PERBANDINGAN KINERJA TEKNOLOGI *FAILOVER* BERBASIS KLASTER (HEARTBEAT) DENGAN TEKNOLOGI *FAILOVER* BERBASIS JARINGAN (KEEPALIVED)

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Dian Pratama
NIM: 165150209111006

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2021

PENGESAHAN

PERBANDINGAN KINERJA TEKNOLOGI *FAILOVER* BERBASIS KLASTER (HEARTBEAT) DENGAN *FAILOVER* BERBASIS JARINGAN (KEEPALIVED)

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Dian Pratama
NIM: 165150209111006

Skripsi ini telah diuji dan dinyatakan lulus pada
29 Juli 2021

Telah diperiksa dan disetujui oleh :

Dosen Pembimbing I	Dosen Pembimbing II
Adhitya Bhawiyuga, S.Kom., M.Sc.	Fariz Andri Bakhtiar, S.T., M.Kom.
NIP: 19890720 201803 1 002	NIK: 2017098403141001

Mengetahui
Ketua Jurusan **Teknik Informatika**

Achmad Basuki, S.T., M.MG., Ph.D
NIP: 19741118 200312 1 002

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, 19 Juli 2021

Dian Pratama

NIM: 165150209111006

ABSTRAK

Dian Pratama, Perbandingan Kinerja Teknologi Failover Berbasis Klaster (Heartbeat) Dengan Teknologi Failover Berbasis Jaringan (Keepalived)

Pembimbing: Adhitya Bhawiyuga, S.Kom., M.Sc. dan Fariz Andri Bakhtiar, S.T., M.Kom.

Dewasa ini, data dan informasi merupakan hal yang sangat penting. Data dan informasi pada umumnya diolah dan disimpan dalam *server*. Supaya data dan informasi tersebut dapat selalu diakses oleh pengguna selama 24 jam maka dibutuhkan *server* dengan ketersediaan tinggi yang dapat beroperasi secara terus menerus dan tidak mengalami kegagalan. Teknik *failover* merupakan salah satu cara untuk mengimplementasikan *server* dengan ketersediaan tinggi. Solusi yang diajukan peneliti ialah implementasi *server* dengan ketersediaan tinggi menggunakan perangkat lunak Heartbeat dan Keepalived. Penelitian ini mengimplementasikan dua topologi yaitu *High Availability Web Server* dan *High Availability Load Balancing*. Pada implementasi *High Availability Web Server* menggunakan LAMP server yaitu Linux Apache, Mysql dan PHP untuk konten *web* menggunakan Wordpress. Pada implementasi *High Availability Load Balancing* menggunakan perangkat lunak Haproxy untuk bertugas sebagai *load balancer*. Pengujian dilakukan dari segi fungsional, *downtime* dan *failback*. Masing-masing pengujian dilakukan dengan 3 skenario berbeda dan dilakukan 12 kali pengujian untuk dicari hasil rata-ratanya. Dari hasil pengujian fungsional didapatkan bahwa implementasi sistem telah berjalan sesuai fungsinya. Dari hasil pengujian pada topologi HA *web server* dan topologi HA *load balancing* didapatkan nilai *downtime* terendah pada skenario 1 sebesar 4 detik dan 4,05 detik, pada skenario 2 sebesar 0 detik dan 0 detik, sedangkan pada skenario 3 sebesar 0,88 detik dan 0,78 detik. Pada pengujian *failback* topologi HA *web server* dan HA *load balancing* didapatkan hasil terendah pada skenario 1 sebesar 0,68 detik dan 0,48 detik, pada skenario 2 sebesar 0 detik dan 0 detik sedangkan pada skenario 3 sebesar 0,88 detik dan 0,53 detik. Berdasarkan hasil tersebut, implementasi Keepalived atau Heartbeat dapat menjadi solusi untuk sistem *web server* atau *load balancing* yang membutuhkan ketersediaan tinggi.

Kata kunci: *high availability, failover, keepalived, heartbeat, lamp server*.

ABSTRACT

Dian Pratama, ***Performance Comparison of Cluster-Based Failover Technology (Heartbeat) With Network-Based Failover Technology (Keepalived)***

Supervisors: Adhitya Bhawiyuga, S.Kom., M.Sc. and Fariz Andri Bakhtiar, S.T., M.Kom.

Today, data and information are very important. Data and information are generally processed and stored on servers. So that the data and information can always be accessed by users for 24 hours, a server with high availability is needed that can operate continuously and does not fail. The failover technique is one way to implement high-availability servers. The solution proposed by the researcher is the implementation of a high-availability server using Heartbeat and Keepalive software. This research implements two topologies, namely High Availability Web Server and High Availability Load Balancing. In the implementation of High Availability Web Server using a LAMP server, namely Linux Apache, Mysql and PHP for web content using WordPress. In the implementation of High Availability Load Balancing, Haproxy software is used to act as a load balancer. Tests are carried out in terms of functionality, downtime and failback. Each test was carried out with 3 different scenarios and 12 times were tested to find the average result. From the results of functional testing, it is found that the implementation of the system has been running according to its function. From the test results on the HA web server topology and HA load balancing topology, the lowest downtime values in scenario 1 are 4 seconds and 4.05 seconds, in scenario 2 it is 0 seconds and 0 seconds, while in scenario 3 it is 0.88 seconds and 0 .78 seconds. In testing the failback topology of HA web server and HA load balancing, the lowest results were obtained in scenario 1 of 0.62 seconds and 0.48 seconds, in scenario 2 it was 0 seconds and 0 seconds while in scenario 3 it was 0.88 seconds and 0.53 second. Based on these results, the implementation of Keepalive or Heartbeat can be a solution for web server or load balancing systems that require high availability.

Keywords: high availability, failover, keepalived, heartbeat, lamp server

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB 1 PENDAHULUAN

1.1 Latar belakang

Data dan informasi merupakan sesuatu yang sangat penting pada saat ini. Agar informasi atau data dapat diakses , maka dibutuhkan server untuk mengolah serta menyimpan data. *Server* dituntut harus dapat bekerja selama 24 jam agar data yang disimpan atau diolah pada server tersebut dapat selalu diakses. Namun, terkadang *server* yang handal sekalipun tetap dapat mengalami kegagalan. Server yang mengalami kegagalan dapat disebabkan oleh berbagai faktor, seperti faktor kondisi seperti kegagalan *hardware*, *software* dan lain sebagainya. Untuk mengatasi hal tersebut, maka munculah sebuah solusi yaitu server yang memiliki sifat ketersediaan tinggi atau dapat disebut *high availability server*. Hal tersebut dapat dilakukan dengan cara memasang perangkat lunak cluster pada *server master* dan *server backup* (Vugt, 2014)

High Availability Server dapat terus berjalan untuk melayani *request* dari pengguna. Salah satu teknik untuk memperoleh *high availability* yaitu dengan menerapkan teknik *failover*. Teknik *failover* diterapkan pada *server*, *server* diduplikasi dan server duplikasi tersebut dijadikan satu klaster dengan *server* utama. Tujuan *failover* adalah agar pengguna tetap dapat mengakses ke *server* atau walaupun *server* sedang mengalami kegagalan *software*, *hardware* atau lainnya yang mengakibatkan *server* tersebut tidak dapat diakses (Hirt, 2009). Secara sederhana, cara kerja *failover* adalah jika server utama mengalami kegagalan maka secara otomatis *server* cadangan akan mengambil alih tugas *server* utama.

Implementasi *failover* pada server klaster dapat membuat *server* klaster tersebut menghindari *single point of failure*. *Single point of failure* adalah sebuah komponen yang penting dalam sistem, yang dimana jika komponen ini berhenti beroperasi maka keseluruhan sistem akan berhenti beroperasi pula. Pada sistem yang memiliki sifat ketersediaan tinggi, tidak membutuhkan komponen *single point of failure*. Ini berarti bahwa, jika ada kegagalan pada sebuah komponen tidak akan menyebabkan sistem berhenti bekerja (Techopedia, 2018). Proses *failover* membutuhkan waktu beberapa saat untuk perpindahan dari *node master* ke *node slave*. Dalam *failover*, terjadi juga proses *failback*. Proses *failback* adalah proses kembalinya *node* yang sebelumnya mengalami kegagalan.

Sebelumnya, Rahmad Dani pada penelitian *failover* yang berjudul "Perancangan dan Pengujian Load Balancing dan *Failover* Menggunakan Nginx" berhasil mengimplementasikan Web Server yang memiliki sifat high availability menggunakan perangkat lunak *failover* Keepalived. Disini, peneliti tidak mencoba untuk menerapkan dengan aplikasi *failover* lain, semisal Heartbeat. Mungkin, jika penulis menggunakan Heartbeat maka hasil pengujian *failover* mendapatkan hasil yang lebih bagus.

Penelitian sebelumnya tentang penerapan *failover* yang dilakukan oleh Irfani dan Hernawan Sulistyanto yang berjudul "Implementasi High Availability Server

dengan Teknik *Failover* Virtual Computer Cluster" berhasil mengimplementasikan sistem *failover virtual computer cluster* menggunakan aplikasi *failover* Heartbeat. Disini, peneliti tidak menerapkan dengan aplikasi *failover* lain, misalnya Keepalived karena tidak mungkin jika hasil pengujian *failover* mendapatkan hasil yang lebih bagus jika menggunakan Keepalived.

Berdasarkan pemaparan dari dua penelitian sebelumnya yang telah disebutkan diatas, maka peneliti akan membuat sebuah sistem untuk membandingkan kinerja dari dua buah software *failover* yang ada, yaitu Heartbeat dan Keepalived. Penelitian ini bertujuan untuk membandingkan kinerja serta mencari tahu perbedaan apa saja dari kedua aplikasi *failover* ini, serta dapat mengetahui kinerja maksimal dari kedua aplikasi *failover* jika aplikasi *failover* tersebut diimplementasikan pada bidang tertentu. Sistem akan dibangun pada sistem operasi Ubuntu Server yang diinstall pada perangkat lunak virtualisasi Virtualbox. Diharapkan hasil akhir dari penelitian dapat membantu pembaca untuk menggunakan aplikasi *failover* yang cocok sesuai dengan kebutuhannya masing-masing.

1.2 Rumusan masalah

Berdasarkan latar belakang yang telah dijelaskan sebelumnya, didapatkan beberapa rumusan masalah yang akan dibahas:

1. Bagaimana pengimplementasian *high availability web server* dan *high availability load balancing* menggunakan *Keepalived* dan *Heartbeat*?
2. Bagaimana perbandingan kinerja teknologi *failover* berbasis klaster (*Heartbeat*) dengan teknologi *failover* berbasis jaringan (*Keepalived*)?

1.3 Tujuan

Tujuan dari penulisan skripsi ini adalah

1. Pengimplementasian *high availability web server* dan *high availability load balancing* menggunakan *Keepalived* dan *Heartbeat*.
2. Menganalisa perbandingan kinerja Heartbeat dan Keepalived dalam pengimplementasian *high availability web server* dan *high availability load balancing*.

1.4 Manfaat

Adapun penelitian skripsi ini diharapkan dapat memberikan manfaat bagi penulis, perguruan tinggi serta pihak lainnya antara lain:

1. Mendapatkan ilmu baru tentang *failover clustering*.
2. Mendapat referensi atau bahan rujukan tentang *failover clustering* yang dapat ditambah ke landasan kepustakaan dipenelitian yang akan datang khususnya di Universitas Brawijaya.

3. Memberikan informasi kepada masyarakat tentang ilmu *failover clustering*, yang nantinya diharapkan *failover clustering* ini dapat diimplementasikan dalam kasus yang nyata.

1.5 Batasan masalah

Batasan masalah dalam penelitian ini adalah:

1. Implementasi *failover clustrering* dilakukan dalam perangkat lunak virtualisasi Virtualbox.
2. Penelitian dilakukan menggunakan Windows 10 sebagai *Operating System Host* dan Linux Ubuntu Server 16.04 LTS sebagai server.
3. Perangkat lunak *failover* yang digunakan untuk implementasi *failover* adalah *Heartbeat* dan *Keepalived*.
4. Perangkat lunak yang digunakan untuk implementasi *web server* adalah *Apache*.
5. Perangkat lunak yang digunakan untuk implementasi load balancing adalah *Haproxy*.

1.6 Sistematika pembahasan

BAB I PENDAHULUAN

Bab ini terdiri dari latar belakang masalah, rumusan masalah, batasan masalah, tujuan, manfaat dan sistematika pembahasan dari

BAB II LANDASAN KEPUSTAKAAN

Bab ini membahas tentang dasar teori yang digunakan dalam penelitian sebagai penunjang penyelesaian penelitian yang akan dilakukan. Dasar teori diambil dari buku, jurnal, e-book atau sumber kredibel lainnya.

BAB III METODOLOGI PENELITIAN

Bab ini membahas langkah-langkah kerja dari penelitian yang akan dilakukan, diantaranya kajian pustaka, analisis kebutuhan simulasi, perancangan simulasi, implementasi proses simulasi, dan pengujian terhadap sistem yang diimplementasi.

BAB IV PERANCANGAN PENGUJIAN

Bab ini membahas tentang pembuatan lingkungan pengujian,

BAB V IMPLEMENTASI

Bab ini membahas tentang implementasi failover untuk kebutuhan web server dan kebutuhan load balancing.

BAB VI PENGUJIAN

Bab ini membahas tentang pengujian *software failover* serta analisis terhadap hasil yang diperoleh dari pengujian.

BAB VII PENUTUP

Bab ini membahas tentang kesimpulan terhadap penelitian yang disusun berdasarkan data serta analisis yang telah dilakukan. Bab ini juga berisi tentang saran untuk pengembangan penelitian selanjutnya.

BAB 2 LANDASAN KEPUSTAKAAN

Bab landasan kepustakaan berisi tentang kajian pustaka dan dasar teori terkait yang dibutuhkan untuk penelitian yang akan dilakukan. Kajian pustaka yang digunakan dalam penelitian yang akan dilakukan ini didasari oleh penelitian terdahulu terkait *High Availability*, Heartbeat, Keepalived dan *Failover*.

2.1 Kajian Pustaka

Pada penelitian pertama dengan judul "Perancangan dan Pengujian Load Balancing dan *Failover* Menggunakan Nginx" yang dilakukan oleh Rahmad Dani pada tahun 2015. Penelitian ini membangun sebuah sistem *web server* yang diberi *load balancing* serta *failover*. Untuk *failover* sendiri menggunakan bantuan aplikasi Keepalived. Pada hasil pengujian ketersediaan layanan, dapat ditarik kesimpulan bahwa sistem *web server* tidak mengalami gangguan ketika ada salah satu *web server* yang mati karena pada sistem yang dibangun telah menerapkan *failover* menggunakan aplikasi *failover* Keepalived. Pada proses pengujian *failover*, dibutuhkan waktu selama 5 detik selama perpindahan dari *server* utama ke *server* backup.

Kemudian penelitian kedua dengan judul "Implementasi High Availability Server dengan Teknik *Failover* Virtual Computer Cluster" yang dilakukan oleh Irfani dan Hernawan Sulistyanto. Penelitian ini membangun sebuah sistem yang memiliki sifat ketersediaan tinggi berupa *failover virtual cluster*. Sistem ini terdiri dari dua (2) buah *server virtual* Ubuntu 14.10. Kedua server menggunakan aplikasi *failover* Heartbeat dan aplikasi DRBD yang digunakan untuk sinkronisasi data. setelah pengujian dapat ditarik kesimpulan bahwa sistem *failover* yang dibangun dapat tetap bekerja dengan baik sehingga user tetap dapat mengakses layanan yang disediakan oleh server.

Pada tabel 2.1 akan dibandingkan tentang penelitian terdahulu dan penelitian yang akan dilakukan.

Tabel 2.1 Kajian Pustaka

NO	Nama Penulis dan Judul	Persamaan	Perbedaan	
			Penelitian Terdahulu	Rencana Penelitian
1.	Dani Rahmad, Perancangan dan Pengujian Load Balancing dan *Failover* menggunakan Nginx	Menggunakan keepalived untuk penerapan *failover*	Membangun sistem *web server* berbasis *High Availability*, kemudian diberikan mekanisme *failover* menggunakan Keepalived kemudian diuji	Menguji dan membandingkan performa antara Heartbeat dan Keepalived untuk sistem *web server failover Clustering*

2.	Irfani, Implementasi High Availability Server dengan Teknik *Failover* Virtual Computer Cluster	Menggunakan heartbeat untuk penerapan *failover*	Membangun server repository berbasis *High Availability,* kemudian diberi mekanisme load balancing dan *failover* kemudian dianalisis	Menguji dan membandingkan performa antara Heartbeat dan Keepalived untuk *failover Clustering*

Berdasarkan penelitian terdahulu yang telah dipaparkan diatas, ada beberapa teori yang akan diulas untuk menganalisis perbandingan performa Heartbeat dan Keepalived untuk *failover*.

2.2 High Availability

High availability dalam teknologi informasi dapat diartikan sebagai sebuah sistem atau komponen yang dapat beroperasi secara terus-menerus dalam jangka waktu yang lama tanpa mengalami gangguan. Pada dasarnya, *high availability* mengimplementasikan satu atau lebih *server* cadangan dalam mode *standby*, yang bisa menjadi siaga dalam beberapa saat ketika ditemukan kegagalan dalam komponen utama (Bookman, 2002).

Implementasi *high availability* klaster pada umumnya untuk tujuan meminimalkan *downtime* serta meningkatkan ketersediaan sehingga sistem yang diciptakan memiliki kehandalan. Masing-masing node dalam klaster saling bertukar informasi tentang keadaannya masing-masing, seperti apakah sedang *online* atau sedang *offline*. Pada teknologi ini, jika terdapat kegagalan dalam satu *node* maka *node* lain akan mengambil alih. Hal ini akan menjamin sistem agar dapat tetap memberikan layanan (Dudnik, 2017).

Salah satu cara untuk mencapai high availability adalah menghilangkan single point of failure. Yang dimaksud dari single point of failure adalah komponen didalam sistem yang dimana jika komponen tersebut mengalami failure, maka keseluruhan sistem akan berhenti beroperasi. Single point of failure dapat dihindari dengan menciptakan sistem redundant (Heidi, 2016).

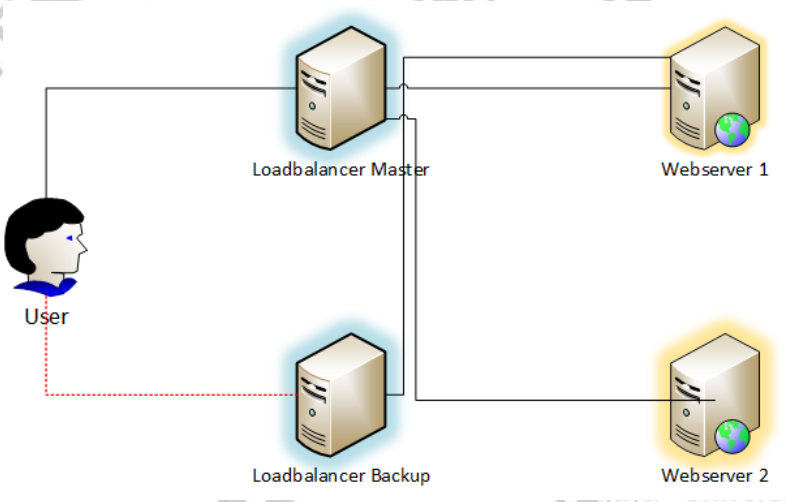
2.3 Failover

Failover merupakan sebuah teknologi untuk menghindari kegagalan pada sistem. Yang dimaksud dari kegagalan adalah berhentinya sebuah komponen dalam sistem sehingga sistem tidak dapat bekerja. Dengan digunakannya teknologi *failover* pada sistem, sistem akan terus berjalan karena failover memindahkan *traffic* dari komponen yang mengalami kegagalan ke komponen cadangan atau *backup*. Proses *failover* dapat dirancang secepat mungkin, setelah saat terjadi kegagalan.

Didalam *failover* terdapat proses *failback*. *Failback* adalah proses kembalinya *node* yang sebelumnya mengalami kegagalan. Sebagai contoh, *node* A telah berjalan normal dari kegagalan yang terjadi sebelumnya, *node* A mengambil alih kembali service dari *node* B.

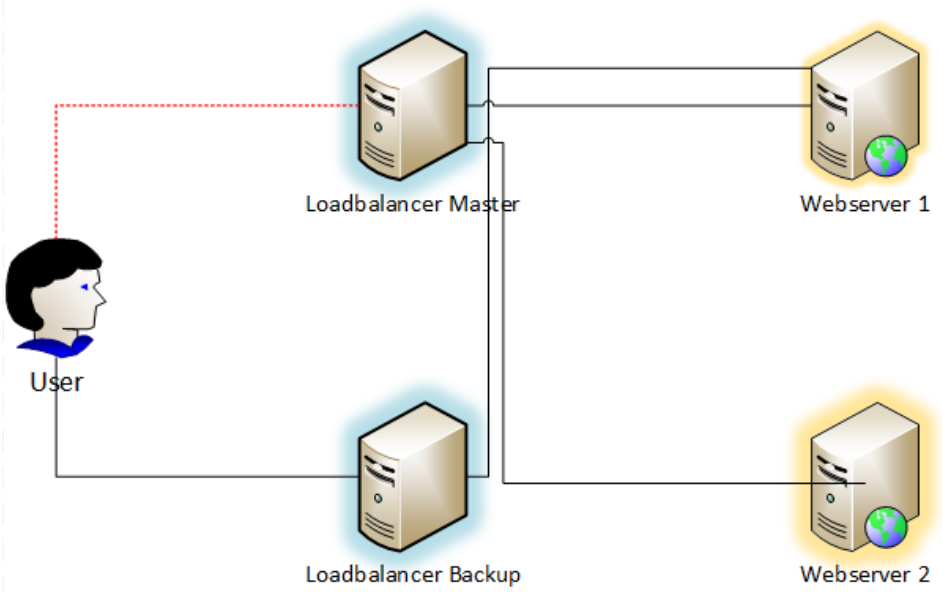
Terdapat dua model dari *failover* yaitu *failover active-active* dan *failover active-standby*. Pada *failover active-active*, semua komponen akan aktif utuk menangani traffic yang masuk. Sedangkan *pada failover active-standby,* kedua komponen tetap tersedia namun hanya komponen utama yang bekerja untuk menangani *request* yang masuk. Jika komponen utama mengagalami kegagalan, maka komponen cadangan akan mengambil alih tugas untuk menangani *request* (Cisco, 2018).

Pada gambar 2.2 menjelaskan sebuah contoh skema sistem saat belum terjadi *failover*. Client melakukan *request* yang ditangani oleh *server load balancer master*, yang bertugas sebagai *server* utama. Kemudian *load balancer master* meneruskan *request* kepada dua *web server* yang terhubung. *Load balancer master* akan terus melayani *request* selama dalam kondisi tidak mengalami sebuah kegagalan.



Gambar 2.1 Sebelum Terjadi Failover

Sedangkan pada gambar 2.3 menjelaskan sebuah contoh skema sistem saat sudah terjadi failover akibat kegagalan load balancer master. Setelah terjadi proses failover akibat kegagalan server load balancer master, server load balancer backup menjadi server yang menangani permintaan pengguna. Server load balancer backup kemudian meneruskan serta membagi beban traffic kepada dua web server yang terhubung. Dengan diterapkannya failover, sistem yang dibangun akan tetap berjalan dan dapat diakses oleh pengguna karena jika terjadi kegagalan pada server utama, server cadangan akan menggantikan tugas server utama.



