GB

2. *Ethernet Card*

Ethernet Card merupakan perangkat pada *physical layer* pada OSI layer. *Ethernet Card* yang digunakan adalah Realtek Semiconductor Co., Ltd. RTL-8169 gigabit ethernet.

3. *Switch*

Switch merupakan perangkat pada layer *datalink*. *Switch* dapat menangani beberapa sambungan sekaligus pada saat yang sama. Tiap-tiap port 100 Base-TX pada sebuah *switch* dapat mengirim dan menerima beberapa *frame* secara bersamaan (*Full-Duplex*).

4.2 Algoritma Penjadwalan *Least Connection*

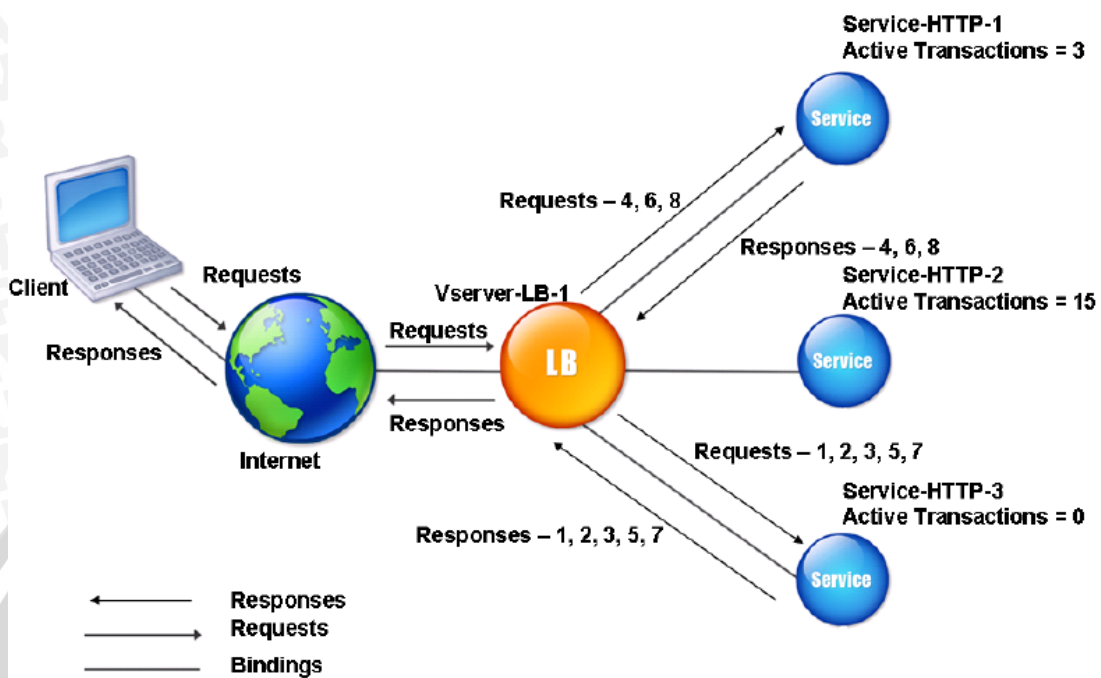
Dalam penelitian ini algoritma yang digunakan adalah *least connection* karena sistem yang dibuat menggunakan protokol TCP yang bersifat *connection oriented*. Algoritma *scheduling least connection* membagi beban berdasarkan dengan jumlah koneksi ke masing-masing *real server* dengan kapasitas yang sama sehingga sangat cocok untuk server *cluster* dengan kemampuan yang sama [RED-07].

Pseudocode algoritma *least connection* adalah sebagai berikut :

```
Misal server adalah S (S0, S1, S1, S3, ..., Sn-1).
W(Si) adalah weight untuk server Si, least connection weight =1.
C(Si) adalah koneksi yang sedang ditangani server Si.
CSUM =  $\sum C(Si)$  (i=0, 1, .. , n-1) adalah jumlah seluruh koneksi
yang sedang ditangani.
Koneksi baru diberikan kepada server j dimana.
 $(C(Sm)/CSUM)/W(Sm) = \min\{(C(Si)/CSUM)/W(Si)\}$  (i=0,1,..., n-1)
Dimana W(Si) ≠ 0
CSUM dapat diabaikan karena memiliki nilai yang sama.
 $C(Sm)/W(Sm) = \min\{C(Si)/W(Si)\}$  (i=0,1, ..., n-1)
Dimana W(Si) ≠ 0

for (m = 0; m < n; m++) {
  if (W(Sm) > 0) {
    for (i = m+1; i < n; i++) {
      if (C(Sm)*W(Si) > C(Si)*W(Sm))
        m = i;
    }
    return Sm;
  }
}
return NULL;
```

Ketika *virtual server* dikonfigurasi untuk menggunakan algoritma *Least-Connection*, akan memilih layanan dengan koneksi aktif paling sedikit [CIT-10].



Gambar 4.2 Mekanisme Algoritma *Least Connection*
Sumber : [CIT-10]

Pada Gambar 4.2 dapat dilihat ada tiga layanan ,antara lain:

- Service-HTTP-1* yang menangani 3 transaksi aktif,
- Service-HTTP-2* yang menangani 15 transaksi aktif,
- Service-HTTP-3* tidak menangani transaksi aktif.

Pada Gambar 4.2, *virtual server* memilih layanan untuk setiap koneksi yang masuk dengan memilih *server* yang memiliki koneksi aktif paling sedikit.

Koneksi diteruskan sebagai berikut:

- Service-HTTP-3* menerima *request* yang pertama karena tidak sedang menangani transaksi yang aktif.
- Service-HTTP-3* menerima *request* kedua dan ketiga karena memiliki jumlah transaksi aktif paling sedikit.
- Service-HTTP-1* menerima *request* ke-empat karena *service-HTTP-1* dan *Service-HTTP-3* memiliki jumlah transaksi aktif yang sama. Pada kasus seperti ini, pemilihan layanan dapat ditentukan dengan menggunakan algoritma *round robin*.

- d. *Service-HTTP-3* menerima *request* kelima.
- e. *Service-HTTP-1* menerima *request* ke-enam, begitu seterusnya sampai jumlah transaksi aktif untuk ketiga *service-HTTP* sama. Ketika sama pemilihan *service-HTTP* yang akan menerima *request* dapat dilakukan dengan algoritma *round robin* [CIT-10].

Tabel 4.1, menjelaskan koneksi didistribusikan ke tiga *Service-HTTP*.

Tabel 4.1 Distribusi Koneksi

Koneksi Baru	<i>Service</i> yang dipilih	Jumlah Koneksi yang Aktif	Keterangan
<i>Request-1</i>	<i>Service-HTTP-3</i> (N=0)	1	<i>Service-HTTP-3</i> memiliki jumlah koneksi aktif paling sedikit.
<i>Request-2</i>	<i>Service-HTTP-3</i> (N=1)	2	
<i>Request-3</i>	<i>Service-HTTP-3</i> (N=2)	3	
<i>Request-4</i>	<i>Service-HTTP-1</i> (N=3)	4	<i>Service-HTTP-1</i> dan <i>Service-HTTP-3</i> memiliki jumlah koneksi yang sama.
<i>Request-5</i>	<i>Service-HTTP-3</i> (N=3)	4	
<i>Request-6</i>	<i>Service-HTTP-1</i> (N=4)	5	
<i>Request-7</i>	<i>Service-HTTP-3</i> (N=4)	5	
<i>Request-8</i>	<i>Service-HTTP-1</i> (N=5)	6	
<i>Service-HTTP-2</i> dipilih ketika telah menyelesaikan koneksi aktifnya atau <i>Service-HTTP-1</i> dan <i>Service-HTTP-3</i> memiliki jumlah koneksi aktif yang sama dengan <i>Service-HTTP-2</i> .			

Sumber: [CIT-10]

BAB V

IMPLEMENTASI

Pada bab implementasi akan dibahas langkah-langkah pembuatan sistem *load balancing* pada *web server*. Sesuai dengan rancangan pada bab IV sistem *load balancing* pada *web server* ini dibuat dengan topologi *star*. Sistem *web server* dengan *load balancing* yang dibuat menggunakan satu buah *director* dan tiga buah *real server*.

5.1 Proses Instalasi pada *Real Server*

Real server yang digunakan pada penelitian ini sebanyak tiga buah. Spesifikasi CPU tiap-tiap *real server* adalah sebagai berikut :

- Prosesor : Intel (R) Core (TM) 2 Quad CPU 8300 @ 2.50GHz
- Memori : 2 GB
- Kapasitas Hardisk : 500 GB

Konfigurasi IP *address* masing-masing *real server* diisi dengan IP *address* 10.10.10.2, 10.10.10.3, dan 10.10.10.4. *Gateway* diberi IP *address* 10.10.10.1. *Gateway* merupakan IP *address* dari *interface* yang terhubung dengan *director* (*virtual server*). Setelah dilakukan konfigurasi IP *address* diberikan nama *host* untuk setiap *real server*. Nama *host* IP *address* 10.10.10.2 adalah s1.widhi.web.id, nama *host* IP *address* 10.10.10.3 adalah s2.widhi.web.id, nama *host* IP *address* 10.10.10.4 adalah s3.widhi.web.id.

Real server adalah *server* yang sesungguhnya yang memiliki data dari halaman *website*. Aplikasi yang digunakan untuk membangun *web server* adalah Apache. Apache diinstal pada ketiga *real server*. Versi Apache yang di-install adalah Apache 2.2.3.

Letak *default* konten *website* pada Apache adalah pada *directory* /var/www/html. *Directory website* dipindah pada *directory* /var/www/html/widhi.web.id/ dengan melakukan konfigurasi *virtual host* pada

httpd.conf. Dengan *virtual host* sebuah IP *address* publik dapat digunakan untuk beberapa *website*.

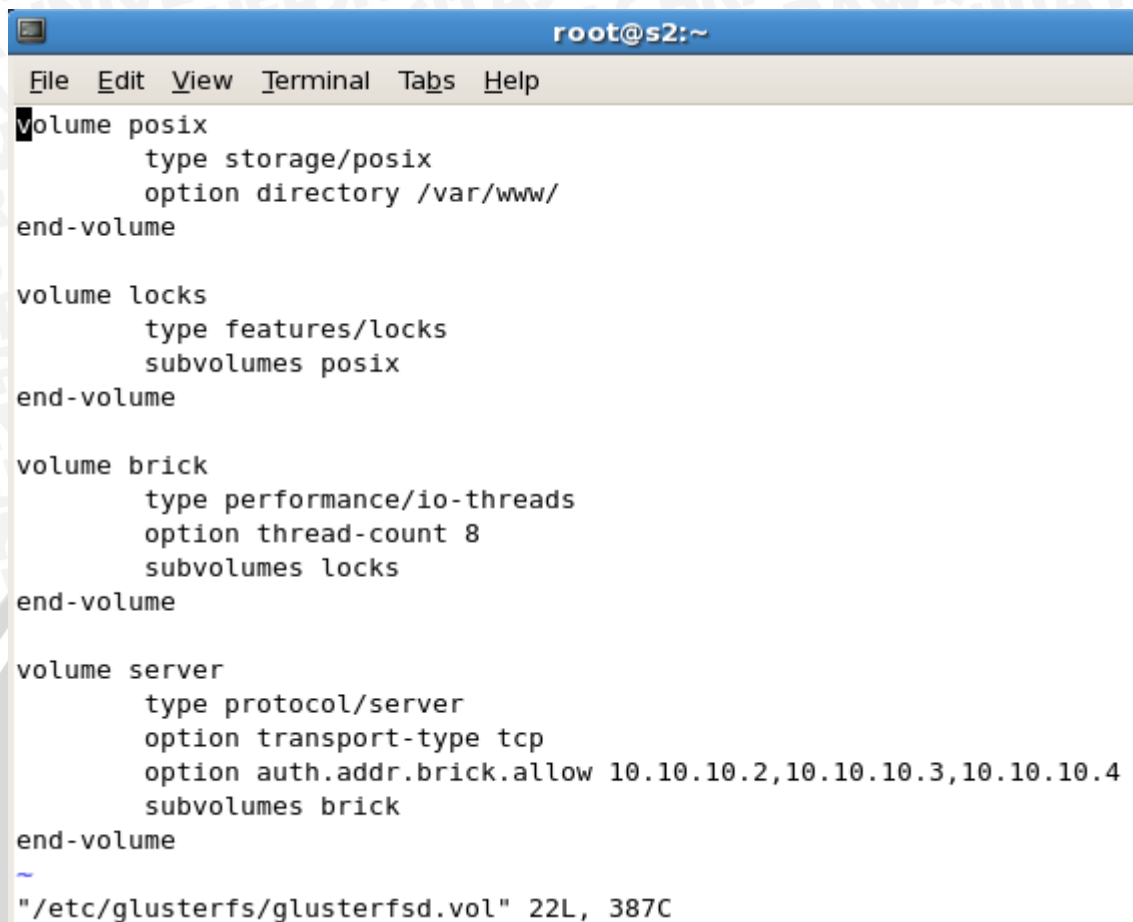
5.2 Proses Instalasi GlusterFS

GlusterFS digunakan untuk replikasi data atau *mirroring* antar ketiga *real server*. Pada proses instalasi GlusterFS, 2 *real server* dijadikan *server* GlusterFS dan 1 *real server* dijadikan *client* GlusterFS. Data akan direplikasikan ke ketiga *real server* tersebut.

5.2.1 Proses Instalasi Server GlusterFS

Server GlusterFS merupakan *server* yang memiliki *directory* yang akan direplikasi oleh *client*. *Server* GlusterFS di-*install* pada *real server* dengan IP *address* 10.10.10.2 dan 10.10.10.3. GlusterFS tidak bisa secara langsung di-*install* pada Centos. Persiapan yang dilakukan sebelum meng-*install server* GlusterFS adalah meng-*install* Development Tools, Development Libraries, libibverbs-devel, dan fuse-devel. Glusterfs yang di-*install* adalah GlusterFS versi 3.1.3. Kemudian dibuat konfigurasi *file* untuk *volume* yang direplikasi dengan nama glusterfsd.vol pada *folder* /etc/glusterfs/. Gambar 5.1 merupakan isi dari glusterfsd.vol yang dibuat.





```
root@s2:~
File Edit View Terminal Tabs Help
volume posix
    type storage/posix
    option directory /var/www/
end-volume

volume locks
    type features/locks
    subvolumes posix
end-volume

volume brick
    type performance/io-threads
    option thread-count 8
    subvolumes locks
end-volume

volume server
    type protocol/server
    option transport-type tcp
    option auth.addr.brick.allow 10.10.10.2,10.10.10.3,10.10.10.4
    subvolumes brick
end-volume
~
"/etc/glusterfs/glusterfsd.vol" 22L, 387C
```

Gambar 5.1 Isi Glusterfsd.vol.

Isi konfigurasi di atas sistem akan mereplikasi data yang pada *directory* /var/www/ (*brick*) dan memperbolehkan replikasi pada *server* dengan IP-address 10.10.10.2, 10.10.10.3, 10.10.10.4. Pada konfigurasi glusterfsd.vol diaktifkan fitur *lock*. Berkas yang dikunci (*locked*) adalah berkas yang terdapat klien yang diberikan hak akses eksklusif untuk melakukan pembacaan dan penulisan terhadapnya.

