PERANCANGAN KLASTER SERVER WEB DENGAN AVAILABILITAS TINGGI MENGGUNAKAN TEKNOLOGI *FAILOVER*, *LOAD BALANCING* DAN *DISTRIBUTED FILE SYSTEM*

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Tiara Erlinda
NIM: 135150201111011

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2018

PENGESAHAN

PERANCANGAN KLASTER SERVER WEB DENGAN AVAILABILITAS TINGGI MENGGUNAKAN TEKNOLOGI *FAILOVER*, *LOAD BALANCING* DAN *DISTRIBUTED FILE SYSTEM*

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Tiara Erlinda
NIM: 135150201111011

Skripsi ini telah diuji dan dinyatakan lulus pada
3 Agustus 2018
Telah diperiksa dan disetujui oleh:

Pembimbing I	Pembimbing II
a.n.	
Mahendra Data, S.Kom., M.Kom	Reza Andria Siregar, S.T., M.Kom.
NIK: 2015038611171001	NIP: 19790621 200604 1 003

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoro Kurniawan, S.T, M.T, Ph.D
NIP: 19710518 200312 1 001

ii

PERNYATAAN ORISINALITAS

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

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

Malang, 18 Juli 2018

Tiara Erlinda

NIM: 135150201111011

iii

KATA PENGANTAR

Puji syukur penulis panjatkan kehadirat Allah SWT, karena berkat rahmat serta bimbingan-Nya, penulis dapat menyelesaikan penulisan skripsi dengan baik. Penulisan skripsi ini diajukan untuk memenuhi sebagian persyaratan untuk memperoleh gelar Sarjana Komputer pada Proram Studi Teknik Informatika Jurusan Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya. Judul yang penulis ajukan adalah Perancangan Klaster Server Web dengan Availabilitas Tinggi Menggunakan Teknologi *Failover*, Load Balancing dan Distributed File System .

Dalam penyusunan dan penulisan skripsi ini tidak terlepas dari bantuan, bimbingan serta dukungan dari berbagai pihak. Oleh karena itu dalam kesempatan ini penulis dengan senang hati menyampaikan terima kasih kepada:

1. Bapak Mahendra Data, S.Kom., M.Kom selaku dosen pembimbing I yang telah tulus dan ikhlas memberikan bimbingan, motivasi, arahan dan saran terhadap penulis selama menyusun skripsi.
2. Bapak Reza Andria Siregar, S.T, M.Kom dan pembimbing II yang telah dengan telah tulus dan ikhlas memberikan bimbingan, motivasi, arahan dan saran terhadap penulis selama menyusun skripsi.
3. Bunda, Papa dan Dede. Keluarga yang senantiasa mendoakan dan mendukung penulis dalam menyelesaikan skripsi.
4. Teman-teman yang selalu mendukung penulis dan terkadang menghambat penulis dalam mengerjakan skripsi.
5. Seluruh Dosen Program Studi Informatika Universitas Brawijaya atas kesediaan membagi ilmunya kepada penulis.
6. Teman–teman angkatan 2013 Informatika dan fakultas ilmu komputer.
7. Seluruh pihak yang telah membantu kelancaran penulisan skripsi yang tidak dapat penulis sebutkan satu persatu.
8. Erziansyah Putra, Terimakasih.

Penulis menyadari bahwa dalam penyusunan skripsi ini masih banyak kekurangan. Penulis mengharapkan adanya saran dan kritik membangun dari para pembaca demi kesempurnaan skripsi ini. Semoga skripsi ini dapat memberikan manfaat bagi semua dan berguna bagi pengembangan ilmu pengetahuan.

Malang, 3 Agustus 2018

Tiara Erlinda

tiara.erlinda14@gmail.com

ABSTRAK

Perkembangan yang terjadi pada teknologi *World Wide Web* (WWW) berdampak kepada bertambahnya situs *web* setiap harinya. Penggunaan *web* untuk kehidupan sehari-hari sangat berpengaruh untuk berbagai aspek kehidupan seperti untuk berkomunikasi, hiburan, serta pendidikan. Dibutuhkan suatu sistem yang memiliki tingkat availabilitas tinggi *(High Availability*) agar dapat beroperasi secara terus-menerus dan menjamin tidak terjadi kegagalan sehingga sistem dapat memberikan layanan secara berkesinambungan tanpa terjadi gangguan kepada pengguna. Solusi yang diajukan peneliti adalah perancangan klaster *server web* dengan tingkat ketersediaan tinggi dengan teknologi *failover* menggunakan Keepalived. Terdapat juga teknologi *load balancing* menggunakan Varnish yang mendukung untuk penditribusian beban kerja kepada *web server* dan *distributed file system* menggunakan GlusterFS untuk berbagi *file* antar *server*. Penelitian dilakukan menggunakan LAMP server yaitu Linux Apache, MySql dan PHP. Apache *server* bertugas sebagai *web server*. Pengujian dilakukan dari segi *downtime*, *throughput*, *CPU usage*, *memory usage* dan fungsional. Hasil dari penelitian didapatkan bahwa nilai *downtime* dengan rata-rata selama 1 detik ketika *server* utama mati. Nilai *throughput* untuk koneksi tertinggi diperoleh saat jumlah koneksi berjumlah 40 dan terendah saat jumlah koneksi berjumlah 100. Nilai *CPU usage* dan *memory usage* mengalami peningkatan sampai 68,3 % untuk *CPU* dan 65.5 % untuk *memory* tergantung jumlah koneksi. Hasil pengujian fungsional didapatkan bahwa sistem sudah berjalan sesuai fungsinya. Berdasarkan hasil tersebut, sistem ini dapat menjadi solusi untuk sistem yang membutuhkan ketersediaan tinggi.

Kata kunci: *high availability, failover, load balancing, distributed file system, varnish, keepalived, glusterfs, lamp server.*

ABSTRACT

The developments of World Wide Web (WWW) technology has an impact on the increasing website every day. The use of the web for everyday life is very influential for various aspects of life such as to communicate, entertainment, and education. It takes a system that has a high availability level to operate continuously and ensures no failure so that the system can provide continuous service without interruption to the user. The researcher's solution is to design a high availability web server cluster with failover technology using Keepalived. There is also load balancing technology using Varnish which supports distributing workloads to web servers and distributed file systems using GlusterFS to share files between servers. The study was conducted using LAMP server ie Linux Apache, MySql and PHP. Apache server is assigned as web server. Testing is done in terms of downtime, throughput, CPU usage, memory usage and functional. The results of the research found that the value of downtime with an average of 1 second when the main server is dead. The highest throughput value for the connection is obtained when the number of connections is 40 and the lowest when the number of connections is 100. The usage and memory usage values increase up to 68.3% for CPU and 65.5% for memory depending on the number of connections. The result from functional test proved that the system is running according to its function. Based on these results, this system can be a solution for systems that require high availability.

Keywords: *high availability, failover, load balancing, distributed file system, varnish, keepalived, glusterfs, lamp server.*

DAFTAR ISI

KATA PENGANTAR iv
DAFTAR ISI vii
DAFTAR TABEL x
DAFTAR GAMBAR xii
DAFTAR LAMPIRAN xiv
BAB 1 PENDAHULUAN 1
1.1 Latar belakang 1
1.2 Rumusan masalah 2
1.3 Tujuan 3
1.4 Manfaat 3
1.5 Batasan masalah 3
1.6 Sistematika pembahasan 3
BAB 2 LANDASAN KEPUSTAKAAN 5
2.1 Kajian Pustaka 5
2.2 High Availability 7
2.3 Klaster Server Web 7
2.4 LAMP Server 8
2.5 Load Balancing 9
2.5.1 Varnish 9
2.6 *Failover* 10
2.6.1 Keepalived 11
2.7 Distributed File System 12
2.7.1 GlusterFS 12
BAB 3 METODOLOGI 13
3.1 Perumusan Masalah 13
3.2 Studi Literatur 13
3.3 Analisa Kebutuhan 14
3.4 Perancangan 14
3.5 Implementasi 14
3.6 Pengujian dan Analisa Hasil 14
3.7 Kesimpulan dan Saran 15

BAB 4 Perancangan 16
4.1 Analisis Kebutuhan 16
4.1.1 Kebutuhan Fungsional 16
4.1.2 Kebutuhan Non Fungsional 16
4.2 Perancangan Topologi 17
4.3 Alur Kerja 17
4.3.1 Alur Kerja Sistem 18
4.3.2 Alur Kerja *Failover* Keepalived 19
4.3.3 Alur Kerja Load Balancing Varnish 20
4.3.4 Alur Kerja Distributed File System GlusterFS 21
4.4 Pengujian Sistem 22
4.4.1 Pengujian Fungsionalitas 23
4.4.2 Pengujian Downtime 24
4.4.3 Pengujian Throughput 24
4.4.4 Pengujian *CPU usagae* dan *Memory* Usage 25
BAB 5 IMPLEMENTASI 26
5.1 Implementasi Ubuntu Server 26
5.2 Implementasi Web Server 27
5.3 Implementasi Database Server 28
5.3.1 Konfigurasi Wordpress 30
5.4 Implementasi GlusterFS 31
5.5 Implementasi Varnish 33
5.6 Implementasi Keepalived 36
BAB 6 Pengujian 38
6.1 Pengujian 38
6.1.1 Pengujian Downtime 38
6.1.2 Pengujian Throughput 40
6.1.3 Pengujian *CPU usagae* dan *Memory* Usage 44
6.1.4 Pengujian Fungsionalitas 51
6.2 Analisa dan Hasil 52
BAB 7 PENUTUP 53
7.1 Kesimpulan 53

7.2 Saran 53
DAFTAR PUSTAKA..... 54
LAMPIRAN I 56
LAMPIRAN II 72

DAFTAR TABEL

Table 2.1. Kajian Pustaka 6
Table 4.1 Skenario Pengujian Sistem 23
Table 4.2 Pengujian Kebutuhan Fungsional 23
Table 4.3 Tabel Pengujian Downtime 24
Table 4.4 Pengujian Throughput 25
Table 4.5 Tabel Pengujian *CPU usagae* dan *Memory* Usage 25
Table 5.1. Perubahan *Interface* 26
Table 5.2 Konfigurasi Alamat IP statik Load Balancer 1 26
Table 5.3 Konfigurasi Alamat IP statik Load Balancer 2 26
Table 5.4 Konfigurasi Alamat IP statik Web Server 1 27
Table 5.5 Tabel Konfigurasi Alamat IP statik Web Server 2 27
Table 5.6 Konfigurasi Alamat IP statik Database Server 27
Table 5.7 Perintah Instalasi PHP 7.0 *modules* 27
Table 5.8 Perintah Untuk Unduh File Wordpress 27
Table 5.9 Perintah *Unpack File* 28
Table 5.10 Perintah Memindahkan Folder 28
Table 5.11 Merujuk ke direktori 28
Table 5.12. Melakukan *restart* apache 28
Table 5.13. Instalasi Mysql-server 28
Table 5.14. Masuk ke root mysql 29
Table 5.15 Membuat database wordpress 29
Table 5.16 Konfigurasi wp-config.php 30
Table 5.17. Instalasi GluterFS 31
Table 5.18. Membuat Volume 31
Table 5.19. Memulai volume glusterfs 31
Table 5.20. Memeriksa Volume Info Glusterfs 32
Table 5.21. Perintah untuk Web Server 2 Mengakses Volume 32
Table 5.22 Instalasi glusterfs client 32
Table 5.23. Membuat Direktori 32
Table 5.24. Mount Pada Web Server 2 33
Table 5.25. Edit fstab 33

Table 5.26. Konfigurasi fstab 33
Table 5.27. Melakukan Copy File 33
Table 5.28. Instalasi Varnish 34
Table 5.29. Membuka file varnish.service 34
Table 5.30. Konfigurasi Port 34
Table 5.31. Membuka file default.vcl 34
Table 5.32. Konfigurasi Varnish Algoritma *Round robin* 34
Table 5.33. instalasi keepalived 36
Table 5.34. Konfigurasi Keepalived load balancer 1 36
Table 6.1. Perintah Ping 38
Table 6.2. Hasil Pengujian Downtime Skenario 1 39
Table 6.3. Hasil Pengujian Downtime Skenario 3 39
Table 6.4 Perintah httperf 40
Table 6.5. Hasil Throughput Skenario 1 41
Table 6.6. Hasil Throughput Skenario 2 42
Table 6.7. Hasil Throughput Skenario 3 43
Table 6.8. Hasil *CPU usagae* dan *Memory* Usage Skeqnario 1 Algoritma *Round robin* 44
Table 6.9. Hasil *CPU usagae* dan *Memory* Usage Skenario 1 Algoritma Fallback. 45
Table 6.10 Hasil *CPU usagae* dan *Memory* Usage Skenario 2 Algoritma *Round robin* 47
Table 6.11 Hasil *CPU usagae* dan *Memory* Usage Skenario 2 Algoritma Fallback 47
Table 6.12 Hasil *CPU usagae* dan *Memory* Usage Skenario 3 Algoritma *Round robin* 49
Table 6.13 Hasil *CPU usagae* dan *Memory* Usage Skenario 3 Algoritma Fallback 49
Table 6.14. Hasil Pengujian Fungsionalitas 51

DAFTAR GAMBAR

Gambar 2.1. Klaster Server Web 8
Gambar 2.2. *Load Balancing* 9
Gambar 2.3. Sebelum terjadi *Failover* 11
Gambar 2.4 Setelah terjadi failover 11
Gambar 2.5. Arsitektur GlusterFS 12
Gambar 3.1. Metodologi Penelitian 13
Gambar 4.1 Perancangan Topologi 17
Gambar 4.2 Alur Kerja Sistem 18
Gambar 4.3 Alur Kerja *Failover* Keepalived 19
Gambar 4.4 Alur Kerja Load Balancing Varnish 20
Gambar 4.5 Alur Kerja Algoritma *Round Robin* 21
Gambar 4.6 Alur Kerja Algoritma *Fallback* 21
Gambar 4.7 Alur Kerja GlusterFS 22
Gambar 5.1 Konfigurasi bind-address 29
Gambar 5.2 Tampilan Wordpress 31
Gambar 5.3. GlusterFS volume info 32
Gambar 5.4 GLusterFS sukses 32
Gambar 5.5. Direktori data-server 33
Gambar 5.6. Direktori data-client 33
Gambar 6.1. Hasil Packet loss 38
Gambar 6.2. Jumlah Packet loss 39
Gambar 6.3. Contoh Hasil Throughput 40
Gambar 6.4. Grafik Hasil Throughput Skenario 1 41
Gambar 6.5 Grafik Hasil Throughput Skenario 2 42
Gambar 6.6. Grafik Hasil Throughput Skenario 3 43
Gambar 6.7. Grafik Hasil Pengujian *CPU usagae* Skenario 1 Algoritma *Round robin* 45
Gambar 6.8. Grafik Hasil Pengujian *Memory* Usage Skenario 1 Algoritma *Round robin* 45
Gambar 6.9. Hasil Pengujian *CPU usagae* Algoritma Fallback 46
Gambar 6.10. Hasil Pengujian *Memory* Usage Algoritma Fallback 47

Gambar 6.11. Hasil Pengujian *CPU usagae* Skenario 2 48
Gambar 6.12. Hasil Pengujian *Memory* Usage Skenario 2 49
Gambar 6.13. Hasil Pengujian *CPU usagae* untuk skenario 3 50
Gambar 6.14. Hasil Pengujian *Memory* Usage skenario 3 51

DAFTAR LAMPIRAN

LAMPIRAN I 56
LAMPIRAN II 72

BAB 1 PENDAHULUAN

1.1 Latar belakang

Perkembangan yang terjadi pada teknologi *World Wide Web (WWW)* menciptakan ribuan situs web setiap harinya. Bukan hanya sebagai sumber informasi, *web* juga dimanfaatkan untuk berkomunikasi antar pengguna, hiburan dan kepentingan bisnis. Hal itu memungkinkan untuk para pengguna agar saling berbagi pengetahuan yang efisien dan efektif bahkan melalui jejaring sosial sebagai bentuk pengorganisasian aktivitas para pengguna tanpa ada batas geografis (Yuan, et al., 2015). Penggunaan web untuk kehidupan sehari hari sangat berpengaruh untuk berbagai kepentingan. Hal itu mengharuskan suatu web untuk mampu melayani banyaknya *request* setiap harinya tanpa ada gangguan atau permasalahan yang menyebabkan web tersebut tidak dapat di akses. Dibutuhkannya suatu sistem yang memiliki tingkat availabilitas tinggi (*High Availability)* agar dapat beroperasi secara terus-menerus dan menjamin tidak terjadi kegagalan pada sistem tersebut. Sehingga sistem dapat memberikan layanan tanpa terjadi gangguan kepada pengguna. Terdapat berbagai teknologi yang mendukung suatu sistem agar memiliki tingkat availabilitas tinggi.

Web klaster merupakan mampu meningkatkan kemampuan pelayanan dan ketersediaan *(availability)* dengan menambahkan lebih dari satu web server. (Zongyu & Xingxuan, 2015). Sebuah server tunggal tidak akan mampu memenuhi permintaan dalam jumlah yang banyak dengan skalabilitas tinggi dan kinerja yang handal. Klaster web server menjadi solusi untuk permasalahan dalam menangani permintaan dalam jumlah yang banyak. Klaster web server berisikan lebih dari satu web server untuk menyediakan layanan kepada pengguna. Web Server memiliki tanggung jawab untuk mengelola sumber daya dan menyediakan layanan untuk pengguna.