Gambar 2.2 Setelah Terjadi Failover

2.4 Heartbeat

Aplikasi Heartbeat merupakan sebuah bagian dari proyek Linux *High Availability* (Robertson, 2000). Aplikasi ini dirancang agar dapat tetap berjalan dalam jangka waktu yang lama tanpa mengalami *error.*

Heartbeat adalah sebuah aplikasi *failover* berorientasi klaster, yang memastikan bahwa *resources* akan selalu tersedia untuk diakses. Heartbeat harus dijalankan pada setiap node yang menjadi anggota dalam sebuah klaster. Produk berbasis klaster seperti Heartbeat memastikan bahwa *shared resources* tidak akan *corrupted* oleh akses yang bersamaan. Jika *shared resources* bisa diakses secara bersamaan, maka itu akan memicu *split brain*. Split brain merupakan suatu kejadian dimana suatu resources diakses oleh lebih dari satu klaster dan akan menyebabkan resources tersebut menjadi rusak.

Dengan jangka waktu tertentu, masing-masing node Heartbeat akan mengirimkan *packet* melewati jaringan kepada node Heartbeat lain sebagai sinyal *keepalive*. Jika tidak ada pesan Heartbeat dari *node master* yang diterima oleh node *slave*, dapat diasumsikan bahwa *node master* telah mati kemudian *node slave* menjalankan teknik *failover* serta mengambil alih peran dari *node master*.

Protokol multicast dari *heartbeat* menggunakan variasi dari kedua teknik tersebut. Dalam heartbeat, setiap penerima dilarang meminta transmisi ulang paket lebih dari satu kali per detik, dan pada gilirannya, setiap pengirim akan mengirimkan kembali paket tidak lebih dari satu kali tiap detik. Dengan menggunakan cara ini, pesan flooding dari NACKs dapat dibatasi.

Heartbeat menggunakan algoritma *digital signature* untuk setiap paket yang dikirim atau yang diterima pada tiap-tiap *node*. Heartbeat menggunakan beberapa algoritma digital signature yaitu CRC, MD5, dan SHA1. Masing-masing dari

algoritma tersebut memiliki keunggulan yang berbeda, misalnya algoritma MD5 yang memiliki security tingkat rendah namun kinerjanya ringan sehingga tidak membebani kerja CPU, algoritma sha1 memiliki kemampuan autentikasi terbaik namun memiliki kekurangan kinerja yaitu membutuhkan *load* lebih tinggi dari CPU.

2.4.1 Konfigurasi Heartbeat

Terdapat 3 file yang harus dikonfigurasi sebelum menggunakan aplikasi Heartbeat. Ketiga file tersebut adalah ha.cf, authkeys dan haresources. Berikut penjelasan dari ketiga file:

1. Ha.cf

   File ini memuat konfigurasi dari aplikasi Heartbeat, memuat daftar *node* yang menjadi bagian dari klaster, mengaktifkan atau menonaktifkan fitur dan sebagainya. Jika konfigurasi dari file ha.cf salah, besar kemungkinan aplikasi Heartbeat tidak akan berjalan sebagaimana mestinya. Pada tabel akan dijelaskan petunjuk konfigurasi Heartbeat:

Tabel 2.2 Directive Aplikasi Heartbeat

apiauth	Digunakan untuk menentukan user dan nama grup tertentu yang dapat terhubung ke API.
autojoin	Autojoin memungkinkan sebuah node dapat berkomunikasi dengan klaster tanpa tambahan petunjuk dalam file ha.cf, karena umumnya komunikasi antar node membutuhkan autentikasi. Terdapat 3 value dari autojoin yaitu none, other dan any.
bcast	Bcast digunakan untuk menentukan interface mana yang digunakan Heartbeat untuk mengirimkan traffic broadcast UDP.
compression	Compression digunakan untuk mengatur metode kompresi yang digunakan ketika ada sebua big message dan membutuhkan kompresi. Jika directive ini tidak diatur maka tidak akan ada kompresi.
compression_threshold	Compression_threshold digunakan untuk mengatur kompresi ambang batas pesan. Misal, jika threshold diatur 1 maka pesan apapun yang lebih besar dari 1KB akan dikompress. Sebelum menggunakan directives ini, harus terlebih dahulu menggunakan directives compression.

conn_logd_time	Directive ini digunakan untuk mengatur waktu Heartbeat terhubung kembali ke daemon logging jika koneksi antara Heartbeat dan daemon logging rusak.
deadtime	Deadtime digunakan Heartbeat untuk memutuskan kondisi node dalam keadaan mati. Jika value yang diisikan terlalu rendah maka dapat menyebabkan sistem salah menyatakan keadaan. Jika value yang diisikan terlalu lama kemungkinan menyebabkan delay saat proses *failover*.
initdead	Parameter initdead digunakan untuk mengatur waktu yang diperlukan untuk menyatakan bahwa node mati ketika Heartbeat diaktifkan. Secara umum, parameter initdead diatur ke nilai yang lebih tinggi karena terkadang sistem operasi membutuhkan waktu yang agak lama agar bisa beroperasi dan berkomunikasi dengan benar
keepalive	Digunakan untuk mengatur interval waktu paket Heartbeat.
logfacility	Digunakan untuk memberi informasi kepada heartbeat yang digunakan untuk mencatat pesan Heartbeat.
node	Digunakan untuk memberitahu node mana saja yang tergabung dalam klaster. Nama node harus sesuai dari jawaban yang diberikan sistem operasi saat pengguna memberi input “uname -n”
Udpport	Directives ini mengatur port UDP yang digunakan untuk komunikasi antar klaster. Default udpport aplikasi Heartbeat adalah 694.
auto_failback	Dalam heartbeat versi terdahulu, directive ini menentukan apakah resources yang gagal akan kembali ke node master. Ada 3 value pada directive ini yaitu on, off dan legacy.

2. Authkeys

File authkeys berisi konfigurasi dari jenis autentikasi yang digunakan oleh anggota klaster. Proses komunikasi antar *node* dalam klaster menjadi lebih aman. File authkeys hanya boleh dibaca dan dikonfigurasi oleh hak akses root. Format file ini terdiri dari dua baris, baris pertama yang memuat aturan penggunaan kunci untuk paket keluar dan baris kedua untuk menanda tangani paket yang masuk. Algoritma signature method yang didukung oleh Heartbeat antara lain MD5, CRC dan SHA1.

File authkeys digunakan untuk mengatur jenis autentikasi dan hash dari klaster yang akan digunakan untuk metode autentikasi. Denan adanya file authkeys, proses hubungan antar *node* dalam klaster menjadi lebih aman. *File* ini hanya dapat diedit oleh root. File authkeys berisi dua konfigurasi, yaitu *signature method* dan *keys* untuk menanda tangani paket yang keluar serta *signature method* dan *keys* untuk menanda tangani paket yang masuk.

3. Haresources

File haresources digunakan untuk mengatur sebuah *sources* yang diinginkan menjadi *highly-available* atau selalu tersedia. Didalam file haresources administrator dapat memasukkan daftar *resources* yang akan dikelola oleh heartbeat. Yang dimaksud *resources* disini dapat berupa IPaddr, *mount file system* atau *init script*.

2.5 Keepalived

Keepalived adalah sebuah aplikasi *failover* berbasis jaringan yang dibuat menggunakan bahasa C. Aplikasi *failover* berbasis jaringan seperti Keepalived akan menyediakan alamat IP yang aktif di kedua node, dan menyebabkan layanan akan tersedia pada semua node. Konfigurasi dari aplikasi Keepalived akan disimpan dalam file keepalived.conf. Aplikasi berbasis jaringan seperti Keepalived akan menyediakan alamat IP yang aktif di kedua node, sehingga layanan akan tersedia pada semua node.

Aplikasi keepalived menyediakan dua *frameworks*, yaitu load-balancing dan *high-availability*. Framework *load balancing* keepalived mengandalkan modul kernel dari *Linux Virtual Server* (LPVS). Modul kernel LPVS menyediakan *load balancing* pada *layer* 4 (*layer transport*). Untuk memelihara dan mengelola health dari server pool, keepalived mengimplementasikan *health checker*. Sedangkan *high-availability* dapat tercapai karena keepalived mengimplementasikan Virtual Redundancy Routing Protocol (VRRP). VRRP merupakan dasar dari *router failover*(Cassen, 2017). Dua frameworks dari keepalived dapat digunakan sendiri atau saling bersamaan untuk menciptakan sebuah sistem yang tangguh.

Keepalived diharapkan untuk menjadi aplikasi yang kokok dan memiliki stabilitas. Untuk memastikan hal tersebut, maka daemon dibagi menjadi 3 proses:

1. Parent process akan bertanggung jawab dari *child process*.
2. Dari dua *child process*, satu bertanggung jawab untuk VRRP *frameworks*.
3. Sedangkan satunya bertanggung jawab untuk *healtchecking*.

Tiap children process memiliki scheduling I/O multiplexer masing-masing, sehingga VRRP scheduling dapat dioptimalkan.

Parent process memiliki *framework* yang disebut *watchdog*. Cara kerja framework ini jika ada sebuah *children process* yang membuka dan menerima soket domain unix, maka parent process akan ikut terhubung pada soket domain

unix dan mengirim *hello packet* secara periodik tiap 5 detik kepada *children*. Jika parent process tidak dapat mengirim hello packet ke soket domain unix yang sedang terhubung, maka *parent process* akan memberikan perintah kepada *children* untuk melakukan *restart*.

Cara kerja watchdog seperti itu memberikan dua keuntungan. Yang pertama, semua hello packet yang dikirim dari *parent process* menuju *children process* melewait I/O *multiplexer schedule* sehingga dapat mendeteksi adanya deadloop pada *children scheduling network*. Manfaat yang kedua karena penggunaan sinyal sys V dapat mendeteksi *children* yang mati.

2.5.2 Komponen Kernel

Aplikasi keepalived menggunakan empat komponen kernel dari Linux. Komponen tersebut akan dijelaskan pada tabel dibawah:

Tabel 2.3 Komponen Kernel yang digunakan Keepalived

Nomor	Komponen	Keterangan
1	LVS Framework	Keepalived menggunakan kernel LVS Framework karena membutuhkan fitur getsockopt dan setsockopt untuk mengatur opsi pada soket.
2	Netfilter Framework	Netfilter adalah sebuah framework di linux yang menyediakan cara untuk memanipulasi paket-paket via kernel. Keepalived membutuhkan kernel ini karena *netfilter framework* mendukung NAT dan *Masquerading*.
3	Netlink Interface	Keepalived menggunakan kernel ini untuk menambah dan menghapus IP *virtual* VRRP pada *network interfaces*.
4	Multicast	Keepalived menggunakan kernel multicast untuk mengirimkan *VRRP advertisement* kepada *VRRP multicast group*.

2.6 Ubuntu

Ubuntu adalah sebuah operating system yang bersifat *Open Source* dan dapat dijalankan di *desktop*, *cloud* atau IoT (Ubuntu, 2018). Ubuntu merupakan salah satu distribusi dari linux yang berbasis dari Debian. Ubuntu tersedia secara bebas dan dapat diunduh di www.ubuntu.com. Ubuntu dirilis oleh Canonical Ltd,sebuah perusahaan swasta dalam bidang perangkat lunak yang berpusat di *United Kingdom*. Setiap rilis versi ubuntu, diberi *code name* contohnya: Ubuntu 14.04 LTS memiliki code name Trusty Tahr.

Ubuntu yang digunakan didalam penelitian ini adalah Ubuntu Server 16.04. LTS 64 bit yang memiliki *code name* Xenial Xerus. Ubuntu server 18.04.3 LTS dirilis dalam dua versi, yaitu versi 32 bit dan 64 bit. Sistem operasi Ubuntu Server dipilih dalam pelaksanaan implementasi karena memiliki *system requirements* yang lebih rendah dibandingkan Ubuntu Desktop. Berikut ini perbandingan *system requirements* dari Ubuntu Server dengan Ubuntu Desktop:

Tabel 2.4 Perbandingan System Requirements Ubuntu Server dan Ubuntu Desktop

Nomor	Ubuntu Server	Ubuntu Desktop
Processor	300 MHz x86 processor	2 GHz dual core processor
Ram	256 MB	2 GB RAM (Memory System)
Harddisk	1.5 GB	25 GB
Graphic Card dan Monitor	Kartu Grafik serta Monitor yang mampu menampilkan resolusi 640*480	Kartu Grafik serta monitor yang mampu menampilkan resolusi 1024*768

2.7 Virtualbox

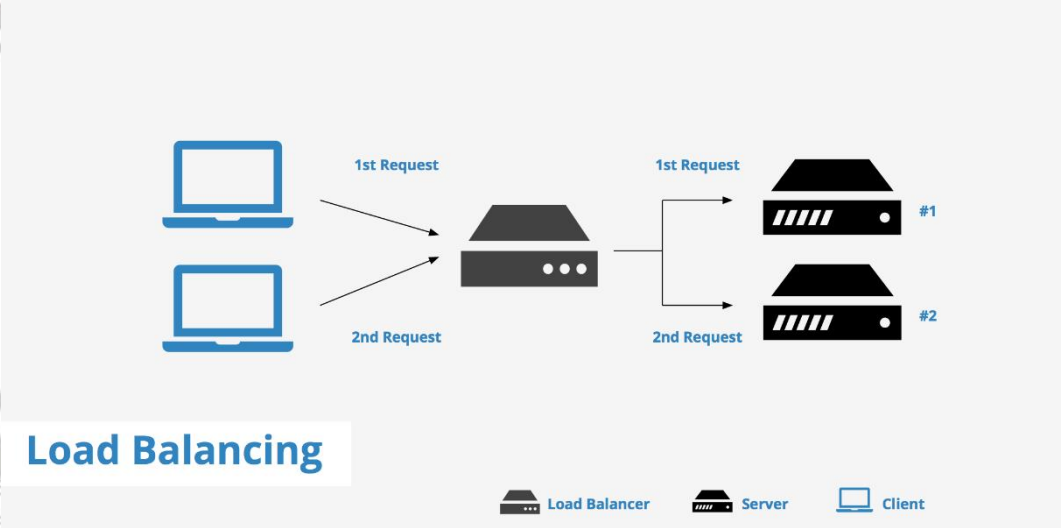
Virtualbox adalah perangkat lunak virtualisasi untuk kebutuhan *enterprise* ataupun pribadi (Virtualbox, 2018). Aplikasi ini dapat dijalankan di Windows, Macintosh, Solaris serta Linux. Virtualbox memudahkan penggunanya untuk dapat menggunakan beberapa sistem operasi dalam satu komputer, serta dapat juga untuk menjalankan simulasi topologi jaringan. Sistem operasi yang didukung oleh Virtualbox antara lain Linux, Microsoft Windows, BSD, Solaris, Mac OS X dan lain sebagainya.

Dalam penelitian ini, versi virtualbox yang digunakan adalah Virtualbox 6.1.16. perangkat lunak Virtualbox memiliki beberapa fitur yang berguna dalam implementasi *failover*, seperti *clone virtual machine*, *network adapter, importing and exporting virtual machine.* Fitur c*lone virtual machine* berguna untuk melakukan *clone* os Ubuntu Server yang menjadi *virtual machine* saat implementasi *failover*, karena dapat menghemat waktu daripada melakukan proses instalasi sistem operasi dari awal saat dibutuhkan sebuah *virtual machine* yang baru. Fitur *network adapter* berguna agar masing masing *virtual machine* dapat saling terhubung dan dapat juga mengakses internet. Virtualbox memiliki beberapa opsi *network adapter* seperti NAT dan *Host-only adapter*.

2.8 Load Balancing

Load balancing merupakan sebuah teknik untuk membagi jumlah pekerjaan yang dilakukan oleh dua buah komputer atau lebih sehingga setiap komputer melakukan pekerjaan yang sama dan merata tergantung dari algoritma yang digunakan sehingga pengguna dapat mengakses lebih cepat. Tujuan dari adanya *load balancing* untuk meminimalkan waktu respon, memaksimalkan *throughput*, mengoptimalkan sumber daya dan menghindari dari kelebihan beban sumber daya tunggal (Lakhe, Shinde, Sukhthankar, & Reddy, 2016).

Perangkat lunak yang berguna untuk mengimplementasikan *load balancing* disebut *load balancer*. Permintaan akses dari user akan merata ke semua server yang ada sehingga beban menjadi seimbang serta kinerja server dapat menjadi lebih baik. Disaat load balancer menerima *request* dari pengguna, maka load balancer akan meneruskan request tersebut ke server yang ada, dan akan dibagi tergantung dari algoritma apa yang dipilih. Ada beberapa algoritma yang dapat digunakan dalam perangkat lunak Haproxy yaitu *roundrobin*, *leastconn* dan *source*.



Gambar 2.3 Load Balancing

2.9 Haproxy

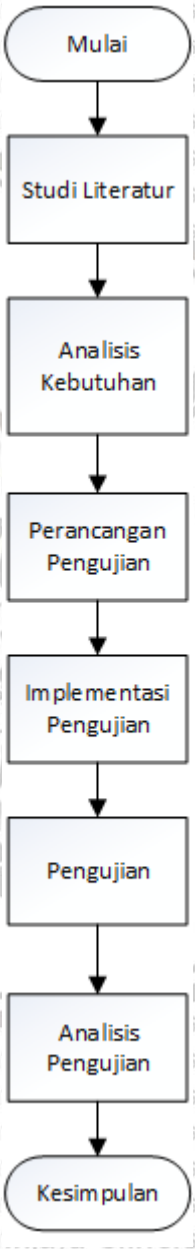
Haproxy merupakan sebuah aplikasi yang bersifat *opensource* berbasis linux yang umumnya digunakan untuk implementasi *load balancing*.(Haproxy, 2021). Selain digunakan untuk load balancing, Haproxy juga dapat digunakan untuk proxy. Pada umumnya load balancing dikelompokkan menjadi dua kategori yaitu Layer 4 dan Layer 7. Layer 4 bertindak pada data di network TCP, FTP, UDP, IP sedangkan layer 7 mendistribusikan permintaan dari pengguna berdasarkan dada yang ditemukan dalam layer Application contohnya HTTP.

2.10 LAMP Stack

LAMP stack merupakan akronim dari Linux, Apache Web Server, MySql dan PHP (DigitalOcean, 2018). LAMP merupakan sebuah paket perangkat lunak *opensource* yang dapat digunakan untuk menjalankan sebuah aplikasi. Linux merupakan sebuah sistem operasi yang bersifat *opensource*. Apache merupakan web server yang bersifat *opensource* dan dapat dijalankan dibanyak sistem operasi seperti windows, linux dan platform lainnya. MySql merupakan *database* server *opensource* yang dapat digunakan untuk membangun web. PHP merupakan akronim dari *Hypertext Preprocessor* yang merupakan sebuah bahasa pemrograman *opensource* yang bersifat server-side sehingga *script* PHP akan diproses pada server.

BAB 3 METODOLOGI PENELITIAN

Pada bab metodologi penelitian akan dijelaskan langkah-langkah yang akan dilakukan untuk melaksanakan penelitian, yaitu studi literatur, analisis kebutuhan, perancangan sistem, implementasi, pengujian, analisis pengujian dan kesimpulan. Gambar 3.1 merupakan diagram alir langkah-langkah metode penelitian yang akan dilakukan



Gambar 3.1 Diagram Alir Penelitian

Berikut ini adalah penjelasan dari gambar diagram alir penelitian:

1. Mulai.

2. Melakukan studi literatur terhadap topik-topik yang berkaitan dengan topik permasalahan yang dibahas.
3. Menganalisis kebutuhan pengujian sistem untuk perancangan lingkungan pengujian *failover* untuk kebutuhan *web server*dan *load balancing*.
4. Menganalisis perancangan pengujian sistem *failover* untuk kebutuhan *web server* dan *load balancing.*
5. Mengimplementasi lingkungan pengujian sistem *failover* sesuai dengan rancangan pengujian sistem *failover* yang telah dibuat.
6. Melakukan pengujian terhadap implementasi *failover* yang telah dibuat.
7. Melakukan analisis terhadap hasil pengujian sistem *failover*.
8. Membuat kesimpulan dari hasil analisis serta saran untuk penelitian selanjutnya.
9. Selesai.

3.1 Studi Literatur

Pada bagian studi literatur membahas tentang dasar teori yang digunakan untuk menunjang pengerjaan penelitian. Studi literatur yang dibutuhkan diambil dari jurnal, e-book, buku, penelitan yang telah dilakukan dan lain sebagainya.

3.2 Analisis Kebutuhan

Pada tahap analisis kebutuhan bertujuan untuk menganalisis kebutuhan apa saja yang dibutuhkan untuk implementasi pengujian pada penelitian ini, sehingga diharapkan dengan adanya tahapan ini dapat menunjang proses implementasi dan pengujian yang akan dilakukan. Analisis kebutuhan juga dapat mencegah penggunaan *resources* yang tidak perlu dalam implementasi. Untuk melakukan tahapan ini menggunakan cara membaca referensi kepustakaan yang memuat penelitian sebelumnya yang sejenis sehingga didapatkan kebutuhan *resource* apa saja untuk melakukan penelitian ini.

Berdasarkan kebutuhan, maka perangkat yang digunakan akan dikelompokkan menjadi dua yaitu :

1. Kebutuhan Perangkat Keras (*Hardware*). *Hardware* yang dibutuhkan terdiri dari beberapa komponen yang mendukung proses implementasi dan pengujian sistem. Berikut komponen yang digunakan dalam penelitian ini

Tabel 3.1 Kebutuhan Perangkat Keras

No	Perangkat Keras	Spesifikasi
1.	CPU	Intel Core i7-4510U
2.	RAM	8GB

3.	HARDDISK	1TB
4.	SSD	256GB
5.	VGA Card	Nvidia Geforce 840M 2GB

2. Kebutuhan Perangkat Lunak (*Software*). Perangkat lunak yang digunakan dalam implementasi ini bersifat *free* dan *open source*, kecuali sistem operasi Windows 10 . Kebutuhan perangkat lunak akan dijelaskan pada Tabel 3.2

Tabel 3.2 Perangkat Lunak yang Digunakan

No	Perangkat Lunak	*Minimum Requirements*	*Requirements* yang digunakan	Fungsi
1.	Windows 10	-Processor 1.00 GHz -1GB RAM -32GB for 64Bit OS -Display 800*600	-Processor Dualcore 2.00 GHz -8GB RAM -256 GB HDD -Display 1280*720p	Sistem operasi yang digunakan untuk *host*
2.	Ubuntu Server 16.04	-300 MHz x 86 Processor -256MB RAM -1.5 GB HDD -Monitor 640*480	-2.00 GHZ Processor -512 MB RAM -6 GB HDD -Monitor 1280*720p	Sistem operasi yang digunakan untuk mesin virtual
3.	Heartbeat	-dapat diinstall di Ubuntu Server	-diinstall di Ubuntu Server	Perangkat lunak *failover*
4.	Keepalived	-dapat diinstall di Ubuntu Server	-diinstall di Ubuntu Server	Perangkat lunak *failover*
5.	Apache2	-dapat diinstall di Ubuntu Server	-diinstall di Ubuntu Server	Perangkat lunak *web server*

6.	MySql	-dapat diinstall di Ubuntu Server	-diinstall di Ubuntu Server	Perangkat lunak pengelola *database*
7.	HaProxy	-dapat diinstall di Ubuntu Server	-diinstall di Ubuntu Server	Perangkat lunak pengelola *load balancer*

3.3 Perancangan Pengujian

Perancangan pengujian dilakukan setelah tahapan analisis kebutuhan telah terpenuhi dan sesuai dengan kebutuhan yang digunakan untuk penelitian. Perancangan pengujian berisi tentang rancangan bagaimana sistem pengujian akan dibangun. Perancangan pengujian mempunyai tujuan untuk mempermudah dalam langkah implementasi dan pengujian sistem topologi *high availability web server* dan *high availability load balancing*.

3.4 Implementasi Pengujian

Tahap ini adalah implementasi rancangan yang telah dibuat sebelumnya pada tahap perancangan sistem. Tahap ini berisi langkah-langkah tentang implementasi dari rancangan penelitian. Implementasi yang dilakukan adalah implementasi perangkat lunak.