5.2.2 Proses Instalasi Client GlusterFS

Proses instalasi *client* GlusterFS tidak berbeda dengan instalasi *server* GlusterFS. *Client* GlusterFS di-*install* pada *host* s3.widhi.web.id. Langkah awal yang dilakukan adalah meng-*install* Development Tools, Development Libraries, libibverbs-devel, dan fuse-devel. GlusterFS yang telah di-*install* adalah Glusterfs

versi 3.1.3. Langkah selanjutnya adalah membuat file konfigurasi *volume* dengan nama *glusterfs.vol*. Gambar 5.2 merupakan skrip yang dibuat pada *glusterfs.vol*.

```

root@s3:~
File Edit View Terminal Tabs Help
volume remotel
  type protocol/client
  option transport-type tcp
  option remote-host s1.widhi.web.id
  option remote-subvolume brick
end-volume

volume remote2
  type protocol/client
  option transport-type tcp
  option remote-host s2.widhi.web.id
  option remote-subvolume brick
end-volume

volume replicate
  type cluster/replicate
  subvolumes remotel remote2
end-volume

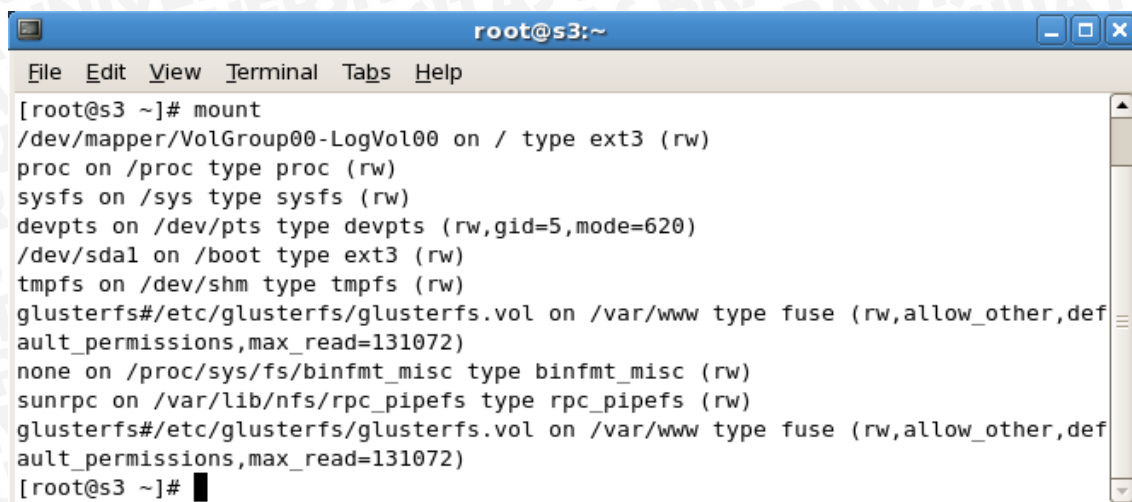
volume writebehind
  type performance/write-behind
  option window-size 1MB
  subvolumes replicate
end-volume

volume cache
  type performance/io-chace
  option cache-size 512MB
  subvolumes writebehind
end-volume

```

Gambar 5.2 Isi Glusterfs.vol.

Volume *remote* adalah *host* yang akan direplikasi oleh *client*. Volume yang direplikasi adalah *remotel* (*s1.widhi.web.id*) dan *remote2* (*s2.widhi.web.id*). Gluster *file system* pada *client* di-mount ke *directory* */var/www/ client*. Volume *replicate* memiliki fungsi yang sama seperti RAID-1. *Write-behind* meningkatkan kinerja *write* secara signifikan saat membaca dengan menggunakan teknik *aggregated background write*. Artinya, beberapa operasi menulis yang lebih kecil dikumpulkan menjadi sedikit operasi menulis yang lebih besar dan ditulis pada proses *background* sehingga jumlah paket yang ditulis lebih sedikit.



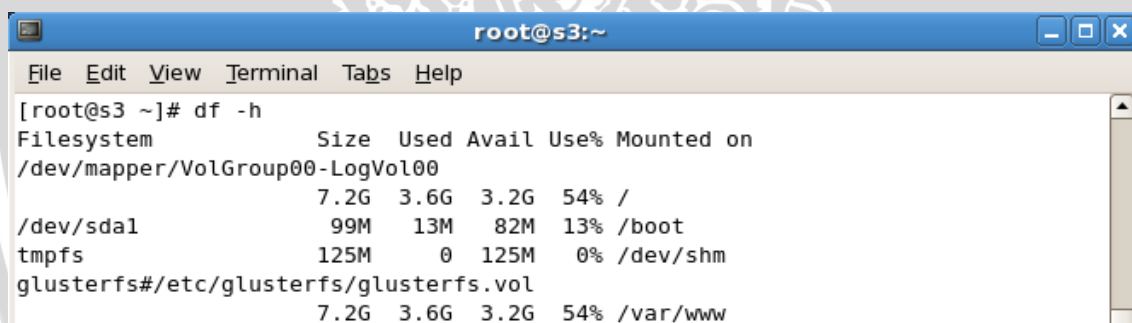
```

root@s3:~
File Edit View Terminal Tabs Help
[root@s3 ~]# mount
/dev/mapper/VolGroup00-LogVol00 on / type ext3 (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
/dev/sda1 on /boot type ext3 (rw)
tmpfs on /dev/shm type tmpfs (rw)
glusterfs#/etc/glusterfs/glusterfs.vol on /var/www type fuse (rw,allow_other,default_permissions,max_read=131072)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
sunrpc on /var/lib/nfs/rpc_pipefs type rpc_pipefs (rw)
glusterfs#/etc/glusterfs/glusterfs.vol on /var/www type fuse (rw,allow_other,default_permissions,max_read=131072)
[root@s3 ~]#

```

Gambar 5.3 Hasil *Mount* GlusterFS ke /var/www

Pada baris terakhir menunjukkan *mounting* GlusterFS *file system* telah berhasil. Hasil *mounting* GlusterFS *file system* dapat dilihat pada Gambar 5.4 berikut.



```

root@s3:~
File Edit View Terminal Tabs Help
[root@s3 ~]# df -h
Filesystem              Size  Used Avail Use% Mounted on
/dev/mapper/VolGroup00-LogVol00
                        7.2G  3.6G  3.2G  54% /
/dev/sda1                99M   13M   82M   13% /boot
tmpfs                   125M   0    125M   0% /dev/shm
glusterfs#/etc/glusterfs/glusterfs.vol
                        7.2G  3.6G  3.2G  54% /var/www

```

Gambar 5.4 Glusterfs.vol Telah Ter-mount pada var/www

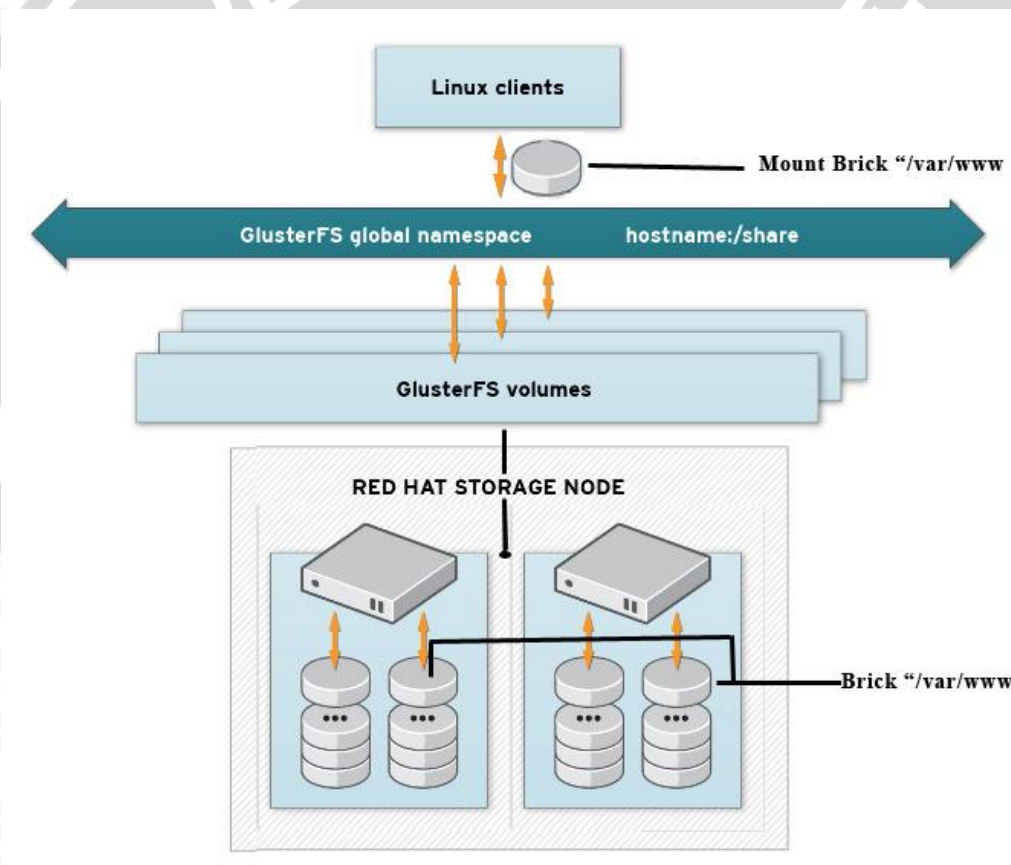
Size yang di-mount oleh *client* adalah sebesar 7,2 Giga Byte. 7,2 Giga Byte tersebut merupakan *size volume brick* dari kedua server glusterFS. *Mount* glusterfs.vol diletakkan pada *fstab* agar dapat secara otomatis ter-mount saat sistem mulai *booting*. Gambar 5.5 merupakan isi dari *fstab* yang telah ditambah konfigurasi untuk *mounting* glusterfs.vol.

```

root@s3:~
File Edit View Terminal Tabs Help
/dev/VolGroup00/LogVol00 /                ext3    defaults    1 1
LABEL=/boot              /boot    ext3    defaults    1 2
tmpfs                    /dev/shm tmpfs    defaults    0 0
devpts                   /dev/pts devpts   gid=5,mode=620 0 0
sysfs                    /sys     sysfs    defaults    0 0
proc                     /proc    proc     defaults    0 0
/dev/VolGroup00/LogVol01 swap                swap     defaults    0 0
/etc/glusterfs/glusterfs.vol /var/www  glusterfs defaults    0 0
    
```

Gambar 5.5 Glusterfs.vol Telah Ter-mount pada var/www.

Secara keseluruhan sistem GlusterFS yang dibuat adalah seperti pada Gambar 5.6.



Gambar 5.6 Sistem GlusterFS

5.3 Proses Instalasi Director (Virtual Server)

Director dalam LVS akan melanjutkan request dari client ke real server yang memiliki data website yang diminta oleh client. Pada penelitian ini menggunakan satu buah director dengan spesifikasi sebagai berikut :



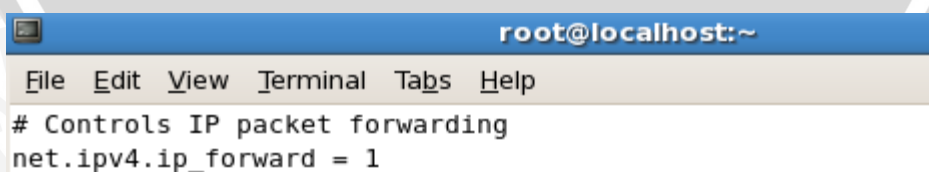
- Prosesor : Intel (R) Core (TM) 2 Quad CPU8300 @ 2.50GHz
- Memori : 2 GB
- Kapasitas Hardisk : 500 GB
- Memiliki 2 Gigabit Ethernet.

Persiapan instalasi dilakukan dengan memberi *IP address* pada kedua ethernet. *IP address* yang menuju *client* adalah 172.20.15.204. *IP address* yang menuju *real server* adalah 10.10.10.1.

Pada penelitian ini *load balancing* dibuat dengan LVS dan *tool* pendukung *interface* Piranha. LVS merupakan fitur *cluster load balancing* yang telah disediakan oleh linux kernel. Untuk mengelolah Linux Virtual Server (LVS) diperlukan *tool* ipvsadm. Pada sistem operasi Linux dengan kernel 2.6.xx sudah terdapat terdapat ipvsadm. ipvsadm atau ipvs *administration* adalah *userspace* program yang berfungsi mengatur kernel modul ip_vs. Pada ipvs terdapat *scheduler* yang menentukan tujuan dari koneksi baru terhadap *real server*.

Pada penelitian ini *load balancing* dibuat dengan metode NAT (*Network Address Translation*) sehingga paket data dari LAN dengan *IP private* akan diteruskan ke internet. Dengan demikian konfigurasi yng dilakukan adalah sebagai berikut :

- *Enable IP forward*
Enable IP forward dilakukan dengan merubah *controls IP packet forwarding* net.ipv4.ip_forwad menjadi sama dengan 1 pada file /etc/sysctl.conf seperti Gambar 5.7.



```

root@localhost:~
File Edit View Terminal Tabs Help
# Controls IP packet forwarding
net.ipv4.ip_forward = 1

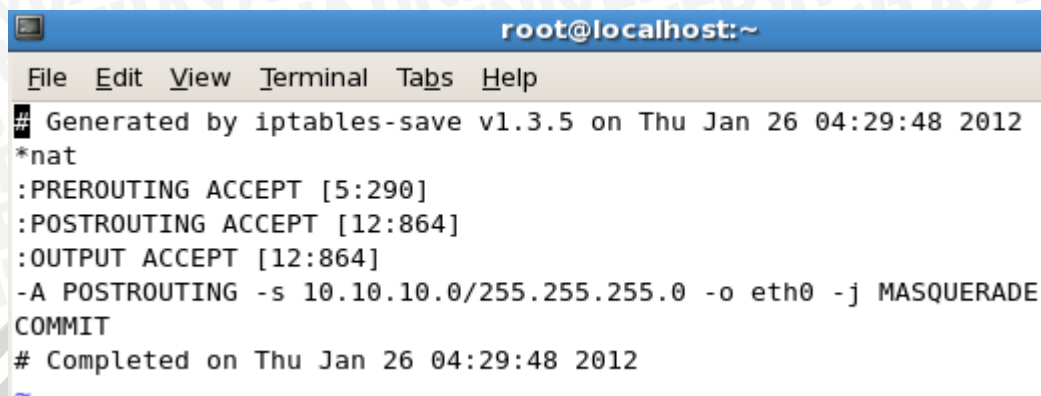
```

Gambar 5.7 *Enable IP Packet Forwarding*

IP forward adalah suatu sistem yang berfungsi untuk meneruskan atau mem-*forward* paket-paket dari suatu *network* ke *network* yang lain.

- *Enable IP masquerade*

Gambar 5.8 berikut merupakan isi dari iptables yang dibuat untuk IP *masquerade*.



```

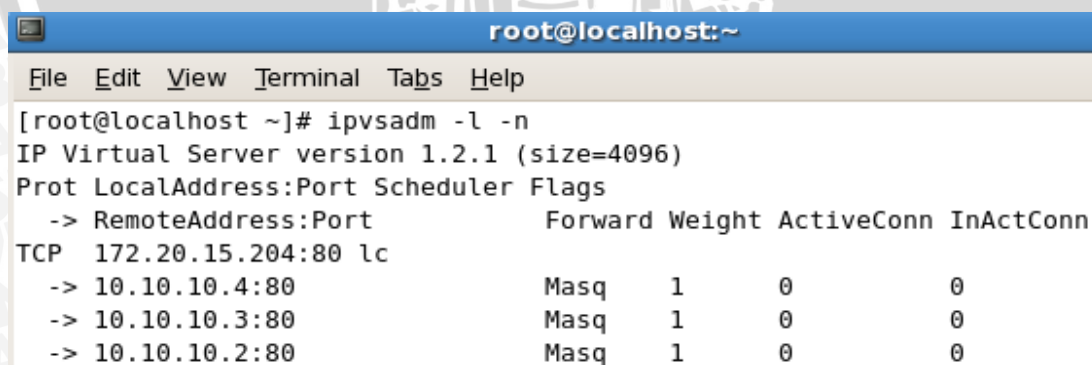
root@localhost:~
File Edit View Terminal Tabs Help
# Generated by iptables-save v1.3.5 on Thu Jan 26 04:29:48 2012
*nat
:PREROUTING ACCEPT [5:290]
:POSTROUTING ACCEPT [12:864]
:OUTPUT ACCEPT [12:864]
-A POSTROUTING -s 10.10.10.0/255.255.255.0 -o eth0 -j MASQUERADE
COMMIT
# Completed on Thu Jan 26 04:29:48 2012

```

Gambar 5.8 *Masquerade* iptables

Fungsi *masquerade* pada linux adalah seperti *one-to-many* NAT (*Network Address Translation*). Fungsi *masquerade* diletakkan pada POSTROUTING yang artinya fungsi akan dieksekusi ketika paket akan meninggalkan *director*.

Langkah selanjutnya adalah konfigurasi LVS-NAT dengan *ipvsadm*. Gambar 5.9 merupakan tabel LVS dari konfigurasi yang dibuat.



```

root@localhost:~
File Edit View Terminal Tabs Help
[root@localhost ~]# ipvsadm -l -n
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
-> RemoteAddress:Port          Forward Weight ActiveConn InActConn
TCP 172.20.15.204:80 lc
-> 10.10.10.4:80                Masq   1       0         0
-> 10.10.10.3:80                Masq   1       0         0
-> 10.10.10.2:80                Masq   1       0         0

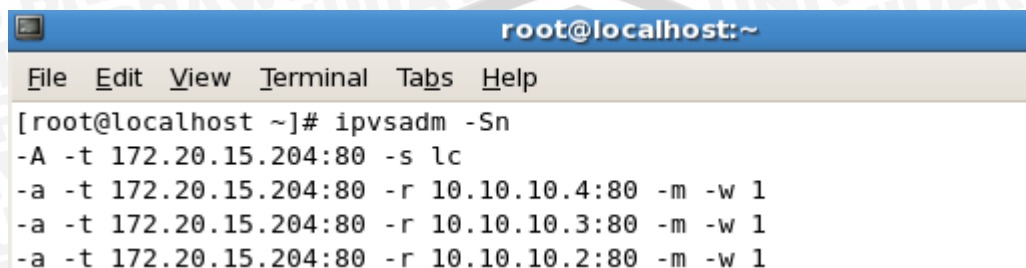
```

Gambar 5.9 Tabel LVS

Pada tabel LVS menunjukkan bahwa algoritma yang digunakan adalah algoritma *lc* (*least connection*). Tipe *forwarding* yang digunakan adalah *masq* (*masquerade*) yang merupakan fungsi untuk NAT pada Linux. Bobot (*weight*) yang diberikan

terhadap masing-masing *real server* bernilai sama karena tiap *real* memiliki spesifikasi *hardware* yang sama.