Menjamin ketersediaan layanan merupakan hal yang penting untuk layanan web. Meskipun sebuah klaster web server memiliki jumlah web server yang banyak, tetapi hanya menggunakan satu *hostname* dan satu alamat IP *virtual* untuk menyediakan layanan untuk para pengguna. Sehingga dibutuhkan sebuah mekanisme yang dapat menangani seluruh permintaan pada situs dan mendistribusikan beban tugas diantara semua web server (Zongyu & Xingxuan, 2014). *Load Balancing* menjadi usulan untuk meningkatkan performa pada klaster web server (Li, et al., 2010). *Load Balancing* merupakan suatu mekanisme yang mendistribusikan beban kerja kepada dua atau lebih jalur koneksi sehingga dapat berjalan dengan optimal dan menghindari *overload* pada salah satu jalur koneksi. *Laod balancer* menentukan server mana yang akan menerima permintaan layanan dari pengguna. Selain itu, *load balancer* juga mampu menghentikan akses ke server yang sedang mengalami masalah dan meneruskannya ke server lain yang dapat memberikan layanan. Varnish merupakan suatu *reverse proxy HTTP* yang memiliki keunggulan utama sebagai *cache.* Selain sebagai *cache,* varnish memiliki

keunggulan dalam *load balancing.* Varnish saat ini banyak digunakan oleh situs *web* terkenal seperti facebook, wikia dan Slashdot (Kamp, 2010).

Terdapat teknologi *failover* yang merupakan komponen penting pada suatu sistem untuk mengoptimalkan ketersediaan dan layanan kepada pengguna. Sehingga aktivitas dalam menangani permintaan kepada pengguna dapat berjalan secara terus menerus. *Failover* melakukan perpindahan dari satu sub sistem utama ke sub sistem *backup.* Keepalived merupakan *software routing.* Tujuan dari keepalived ialah menyediakan fasilitas yang sederhana untuk ketersediaan tinggi pada sistem. Keepalived menggunakan protokol VRRP sehingga perlu dipasang sebuah alamat IP yang akan digunakan sebagai *gateway* dari suatu jaringan lokal (Ha, et al., 2015). Perpindahan dapat dilakukan secara otomatis oleh sistem dan sistem akan berjalan terus menerus dan tidak mengganggu layanan.

Selain itu, kebutuhan dalam penyimpanan file terdistribusi pada suatu sistem juga dibutuhkan agar mampu untuk mengakses dan memproses data. *Distributed File System* mempermudah dalam melakukan berbagi informasi dan *file* antara banyak *user* dalam satu jaringan dengan terkendali dan aman. GlusterFS merupakan aplikasi *open sourse* yang bermanfaat untuk ruang penyimpanan untuk manajemen sistem berkas terdistribusi, aplikasi ini memiliki berbagai karakteristik seperti *scalable, reliable* dan *flexible* ekspansi disk. GlusterFS mempermudah untuk penyimpanan terdistribusi. Menggunakan GlusterFS mampu memudahkan dalam pengembangan, *update* dan *debug* (Huang, et al., 2015).

Penelitian yang dilakukan oleh peneliti ialah membuat suatu klaster server web dengan tingkat availabilitas tinggi. Varnish bertugas sebagai *load balancer,* Keepalived sebagai *failover* dan glusterFS sebagai *distributed file system.* Penelitian menggunakan LAMP server yaitu linux, apache, mysql dan php sebagai web server. Kelebihan dari apache web server ialah mudah dan fleksibel dalam melakukan konfigurasi. Terdapat situs besar yang menjadikan apache sebagai web server seperti alibaba, sina, dan baidu. (Hong, et al., 2010). Sehingga penggunaan apache sebagai web server pada penelitian menjadi komponen mendukung yang mengoptimalkan berjalannya sistem dan penyediaan layanan kepada pengguna.

1.2 Rumusan masalah

Berdasarkan latar belakang yang telah dijelaskan, maka terdapat rumusan masalah pada penelitian adalah sebagai berikut:

1. Bagaimana perancangan klaster server web dengan availabilitas tinggi menggunakan teknologi *failover*, *load balancing* dan *distributed file system?*
2. Bagaimana performa dan kehandalan yang dihasilkan dari perancangan klaster server web dengan availabilitas tinggi menggunakan teknologi *failover*, *load balancing* dan *distributed file system?*

1.3 Tujuan

Dari rumusan masalah yang ada, maka tujuan yang ingin dicapai dalam penelitian ini adalah sebagai berikut:

1. Untuk melakukan perancangan klaster server web dengan availabilitas tinggi menggunakan teknologi *failover*, *load balancing* dan *distributed file system.*
2. Untuk menguji performa dan kehandalan yang dihasilkan dari perancangan perancangan klaster server web dengan availabilitas tinggi menggunakan teknologi *failover*, *load balancing* dan *distributed file system.*

1.4 Manfaat

Penelitian yang dilakukan diharapkan dapat memberikan manfaat yang baik dan berguna. Manfaat yang diharapkan dari penelitian ini adalah sebagai berikut:

1. Untuk merancang klaster server web dengan availibilitas tinggi yang menggunakan teknologi *failover*, *load balancing* dan *distributed file system*.
2. Untuk dapat mengetahui performa dan kehandalan klaster server web yang telah dirancang.
3. Penelitian yang dilakukan dapat dikembangkan untuk penelitian selanjutnya.

1.5 Batasan masalah

Batasan masalah dari penelitian yang dillakukan oleh penulis adalah sebagai berikut:

1. Penelitian ini dilakukan pada topologi sederhana dan spesifikasi server minimalis.
2. Pengujian dilakukan secara virtual menggunakan VirtualBox.
3. Pengujian tingkat availabilitas diuji dengan parameter *downtime.*
4. Pengujian hanya melakukan *request* pada satu halaman (page).

1.6 Sistematika pembahasan

1. **BAB I. PENDAHULUAN**
   Bab pertama ialah mengejelaskan semua hal yang terkait dengan penelitian. Diantaranya ialah latar belakang yang berisi secara ringkas teori yang berhubungan dengan penelitian, rumusan masalah, tujuan masalah, manfaat penilitian, batasan masalah dan sistermatika pembahasan.
2. **BAB II. LANDASAN KEPUSTAKAAN**
   Bab ini membahas mengenai dasar teori dan ulasan referensi-refernsi yang terkait dengan penelitian. Teori-teori yang diperlukan untuk mendukung pengerjaan penelitian. Ulaan-ulasan mengenai topik yang berhubungan dengan *high availability,* LAMP, varnish, glusterFS dan keepalived.
3. **BAB III. METODOLOGI**
   Pada bab ini menjelaskan mengenai langkah-langkah yang dilakukan penulis dalam melakukan penelitian. Diantaranya ialah perumusan masalah, studi

literature, analisa kebutuhan dan perancangan, implementasi, pengujian dan analisa hasil lalu kesimpulan dan saran.

4. **BAB IV. PERANCANGAN**
   Bab ini menjelaskan mengenai perancangan untuk penelitian yang akan dilakukan. Perancangan yang dibuat yaitu *load balancer* yang bertugas dalam membagi kerja, web *server* yang digunakan untuk penelitian, database server dan GlusterFS server.

5. **BAB V. IMPLEMENTASI**
   Bab ini menjelaskan mengenai sistem yang akan diterapkan berdasarkan perancangan yang telah dibuat dan langkah-langkah dalam pengerjaan penelitian akan ditampilkan beserta dengan gambar-gambar dari implementasi yang dilakukan.

6. **BAB VI. PENGUJIAN**
   Bab pengujian menjelaskan mengenai pengujian beserta pembahasan yang dilakukan. Lalu menyajikan hasil dari penelitian yang telah dilakukan sesuai dengan skenario dan perancangan yang telah dibuat.

7. **BAB VII. PENUTUP**
   Bab yang terakhir akan memuat kesimpulan dari hasil dari penelitian yang dilakukan secara rinci. Serta keseluruhan uraian bab-bab sebelumnya,dan saran-saran dari hasil yang diperoleh diharapkan dapat bermanfaat dalam pengembangan selanjutnya.

BAB 2 LANDASAN KEPUSTAKAAN

2.1 Kajian Pustaka

Terdapat beberapa penelitian – penelitian sebelumnya yang menjadi referensi untuk pengerjaan penelitian ini. Penelitian pertama ialah berjudul *Perfomance Comparisons of Web Server Load Balancing Algorithms on HAProxy and Heartbeat.* Penelitian ini menggunakan HAProxy sebagai *load balancer* serta *Heartbeat* sebagi *failover.* Terdapat dua buah server yang menjadi *load balancer* dan tiga buah server sebagai *web server.* Kedua *load balancer* server akan digunakan secara aktif – *standby* sehingga ketika *load balancer* utama mengalami kegagalan maka akan dialihkan ke *load balancer* cadangan menggunakan heartbeat. Untuk sinkronisasi pada ketiga web server, penelitian ini menggunakan unison. Penelitian ini melakukan analisa dari tiga algoritma yang terdapat pada HAProxy diantaranya ialah *round robin*, *source* dan *leastconn*. Hasilnya algoritma *leastconn* lebih unggul dibandingkan algoritma lainnya. *Failover service* pada penelitian ini membutuhkan waktu 10 ms untuk melakukan perpindahan ke *load balancer* cadangan. (Prasetijo, et al., 2016)

Penelitian kedua berjudul *Highly Available Web Service Using the Raspberry Pi Cluster.* Penelitian ini menjelaskan tentang ketersedian tinggi pada layanan web dengan menggunakan klaster Raspberry pi. Ketersediaan tinggi ditunjukan dengan menerapkan web server dan *load balancing* di klaster Raspberry Pi. Penelitian ini dilakukan dalam dua bagian, bagian pertama ialah menginisialisasi Raspberry Pi, menginstal dan mengkonfigurasi server web pada dua node dan menyiapkan satu load balancer. Kedua, penelitian ini bertujuan untuk pencapaian ketersediaan tinggi dengan instalasi tambahan *load balancer* dengan *failover*. Penelitian ini berhasil diterapkan menggunakan Raspberry Pi dan mencapai tujuan *high availability* dengan perangkat lunak *open source* yang menjadi pendukung pada penelitian. (Aryal, 2017)

Penelitian ketiga berjudul *Performance Evaluation of the Apache Traffic Server and Varnish Reverse Proxies* .Penelitian ini bertujuan untuk menyelidiki kinerja dari dua reverse proxy yaitu Varnish dan Apache Traffic Server. Penelitian ini menggunakan alat Web Polygraph untuk menghasilkan berbagai jenis beban kerja *traffic web*. Untuk percobaan yang dilakukan dalam penelitian ini, hasil menunjukkan bahwa Apache Traffic Server mencapai tingkat *cache* yang lebih baik dan *throughput bandwidth* sedikit lebih baik dengan biaya penggunaan sumber daya yang lebih tinggi dari sistem dan jaringan. Varnish di sisi lain berhasil melakukan *respons rate* lebih tinggi dari *request* dengan waktu respons yang lebih baik, terutama untuk *cache*. Temuan dalam penelitian ini menunjukkan bahwa Varnish nampaknya merupakan *reverse proxy* yang lebih handal. (Bakhtiyari, 2012)

Table 2.1. Kajian Pustaka

No	Judul	Objek	Metode	Hasil
1	*Perfomance Comparisons of Web Server Load Balancing Algorithms on HAProxy and Heartbeat*	Apache HAproxy Heartbeat Unison	Penelitian ini menggunakan Apache sebagai web server,HAProxy sebagai *load balancer* serta Heartbeat sebagai *failover.* Unison sebagai sinkronisasi antar web server.	Penelitian ini menunjukan bahwa sistem mampu menangani permasalahan *load balancer* dan *failover.* Algoritma *leastconn* mengungguli dibandingkan algoritma lainnya. Serta *heartbeat* membutuhkan waktu 10ms untuk berpindah dari *load balancer* utama ke *load balancer* cadangan.
2	*Highly Available Web Service Using the Raspberry Pi Cluster*	*Raspberry Pi, Cluster Computing, High Availability, Raspbian, HAProxy, Keepalived,*	Penelitian ini diterapkan pada Raspberry Pi, dengan menggunakan Apache sebagai web server, Rsync sebagai sinkornisasi antar server, HAProxy sebagai *load balancing* dan *keepalived* sebagai *failover.*	Penelitian ini menghasilkan tujuan yang diinginkan yaitu ketersediaan tinggi pada sistem dengan menggunakan apache, rsync, haporxy dan *failover* sebagai pendukung.
3	*Performance Evaluation of the Apache Traffic Server and Varnish Revers Proxies*	*Apache Traffic Server, Varnish Reverse Proxy*	Penelitian ini meyelidiki kinerja dari Apache Traffic Server dan Varnish Reverse Proxy sebagai cache untuk meningkatkan perofrma.	Penelitian ini menghasilkan bahwa varnish lebih unggul dalam melakukan respon dan lebih unggu dibandingkan Apache Traffic Server.

2.2 High Availability

Pada teknologi informasi, high availability (HA) dapat diartikan sebagai sebuah sistem atau komponen yang dapat beroperasi secara terus – menerus atau dalam jangka waktu yang cukup lama. HA menjamin ketersediaan agar tidak terjadi kegagalan pada sistem dan menghindari resiko dari pemadaman sistem, *loss data*, data tidak lengkap atau kesalahan dalam pemrosesan pengiriman data. HA dapat di implementasikan untuk meminimalkan *downtime* aplikasi sistem dan menciptakan sistem yang dapat diandalkan. Pada teknologi klaster *high availability,* jika terdapat kegagalan pada satu node, maka dapat dipertanggung jawabkan untuk dialihkan ke node server lain. Hal itu menjamin sistem agar tetap dapat memberikan layanan. (Dudnik, 2017).

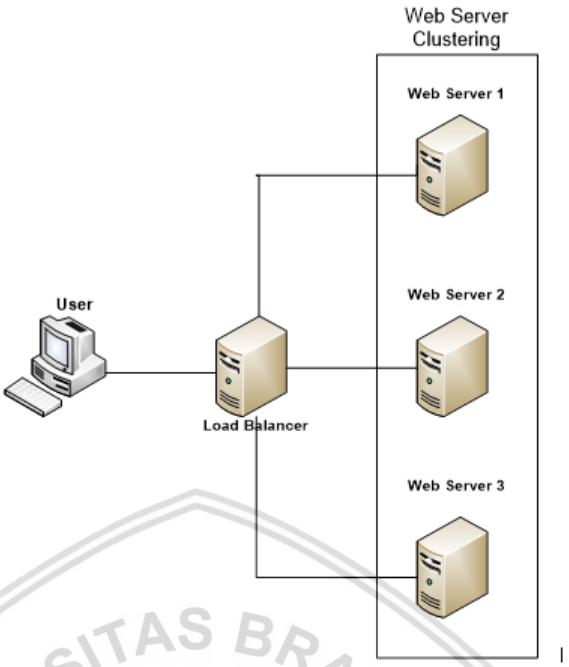
Ketersediaan layanan pada suatu sistem juga dapat dipengaruhi oleh jenis masalah yang dialami oleh suatu server. Permasalahan pada server dapat ditoleransi jika permasalahan tersebut tidak menurunkan kinerja yang signifikan. Hal ini dapat diartikan penggunaan layanan dapat terus berjalan tanpa menyadari permasalahan yang teradi. (Hidayat, 2012). Sehingga menjamin HA pada suatu sistem sangat berperan penting untuk proses kinerja dan keandalan yang disediakan agar tidak menggangu layanan untuk pengguna.

2.3 Klaster Server Web

Clustering atau klaster server web merupakan kumpulan dari dua atau lebih server yang saling berkerja sama untuk memberikan layanan kepada pengguna. Teknologi klaster memberikan solusi untuk menangani perpindahan tugas atau pemerataan beban diantara seluruh server. jika terdapat permasalahan pada salah satu server maka akan dialihkan pada server yang berada pada klaster yang sama. Sehingga pengguna hanya mengetahui jika hanya terdapat server tunggal yang tersedia tanpa menyadari jika terjadi kegagalan pada salah satu server pada klaster (Putra & Sugeng, 2016).

Kelebihan dari teknologi klaster ialah menghasilkan suatu sistem yang memiliki tinggal realibilitas dan availabilitas yang tinggi. Dengan menggunakan teknologi klaster maka mengantisipasi kegagalan atau kerusakan yang terjadi pada salah satu komponen yang dapat mengganggu kinerja sistem. Salah satu gangguan yang terjadi biasanya disebabkan karena server utama mati dan tidak memiliki server cadangan untuk menggantikan fungsi server utama. (Sulistyanto, 2015).

Pada Gambar 2.1 menjelaskan mengenai penggunaan klaster *web server. Load balancer* terhubung dengan klaster *web server* untuk membagi beban kerja sesuai dengan algoritma yang ditentukan. Sehingga dengan menggunakan klaster *web server* tidak hanya membebani satu server tunggal. Jika salah satu *web server* pada klaster mengalami kegagalan atau tidak tersedia, maka server lainnya yang berada pada klaster akan menjalankan tugas untuk memberikan layanan kepada pengguna.



Gambar 2.1. Klaster Server Web

2.4 LAMP Server

LAMP merupakan gabungan dari empat perangkat lunak yang dapat berkerja sama dengan baik yaitu Linux, Apache, MySQL dan PHP. Apache merupakan suatu web server yang dikembangkan oleh *Apache Software Foundation* (ASF). Apache dikemas dengan berbagai fitur, sangat cepat dan berkerja dengan baik dengan sistem operasi Linux. Fitur yang didukung oleh Apache ialah proteksi akses direktori, CGI, SSL dan beberapa fitur lainnya yang memungkinkan untuk mengembangkan suatu situs. Popularitas Apache berkembang sangat cepat menjadi solusi *web server* nomor satu. Selanjutnya MySQL merupakan suatu *database* yang memungkinkan untuk menyimpan berbagai jenis data dan mengambil data. MySQL dikemas dengan berbagai fitur seperti replikasi data, *table locking, query limiting* dan berbagai fitur lainnya. Dan PHP yang merupakan singkatan dari *HyperText Preprocessor* ialah bahasa pemrogramman yang mudah digunakan dan sesuai untuk pengembangan web dan biasa digunakan dengan HTML. (Rosebrock & Filson, 2004).