Pada bab implementasi pengujian akan dijelaskan tentang perancangan sistem hingga implementasi sistem. Implementasi dilakukan mengacu pada tahapan perancangan sistem yang telah dibuat terlebih dahulu. Implementasi pengujian meliputi:

1. Implementasi topologi sistem *failover* untuk *High Availability Web Server*.

   Implementasi ini mencakup proses konfigurasi sistem operasi Ubuntu Server, implementasi dan konfigurasi Apache2 *web server*, implementasi dan konfigurasi perangkat lunak *failover Heartbeat* dan *Keepalived*.
2. Implementasi topologi sistem *failover* untuk H*igh Availability Load Balancing*.

   Implementasi ini mencakup proses konfigurasi sistem operasi Ubuntu Server, implementasi dan konfigurasi Apache2 *web server*, implementasi dan konfigurasi perangkat lunak *failover* Heartbeat dan Keepalived serta implementasi dan konfigurasi perangkat lunak *load balancer Haproxy*.

3.5 Pengujian

Tahapan pengujian dilakukan setelah tahapan implementasi pengujian selesai dilaksanakan. Tahapan ini berisi tentang pengujian-pengujian yang akan dilakukan pada sistem yang telah diimplementasikan. Pengujian dilakukan untuk

mengetahui aplikasi *failover* mana yang memiliki kemampuan paling baik dalam masing-masing implementasi. Setelah pengujian selesai dilakukan, akan dilakukan analisis terhadap hasil dari pengujian. Dalam pengujian ini terdapat 3 skenario pengujian, yaitu pengujian fungsional, pengujian *downtime* dan pengujian *failback*.

3.5.1 Pengujian Fungsional

Pengujian fungsional dilakukan dengan tujuan untuk mengetahui konfigurasi dan implementasi yang telah dilakukan pada masing-masing topologi telah berfungsi sebagaimana mestinya.

3.5.2 Pengujian Downtime

Pengujian *downtime* dilakukan dengan tujuan untuk jeda waktu sementara Ketika server tidak dapat menangani *request client* karena adanya kerusakan (*failure*).

3.5.3 Pengujian Failback

Pengujian *failback* dilakukan dengan tujuan untuk jeda waktu sementara ketika server tidak dapat menangani *request client* karena ada perpindahan *traffic* dari *server backup* ke *server master*.

3.6 Analisis Pengujian

Tahapan analisis pengujian membahas tentang hasil pengujian yang telah dilakukan. Hasil pengujian tersebut akan dianalisis dan diharapkan hasil dari analisis tersebut dapat menunjukkan aplikasi mana yang mempunyai performa dan keunggulan terbaik dalam masing masing topologi implementasi sistem *high availability*.

3.7 Kesimpulan dan Saran

Kesimpulan dan saran merupakan tahapan akhir dari proses penelitian. Tahapan ini dilakukan setelah semua tahapan dari proses perancangan dan pengujian telah berhasil dilakukan. Kesimpulan diambil dari hasil pengujian sistem terhadap sistem yang berhasil di implementasikan. Saran dimaksudkan untuk memberi masukkan terhadap berbagai macam kekurangan yang ada didalam implementasi untuk dipertimbangkan dalam melaksanakan penelitian selanjutnya.

BAB 4 PERANCANGAN PENGUJIAN

Pada bab perancangan pengujian akan dijelaskan lebih detail tentang langkah-langkah perancangan dan implementasi sistem *failover* menggunakan *Heartbeat* dan *Keepalived* pada *Ubuntu Server* 16.04. Beberapa hal yang menjadi bahasan pada bab ini adalah analisa kebutuhan dan rancangan sistem.

4.1 Analisis Kebutuhan

Analisis kebutuhan digunakan untuk membahas komponen-komponen yang dibutuhkan dalam implementasi sistem. Analisis kebutuhan menjelaskan komponen-komponen yang dibutuhkan secara fungsional dan non fungsional. Dengan adanya analisis kebutuhan, dapat mempermudah proses melakukan perancangan.

4.1.1 Kebutuhan Fungsional

Kebutuhan fungsional berisi tugas-tugas yang dapat dilakukan oleh sistem yang akan dibangun. Dengan adanya kebutuhan fungsional dapat memastikan sistem yang telah dibangun telah berjalan dengan baik.

4.1.1.1 Kebutuhan Fungsional Topologi High Availability Web Server

1. Masing-masing mesin virtual Web Server yang telah diimplementasi mampu memberikan layanan *website* yang memuat konten Wordpress kepada pengguna.
2. Ketika *Web Server* 1 dalam kondisi mati, pengguna masih dapat mengakses layanan *website* karena adanya implementasi *failover* menggunakan perangkat lunak Heartbeat maupun Keepalived yang mampu mengalihkan *traffic* ke *Web Server 2*.
3. Ketika semua *Web Server* dalam kondisi mati, pengguna tidak dapat mengakses layanan website karena tidak ada *server* yang bekerja untuk menyediakan layanan.

4.1.1.2 Kebutuhan Fungsional Topologi High Availability Load Balancing

1. Masing-masing *server virtual* yang bertindak sebagai *Web Server* dapat memberikan layanan *website* yang memuat konten Wordpress kepada pengguna.
2. Server Loadbalancer1 dan Loadbalancer2 yang telah diimplementasikan Haproxy, mampu mendistribusikan beban traffic

dengan algoritma round robin kepada dua *server virtual* Web Server, yaitu Web Server 1 dan Web Server 2.

3. Ketika server Loadbalancer1 dalam kondisi mati, pengguna masih dapat mengakses layanan *website,* karena pada server Loadbalancer1 dan Loadbalancer2 telah diimplementasikan *failover* menggunakan perangkat lunak *failover* Heartbeat dan Keepalived yang mampu mengalihkan *traffic* menuju *server backup* Loadbalancer2 jika server master Loadbalancer1 sedang tidak aktif.
4. Ketika server Loadbalancer1 dan Loadbalancer2 dalam kondisi mati, *client* tidak dapat mengakses layanan , karena tidak ada server *Loadbalancer* yang aktif untuk melayani *request client*.

4.1.2 Kebutuhan Non Fungsional

4.1.2.1 Kebutuhan Hardware

Agar sistem yang diimplementasi hasilnya dapat sesuai dengan tujuan, maka spesifikasi perangkat keras yang digunakan harus mencukupi. Sistem *failover* akan diimplementasikan pada satu buah laptop. Spesifikasi laptop akan dijelaskan pada Tabel 4.1.

Tabel 4.1 Spesifikasi Perangkat Keras Laptop

No.	Deskripsi	Spesifikasi
1	CPU	Intel Core i7 4510u @2.0 GHz
2	RAM	8 GB
3	Sistem Operasi	Windows 10 64-bit
4	SSD	256 GB
5	*Harddisk*	1 TB
6	*Graphic Card*	NVIDIA GT 840M

4.1.2.2 Kebutuhan Software

Perangkat lunak yang akan digunakan dalam penelitian ini dijelaskan pada Tabel 4.2.

Tabel 4.2 Kebutuhan Non-fungsional Perangkat Lunak

No.	Nama	Deskripsi
1.	Virtualbox	Perangkat lunak untuk virtualisasi

2.	Ubuntu Server 16.04	Sistem operasi yang digunakan untuk implementasi *failover*
2	Heartbeat	Perangkat lunak *failover*
3	Keepalived	Perangkat lunak *failover*
4	Haproxy	Perangkat lunak untuk implementasi *load balancer*
5	Apache2	Perangkat lunak untuk implementasi *web server*

4.1.2.3 Kebutuhan Server Virtual Machine

Untuk membuat *virtual machine* yang jumlahnya lebih dari 1 pada tiap topologi, peneliti m enggunakan fitur *cloning* dari aplikasi Virtualbox sehingga tidak perlu melakukan instalasi beserta konfigurasi sisem operasi Ubuntu Server dari awal ketika dibutuhkan *virtual machine* baru. Pada tabel 4.3 akan dijelaskan spesifikasi dari *virtual machine*.

Tabel 4.3 Spesifikasi Virtual Machine

No	Deskripsi	Spesifikasi
1	Memory	512 MB
2	Processor	1 CPU
3	*Harddisk*	6 GB

4.2 Perancangan Topologi Implementasi *Failover*

Perancangan topologi implementasi *failover* digunakan untuk mempermudah menggambarkan topologi sistem yang akan diimplementasikan. Ada dua (2) buah topologi yang akan dirancang, masing masing topologi akan mengimplementasikan sistem *failover* untuk kebutuhan yang berbeda.

4.2.1 Perancangan Topologi High Availability Web Server

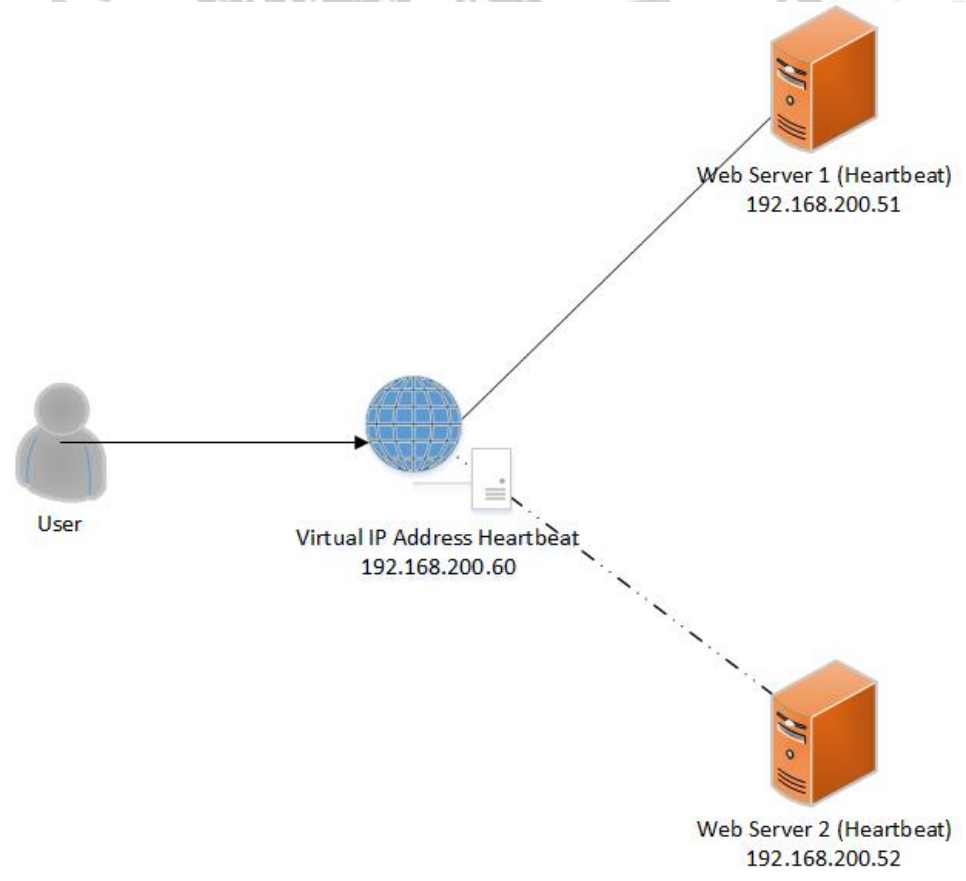
Pada perancangan topologi ini akan mengimplementasikan *failover* untuk kebutuhan *web server*. Perancangan topologi ini secara keseluruhan menggunakan 2 (dua) buah *virtual machine*. *Virtual machine* tersebut terdiri dari *Web Server* 1 dan *Web Server* 2 yang keduanya menggunakan sistem operasi Ubuntu Server versi 16.04.

Agar layanan *web server* tersedia, digunakan perangkat lunak Apache2 *web server*. Yang dilakukan pertama ialah *client* melakukan *request* ke alamat IP *virtual*. *Request* yang masuk di alamat IP *virtual* tersebut akan diterima oleh *Web Server* 1 yang berperan sebagai *server master* dalam klaster. Selanjutnya, *Web Server* 1 memproses *request* tersebut dan mengirim jawaban dari *request* yang diminta

oleh *client* sebagai respon. Jika pada saat pengguna mengirimkan *request* namun kondisi *Web Server* 1 sedang dalam keadaan tidak tersedia, aplikasi *failover* akan memindahkan *traffic* permintaan *request* tersebut kepada *server backup* yang ada dalam klaster, dalam hal ini *Web Server* 2 sehingga *Web Server* 2 untuk sementara akan menggantikan peran tugas *Web Server* 1 sebagai *server master* dalam klaster. Ketika *server* master dalam klaster sedang aktif, server yang bertindak *backup* akan tetap aktif namun dalam kondisi *standby* untuk *memonitoring server master* hingga dalam kondisi tidak tersedia.

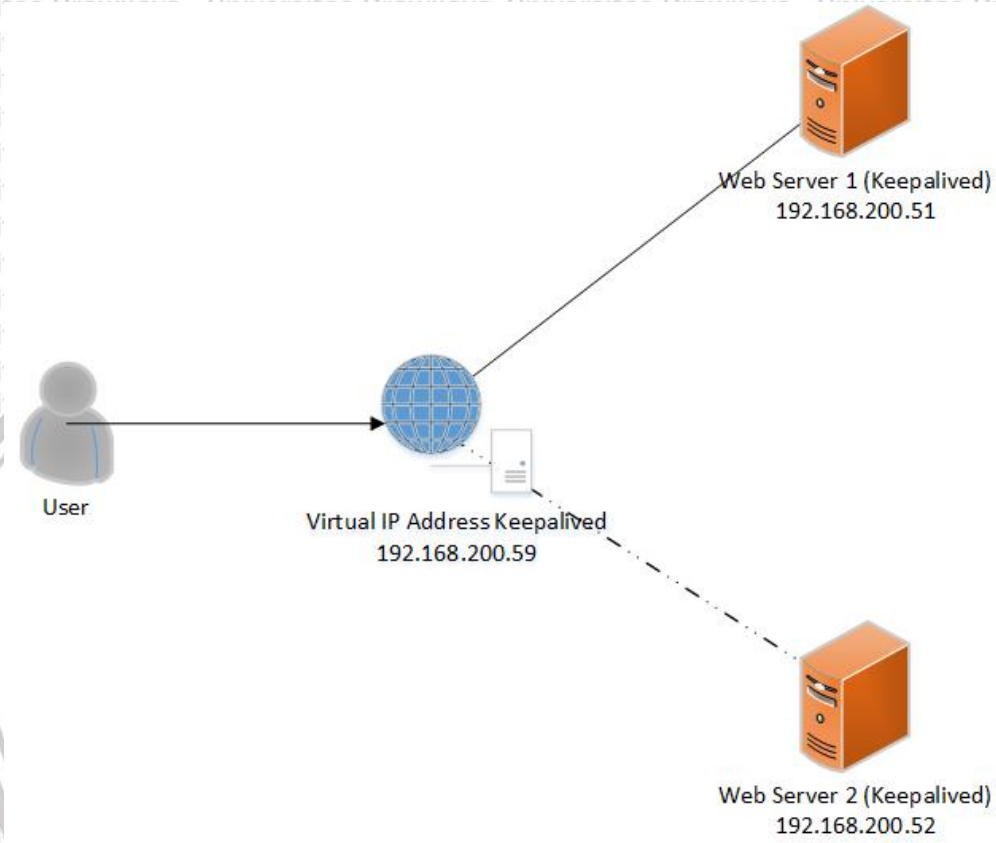
Perancangan topologi ini terdiri dari dua perancangan. Pada perancangan pertama akan menggunakan perangkat lunak *failover* Heartbeat dan pada perancangan yang kedua menggunakan perangkat lunak *failover* Keepalived. Berikut adalah penjelasan dari masing masing perancangan :

1. Pada perancangan pertama sistem *failover* akan diterapkan dengan bantuan perangkat lunak Heartbeat. *Web Server* 1 akan menjadi *server* yang bertindak menjadi *master* dalam klaster, sedangkan *Web Server* 2 akan menjadi *server* yang berperan menjadi *backup* dalam klaster. *Web Server* 1 diberi alamat IP 192.168.200.51 dan *Web Server* 2 diberi alamat IP 192.168.200.52. Virtual *IP address* berada pada 192.168.200.60. Untuk lebih jelas, perancangan ini akan digambarkan pada Gambar 4.1



Gambar 4.1 Topologi Sistem Failover Web Server Menggunakan Heartbeat

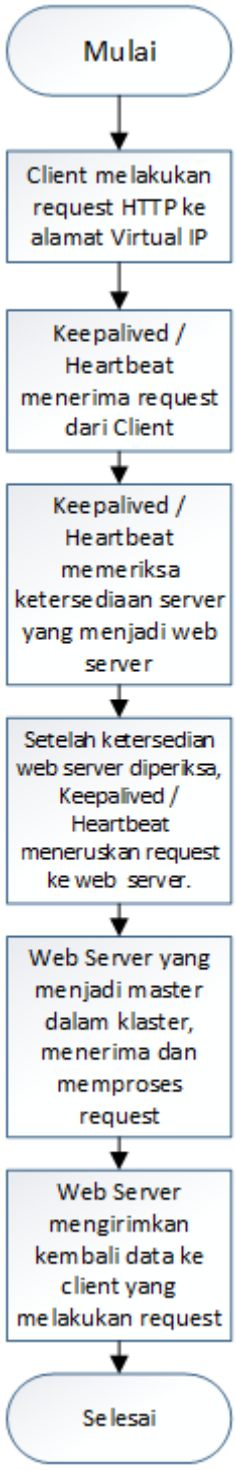
2. Pada perancangan kedua, sistem *failover* akan diterapkan dengan bantuan perangkat lunak Keepalived. *Web Server* 1 akan menjadi *server* yang bertindak menjadi *master* dalam klaster, sedangkan *Web Server* 2 akan menjadi *server* yang berperan menjadi *backup* dalam klaster. *Web Server* 1 diberi alamat IP 192.168.200.51 dan *Web Server* 2 diberi alamat IP 192.168.200.52. Virtual *IP address* berada pada 192.168.200.59. Untuk lebih jelas, perancangan ini akan digambarkan pada Gambar 4.2.



Gambar 4.2 Topologi Sistem Failover Web Server Menggunakan Keepalived

4.2.1.1 Alur Kerja Sistem

Pada Gambar 4.3 dijelaskan secara bertahap terkait alur kerja system yang dimana dengan dimulainya proses *client* mengakses *web*. Agar dapat mengakses *web*, *client* akan melakukan *request* HTTP ke alamat IP *virtual*. Kemudian aplikasi *failover*, Heartbeat atau Keepalived akan menerima *request* kemudian memeriksa kondisi dari *web server*. Selanjutnya, aplikasi *failover* akan meneruskan *request* HTTP tersebut kepada *web server* yang bertindak sebagai *server master* pada kondisi saat itu. *Web server* yang bertindak sebagai *server master* akan menerima *request* dari *client*, kemudian memprosesnya. Setelah *request* diproses, *web server* akan mengirimkannya kembali kepada *client* sebagai respon.

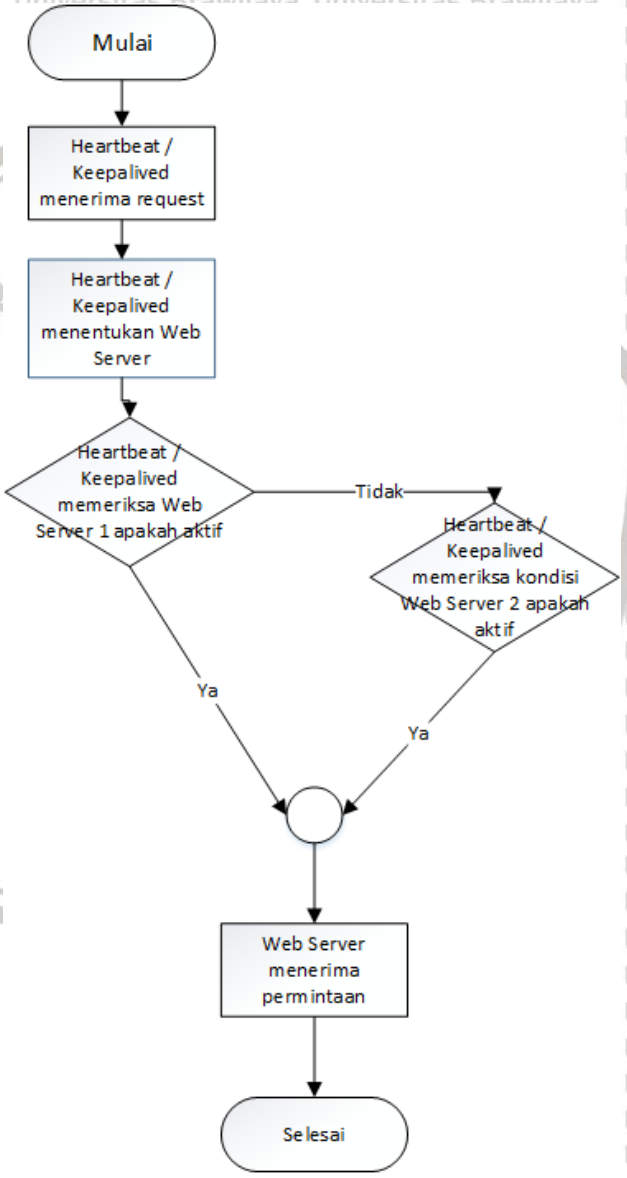


Gambar 4.3 Alur Kerja Sistem Topologi Failover untuk Web Server

4.2.1.2 Alur Kerja Failover

Pada Gambar 4.4 menjelaskan bagaimana cara kerja *failover* untuk kebutuhan *web server*. Heartbeat dan Keepalived masing-masing akan memiliki alamat IP

virtual. Untuk dapat mengakses *web*, pengguna akan mengakses virtual IP. Sehingga request dari pengguna akan masuk pada perangkat lunak *failover*. Perangkat lunak *failover* selanjutnya akan menentukan *Web Server* mana yang akan menerima *request* dari pengguna. Jika *Web Server* 1 yang bertindak sebagai *server master* sedang dalam kondisi aktif, maka *request* akan diteruskan menuju *Web Server* 1. Namun jika *Web Server* 1 sedang dalam kondisi tidak aktif, maka perangkat lunak failover akan meneruskan request dari pengguna kepada server yang berperan sebagai backup dalam klaster, yaitu *Web Server* 2. Jika semua *Web Server* dalam klaster sedang aktif, maka Web Server 2 hanya bertugas me*monitoring* hingga *Web Server* 1 menjadi tidak aktif.



Gambar 4.4 Alur Kerja Failover untuk Web Server

4.2.2 Perancangan Topologi High Availability Load Balancing

Topologi High Availability Load Balancing yang dimplementasikan secara keseluruhan terdiri dari empat (4) buah *virtual machine* antara lainLoadbalancer1, Loadbalancer2, Web Server 1 dan Web Server 2. Mesin virtual Loadbalancer1 dan Loadbalancer2 bertugas untuk meneruskan *request* yang dilakukan pengguna. Selain itu, Loadbalancer1 dan Loadbalancer2 juga bertugas membagi beban *traffic* kepada Web Server 1 dan Web Server 2.