Agar *virtual service* secara otomatis aktif ketika sistem booting dilakukan perintah pada *terminal* Linux seperti pada Gambar 5.10.



```

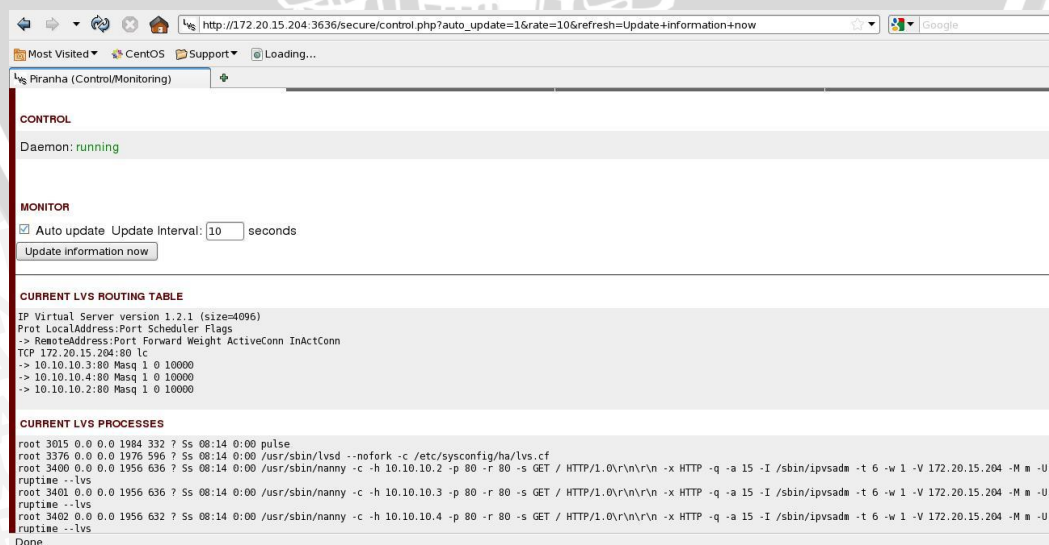
root@localhost:~
File Edit View Terminal Tabs Help
[root@localhost ~]# ipvsadm -Sn
-A -t 172.20.15.204:80 -s lc
-a -t 172.20.15.204:80 -r 10.10.10.4:80 -m -w 1
-a -t 172.20.15.204:80 -r 10.10.10.3:80 -m -w 1
-a -t 172.20.15.204:80 -r 10.10.10.2:80 -m -w 1

```

Gambar 5.10 *Virtual Service*

Konfigurasi LVS juga dapat dilakukan dengan menggunakan *tool* dengan GUI (*Graphical User Interface*) yaitu dengan meng-*install* Piranha. Aplikasi Piranha berjalan pada *web browser* (*web based*). Aplikasi Piranha dapat dijalankan dengan mengakses *IP address director* dari web browser. Piranha berjalan pada port 3636. Fungsi dari Piranha adalah sebagai *web monitoring* dan untuk konfigurasi.

Gambar 5.11 merupakan halaman monitoring tabel LVS pada Piranha.



```

CONTROL
Daemon: running

MONITOR
 Auto update Update Interval: 10 seconds
Update information now

CURRENT LVS ROUTING TABLE
IP Virtual Server version 1.2.1 (size=096)
Prot LocalAddress:Port Scheduler Flags
-> RemoteAddress:Port Forward Weight ActiveConn InActConn
TCP 172.20.15.204:80 lc
-> 10.10.10.3:80 Masq 1 0 10000
-> 10.10.10.4:80 Masq 1 0 10000
-> 10.10.10.2:80 Masq 1 0 10000

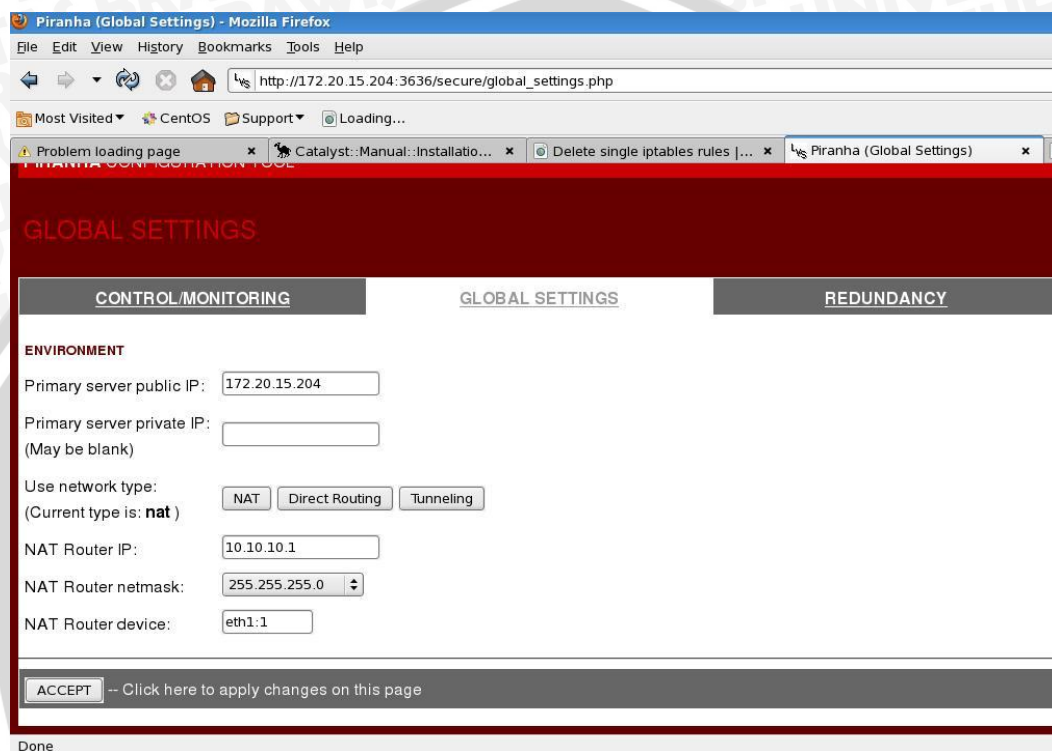
CURRENT LVS PROCESSES
root 3015 0.0 0.0 1984 332 ? Ss 08:14 0:00 pulse
root 3376 0.0 0.0 1976 596 ? Ss 08:14 0:00 /usr/sbin/lvsd --nofork -c /etc/sysconfig/ha/lvs.cf
root 3400 0.0 0.0 1956 636 ? Ss 08:14 0:00 /usr/sbin/nanny -c -h 10.10.10.2 -p 80 -r 80 -s GET / HTTP/1.0\r\n\r\n -x HTTP -q -a 15 -I /sbin/ipvsadm -t 6 -w 1 -V 172.20.15.204 -M m -U
ruptime --lvs
root 3401 0.0 0.0 1956 636 ? Ss 08:14 0:00 /usr/sbin/nanny -c -h 10.10.10.3 -p 80 -r 80 -s GET / HTTP/1.0\r\n\r\n -x HTTP -q -a 15 -I /sbin/ipvsadm -t 6 -w 1 -V 172.20.15.204 -M m -U
ruptime --lvs
root 3402 0.0 0.0 1956 632 ? Ss 08:14 0:00 /usr/sbin/nanny -c -h 10.10.10.4 -p 80 -r 80 -s GET / HTTP/1.0\r\n\r\n -x HTTP -q -a 15 -I /sbin/ipvsadm -t 6 -w 1 -V 172.20.15.204 -M m -U
ruptime --lvs
Done

```

Gambar 5.11 Halaman *Monitoring* Piranha.

Pada halaman *monitoring* Piranha menampilkan LVS *routing table* dan *current LVS processes*. Pada Gambar 5.11 dapat diatur interval waktu untuk meng-*update LVS routing table*.

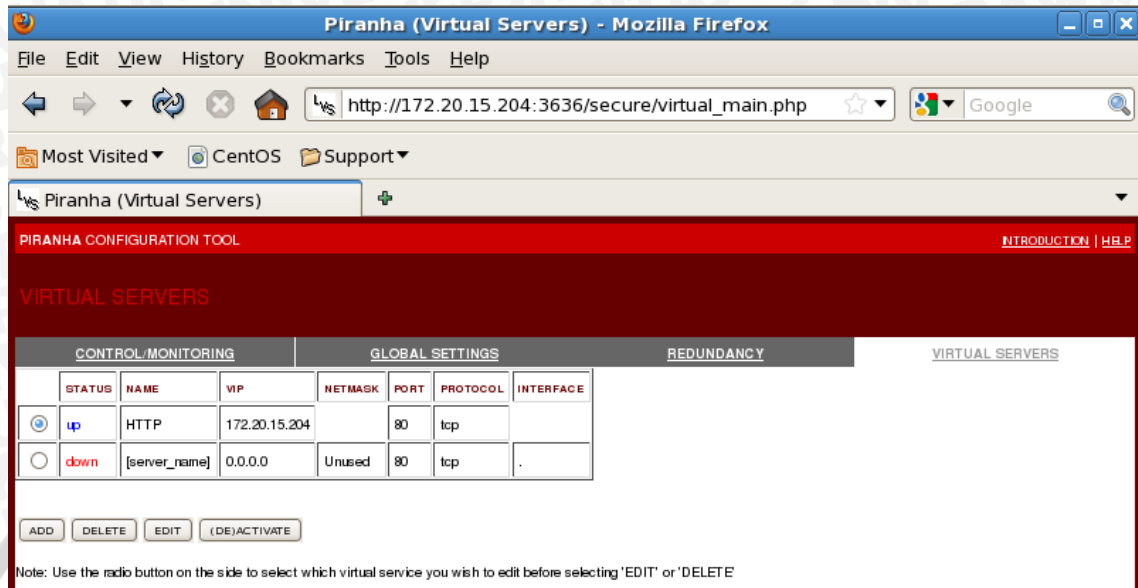
Piranha juga menyediakan GUI untuk konfigurasi LVS. Gambar 5.12 merupakan halaman *global setting* pada Piranha.



Gambar 5.12 Halaman *Global Setting* Piranha

Pada *global setting* Piranha, Primary server public IP diisi dengan 172.20.15.204. *Primary server public* merupakan IP address sesungguhnya dari *director*. Sedangkan *Primary server private* IP dibiarkan kosong. Primary server private IP adalah IP address untuk *backup server* dari *director* jika menggunakan Heartbeat. Penelitian ini menggunakan tipe jaringan NAT. NAT Router IP diisi dengan 10.10.10.1. NAT router IP merupakan *gateway* dari *real server*. NAT Router netmask adalah netmask dari *floating IP* NAT. NAT Router *device* merupakan nama dari *network interface* yang terhubung dengan jaringan NAT.

Pada Gambar 5.13 merupakan halaman *virtual server*,



Gambar 5.13 Halaman *Virtual Server* Piranha

Halaman *virtual server* berisi konfigurasi *director (virtual server)*. Pada halaman ini dapat menambah jumlah *virtual server* dan menon-aktifkan *virtual server*. Konfigurasi lebih lengkap dapat dilakukan dengan memilih menu edit pada bagian bawah tabel *virtual server*. Pada menu edit dapat dilakukan konfigurasi terhadap *virtual server*, *real server*, dan *monitoring script*.

Gambar 5.14 merupakan halaman *edit virtual server*,

The screenshot shows a web browser window with the URL `http://172.20.15.204:3636/secure/virtual_edit_virt.php?selected_host=1`. The browser tabs include 'Problem loading page', 'Catalyst::Manual::Installatio...', and 'Delete single iptables rules |...'. The main content area has two tabs: 'CONTROL/MONITORING' (selected) and 'GLOBAL SETTINGS'. Below the tabs, there are three sub-sections: 'EDIT: VIRTUAL SERVER', 'REAL SERVER', and 'MONITORING SCRIPTS'. The 'EDIT: VIRTUAL SERVER' section contains the following configuration fields:

- Name: HTTP
- Application port: 80
- Protocol: tcp
- Virtual IP Address: 172.20.15.204
- Virtual IP Network Mask: Unused
- Sorry Server: (empty)
- Firewall Mark: (empty)
- Device: eth0:1
- Re-entry Time: 15
- Service timeout: 6
- Quiesce server: Yes No
- Load monitoring tool: ruptime
- Scheduling: Least-connection
- Persistence: (empty)
- Persistence Network Mask: Unused

Gambar 5.14 Halaman *Edit Virtual Server* Piranha.

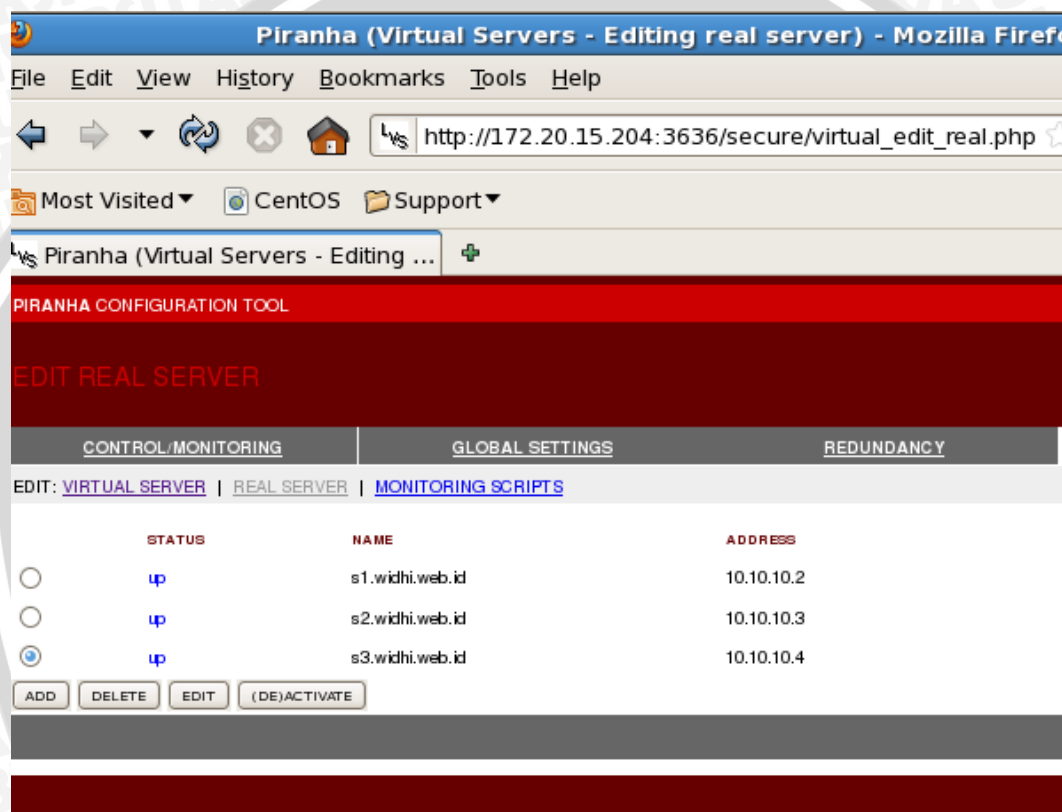
Konfigurasi yang dilakukan pada halaman *edit virtual server* antara lain :

1. *Name* : bagian ini untuk memberi nama *virtual server* dengan nama HTTP.
2. *Application port* : diisi dengan port yang di-*monitoring* oleh aplikasi Piranha. Pada penelitian ini diisi dengan 80 karena yang dimonitoring adalah HTTP *services*.
3. *Protocol* : *Protocol* yang digunakan pada penelitian ini adalah TCP.