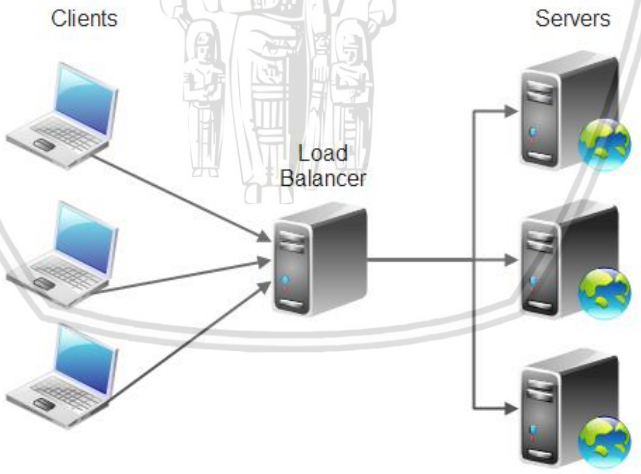
Web server menerima permintaan HTTP dari pengguna dan melayani permintaan tersebut. Web server berisikan semua sumber daya penting di internet dan berhubungan dengan permintaan dari pengguna untuk memberikan sumber daya tersebut (Singh & Kumar, 2011). Permintaan dalam jumlah besar dari pengguna dapat membenani web server menjadi *overload,* keterlambatan performa bahkan *server down.* Sehingga diperlukannya web server dengan kinerja yang efektif dan handal. Web server mengelola pelayanan untuk pengguna seperti penyimpanan, pengembalian, pembaruan dan sebagaianya. (Lan & Wang, 2012).

2.5 Load Balancing

Load balancing adalah sebuah metode yang mendistribusikan beban kerja kepada dua server atau lebih yang tersedia. Sehingga dengan *load balancing* akan memastikan bahwa beban kerja diberikan secara adil dan tidak ada server yang mengalami *overload.* Dengan *load balancing,* meningkatnya waktu respon dan juga pemanfaat sumber daya yang menjadi lebih efisien. Beban yang ada pada web server setiap detiknya tidak dapat diprediksi, sehingga akan dialokasikan secara dinamis diantara server-server untuk memenuhi kebutuhan pengguna dan menyediakan sumber daya maksimum dengan mengelompokkan keseluruhan beban yang tersedia ke server yang berbeda-beda (Gopinath & Vasudevan, 2015).

Proses *load balancing* bersifat transparan bagi end user. Server akan terlihat sebagai satu *web server* ke web *client. Load balancing* memastikan pemroresan yang cepat dan pemanfaatan yang baik. Penyebaran beban yang dilakukan oleh *load balancing* akan meningkatkan sumber daya yang tersedia (Enesi, dkk., 2017).

Pada Gambar 2.2 menjelaskan mengenai tugas dari *laod balancing. Load balancer* membagi beban kerja kepada beberapa *backend server* yang terhubung sesuai dengan algoritma yang digunakan. Dengan penggunaan *load balancer* maka tidak membebani pada satu server tunggal saja. *Client* terhubung dengan *load balancer* dan selanjutnya *load balancer* meneruskan ke beberapa server untuk memberikan layanan kepada *client.*



Gambar 2.2. *Load Balancing*

2.5.1 Varnish

Varnish adalah HTTP *reverse proxy* atau disebut juga sebagai HTTP *accelerator* atau web *accelerator*. *Reverse proxy* adalah sebuah server proxy yang diperlihatkan kepada client sebagai sebuah server biasa. Varnish menggunakan Varnish Configuration Language (VCL) untuk penggunaannya dan dalam meggabungkan fitur-fitur yang terdapat di varnish. VCL diterjamhkan ke kode

bahasa pemrograman C lalu dikompilasi dengan kompiler C standard an kemudian secara dinamis dihubungkan langsung ke varnish pada saat run-time.

Varnish bukan hanya sebagai reverse proxy yang mampu menyimpan content (*caching)* . Varnish dapat juga sebagai *load balancer* yaitu menyediakan *load balancing* pada server *backend* yang tersedia*. Load balancing* pada varnish biasa disebut sebagai *backend director* atau hanya *director* (Lyngstol, 2017). Varnish memiliki beberapa *backend* server dengan jumlah yang dapat ditentukan dan menjadi sebuah klaster dengan tujuan untuk proses *load balancing.* Terdapat beberapa algoritma yang digunakan diantaranya ialah *roundrobin, random , fallback* dan *hash.* Dengan metode yang ada, pemilihan *backend* disesuaikan dengan metode yang digunakan. Algoritma yang paling mudah digunakan ialah *round robin* dan *fallback*.

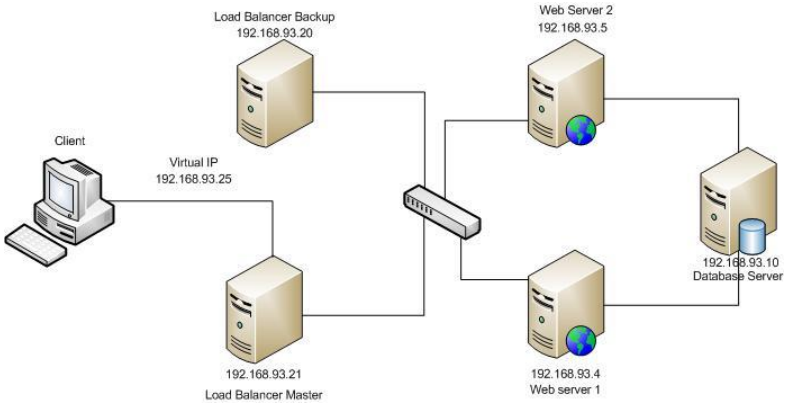
Metode roundrobin pada varnish sama seperti pada umumnya, *roundrobin* hanya memilih *backend* sesuai dengan argument. Tipe *director* ini memilih *backend* pertama lalu permintaan selanjutnya dipilih *backend* kedua dan seterusnya. Jika sudah sampai pada *backend* terakhir, maka akan mengulang ke *backend* pertama. Jika salah satu backend mengalami kegagalan, maka *roundrobin* akan melewatinya. Sedangkan metode *fallback* bekerja dengan cara memilih *backend* pertama secara terus menerus sampai terjadi kegagalan pada *backend* pertama dan akan berlanjut pada seterusnya. Pada *fallback*, *director j*uga bisa menentukan klaster aktif dan pasif. (Velazquez, dkk., 2016).

2.6 *Failover*

Teknologi *failover* merupakan sebuah mekanisme pada sistem untuk menghindari kegagalan. Jika salah satu komponen pada sistem mengalami kegagalan maka akan memindahkan trafik dari komponen utama ke komponen cadangan. Sehingga dengan *failover* maka sistem akan terus berjalan meskipun salah satu komponen mengalami kegagalan. *Failover* dapat diterapkan di berbagai aspek pada suatu sistem.

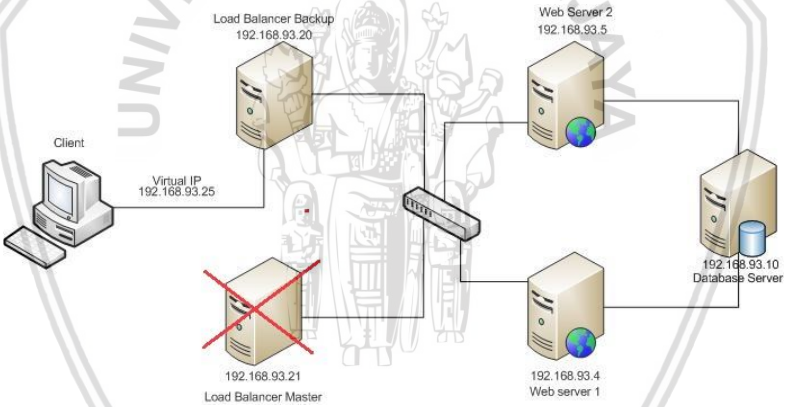
Terdapat dua model dari *failover* yaitu *active - active failover* dan *active – standby failover*. Pada *active – a*ctive *failover*, kedua komponen akan tersedia dan menangani trafik. *Active – active failover* hanya terdia pada mode *multiple context.* Sedangkan pada *active – standby failover* kedua komponen akan tersedia, yaitu komponen utama dan komponen cadangan. Komponen utama akan menangani trafik terlebih dahulu, jika komponen mengalami kegagalan maka trafik akan berpindah secara otomatis ke komponen cadangan (Cisco, 2017).

Pada Gambar 2.3 menjelaskan sistem sebelum terjadi *failover. Client* melakukan *request* dan akan diterima oleh *load balancer 1* yang bertugas sebagai *load balancer master.* Selanjutnya *load balancer master* meneruskan *request* kepada beberapa *web server* yang terhubung. *Load balancer master* akan melakukan tugasnya sampai *load balancer master* mengalami kegagalan atau tidak dapat menjalankan tugasnya.



Gambar 2.3. Sebelum terjadi *Failover*

Pada Gambar 2.4 menjelaskan setelah terjadi *failover.* Setelah terjadi *failover* maka *request* dari *client* akan dialihkan ke *load balancer backup. Load balancer backup* akan membagi beban kerja kepada *web server* yang terhubung untuk memberikan layanan kepada *client.* Dengan penggunaan *failover* pada sistem maka sistem akan tetap berjalan meski *load balancer master* mati atau tidak dapat diakses karena *load balancer backup* akan menggantikan tugas dari *load balancer master.*



Gambar 2.4 Setelah terjadi failover

2.6.1 Keepalived

Keepalived merupakan suatu *software routing.* Tujuan dari keepalived ialah menyediakan fasilitas yang sederhana untuk ketersediaan tinggi pada sistem. Keepalived menggunakan protokol VRRP sehingga akan dipasah sebuah alamt IP yang akan digunakan sebagai gateway dari suatu jaringan lokal. VRRP merupakan dasar dari *failover* (Ha, et al., 2015).

Keepalived menerapkan seperangkat *healthchecker* untuk secara dinamis dan adaptif menjaga dan mengelola beban seimbang. Ketersediaan tinggi dicapai oleh *Virtual Redundancy Routing Protocol* (VRRP). Setiap kerangka Keepalived bisa Digunakan secara independen atau bersama-sama untuk menyediakan infrastruktur yang tangguh. (Cassen, 2017).

2.7 Distributed File System

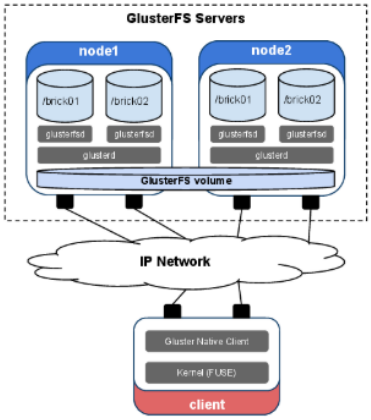
Distributed File System (DFS) merupakan implementasi distribusi pada model *time-sharing* pada suatu file sistem dimana beberapa pengguna dapat membagii file dan melakukan penyimpanan sumberdaya. DFS menangani penyebaran perangkat penyimpanan. Pada DFS dapat menyimpan data pada berbagai node, hal itu disebut dengan replikasi. Replikasi ddigunakan untuk mencapai kinerja sistem yang lebih atau mencapai keandalan pada sistem. data pada DFS lebih aman dari kegagalan node. Jika satu atau lebih node mengalai kegagalan, node lainnya akan melayanani semua layanan. Hal ini diketahui sebagai *availability* dan *reliability. Availability* berarti sistem dapat melayani permintaan dari klien pada saat klien terkoneksi dengan sistem. sedangkan *reliability* berarti sistem selalu tersedia ketika klien terkoneksi (Bzoch, 2012).

2.7.1 GlusterFS

GlusterFS merupakan aplikasi *opensourse* yang bermanfaat untuk ruang penyimpanan untuk manajemen sistem berkas terdistribusi, aplikasi ini memiliki berbagai karakteristik seperti *scalable, reliability* dan *fleksibilitas* ekspansi disk. GlusterFS mempermudah gruntuk penyimpanan terdistribusi. Menggunakan GlusterFS mampu memudahkan dalam pengembangan, *update* dan *debug.*

GlusterFS adalah sistem file terdistribusi open source tanpa server metadata. Sumber daya penyimpanan didistribusikan pada komputer yang berbeda bersama melalui interkoneksi TCP / IP atau InfiniBand RDMA, dan mengelola data menggunakan namespace global tunggal. Pengguna dapat mengakses volume Gluster melalui Gluster Native Client di klien GNU / Linux. Untuk sistem operasi lain yang tidak bisa menjalankan Gluster Native Client, seperti Windows, pengguna juga bisa menggunakan protokol NFS / CIFS untuk mengakses data (Huang, et al., 2015).

Pada Gambar 2.5 menjelaskan mengenai GlusterFS. Terdapat dua buah GlusterFS server yang terhubung. GlusterFS volume merupakan kumpulan dari server GlusterFS yang saling terhubung. Pada setia server GlusterFS terdapat *brick* atau direktori yang merupakan penyimpanan dari *file* diantara server GlusterFS. Sehingga untuk berbagi *file* antar server dilakukan pada *brick.*



Gambar 2.5. Arsitektur GlusterFS

BAB 3 METODOLOGI

Metodologi penelitian yang dilakukan oleh penulis, secara umum ditunjukkan pada gambar 3.1 dibawah ini.

Gambar 3.1. Metodologi Penelitian

3.1 Perumusan Masalah

Perumusan masalah menjelaskan mengenai permasalahan yang menjadi dasar dari penelitian. Penelitian dilakukan untuk menyelesaikan dan menjawab permasalahan yang ada. Perumusan masalah memudahkan dalam melakukan penelitian dan sesuai dengan tujuan yang dicapai yang dilakukan pada penelitian.

3.2 Studi Literatur

Studi literatur dilakukan bertujuan untuk mempelajari serta memahami konsep-konsep sistem agar ketika dilakukan perancangan tidak terlalu mengalami kendala. Jenis literatur yang digunakan adalah artikel jurnal. Adapun yang perlu menjadi bahan studi literatur pada penelitian ini meliputi :

a. *High Availability*

b. Klaster Server Web
c. LAMP Server
d. *Load Balancing*
e. Varnish
f. *Failover*
g. Keepalived
h. Distributed File System
i. GlusterFS

3.3 Analisa Kebutuhan

Pada tahap ini merupakan tahap menganalisa kebutuhan sistem yang akan digunakan. Analisis Kebutuhan sistem diperlukan agar dapat mengetahui hal – hal yang diperlukan dalam pengembangan sistem untuk menghindari penggunaan sumberdaya yang tidak perlu, analisis kebutuhan juga digunakan untuk acuan dalam merancang sistem, sehingga perancangan nantinya dapat terbentuk secara sistematis dan terarah. Dalam melakukan analisis kebutuhan salah satu cara yang dapat digunakan adalah melalui kepustakaan yang telah ada, kepustakaan berlandaskan pada penelitian – penelitian sejenis yang telah dilakukan untuk mengetahui perangkat apa saja yang dapat digunakan dalam pengembangan sistem yang akan dibuat selanjutnya.

3.4 Perancangan

Perancangan sistem dilakukan setelah kebutuhan telah terpenuhi dan sesuai dengan sistem pada penelitian. Perancangan dilakukan agar mengetahui bagaimana struktur dari sistem yang dilaksanakan. Selain itu, menentukan kebutuhan yang menjadi pendukung dari penelitian. Kebutuhan tersebut ialah kebutuhan fungsional dan kebutuhan non fungsional.

3.5 Implementasi

Implementasi sistem dilaksanakan berdasarkan perancangan yang telah dibuat sebelumnya. Untuk melakukan implementasi sistem, ada beberapa tahapan yaitu menentukan jumlah web *server* yang akan digunakan dalam percobaan beserta keperluan dari web *server* itu sendiri. Server lainnya yang diperlukan ialah dua *load balancer* server dan satu database server.

3.6 Pengujian dan Analisa Hasil

Pengujian sistem dilakukan untuk mengetahui keberhasilan dan kekurangan yang dimiliki oleh sistem yang telah dibuat. Tingkat availabilitias dari sistem serta *load balancer* dalam membagi tugas. Terdapat beberapa pengujian dilakukan diantaranya ialah pengujian *downtime,* pengujian *throughput,* pengujian *CPU usagae* dan *memory usage.*

3.7 Kesimpulan dan Saran

Pengambilan kesimpulan dan saran merupakan tahapan akhir dari proses penelitian, yang nantinya dapat disimpulkan mengenai bagaimana sistem yang telah dibuat, kesimpulan dapat diambil dari kelebihan serta kekurangan sistem yang telah dibuat, dan kesesuaian sistem antara teori dan praktik, yang menjawab rumusan masalah yang telah dirumuskan sebelumnya. Saran dimaksud untuk memberikan masukan terhadap kekurangan – kekurangan yang ada di dalam sistem yang telah dibuat, yang menjadi pertimbangan untuk penelitian selanjutnya.

BAB 4 PERANCANGAN

4.1 Analisis Kebutuhan

Analisa kebutuhan diperlukan untuk melakukan perancangan sistem. Analisa kebutuhan menjelaskan komponen-komponen yang dibutuhkan secara fungsional dan non fungsional. Analisa kebutuhan dapat mempermudah dalam proses perancangan sistem. Kebutuhan digunakan sesuai dengan fungsinya dalam melakukan perancangan..

4.1.1 Kebutuhan Fungsional

Kebutuhan fungsional berisikan proses-proses yang mampu dilakukan oleh sistem. terdapat beberapa kebutuhan fungsional untuk memastikan bahwa sistem telah berjalan dengan baik beserta komponen yang terdapat pada sistem.