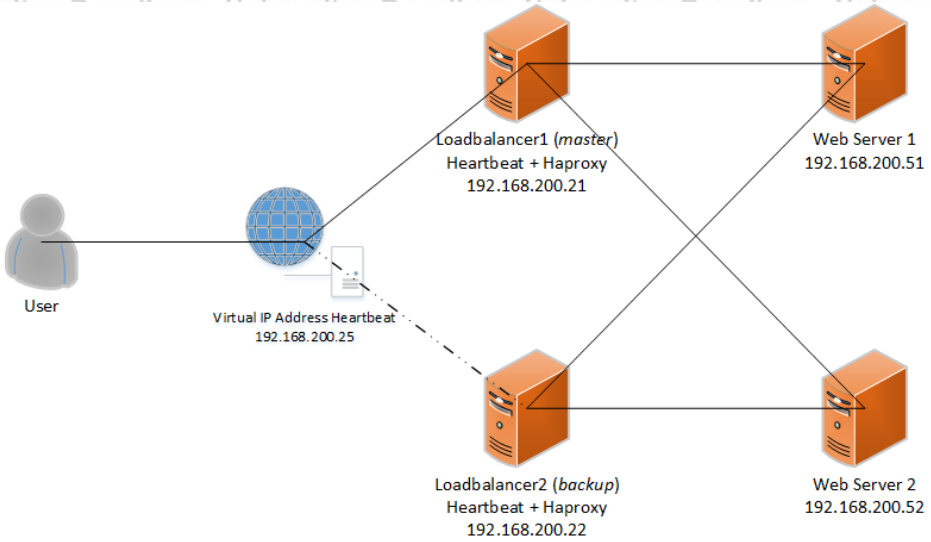
Untuk dapat mengimplementasikan *load balancing*, pada Loadbalancer1 dan Loadbalancer2 menggunakan perangkat lunak Haproxy. Selain mengimplementasikan *load balancing*, pada Loadbalancer1 dan Loadbalancer juga menerapkan *failover* dengan perangkat lunak Heartbeat dan Keepalived. Sehingga jika Loadbalancer1 mengalami kegagalan, maka Loadbalancer2 akan mengambil alih tugas dari Loadbalancer1. Pada Web Server 1 dan Web Server diimplementasikan *web server* menggunakan perangkat lunak Apache2 .

Yang dilakukan pertama adalah *client* melakukan *request* ke alamat *virtual* IP. *Request* tersebut diterima oleh Loadbalancer1 yang berperan sebagai *server master* dalam klaster. Selanjutnya Loadbalancer1 mendistribusikan *request* tersebut pada *virtual machine* Web Server 1 dan Web Server 2. Kemudian Web Server 1 atau Web Server 2 menerima *request* tersebut, kemudian mengirimkan jawaban kepada pengguna.

Jika Loadbalancer1 dalam keadaan tidak tersedia, pengguna masih dapat mengirimkan *request* karena perangkat lunak *failover* memindahkan *traffic* yang masuk menuju *server backup* Loadbalancer2. Sehingga untuk sementara Loadbalancer2 akan menggantikan tugas Loadbalancer1.

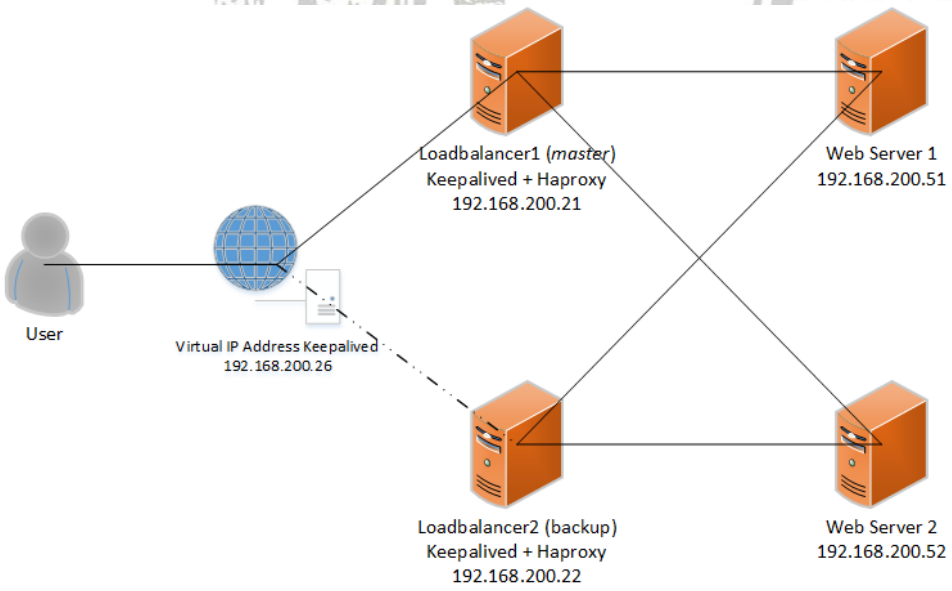
Perancangan topologi *high availability load balancing* akan terdiri dari dua perancangan, yaitu perancangan dengan perangkat lunak Heartbeat dan perancangan dengan perangkat lunak Keepalived. Masing masing perancangan akan dijelaskan dibawah :

1. Pada perancangan pertama,*failover* akan diterapkan dengan bantuan perangkat lunak Heartbeat. Loadbalancer1 akan menjadi *load balancer* utama (*master*) sedangkan Loadbalancer2 akan menjadi *load balancer* cadangan (*backup*). Loadbalancer1 diberi alamat IP 192.168.200.21 sedangkan Loadbalancer2 diberi alamat 192.168.200.22. Web Server 1 diberi alamat IP 192.168.200.51 sedangkan Web Server diberi alamat IP 192.168.200.52. *Virtual IP address* Heartbeat berada pada 192.168.200.25. Untuk lebih jelas, perancangan ini akan digambarkan pada Gambar 4.5.



Gambar 4.5 Topologi High Availability Load Balancing Aplikasi Heartbeat

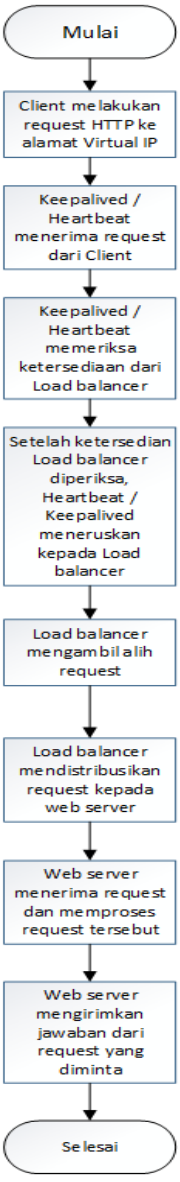
2. Pada perancangan kedua, *failover* akan diterapkan dengan bantuan perangkat lunak Keepalived. Loadbalancer1 akan menjadi *load balancer* utama (master) sedangkan Loadbalancer2 akan menjadi *load balancer* cadangan (backup). Loadbalancer1 diberi alamat IP 192.168.200.21 sedangkan Loadbalancer2 diberi alamat 192.168.200.22. Web Server 1 diberi alamat IP 192.168.200.51 sedangkan Web Server diberi alamat IP 192.168.200.52. *Virtual IP address* berada pada 192.168.200.26. Untuk lebih jelas, perancangan ini akan digambarkan pada Gambar 4.6.



Gambar 4.6 Topologi High Availability Load Balancing Menggunakan Keepalived

4.2.2.1 Alur Kerja Sistem

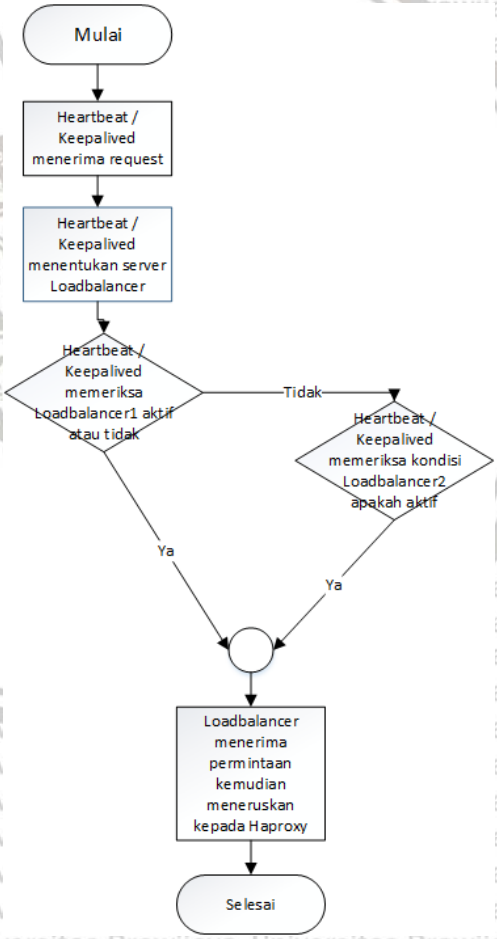
Pada Gambar 4.7 merupakan penjelasan bagaimana sistem yang dirancang dapat berjalan, dimulai dari proses pengguna mengakses *web*. Agar dapat mengakses *web*, pengguna akan melakukan *request* HTTP di alamat *virtual* IP. Kemudian, aplikasi *failover* akan menerima *request* kemudian memeriksa kondisi dari *load balancer*. Selanjutnya, aplikasi failover akan meneruskan request kepada *load balancer*. Kemudian *load balancer* akan mendistribusikan *request* tersebut kepada *web server*. *Web server* memproses *request* tersebut dan mengirimkannya kembali kepada pengguna sebagai respon.



Gambar 4.7 Alur Kerja Sistem Topologi Failover untuk Load Balancing

4.2.2.2 Alur Kerja Failover

Pada Gambar 4.8 menjelaskan bagaimana alur kerja *failover* pada topologi *high availability load balancing*. Heartbeat dan Keepalived masing-masing akan memiliki alamat *virtual* IP. Untuk dapat mengakses web, pengguna akan mengakses virtual IP. Sehingga *request* dari *client* pada awalnya akan masuk diterima oleh *server* Loadbalancer yang telah diimplementasi *failover* dan *loadbalancer*. Perangkat lunak *failover* akan menentukan *server* Loadbalancer mana yang sedang aktif untuk menerima *request* dari *client*. Jika Loadbalancer1 yang bertindak sebagai *server master* dalam klaster sedang dalam kondisi aktif, maka *request* akan diproses oleh Loadbalancer1. Namun jika Loadbalancer1 sedang dalam kondisi tidak aktif, maka perangkat lunak *failover* akan memindahkan *request* kepada *server backup*, yaitu Loadbalancer2. Jika semua *server* Loadbalancer sedang aktif, maka Loadbalancer2 hanya me*monitoring* , Ketika *server* Loadbalancer1 tidak aktif maka Loadbalancer2 akan segera mengambil alih posisi dari Loadbalancer1 untuk melayani *request* dari *client.*

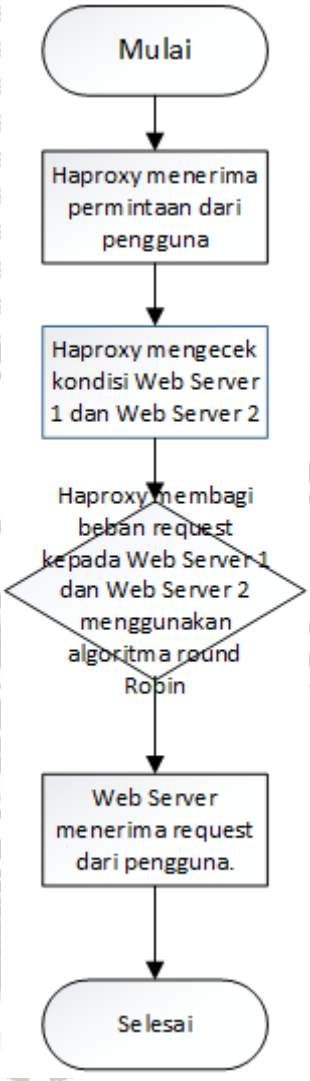


Gambar 4.8 Alur Kerja Failover untuk Load Balancing

4.2.2.3 Alur Kerja Load Balancing

Pada Gambar 4.9 menjelaskan bagaimana alur kerja *load Balancing* pada topologi *high availability load balancing*. Perangkat lunak Haproxy akan menerima

request yang masuk dari *client*. Kemudian Haproxy membagi beban request tersebut kepada dua server Web Server, yaitu Web Server 1 dan Web Server 2 dengan algoritma *round robin*.



Gambar 4.9 Alur Kerja Topologi Load Balancing

4.3 Pengujian Sistem

Terdapat beberapa pengujian yang akan dilakukan untuk menguji sistem. Masing-masing topologi sistem *failover* akan menjalani pengujian sistem dengan skenario yang berbeda. Pengujian yang akan dilakukan adalah pengujian fungsional, pengujian downtime dan pengujian failback.

4.3.1 Pengujian Pada Topologi Sistem Failover Untuk High Availability Web Server

Pengujian sistem pada topologi sistem *failover* untuk *high availability web server* mencakup pengujian fungsional, pengujian *downtime* dan pengujian failback. Skenario pengujian akan dijelaskan pada Tabel 4.4

Tabel 4.4 Skenario Pengujian Topologi High Availability Web Server

Skenario	Jumlah Koneksi Tiap Detik	Skenario Pengujian
1	5	Dilakukan pada saat sistem berjalan, kemudian server utama dimatikan.
2	5	Dilakukan pada saat sistem berjalan, kemudian service Apache2 pada server utama dimatikan.
3	5	Dilakukan pada saat sistem berjalan, kemudian service perangkat lunak *failover* (Heartbeat atau Keepalived) pada server utama dimatikan.

4.3.1.1 Pengujian Fungsional

Pengujian fungsional dilakukan untuk mengetahui implementasi sistem pengujian telah berjalan baik dan benar sesuai dengan kebutuhan fungsional yang telah ditentukan sebelumnya. Pengujian fungsional akan dijelaskan pada Tabel 4.5

Tabel 4.5 Pengujian Fungsional Topologi Sistem Failover Untuk HA Web Server

No	Kebutuhan Fungsional	Hasil
1.	Website masing-masing *web server* dapat diakses oleh pengguna melalui IPnya masing-masing.	
2.	Ketika Web Server 1 mati, *website* dapat diakses oleh pengguna melalu *virtual* IP karena adanya *failover* yang memindahkan *traffic*.	
3.	Ketika Web Server 1 dan 2 mati, *website* masih dapat diakses oleh pengguna melalui *virtual* IP karena adanya *failover* yang memindahkan *traffic*.	
4.	Ketika semua Web Server dalam kondisi mati, pengguna tidak dapat mengakses *website*	

	karena tidak ada *server* yang memberikan layanan.	

4.3.1.2 Pengujian Downtime

Pengujian *downtime* adalah sebuah pengujian yang dijalankan untuk mengetahui berapa lama waktu ketika terjadinya suatu kegagalan atau kerusakan sebuah komponen dalam sistem, namun sistem tersebut masih mampu beroperasi lagi karena terdapat komponen lain yang menjadi *backup* saat komponen utama tersebut mengalami kegagalan. Pengujian *downtime* berpengaruh pada performa sistem ketika terjadi sebuah kegagalan. Pengujian downtime akan dilaksanakan sesuai dengan skenario yang telah dibuat pada Tabel 4.6. Pada tiap skenario pengujian *downtime*, akan dilakukan 12 kali pengujian kemudian hasil dari 12 pengujian tersebut akan dijumlah kemudian di rata-rata agar mendapat hasil yang valid.

Tabel 4.6 Pengujian Downtime Topologi Sistem Failover Untuk HA Web Server

No	Jumlah Koneksi Tiap Detik	Pengujian ke	Nilai *Failback* (detik)	Rata-rata *Failback* (detik)
1	5	1		
		2		
		3		
		4		
		5		
		6		
		7		
		8		
		9		
		10		
		11		
		12		

4.3.1.3 Pengujian Failback

Pengujian failback adalah sebuah pengujian yang dijalankan untuk mengetahui berapa lamu waktu ketika tidak dapat diaksesnya sebuah sumberdaya dikarenakan kembalinya lagi *server master* pada klaster. Pengujian failback akan dilaksanakan sesuai dengan skenario sesuai Tabel 4.7. Akan dilakukan 12 kali pengujian kemudian hasil dari 12 pengujian tersebut akan dirata-rata agar mendapat hasil yang valid.

Tabel 4.7 Pengujian Failback Untuk Topologi Sistem Failover Ha Web Server

No	Jumlah Koneksi Tiap Detik	Pengujian ke	Nilai *Failback* (detik)	Rata-rata *Failback* (detik)
1	5	1		
		2		
		3		
		4		
		5		
		6		
		7		
		8		
		9		
		10		
		11		
		12		

4.3.2 Pengujian Pada Topologi Sistem Failover Untuk High Availability Load Balancing

Pengujian sistem pada topologi sistem *failover* untuk *high availability load balancing* mencakup pengujian fungsional, pengujian *downtime*, dan pengujian *failback*.

4.3.2.1 Pengujian Fungsional

Pengujian fungsional dilakukan untuk mengetahui implementasi sistem pengujian telah berjalan baik dan benar sesuai dengan kebutuhan fungsional yang telah ditentukan sebelumnya. Pengujian fungsional akan dijelaskan pada Tabel 4.8.

Tabel 4.8 Pengujian Fungsional Topologi Sistem Failover Untuk HA Load Balancing

No	Pengujian	Keterangan
1.	Konten Wordpress pada Web Server 1 dapat diakses oleh pengguna	

2.	Konten Wordpress pada Web Server 2 dapat diakses oleh pengguna	
3.	Haproxy pada server Loadbalancer1 dapat membagi *traffic* yang masuk menuju Web Server 1 dan Web Server 2	
4.	Haproxy pada server Loadbalancer2 dapat membagi *traffic* yang masuk menuju Web Server 1 dan Web Server 2	
5.	Ketika server Loadbalancer1 dalam kondisi mati, pengguna masih dapat mengakses alamat virtual IP karena ada *server* Loadbalancer2 ada implementasi *failover* yang mengalihkan *traffic* menuju *server backup*, yaitu Loadbalancer2.	
6.	Ketika *server* Loadbalancer2 dalam kondisi mati, pengguna tidak dapat mengakses website karena tidak ada server aktif untuk menangani *traffic* yang masuk.	

4.3.2.2 Pengujian Downtime

Pengujian *downtime* adalah sebuah pengujian yang dijalankan untuk mengetahui berapa lama waktu ketika terjadinya suatu kegagalan atau kerusakan sebuah komponen dalam sistem, namun sistem tersebut masih mampu beroperasi lagi karena terdapat komponen lain yang menjadi *backup* saat komponen utama tersebut mengalami kegagalan. Pengujian *downtime* berpengaruh pada performa sistem ketika terjadi sebuah kegagalan. Pengujian downtime akan dilaksanakan sesuai dengan skenario yang telah dibuat pada Tabel 4.9. Pada tiap skenario pengujian *downtime*, akan dilakukan 12 kali pengujian kemudian hasil dari 12 pengujian tersebut akan dijumlah kemudian di rata-rata agar mendapat hasil yang valid.

Tabel 4.9 Pengujian Downtime Topologi Sistem Failover Untuk HA Load Balancing

No	Jumlah Koneksi Tiap Detik	Pengujian ke	Nilai *Failback* (detik)	Rata-rata *Failback* (detik)
1	5	1		
		2		
		3		
		4		

		5		
		6		
		7		
		8		
		9		
		10		
		11		
		12		

4.3.2.3 Pengujian Failback

Pengujian *failback* adalah sebuah pengujian yang dijalankan untuk mengetahui berapa lamu waktu ketika tidak dapat diaksesnya sebuah sumberdaya dikarenakan kembalinya lagi *server master* pada klaster. Pengujian *failback* akan dilaksanakan sesuai dengan skenario sesuai Tabel 4.10**.** Akan dilakukan 12 kali pengujian kemudian hasil dari 12 pengujian tersebut kan dirata-rata agar mendapat hasil yang valid.

Tabel 4.10 Pengujian Failback untuk Topologi HA Load Balancing

No	Jumlah Koneksi Tiap Detik	Pengujian ke	Nilai *Failback* (detik)	Rata-rata *Failback* (detik)
1	5	1		
		2		
		3		
		4		
		5		
		6		
		7		
		8		
		9		
		10		
		11		
		12		

BAB 5 IMPLEMENTASI

Bab ini berisi penjelasan tentang proses dan langkah-langkah implementasi dari bab perancangan. Implementasi yang dilakuan adalah implementasi sistem *failover* untuk kebutuhan *High Availaibility Web Server dan High Availability Load Balancing* menggunakan aplikasi *Heartbeat* dan *Keepalived*.

5.1 Implementasi Topologi Sistem Failover untuk Kebutuhan High Availability Web Server

Implementasi pada topologi ini menggunakan 2 buah mesin virtual yaitu *Server1*, dan *Server2*. Semua mesin virtual menggunakan sistem operasi Ubuntu Server. Tahapan-tahapan implementasi didalam topologi ini, akan dijelaskan pada sub bab dibawah.

5.1.1 Konfigurasi Ubuntu Server

Dalam penelitian ini menggunakan perangkat lunak virtualisasi Virtualbox. Semua mesin virtual akan mendapat konfigurasi *network* yang sama yaitu *Host-only network* dan *NAT network*. Fungsi *Host-only network* agar mesin virtual dapat saling berkomunikasi dengan mesin virtual yang lain. Sedangkan fungsi dari *NAT network* berfungsi agar mesin virtual dapat mengakses internet.

Sistem operasi yang digunakan pada seluruh mesin virtual adalah Ubuntu Server. Server-server yang dibangun adalah *Server 1* dan *Server 2*.Pada masing-masing mesin virtual akan dikonfigurasi alamat IP *static*. Untuk mengganti alamat IP dengan menggunakan perintah seperti pada tabel 5.1.

Tabel 5.1 Perintah Edit Interface

1	$ sudo nano /etc/network/interfaces

Langkah selanjutnya adalah konfigurasi untuk melakukan pengaturan alamat IP mesin virtual dari IP DHCP menjadi IP *static*. Alamat IP *static* untuk Server 1 192.168.200.51, Server 2 ialah 192.168.200.52.

Tabel 5.2 Konfigurasi IP Static Server 1

1	$ source /etc/network/interfaces/
2	$ auto enp0s3
3	$ iface enp0s3 inet dhcp
4	$ auto enp0s8
5	$ iface enp0s8 inet static
6	$ address 192.168.200.51
7	$ netmask 255.255.255.0

Tabel 5.3 Konfigurasi IP Static Server 2

```
1  $ source /etc/network/interfaces/
2  $ auto enp0s3
3  $ iface enp0s3 inet dhcp
4  $ auto enp0s8
5  $ iface enp0s8 inet static
6  $ address 192.168.200.52
7  $ netmask 255.255.255.0
```

Setelah mengganti semua alamat IP mesin virtual menjadi static, langkah selanjutnya adalah mengganti *hostname* semua mesin virtual. Untuk melakukan perubahan dilakukan dengan menuliskan perintah seperti pada Tabel 5.4

Tabel 5.4 Perintah Edit Hostname

```
1  $ sudo nano /etc/hostname
```

Hostname dari Web Server 1 adalah Server1 dan Web Server 2 adalahServer2.