4. *Virtual IP Address* : diisi dengan *IP address* dari *virtual server* yaitu 172.20.15.204.
5. *Virtual IP Network Mask* : Pada penelitian ini tidak digunakan *network mask* untuk *virtual server*.
6. *Firewall mask* : firewall mark tidak digunakan pada multi-port protokol. Karena dalam penelitian ini hanya menggunakan port 80 (HTTP) maka firewall mark dapat diisi 80.
7. *Device* : diisi dengan interface virtual server. Nama harus berbeda dengan interface-nya misal, interface yang digunakan *virtual server* adalah eth0 maka device diberi nama eth0:1.
8. *Re-entry Time* : merupakan lama waktu yang diperlukan untuk membawa kembali *real server* setelah mengalami kegagalan.
9. *Service Timeout* : Lama waktu yang diperlukan untuk mengeluarkan *real server* yang mengalami kegagalan dari *cluster*.
10. *Quiesce Server* : ketika *radio button* dipilih adalah *yes* maka setiap ada *real server* baru yang terhubung dengan sistem maka LVS tabel akan di-*reset* menjadi 0 untuk menghindari kemacetan terhadap *request* yang banyak pada *real server* baru.
11. *Load Monitoring Tool* : dipilih *ruptime* untuk menunjukkan status *host* pada sistem lokal.
12. *Scheduling* : dipilih algoritma penjadwalan yang digunakan. Pada penelitian ini menggunakan algoritma *least connection*.

13. *Persistence* : selang waktu yang ditentukan sebelum koneksi *client* terhadap *virtual server* menjadi *timeout*. Pada penelitian ini *persistence* tidak ditentukan (*default*).
14. *Persistence Network Mask* : untuk membatasi subnet.

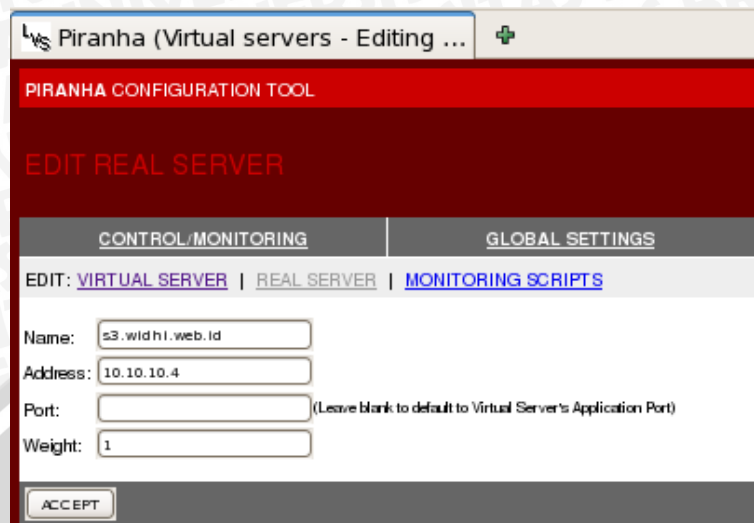
Gambar 5.15 merupakan halaman *edit real server*.



Gambar 5.15 Halaman *Edit Real Server*

Halaman *edit real server* adalah untuk konfigurasi *real server*. Pada halaman edit *real server* dapat menambah, mengurangi, mengaktifkan dan me-*non*-aktifkan *real server*.

Gambar 5.16 adalah contoh meng-*edit real server* baru,



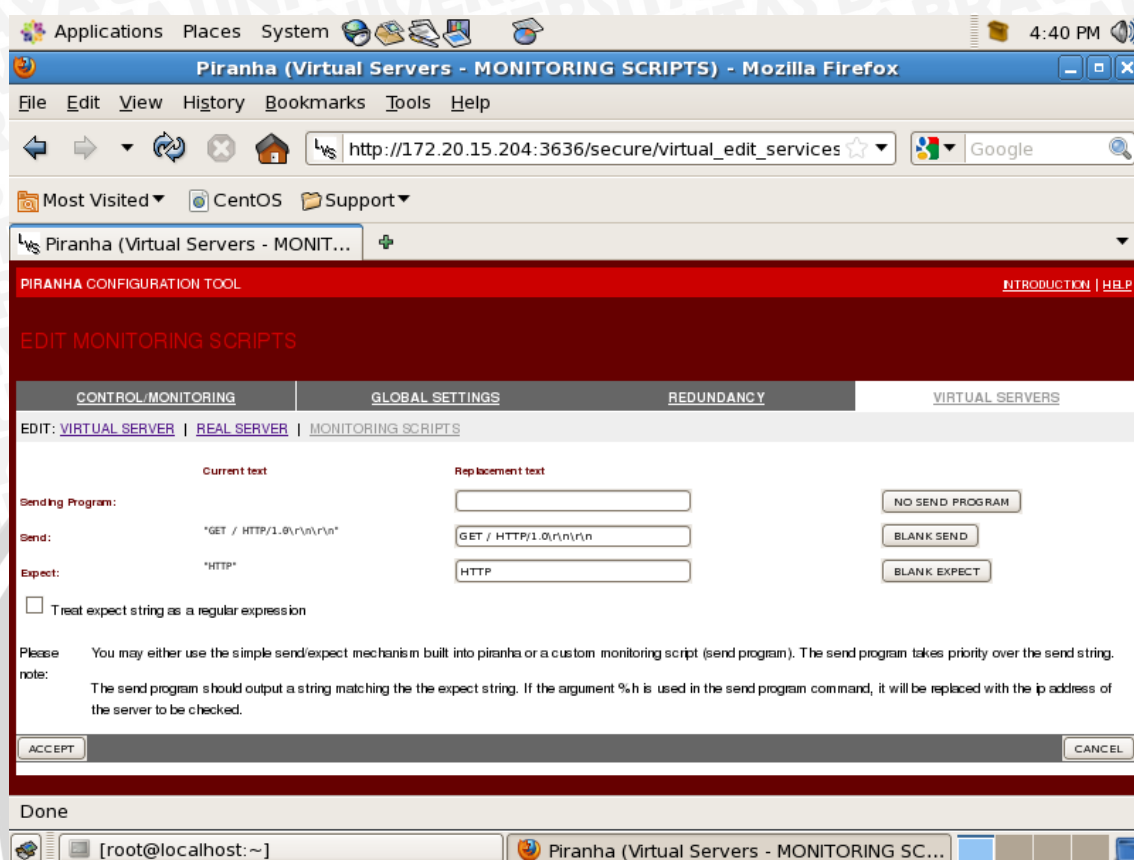
The screenshot shows the Piranha Configuration Tool window titled "Piranha (Virtual servers - Editing ...)". The main title bar is red and contains the text "PIRANHA CONFIGURATION TOOL". Below this, the window is divided into two tabs: "CONTROL/MONITORING" and "GLOBAL SETTINGS". The "GLOBAL SETTINGS" tab is active, and the sub-tab "EDIT: REAL SERVER" is selected. The form contains the following fields:

- Name:
- Address:
- Port: (Leave blank to default to Virtual Server's Application Port)
- Weight:

At the bottom of the form is an "ACCEPT" button.

Gambar 5.16 Edit Real server

Parameter yang harus diisi pada saat meng-*edit real server* adalah *name*, *address*, *port* dan *weight*. *Name* adalah nama yang diberikan untuk *real server*. *Address* merupakan *IP address* dari *real server*. *Port* pada penelitian ini menggunakan *default port* yang telah ditentukan pada konfigurasi *virtual server*. *Weight* merupakan beban prioritas yang diberikan terhadap *real server*. Semakin besar *weight* suatu *real server* maka semakin banyak *request* yang ditujukan ke *real server* tersebut. Besar *default* dari *weight* adalah 1.

Gambar 5.17 merupakan halaman *Edit Monitoring Scripts*Gambar 5.17 *Edit Monitoring Scripts*

Karena me-monitoring sebuah *web server*, maka *monitoring scripts* yang digunakan adalah `GET /HTTP/1.0\r\n\r\n`.

BAB VI

PENGUJIAN DAN ANALISIS

Pengujian pada sebuah sistem adalah untuk melihat keberhasilan sistem menjalankan fungsionalitasnya dan mengetahui tingkat reliabilitas dari sistem ketika memproses masukan. Pengujian sistem *load balancing* pada *web server* ini terdiri dari dua pengujian yaitu, pengujian terhadap kinerja dan pengujian reliabilitas sistem sesuai parameter-parameter yang ditentukan.

6.1 Pengujian Kinerja

Parameter keberhasilan dalam pengujian kinerja sistem *Load Balancing* pada *web server* dengan algoritma *Least Connection* adalah :

1. Sistem dapat melayani permintaan halaman *website* yang dilakukan oleh *client*.
2. *Director* meneruskan *request* ke *real server*.
3. *Request* yang datang diteruskan ke *real server* yang memiliki jumlah koneksi paling sedikit.
4. Sistem dapat melayani *request* ketika ada salah satu *real server* yang mengalami kegagalan atau *down*.

Pada pengujian kinerja dilakukan beberapa skenario untuk menguji sistem.

a. Skenario 1

Pada skenario 1 pengujian dilakukan untuk mengetahui sistem dapat melayani permintaan halaman *website* yang dilakukan oleh *client*.

Langkah yang dilakukan:

1. Dilakukan *request* halaman *myfav.html* oleh *client* Windows dengan IP 172.20.15.90 ke *webserver*.
2. *Capture* pada *port* ethernet *client* dengan Wireshark, *capture port* ethernet *director* dengan *tcpdump*, dan *capture port* ethernet *real server*.

Hasil skenario 1 :

No.	Source	Destination	Time	Protocol	Info
13	172.20.15.15	172.20.15.204	10.238486000	HTTP	50353
19	172.20.15.204	172.20.15.15	10.240828000	HTTP	http
30	172.20.15.15	172.20.15.204	10.273351000	HTTP	50355
39	172.20.15.15	172.20.15.204	10.275179000	HTTP	50354
49	172.20.15.204	172.20.15.15	10.276977000	HTTP	http
52	172.20.15.15	172.20.15.204	10.277608000	HTTP	50356
57	172.20.15.204	172.20.15.15	10.278104000	HTTP	http
59	172.20.15.15	172.20.15.204	10.278267000	HTTP	50357
100	172.20.15.204	172.20.15.15	10.283910000	HTTP	http
103	172.20.15.204	172.20.15.15	10.284302000	HTTP	http
115	172.20.15.15	172.20.15.204	10.486196000	HTTP	50358
117	172.20.15.204	172.20.15.15	10.487450000	HTTP	http
127	172.20.15.15	172.20.15.204	12.738155000	HTTP	50362
133	172.20.15.204	172.20.15.15	12.740304000	HTTP	http
155	172.20.15.15	172.20.15.204	12.799878000	HTTP	50363
156	172.20.15.15	172.20.15.204	12.800633000	HTTP	50364

Frame 115: 385 bytes on wire (3080 bits), 385 bytes captured (3080 bits)
 Ethernet II, Src: Inventec_f7:57:e3 (00:1e:33:f7:57:e3), Dst: Elitegro_c6:fe:92 (10:78:d2:c6:fe:92)
 Internet Protocol Version 4, Src: 172.20.15.15 (172.20.15.15), Dst: 172.20.15.204 (172.20.15.204)
 Transmission Control Protocol, Src Port: 50358 (50358), Dst Port: http (80), Seq: 1, Ack: 1, Len: 319
 Hypertext Transfer Protocol

Gambar 6.1 Capture Request HTTP Client ke Server dengan Wireshark.

Pada Gambar 6.1 saat mengirim request IP source (client) adalah 172.20.15.15 sedangkan IP destination adalah 172.20.15.204 atau IP dari director.

No.	Source	Destination	Time	Protocol	Info
8	172.20.15.15	172.20.15.204	3.879703	HTTP	50353
10	172.20.15.204	172.20.15.15	3.880795	HTTP	http
11	172.20.15.204	172.20.15.15	3.880920	HTTP	http
12	172.20.15.204	172.20.15.15	3.881044	HTTP	http
13	172.20.15.204	172.20.15.15	3.881049	HTTP	http
28	172.20.15.15	172.20.15.204	3.914544	HTTP	50355
33	172.20.15.204	172.20.15.15	3.915519	HTTP	http
34	172.20.15.204	172.20.15.15	3.915643	HTTP	http
35	172.20.15.204	172.20.15.15	3.915764	HTTP	http
36	172.20.15.15	172.20.15.204	3.916394	HTTP	50354
39	172.20.15.204	172.20.15.15	3.917070	HTTP	http
40	172.20.15.204	172.20.15.15	3.917190	HTTP	http
41	172.20.15.204	172.20.15.15	3.917312	HTTP	http
42	172.20.15.204	172.20.15.15	3.917349	HTTP	http
47	172.20.15.204	172.20.15.15	3.918148	HTTP	http
48	172.20.15.204	172.20.15.15	3.918260	HTTP	http

Frame 8: 480 bytes on wire (3840 bits), 96 bytes captured (768 bits)
 Ethernet II, Src: Inventec_f7:57:e3 (00:1e:33:f7:57:e3), Dst: Elitegro_c6:fe:92 (10:78:d2:c6:fe:92)
 Internet Protocol Version 4, Src: 172.20.15.15 (172.20.15.15), Dst: 172.20.15.204 (172.20.15.204)
 Transmission Control Protocol, Src Port: 50353 (50353), Dst Port: http (80), Seq: 1, Ack: 1, Len: 414
 Hypertext Transfer Protocol
 [Packet size limited during capture: HTTP truncated]

Gambar 6.2 Capture Request HTTP pada eth0 director (IP public).

No.	Source	Destination	Time	Protocol	Info
7	172.20.15.15	10.10.10.2	2.249652	HTTP	50487
9	10.10.10.2	172.20.15.15	2.250787	HTTP	http
10	10.10.10.2	172.20.15.15	2.250910	HTTP	http
11	10.10.10.2	172.20.15.15	2.251083	HTTP	http
12	10.10.10.2	172.20.15.15	2.251138	HTTP	http
37	172.20.15.15	10.10.10.2	2.294339	HTTP	50488
39	172.20.15.15	10.10.10.3	2.294988	HTTP	50489
40	10.10.10.2	172.20.15.15	2.295285	HTTP	http
41	10.10.10.2	172.20.15.15	2.295408	HTTP	http
43	10.10.10.2	172.20.15.15	2.295538	HTTP	http
44	172.20.15.15	10.10.10.3	2.295810	HTTP	50490
45	10.10.10.3	172.20.15.15	2.296140	HTTP	http
47	10.10.10.3	172.20.15.15	2.296264	HTTP	http
49	172.20.15.15	10.10.10.2	2.296465	HTTP	50491
50	10.10.10.3	172.20.15.15	2.296613	HTTP	http
51	10.10.10.3	172.20.15.15	2.296737	HTTP	http

Frame 7: 490 bytes on wire (3920 bits), 96 bytes captured (768 bits)
 Ethernet II, Src: Tp-LinkT_f1:de:dc (00:27:19:f1:de:dc), Dst: Elitegro_c7:01:2b (10:78:d2:c7:01:2b)
 Internet Protocol Version 4, Src: 172.20.15.15 (172.20.15.15), Dst: 10.10.10.2 (10.10.10.2)
 Transmission Control Protocol, Src Port: 50487 (50487), Dst Port: http (80), Seq: 1, Ack: 1, Len: 424
 Hypertext Transfer Protocol
 [Packet size limited during capture: HTTP truncated]

Gambar 6.3 Capture Request HTTP pada eth1 director (IP NAT ke real-server).

Pada Gambar 6.3 IP *source (client)* adalah 172.20.15.15, IP *destination* adalah 10.10.10.2 dan 10.10.10.3, menunjukkan bahwa *request* dari *client* ke *director* diteruskan ke *real server*.

No.	Source	Destination	Time	Protocol	Info
9	172.20.15.15	10.10.10.3	7.167719	HTTP	50353
11	10.10.10.3	172.20.15.15	7.168693	HTTP	http
12	10.10.10.3	172.20.15.15	7.168943	HTTP	http
13	10.10.10.3	172.20.15.15	7.168945	HTTP	http
14	10.10.10.3	172.20.15.15	7.168946	HTTP	http
30	172.20.15.15	10.10.10.2	7.204382	HTTP	50354
32	10.10.10.2	172.20.15.15	7.204743	HTTP	http
35	172.20.15.15	10.10.10.2	7.206660	HTTP	50356
37	10.10.10.2	172.20.15.15	7.206891	HTTP	http
38	10.10.10.2	172.20.15.15	7.206898	HTTP	http
39	172.20.15.15	10.10.10.2	7.207221	HTTP	50357
41	10.10.10.2	172.20.15.15	7.207421	HTTP	http
42	10.10.10.2	172.20.15.15	7.207428	HTTP	http
44	10.10.10.2	172.20.15.15	7.208161	HTTP	http
45	10.10.10.2	172.20.15.15	7.208164	HTTP	http
46	10.10.10.2	172.20.15.15	7.208166	HTTP	http

Frame 9: 480 bytes on wire (3840 bits), 96 bytes captured (768 bits)
 Ethernet II, Src: Tp-LinkT_f1:de:dc (00:27:19:f1:de:dc), Dst: Elitegro_c6:fe:6e (10:78:d2:c6:fe:6e)
 Internet Protocol Version 4, Src: 172.20.15.15 (172.20.15.15), Dst: 10.10.10.3 (10.10.10.3)
 Transmission Control Protocol, Src Port: 50353 (50353), Dst Port: http (80), Seq: 1, Ack: 1, Len: 414
 Hypertext Transfer Protocol
 [Packet size limited during capture: HTTP truncated]

Gambar 6.4 Capture Request HTTP pada Real Server dengan tcpdump.