1. Sistem mampu memberikan layanan meski salah satu *load balancer* mengalami kegagalan.
2. Load balancer mampu membagikan beban tugas kepada web server menggunakan algoritma *round robin* dan *fallback*.
3. *Failover* mampu memidahkan trafik dari *load balancer* 1 ke *load balancer 2* jika *load balancer 1* mengalami kegagalan.
4. Web server mampu menerima tugas yang diberikan oleh *load balancer*.
5. Jika salah satu web server mati, maka server lainnya tetap dapat memberikan layanan.
6. *Database server* mampu menyimpan data dan memberikan data yang diminta oleh *client.*
7. Penggunaan GlusterFS untuk sinkornisasi file antara web server 1 dan web server2.

4.1.2 Kebutuhan Non Fungsional

Kebutuhan Hardware :

- Laptop Lenovo Intel(R) Core(TM) i5
- Ram 8.00GB
- 64-*bit Operating System*

Kebutuhan Software :

- Virtual Box
- Ubuntu *Server* 16.04
- Linux Apache Mysql PHP (LAMP)
- Varnish
- Keepalived
- GlusterFS
- HTTPerf
- Mysql Server

Kebutuhan Server

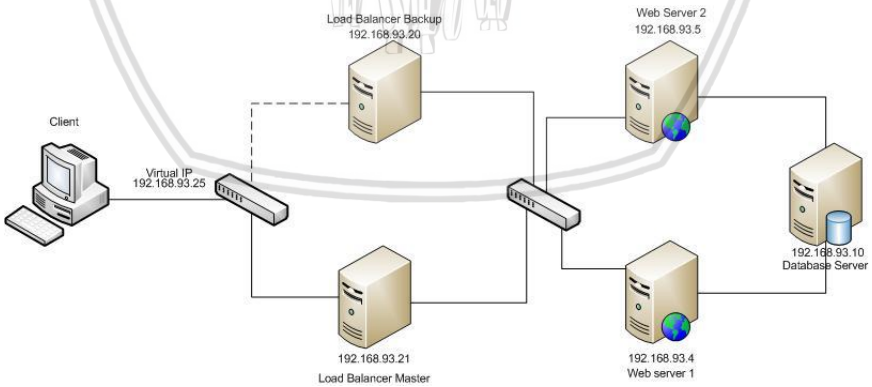
- Linux OS

- RAM 1GB
- Disk 20GB

4.2 Perancangan Topologi

Gambar 4 merupakan perancangan topologi sistem. Topologi perancangan sistem secara keseluruhan terdiri dari *client, load balancer 1, load balancer 2, web server 1, web server 2* serta *database server.* Varnish akan berperan sebagai *load balancer,* Apache sebagai *web server* serta MySql sebagai *database server.* Selain itu terdapat juga keepalived yang berada di *load balancer 1* serta *load balancer 2* dan bertugas sebagai *failover*. Sehingga jika *load balancer* utama atau *load balancer* 1 mengalami kegagalan atau mati, *keepalived* akan memindahkan *traffic* dari *load balancer 1* ke *load balancer* cadangan atau *load balancer 2.*

Pertama yang dilakukan ialah *client* melakukan *request* http ke alamat *virtual IP* keepalived yang akan diterima oleh server *load balancer* 1. Selanjutnya *load balancer 1* meneruskan tugas yang diberikan oleh *client* kepada web server untuk meminta *request* yang di inginkan oleh client. Setelah itu *request* dikembalikan kepada *client* sesuai yang diinginkan. Jika pada saat *client* mengirimkan *request* dan *load balancer 1* dalam keadaan tidak tersedia, maka keepAlived bertindak sebagai *failover* yang meneruskan *request* yang diminta oleh *client* kepada *load balancer 2.* Sehingga *load balancer 2* akan menggantikan tugas *load balancer 1* untuk meneruskan *request* ke web server. Pada saat *load balancer 1* berjalan atau aktif, *load balancer 2* juga aktif namun berfungsi untuk memonitoring saja sampai *load balancer 1* tidak aktif atau mati. Pada web server terdapat GlusterFS sebagai penyimpanan file terdistribusi antara web server 1 dan web server 2. Serta terdapat database server yang digunakan sebagai penyimpanan untuk kedua buah web server.



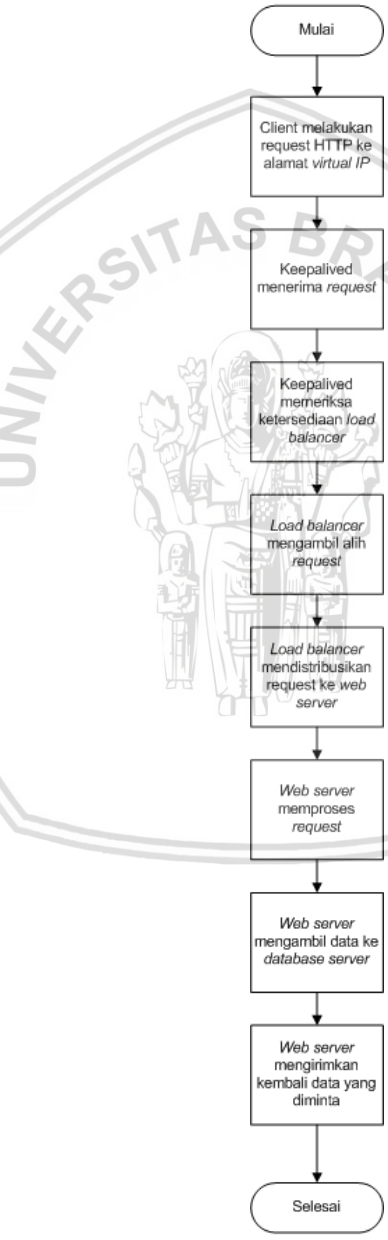
Gambar 4.1 Perancangan Topologi

4.3 Alur Kerja

Alur kerja sistem menjelaskan secara bertahap bagaimana sistem berjalan serta proses yang terjadi. Terdapat *client, load balancer, web server* serta beberapa komponen lain yang tersedia pada sistem agar sistem mampu berjalan dengan baik dan sesuai tujuan.

4.3.1 Alur Kerja Sistem

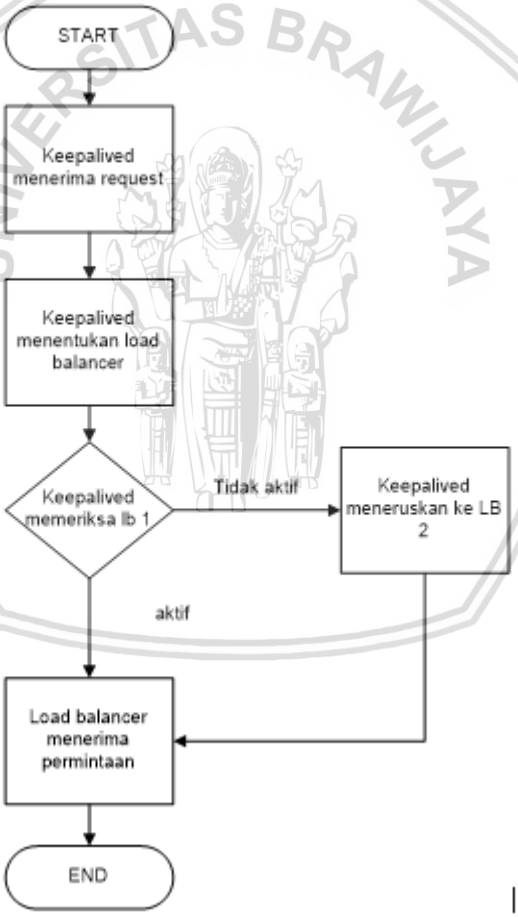
Pada gambar 4.2 menjelaskan secara bertahap bagaimana sistem berjalan dimulai dengan *client* mengakses web. Client melakukan *request HTTP* ke alamat *virtual IP.* Keepalived menerima *request* kemudian melakukan pemeriksaan ketersediaan *load* balancer lalu selanjutnya *load balancer* mengambil alih *request* tersebut untuk di distribusikan ke *web server.* W*eb server* memproses *request* dan mengambil data ke *database server* dan mengirimkannya kembali ke *client* sebagai respon.



Gambar 4.2 Alur Kerja Sistem

4.3.2 Alur Kerja *Failover* Keepalived

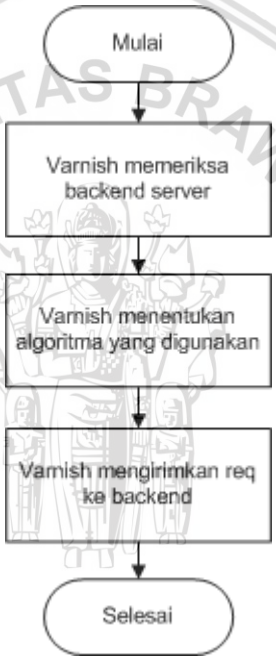
Gambar 4.3 merupakan alur kerja dari keepalived. Terdapat virtual IP yang di miliki oleh keepalived, sehingga *request* terlebih dahulu sampai ke keepalived sebelum diteruskan kepada *load balancer.* Keepalived menerima *request* yang dikirimkan oleh *client* dikarenakan *client* melakukan *request* dengan memasukkan *virtual* IP yang telah ditentukan. Keepalived selanjutnya menentukan *load balancer* yang aktif untuk menerima *request.*Keepalived memeriksa *load balancer* 1, jika *load balancer* 1 mati maka *request* diteruskan ke *load balancer 2* yang merupakan *backup.* Setelah keepalived menentukan *load balancer* yang mampu menerima permntaan, maka permintaan tersebut diteruskan kepada *load balancer* yang aktif. Pada saat *load balancer 1* berjalan, *load balancer 2* hanya bertugas dalam memonitoring sampai *load balancer 1* mengalami kegagalan selanjutnya dialihkan ke *load balancer 2.*



Gambar 4.3 Alur Kerja *Failover* Keepalived

4.3.3 Alur Kerja Load Balancing Varnish

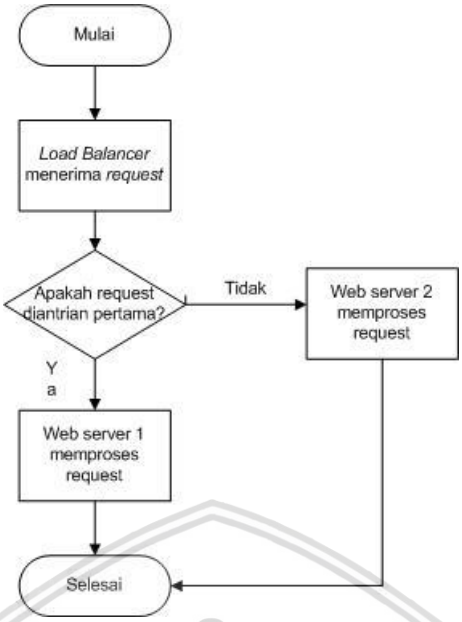
Gambar 4.4 merupakan alur kerja dari *load balancing* varnish. Terdapat dua web server yang menjadi *backend server* yaitu web server 1 dan web server 2. Hal pertama yang dilakukan ialah Varnish selalu melakukan *health check* pada *backend server* yang ditentukannya yaitu web server 1 dan web server 2 lalu setelah itu varnish menentukan algoritma yang digunakan. Algoritma yang digunakan pada pengujian ialah *round robin* dan *fallback.* Untuk algoritma *round robin* maka varnish mendistribusikan *request* kepada web server 1 dan web server 2 secara bergantian. Sedangkan untuk algoritma *fallback* ialah varnish mendistribusikan ke backend server 1 terlebih dahulu sampai backend server 1 mengalami kegagalan baru dialihkan ke backend server 2. Lalu varnish melanjutkan proses *request* menuju *backend server.*



Gambar 4.4 Alur Kerja Load Balancing Varnish

4.3.3.1 Alur Kerja Algoritma *Round Robin*

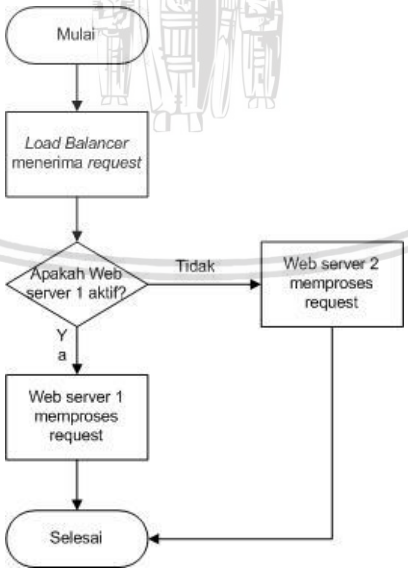
Pada Gambar 4.5 merupakan alur kerja dari algoritma *round robin.* Pada algoritma *round robin,* ketika *load balancer* menerima *request* pada antrian pertama maka *request* tersebut diteruskan kepada *web server 1* dan *web server 1* memproses *request* yang diteruskan tersebut. Pada *request* selanjutnya *load balancer* meneruskan *request* tersebut kepada *web server 2* dan diproses oleh *web server 2. Request* yang diterima akan diproses oleh *web server* 1 dan *web server 2* secara bergantian.



Gambar 4.5 Alur Kerja Algoritma *Round Robin*

4.3.3.2 Alur Kerja Algoritma *Fallback*

Pada Gambar 4.6 merupakan alur kerja dari algoritma *fallback.* Ketika *loaa balancer* menerima *request* maka *web server 1* akan memproses *request* tersebut sampai *web server 1* tidak bisa melayani permintaan atau tidak tersedia. Jika *web server 1* tidak tersedia atau tidak bisa melayani *request* maka *web server 2* menerima *request* dan menggantikan tugas dari *web server 1* untuk selanjutnya diproses

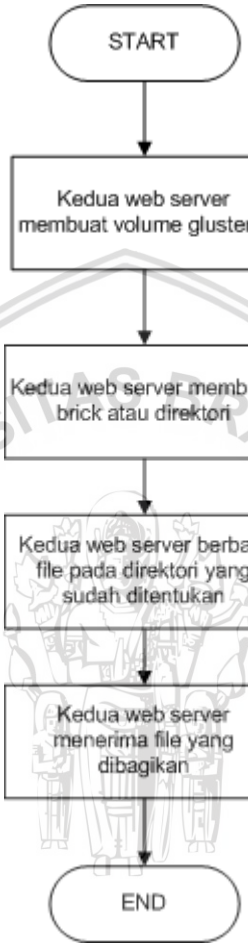


Gambar 4.6 Alur Kerja Algoritma *Fallback*

4.3.4 Alur Kerja Distributed File System GlusterFS

GlusterFS diterapkan pada kedua web server. Sehingga kedua web server dapat berbagi file data. Hal itu mempermudah dalam melakukan distribusi data.

Setelah memastikan bahwa kedua web server sudah dapat saling berbagi hal yang dilakukan ialah dengan membuat *volume* glusterfs pada kedua web server lalu setelah itu membuat direktori atau *brick* yang menjadi tempat untuk saling berbagi. Setelah itu kedua web server dapat berbagi file pada direktori yang sudah ditentukan. Web server 1 pada direktori /var/www/data-server sedangkan Web server 2 pada direktori /var/www/data-client.



Gambar 4.7 Alur Kerja GlusterFS

4.4 Pengujian Sistem

Terdapat beberapa pengujian yang dilakukan untuk menguji sistem. pengujian yang dilakukan ialah pengujian fungsionalitas, pengujian *downtime* dan *packet loss*, pengujian *throughput* dan pengujian *CPU Usage* dan *memory usage*. Pengujian dilakukan selama 30 detik dan setiap detiknya dilakukan jumlah koneksi yang berbeda – beda dan semakin meningkat. Skenario pengujian sistem terdapat pada table 2. Pengujian dilakukan dengan menggunakan algoritma *round robin* dan algoritma *fallback.*

Table 4.1 Skenario Pengujian Sistem

No	Jumlah Koneksi (/sec)	Skenario Pengujian
1	20	Pada saat sistem berjalan, *load balancer 1* sengaja dimatikan.
2	40	
3	60	
4	80	
5	100	
1	20	Pada saat sistem berjalan, web server 1 sengaja dimatikan.
2	40	
3	60	
4	80	
5	100	
1	20	Pada saat sistem berjalan, *load balancer 1* dan *web server 1* dimatikan.
2	40	
3	60	
4	80	
5	100	

4.4.1 Pengujian Fungsionalitas

Pengujian fungsional dilakukan untuk mengetahui jika sistem telah berjalan dengan baik dan benar sesuai dengan kebutuhan fungsional yang telah ditentukan pada penelitian.

Table 4.2 Pengujian Kebutuhan Fungsional

NO	Kebutuhan Fungsional	Hasil
1	Sistem mampu memberikan layanan meski salah satu *load balancer* mengalami kegagalan.	
2	Load balancer mampu membagikan beban tugas kepada web server menggunakan algoritma *round robin* dan *fallback*.	
3	*Failover* mampu memidahkan trafik dari *load balancer* 1 ke *load balancer 2* jika *load balancer 1* mengalami kegagalan.	

4	Web server mampu menerima tugas yang diberikan oleh *load balancer.*	
5	Jika salah satu web server mati, maka server lainnya tetap dapat memberikan layanan.	
6	Database server mampu menyimpan data dan memberikan data yang diminta oleh *client.*	
7	Penggunaan GlusterFS untuk sinkornisasi file antara web server 1 dan web server2.	

4.4.2 Pengujian Downtime

Pengujian *downtime* ialah untuk mengetahui waktu ketika terjadinya suatu kerusakan atau gagalan pada salah satu komponen sistem tetapi secara keseluruhan sistem masih mampu beroperasi karena terdapat komponen lain yang menggantikan fungsi komponen yang mengalami kegagalan tersebut.. Sehingga pengujian *downtime* sangat berpengaruh pada performa sistem ketika terjadi kegagalan. Pengujian *downtime* dan *packet loss* dilakukan sesuai dengan skenario pengujian yang telah dibuat.