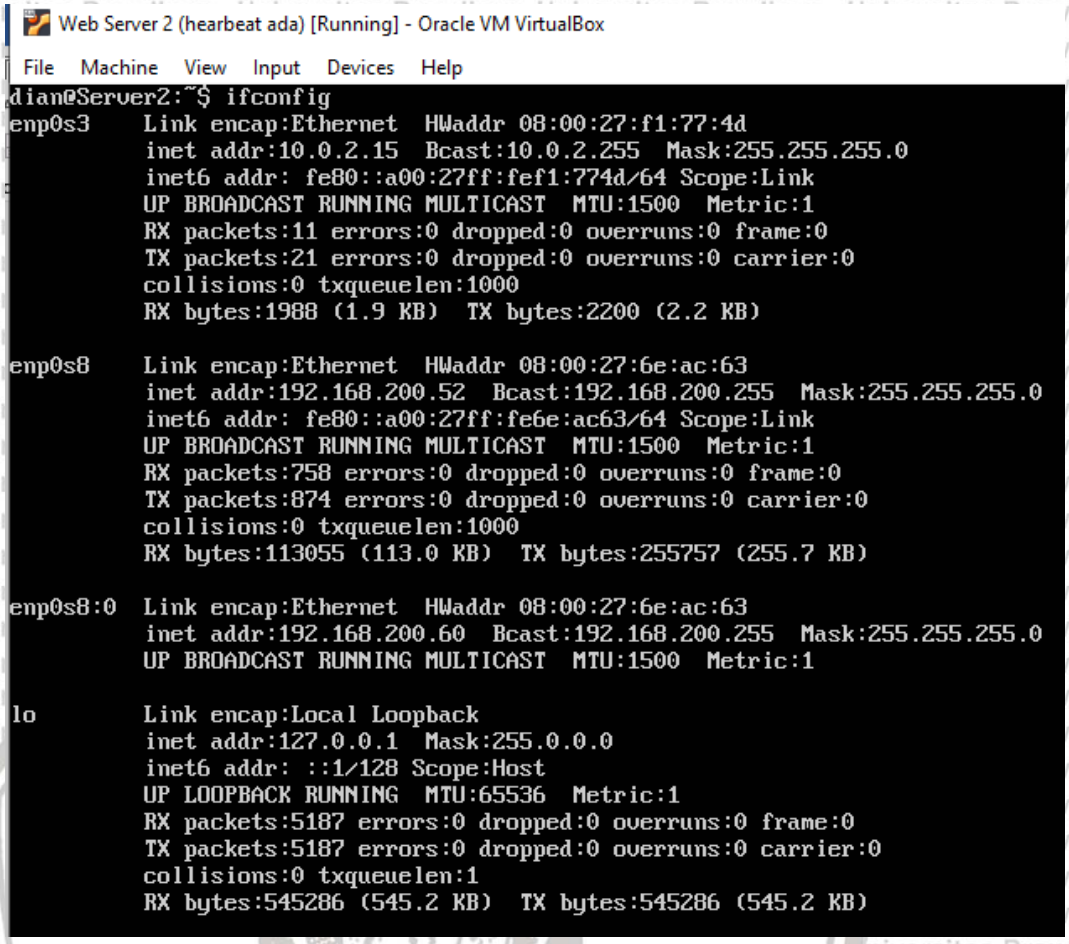
Tabel 5.5 Hostname dari Server 1

```
1  $ Server1
```

Tabel 5.6 Hostname dari Server 2

```
1  $ Server2
```

Setelah dilakukan konfigurasi IP *static* dan *hostname*, semua mesin virtual akan di*restart* agar mesin-mesin virtual dikenal dengan IP *static* dan *hostname* yang telah ditetapkan. Pada Gambar 5.1 ditampilkan hasil dari konfigurasi IP static dan hostname.



Gambar 5.1 Hasil Konfigurasi IP Static dan Hostname Pada Mesin Virtual Server 2

Setelah mengubah hostname, selanjutnya ialah mengubah nama hosts pada masing masing Server dengan menuliskan perintah $sudo nano /etc/hosts dengan konfigurasi pada Tabel 5.7 Konfigurasi File Hosts

Tabel 5.7 Konfigurasi File Hosts

Konfigurasi file etc/hosts pada Server1	
1.	127.0.0.1 localhost 127.0.1.1 Server1 192.168.200.51 Server1
Konfigurasi file etc/hosts pada Server2	
2.	127.0.0.1 localhost 127.0.1.1 Server2 192.168.200.52 Server2

5.1.2 Implementasi dan Konfigurasi Web Server

Pada penelitian ini menggunakan perangkat lunak Apache2 untuk perangkat lunak *web server*. Untuk database menggunakan perangkat lunak MySql. Apache2 dan MySql dapat di*install* langsung dari *repository* Ubuntu menggunakan perintah

Tabel 5.8 Perintah Installasi Apache2

```
1.  $ sudo apt-get install lamp-server^ -y
```

Setelah melakukan perintah tersebut, proses unduhan dan instalasi akan berjalan. Setelah proses selesai, maka perangkat lunak Apache2, MySql dan PHP berhasil di*install* pada mesin virtual. Kemudian langkah selanjutnya adalah melakukan unduhan pada *website official* Wordpress dengan melakukan perintah

Tabel 5.9 Perintah Download Wordpress

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

Setelah proses unduhan selesai, langkah selanjutnya adalah mengekstrak file tersebut dengan perintah

Tabel 5.10 Ekstrak File Wordpress

```
1.  $ sudo tar -xzvf latest.tar.gz
```

Setelah proses ekstrak selesai, langkah selanjutnya adalah memindahkan file hasil ekstrak ke direktori /var/www/html dengan perintah

Tabel 5.11 Perintah Memindahkan Hasil Ekstrak

```
1.  $ sudo rsync -av wordpress/* /var/www/html
```

Setelah semua file selesai dipindahkan, langkah selanjutnya adalah mengatur permissions pada direktori /var/www/html/ dengan perintah

Tabel 5.12 Perintah Mengatur Permissions

```
1.  $ sudo chown -R www-data:www-data /var/www/html
2.  $ sudo chmod  -R 755 /var/www/html
```

Pada baris 1, digunakan untuk mengganti *ownership* dari direktori tersebut menjadi milik user www-user. Sedangkan pada baris 2, CHMOD bermakna *change mode*. Jadi baris 2 digunakan untuk mengganti hak akses tersebut menjadi 755 yang artinya *owner* dapat melakukan rwx, group dari owner dapat melakukan r-x dan selain *owner* dan group dapat melakukan r-x. Setelah langkah ini selesai, langkah selanjutnya adalah menjalankan MySql pada mesin virtual dengan perintah

Tabel 5.13 Perintah Masuk ke Mysql

```
1.  $ sudo mysql -u root -p
```

Setelah menginputkan password, maka akan masuk dalam mode MySql. Langkah selanjutnya adalah membuat sebuah database baru bernama "wordpress" dengan perintah

Tabel 5.14 Perintah Membuat Database Wordpress

```
1.  $ mysql > CREATE DATABASE wordpress;
```

Kemudian langkah selanjutnya adalah membuat user baru dalam MySql dengan perintah

Tabel 5.15 Perintah Membuat User Baru di MySql

```
1.  $ mysql > CREATE USER dian IDENTIFIED BY '';
```

Setelah user berhasil dibuat, langkah selanjutnya adalah mengatur *privileges database* "wordpress" dengan perintah

Tabel 5.16 Mengatur Privileges Pada Database Wordpress

```
1.  $ mysql > GRANT ALL PRIVILEGES ON wordpress.* To 'dian'@'localhost'
    IDENTIFIED BY '';
2.  $ mysql > FLUSH PRIVILEGES ;
```

Setelah berhasil, keluar dari mode MySql dengan perintah

Tabel 5.17 keluar dari MySql

```
1.  $ mysql > EXIT;
```

Langkah selanjutnya adalah masuk kedalam direktori /var/www/html kemudian mengcopy wp-config-sample.php menjadi wp-config.php dengan perintah

```
1.  $ cd /var/www/html
2.  $ sudo mv wp-config-sample.php wp-config.php
```

Setelah selesai mengcopy file wp-config.php, kemudian buka file ini dengan perintah

Tabel 5.18 Perintah Membuka wp-config.php

```
1.  $ sudo nano wp-config.php
```

Setelah file berhasil terbuka, maka dilakukan edit agar wordpress nanti dapat terkoneksi dengan database. Pada Tabel 5.19 akan dijelaskan bagian mana yang diedit

Tabel 5.19 Konfigurasi Wp-config.php

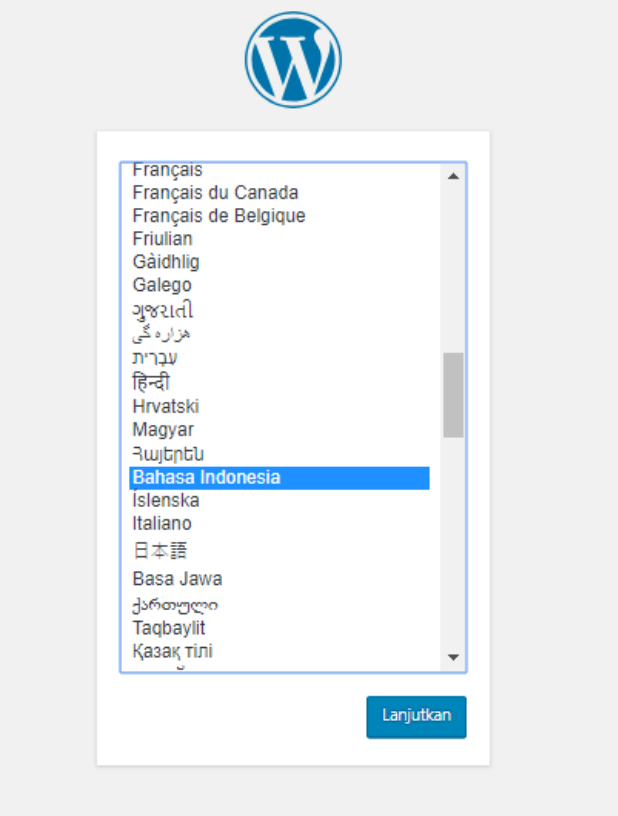
	wp-config.php
1.	$ define('DB_NAME', 'wordpress');
2.	$ define('DB_USER', 'dian');
3.	$define ('DB_PASSWORD', '';) $define ('DB_HOST, 'localhost')

Setelah konfigurasi file wp-config.php selesai, langkah selanjutnya adalah melakukan restart service Apache2 dan MySql dengan perintah

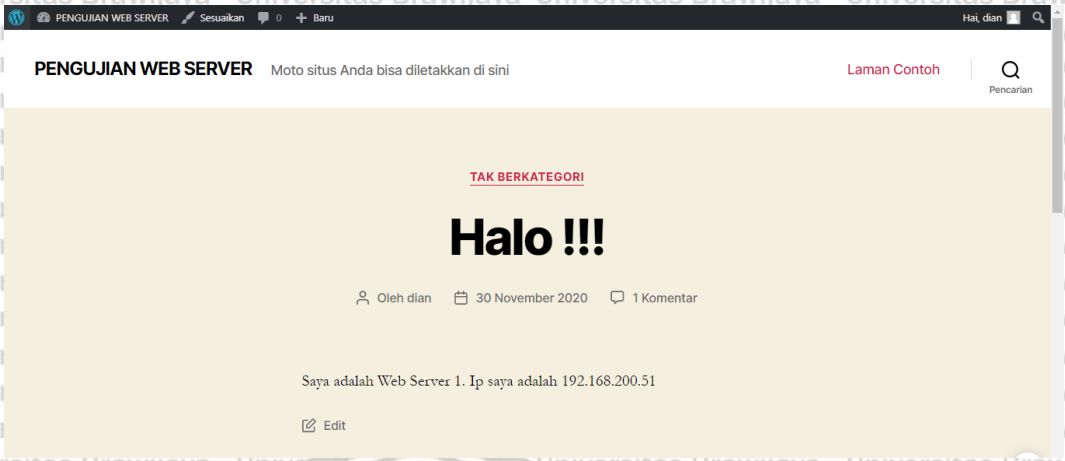
Tabel 5.20 Perintah Melakukan Restart Service

1.	$ sudo systemctl restart apache2.service
2.	$ sudo systemctl restart mysql.service

Kemudian langkah selanjutnya adalah membuka browser dan melakukan sedikit konfigurasi pada wordpress



Gambar 5.2 Tampilan Awal Wordpress



Gambar 5.3 Tampilan Akhir Wordpress Setelah Konfigurasi

Proses instalasi, konfigurasi LAMP Server, database Mysql telah selesai dilakukan pada Server 1. Untuk proses instalasi pada Server 2, dilakukan sama dengan cara pada tabel 5.1 sampai dengan 5.20.

5.1.3 Implementasi dan Konfigurasi Failover Menggunakan Heartbeat

Langkah selanjutnya adalah melakukan implementasi aplikasi *failover* Heartbeat. Yang pertama dilakukan adalah melakukan instalasi pada Server1 dan Server2 dengan melakukan perintah seperti pada Tabel 5.21.

Tabel 5.21 Perintah Instalasi Heartbeat

1	`$ sudo apt-get install heartbeat`

Setelah proses instalasi selesai, selanjutnya adalah melakukan konfigurasi Heartbeat pada Server1 dan Server2. Terdapat 3 file yang harus dikonfigurasi jika ingin menggunakan Heartbeat. Ketiga file tersebut masing-masing adalah Ha.cf, authkeys dan haresources.

File pertama yang dikonfigurasi adalah Ha.cf. Untuk membuat file ha.cf dapat dilakukan dengan perintah seperti pada Tabel 5.22

Tabel 5.22 Perintah Membuat File Ha.cf

1	`sudo nano /etc/ha.d/ha.cf`

Setelah file ha.cf berhasil dibuat, langkah selanjutnya ialah melakukan konfigurasi file tersebut dengan perintah seperti pada Tabel 5.23

Tabel 5.23 Konfigurasi File Ha.cf

	ha.cf
1	`use_logd on`
2	`bcast enp0s8`
3	`udpport 694`
4	`udp enp0s8`
5	`keepalive 2`

```
6   deadtime 5
7   initdead 30
8   warntime 10
9   auto_failback on
10  node Server1
11  node Server2
```

Berikut ini penjelasan dari tabel diatas:

Baris 1 : mengatur Heartbeat agar mencatat log message pada direktori /var/log/ha-log

Baris 2 : mengatur interface yang digunakan Heartbeat untuk melakukan broadcast

Baris 3 : menggunakan port untuk UDP yaitu 694. 694 merupakan default port untuk UDP

Baris 4 : mengatur UDP untuk menggunakan interface enp0s8

Baris 5 : mengatur interval antar packets Heartbeat selama 2 detik

Baris 6 : mengatur seberapa cepat Heartbeat mendefinisikan sebuah node berubah menjadi tidak aktif

Baris 7 : parameter yang digunakan untuk mengatur waktu yang dibutuhkan untuk menentukan sebuah node dalam kondisi tidak aktif saat Heartbeat pertama kali dijalankan

Baris 8 : menentukan waktu berapa lama pesan heartbeat dikategorikan terlambat

Baris 9 : menentukan opsi failback. Jika di on, master akan mengambil alih posisi slave lagi ketika master menjadi aktif

Baris 10-12 : mengatur node mana saja yang menjadi anggota klaster

Kemudian langkah selanjutnya membuat file authkeys. File authkeys pada berfungsi sebagai metode autentikasi *node* dalam klaster. File authkeys harus memuat konfigurasi yang sama pada semua *node*, agar masing-masing *node* dalam klaster dapat saling berkomunikasi. Untuk membuat file authkeys menggunakan perintah seperti pada Tabel 5.24.

Tabel 5.24 Perintah Membuat File Authkeys

```
1   Sudo nano /etc/ha.d/authkeys
```

Setelah *file* authkeys berhasil dibuat, langkah selanjutnya adalah melakukan konfigurasi pada *file* tersebut seperti pada Tabel 5.25

Tabel 5.25 Konfigurasi File Authkeys

```
    authkeys
1   auth 5
```

2	5 md5 dianpratama

Pada Tabel 5.25 dijelaskan bahwa autentikasi menggunakan adalah md5 dengan *key* "dianpratama".

Setelah konfigurasi file authkeys selesai, Langkah selanjutny adalah mengubah permissions file authkeys dengan perintah seperti pada Tabel 5.26 Ubah Permissions Authkeys.

Tabel 5.26 Ubah Permissions Authkeys

1	Sudo chmod 600 /etc/ha.d/authkeys

Kemudian langkah selanjutnya membuat *file* haresources. Untuk membuat *file* haresources dilakukan dengan perintah seperti pada tabel Tabel 5.27.

Tabel 5.27 Perintah Membuat File Haresources

1	Sudo nano /etc/ha.d/haresources

Setelah *file* haresources dibuat, kemudian mengkonfigurasi *file* haresources seperti pada tabel Tabel 5.28 Konfigurasi File Haresources.

Tabel 5.28 Konfigurasi File Haresources

	haresources
1	Server 1 192.168.200.60/24/enp0s8

File haresources berfungsi untuk mengkonfigurasi IP *virtual* untuk mewakili beberapa IP *address* yang dimiliki oleh masing-masing *web server*. Server1 merupakan node yang menjadi *server master* dalam klaster.

Langkah-langkah ini dijalankan pada semua *virtual machine web server*. Konfigurasi *file* authkeys,haresources dan ha.cf pada semua mesin virtual harus identik. Setelah selesai mengkonfigurasi pada semua mesin virtual, langkah berikutnya adalah mengakses alamat virtual IP dari Heartbeat 192.168.200.60 untuk pada *browser* untuk memastikan sistem *failover* yang dirancang menggunakan perangkat lunak Heartbeat sukses dan berhasil.

5.1.4 Implementasi dan Konfigurasi Failover Menggunakan Keepalived

Langkah pertama adalah melakukan instalasi perangkat lunak Keepalived pada 2 (dua) mesin virtual. Untuk melakukan instalasi, dapat dilakukan dengan perintah seperti pada Tabel 5.29.

Tabel 5.29 Perintah Instalasi Keepalived

1.	$ sudo apt-get install keepalived

Setelah proses instalasi Keepalived selesai, langkah selanjutnya adalah melakukan konfigurasi. Untuk dapat menggunakan Keepalived, ada 1 (satu) file yang harus dikonfigurasi yaitu file keepalived.conf. File tersebut terdapat di

direktori /etc/keepalived. Untuk membuka dan mengkonfigurasi file tersebut dapat dilakukan dengan perintah seperti pada Tabel 5.30.

Tabel 5.30 Perintah Mengkonfigurasi File Keepalived.conf

1.	$ sudo nano /etc/keepalived/keepalived.conf

Setelah *file* keepalived.conf terbuka, kemudian lakukan konfigurasi seperti pada Tabel 5.31 dan Tabel 5.32.

Tabel 5.31 Konfigurasi Keepalived.conf pada Web Server 1

	keepalived.conf
1	vrrp_instance apache2 {
2	Interface enp0s8
3	state MASTER
4	virtual_router_id 15
5	priority 110
6	advert_int 1
7	authentication {
8	auth_type PASS
9	auth_pass dian
10	}
11	virtual_ipaddress {
12	192.168.200.59
13	}
14	}

Tabel 5.32 Konfigurasi Keepalived.conf pada Web Server 2

	keepalived.conf
1	vrrp_instance apache2 {
2	interface enp0s8
3	state MASTER
4	virtual_router_id 15
5	priority 109
6	advert_int 1
7	authentication {
8	auth_type PASS
9	auth_pass dian
10	}
11	virtual_ipaddress {
12	192.168.200.59
13	}

```
14  }
```

Penjelasan dari konfigurasi keepalived.conf diatas adalah

Baris 1 : mendefinisikan blok contoh VRRP

Baris 2 : menentukan status *instance* dalam penggunaan standard

Baris 3 : virtual_router_id adalah pengenal numerik untuk instance Virtual Router. Konfigurasi ini harus sama pada semua Web Server yang akan dijadikan high availability.

Baris 4 : menentukan prioritas instance di router VRRP. Semakin tinggi angka prioritasnya makan instance tersebut akan dijadikan master

Baris 5 : menentukan interval *advert* dalam hitungan detik

Baris 6 – 9 : menentukn jenis autentikasi yang digunakan dan menentukan password otentikasi

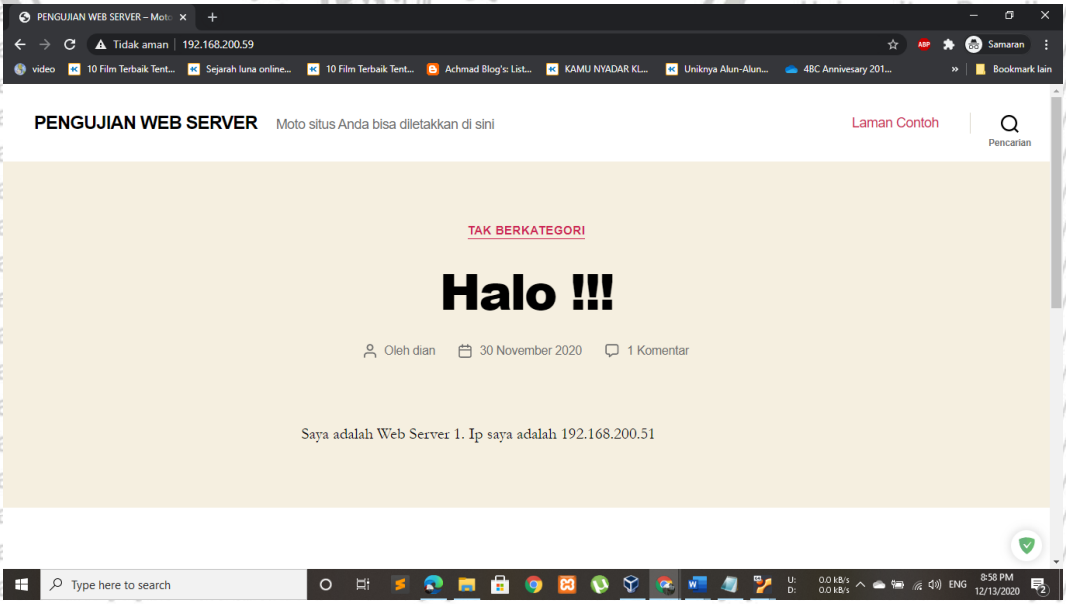
Baris 10 – 12 : menentukan alamat IP virtual

Setelah mengkonfigurasi file keepalived.conf pada semua mesin virtual, langkah selanjutnya adalah melakukan perintah untuk memulai service Keepalived pada semua mesin virtual seperti pada Tabel 5.33.

Tabel 5.33 Perintah Start Keepalived

```
1  $ sudo service keepalived start
```

Setelah *service* Keepalived berhasil dijalankan, langkah akhir adalah membuka *browser* kemudian memasukkan alamat IP *virtual* Keepalived untuk memeriksa apakah implementasi dan konfigurasi sudah berjalan baik dan benar.



Gambar 5.4. Mencoba Akses Alamat Virtual IP Keepalived

5.2 Implementasi Topologi Sistem Failover untuk High Availability Load Balancing

Pada topologi *high availability load balancing* ini akan menggunakan empat (4) buah mesin virtual. Dari empat (4) mesin virtual tersebut terdiri dari 2 (dua) buah mesin virtual untuk *load balancer* dan 2 (dua) buah mesin virtual untuk *web server*. Didalam mesin virtual yang berperan sebagai *load balancer*, masing-masing akan dilakukan instalasi perangkat lunak *load balancer* Haproxy serta perangkat lunak *failover* Heartbeat dan Keepalived. Sedangkan pada mesin *virtual* yang berperan sebagai *web server*, akan dilakukan instalasi Apache2 agar *service web server* dapat tersedia. Tahap-tahapan implementasi dalam topologi ini akan dijelaskan pada sub bab dibawah

5.2.1 Konfigurasi Ubuntu Server Untuk Sistem High Availability Load Balancing

Server yang dibangun dalam penelitian ini menggunakan perangkat lunak virtualisasi Virtualbox. Semua mesin virtual akan mendapat konfigurasi *network* yang sama yaitu Host-only *network* dan NAT *network*. Fungsi Host-only network agar mesin virtual dapat saling berkomunikasi dengan mesin virtual yang lain. Sedangkan fungsi dari NAT *network* agar mesin virtual dapat mengakses *internet*.

Sistem operasi yang digunakan pada seluruh mesin virtual adalah Linux Ubuntu Server. Server-server yang dibangun adalah Loadbalancer1, Loadbalancer2, Server1 dan Server2. Masing-masing mesin virtual akan dikonfigurasi alamat IP static. Untuk mengganti alamat IP dengan menggunakan perintah seperti pada Tabel 5.34**Error! Reference source not found.**.

Tabel 5.34 Perintah Edit Interface

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

Langkah selanjutnya adalah konfigurasi untuk melakukan pengaturan alamat IP mesin virtual dari IP DHPC menjadi IP *static*. Alamat IP *static* untuk Loadbalancer1 192.168.200.21, Loadbalancer2 192.168.200.22, Server1 192.168.200.51 dan Server2 192.168.200.52.