Pada Gambar 6.4 menunjukkan bahwa *real server* langsung menerima *request* yang berasal langsung dari *client* (172.20.15.15).



Gambar 6.5 Screenshot Halaman Web yang Diminta oleh Client

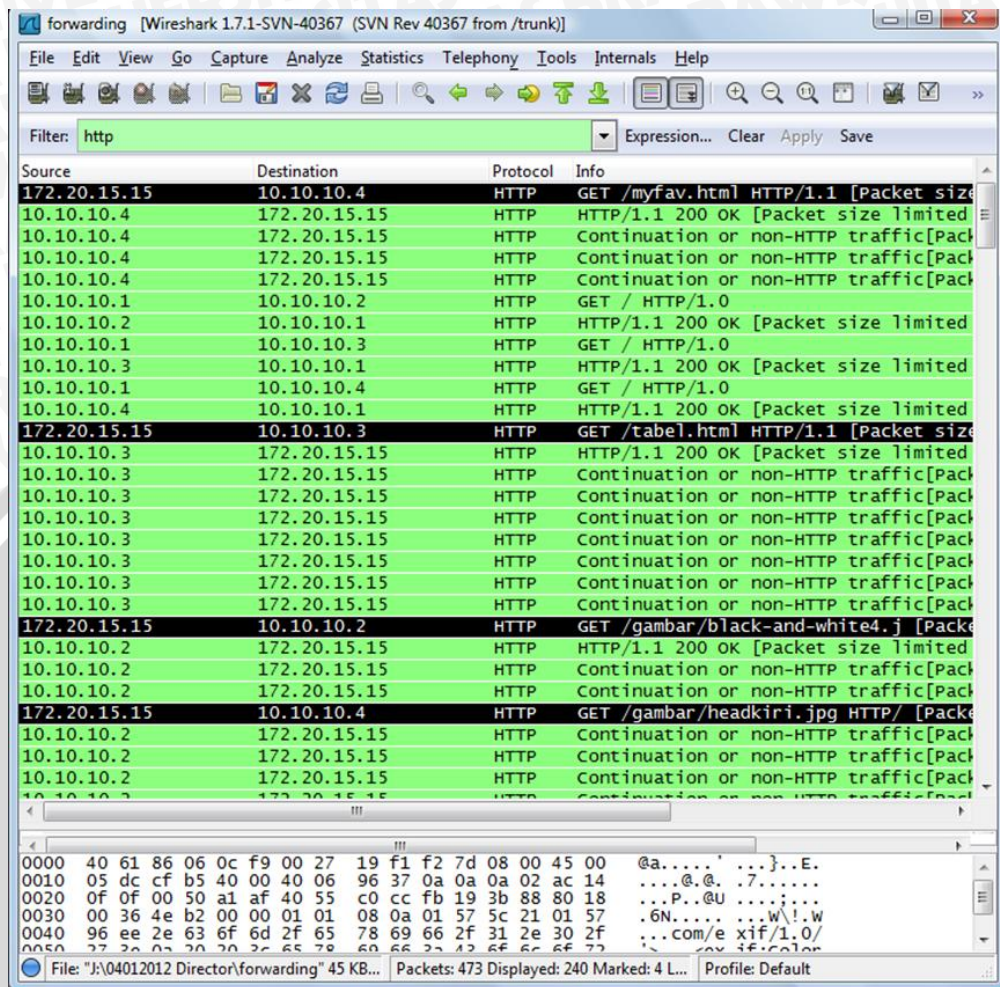
b. Skenario 2

Pada skenario 1 pengujian dilakukan untuk mengetahui *director* meneruskan *request* ke *real server*.

Langkah yang dilakukan:

1. Dilakukan *request* oleh *client* IP 172.20.15.15 ke *server website*.
2. Dilakukan *capture tcpdump* pada port 80 *director*.
3. Hasil *tcpdump* dibuka pada Wireshark

Hasil skenario 2 :



Gambar 6.6 Screenshot Halaman Web yang Diminta oleh Client.

Pada Gambar 6.6 *request* diteruskan ke *real server* dengan IP 10.10.10.2, 10.10.10.3, dan 10.10.10.4.

c. Skenario 3

Pada skenario 3 pengujian dilakukan untuk mengetahui *request* yang datang dari *client* diteruskan ke *real server* yang memiliki jumlah koneksi paling sedikit. Pengujian dalam skenario 3 menunjukkan kerja dari algoritma *least connection*.

Langkah yang dilakukan:

1. Dilakukan koneksi dengan menggunakan *tool* *httperf* sebanyak 5 koneksi ke sistem *load balancing* dengan 3 *real server*.

2. Dilakukan 1 koneksi ke sistem *load balancing* dengan 3 *real server* yang telah mendapat koneksi dari langkah 1.
3. Langkah 1 dan 2 dilakukan secara simultan.

Gambar 6.7 merupakan gambar *LVS Routing Table* ketika sistem sedang tidak menerima koneksi dari *client*.

CURRENT LVS ROUTING TABLE

```
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
-> RemoteAddress:Port Forward Weight ActiveConn InActConn
TCP 172.20.15.204:80 lc
-> 10.10.10.4:80 Masq 1 0 0
-> 10.10.10.3:80 Masq 1 0 0
-> 10.10.10.2:80 Masq 1 0 0
```

Gambar 6.7 LVS Routing Table dengan 0 Koneksi.

Hasil dari skenario 3 :

Pada Gambar 6.8 merupakan tabel *routing* LVS setelah dilakukan 5 koneksi.

CURRENT LVS ROUTING TABLE

```
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
-> RemoteAddress:Port Forward Weight ActiveConn InActConn
TCP 172.20.15.204:80 lc
-> 10.10.10.4:80 Masq 1 0 2
-> 10.10.10.3:80 Masq 1 0 2
-> 10.10.10.2:80 Masq 1 0 1
```

Gambar 6.8 LVS Routing Table dengan 5 Koneksi.

Pada Gambar 6.8 menunjukkan LVS telah membagi koneksi ke tiga *real server*. *Real server* 10.10.10.3 dan 10.10.10.4 menerima 2 koneksi sedangkan *real server* 10.10.10.2 hanya menerima 1 koneksi.

Koneksi selanjutnya dilakukan dengan jumlah koneksi 1 ke LVS. Setelah dilakukan 1 koneksi ke LVS yang sudah menangani 5 koneksi maka dapat dilihat *update* tabel LVS yang ditunjukkan pada Gambar 6.9.

CURRENT LVS ROUTING TABLE

```
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
-> RemoteAddress:Port Forward Weight ActiveConn InActConn
TCP 172.20.15.204:80 lc
-> 10.10.10.4:80 Masq 1 0 2
-> 10.10.10.3:80 Masq 1 0 2
-> 10.10.10.2:80 Masq 1 0 2
```

Gambar 6.9 LVS Routing Table dengan 6 Koneksi.

Koneksi yang baru datang diarahkan ke *real server* 10.10.10.2 karena memiliki jumlah koneksi paling sedikit.

d. Skenario 4

Pengujian pada skenario 4 sama dengan pengujian pada skenario 3. Pada skenario 4 komputer dengan IP *address* 10.10.10.2 di *set* dalam keadaan *down*. Pengujian yang dilakukan dengan skenario 4 memisalkan suatu keadaan dimana ada *real server* yang tiba-tiba mati atau dalam proses *maintenance*.

Langkah yang dilakukan:

1. Mengatur *real server* dengan IP *address* 10.10.10.2 dalam keadaan *down*.
2. Dilakukan koneksi dengan menggunakan *tool* *httperf* sebanyak 5 koneksi ke sistem *load balancing*.
3. Dilakukan 1 koneksi ke sistem *load balancing* dengan 3 *real server* yang telah mendapat koneksi dari langkah 1.
4. Langkah 1 dan 2 dilakukan secara simultan.

Hasil dari sekenario 4 :

CURRENT LVS ROUTING TABLE

```
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
-> RemoteAddress:Port Forward Weight ActiveConn InActConn
TCP 172.20.15.204:80 lc
-> 10.10.10.4:80 Masq 1 0 3
-> 10.10.10.3:80 Masq 1 0 2
-> 10.10.10.2:80 Masq 0 0 0
```

Gambar 6.10 LVS Routing Table dengan 5 Koneksi.

Pada Gambar 6.10 menunjukkan *real server* dengan IP *adres* 10.10.10.2 dalam keadaan *down*. 5 koneksi yang diberikan di teruskan ke *real server* dengan IP *address* 10.10.10.3 dan 10.10.10.4. Koneksi selanjutnya akan didistribusikan ke *real server* dengan *address* 10.10.10.3 karena memiliki jumlah koneksi paling sedikit.

Pada gambar 6.11 merupakan gambar *routing table* setelah sistem diberikan koneksi sebanyak 1 koneksi.

CURRENT LVS ROUTING TABLE

```
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
-> RemoteAddress:Port Forward Weight ActiveConn InActConn
TCP 172.20.15.204:80 lc
-> 10.10.10.4:80 Masq 1 0 3
-> 10.10.10.3:80 Masq 1 0 3
-> 10.10.10.2:80 Masq 0 0 0
```

Gambar 6.11 LVS Routing Table dengan 6 Koneksi.

6.2 Pengujian Performa

Pengujian performa dilakukan untuk mengetahui performa dari *web server*. parameter yang dianalisis dari pengujian performa antara lain *throughput*, waktu respon dan CPU *utilization*. Pada pengujian performa akan dibandingkan performa dari penggunaan *single web server* dengan *web server* yang menggunakan sistem *load balacing*. *Web server* dengan sistem *load balancing* yang diuji menggunakan 2 *real server* dan 3 *real server*.

Pada pengujian performa dilakukan koneksi dengan jumlah koneksi sebanyak 10000 dengan rate 1000 koneksi/detik, 30000 dengan rate 3000 koneksi/detik dan 60000 dengan rate 6000 koneksi/detik yang ditujukan kepada *single web server*, web server dengan sistem *load balancing* yang menggunakan 2 *real server* dan web server dengan sistem *load balancing* yang menggunakan 3 *real server*. Koneksi pada pengujian performa dibuat dengan menggunakan *software* penghitung performa web server HTTPERF.

Pengujian performa pada *web server* dilakukan dengan menggunakan HTTPERF. Contoh perintah yang dijalankan pada *terminal* Linux adalah sebagai berikut :

```
root@localhost ~]# httpperf --hog --timeout=5 --
server=172.20.15.204 --port=80 --uri=/index.html --rate=1000
--num-conns=10000
httpperf --hog --timeout=5 --client=0/1 --server=172.20.15.204 --
port=80 --uri=/index.html --rate=1000 --send
-buffer=4096 --recv-buffer=16384 --num-conns=10000 --num-calls=1
Maximum connect burst length: 1

Total: connections 10000 requests 10000 replies 10000 test-
duration 10.000 s

Connection rate: 1000.0 conn/s (1.0 ms/conn, <=895 concurrent
connections)
Connection time [ms]: min 0.6 avg 241.5 max 3002.3 median 0.5
stddev 799.3
Connection time [ms]: connect 229.1
Connection length [replies/conn]: 1.000

Request rate: 1000.0 req/s (1.0 ms/req)
Request size [B]: 76.0

Reply rate [replies/s]: min 894.1 avg 894.1 max 894.1 stddev 0.0
(1 samples)
Reply time [ms]: response 12.4 transfer 0.0
Reply size [B]: header 264.0 content 958.0 footer 0.0 (total
1222.0)
Reply status: 1xx=0 2xx=10000 3xx=0 4xx=0 5xx=0

CPU time [s]: user 1.50 system 8.50 (user 15.0% system 85.0% total
100.0%)
Net I/O: 1267.6 KB/s (10.4*10^6 bps)

Errors: total 0 client-timo 0 socket-timo 0 connrefused 0
connreset 0
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0
```

Perintah `httpperf --hog --timeout=5 --server=172.20.15.204 --port=80 --uri=/index.html --rate=1000 --num-conns=10000` adalah untuk membuat koneksi ke *host* dengan *IP address* 172.20.15.204 dengan jumlah koneksi 10000 dan rate 1000 koneksi per detik, mengirim *request* untuk *root* dokumen (`http://www/index.html`), menerima balasan, menutup koneksi dan mencetak pada layar statistik performa *web server*. *Timeout* adalah selang waktu yang digunakan *client* untuk menunggu balasan atau respon dari *server*. *Timeout* pada pengujian performa ditetapkan sebesar 5 detik (`--timeout=5`). Jika selama 5 detik *client* tidak mendapatkan respon dari *server*, maka panggilan dinyatakan gagal oleh *tool* HTTPERF. Hasil pengujian dengan HTTPERF diatas didapatkan waktu respon sebesar 12,4 ms (`Reply time [ms]: response 12.4`) dan *throughput* sebesar 1267,6 KB/s (`Net I/O: 1267.6 KB/s`).

6.2.1 Single Web Server

Tabel 6.1 merupakan hasil pengujian performa pada *single web server* dengan dilakukan koneksi sebanyak 10000 koneksi dan rate 1000 koneksi/detik.

Tabel 6.1 Tabel Koneksi Single Web Server yang Menerima 10000 Koneksi dengan Rate 1000 Koneksi/Detik

No	Response (ms)	Throughput (KB/s)	CPU utilization (%)		
			User	System	Idle
1	12,4	1267,6	0,53	0,62	98,83
2	0,5	1267,6	0,63	0,83	98,54
3	0,5	1267,6	0,62	0,55	98,82
4	0,5	1267,6	0,62	0,53	98,84
5	0,5	1267,6	0,75	0,72	98,52
Rata-rata	2,88	1267,6	0,63	0,65	98,71

Tabel 6.1 menunjukkan data setelah dilakukan 5 kali pengujian dengan koneksi 10000 terhadap sistem. Pada tabel 6.1 menunjukkan waktu respon dan *throughput* yang cukup stabil dan menunjukkan bahwa *single web server* dapat menangani 10000 koneksi dengan *rate* 1000 koneksi/detik secara baik. Pada 10000 koneksi yang dilakukan kepada *single web server*, sistem hanya menggunakan CPU

sebesar 1,29%. Koneksi yang dibangkitkan tidak berdampak signifikan terhadap penggunaan CPU sehingga untuk analisis terhadap parameter CPU *utilization* dilakukan dengan cara membandingkan penggunaan CPU pada setiap sistem yang dibuat.

Tabel 6.2 merupakan hasil pengujian performa pada *single web server* dengan dilakukan koneksi sebanyak 30000 koneksi dan rate 3000 koneksi/detik.

Tabel 6.2 Tabel Koneksi *Single Web Server* yang Menerima 30000 Koneksi dengan *Rate* 3000 Koneksi/Detik.

No	Response (ms)	Throughput (KB/s)	CPU utilization (%)		
			User	System	Idle
1	7,3	3802,8	1,50	2,88	95,57
2	0,6	3802,7	0,93	2,13	96,89
3	5,6	3653	1,12	2,05	96,82
4	0,6	3802,7	1,08	2,26	96,56
5	1,3	3802,7	1,12	1,88	96,99
Rata-rata	3,08	3772,78	1,15	2,24	96,57

Tabel 6.2 menunjukkan data setelah dilakukan 5 kali pengujian dengan koneksi 30000 terhadap sistem. Data waktu respon dan *throughput* lebih bervariasi dibandingkan pengujian dengan 10000 koneksi. Penggunaan CPU saat sistem menerima koneksi 30000 dengan rate 3000 koneksi/detik rata-rata sebesar 3,43% yang berasal dari 100% total penggunaan CPU dikurangi 96,7% CPU idle rata-rata.

Tabel 6.3 merupakan hasil pengujian performa pada *single web server* dengan dilakukan koneksi sebanyak 60000 koneksi dan rate 6000 koneksi/detik.

Tabel 6.3 Tabel Koneksi *Single Web Server* yang Menerima 360000 Koneksi dengan *Rate* 6000 Koneksi/Detik.

No	Response (ms)	Throughput (KB/s)	CPU utilization (%)		
			User	System	Idle
1	7,6	7604,7	6,36	10,08	83,53
2	5,7	6545	5,64	9,98	84,33
3	4,5	7604,8	4,41	9,57	85,96
4	4,9	7431,1	5,03	9,65	85,15
5	4,4	7604,6	4,96	10,05	84,84
Rata-rata	5,42	7358,04	5,28	9,87	84,76

Tabel 6.3 menunjukkan data setelah dilakukan 5 kali pengujian dengan koneksi sebanyak 60000 koneksi terhadap sistem. Penggunaan CPU meningkat dari CPU idle sebesar 96,57% pada saat menerima 30000 koneksi dengan rate 3000 koneksi/detik, menjadi 84,76% setelah menerima 60000 koneksi dengan rate 6000 koneksi/detik. Waktu respon dari web server juga meningkat menjadi sebesar 5,42 ms.