Table 4.3 Tabel Pengujian Downtime

NO	Jumlah koneksi (/sec)	Nilai Downtime (/ms)	
		Round robin	Fallback
1	20		
2	40		
3	60		
4	80		
5	100		

4.4.3 Pengujian Throughput

Pengujian *throughput* ialah untuk mengetahui nilai rata-rata pada saat pengiriman data yang sukses melalui link komunikasi dan data tersebut dikirim lelaui *link physical, logical* atau melewati *network node* tertentu. Pegujian

dilakukan berdasarkan skenario pengujian yang telah dibuat dan dengan jumlah koneksi yang berbeda dan semakin meningkat.

Table 4.4 Pengujian Throughput

NO	Jumlah koneksi (/sec)	Nilai Throughput (KB/sec)	
		Round robin	Fallback
1	20		
2	40		
3	60		
4	80		
5	100		

4.4.4 Pengujian *CPU usagae* dan *Memory* Usage

CPU usagae adalah waktu penggunaan prosesor yang dillakukan oleh program dalam beroperasi. *Memory usage* ialah untuk mengetahui jumlah penggunaan *memory* saat sistem berjalan atau beroperasi. Skenario pengujian dillakukan sesuai dengan skenario yang telah dibuat. Pengujian *CPU usagae* dan *memory usage* pada penelitian ini menggunakan *tool* top yang berfungsi untuk memonitoring.

Table 4.5 Tabel Pengujian *CPU usagae* dan *Memory* Usage

NO	Jumlah koneksi (/sec)	*CPU usagae* (%)		*Memory* Usage (KB)	
		Round robin	Fallback	*Round robin*	Fallback
1	20				
2	40				
3	60				
4	80				
5	100				

BAB 5 IMPLEMENTASI

5.1 Implementasi Ubuntu Server

Ubuntu server merupakan sistem operasi yang digunakan pada seluruh mesin virtual yang digunakan untuk penelitian. Server – server yang dibangun ialah *load balancer 1, load balancer 2, web server 1, web server 2* dan *database server.* Pada *web server 1* dan *web server 2* dilakukan instalasi LAMP server sehingga sudah terdapat apache2, mysql dan php. Hal pertama yang dilakukan ialah melakukan konfigurasi untuk merubah alamat IP pada seluruh server menjadi alamat IP statik. Perubahan dilakukan dengan menuliskan perintah seperti pada table 5.1

Table 5.1. Perubahan *Interface*

```
1  $ sudo nano /etc/network/interfaces
```

Selanjutnya melakukan konfigurasi untuk melakukan perubahan alamat IP menjadi alamat IP statik. Alamat IP statik untuk *load balancer 1* ialah 192.168.93.20, *load balancer 2* ialah 192.168.93.21, *web server 1* ialah 192.168.93.4, *web server 2* ialah 192.168.93.5 dan untuk *database* server ialah 192.168.93.10. Setelah dilakukan perubahan, maka server akan dikenal dengan alamat IP static yang sudah ditetapkan.

Table 5.2 Konfigurasi Alamat IP statik Load Balancer 1

```
1  # The primary network interface
2  Auto enp0s3
3  Iface enp0s3 inet dhcp
4  Auto enp0s8
5  Iface enp0s8 inet static
6           Address 192.168.93.20
7           Netmask 255.255.255.0
```

Table 5.3 Konfigurasi Alamat IP statik Load Balancer 2

```
1  # The primary network interface
2  Auto enp0s3
3  Iface enp0s3 inet dhcp
4  Auto enp0s8
5  Iface enp0s8 inet static
6           Address 192.168.93.21
7           Netmask 255.255.255.0
```

Table 5.4 Konfigurasi Alamat IP statik Web Server 1

```
1  # The primary network interface
2  Auto enp0s3
3  Iface enp0s3 inet dhcp
4  Auto enp0s8
5  Iface enp0s8 inet static
6           Address 192.168.93.4
7           Netmask 255.255.255.0
```

Table 5.5 Tabel Konfigurasi Alamat IP statik Web Server 2

```
1  # The primary network interface
2  Auto enp0s3
3  Iface enp0s3 inet dhcp
4  Auto enp0s8
5  Iface enp0s8 inet static
6           Address 192.168.93.5
7           Netmask 255.255.255.0
```

Table 5.6 Konfigurasi Alamat IP statik Database Server

```
1  # The primary network interface
2  Auto enp0s3
3  Iface enp0s3 inet dhcp
4  Auto enp0s8
5  Iface enp0s8 inet static
6           Address 192.168.93.10
7           Netmask 255.255.255.0
```

5.2 Implementasi Web Server

Penelitian menggunakan wordpress sebagai konten dari web. Pada web server 1 dan web server 2, sudah terdapat LAMP server yang terdiri dari apache2, mysql dan php. Sehingga hal yang dilakukan selanjutnya ialah melakukan instalasi tambahan pada php 7.0 untuk kebutuhan wordpress pada seluruh web server dengan melakukan perintah seperti pada tabel 5.7

Table 5.7 Perintah Instalasi PHP 7.0 *modules*

```
1  $ sudo apt-get install php7.0 php7.0-mysql libapache2-
   mod-php7.0 php7.0-cgi php7.0-gd
```

Setelah itu hal yang dilakukan ialah melakukan unduh wordpress dengan melakukan perintah seperti pada table 5.8

Table 5.8 Perintah Untuk Unduh File Wordpress

```
1  $ wget -c http://wordpress.org/latest.tar.gz
```

Setelah proses unduh wordpress telah berhasil dilakukan, selanjutnya ialah melakukan unpack folder tersebut terlebih dahulu dengan melakukan perintah seperti pada table 5.9

Table 5.9 Perintah *Unpack File*

```
1  $ tar –xzvf latest.tar.gz
```

Folder wordpress yang telah di *unpack,* selanjutnya dipindahkan ke folder /var/www/html dengan melakukan perintah seperti pada table 16.

Table 5.10 Perintah Memindahkan Folder

```
1  $sudo cp –r wordpress/ /var/www/html
```

Setelah berhasil dipindahkan, proses selanjutnya ialah mendaftarkan wordpress sebagai layanan pada apache2 web server dengan melakukan konfigurasi pada apache2. Perintah yang dilakukan ialah seperti pada tabel 5.11 Dan juga mengganti document root untuk merujuk ke direktori tempat penyimpanan wordpress.

Table 5.11 Merujuk ke direktori

```
1  $DocumentRoot /var/www/html/wordpress
```

Selanjutnya ialah melakukan *restart* pada apache2 dengan melakukan perintah seperti pada tabel.

Table 5.12. Melakukan *restart* apache

```
1  $sudo service apache2 restart
```

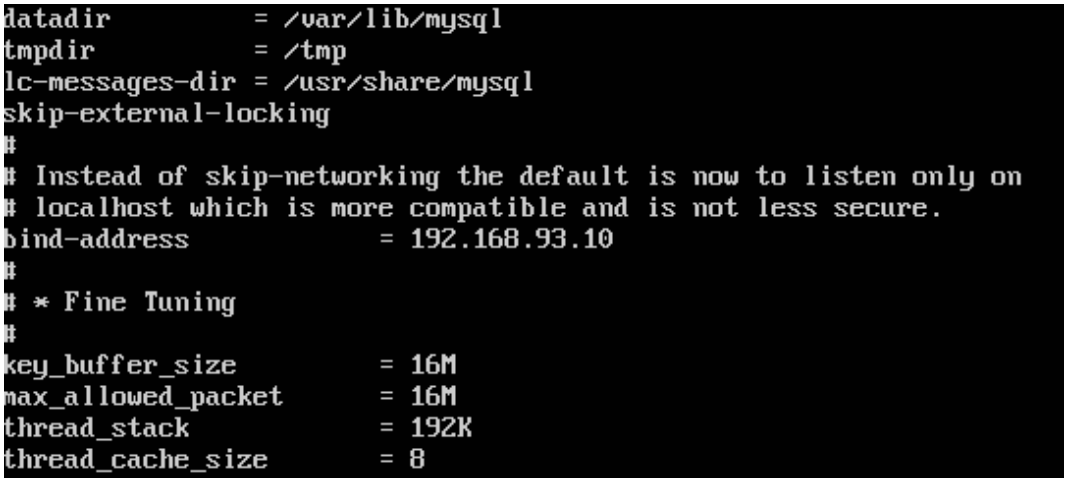
5.3 Implementasi Database Server

Implementasi pada database server dengan melakukan instalasi mysql-server terlebih dahulu. Hal ini menunjukan bahwa database server sebagai server sedangkan web server 1 dan web server 2 bertugas sebagai *client.* Instalasi mysql-server dilakukan dengan melakukan perintah seperti pada tabel 5.13

Table 5.13. Instalasi Mysql-server

```
1  $sudo apt-get install mysql-server
```

Setelah mysql-server terlah berhasil terinstalasi, selanjutnya ialah mengganti alamat IP bind-address pada file mysql.cnf menjadi alamat IP dari *database server* seperti pada gambar 5.1



Gambar 5.1 Konfigurasi bind-address

Selanjutnya ialah membuat database sebagai kebutuhan wordpress. Pertama ialah masuk ke mysql terlebih dahulu dengan *root* sebagai user dan memasukan *password* yang telah ditentukan sebelumnya.

Table 5.14. Masuk ke root mysql

```
1  $mysql –u root –p
```

Setelah masuk ke mysql, selanjutnya melakukan pembuatan database untuk keperluan wordpress seperti pada tabel 5.15

Table 5.15 Membuat database wordpress

```
 1  mysql> create database wordpress;
 2
 3  mysql> create user 'wpuser'@'localhost' identified
 4  by 'pass';
 5
 6  mysql> grant all privileges on wordpress.* to
 7  'wpuser'@'localhost';
 8
 9  mysql>    create    user    'wpuser'@'192.168.93.4'
10  identified by 'pass';
11
12  mysql> grant select, delete, insert, update on
13  wordpress.* to 'wpuser'@'192.168.93.4';
14
15  mysql> grant all privileges on wordpress.* to
16  'wpuser'@'192.168.93.4';
17
18  mysql>    create    user    'wpuser'@'192.168.93.5'
19  identified by 'pass';
20
21
```

```
22 mysql> grant select, delete, insert, update on
23 wordpress.* to ‘wpuser’@’192.168.93.5’;
24
25 mysql> grant all privileges on wordpress.* to
26 ‘wpuser’@’192.168.93.5’;
27
   mysql> Flush privileges;
```

Baris ke 1 menunjukan bahwa membuat database dengan nama wordpress. Baris ke 3 sampai baris ke 6 ialah membuat user ‘wpuser@localhost’ dengan kata sandi pass dan memperbolehkan untuk user tersebut mengendalikan dan mengakses database wordpress. Baris ke 9 sampai dengan baris ke 25 ialah membuat user untuk mysql client yaitu web server 1 dan web server 2 agar dapat mengakses dan mengendalikan database wordpress.

5.3.1 Konfigurasi Wordpress

Konfigurasi wordpress dilakukan dengan menyalin file wp-config-sample.php pada folder wordpress dengan wp-config.php. setelah itu melakukan konfigurasi pada file wp-config.php seperti pada table 5.16

Table 5.16 Konfigurasi wp-config.php

```
   Wp-config.php
 1 define('WP_HOME', 'http://192.168.93.4');
 2 define('WP_SITEURL', 'http://192.168.93.4');
 3
 4 define('DB_NAME', 'wordpress');
 5 /** MySQL database username */
 6 define('DB_USER', 'wpuser');
 7 /** MySQL database password */
 8 define('DB_PASSWORD', 'pass');
 9 /** MySQL hostname */
10 define('DB_HOST', '192.168.93.10');
11 /** Database Charset to use in creating database
12 tables. */
13 define('DB_CHARSET', 'utf8');
14 /** The Database Collate type. Don't change this if
15 in doubt. */
16 define('DB_COLLATE', '');
```

Pada file wp-config.php, perlu dilakukannya perubahan yaitu pada DB_NAME yang diisikan nama dari database yang digunakan, DB_USER diisikan dengan nama user pada database dan DB_PASSWORD ialah *password* yang digunakan. DB_HOST diisian dengan alamat IP dari database server. Sedangkann untuk WP_HOME dan WP_SITEURL di isikan alamat IP dari web server itu sendiri. Lalu selanjutnya mengakses alamat IP web server 1 untuk membuat wordpress.

Gambar 5.2 Tampilan Wordpress

5.4 Implementasi GlusterFS

GlusterFs digunakan untu melakukan pembagian data dari web server 1 ke web server 2. File wordpress dari web server 1 akan dipindah ke web server 2 menggunakan glusterFS. Instalasi yang dilakukan ialah dengan melakukan instalasi glusterfs server pada web server 1 dengan perintah seperti table 19 pada web server 1.`

Table 5.17. Instalasi GluterFS

1	`$sudo apt-get install glusterfs-server`

Setelah melakukan instalasi glusterfs-server, selanjutnya ialah membuat volume. Dengan melakukan perintah pada table 5.18.

Table 5.18. Membuat Volume

1	`$sudo gluster volume create gfs 192.168.93.4:/var/www/data-server force`

Setelah itu memulai volume glusterfs dengan melakukan perintah pada table 5.19.

Table 5.19. Memulai volume glusterfs

1	`$sudo gluster volume start gfs`

Lalu memeriksa volume info dengan melakukan perintah pada table 5.20.

Table 5.20. Memeriksa Volume Info Glusterfs

1	$sudo gluster volume info

```
webserver1@webserver1:~$ sudo gluster volume info

Volume Name: gfs
Type: Distribute
Volume ID: b884cae7-1270-448e-ab0b-ba95f9ba3d1d
Status: Started
Number of Bricks: 1
Transport-type: tcp
Bricks:
Brick1: 192.168.93.4:/var/www/data-server
Options Reconfigured:
performance.readdir-ahead: on
auth.allow: 192.168.93.5
```

Gambar 5.3. GlusterFS volume info

Lalu melakukan set volume glusterfs untuk memperbolehkan web server 2 dapat saling mengakses volume dengan melakukan perintah pada table 5.21

Table 5.21. Perintah untuk Web Server 2 Mengakses Volume

1	$sudo gluster volume set gfs auth.allow 192.168.93.5

Selanjutnya ialah melakukan konfigurasi glusterfs pada web server 2. Hal yang dilakukan pertama ialah melakukan instalasi glusterfs-client pada web server 2 dengan melakukan perintah seperti pada table 5.22.

```
[2018-06-07 09:43:33.044894]  : volume create gfs 192.168.93.4:/var/www/data-server force : SUCCESS
[2018-06-07 09:43:50.848738]  : volume start gfs : SUCCESS
[2018-06-07 09:44:23.352456]  : volume set gfs auth.allow 192.168.93.5 : SUCCESS
[2018-06-07 09:52:14.400258]  : volume set gfs auth.allow 192.168.93.5 : SUCCESS
[2018-06-07 09:56:39.124782]  : volume set gfs auth.allow 192.168.93.5 : SUCCESS
```

Gambar 5.4 GLusterFS sukses

Table 5.22 Instalasi glusterfs client

1	$sudo apt-get install glusterfs-client

Lalu selanjutnya ialah membuat direktori dengan nama data-client pada /var/ww/ dengan melakukan perintah seperti pada table 5.23

Table 5.23. Membuat Direktori

1	$sudo mkdir /var/www/data-client

Lalu selanjutnya ialah melakukan mount pada web server 2 agar dapat diakses oleh web server 1 dengan melakukan perintah seperti pada table 5.24

Table 5.24. Mount Pada Web Server 2

1	$sudo mount.glusterfs 192.168.93.4:/gfs /var/www/data-client

Lalu selanjutnya ialah melakukan edit /etc/fstab pada web server 2 dengan melakukan perintah

Table 5.25. Edit fstab

1	$sudo nano /etc/fstab

Dengan menambahkan konfigurasi untuk glusterFS pada web server seperti pada table 5.26

Table 5.26. Konfigurasi fstab

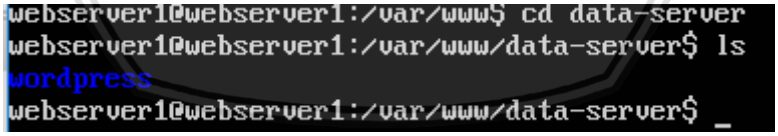
1	192.168.93.5:/gfs /var/www/data-client glusterfs defaults,_netdev 0 0

Lalu selanjutnya ialah melakukan copy file wordpress dari web server 1 ke web server 2 dengan melakukan perintah seperti pada table 5.27

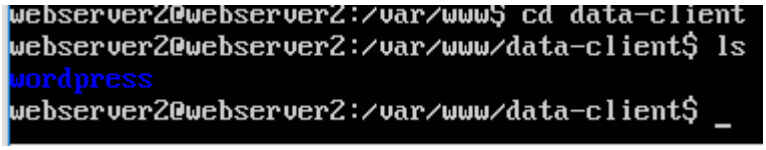
Table 5.27. Melakukan Copy File

1	$sudo cp –r /var/www/wordpress /var/www/data-server

Sehingga file wordpress pada folder data-server web server 1 juga sampai pada folder data-client di web server 2.



Gambar 5.5. Direktori data-server



Gambar 5.6. Direktori data-client

5.5 Implementasi Varnish

Hal yang dilakukan ialah pertama melakukan instalasi varnish pada kedua load balancer server dengan melakukan perintah seperti pada table 5.28.

Table 5.28. Instalasi Varnish

```
1  $sudo apt-get install varnish
```

Selanjutnya ialah dengan melakukan edit *port* pada varnish agar melakukan *listening* ke *port* 80. Dengan membuka file varnish pada direktori /etc/default pada *load balancer* 1 dan *load balancer* 2 dengan melakukan perintah seperti pada table 5.29

Table 5.29. Membuka file varnish.service

```
1  $ sudo nano /etc/default/varnish
```

Lalu mengisikan file tersebut dengan konfigurasi seperti pada table 5.30.