Tabel 5.35 Konfigurasi Alamat IP Static Loadbalancer1

```
1  $ source /etc/network/interfaces/
2   $ auto enp0s8
3   $ iface enp0s8 inet dhcp
4   $ auto enp0s3
5   $ iface enp0s3 inet static
6   $ address 192.168.200.21
```

7	$ netmask 255.255.255.0

Tabel 5.36 Konfigurasi Alamat IP Static Loadbalancer2

1	$ source /etc/network/interfaces/
2	$ auto enp0s8
3	$ iface enp0s8 inet dhcp
4	$ auto enp0s3
5	$ iface enp0s3 inet static
6	$ address 192.168.200.22
7	$ netmask 255.255.255.0

Tabel 5.37 Konfigurasi Alamat IP Static Server1

1	$ source /etc/network/interfaces/
2	$ auto enp0s8
3	$ iface enp0s8 inet dhcp
4	$ auto enp0s3
5	$ iface enp0s3 inet static
6	$ address 192.168.200.51
7	$ netmask 255.255.255.0

Tabel 5.38 Konfigurasi Alamat IP Static Server2

1	$ source /etc/network/interfaces/
2	$ auto enp0s8
3	$ iface enp0s8 inet dhcp
4	$ auto enp0s3
5	$ iface enp0s3 inet static
6	$ address 192.168.200.52
7	$ netmask 255.255.255.0

Langkah selanjutnya adalah mengganti hostname semua mesin virtual. Untuk melakukan perubahan dilakukan dengan menuliskan perintah seperti pada Tabel 5.39

Tabel 5.39 Perintah Edit Hostname

1	$ sudo nano /etc/hostname

Masing-masing dari mesin virtual akan diatur hostnamenya yaitu Loadbalancer1, Loadbalancer2, Server1 dan Server2. Pada Tabel 5.40 hingga Tabel 5.43 akan ditampilkan konfigurasi hostname masing-masing mesin virtual.

Tabel 5.40 Hostname dari Loadbalancer1

1	$ loadbalancer1

Tabel 5.41 Hostname dari Loadbalancer2

1	$ loadbalancer2

Tabel 5.42 Hostname dari Server1

1	$ Server1

Tabel 5.43 Hostname dari Server2

1	$ Server2

Langkah selanjutnya mengubah konfigurasi file hosts yang dengan menggunakan perintah seperti pada Tabel 5.44

Tabel 5.44 Perintah Edit File Hosts

1	$sudo nano /etc/hosts

Setelah file tersebut terbuka, kemudian ganti konfigurasi pada masing-masing virtual mesin seperti pada Tabel 5.45.

Tabel 5.45 File Hosts pada Virtual Machine WebServer 1

$sudo nano /etc/hosts	
1	127.0.0.1 localhost
2	127.0.1.1 Server1
3	192.168.200.21 Server1

Tabel 5.46 File Hosts pada Virtual Machine WebServer 2

$sudo nano /etc/hosts	
1	127.0.0.1 localhost
2	127.0.1.1 Server2
3	192.168.200.22 Server2

Tabel 5.47 File Hosts pada Virtual Machine Loadbalancer1

$sudo nano /etc/hosts	
1	127.0.0.1 localhost
2	127.0.1.1 loadbalancer1
3	192.168.200.21 loadbalancer1

Tabel 5.48 File Hosts pada Virtual Machine Loadbalancer2

$sudo nano /etc/hosts	
1	127.0.0.1 localhost
2	127.0.1.1 loadbalancer2
3	192.168.200.22 loadbalancer2

Setelah dilakukan konfigurasi IP *static* dan *hostname*, semua mesin virtual akan di*restart* agar mesin-mesin virtual dikenal dengan IP *static* dan *hostname* yang telah ditetapkan.

5.2.2 Implementasi dan Konfigurasi Web Server

Untuk Implementasi dan Konfigurasi Web Server pada topologi system failover Untuk High Availability Load Balancing prosesnya sama dengan sub bab 5.1.2 Implementasi dan Konfigurasi Web Server.

5.2.3 Implementasi dan Konfurasi Haproxy

Langkah pertama adalah melakukan instalasi perangkat lunak Haproxy pada 2 buah mesin virtual yang berperan sebagai *Loadbalancer*, yaitu pada Loadbalancer1 dan Loadbalancer2. Untuk men*download* dan meng*install* Haproxy dapat dilakukan dengan perintah seperti pada Tabel 5.49**Error! Reference source not found.**.

Tabel 5.49 Perintah Instalasi Haproxy

1.	$sudo apt-get install haproxy

Setelah langkah tersebut selesai, selanjutnya adalah melakukan konfigurasi Haproxy. Untuk dapat menggunakan Haproxy, terdapat 1 (satu) file yang harus dikonfigurasi yaitu haproxy.cfg. Untuk membuka dan mengedit konfigurasi file tersebut dapat dilakukan dengan perintah seperti pada Tabel 5.50.

Tabel 5.50 Perintah Membuka Haproxy.cfg

1.	$ sudo nano /etc/haproxy/haproxy.cfg

Setelah file haproxy.cfg terbuka, langkah selanjutnya menambah konfigurasi haproxy seperti pada Tabel 5.51. Konfigurasi Haproxy

Tabel 5.51. Konfigurasi Haproxy

No	/etc/haproxy/haproxy.cfg
1.	listen firstbalance
2.	bind *:80
3.	balance roundrobin
4.	option forwardfor
5.	option httpchk
6.	server Server1 192.168.200.51:80 check
7.	server Server2 192.168.200.52:80 check

Setelah langkah konfigurasi selesai, langkah akhir adalah menjalankan service Haproxy agar berjalan dengan perintah pada Tabel 5.52.

Tabel 5.52 Perintah Menjalankan Haproxy

1.	$ sudo service haproxy start

Setelah service Haproxy, dapat dicoba dengan mengakses masing-masing alamat IP Loadbalancer1 dan Loadbalancer2 pada browser apakah konfigurasi yang dilakukan sudah dapat berjalan dengan baik.

5.2.4 Implementasi dan Konfigurasi Failover Menggunakan Heartbeat

Langkah selanjutnya adalah melakukan implementasi aplikasi *failover* Heartbeat. Yang pertama dilakukan adalah melakukan instalasi pada Loadbalancer1 dan Loadbalancer2 dengan melakukan perintah seperti pada Tabel 5.53Tabel 5.21.

Tabel 5.53 Perintah Instalasi Heartbeat

1	$ sudo apt-get install heartbeat

Setelah proses instalasi selesai, selanjutnya adalah melakukan konfigurasi Heartbeat pada Loadbalancer1 dan Loadbalancer2. Terdapat 3 file yang harus dikonfigurasi jika ingin menggunakan Heartbeat. Ketiga file tersebut masing-masing adalah Ha.cf, authkeys dan haresources.

File pertama yang dikonfigurasi adalah Ha.cf. Untuk membuat file ha.cf dapat dilakukan dengan perintah seperti pada Tabel 5.54.

Tabel 5.54 Perintah Membuat File Ha.cf

1	sudo nano /etc/ha.d/ha.cf

Setelah file ha.cf berhasil dibuat, langkah selanjutnya ialah melakukan konfigurasi file tersebut dengan perintah seperti pada Tabel 5.55.

Tabel 5.55 Konfigurasi File Ha.cf

	ha.cf

```
1   use_logd on
2   bcast enp0s8
3   udpport 694
4   udp enp0s8
5   keepalive 2
6   deadtime 5
7   initdead 30
8   warntime 10
9   auto_failback on
10  node Loadbalancer1
11  node Loadbalancer2
```

Berikut ini penjelasan dari tabel diatas:

Baris 1 : mengatur Heartbeat agar mencatat log message pada direktori /var/log/ha-log

Baris 2 : mengatur interface yang digunakan Heartbeat untuk melakukan broadcast

Baris 3 : menggunakan port untuk UDP yaitu 694. 694 merupakan default port untuk UDP

Baris 4 : mengatur UDP untuk menggunakan interface enp0s8

Baris 5 : mengatur interval antar packets Heartbeat selama 2 detik

Baris 6 : mengatur seberapa cepat Heartbeat mendefinisikan sebuah node berubah menjadi tidak aktif

Baris 7 : parameter yang digunakan untuk mengatur waktu yang dibutuhkan untuk menentukan sebuah node dalam kondisi tidak aktif saat Heartbeat pertama kali dijalankan

Baris 8 : menentukan waktu berapa lama pesan heartbeat dikategorikan terlambat

Baris 9 : menentukan opsi failback. Jika di on, master akan mengambil alih posisi slave lagi ketika master menjadi aktif

Baris 10-12 : mengatur node mana saja yang menjadi anggota klaster

Kemudian langkah selanjutnya membuat file authkeys. File authkeys pada berfungsi sebagai metode autentikasi *node* dalam klaster. File authkeys harus memuat konfigurasi yang sama pada semua *node*, agar masing-masing *node* dalam klaster dapat saling berkomunikasi. Untuk membuat file authkeys menggunakan perintah seperti pada Tabel 5.56Tabel 5.24.

Tabel 5.56 Perintah Membuat File Authkeys

1	Sudo nano /etc/ha.d/authkeys

Setelah *file* authkeys berhasil dibuat, langkah selanjutnya adalah melakukan konfigurasi pada *file* tersebut seperti pada Tabel 5.57.Tabel 5.25

Tabel 5.57 Konfigurasi File Authkeys

	authkeys
1	auth 5
2	5 md5 dianpratama

Pada Tabel 5.57Tabel 5.25 dijelaskan bahwa autentikasi menggunakan algoritma md5 dengan *key* "dianpratama".

Setelah konfigurasi file authkeys selesai, langkah selanjutnya membuat *file* haresources. Untuk membuat *file* haresources dilakukan dengan perintah seperti pada tabel Tabel 5.58.

Tabel 5.58 Perintah Membuat File Haresources

1	Sudo nano /etc/ha.d/haresources

Setelah *file* haresources dibuat, kemudian konfigurasi *file* haresources seperti pada tabel Tabel 5.59.

Tabel 5.59 Konfigurasi Haresource

	haresources
1	Loadbalancer1 IPAddr::192.168.200.25/24/enp0s8

File haresources berfungsi untuk menjalankan IP *virtual* untuk mewakili beberapa IP *address* yang dimiliki oleh masing-masing *server loadbalancer*. Loadbalancer1 merupakan node yang menjadi *server master* dalam klaster, IPaddr merupakan *virtual* IP yang akan diaktifkan jika *service* Heartbeat aktif, sedangkan Haproxy merupakan service yang dijalankan Heartbeat.

Langkah-langkah ini dijalankan pada semua *virtual machine web server*. Konfigurasi *file* authkeys,haresources dan ha.cf pada semua mesin *virtual* harus identik. Setelah selesai mengkonfigurasi pada semua mesin virtual, langkah berikutnya adalah mengakses alamat virtual IP dari Heartbeat 192.168.200.25 pada *browser* untuk memastikan sistem *failover* yang dirancang menggunakan perangkat lunak Heartbeat sukses dan berhasil.

5.2.5 Implementasi dan Konfigurasi Failover Menggunakan Keepalived

Langkah pertama adalah melakukan instalasi perangkat lunak Keepalived pada 2 (dua) buah mesin virtual, yaitu pada loadbalancer1 dan loadbalancer2. Untuk melakukan instalasi, dapat dilakukan dengan perintah seperti pada Tabel 5.60.

Tabel 5.60 Perintah Instalasi Keepalived

1.	$ sudo apt-get install keepalived

Langkah selanjutnya adalah melakukan konfigurasi pada sysctl.conf dengan perintah seperti padaTabel 5.61.

Tabel 5.61 Perintah Edit Sysctl.conf

```
1.  $ sudo nano /etc/sysctl.conf
```

Kemudian lakukan konfigurasi pada file tersebut dengan menambahkan satu baris konfigurasi untuk mengizinkan Haproxy melakukan bind ke alamat virtual IP. Pada Tabel 5.62 merupakan baris tambahan konfigurasi tersebut.

Tabel 5.62 Baris Konfigurasi Sysctl.conf

```
1.  net.ipv4.ip_nonlocal_bind=1
```

Setelah menambahkan baris konfigurasi, langkah selanjutnya adalah melakukan perintah seperti pada Tabel 5.63.

Tabel 5.63 Perintah Sysctl

```
1.  $ sudo sysctl -p
```

Langkah selanjutnya adalah melakukan konfigurasi Keepalived. Untuk dapat menggunakan Keepalived, terdapat 1 (satu) file yang harus dikonfigurasi yaitu file keepalived.conf. File tersebut terdapat di direktori /etc/keepalived. Untuk membuka dan mengkonfigurasi file tersebut dapat dilakukan dengan perintah seperti pada Tabel 5.64.

Tabel 5.64 Perintah Konfigurasi keepalived.conf

```
1.  $ sudo nano /etc/keepalived/keepalived.conf
```

Setelah *file* keepalived.conf terbuka, kemudian lakukan konfigurasi seperti pada Tabel 5.65.

Tabel 5.65 Konfigurasi keepalived.conf pada Loadbalancer1

```
    keepalived.conf
1   vrrp_instance apache2 {
2     Interface enp0s8
3     state MASTER
4     virtual_router_id 15
5     priority 110
6     advert_int 1
7     authentication {
8     auth_type PASS
```

9	auth_pass dian
10	}
11	virtual_ipaddress {
12	192.168.200.26
13	}
14	}

Tabel 5.66 Konfigurasi Keepalived.conf pada Loadbalancer2

	keepalived.conf
1	vrrp_instance apache2 {
2	Interface enp0s8
3	state MASTER
4	virtual_router_id 15
5	priority 109
6	advert_int 1
7	authentication {
8	auth_type PASS
9	auth_pass dian
10	}
11	virtual_ipaddress {
12	192.168.200.26
13	}
14	}

Penjelasan dari konfigurasi keepalived.conf diatas adalah

Baris 1 : mendefinisikan blok contoh VRRP

Baris 2 : menentukan status *instance* dalam penggunaan standard

Baris 3 : virtual_router_id adalah pengenal numerik untuk instance Virtual Router. Konfigurasi ini harus sama pada semua Web Server yang akan dijadikan high availability.

Baris 4 : menentukan prioritas instance di router VRRP. Semakin tinggi angka prioritasnya makan instance tersebut akan dijadikan master

Baris 5 : menentukan interval *advert* dalam hitungan detik

Baris 6 – 9 : menentukn jenis autentikasi yang digunakan dan menentukan password otentikasi

Baris 10 – 12 : menentukan alamat IP virtual

Setelah mengkonfigurasi file keepalived.conf pada semua mesin virtual, langkah selanjutnya adalah melakukan perintah untuk memulai service Keepalived pada semua mesin virtual seperti pada Tabel 5.67

Tabel 5.67 Menjalankan Service Keepalived

```
1   $ sudo service keepalived start
```

Setelah *service* Keepalived berhasil dijalankan, langkah akhir adalah membuka peramban kemudian memasukkan alamat IP virtual Keepalived untuk memeriksa apakah implementasi dan konfigurasi sudah berjalan baik dan benar.

BAB 6 PENGUJIAN

Pengujian pada sistem dilakukan untuk mengetahui performa dan keunggulan masing-masing perangkat lunak *failover* dari skenario pengujian yang berbeda. Ada 2 perancangan topologi yang akan diuji, yaitu pengujian topologi sistem *failover* untuk kebutuhan *high availability web server* dan pengujian topologi sistem *failover* untuk kebutuhan *high availability load balancing*.

6.1 Pengujian Topologi Sistem Failover Untuk Kebutuhan High Availability Web Server

Pengujian pada topologi ini mencakup pengujian fungsional, pengujian *downtime* dan pengujian *failback*.

6.1.1 Pengujian Fungsional

Pengujian fungsional dilakukan untuk mengetahui apakah sistem yang diimplementasi telah berjalan dengan baik dan benar sesuai fungsinya. Pada Tabel 6.1 dan Tabel 6.2 akan ditampilkan hasil dari pengujian fungsional dari topologi *high availability web server*.

Tabel 6.1 Pengujian Fungsional Topologi High Availability Web Server Aplikasi Heartbeat

No	Pengujian	Keterangan
1.	Konten wordpress pada Web Server 1 dapat diakses oleh pengguna	Valid
2.	Konten wordpress pada Web Server 2 dapat diakses oleh pengguna	Valid
3.	Konten wordpress dapat diakses pengguna melalui virtual IP walaupun Web Server 1 dalam kondisi tidak aktif, karena telah diimplementasikan *failover* menggunakan Heartbeat yang dapat mengalihkan *traffic* ketika *server master* sedang tidak aktif.	Valid
4.	Konten wordpress tidak dapat diakses pengguna ketika semua Web Server dalam kondisi tidak aktif.	Valid

Tabel 6.2 Pengujian Fungsional Topologi High Availability Web Server Aplikasi Keepalived

No	Pengujian	Keterangan
1.	Konten wordpress pada Web Server 1 dapat diakses oleh pengguna	Valid

2.	Konten wordpress pada Web Server 2 dapat diakses oleh pengguna	Valid
3.	Konten wordpress dapat diakses pengguna melalui virtual IP walaupun Web Server 1 dalam kondisi tidak aktif, karena telah diimplementasikan *failover* menggunakan Heartbeat yang dapat mengalihkan *traffic* ketika *server master* sedang tidak aktif.	Valid
4.	Konten wordpress tidak dapat diakses pengguna ketika semua Web Server dalam kondisi tidak aktif.	Valid

6.1.2 Pengujian Downtime

Downtime adalah waktu dimana sebuah *resource* untuk sementara tidak dapat diakses ketika sedang terjadi proses *failover*. Pengujian *downtime* dilaksanakan untuk mengetahui berapa lama waktu proses perpindahan *failover* dari server utama yang mengalami kegagalan menuju ke server cadangan. *Resource* akan tetap bisa diakses setelah proses *failover* selesai, karena ada komponen lain yang menjadi *backup* ketika komponen utama mengalami kegagalan.

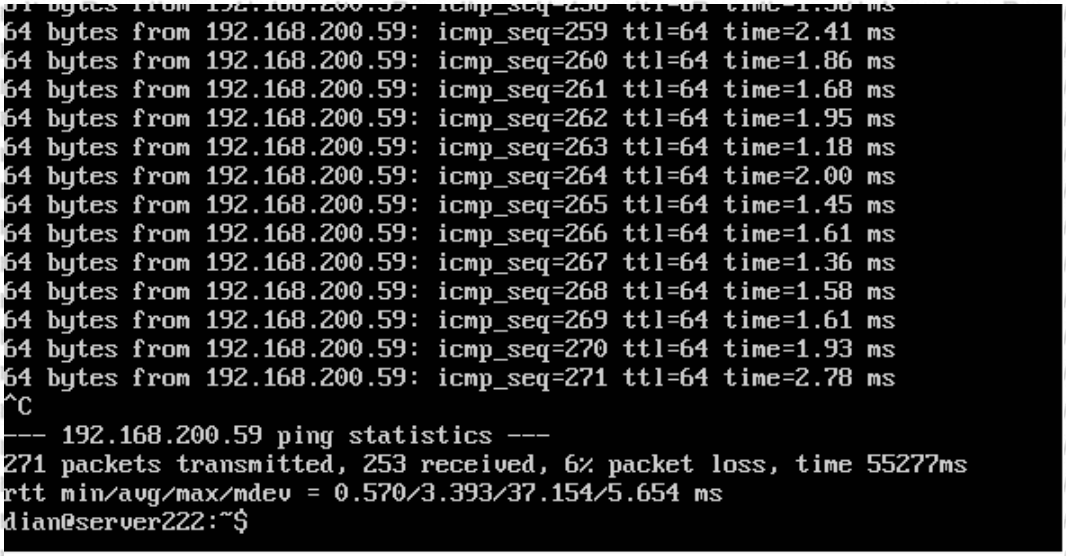
Pengujian *downtime* ini memiliki tujuan untuk mengetahui berapa lama proses yang dibutuhkan perangkat lunak *failover* untuk memindahkan *traffic* dari Web Server 1 ke Web Server 2. Yang menjadi *server master* dalam pengujian ini adalah Web Server 1, sedangkan yang menjadi *server backup* adalah Web Server 2.

Pengujian *downtime* akan dilaksanakan dalam skenario 1, 2 dan 3 untuk mengetahui apakah ada perbedaan waktu *failover* antar skenario. Pengujian *downtime* dilakukan dengan melakukan ping ke alamat IP virtual Heartbeat 192.168.200.59 dan IP virtual Keepalived 192.168.200.60. Untuk dapat mengetahui waktu *downtime* dapat dilakukan dengan perintah seperti pada Tabel 6.3 untuk mengirimkan paket ping dengan interval tiap 0.2 detik.

Tabel 6.3 Perintah Ping

1	`$ ping 192.168.200.59 -i 0.2`

Setelah melakukan ping seperti pada Tabel 6.3**Error! Reference source not found.** maka akan menampilkan hasil seperti pada Gambar 6.1.



Gambar 6.1 Hasil Perintah Ping

Pada Gambar 6.1**Error! Reference source not found.** dapat dilihat bahwa dari 271 paket yang dikirimkan, hanya ada 253 paket yang diterima, jadi ada 18 paket yang hilang. Untuk menghitung waktu *downtime*, jumlah paket yang hilang dikali 0.2 detik, sehingga waktu *downtime*nya adalah 3.6 detik.

6.1.2.1 Pengujian Downtime Topologi High Availability Web Server Aplikasi Heartbeat Skenario 1

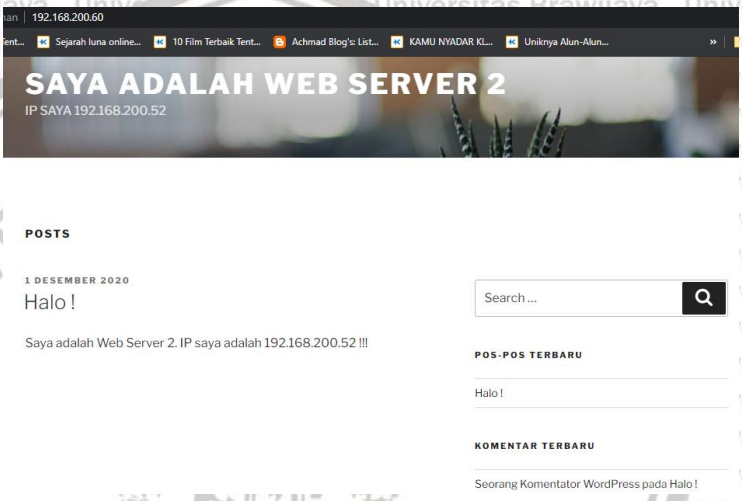
Pengujian *downtime* dilakukan dengan melakukan ping ke alamat IP virtual 192.168.200.60. Pada pengujian *downtime* skenario 1, *server virtual* Web Server 1 akan dimatikan secara sengaja, kemudian akan dihitung waktu *downtime* yang dibutuhkan untuk proses *failover* dari Web Server 1 menuju Web Server2. Hasil dari pengujian *downtime* skenario 1 dijelaskan pada Tabel 6.4.