6.2.2 Load Balancing dengan 2 Real Server

Pengujian dilakukan sama dengan pengujian pada single web server. Pengujian pertama dengan melakukan koneksi sebesar 10000 koneksi dengan rate 1000 koneksi/detik terhadap sistem. Hasil pengujian dapat dilihat pada tabel 6.4 berikut

Tabel 6.4 Tabel Koneksi *Load Balancing* dengan 2 *Real Server* yang Menerima 10000 Koneksi dengan *Rate* 1000 Koneksi/Detik

No	Response (ms)	Throughput (KB/s)	CPU 1 utilization (%)			CPU 2 Utilization (%)			AVG CPU Utilization (%)		
			User	System	Idle	User	System	Idle	User	System	Idle
1	0,5	1267,1	0,03	0,02	99,93	0,92	1,38	97,70	0,48	0,70	98,82
2	0,5	1267,1	0,38	0,02	99,60	0,43	0,45	99,10	0,41	0,24	99,35
3	0,5	1267,1	0,43	0,05	99,52	1,00	0,97	98,04	0,72	0,51	98,78
4	0,5	1267,1	0,45	0,03	99,52	0,57	0,68	98,75	0,51	0,36	99,14
5	0,5	1267,1	0,40	0,02	99,57	1,23	1,23	97,52	0,82	0,63	98,55
Avg	0,5	1267,1	0,34	0,03	99,63	0,83	0,94	98,22	0,58	0,49	98,93

Hasil pengujian yang pada tabel 6.4 menunjukkan waktu respon dan *throughput* yang stabil pada saat sistem menangani 10000 koneksi dengan rate 1000 koneksi/detik. Rata-rata penggunaan CPU sebesar 1,07%.

Koneksi terhadap sistem ditingkatkan menjadi 30000 koneksi dengan rate 3000 koneksi/detik. Hasil pengujian dengan jumlah koneksi 30000 koneksi dapat dilihat pada tabel 6.5,

Tabel 6.5 Tabel Koneksi *Load Balancing* dengan 2 *Real Server* yang Menerima 30.000 Koneksi dengan *Rate* 3000 Koneksi/Detik

No	Response (ms)	Throughput (KB/s)	CPU 1 utilization (%)			CPU 2 Utilization (%)			AVG CPU Utilization (%)		
			User	System	Idle	User	System	Idle	User	System	Idle
1	0,6	3801,3	1,18	2,01	96,80	1,02	2,23	96,74	1,10	2,12	96,77
2	0,6	3801,3	1,61	2,30	96,70	1,25	1,93	96,80	1,43	2,12	96,75
3	0,6	3801,3	1,35	2,10	96,62	1,31	2,25	96,42	1,33	2,18	96,52
4	0,6	3801,3	1,31	2,35	96,31	1,28	2,00	96,72	1,30	2,18	96,52
5	0,6	3801,3	1,58	1,86	96,55	1,25	1,95	96,79	1,42	1,91	96,67
Avg	0,6	3801,3	1,41	2,12	96,60	1,22	2,07	96,69	1,31	2,10	96,65

Tabel 6.5 menunjukkan waktu respon dan *throughput* stabil pada saat menerima koneksi 30000 koneksi dengan rate 3000 koneksi/detik. Penggunaan CPU sebesar 3,35% (100% - rata-rata CPU idle). Waktu respon dari sistem *load balancing* dengan 2 *real server* sebesar 0,6 ms.

Tabel 6.6 Tabel Koneksi *Load Balancing* dengan 2 *Real Server* yang

Menerima 60.000 Koneksi dengan *Rate* 6000 Koneksi/Detik

No	Response (ms)	Throughput (KB/s)	CPU 1 utilization (%)			CPU 2 Utilization (%)			AVG CPU UTILIZATION		
			User	System	Idle	User	System	Idle	User	System	Idle
1	1	7602,2	3,69	5,37	90,90	3,36	4,96	91,38	3,53	5,17	91,14
2	0,8	7602	3,21	5,78	91,00	3,96	5,19	90,84	3,59	5,49	90,92
3	1	7602,2	4,19	5,09	90,68	4,34	4,99	90,66	4,27	5,04	90,67
4	0,9	7601,8	3,38	5,53	91,06	4,46	5,61	89,82	3,92	5,57	90,44
5	1,1	7601,6	3,35	5,66	90,95	4,36	5,64	89,97	3,86	5,65	90,46
Avg	0,96	7601,96	3,56	5,49	90,92	4,10	5,28	90,53	3,83	5,38	90,73

Pada Tabel 6.6 menunjukkan waktu respon dan *throughput* pada pengujian sistem dengan koneksi sebesar 60000 koneksi dan rate 6000 koneksi perdetik memiliki besar yang bervariasi. Penggunaan CPU sebesar 9,27% (100% - rata-rata idle).

6.2.3 Load Balancing dengan 3 *Real Server*

Pengujian sistem *load balancing* dengan 3 *real server* dilakukan dengan membuat koneksi ke sistem dengan jumlah 10000, 30000 dan 60000 koneksi. Hasil pengujian dapat dilihat pada Tabel 6.7, Tabel 6.8 ,dan Tabel 6.9.

Tabel 6.7 Tabel Koneksi Load Balancing dengan 3 Real Server yang Menerima 10000 Koneksi dengan Rate 1000 Koneksi/Detik

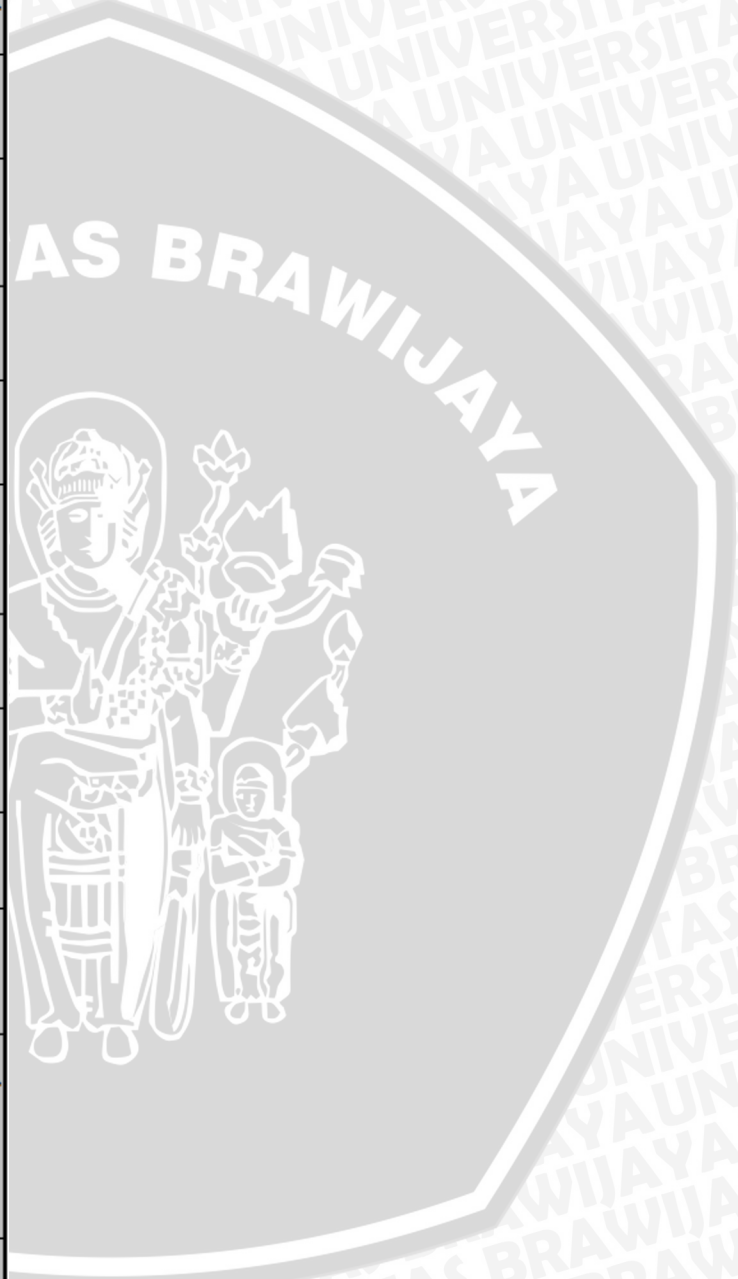
No	Response (ms)	Throughput (KB/s)	CPU 1 utilization (%)			CPU 2 Utilization (%)			CPU 3 Utilization (%)			AVG CPU Utilization (%)		
			User	System	Idle	User	System	Idle	User	System	Idle	User	System	Idle
1	0,50	1267,30	0,05	0,05	99,87	0,15	0,10	99,73	0,17	0,48	99,33	0,12	0,21	99,64
2	0,50	1267,30	0,38	0,02	99,60	0,15	0,07	99,77	0,95	0,80	98,25	0,49	0,30	99,21
3	0,50	1267,30	0,50	0,70	98,80	0,35	0,40	99,23	0,98	0,82	98,20	0,61	0,64	98,74
4	0,50	1267,30	0,40	0,03	99,57	0,10	0,15	99,73	0,88	0,80	98,30	0,46	0,33	99,20
5	0,50	1267,30	0,55	3,03	96,40	0,15	0,10	99,73	0,80	0,58	98,62	0,50	1,24	98,25
AVG	0,50	1267,30	0,38	0,77	98,85	0,18	0,16	99,64	0,76	0,70	98,54	0,44	0,54	99,01

Tabel 6.8 Tabel Koneksi Load Balancing dengan 3 Real Server yang Menerima 30000 Koneksi dengan Rate 3000 Koneksi/Detik

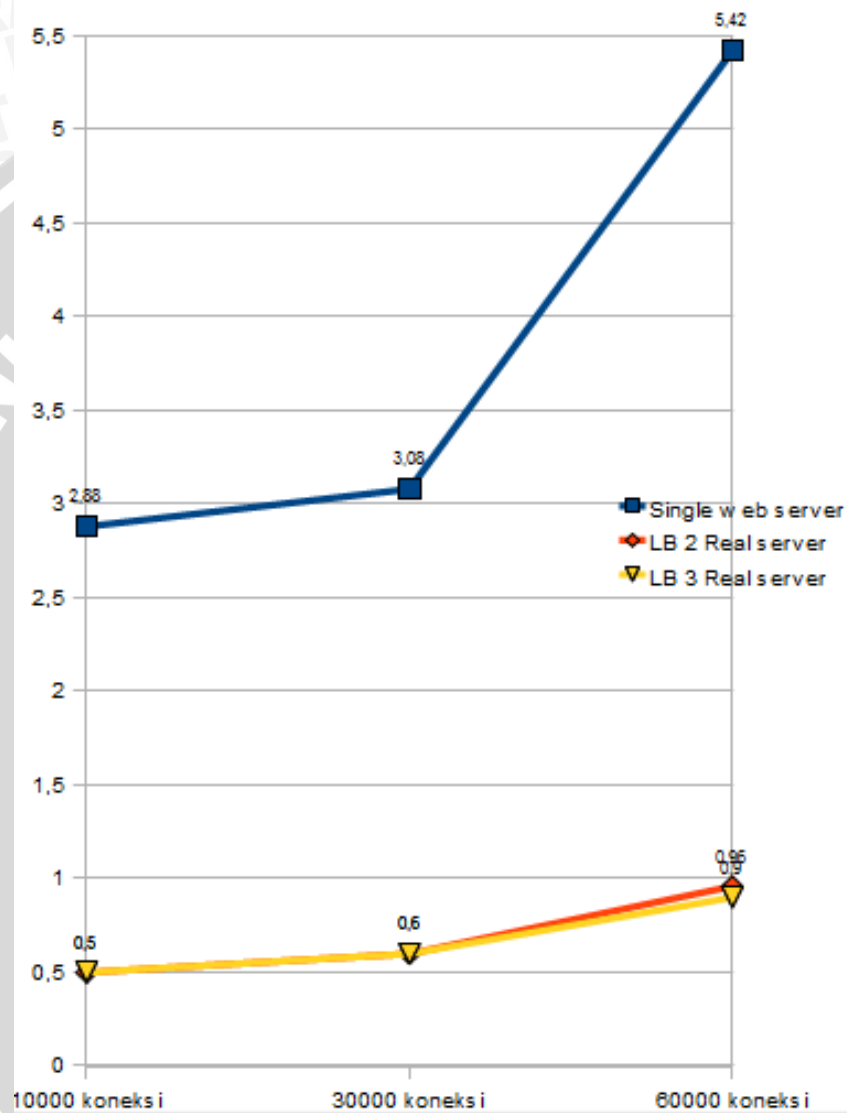
No	Response (ms)	Throughput (KB/s)	CPU 1 utilization (%)			CPU 2 Utilization (%)			CPU 3 Utilization (%)			AVG CPU Utilization (%)		
			User	System	Idle	User	System	Idle	User	System	Idle	User	System	Idle
1	0,60	3801,80	0,67	0,85	98,47	0,34	1,14	98,49	0,65	0,68	98,67	0,55	0,89	98,54
2	0,60	3801,80	0,63	0,80	98,57	0,53	0,78	98,65	0,75	0,68	98,55	0,64	0,75	98,59
3	0,60	3801,80	0,68	0,68	98,64	0,27	1,03	98,67	0,81	0,62	98,55	0,59	0,78	98,62
4	0,60	3801,80	0,62	0,57	98,82	0,28	0,85	98,83	0,72	0,80	98,47	0,54	0,74	98,71
5	0,60	3801,80	0,83	0,67	98,50	0,33	0,50	99,13	0,67	1,00	98,62	0,61	0,72	98,75
AVG	0,60	3801,80	0,69	0,71	98,60	0,35	0,86	98,75	0,72	0,76	98,57	0,59	0,78	98,64

Tabel 6.9 Tabel Koneksi Load Balancing dengan 3 Real Server yang Menerima 60000 Koneksi dengan Rate 6000 Koneksi/Detik

No	Response (ms)	Throughput (KB/s)	CPU utilization			CPU UTILIZATION 2			CPU UTILIZATION 3			CPU UTILIZATION AVG		
			System	User	Idle	System	User	Idle	System	User	Idle	System	User	Idle
1	0,90	7603,00	2,75	3,56	93,59	1,77	3,25	94,93	2,83	3,44	93,73	2,45	3,42	94,08
2	0,90	7602,80	2,76	3,66	93,56	2,25	2,99	94,71	2,33	3,84	93,81	2,45	3,50	94,03
3	0,90	7602,80	2,48	4,03	93,48	1,93	3,26	94,74	2,55	3,36	94,09	2,32	3,55	94,10
4	0,90	7602,70	2,65	3,86	93,49	2,18	2,78	94,98	2,55	3,68	93,77	2,46	3,44	94,08
5	0,90	7602,80	2,65	3,79	93,54	1,89	3,09	94,94	2,55	3,56	93,87	2,36	3,48	94,12
AVG	0,90	7602,82	2,658	3,78	93,53	2,00	3,07	94,86	2,56	3,58	93,85	2,41	3,48	94,08



Hasil pengujian pada *web server* dengan *load balancing* menunjukkan bahwa sistem lebih baik daripada *single web server* dan sistem *load balancing* dengan 2 *real server*. Waktu respon paling baik ketika *web server* menggunakan *load balancing* dengan 3 *real server* yang dapat dilihat pada Gambar 6.12 berikut,



Gambar 6.12 Grafik Waktu Respon (ms) Hasil Pengujian Performa.

Pada Gambar 6.12 *single web server* memiliki waktu respon yang lebih besar daripada *web server* yang menggunakan *load balancing*. *Web server* yang menggunakan *load balancing* dengan 2 *real server* dan 3 *real server* memiliki waktu respon yang tidak berbeda jauh. Pada saat pengujian dengan jumlah koneksi 10000 koneksi dengan rate 1000 koneksi/detik dan 30000 koneksi dengan

rate 3000 koneksi/detik, sistem *load balancing* dengan 2 *real server* memiliki rata-rata waktu respon yang sama dengan sistem *load balacing* dengan 3 *real server* yaitu sebesar 0,6 ms. Pada pengujian dengan 60000 koneksi dengan rate 6000 koneksi/detik waktu respon *single web server* adalah 5,42 ms, waktu respon *sistem load balancing* dengan 2 *real server* adalah 0,96 ms ,dan waktu *sistem load balancing* dengan 3 *real server* adalah 0,9 ms. Waktu respon paling baik pada pengujian performa yang dilakukan adalah waktu respon pada *web server* dengan *load balancing* yang menggunakan 3 *real server*.