Table 5.30. Konfigurasi Port

```
   /etc/default/varnish
1  DAEMON_OPTS="-a :80 \
2               -t localhost:6082 \
3               -f /etc/varnish/default.vcl \
4               -S /etc/varnish/secret \
5               -s malloc,256m"
```

Lalu setelah melakukan konfigurasi tersebut, hal selanjutnya yang dilakukan ialah melakukan konfigurasi pada file default.vcl pada directori /etc/varnish di *load balancer 1* dan *load balancer 2* dengan melakukan perintah

Table 5.31. Membuka file default.vcl

```
1  $ sudo nano /etc/varnish/default.vcl
```

Dengan melakukan konfigurasi seperti dibawah pada table 5.32.

Table 5.32. Konfigurasi Varnish Algoritma *Round robin*

```
   default.vcl
1  vcl 4.0;
2  import directors;
3  import std;
4
5  probe www_healthy_probe {
6          .url = "/";
7          .timeout = 1s;
8          .interval = 5s;
9          .window = 5;
10         .threshold = 3;
11 }
12
13 backend www1 {
```

```
     default.vcl
14       .host = "192.168.93.4";
15       .port = "80";
16       .connect_timeout = 1s;
17       .first_byte_timeout = 5s;
18       .between_bytes_timeout = 2s;
19       .probe = www_healthy_probe;
20  }
21
22  backend www2 {
23       .host = "192.168.93.5";
24       .port = "80";
25       .connect_timeout = 1s;
26       .first_byte_timeout = 5s;
27       .between_bytes_timeout = 2s;
28       .probe = www_healthy_probe;
29  }
30
31  sub vcl_init {
32            new cluster1 = directors.round_robin();
33            cluster1.add_backend(www1);
34            cluster1.add_backend(www2);
35  }
36  sub vcl_recv {
37            set req.backend_hint = cluster1.backend();
38            if                (req.url                ~
39  "(?i)\.(jpeg|jpg|png|gif|ico|swf|js|css|gz|rar|txt|
40  bzip)$") {
41                      unset req.http.cookie;
42                      return (hash);
43            } else {
44                      return (pass);
45            }
46  }
```

Penjelasan dari *source code* konfigurasi varnish diatas ialah pada baris ke 2 dan 3 merupakan *module* yang dibutuhkan oleh varnish. *Module directors* diperlukan untuk *load balancing.* Sedangkan untuk std sudah menjadi *default* yang diperlukan untuk konfigurasi. Pada baris ke 5 sampai ke 10 menjelaskan mengenai probe yang berguna untuk memeriksa *healthy* dari *backend server.* Terdapat beberapa ketentuan pada probe yaitu url berarti *request* GET yang dikirim oleh varnish. Lalu *timeout* menjelaskan lama waktu timeout. *Interval* berarti varnish akan memeriksa *healthy backend* selama 5 detik. Untuk *window* dan *threshold* mengartika jika 3 dari 5 proses pada *backend* berhasil maka *backend* ditandai sebagai *health backend* namun jika tidak maka akan ditandai sebagai *sick backend.* Untuk baris ke 13 sampai 20 menjelaskan mengenai *backend server* yaitu web

server 1 atau ditandai dengan www 1. Host yang diisikan ialah alamat host dari web server 1. Lalu port berarti akan melakukan *listening* pada *port* 80. Baris ke 25 sampai 27 mengisika nilai *timeout.* Lalu baris ke 28 untuk melakukan *probe.* Baris ke 22 sampai 29 menjelaskan mengenai *backend server* yang kedua yaitu web server 2 atau www 2. Pada baris ke 31 sampai 35 yaitu vcl_init menjelaskan mengenai algoritma yang digunakan untuk proses distribusi dan kemana algoritma tersebut bertuju. Pada baris 36 sampai 46 ialah vcl_recv konfigurasi HTTP *request* dari varnish. Pada baris ke 37 untuk mendefinisikan backend yang dituju. Baris 38 sampai 40 menjelaskan mengenai *request* url yang disimpan. Baris ke 41 merupakan perintah untuk mengapaikan *cookie.* Baris ke 41 perintah untuk mencari *request* pada *cache* sedangkan perintah pada baris ke 44 untuk melanjutkan proses tanpa mencari *request* pada *cache.*

5.6 Implementasi Keepalived

Selanjutnya ialah keepalived sebagai *failover*. Hal yang dilakukan pertama ialah melakukan instalasi keepalived pada kedua *load balancer* dengan melakukan perintah seperti pada table 5.33.

Table 5.33. instalasi keepalived

```
1  $sudo apt-get install keepalived
```

Selanjutnya ialah membuat file keepalived.conf untuk melakukan konfigurasi pada keepalived pada kedua *load balancer.* Untuk *load balancer 1* sebagai *load balancer* cadangan, dilakukan konfigurasi seperti pada table 5.34

Table 5.34. Konfigurasi Keepalived load balancer 1

```
    keepalived.conf
 1  vrrp_script chk_varnish {
 2
 3          script "killall -0 varnish"
 4          interval 2
 5          weight 2
 6  }
 7
 8  vrrp_instance VI_2
 9          interface enp0s8
10          state MASTER
11          virtual_router_id 52
12          priority 101
13          virtual_ipaddress {
14                  192.168.93.25
15          }
16
17          track_script {
18                  chk_varnish
19          }
```

20	}

Table 5.34 Konfigurasi Keepalived load balancer 2

	keepalived.conf
1	vrrp_script chk_varnish {
2	
3	script "killall -0 varnish"
4	interval 2
5	weight 2
6	}
7	
8	vrrp_instance VI_2
9	interface enp0s8
10	state MASTER
11	virtual_router_id 52
12	priority 100
13	virtual_ipaddress {
14	192.168.93.25
15	}
16	
17	track_script {
18	chk_varnish
19	}
20	}

Source code pada keepalived.conf menjelaskan bahwa pada baris ke 1 sampai 6 ialah mendefinisikan varnish atau memeriksa varnish apakah berjalan atau tidak. Pada baris ke 8 ialah menjelaskan konfigurasi yang membuat virtual IP. Baris ke 9 ialah *interface* nama *physical interface* untuk virtual IP yang dibuat. Pada baris ke 10 menjelaskan bahwa server utama sebagai *master* sedangka untuk server cadangan sebagai *backup.* Baris ke 11 mengidentifikasikan nomor untuk *virtual router.* Baris ke 12 ialah nilai *priority* untuk menspesifikasikan urutan *interface* untuk mengambil alih *failover,* semakin besar angka maka semakin tinggi prioritas. Untuk server master harus memiliki nilai priority yang lebih tinggi dibandingkan degan *back up* server. baris ke 13 mengisikan alamat IP yang menjadi alamat virtual IP.

BAB 6 PENGUJIAN

6.1 Pengujian

Pengujian pada sistem dilakukan untuk mengetahui performa serta kehandalan pada sistem. Terdapat berbagai pengujian yang dilakukan oleh peneliti diantaranya ialah pengujian *downtime,* pengujian *throughput,* pengujian *CPU usagae* dan *memory usage* serta pengujian fungsionalitas berdasarkan kebutuhan fungsional pada sistem.

6.1.1 Pengujian Downtime

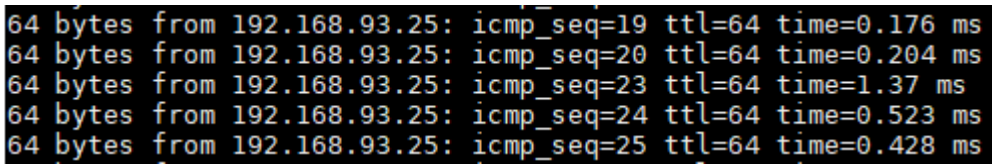
Pengujian *downtime* dilakukan untuk mengetahui berapa lama waktu yang dibutuhkan ketika terjadi kegagalan pada salah satu komponen sistem namun sistem masih beroperasi karena adanya komponen pendukung lain yang menggantikan peran komponen yang mati pada sistem tersebut.

Pengujian dilakukan untuk mengetahui berapa lama waktu yang dibutuhkan oleh *failover* untuk memindahkan trafik dari *load balancer 1* ke *load balancer 2.* Pengujian *downtime* dilakukan hanya pada Skenario 1 dan Skenario 3 karena pengujian downtime menguji *failover* keepalived pada sistem. Pengujian downtime dilakukan dengan melakukan ping kepada alamat *virtual IP* yaitu 192.168.93.25 saat sedang dilakukannya pengujian sesuai dengan skenario 1 dan skenario 3 menggunakan httperf. Dengan menuliskan perintah seperti pada tabel 42 yang menunjukan bahwa interval selama 0.5 detik. Sehingga paket dikirimkan setiap 0.5 detik.

Table 6.1. Perintah Ping

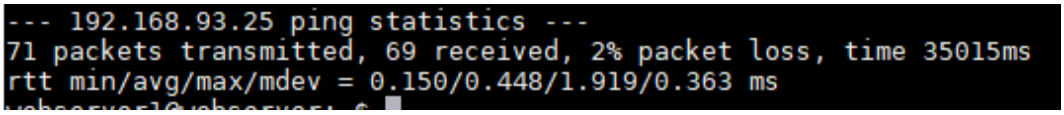
1	`$ping 192.168.93.25 -i 0.5`

Setelah melakukan ping pada pengujian akan meghasilkan hasil seperti pada gambar 6.1 dan gambar 6.2. Pada gambar 6.1 dapat dilihat bahwa terdapat *packet loss* yaitu *packet* dengan nomor 21 dan 22. Karena pengiriman paket dari nomor 20 langsung menjadi paket 23.



Gambar 6.1. Hasil Packet loss

Gambar 6.2 merupakan hasil akhir dari pengujian ping. Dapat dilihat bahwa terdapat 71 paket yang terkirim dan 69 paket yang diterima. Sehingga terdapat dua paket yang hilang.



Gambar 6.2. Jumlah Packet loss

6.1.1.1 Pengujian Downtime Skenario 1

Pengujian downtime pada skenario 1 ialah dengan melakukan ping ke alamat host pada saat sedang melakukan pengujian menggunakan httperf. Dengan mengirimkan beban yang berbeda dan terus meningkat.

Table 6.2. Hasil Pengujian Downtime Skenario 1

NO	Jumlah koneksi (/sec)	Nilai Downtime (/sec)	
		Round robin	Fallback
1	20	1	1
2	40	1	1
3	60	1	1
4	80	1	1
5	100	1	1

Dari hasil yang ditunjukan dapat disimpulkan bahwa lama waktu keepalived dalam melakukan perpindahan trafik dari *load balancer* 1 ke *load balancer* 2 hanya membutuhkan waktu 1 detik dimulai dari ketika *load balancer* 1 mati.

6.1.1.2 Pengujian Downtime Skenario 3

Pengujian downtime pada skenario 3 ialah dengan melakukan pingke alamat host pada saat sedang melakukan pengujian menggunakan httperf. Dengan mengirimkan beban yang berbeda – beda.

Table 6.3. Hasil Pengujian Downtime Skenario 3

NO	Jumlah koneksi (/sec)	Nilai Downtime (/sec)	
		Round robin	Fallback
1	20	1	1
2	40	1	1
3	60	1	1
4	80	1	1
5	100	1	1

Dari hasil yang ditunjukan dapat disimpulkan bahwa lama waktu keepalived dalam melakukan perpindahan trafik dari *load balancer* 1 ke *load balancer* 2 hanya membutuhkan waktu 1 detik dimulai dari ketika *load balancer* 1 mati.

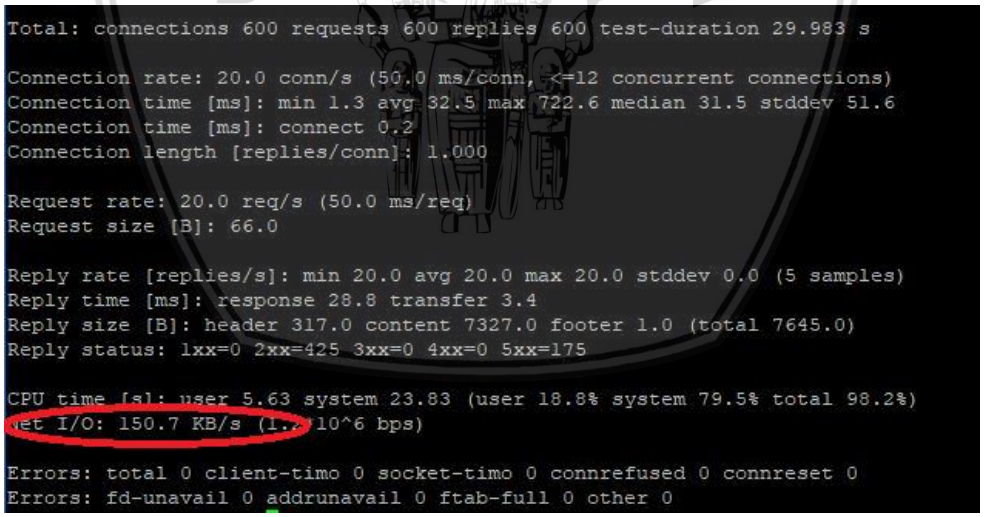
6.1.2 Pengujian Throughput

Pengujian throughput dilakukan pada setiap skenario dengan menggunakan httperf. Pengujian httperf dilakukan dengan cara menuliskan perintah seperti pada table 43.

Table 6.4 Perintah httperf

1	`$httperf –hog –server=x --num-conns=y –rate=z`

Untuk melakukan pengujian menggunakan httperf, harus mengisikan nilai yaitu X menunjukan alamat IP host yang ingin dituju. Nilai Y menunjukan jumlah koneksi secara keseluruhan dan nilai z menentukan jumlah *request* per detik. Pengujian yang dilakukan oleh penulis ialah dengan melakukan pengujian selama 30 detik dengan jumlah koneksi yang berbeda-beda setiap detiknya dan semakin meningkat. Gambar 6.3 menunjukan hasil dari pengujian httperf. Hasil throughput dari pengujian ditunjukan pada Net I/O yang ada pada hasil percobaan httperf.



Gambar 6.3. Contoh Hasil Throughput

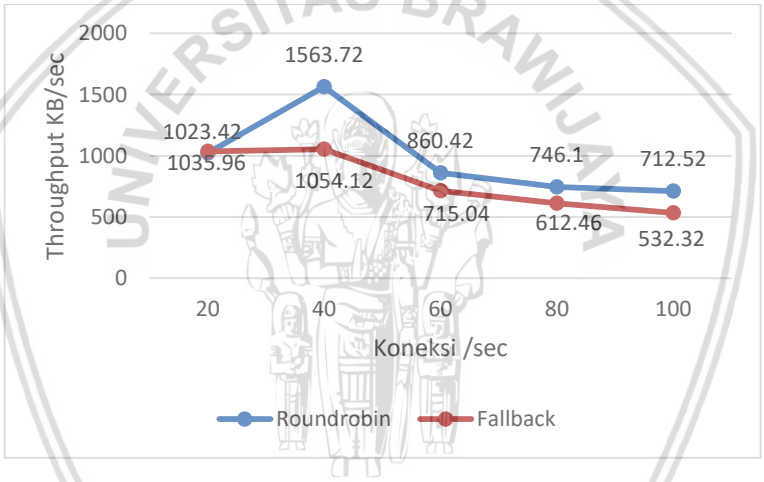
6.1.2.1 Hasil Pengujian Throughput Skenario 1

Pengujian throughput pada skenario 1 yaitu dengan mematikan *load balancer 1* secara sengaja pada saat sistem sedang berjalan dengan normal. Percobaan dilakukan selama 30 detik dengan jumlah koneksi yang berbeda – beda setiap detiknya yaitu 20, 40, 60, 80 dan 100.

Table 6.5. Hasil Throughput Skenario 1

NO	Jumlah koneksi (/sec)	Nilai Throughput (KB/sec)	
		Round robin	Fallback
1	20	1023.42	1035.96
2	40	1563.72	1054.12
3	60	860.42	715.04
4	80	746.1	612.46
5	100	712.52	532.32

Grafik dari hasil pengujian throughput skenario 1 dapat dilihat pada gambar dibawah ini.



Gambar 6.4. Grafik Hasil Throughput Skenario 1

Dari hasil pengujian throughput skenario 1, dapat dilihat bahwa untuk pengujan dengan jumlah koneksi 20 menghasilkan nilai thrmoughput 1023.42 untuk algoritma *round robin*, sedangkan untuk algoritma fallback menghasilkan nilai 1035.96. untuk jumlah koneksi 40, algoritma *round robin* menghasil nilai 1563.72 sedangkan untuk algoritma fallback ialah 1054.12. Lalu nilai *throughput* mengalami penurunan pada jumlah koneksi 60 yaitu algortima *round robin* menjadi 860.42 sedangkan fallback menjadi 715.04. Jumlah koneksi 80 menjadi 746.1 untuk algorittma *round robin*, sedangkan fallback ialah 612.46. Dan semakin menurun ketika jumlah koneksi 100 dengan hasil 712.52 untuk algoritma *round robin* dan 532.32 untuk algoritma fallback.

6.1.2.2 Hasil Pengujian Throughput Skenario 2

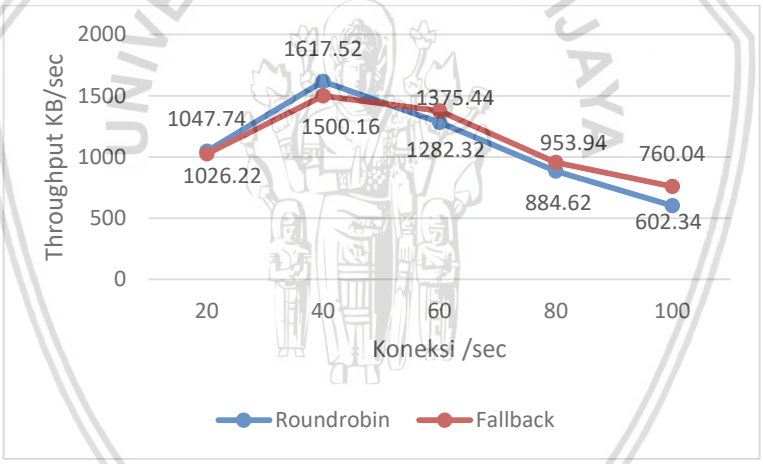
Pengujian throughput pada skenario 2 yaitu dengan mematikan *web server 2* secara sengaja pada saat sistem sedang berjalan dengan normal. Percobaan

dilakukan selama 30 detik dengan jumlah koneksi yang berbeda – beda setiap detiknya yaitu 20, 40, 60, 80 dan 100.