Tabel 6.4 Pengujian Downtime Topologi High Availability Web Server Aplikasi Hearbeat Skenario 1

Pengujian Ke	Hasil Pengujian (detik)	Rata-rata (detik)
1	6,4	5
2	5,2	
3	4,4	
4	5,4	
5	4,6	
6	5	
7	3,2	
8	5	
9	5	

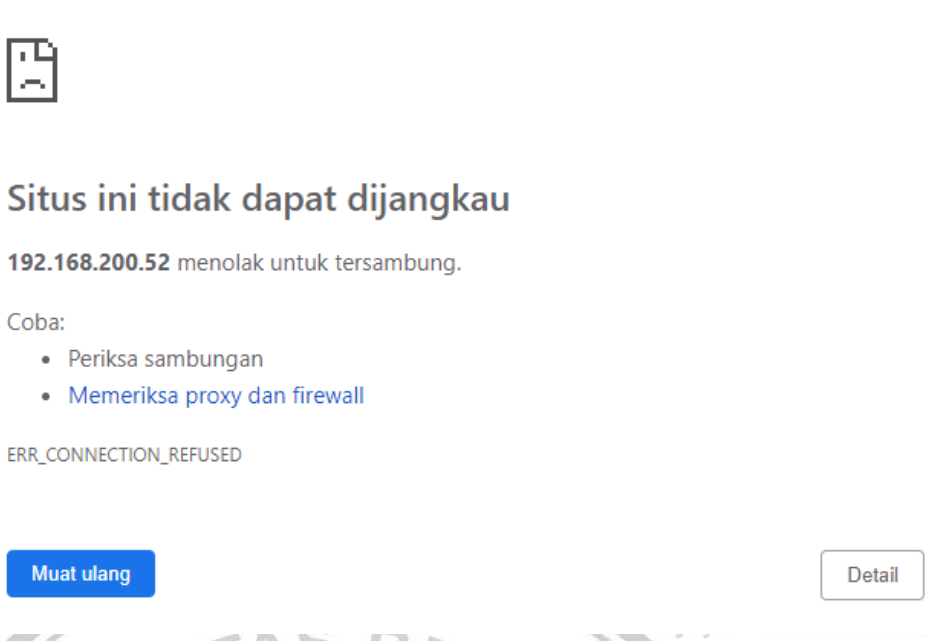
10	5,6	
11	5,6	
12	4,6	

Setelah pengujian ini selesai, diketahui bahwa pada pengujian skenario 1 ini terjadi proses *failover*. *Downtime* terjadi karena adanya perpindahan traffic dari Web Server 1 menuju Web Server 2 sehingga untuk sementara resource tidak dapat diakses oleh pengguna. Hal tersebut dibuktikan dengan adanya packet loss setelah *virtual machine* Web Server 1 dimatikan. Hasilnya adalah proses *failover* berkisar antara 3,2 detik hingga 6,4 detik. Hasil dari pengujian mengakses alamat IP virtual Heartbeat setelah Web Server 1 dimatikan ditampilkan pada Gambar 6.2.



Gambar 6.2 Mengakses IP Virtual Heartbeat Saat Web Server 1 Dalam Kondisi Mati

Dari Gambar 6.2 dapat ditarik kesimpulan Web Server 2 akan menangani *traffic* yang masuk saat Web Server 1 dalam kondisi mati. Pada saat pengujian, ditemukan fakta bahwa ketika semua *server* dalam kondisi aktif, pengguna tidak dapat mengakses alamat IP Web Server 2 192.168.200.52 pada *browser* walaupun Web Server 2 dalam kondisi aktif. Ketika mencoba mengakses alamat tersebut, muncul pesan *error* seperti pada Gambar 6.3.



Gambar 6.3 Pesan Error Saat Mengakses IP Web Server 2

Dan ketika Web Server 1 dalam kondisi mati, alamat IP Web Server 2 192.168.200.52 dapat diakses langsung pada browser, tidak ada pesan *error* seperti yang terjadi ketika semua server dalam kondisi aktif.

6.1.2.2 Pengujian Downtime Topologi High Availability Web Server Aplikasi Keepalived Skenario 1

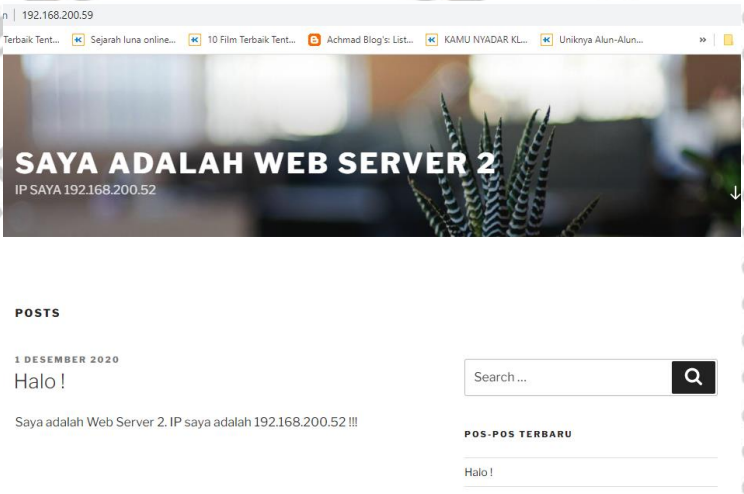
Pengujian ini dilakukan dengan melakukan ping ke alamat IP 192.168.200.59. Pada pengujian *downtime* skenario 1, server virtual Web Server 1 akan dimatikan secara sengaja, kemudian akan dihitung waktu *downtime* yang dibutuhkan untuk proses *failover* dari Web Server 1 ke Web Server2. Hasil dari pengujian *downtime* skenario 1 dijelaskan pada Tabel 6.5.Tabel 6.4

Tabel 6.5 Pengujian Downtime Topologi High Availability Web Server Aplikasi Keepalived Skenario 1

Pengujian Ke	Hasil Pengujian (detik)	Rata-rata (detik)
1	3.6	4
2	4.2	
3	4	
4	4	
5	4.2	
6	4	
7	4.2	
8	4.2	

9	3.6	
10	4	
11	4	
12	4	

Setelah selesai melaksanakan pengujian ini, diketahui bahwa pada pengujian skenario 1 ini terjadi proses *failover*. Hal ini dibuktikan dari adanya *packet loss* sehingga menyebabkan *downtime* dan pengujian dengan mengakses alamat IP virtual Keepalived setelah Web Server 1 mati. Hasil pengujian *downtime* memiliki hasil *downtime* antara 3,6 detik hingga 4,2 detik. Hasil dari pengujian mengakses alamat IP virtual Keepalived setelah Web Server 1 dimatikan ditampilkan pada Gambar 6.4.



Gambar 6.4 Mengakses Alamat IP Virtual Keepalived Setelah Web Server 1 Mati

Dari Gambar 6.4Gambar 6.2 dapat ditarik kesimpulan Web Server 2 menangani *traffic* yang masuk saat Web Server 1 dalam kondisi mati. Waktu *downtime* terjadi karena adanya proses *failover* yang memindahkan *traffic* dari Web Server 1 menuju Web Server 2 sehingga *resource* untuk sementara waktu tidak dapat diakses. Rata-rata pengujian waktu *downtime* sebesar 4 detik.

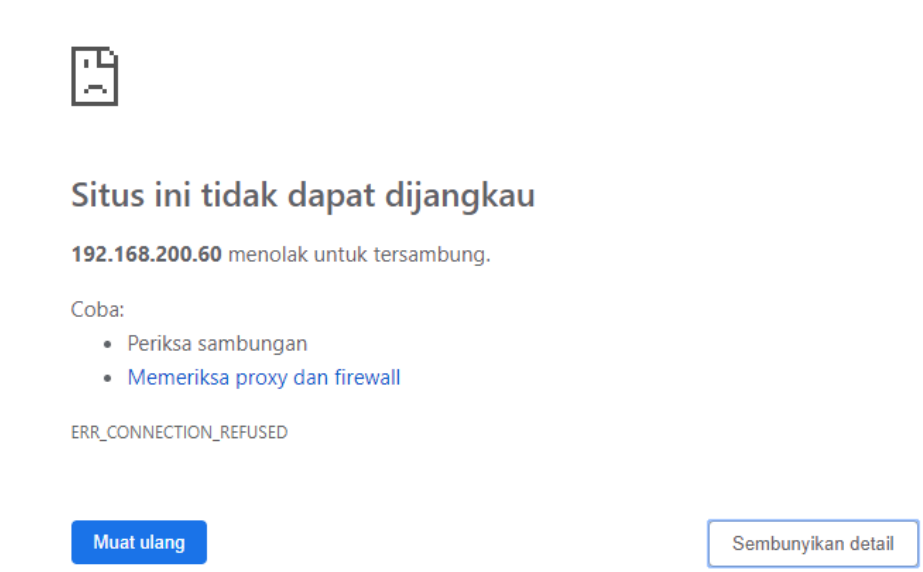
6.1.2.3 Pengujian Downtime Topologi High Availability Web Server Aplikasi Heartbeat Skenario 2

Pengujian ini dilakukan dengan melakukan ping ke alamat IP virtual 192.168.200.60. Pada pengujian *downtime* skenario 2, *service* Apache2 *web server* pada *server virtual* Web Server 1 akan dimatikan secara sengaja, kemudian akan dihitung hasil waktu proses *failover* dari Web Server 1 ke Web Server 2. Hasil dari pengujian *downtime* skenario 2 dijelaskan pada Tabel 6.6.

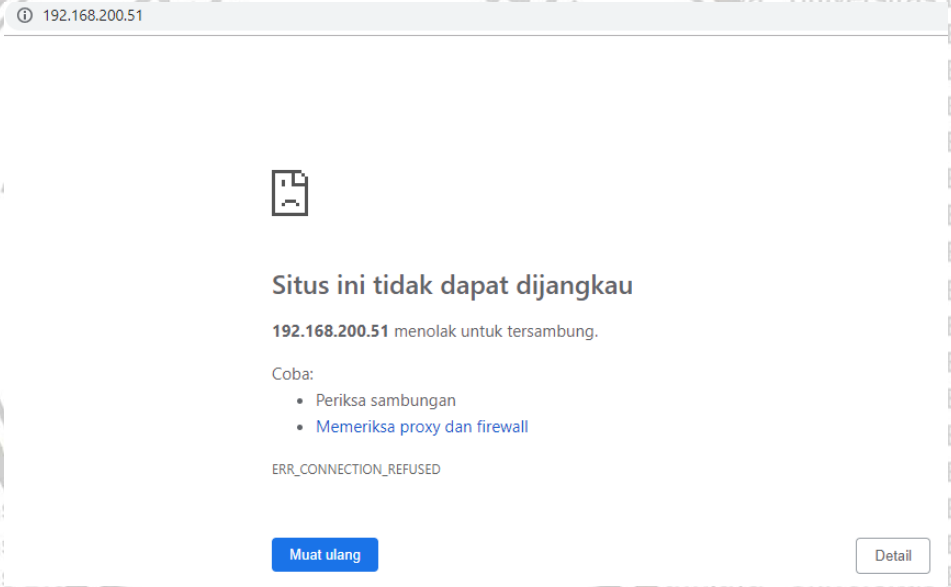
Tabel 6.6 Pengujian Downtime Topologi High Availability Web Server Aplikasi Heartbeat Skenario 2

Pengujian Ke	Hasil Pengujian (detik)	Rata-rata (detik)
1.	0	0
2.	0	
3.	0	
4.	0	
5.	0	
6.	0	
7.	0	
8.	0	
9.	0	
10.	0	
11.	0	
12.	0	

Setelah selesai melaksanakan pengujian ini, diketahui bahwa pada pengujian skenario 2 ini tidak terjadi proses *failover* yang dapat dibuktikan dari hasil pengujian tidak ada *packet loss* dan percobaan untuk mengakses alamat IP virtual Heartbeat, alamat IP Web Server 1 dan Alamat IP Web Server 2 setelah service Apache2 pada Web Server 1 dalam kondisi mati. Hasil pengujian *downtime* memiliki hasil *packet loss* sebesar 0 detik di pengujian ke 1 sampai dengan pengujian ke 12. Percobaan mengakses alamat IP virtual Heartbeat, IP Web Server 1 dan IP Web Server 2 setelah *service* Apache2 pada Web Server 1 dimatikan akan ditampilkan masing-masing pada Gambar 6.5, Gambar 6.6 dan Gambar 6.7



Gambar 6.5 Mengakses IP virtual Saat Service Apache2 Mati



Gambar 6.6 Mengakses IP Web Server 1 Saat Service Apache2 Pada Web Server 1 Mati

Gambar 6.7 Mengakses IP Web Server 2 Ketika Service Apache2 Pada Web Server 1 Mati

Hasil dari pengujian mengakses alamat IP virtual Heartbeat, menampilkan pesan *error* seperti pada Gambar 6.5, yang dimana pesan error ini muncul karena Heartbeat tidak memindahkan traffic yang masuk ke Web Server 2. Hal tersebut terjadi karena aplikasi Heartbeat mendeteksi bahwa Web Server 1 dalam kondisi aktif walaupun *service* Apache2 dalam kondisi mati. *Error* yang muncul ketika mencoba mengakses alamat IP Web Server 1 seperti pada Gambar 6.6 memiliki arti *service* Apache2 dalam Web Server dalam kondisi mati. Ketika mengakses alamat IP Web Server 2 tidak mengalami *error* karena service Apache2 dalam kondisi aktif. Dari ketiga gambar tersebut dapat disimpulkan dalam pengujian ini tidak terjadi proses *failover* karena *traffic* yang masuk tetap dilayani oleh Web Server 1. Jika pada pengujian ini terjadi proses *failover*, mengakses alamat IP virtual Heartbeat 192.168.200.60 seharusnya muncul konten Wordpress dari Web Server 2.

6.1.2.4 Pengujian Downtime Topologi High Availability Web Server Aplikasi Keepalived Skenario 2

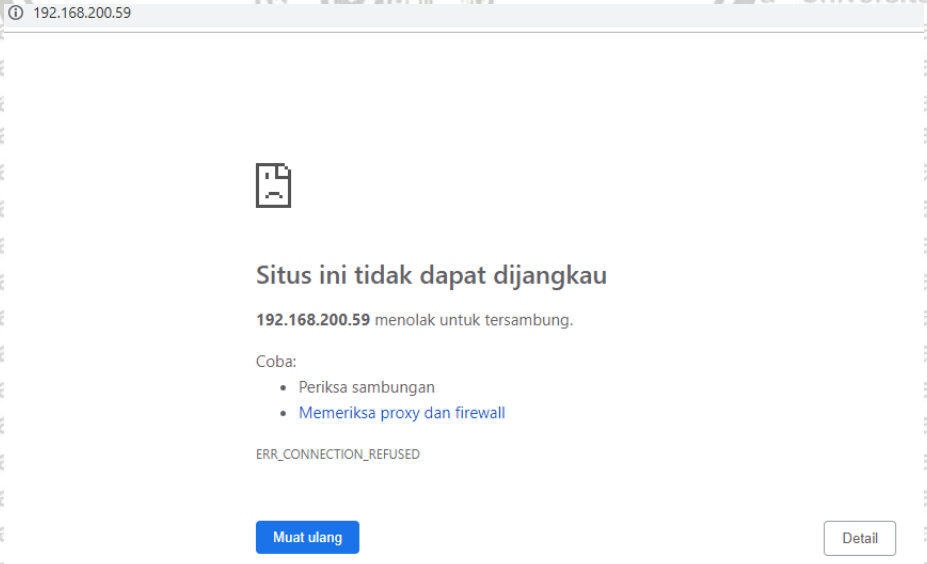
Pengujian *downtime* dilakukan dengan melakukan ping ke alamat IP virtual 192.168.200.59. Pada pengujian downtime skenario 2, service Apache2 *web server* pada *server virtual* Web Server 1 akan dimatikan secara sengaja, kemudian akan dihitung hasil waktu proses *failover* dari Web Server 1 ke Web Server 2. Hasil dari pengujian *downtime* skenario 2 dijelaskan pada Tabel 6.7.

Tabel 6.7 Pengujian Downtime Topologi High Availability Web Server Aplikasi Keepalived Skenario 2

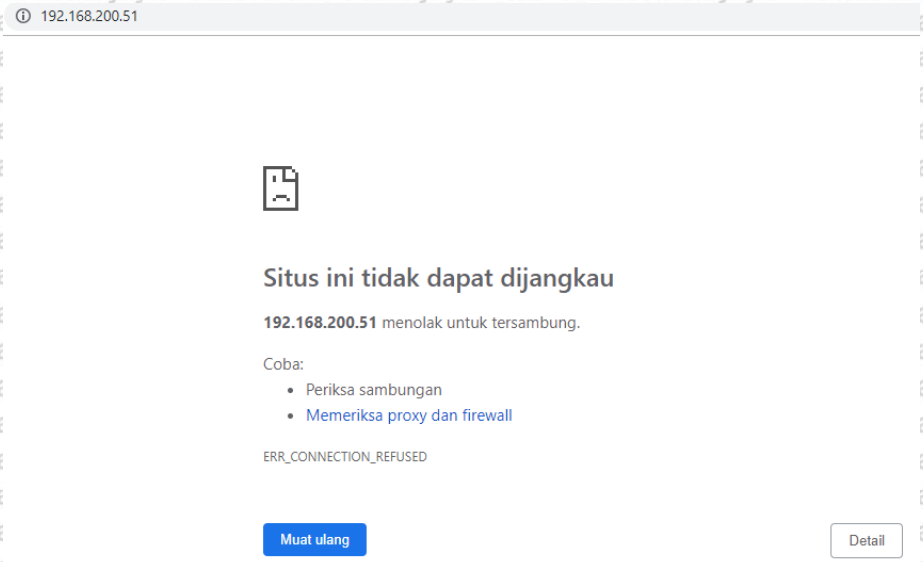
Pengujian Ke	Hasil Pengujian (detik)	Rata-rata (detik)

1.	0	
2.	0	
3.	0	
4.	0	
5.	0	
6.	0	0
7.	0	
8.	0	
9.	0	
10.	0	
11.	0	
12.	0	

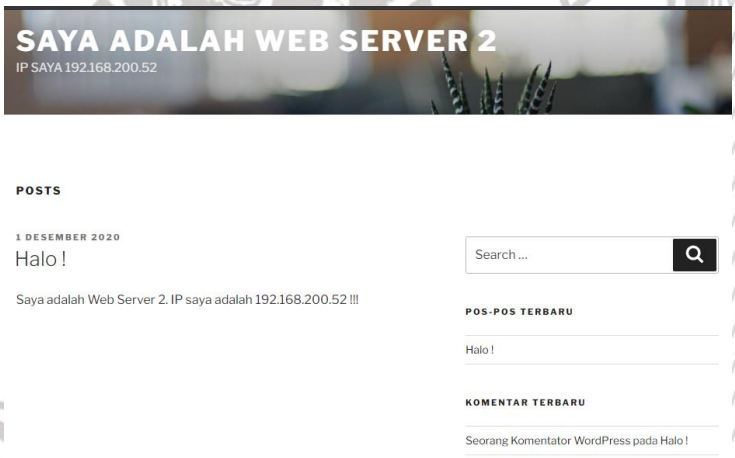
Setelah selesai melaksanakan pengujian ini, diketahui bahwa pada pengujian skenario 2 ini tidak terjadi proses *failover* yang dapat dibuktikan dari hasil pengujian tidak ada *packet loss* dan percobaan untuk mengakses alamat IP virtual Keepalived, alamat IP Web Server 1 dan Alamat IP Web Server 2 setelah service Apache2 pada Web Server 1 dalam kondisi mati. Hasil pengujian *downtime* memiliki hasil *packet loss* sebesar 0 detik di pengujian ke 1 sampai dengan pengujian ke 12. Percobaan mengakses alamat IP virtual Keepalived, IP Web Server 1 dan IP Web Server 2 setelah *service* Apache2 pada Web Server 1 dimatikan akan ditampilkan masing-masing pada Gambar 6.8, Gambar 6.9 dan Gambar 6.10.



Gambar 6.8 Mengakses IP Virtual Keepalived Ketika Service Apache2 Pada Web Server 1 Mati



Gambar 6.9 Mengakses IP Web Server 1Ketika Service Apache2 Pada Web Server 1 Mati



Gambar 6.10 Mengakses IP Web Server 2 Ketika Service Apache2 Pada Web Server 1 Mati

Hasil dari pengujian mengakses alamat IP virtual Keepalived, menampilkan pesan *error* seperti pada Gambar 6.8 yang dimana pesan error ini muncul karena Keepalived tidak memindahkan *traffic* yang masuk ke Web Server 2. Hal tersebut terjadi karena aplikasi Keepalived mendeteksi bahwa Web Server 1 dalam kondisi aktif walaupun *service* Apache2 dalam kondisi mati. *Error* yang muncul ketika mencoba mengakses alamat IP Web Server 1 seperti pada Gambar 6.9 memiliki arti *service* Apache2 dalam Web Server dalam kondisi mati. Ketika mengakses alamat IP Web Server 2 tidak mengalami *error* karena *service* Apache2 dalam

kondisi aktif. Dari ketiga gambar tersebut dapat disimpulkan dalam pengujian ini tidak terjadi proses *failover* karena *traffic* yang masuk tetap dilayani oleh Web Server 1. Jika pada pengujian ini terjadi proses *failover*, mengakses alamat IP virtual Keepalived 192.168.200.59 seharusnya muncul konten Wordpress dari Web Server 2.

6.1.2.5 Pengujian Downtime Topologi High Availability Web Server Aplikasi Heartbeat Skenario 3

Pengujian *downtime* dilakukan dengan melakukan ping ke alamat IP virtual Heartbeat. Pada pengujian *downtime* skenario 3, service perangkat lunak *failover* Heartbeat pada *server virtual* Web Server 1 akan dimatikan secara sengaja, kemudian akan dihitung *downtime* dari proses *failover* dari Web Server 1 ke Web Server 2. Hasil dari pengujian *downtime* skenario 3 dijelaskan pada Tabel 6.8.

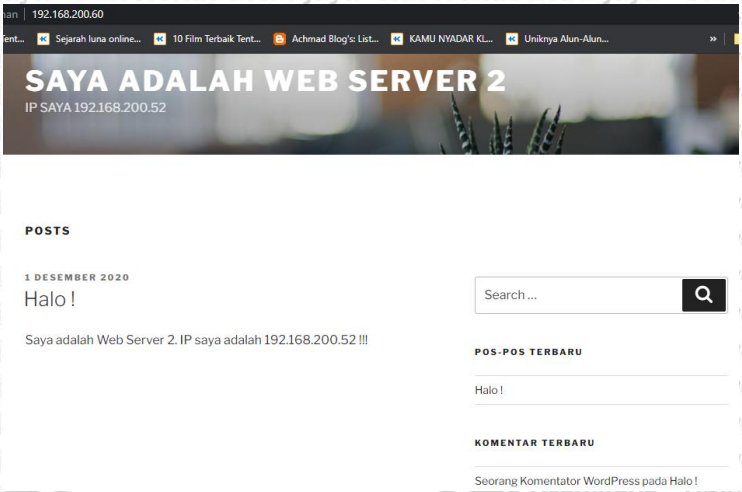
Tabel 6.8 Pengujian Downtime Topologi High Availability Web Server Aplikasi Heartbeat Skenario 3

Pengujian Ke	Hasil Pengujian (detik)	Rata-rata (detik)
1	0.6	0,88
2	0.6	
3	0.8	
4	0.8	
5	1	
6	1	
7	1.2	
8	0.8	
9	1	
10	1.2	
11	1	
12	0.6	

Setelah selesai melaksanakan pengujian ini, diketahui bahwa pada pengujian topologi *high availability web server* skenario 3 ini terjadi proses *failover*. Hal ini dibuktikan dari hasil pengujian *downtime* dan pengujian mengakses alamat virtual Heartbeat. Hasil pengujian *downtime* memiliki *downtime* berkisar antara 0,6 detik hingga 1,2 detik. Hasil rata-rata waktu *downtime* dari pengujian ini sebesar 0,88 detik.

Adanya waktu *downtime* terjadi karena proses *failover* yang memindahkan *traffic* dari Web Server 1 ke Web Server2. *Traffic* berpindah karena *service* Heartbeat pada Web Server 1 mati. Hasil pengujian mengakses alamat IP virtual

Heartbeat setelah *service* Heartbeat pada Web Server 1 dimatikan, ditampilkan pada Gambar 6.11.



Gambar 6.11 Mengakses IP Virtual Heartbeat setelah Service Heartbeat Pada Web Server 1 Mati

Dari Gambar 6.11 dapat ditarik kesimpulan bahwa Web Server 2 menangani *traffic* yang masuk ketika *service* Heartbeat pada Web Server 1 mati.