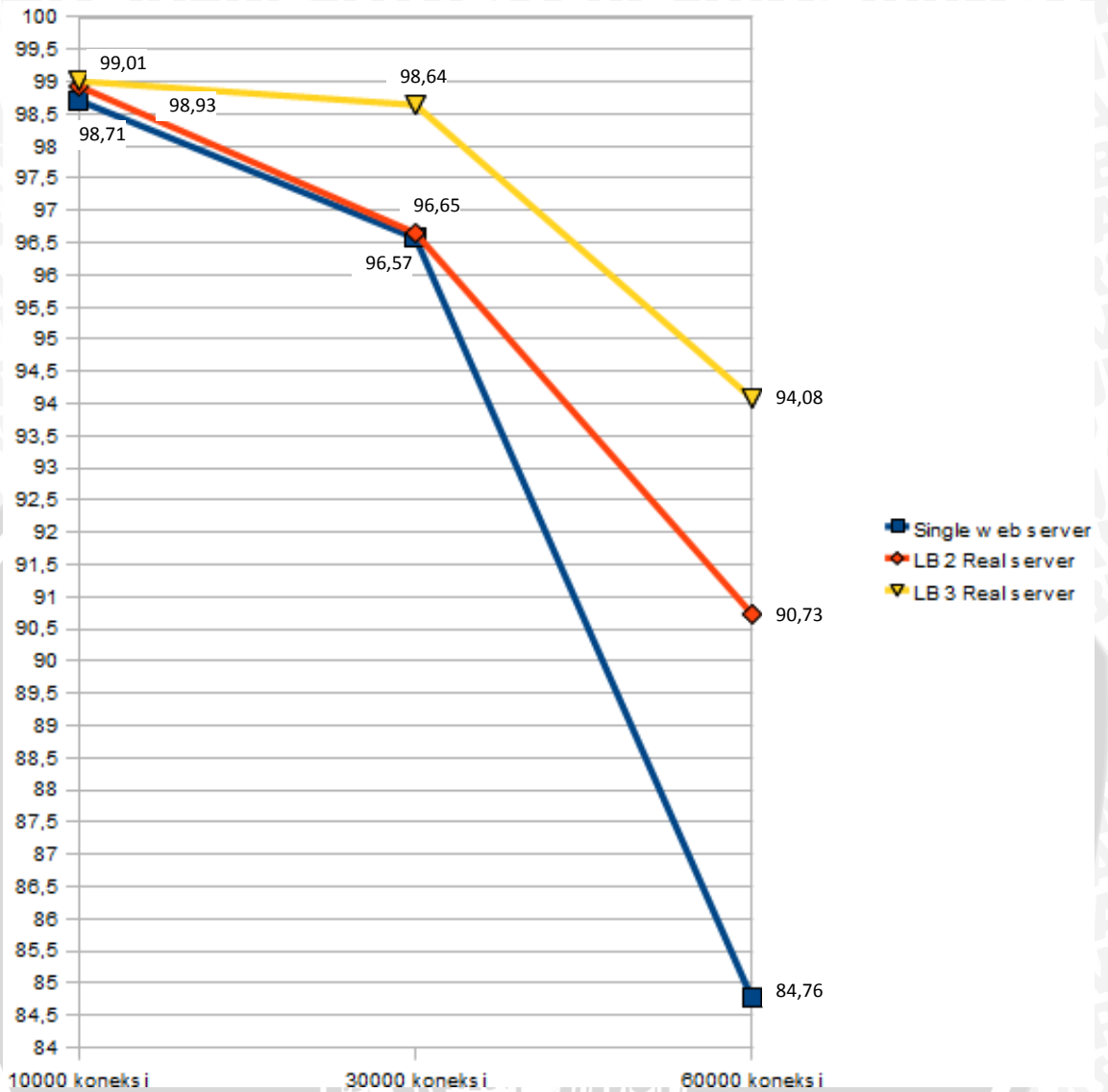
Throughput hasil pengujian paling stabil pada *web server* dengan *load balancing* menggunakan 3 *real server*. Tabel 6.10 merupakan tabel standar deviasi parameter *throughput* setelah dilakukan 5 kali pengujian terhadap *single web server* dan *web server* dengan *sistem load balancing*.

Tabel 6.10 Standar Deviasi *Throughput*

	Jumlah Koneksi		
	10000	30000	60000
<i>Single Web Server</i>	0,00	66,96	460,68
LB dengan 2 <i>Real Server</i>	0,00	0,00	0,26
LB dengan 3 <i>Real Server</i>	0,00	0,00	0,11

Standar deviasi *throughput* menunjukkan sebaran data parameter *throughput* pada lima kali pengujian. Sebaran data *throughput* semakin kecil menunjukkan *throughput* pada sistem lebih stabil ketika dilakukan pengujian sebanyak 5 kali. Sistem yang paling stabil pada pengujian performa adalah *web server* dengan *load balancing* yang menggunakan 3 *real server* dengan sebaran data pada 5 kali pengujian adalah 0; 0; dan 0,11.

Pada sistem *load balancing* beban (*request*) yang ditujukan terhadap server akan didistribusikan ke *real server* yang merupakan sebuah *cluster* komputer sehingga peeggunaan CPU lebih baik dari sistem *single web server*. Pada Gambar 6.13 ini merupakan grafik CPU *idle* pada tiap sistem yang diuji.



Gambar 6.13 Grafik CPU *Utilization* (%) Hasil Pengujian Performa

Dengan *load balancing*, *request* ditangani oleh 3 buah CPU sehingga kinerja masing-masing CPU menjadi lebih ringan. Pada Gambar 6.13 menunjukkan penggunaan CPU paling sedikit yaitu pada *web server* dengan *load balancing* yang menggunakan 3 *real server*.

BAB VII

PENUTUP

Penutup berisi kesimpulan dari analisis dan implementasi sistem serta saran untuk pengembangan sistem.

7.1 Kesimpulan

Dari analisis dan implementasi *load balancing* pada *web server* dengan algoritma *least connection*, dapat disimpulkan bahwa :

1. *Web server* dengan menggunakan sistem *load balancing* lebih *high availability* atau memiliki tingkat ketersediaan yang tinggi. Ketika ada salah satu *server down* atau sedang dalam *maintenance* sistem secara keseluruhan tetap bisa melayani *request* dari *client*.
2. Performa dari *web server* yang mengimplementasikan *load balancing* lebih baik daripada penggunaan *single web server*.
3. Penggunaan jumlah *real server* dapat mempengaruhi performa dari sistem *load balancing*.
4. Sistem *load balancing* dengan tiga *real server* memiliki performa lebih baik daripada sistem yang menggunakan dua *real server*.
5. Algoritma penjadwalan *least connection* dapat berjalan dengan baik dan membagi beban berdasarkan jumlah koneksi paling sedikit pada *real server* sehingga jumlah koneksi yang diterima untuk masing-masing *real server* rata.
6. Penjadwalan berdasarkan koneksi membagi beban tidak sama-rata tergantung pada proses yang sedang dijalankan seperti proses *download* dan proses akses *website*.

7.2 Saran

Saran dalam pengembangan sistem *load balancing* pada *web server* adalah:

1. *Weight* diberikan kepada *rule* penjadwalan pada penggunaan *real server* dengan spesifikasi dan tugas yang berbeda sehingga pembagian tugas CPU dapat rata untuk semua *real server* pada sistem *load balancing*.
2. Pada implementasi *load balancing* pada *web server* yang telah dilakukan tidak membahas detail tentang *high availability storage*. Walaupun dalam implementasi menggunakan *high availability storage* Glusterfs namun produk tersebut tidak bergaransi sehingga penggunaan dalam sistem yang menyimpan dan mendistribusikan data yang bersifat penting, perlu dilakukan penelitian ulang tentang *high availability storage*.
3. Sistem *load balancing* yang dibuat tidak menerapkan *high availability* pada *director* sehingga apabila *director* mengalami kerusakan (*down*) maka sistem secara keseluruhan tidak dapat berfungsi. Sehingga disarankan dibuat *backup server* dari *director* agar sistem lebih *high availability*.
4. Perlu dilakukan penelitian lebih lanjut tentang *sistem load balancing* dengan metode penerapan dan algoritma *load balancing* yang lain sehingga dapat diketahui karakteristik sistem yang menggunakan metode dan algoritma yang berbeda.
5. Pada *web server* dengan website yang menggunakan *session* perlu ada *memcache server* untuk meng-*handle session*.

DAFTAR PUSTAKA

- [BOU-01] Bourke, Tony. 2001, "Server Load Balancing", USA: O'Reilly & Associates, Inc.
- [HID-04] Hidayat, Rudi, dkk. 2004, "Teknologi Informasi & Komunikasi", Jakarta: Erlangga.
- [IRA-05] Irawan, Budhi. 2005, "Jaringan Komputer", Yogyakarta: Graha Ilmu.
- [OET-03] Oetomo, Budi Sutedjo Dharma, S. Kom. 2003, "Konsep dan Perancangan Jaringan Komputer", Yogyakarta : Andi.
- [MAD-03] Madcoms. 2003, "Dasar Teknis Instalasi Jaringan Komputer", Yogyakarta : Andi.
- [IRA-05] Irawan, Budhi. 2005, "Jaringan Komputer", Yogyakarta: Graha Ilmu.
- [TIT-04] Titte, I Ed. 2004, "Computer Networking", Jakarta: Gramedia.
- [SYA-03] Syahputra, Andi. 2003, "Apache Web Server", Yogyakarta : Andi.
- [ALE-02] Alex, Ferrianto, Gozali. 2002, "Virtual Server", JETri 2, 1: 53-68.
- [SIE-09] Sierra, Katja Gilly de la, "An adaptive admission control and load balancing algorithm for a QoS-aware Web system", Dissertation, Universitat de les Illes Balears Palma de Mallorca, Spain.
- [PIE-09] Pietruszka, Mike; Host, Rudy. 2009, "High-Availability Solutions: Building Low Cost Data Centers Using Open Source UNIX-based Server Clusters".
- [CIS-07] Cisco IOS Release 12.0(10)W5(18). 2007, "Catalyst 4840G Software Feature and Configuration Guide".
- [RED-07] Red Hat, Inc (USA). 2007, "Linux VirtualServer (LVS) for Red Hat Enterprise Linux 5.1".
- [MOS-02] Mosberger, David; Jin, Tai. 2002, "httpperf-A Tool For Measuring Web Server Performance", Hewlett-Packard Co.
- [CIT-10] Citrix Systems, Inc. 2010, "The Least Connection Method", URL:<http://support.citrix.com/proddocs/topic/netScaler-load-q-balancing-93/ns-lb-customizing-about-leastconnection-con.html>
- [RED-10] Red Hat, Inc. 2010, "IP Load Balancing", URL:<http://www.redhat.com/software/rha/cluster/piranha/>.

UNIVERSITAS BRAWIJAYA

LAMPIRAN-LAMPIRAN



Perintah untuk konfigurasi ethernet :

```
vi /etc/sysconfig/network-scripts/ifcfg-eth1
```

Isi File ifcfg-eth1 :

```
DEVICE=eth1
BOOTPROTO=none
ONBOOT=yes
HWADDR=00:0c:29:64:6b:c6
IPADDR=10.10.10.x
NETMASK=255.255.255.0
GATEWAY=10.10.10.1
```

Konfigurasi nama hosts dengan perintah vi /etc/hosts :

```
DEVICE=eth1
BOOTPROTO=none
ONBOOT=yes
HWADDR=00:0c:29:64:6b:c6
IPADDR=10.10.10.x
NETMASK=255.255.255.0
GATEWAY=10.10.10.1
```

Apache diinstal pada ketiga *real server* dengan perintah sebagai berikut :

```
yum install httpd -y
```

Perintah tersebut dilakukan pada terminal linux dan sistem merespon sebagai berikut :

```
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: mirrors.hostemo.com
 * extras: mirrors.hostemo.com
 * updates: mirrors.hostemo.com
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package httpd.i386 0:2.2.3-63.el5.centos.1 set to be updated
base/filelists                               | 3.0 MB
00:48
extras/filelists_db                           | 212 kB
00:03
updates/filelists_db                          | 1.0 MB
```

```

00:16
--> Finished Dependency Resolution

Dependencies Resolved

=====
Package Arch Version Repository
Size
=====
Installing:
  httpd i386 2.2.3-63.el5.centos.1 updates
1.2 M

Transaction Summary
=====
Install 1 Package(s)
Upgrade 0 Package(s)

Total download size: 1.2 M
Is this ok [y/N]: Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: mirrors.hostemo.com
 * extras: mirrors.hostemo.com
 * updates: mirrors.hostemo.com
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package httpd.i386 0:2.2.3-63.el5.centos.1 set to be updated
base/filelists | 3.0 MB
00:48
extras/filelists_db | 212 kB
00:03
updates/filelists_db | 1.0 MB
00:16
--> Finished Dependency Resolution

Dependencies Resolved

=====
Package Arch Version Repository
Size
=====
Installing:
  httpd i386 2.2.3-63.el5.centos.1 updates
1.2 M

Transaction Summary
=====
Install 1 Package(s)
Upgrade 0 Package(s)

Total download size: 1.2 M
Is this ok [y/N]: y
Downloading Packages:
httpd-2.2.3-63.el5.centos.1.i386.rpm | 1.2 MB

```



```
00:20
Running rpm_check_debug
Running Transaction Test
Finished Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing                               :      httpd
1/1

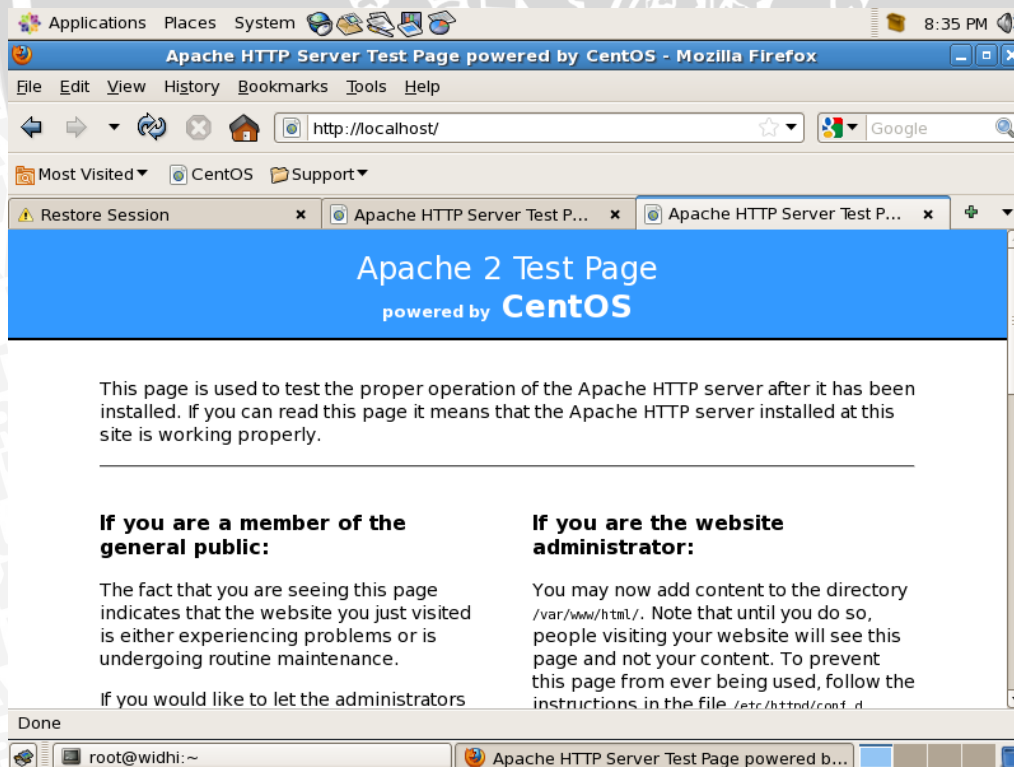
Installed:
  httpd.i386 0:2.2.3-63.el5.centos.1

Complete!
```

Untuk mengetahui apakah Apache sudah ter-*install* yaitu dengan menjalankan *service* httpd dengan menggunakan perintah sebagai berikut :

```
[root@widhi ~]# service httpd start
Starting httpd: [ OK ]
```

Halaman Test Page Apache :



Applications Places System 8:35 PM

Apache HTTP Server Test Page powered by CentOS - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://localhost/ Google

Most Visited CentOS Support

Restore Session Apache HTTP Server Test P... Apache HTTP Server Test P...

Apache 2 Test Page

powered by CentOS

This page is used to test the proper operation of the Apache HTTP server after it has been installed. If you can read this page it means that the Apache HTTP server installed at this site is working properly.