Table 6.6. Hasil Throughput Skenario 2

NO	Jumlah koneksi (/sec)	Nilai Throughput (KB/sec)	
		Round robin	Fallback
1	20	1047.74	1026.22
2	40	1617.52	1500.16
3	60	1282.32	1375.44
4	80	884.62	953.94
5	100	602.34	760.04

Grafik dari hasil pengujian throughput skenario 2 dapat dilihat pada gambar dibawah ini.



Gambar 6.5 Grafik Hasil Throughput Skenario 2

Dari hasil pengujian throughput skenario 2, dapat dilihat bahwa untuk pengujan dengan jumlah koneksi 20 menghasilkan nilai throughput 1047.74 untuk algoritma *round robin*, sedangkan untuk algoritma fallback menghasilkan nilai 1026.22. untuk jumlah koneksi 40, algoritma *round robin* menghasil nilai 1617.52 sedangkan untuk algoritma fallback ialah 1500.16. lalu nilai throughput mengalami penurunan pada jumlah koneksi 60 yaitu algortima *round robin* menjadi 1375.44 sedangkan fallback menjadi 1282.32. Jumlah koneksi 80 menjadi 884.62 untuk algorittma *round robin*, sedangkan fallback ialah 953.94. Dan semakin menurun ketika jumlah koneksi 100 dengan hasil 602.34 untuk algoritma *round robin* dan 760.04 untuk algoritma fallback.

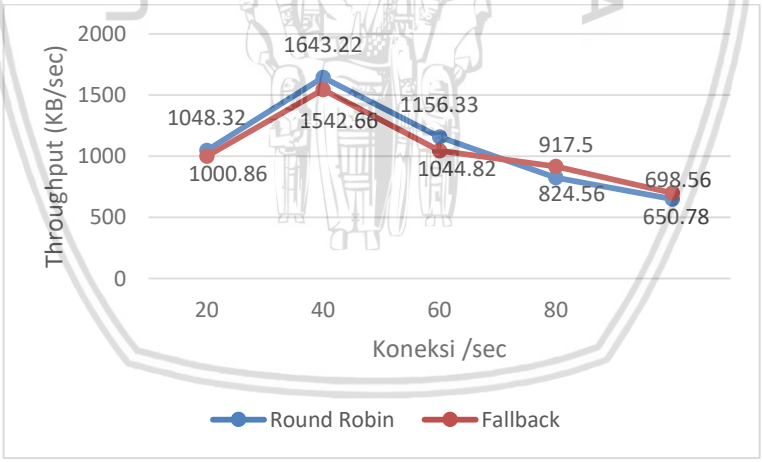
6.1.2.3 Hasil Pengujian Throughput Skenario 3

Pengujian throughput pada skenario 3 yaitu dengan mematikan *load balancer 1* dan *web server 1* secara sengaja pada saat sistem sedang berjalan dengan normal. Percobaan dilakukan selama 30 detik dengan jumlah koneksi yang berbeda – beda setiap detiknya yaitu 20, 40, 60, 80 dan 100.

Table 6.7. Hasil Throughput Skenario 3

NO	Jumlah koneksi (/sec)	Nilai Throughput (KB/sec)	
		Round robin	Fallback
1	20	1048.32	1000.86
2	40	1643.22	1542.66
3	60	1156.33	1044.82
4	80	824.56	917.5
5	100	650.78	698.56

Grafik dari hasil pengujian throughput skenario 3 dapat dilihat pada gambar dibawah ini.



Gambar 6.6. Grafik Hasil Throughput Skenario 3

Dari hasil pengujian throughput skenario 3, dapat dilihat bahwa untuk pengujan dengan jumlah koneksi 20 menghasilkan nilai throughput 1048.32 untuk algoritma *round robin*, sedangkan untuk algoritma fallback menghasilkan nilai 1000.86. untuk jumlah koneksi 40, algoritma *round robin* menghasil nilai 1643.22 sedangkan untuk algoritma fallback ialah 1542.66. lalu nilai throughput mengalami penurunan pada jumlah koneksi 60 yaitu algortima *round robin* menjadi 1156.33 sedangkan fallback menjadi 1044.82. Jumlah koneksi 80 menjadi 824.56 untuk algoritma *round robin*, sedangkan fallback ialah 917.5. Dan semakin menurun ketika jumlah koneksi 100 dengan hasil 650.87 untuk algoritma *round robin* dan 698.56 untuk algoritma *fallback*.

6.1.3 Pengujian *CPU usagae* dan *Memory* Usage

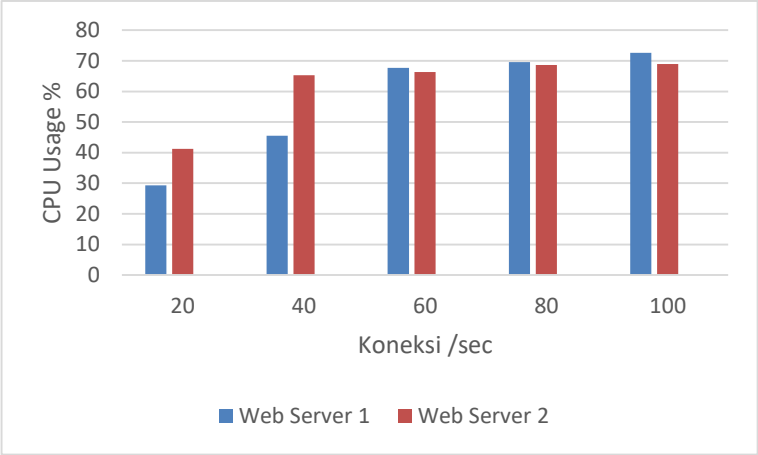
6.1.3.1 Pengujian *CPU usagae* dan *Memory* Usage Skenario 1

Pengujian *CPU usagae* dan *Memory Usage* pada skenario 1 yaitu dengan mematikan secara sengaja *load balancer 1* pada saat sedang berjalan dengan normal. Lalu selanjutnya *load balancer 2* menggantikan tugas dari *load balancer 1* yaitu meneruskan *request* ke web server. sedangkan kedua web server 1 dan web server 2 tetap berjalan untuk melakukan proses. Percobaan dilakukan selama 30 detik pada setiap koneksi dengan jumlah koneksi perdetik yang berbeda-beda dan semakin meningkat.

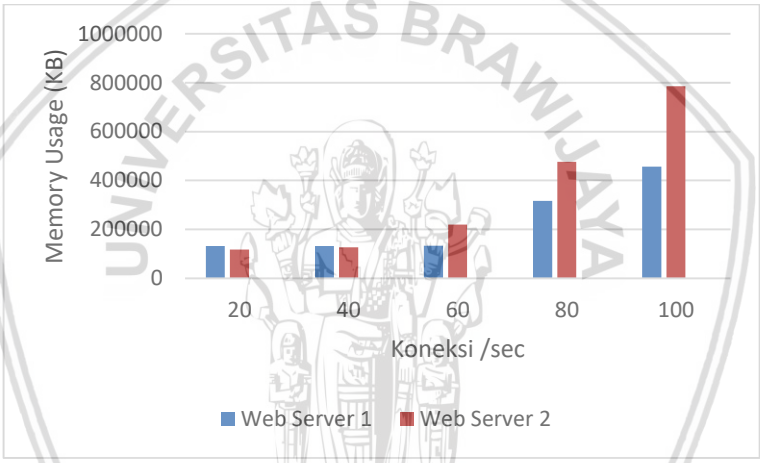
Table 6.8. Hasil *CPU usagae* dan *Memory* Usage Skeqnario 1 Algoritma *Round robin*

NO	Jumlah koneksi (/sec)	*CPU usagae* (%)		*Memory* Usage (KB)	
		Web Server 1	Web Server 2	Web Server 1	Web Server 2
1	20	29.3	41.2	133072	118300
2	40	45.5	65.3	133706	127708
3	60	67.7	66.3	133716	220596
4	80	69.6	68.6	316492	476460
5	100	72.6	69.0	456672	784832

Dengan hasil pengujian seperti table diatas, dapat dilihat grafik dari pengujian seperti pada gambar grafik dibawah ini. Dari hasil yang diperoleh untuk pengujian *CPU usagae* ialah untuk algoritma *round robin* pada web server 1 ketika jumlah koneksi 20 *CPU usagae* bernilai 29.3 %, untuk koneksi 40 dengan hasil 45.5 %,konneksi 60 dengan jumlah hasil 67.7 %, sedangkan untuk jumlah koneksi 80 menghasilkan jumlah *CPU usagae* 69.6 dan jumlah koneksi 100 menghasilkan nilai 72.6 %. Lalu untuk web server 2 memperoleh hasil 41.2 % untuk jumlah koneksi 20, 65.3% untuk jumlah koneksi 40, untuk jumlah koneksi 60 menghasilkan nilai sebesar 66.3 ,jumlah koneksi 80 menghasilkan nila sebesar 68.6 dan ketika jumlah koneksi 100 maka menghasilkan nilai sebesar 69.6. Nilai *CPU usagae* mengingkat seiring dengan bertambahnya jumlah koneksi yang ada. Hal itu berlaku juga untuk nilai *Memory* Usage, untuk web server 1 menghasilkan nilai sebesar 133072 KB dengan jumlah koneksi 20, untuk jumlah koneksi 40 menghasilkan nilai sebesar 133706 KB, lalu jumlah koneksi 60 menghasilkan nilai sebesar 133716 KB. Nilai untuk jumlah koneksi lalu jumlah koneksi 80 menghasilkan nilai sebesar 316492 KB dan jumlah koneksi 100 menghasilkan nilai sebesar 456672 KB. Selanjutnya untuk nilai *Memory* Usage pada web server 2 ialah 11830 KB untuk jumlah koneksi 20, 122708 KB untuk jumlah koneksi 40, 220596 KB untuk jumlah koneksi 60, 476460 KB untuk jumlah koneksi 80 dan 784832 KB untuk jumlah koneksi 100. Dapat disimpulkan bahwa meningkatkan jumlah koneksi maka nilai *CPU usagae* dan *Memory* Usage juga meningkat.



Gambar 6.7. Grafik Hasil Pengujian *CPU usagae* Skenario 1 Algoritma *Round robin*



Gambar 6.8. Grafik Hasil Pengujian *Memory* Usage Skenario 1 Algoritma *Round robin*

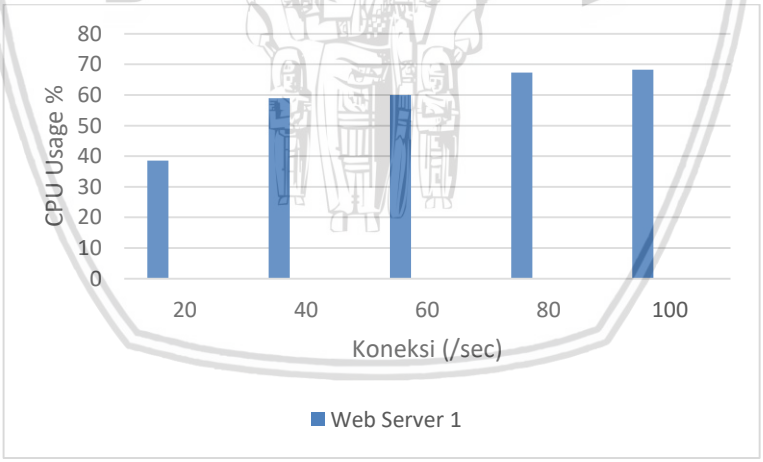
Untuk algoritma *fallback* pengujian *CPU usagae* dan *Memory Usage* pada skenario 1 hanya diuji pada web server 1 dikarenakan algoritma *fallback* hanya memilih *web server* 1 secara terus menerus untuk memberikan layanan dikarenakan algoritma *fallback* yang hanya memilih satu *backend server* secara terus menerus selama server tersebut masih berstatus *health* atau masih sehar. Hasil pengujian *CPU usagae* dan *Memory* Usage dapat dilihat dari table dibawah ini.

Table 6.9. Hasil *CPU usagae* dan *Memory* Usage Skenario 1 Algoritma Fallback

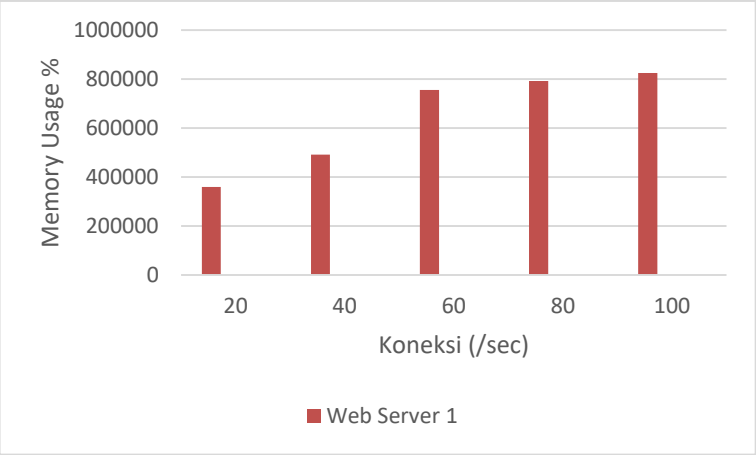
NO	Jumlah koneksi (/sec)	*CPU usagae* (%)		*Memory* Usage (KB)	
		Web Server 1	Web Server 2	Web Server 1	Web Server 2

1	20	38.5	0	360172	0
2	40	59	0	491948	0
3	60	60	0	755484	0
4	80	67.3	0	792212	0
5	100	68.3	0	825264	0

Grafik dari hasil pengujian *CPU usagae* dan *Memory* Usage untuk algoritma *fallback* skenario 1 dapat dilihat pada gambar dibawah ini. Dari hasil yang diperoleh untuk skenario 1 algoritma fallback untuk *CPU usagae* meghasilkan nilai sebesar 38.5 % untuk jumlah koneksi 20, sedangkan nilai *CPU usagae* 59 % untuk jumlah koneksi 40, 60 % untuk jumlah koneksi 60, dan 67.3 % untuk jumlah koneksi 80. Dan yang paling tinggi ialah 68.3 % untuk jumlah koneksi 100. Lalu untuk pengujian *Memory* Usage dihasilkan nilai sebesar 360172 KB untuk jumlah koneksi 20, 491948 untuk jumlah koneksi 40, nilai 755484 KB untuk jumlah koneksi 60, 792212 KB untuk jumlah koneksi 80 dan nilai 825264 KB untuk jumlah koneksi 100. Dapat disimpulkan bahwa semakin tinggi nilai jumlah koneksi maka hasil yang diperoleh semakin meningkat.



Gambar 6.9. Hasil Pengujian *CPU usagae* Algoritma Fallback



Gambar 6.10. Hasil Pengujian *Memory* Usage Algoritma Fallback

6.1.3.2 Pengujian *CPU usagae* dan *Memory* Usage Skenario 2

Pengujian *CPU usagae* dan *Memory* Usage pada skenario 2 yaitu dengan mematikan secara sengaja *web server 1* pada saat sedang berjalan dengan normal. Percobaan dilakukan selama 30 detik pada setiap koneksi dengan jumlah koneksi perdetik yang berbeda-beda dan semakin meningkat. Sehingga *load balancer* akan memfokuskan kepada *web server 2* untuk memberikan layanan. Hasil dari pengujian *CPU usagae* dan *Memory* Usage untuk algoritma *round robin* dan fallback dapat dilihat dari table dibawah ini.