If you are a member of the general public:	If you are the website administrator:
The fact that you are seeing this page indicates that the website you just visited is either experiencing problems or is undergoing routine maintenance.	You may now add content to the directory <code>/var/www/html/</code> . Note that until you do so, people visiting your website will see this page and not your content. To prevent this page from ever being used, follow the instructions in the file <code>/etc/httpd/conf.d</code>
If you would like to let the administrators	

Done

root@widhi:~ Apache HTTP Server Test Page powered b...

Pada penelitian ini halaman *website* diletakkan pada direktori berikut:

```
[root@sl ~]# ls /var/www/html/widhi.web.id/
favlink.html   gambar          latar.css       myprofile.html  tabel.html
form.html      index.html      myfav.html      portofolio.html
```

Pada penelitian ini agar konten *website* dapat diakses pada *directory* /var/www/html/widhi.web.id/ dilakukan konfigurasi virtual host yaitu dengan perintah pada terminal Linux sebagai berikut :

```
vi /etc/httpd/conf/httpd.conf
```

Perintah diatas adalah untuk memodifikasi *file* httpd.conf. Kemudian pada skript httpd.conf dimasukan skript berikut:

```
NameVirtualHost 10.10.10.3:80

<VirtualHost 10.10.10.3:80>
    ServerAdmin webmaster@widhi.web.id
    DocumentRoot /var/www/html/widhi.web.id/
    ServerName widhi.web.id
    ErrorLog logs/widhi.web.id-error_log
    CustomLog logs/widhi.web.id-access_log common
</VirtualHost>

<VirtualHost 10.10.10.3:80>
    ServerAdmin webmaster@widhi.web.id
    DocumentRoot /var/www/html/widhi.web.id/
    ServerName widhi2.web.id
    ErrorLog logs/widhi2.web.id-error_log
    CustomLog logs/widhi2.web.id-access_log common
</VirtualHost>
```

Instalasi GlusterFS server.

Persiapan yang dilakukan sebelum meng-*install* server GlusterFS adalah meng-*install* Development Tools, Development Libraries, libibverbs-devel, dan fuse-devel menggunakan perintah sebagai berikut:

```
yum groupinstall 'Development Tools'
```

```
yum groupinstall 'Development Libraries'
```

```
yum install libibverbs-devel fuse-devel
```

GlusterFS di-*download* dengan menggunakan perintah berikut :

```
wget
http://download.gluster.com/pub/gluster/glusterfs/3.1/3.1.3/glusterfs-3.1.3.tar.gz
```

Kemudian *file* glusterfs-3.1.3.tar.gz diekstrak dengan perintah berikut:

```
tar xvfz glusterfs-3.1.3.tar.gz
```

Langkah selanjutnya adalah masuk *folder* glusterfs-3.1.3 dan melakukan *./configure* dengan perintah sebagai berikut:

```
[root@sl ~]# cd glusterfs-3.1.3
[root@sl glusterfs-3.1.3]# ./configure
```

Pada akhir *./configure* akan dilihat skrip berikut :

```
[...]
GlusterFS configure summary
=====
FUSE client      : yes
Infiniband verbs : yes
epoll IO multiplex : yes
argp-standalone  : no
fusermount       : no
readline         : yes
```

```
georeplication      : no
```

Setelah dilakukan *configure* GlusterFS di-*install* dengan perintah berikut :

```
Make && make install
ldconfig
```

Setelah ter-*install* dapat dilihat versi GlusterFS dengan perintah berikut :

```
[root@sl ~]# glusterfs --version
glusterfs 3.1.3 built on Jan 11 2012 02:23:39
Repository revision: v3.1.3
Copyright (c) 2006-2010 Gluster Inc. <http://www.gluster.com>
GlusterFS comes with ABSOLUTELY NO WARRANTY.
You may redistribute copies of GlusterFS under the terms of the GNU
Affero General Public License.
```

Glusterfs yang telah di-*install* adalah Glusterfs versi 3.1.3. langkah selanjutnya adalah membuat beberapa folder untuk tempat *file* konfigurasi Glusterfs.

```
mkdir /etc/glusterfs
```

File konfigurasi dibuat dengan perintah berikut :

```
vi /etc/glusterfs/glusterfsd.vol
```

File glusterfsd.vol diisi dengan konfigurasi sebagai berikut :

```
volume posix
    type storage/posix
    option directory /var/www/
end-volume

volume locks
    type features/locks
    subvolumes posix
end-volume

volume brick
    type performance/io-threads
    option thread-count 8
    subvolumes locks
end-volume
```

```
volume server
  type protocol/server
  option transport-type tcp
  option auth.addr.brick.allow 10.10.10.2,10.10.10.3,10.10.10.4
  subvolumes brick
end-volume
```

Isi konfigurasi diatas sistem akan mereplikasi data yang pada *directory /var/www/* dan memperbolehkan replikasi pada *server* dengan *IP-address 10.10.10.2, 10.10.10.3, 10.10.10.4*. kemudian dibuat symlink dengan menggunakan perintah berikut :

```
ln -s /usr/local/sbin/glusterfsd /sbin/glusterfsd
```

Agar *service GlusterFS* berjalan otomatis ketika sistem mulai hidup dibuat *startup link* dengan perintah berikut :

```
chkconfig --levels 35 glusterfsd on
```

Service dapat dijalankan manual dengan perintah :

```
/etc/init.d/glusterfsd start
```


Instalasi GlusterFS client.

Langkah awal yang dilakukan adalah meng-*install* Development Tools, Development Libraries, libibverbs-devel, dan fuse-devel menggunakan perintah sebagai berikut:

```
yum groupinstall 'Development Tools'
```

```
yum groupinstall 'Development Libraries'
```

```
yum install libibverbs-devel fuse-devel
```

Kemudian *load* modul kernel fuse dengan perintah :

```
modprobe fuse
```

Selanjutnya dibuat *file* /etc/rc.module dengan perintah sebagai berikut :

```
vi /etc/rc.modules
```

Isi *file* /etc/rc.module adalah sebagai berikut :

```
modprobe fuse
```

Dengan demikian modul kernel fuse akan ter-*load* otomatis ketika sistem *booting*. Dan langkah berikutnya membuat *file* /etc/rc.module menjadi *executable* dengan menggunakan perintah berikut :

```
chmod +x /etc/rc.modules
```

Setelah proses persiapan selesai langkah berikutnya adalah meng-*install* GlusterFS 3.1.3 yang di-*download* dengan menggunakan perintah :

```
wget
http://download.gluster.com/pub/gluster/glusterfs/3.1/3.1.3/glusterfs-3.1.3.tar.gz
```

Kemudian *file* `glusterfs-3.1.3.tar.gz` diekstrak dengan perintah berikut:

```
tar xvfz glusterfs-3.1.3.tar.gz
```

Langkah selanjutnya adalah masuk *folder* `glusterfs-3.1.3` dan melakukan `./configure` dengan perintah sebagai berikut:

```
[root@s1 ~]# cd glusterfs-3.1.3
[root@s1 glusterfs-3.1.3]# ./configure
```

Pada akhir `./configure` akan dilihat skrip berikut :

```
[...]
GlusterFS configure summary
=====
FUSE client           : yes
Infiniband verbs     : yes
epoll IO multiplex   : yes
argp-standalone      : no
fusermount           : no
readline             : yes
georeplication       : no
```

Setelah dilakukan *configure* GlusterFS di-*install* dengan perintah berikut :

```
Make && make install
ldconfig
```

Setelah ter-*install* dapat dilihat versi GlusterFS dengan perintah berikut :

```
[root@s1 ~]# glusterfs --version
glusterfs 3.1.3 built on Jan 11 2012 02:23:39
Repository revision: v3.1.3
Copyright (c) 2006-2010 Gluster Inc. <http://www.gluster.com>
GlusterFS comes with ABSOLUTELY NO WARRANTY.
You may redistribute copies of GlusterFS under the terms of the GNU
Affero General Public License.
```

GlusterFS yang telah di-*install* adalah Glusterfs versi 3.1.3. langkah selanjutnya adalah membuat beberapa folder untuk tempat *file* konfigurasi Glusterfs.

```
mkdir /etc/glusterfs
```

File konfigurasi dibuat dengan perintah berikut :

```
vi /etc/glusterfs/glusterfs.vol
```

File glusterfs.vol diisi dengan skrip sebagai berikut :

```
volume remotel
  type protocol/client
  option transport-type tcp
  option remote-host s1.widhi.web.id
  option remote-subvolume brick
end-volume

volume remote2
  type protocol/client
  option transport-type tcp
  option remote-host s2.widhi.web.id
  option remote-subvolume brick
end-volume

volume replicate
  type cluster/replicate
  subvolumes remotel remote2
end-volume

volume writebehind
  type performance/write-behind
  option window-size 1MB
  subvolumes replicate
end-volume

volume cache
  type performance/io-cache
  option cache-size 512MB
  subvolumes writebehind
end-volume
```

Instalasi LVS dan Piranha

Untuk memeriksa kernel linux telah mendukung LVS dilakukan perintah sebagai berikut :

```
[root@localhost ~]# grep -i ip_vs /boot/config-2.6.18-238.el5
CONFIG_IP_VS=m
# CONFIG_IP_VS_DEBUG is not set
CONFIG_IP_VS_TAB_BITS=12
CONFIG_IP_VS_PROTO_TCP=y
CONFIG_IP_VS_PROTO_UDP=y
CONFIG_IP_VS_PROTO_ESP=y
CONFIG_IP_VS_PROTO_AH=y
CONFIG_IP_VS_RR=m
CONFIG_IP_VS_WRR=m
CONFIG_IP_VS_LC=m
CONFIG_IP_VS_WLC=m
CONFIG_IP_VS_LBLC=m
CONFIG_IP_VS_LBLCR=m
CONFIG_IP_VS_DH=m
CONFIG_IP_VS_SH=m
CONFIG_IP_VS_SED=m
CONFIG_IP_VS_NQ=m
CONFIG_IP_VS_FTP=m
```

Jika keluaran sama dengan diatas maka kernel telah mendukung LVS. Untuk mengelolah Linux Virtual Server (LVS) diperlukan *tool* ipvsadm. Untuk menginstall ipvsadm digunakan perintah berikut :

```
[root@localhost ~]# yum install ipvsadm -y
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: mirrors.hostemo.com
 * extras: mirrors.hostemo.com
 * updates: mirrors.hostemo.com
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package ipvsadm.i386 0:1.24-13.el5 set to be updated
--> Finished Dependency Resolution

Dependencies Resolved

=====
Package Arch Version Repository
Size
=====
Installing:
ipvsadm i386 1.24-13.el5 base
33 k
```

```

Transaction Summary
=====
=====
Install      1 Package(s)
Upgrade     0 Package(s)

Total download size: 33 k
Is this ok [y/N]: y
Downloading Packages:
ipvsadm-1.24-13.el5.i386.rpm | 33 kB
00:00
Running rpm_check_debug
Running Transaction Test
Finished Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing                          : ipvsadm
1/1

Installed:
  ipvsadm.i386 0:1.24-13.el5

Complete!

```

Konfigurasi LVS-NAT dengan ipvsadm melalui *command line* berikut :

```
[root@localhost ~]#ipvsadm -A -t 172.20.15.204:80 -s lc
```

Mendefinisikan *real server* dengan perintah berikut :

```
[root@localhost ~]#ipvsadm -a -t 172.20.15.204:80 -r 10.10.10.2:80 -m
[root@localhost ~]#ipvsadm -a -t 172.20.15.204:80 -r 10.10.10.3:80 -m
[root@localhost ~]#ipvsadm -a -t 172.20.15.204:80 -r 10.10.10.4:80 -m
```

Untuk melihat tabel virtual service kernel Linux adalah dengan perintah sebagai berikut:

```
[root@localhost ~]# ipvsadm -l -n
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port          Forward Weight ActiveConn InActConn
TCP  172.20.15.204:80 lc
  -> 10.10.10.4:80                 Masq    1      0          0
  -> 10.10.10.3:80                 Masq    1      0          0
  -> 10.10.10.2:80                 Masq    1      0          0
```

Agar virtual service secara otomatis aktif ketika sistem booting adalah dengan melakukan perintah berikut :

```
[root@localhost ~]# ipvsadm -Sn
```

```
-A -t 172.20.15.204:80 -s lc
-a -t 172.20.15.204:80 -r 10.10.10.4:80 -m -w 1
-a -t 172.20.15.204:80 -r 10.10.10.3:80 -m -w 1
-a -t 172.20.15.204:80 -r 10.10.10.2:80 -m -w 1
[root@localhost ~]# ipvsadm -Sn > /etc/ipvsadm.rules
[root@localhost ~]# chkconfig ipvsadm on
```

Konfigurasi LVS juga dapat dilakukan dengan menggunakan *tool* dengan GUI (*Graphical User Interface*) yaitu dengan meng-*install* Piranha. Cara meng-*install* piranha dapat dilakukan dengan perintah berikut :

```
[root@localhost ~]# yum install piranha
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: centos.idrepo.or.id
 * extras: centos.idrepo.or.id
 * updates: mirrors.sin3.sg.voxel.net
Setting up Install Process
Resolving Dependencies
--> Running transaction check
----> Package piranha.i386 0:0.8.4-24.el5 set to be updated
--> Processing Dependency: php for package: piranha
--> Processing Dependency: httpd for package: piranha
--> Running transaction check
----> Package httpd.i386 0:2.2.3-63.el5.centos.1 set to be updated
base/filelists | 3.0 MB
01:21
extras/filelists_db | 212 kB
00:05
updates/filelists_db | 1.0 MB
00:15
----> Package php.i386 0:5.1.6-32.el5 set to be updated
--> Processing Dependency: php-common = 5.1.6-32.el5 for package: php
--> Processing Dependency: php-cli = 5.1.6-32.el5 for package: php
--> Running transaction check
----> Package php-cli.i386 0:5.1.6-32.el5 set to be updated
--> Processing Dependency: php-common = 5.1.6-27.el5_5.3 for package:
php-ldap
----> Package php-common.i386 0:5.1.6-32.el5 set to be updated
--> Running transaction check
----> Package php-ldap.i386 0:5.1.6-32.el5 set to be updated
--> Finished Dependency Resolution

Dependencies Resolved

=====
Package Arch Version Repository
Size
=====
Installing:
 piranha i386 0.8.4-24.el5 base
710 k
Installing for dependencies:
 httpd i386 2.2.3-63.el5.centos.1 updates
1.2 M
```

```

php                i386                5.1.6-32.e15                base
2.3 M
Updating for dependencies:
php-cli            i386                5.1.6-32.e15                base
2.1 M
php-common         i386                5.1.6-32.e15                base
153 k
php-ldap           i386                5.1.6-32.e15                base
37 k

Transaction Summary
=====
Install            3 Package(s)
Upgrade            3 Package(s)

Total download size: 6.5 M
Is this ok [y/N]: y
Downloading Packages:
(1/6): php-ldap-5.1.6-32.e15.i386.rpm | 37 kB
00:01
(2/6): php-common-5.1.6-32.e15.i386.rpm | 153 kB
00:03
(3/6): piranha-0.8.4-24.e15.i386.rpm | 710 kB
00:28
(4/6): httpd-2.2.3-63.e15.centos.1.i386.rpm | 1.2 MB
00:19
(5/6): php-cli-5.1.6-32.e15.i386.rpm | 2.1 MB
00:55
(6/6): php-5.1.6-32.e15.i386.rpm | 2.3 MB
01:13
-----
Total                                                    36 kB/s | 6.5 MB
03:04
Running rpm_check_debug
Running Transaction Test
Finished Transaction Test
Transaction Test Succeeded
Running Transaction
  Updating : php-common
 1/9
  Installing : httpd
 2/9
  Updating : php-cli
 3/9
  Installing : php
 4/9
  Installing : piranha
 5/9
  Updating : php-ldap
 6/9
  Cleanup : php-ldap
 7/9
  Cleanup : php-cli
 8/9
  Cleanup : php-common
 9/9

Installed:
piranha.i386 0:0.8.4-24.e15

```



```

Dependency Installed:
  httpd.i386 0:2.2.3-63.el5.centos.1          php.i386 0:5.1.6-32.el5

Dependency Updated:
  php-cli.i386 0:5.1.6-32.el5              php-common.i386 0:5.1.6-32.el5
  php-ldap.i386 0:5.1.6-32.el5

Complete!

```

Setelah Piranha ter-*install* langkah berikutnya adalah menentukan *password* untuk masuk aplikasi tersebut.

```

[root@localhost ~]# /usr/sbin/piranha-passwd
New Password:
Verify:
Adding password for user piranha

```

Aplikasi Piranha berjalan pada *web browser (web based)*. Untuk menjalankannya dilakukan perintah berikut :

```

[root@localhost ~]# /etc/init.d/piranha-gui start
Starting piranha-gui: [ OK ]

```

Tampilan awal Piranha