Table 6.10 Hasil *CPU usagae* dan *Memory* Usage Skenario 2 Algoritma *Round robin*

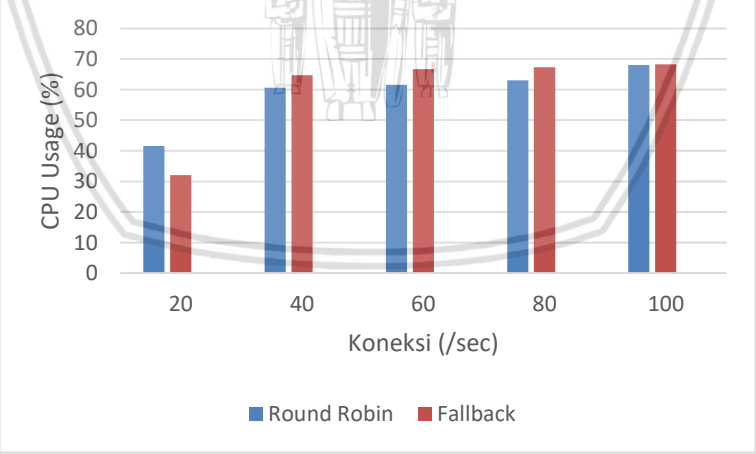
NO	Jumlah koneksi (/sec)	*CPU usagae* (%)		*Memory* Usage (KB)	
		Web Server 1	Web Server 2	Web Server 1	Web Server 2
1	20	0	41.6	0	112624
2	40	0	60.6	0	140988
3	60	0	61.6	0	191608
4	80	0	63.0	0	363516
5	100	0	68	0	525172

Table 6.11 Hasil *CPU usagae* dan *Memory* Usage Skenario 2 Algoritma Fallback

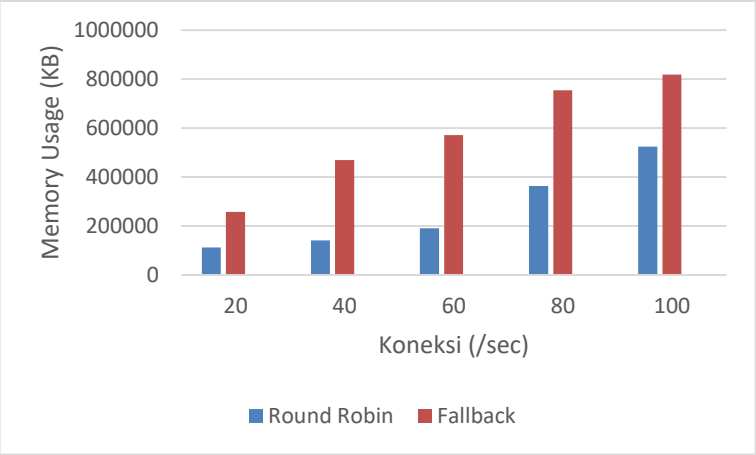
NO	Jumlah koneksi (/sec)	*CPU usagae* (%)		*Memory* Usage (KB)	
		Web Server 1	Web Server 2	Web Server 1	Web Server 2
1	20	0	32.1	0	258128
2	40	0	64.7	0	469696

3	60	0	66.7	0	572040
4	80	0	67.3	0	754304
5	100	0	68.3	0	819128

Dengan hasil pengujian seperti table diatas, dapat dilihat grafik dari pengujian seperti pada grafik dibawah. Hasil yang diperoleh dari pengujian *CPU usagae* untuk skenario 2 ialah pada web server 2 dengan algoritma *round robin* ketika jumlah koneksi 20 maka menghasilkan nilai sebesar 41.6 %, jumlah koneksi 40 menghasilkan nilai sebesar 60.6%, jumlah koneksi 60 menghasilkan nilai sebesar 61.6%,jumlah koneksi 80 menghasilkan nilai sebesar 63.0% dan jumlah koneksi 100 menghasilkan nilai sebesar 68%. Selanjutnya pengujian *Memory* usage menghasilkan nilai sebesar 112624 KB untuk jumlah koneksi 20, 140988 KB untuk jumlah koneksi 40, 191608 KB untuk jumlah koneksi 60, 363516 KB untuk jumlah koneksi 80, dan nilai sebesar 525127 KB untuk jumlah koneksi 100. Selanjutnya untuk algoritma fallback untuk pengujian *CPU usagae* menghasilkan nilai 32.1% untuk jumlah koneksi 20, 64.7% untuk jumlah koneksi 40, 66.7% untuk jumlah koneksi 60, 67.3 % untuk jumlah koneksi 80 dan 68.3% untuk jumlah koneksi 100. Lalu untuk pengujian *memory* usage menghasilkan nilai sebesar 258128 KB untuk jumlah koneksi 20, 469696 KB untuk jumlah koneksi 40, 572040 KB untuk jumlah koneksi 60, 754304 KB untuk jumlah koneksi 80 dan 819218 KB untuk jumlah koneksi 100. Dapat disimpulkan bahwa semakin meningkat jumlah koneksi maka nilai *CPU usagae* dan *Memory* Usage juga ikut meningkat.



Gambar 6.11. Hasil Pengujian *CPU usagae* Skenario 2



Gambar 6.12. Hasil Pengujian *Memory* Usage Skenario 2

6.1.3.3 Pengujian *CPU usagae* dan *Memory* Usage Skenario3

Pengujian *CPU usagae* dan *Memory* usage pada skenario 3 yaitu dengan mematikan secara sengaja *load balancer 1* dan *web server 1* pada saat sedang berjalan dengan normal. Sehingga pengujian *CPU usagae* dan *Memory* Usage dilakukan pada web server 2. Percobaan dilakukan selama 30 detik pada setiap koneksi dengan jumlah koneksi perdetik yang berbeda-beda dan semakin meningkat yaitu 20, 40 , 60, 80 dan 100 setiap detiknya. Pengujian dilakukan kepada dua algoritma yaitu algoritma *round robin* dan algoritma fallback.

Table 6.12 Hasil *CPU usagae* dan *Memory* Usage Skenario 3 Algoritma *Round robin*

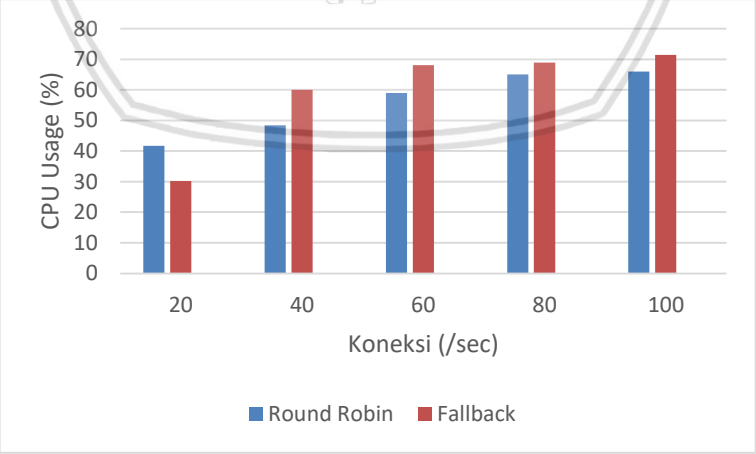
NO	Jumlah koneksi (/sec)	*CPU usagae* (%)		*Memory* Usage (KB)	
		Web Server 1	Web Server 2	Web Server 1	Web Server 2
1	20	0	41.7	0	271924
2	40	0	44.9	0	276720
3	60	0	59	0	277152
4	80	0	65.3	0	642611
5	100	0	66	0	680376

Table 6.13 Hasil *CPU usagae* dan *Memory* Usage Skenario 3 Algoritma Fallback

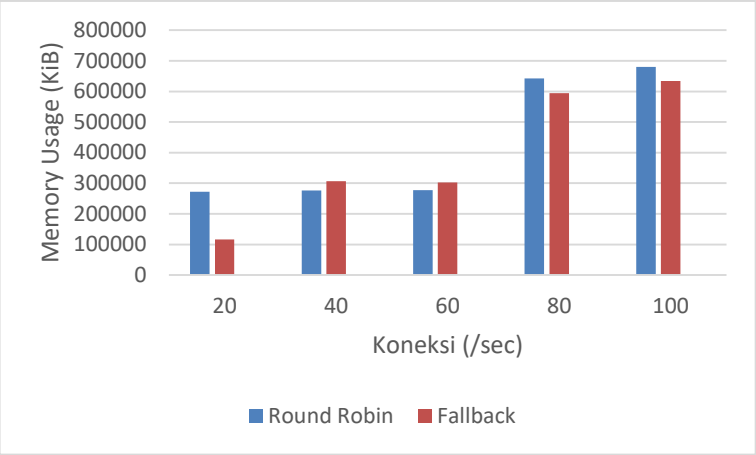
NO	Jumlah koneksi (/sec)	*CPU usagae* (%)		*Memory* Usage (KB)	
		Web Server 1	Web Server 2	Web Server 1	Web Server 2
1	20	0	30.2	0	116296
2	40	0	60	0	302532

3	60	0	68.1	0	306284
4	80	0	68.9	0	594380
5	100	0	71.4	0	633960

Dari hasil pengujian *CPU usagae* dan *Memory Usage* yang dilakukan pada web server 2 untuk skenario 3 dan menggunakan algoritma yang berbeda yaitu algoritma *fallback* dan algoritma *round robin*. Maka dapat di simpulkan hasil seperti pada table 6.12 dan 6.13. Grafik untuk hasil pengujian diatas dapat dilihat pada gambar 6.13 dan 6.14. Hasil yang diperoleh dari pengujian *CPU usagae* untuk skenario 3 ialah pada web server 2 dengan algoritma *round robin* ketika jumlah koneksi 20 maka menghasilkan nilai sebesar 41.7 %, jumlah koneksi 40 menghasilkan nilai sebesar 44.9%, jumlah koneksi 60 menghasilkan nilai sebesar 59%,jumlah koneksi 80 menghasilkan nilai sebesar 65.3% dan jumlah koneksi 100 menghasilkan nilai sebesar 66%. Selanjutnya pengujian *Memory* usage menghasilkan nilai sebesar 271924 KB untuk jumlah koneksi 20, 276720 KB untuk jumlah koneksi 40, 227152 KB untuk jumlah koneksi 60, 642611 KB untuk jumlah koneksi 80, dan nilai sebesar 680376 KB untuk jumlah koneksi 100. Selanjutnya untuk algoritma fallback untuk pengujian *CPU usagae* menghasilkan nilai 30.2% untuk jumlah koneksi 20, 60% untuk jumlah koneksi 40, 68.1% untuk jumlah koneksi 60, 68.9 % untuk jumlah koneksi 80 dan 70% untuk jumlah koneksi 100. Lalu untuk pengujian *memory* usage menghasilkan nilai sebesar 116296 KB untuk jumlah koneksi 20, 302532 KB untuk jumlah koneksi 40, 306284 KB untuk jumlah koneksi 60, 594380 KB untuk jumlah koneksi 80 dan 633960 KB untuk jumlah koneksi 100. Dapat disimpulkan bahwa semakin meningkat jumlah koneksi maka nilai *CPU usagae* dan *Memory* Usage juga ikut meningkat.



Gambar 6.13. Hasil Pengujian *CPU usagae* untuk skenario 3



Gambar 6.14. Hasil Pengujian *Memory* Usage skenario 3

6.1.4 Pengujian Fungsionalitas

Pengujian fungsionalitas dilakukan untuk mengetahui apakah sistem sudah berjalan dengan baik dan benar sesuai fungsinya. Terdapat kebutuhan fungsional untuk mengujian fungsionalitas dari sistem. Kebutuhan yang ada meliputi komponen secara keseluruhan pada sistem yaitu *load balancer* 1 dan *load balancer 2, web server 1* dan *web server 2* dan *database server.*

Table 6.14. Hasil Pengujian Fungsionalitas

NO	Kebutuhan Fungsional	Hasil
1	Sistem mampu memberikan layanan meski salah satu *load balancer* mengalami kegagalan.	Valid
2	Load balancer mampu membagikan beban tugas kepada web server menggunakan algoritma *round robin* dan *fallback*.	Valid
3	*Failover* mampu memidahkan trafik dari *load balancer* 1 ke *load balancer 2* jika *load balancer 1* mengalami kegagalan.	Valid
4	Web server mampu menerima tugas yang diberikan oleh *load balancer.*	Valid

5	Jika salah satu web server mati, maka server lainnya tetap dapat memberikan layanan.	Valid
6	Database server mampu menyimpan data dan memberikan data yang diminta oleh *client.*	Valid
7	Penggunaan GlusterFS untuk sinkornisasi file antara web server 1 dan web server2.	Valid

6.2 Analisa dan Hasil

Dari pengujian-pengujian yang telah dilakukan dapat disimpulkan hasil pengujian seperti dibawah ini.

1. Dari pengujian downtime yang dilakukan untuk skenario 1 dan skenario 2 diperoleh hasil bahwa *failover* hanya membutuhkan waktu 1 detik untuk berpindah dari *load balancer 1* ke *load balancer 2.*
2. Dari hasil pengujian *throughput* yang dilakukan pada setiap skenario, ketika jumlah koneksi memasuki nilai 60 per detik maka *throughput* mengalami penurunan. Hal tersebut dikarenakan terjadi *bottleneck* pada saat jumlah koneksi bertambah. *Bottleneck* merupakan keadaan dimana salah satu komponen pada sistem memyebabkan perfoma pada sistem menjadi terhambat.
3. Nilai throughput tertinggi secara keseluruh ialah dengan menggunakan algoritma *round robin.* Dibandingkan dengan algoritma *fallback,* algoritma *round robin* lebih unggul.
4. Dari hasil pengujian *CPU* dan *memory usage* yang dilakukan pada keseluruhan skenario, menjelaskan bahwa seiring meningkatnya jumlah koneksi maka terjadi pula peningkatan pada nilai *CPU* dan *memory.*
5. Penggunaan *CPU usagae* dan *memory usage* lebih tinggi dengan menggunakan algoritma *fallback*. Sehingga dapat disimpulan *algoritma fallback* lebih banyak memanfaatkan sumber daya dibandingkan dengan algoritma *round robin.*
6. Pengujian fungsional pada penelitian ini dapat dilakukan sesuai dengan kebutuhan fungsional yang ada.

BAB 7 PENUTUP

7.1 Kesimpulan

Berdasarkan penelitian dengan perancangann dan pengujian yang telah dilakukan maka menghasilkan beberapa kesimpulan yang dilakukan pada penelitian. Kesimpulan berisikan hasil dan analisis terhadap penelitian yang telah dilakukan.

1. Berdasarkan perancangan dan pengujian yang dilakukan untuk penelitian dapat disimpulkan bahwa sistem mampu menjamin ketersediaan tinggi dengan nilai *downtime* selama 1 detik menggunakan teknologi *failover* keepalived. Pada *load balancer* Varnish mampu mebagi beban kerja kepada web server untuk memberikan layanan sesuai dengan algoritma yang ditentukan yaitu *round robin* dan *fallback*. Serta *Distributed File System* GlusterFS mampu melakukan distribusi file data dari web server 1 ke web server 2 sehingga kedua server tersebut dapat memilki file yang sama.
2. Dari pengujian yang telah dilakukan menggunakan teknologi *failover, load balancing,* dan *distributed file system* menghasilkan nilai *throughput* semakin menurun berdasarkan tingginya jumlah koneksi. Nilai *throughput* tertinggi diperoleh dari jumlah koneksi 40 dan nilai *throughput* terendah diperoleh dari jumlah koneksi terbesar yaitu 100. Secara keseluruhan algoritma *round robin* lebih unggul dibandingkan dengan algoritma *fallback.* Penggunaan sumber daya CPU dan *Memory* memperoleh hasil bahwa semakin tinggi jumlah koneksi yang dilakukan maka semakin tinggi nilai *CPU* dan *Memory*. Algoritma *fallback* lebih banyak memanfaatkan sumber daya dibandingkan dengan algoritma *round robin.*

7.2 Saran

Terdapat beberapa saran yang diajukan berdasarkan penelitian yang telah dilaksanakan. Saran yang diberikan bisa menjadi acuan untuk penelitian selanjutnya.

1. Penelitian selanjutnya diharapkan dapat menggabungkan teknologi *failover* dengan web server lain.
2. Menambahkan teknologi lain untuk menambah kehandalan dari sistem.
3. Menambahkan parameter pengujian lain pada sistem.
4. Menggunakan *load balancer* dan algoritma lain untuk melakukan pembagian beban kerja.

DAFTAR PUSTAKA

Aryal, S., 2017. *Highly Available Web Service Using the Raspberry Pi Cluster,* Finland: s.n.

Bakhtiyari, S., 2012. *Performance Evaluation of the Apache Traffic Server and Varnish Reverse Proxy,* Oslo: s.n.

Braastad, E., 2006. *Management of High Availability Services Using Virtualization,* Oslo: s.n.

Bzoch, P., 2012. *Distributed File System,* Czech Republic: s.n.

Cassen, A., 2017. *Keepalived User Guide.* s.l.:s.n.

Cisco, 2017. *ASDM Book 1: Cisco ASA Series General Operations ASDM Configuration Guide 7.7,* s.l.: s.n.

Dudnik, A., 2017. *Creating a High Availability Cluster With Two Physical Servers and Virtual Machines,* Finland: s.n.

Gopinath, G. P. P. & Vasudevan, S. K., 2015. *An in-depth analysis and study of Load balancing techniques in the cloud computing environment,* Coimbatore: s.n.

Ha, L. Q., Xie, J., Millington, D. & Waniss, A., 2015. *Comparative Perfomance Analysis of PostgreSQL High Availability Database Clusters through Containment,* Canada: s.n.

Hidayat, F., 2012. *Implementasi dan Analisa Redundansi dan High Availability Dalam Server untuk Diskless Thing Client Berbasis Storage Area Network,* Depok: s.n.

Hong, L. G., Hua, Z. & Zhi, L. G., 2010. *Building A Secure Web Server Based on OpenSSL and Apache,* China: s.n.

Huang, C. W., Lai, C. C., Lin, C. A. & Lie, C. M., 2015. *File System Allocation in Cloud Storage Service with GlusterFS and Lustre,* Taipei, Taiwan: s.n.

Kamp, H. P., 2010. *You're Doing It Wrong,* s.l.: s.n.

Lan, H. & Wang, X., 2012. *Research and Design of Concurrent Web Server on Linux System,* Ganzhou: s.n.

Li, J., Lin, C. & Shi, F., 2010. *Availability Analysis of Web-server Clusters with QoS-aware Load Balancing,* Beijing: s.n.

Lyngstol, K., 2017. *Varnish Foo.* s.l.:s.n.

Prasetijo, A. B., Widianto, E. D. & Hidayatullah, E. T., 2016. *Performance Comparisons of Web Server Load Balancing Algorithms on HAProxy and Heartbeat,* Semarang: s.n.

Putra, R. H. & Sugeng, W., 2016. *Implementasi Cluster Server pada Raspberry Pi dengan Menggunakan Metode Load Balancing,* Bandung: s.n.

Rosebrock, E. & Filson, E., 2004. *Setting Up LAMP: Getting Linux, Apache, MySQL and PHP Working Together.* San Fransisco: s.n.

Singh, H. & Kumar, S., 2011. *Dispatcher Based Dynamic Load Balancing on Web Server System,* India: s.n.

Sulistyanto, I. H., 2015. *Implementasi High Availability Server dengan Teknik Failover Virtual Computer Cluster,* Surakarta: s.n.

Yuan, P., Li, Y., Jin, H. & Liu, L., 2015. *Self-Adaptive Extracting Academic Entities from World Wide Web,* China: s.n.

Zongyu, X. & Xingxuan, W., 2014. *A Modified Round-robin Load-balancing Algorithm for Cluster-based,* Nanjing: s.n.

Zongyu, X. & Xingxuan, W., 2015. *A Predictive Modified Round robin Scheduling Algorithm for Web Server Clusters ,* Shanghai: s.n.