6.1.2.6 Pengujian Downtime Topologi High Availability Web Server Aplikasi Keepalived Skenario 3

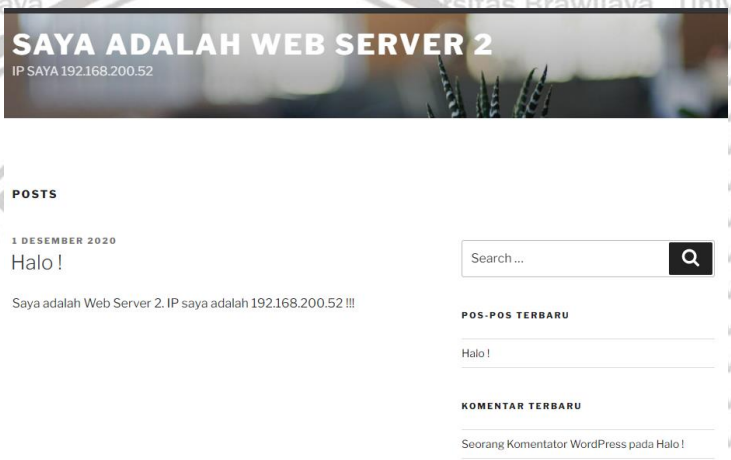
Pengujian *downtime* dilakukan dengan melakukan ping ke alamat IP virtual 192.168.200.59. Pada pengujian *downtime* skenario 3, *service* Keepalived pada *server* virtual Web Server 1 akan dimatikan secara sengaja, kemudian akan dihitung hasil waktu proses *failover* dari Web Server 1 menuju Web Server 2. Hasil dari pengujian *downtime* skenario 3 dijelaskan pada Tabel 6.9.

Tabel 6.9 Pengujian Downtime Topologi High Availability Web Server Aplikasi Keepalived Skenario 3

Pengujian Ke	Hasil Pengujian (detik)	Rata-rata (detik)
1	1.4	1,51
2	1.6	
3	1,6	
4	1.4	
5	1.6	
6	1.4	
7	1.6	
8	1.6	
9	1.4	

10	1.6	
11	1.6	
12	1.4	

Setelah selesai melaksanakan pengujian ini, diketahui bahwa pada pengujian skenario 3 ini terjadi proses *failover*. Hal ini dapat dibuktikan dari adanya *packet loss* pengujian mengakses alamat IP virtual Keepalived setelah *service* Keepalived pada Web Server 1 dimatikan. Hasil pengujian ini memiliki hasil *downtime* antara 1,4 detik hingga 1,6 detik. Hasil rata-rata waktu *downtime* dari pengujian ini sebesar 1,51 detik. Pengujian mengakses alamat IP virtual Keepalived setelah *service* Keepalived pada Web Server 1 dimatikan akan ditampilkan pada Gambar 6.12.



Gambar 6.12 Mengakses Alamat IP Virtual Keepalived Setelah Service Keepalived Pada Web Server 1 Mati

Dari Gambar 6.12 dapat ditarik kesimpulan bahwa Web Server 2 menangani *traffic* setelah service Keepalived pada Web Server 1 dimatikan. Adanya waktu *downtime* terjadi karena proses *failover* yang memindahkan *traffic* dari Web Server 1 ke Web Server 2.

6.1.3 Pengujian Failback

Failback adalah sebuah proses kembalinya server *master* kedalam sebuah klaster, yang sebelumnya server *master* mengalami kegagalan sehingga server backup yang menangani *traffic*. Dalam pengujian perancangan topologi ini, yg menjadi server *master* adalah Web Server 1 dan server *backup* adalah Web Server 2.

Pengujian *failback* dilakukan untuk mengetahui berapa lama waktu proses *failback* yang diperlukan untuk proses perpindahan *traffic* dari Web Server 2 ke Web Server 1 yang baru saja menyala. Pengujian *failback* akan dilakukan pada skenario 1, 2 dan 3 untuk mengetahui apakah ada perbedaan waktu *failback* antar skenario. Pengujian *failback* dilakukan dengan melakukan ping ke alamat IP virtual

Heartbeat 192.168.200.60 dan IP virtual Keepalived 192.168.200.59. Untuk dapat mengetahui waktu *failback* dapat dilakukan dengan perintah seperti pada Tabel 6.3 untuk mengirimkan paket ping dengan interval tiap 0.2 detik.

Tabel 6.10 Perintah Ping

1	$ ping 192.168.200.59 -i 0.2

Setelah melakukan ping seperti pada Tabel 6.10**Error! Reference source not found.** maka akan menampilkan hasil seperti pada Gambar 6.13.

Gambar 6.13 Hasil Perintah Ping

```
64 bytes from 192.168.200.59: icmp_seq=259 ttl=64 time=2.41 ms
64 bytes from 192.168.200.59: icmp_seq=260 ttl=64 time=1.86 ms
64 bytes from 192.168.200.59: icmp_seq=261 ttl=64 time=1.68 ms
64 bytes from 192.168.200.59: icmp_seq=262 ttl=64 time=1.95 ms
64 bytes from 192.168.200.59: icmp_seq=263 ttl=64 time=1.18 ms
64 bytes from 192.168.200.59: icmp_seq=264 ttl=64 time=2.00 ms
64 bytes from 192.168.200.59: icmp_seq=265 ttl=64 time=1.45 ms
64 bytes from 192.168.200.59: icmp_seq=266 ttl=64 time=1.61 ms
64 bytes from 192.168.200.59: icmp_seq=267 ttl=64 time=1.36 ms
64 bytes from 192.168.200.59: icmp_seq=268 ttl=64 time=1.58 ms
64 bytes from 192.168.200.59: icmp_seq=269 ttl=64 time=1.61 ms
64 bytes from 192.168.200.59: icmp_seq=270 ttl=64 time=1.93 ms
64 bytes from 192.168.200.59: icmp_seq=271 ttl=64 time=2.78 ms
^C
--- 192.168.200.59 ping statistics ---
271 packets transmitted, 253 received, 6% packet loss, time 55277ms
rtt min/avg/max/mdev = 0.570/3.393/37.154/5.654 ms
dian@server222:~$
```

Gambar 6.14 Hasil Perintah Ping

Pada Gambar 6.14 dapat dilihat bahwa dari 271 paket yang dikirimkan, hanya ada 253 paket yang diterima, jadi ada 18 paket yang hilang. Untuk menghitung waktu *failback*, jumlah paket yang hilang dikali 0.2 detik, sehingga waktu failbacknya adalah 3.6 detik.

6.1.3.1 Pengujian Failback Topologi High Availabilty Web Server Aplikasi Heartbeat Skenario 1

Pengujian *failback* dilakukan dengan melakukan ping ke alamat IP virtual 192.168.200.60. Pada pengujian *failback* skenario 1, *server virtual* Web Server 1 akan dihidupkan kembali pada saat server virtual Web Server 2 sedang menangani *traffic* yang masuk, kemudian akan dihitung waktu proses *failback* dari Web Server 2 ke Web Server 1. Hasil dari pengujian *failback* skenario 1 dijelaskan pada Tabel 6.11.

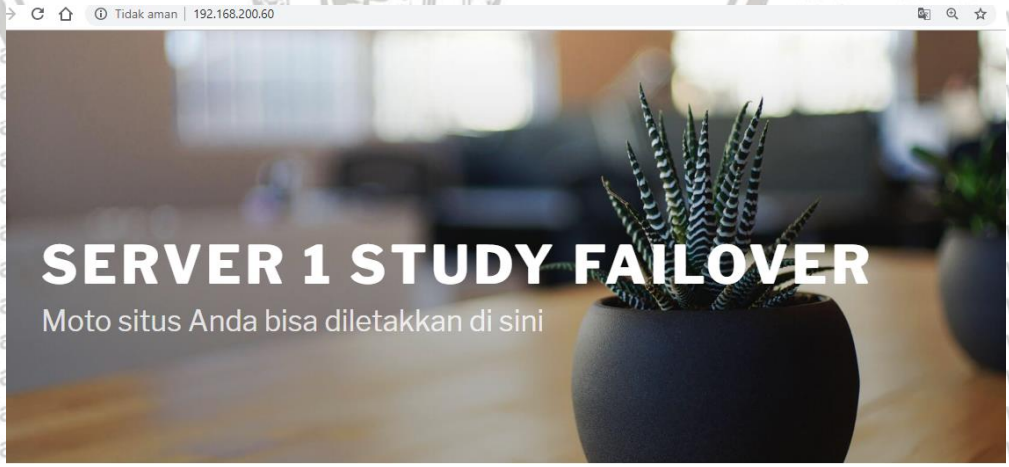
Tabel 6.11 Pengujian Failback Topologi High Availability Web Server Aplikasi Heartbeat Skenario 1

Pengujian Ke	Hasil Pengujian (detik)	Rata-rata (detik)

1	1	
2	0.6	
3	0.8	
4	0.6	
5	0.6	
6	0.6	0,68
7	0.4	
8	0.6	
9	0.8	
10	0.6	
11	0.6	
12	1	

Setelah selesai melaksanakan pengujian ini, diketahui bahwa pada pengujian skenario 1 ini terjadi proses *failback*. Hal ini dibuktikan dari hasil pengujian *failback* dan dilakukan percobaan mengakses alamat IP virtual Heartbeat pada *browser* setelah Web Server 1 kembali dihidupkan.

Hasil pengujian *failback* antara 0 detik hingga 0,8 detik. Adanya p*acket loss* karena adanya proses *failback* yang memindahkan *traffic* dari Web Server 2 menuju Web Server 1. Hasil rata-rata waktu *failback* pada pengujian ini sebesar 0,68 detik. Percobaan mengakses alamat IP virtual pada browser setelah proses *failback* pada Gambar 6.15.



Gambar 6.15 Mengakses IP Virtual Heartbeat Setelah Web Server 1 Kembali Aktif

Dari Gambar 6.15 dapat disimpulkan bahwa terjadi proses *failback*, karena saat mengakses alamat IP virtual Heartbeat pada saat Web Server 1 dalam kondisi mati,

konten yang tampil adalah konten dari Web Server 2, seperti yang ditampilkan pada Gambar 6.2.

6.1.3.2 Pengujian Failback Topologi High Availability Web Server Aplikasi Keepalived Skenario 1

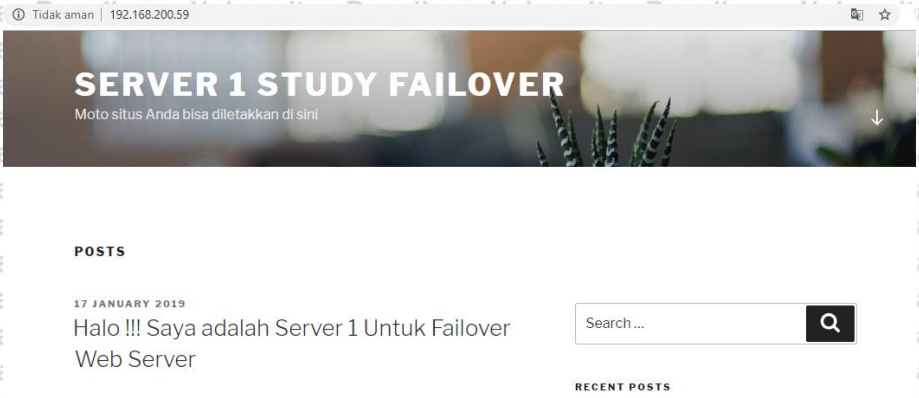
Pengujian *failback* dilakukan dengan melakukan ping ke alamat IP *virtual* Keepalived 192.168.200.59. Pada pengujian *failback* skenario 1, *server virtual* Web Server 1 akan dinyalakan kembali, yang dimana sebelumnya pada kondisi mati. *Traffic* yang masuk ditangani oleh Web Server 2. Setelah Web Server 1 menyala, kemudian akan dihitung waktu proses *failback* dari Web Server 2 ke Web Server 1. Hasil dari pengujian *failback* skenario 1 dijelaskan pada Tabel 6.12.

Tabel 6.12 Pengujian Failback Topologi High Availability Web Server Aplikasi Keepalived Skenario 1

Pengujian Ke	Hasil Pengujian (detik)	Rata-rata (detik)
1	1	0,98
2	1	
3	1	
4	1	
5	0.8	
6	1	
7	1	
8	1	
9	1	
10	1	
11	1	
12	1	

Setelah selesai melaksanakan pengujian ini, diketahui bahwa pada pengujian skenario 1 ini terjadi proses *failback*. Hal ini dibuktikan dari hasil pengujian *failback* dan dilakukan percobaan mengakses alamat IP virtual Keepalived pada *browser* setelah Web Server 1 kembali dihidupkan.

Hasil pengujian *failback* memiliki hasil antara 0,8 hingga 1 detik. *Packet loss* terjadi karena adanya proses *failback* yang memindahkan *traffic* dari Web Server 2 menuju Web Server 1. Hasil rata-rata waktu *failback* pada pengujian ini sebesar 0,98 detik. Hasil dari pengujian mengakses alamat IP virtual Keepalived setelah proses *failback* ditampilkan pada Gambar 6.16.



Gambar 6.16 Mengakses IP Virtual Keepalived Setelah Web Server 1 Kembali Aktif

Dari Gambar 6.16Gambar 6.15 dapat disimpulkan bahwa proses *failback* benar terjadi, karena pada pengujian mengakses alamat IP virtual Keepalived saat Web Server 1 dalam kondisi mati menampilkan konten dari Web Server 2, seperti yang ditampilkan pada gambar Gambar 6.4.

6.1.3.3 Pengujian Failback Topologi High Availability Web Server Alikasi Heartbeat Skenario 2

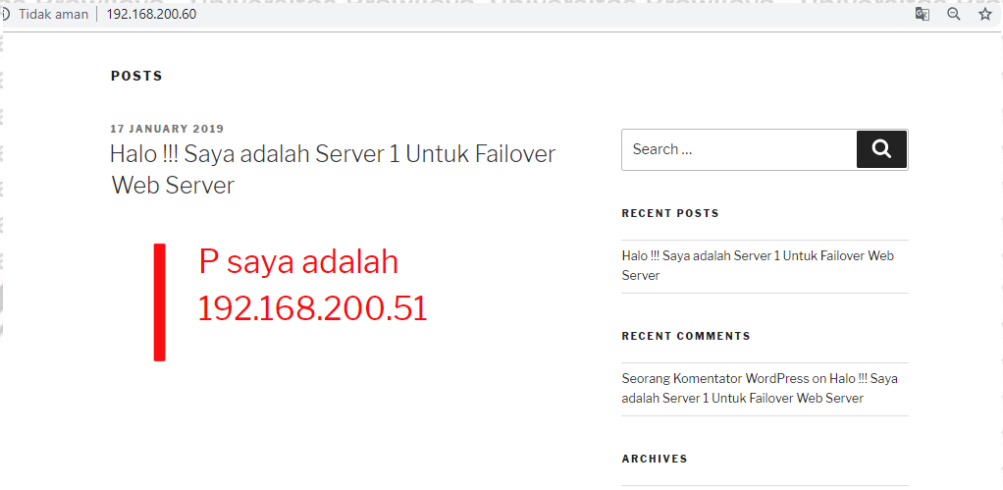
Pengujian *failback* dilakukan dengan melakukan ping ke alamat IP *virtual* Heartbeat 192.168.200.59. Pada pengujian *failback* skenario 2, *service* Apache2 pada Web Server 1 akan dihidupkan kembali yang dimana sebelumnya *service* tersebut dalam kondisi tidak aktif. Kemudian akan dihitung waktu proses *failback*nya. Hasil dari pengujian *failback* skenario 2 dijelaskan pada Tabel 6.13.

Tabel 6.13 Pengujian Failback Topologi High Availability Web Server Aplikasi Heartbeat Skenario 2

Pengujian Ke	Hasil Pengujian (detik)	Rata-rata (detik)
1	0	0
2	0	
3	0	
4	0	
5	0	
6	0	
7	0	
8	0	
9	0	
10	0	
11	0	

12	0	

Setelah selesai melaksanakan pengujian ini, diketahui bahwa pada pengujian skenario 2 ini tidak terjadi proses *failback*. Proses *failback* tidak terjadi karena hasil pengujian menampilkan hasil 0 detik pada semua pengujian dan Ketika dilakukan percobaan mengakses alamat IP virtual, konten tidak muncul dan menampilkan pesan error seperti pada Gambar 6.5. Percobaan mengakses alamat IP virtual Heartbeat setelah *service* Apache2 pada Web Server 1 diaktifkan ditampilkan pada Gambar 6.17.



Gambar 6.17 Mengakses Alamat IP Virtual Heartbeat Saat Service Apache2 Pada Web Server 1 Aktif

Proses *failback* tidak terjadi karena pada pengujian *downtime* skenario 2 yang telah dilakukan sebelumnya, tidak terjadi proses *failover* yang memindahkan traffic dari Web Server 1 ke Web Server 2 ketike service Apache2 pada Web Server 1 dimatikan.

Hasil dari pengujian Gambar 6.17 memang menampilkan konten Wordpress dari Web Server 1, namun tidak ada proses failback yang terjadi karena munculnya konten Wordpress semata-mata karena *service* Apache2 pada Web Server 1 yang kembali dijalankan.

6.1.3.4 Pengujian Failback Topologi High Availability Web Server Aplikasi Keepalived Skenario 2

Pengujian *failback* dilakukan dengan melakukan ping ke alamat IP virtual Keepalived 192.168.200.59. Pada pengujian *failback* skenario 2, service Apache2 pada Web Server 1 akan dihidupkan kembali yang sebelumnya *service* tersebut dalam kondisi tidak aktif. Kemudian akan dihitung waktu proses *failback*. Hasil dari pengujian *failback* skenario 2 dijelaskan pada Tabel 6.14.

Tabel 6.14 Pengujian Failback Topologi High Availability Web Server Aplikasi Keepalived Skenario 2

Pengujian Ke	Hasil Pengujian (detik)	Rata-rata (detik)
1	0	0
2	0	
3	0	
4	0	
5	0	
6	0	
7	0	
8	0	
9	0	
10	0	
11	0	
12	0	

Setelah selesai melaksanakan pengujian ini, diketahui bahwa pada pengujian skenario 2 ini tidak terjadi proses *failback*. Proses *failback* tidak terjadi karena hasil pengujian menampilkan hasil 0 detik pada semua pengujian dan ketika dilakukan percobaan mengakses alamat IP virtual, konten tidak muncul dan menampilkan pesan *error* seperti pada Gambar 6.8. Percobaan mengakses alamat IP virtual Heartbeat setelah *service* Apache2 pada Web Server 1 diaktifkan ditampilkan pada Gambar 6.18.

Gambar 6.18 Mengakses IP Virtual Keepalived Setelah Service Apache2 Pada Web Server 1 Kembali Aktif

Proses *failback* tidak terjadi karena pada pengujian *downtime* skenario 2 yang telah dilakukan sebelumnya tidak terjadi proses *failover* yang memindahkan *traffic* dari Web Server 1 ke Web Server 2 ketike *service* Apache2 pada Web Server 1 dimatikan.

Hasil dari pengujian Gambar 6.18Gambar 6.17 memang menampilkan konten Wordpress dari Web Server 1, namun tidak ada proses *failback* yang terjadi karena munculnya konten Wordpress semata-mata karena *service* Apache2 pada Web Server 1 yang kembali dijalankan.

6.1.3.5 Pengujian Failback Topologi High Availability Web Server Aplikasi Heartbeat Skenario 3

Pengujian *failback* dilakukan dengan melakukan ping ke alamat IP virtual Heartbeat 192.168.200.60. Pada pengujian *failback* skenario 3, *service* perangkat lunak Heartbeat pada *server virtual* Web Server 1 akan dihidupkan kembali dari kondisi sebelumnya yang sedang mati. Sehingga, Web Server 1 akan mengambil alih *traffic* dari server *backup* yaitu Web Server 2. Kemudian waktu proses *failback* perpindahan *traffic* dari Web Server 2 ke Web Server 1 dihitung. Hasil dari pengujian *failback* skenario 3 dijelaskan pada Tabel 6.15Tabel 6.13.

Tabel 6.15 Pengujian Failback Topologi High Availability Web Server Aplikasi Heartbeat Skenario 3

Pengujian Ke	Hasil Pengujian (detik)	Rata-rata (detik)
1	1	0,88
2	0.8	
3	1	
4	0.8	
5	1	
6	1	
7	0.6	
8	1	
9	0.6	
10	0.8	
11	0.6	
12	0.4	

Setelah selesai melaksanakan pengujian ini, diketahui bahwa pada pengujian skenario 3 ini terjadi proses *failback*. Hal ini dibuktikan dari hasil pengujian dan dilakukan percobaan mengakses alamat IP virtual Heartbeat pada *browser* setelah *service* perangkat lunak Heartbeat pada Web Server 1 kembali dihidupkan.

Hasil pengujian *failback* memiliki hasil antara 0,6 detik hingga 1 detik. *Packet loss* terjadi karena adanya proses *failback* yang memindahkan *traffic* dari Web Server 2 menuju Web Server 1. Hasil rata-rata waktu *failback* pada pengujian ini sebesar 0,88 detik. Hasil dari pengujian mengakses alamat IP virtual Heartbeat setelah proses *failback* ditampilkan pada Gambar 6.19.

Gambar 6.19 Mengakses IP Virtual Heartbeat Setelah Service Heartbeat Pada Web Server 1 Aktif

Dari Gambar 6.19 dapat disimpulkan bahwa proses *failback* benar terjadi, karena pada percobaan mengakses alamat IP virtual Heartbeat saat *service* Heartbeat pada Web Server 1 sedang dalam kondisi mati,konten yang tampil merupakan konten dari Web Server 2, seperti yang ditampilkan pada Gambar 6.11.

6.1.3.6 Pengujian Failback Topologi High Availability Web Server Aplikasi Keepalived Skenario 3

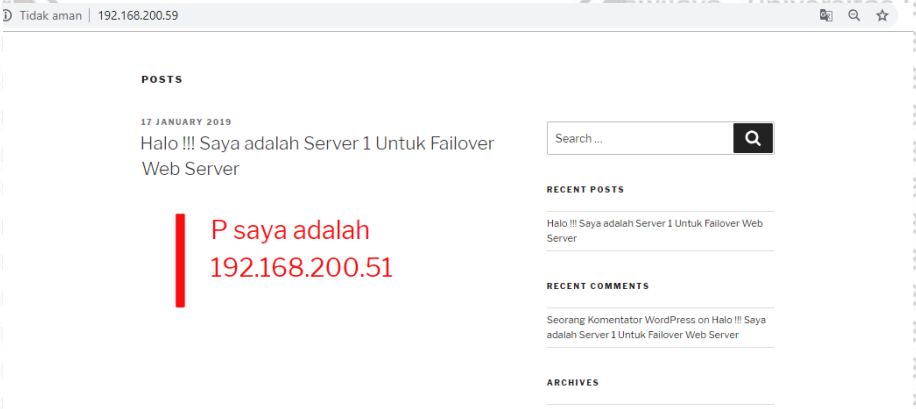
Pengujian *failback* dilakukan dengan melakukan ping ke alamat IP virtual Keepalived 192.168.200.59. Pada pengujian *failback* skenario 3, *service* perangkat lunak Keepalived pada *server virtual* Web Server 1 akan dihidupkan dari kondisi yang sedang mati. Sehingga, Web Server 1 akan mengambil alih *traffic* dari *server backup* yaitu Web Server 2. Kemudian waktu proses *failback* perpindahan *traffic* dari Web Server 2 ke Web Server 1 dihitung. Hasil dari pengujian *failback* skenario 3 dijelaskan pada Tabel 6.16.

Tabel 6.16 Pengujian Failback Topologi High Availability Web Server Aplikasi Keepalived Skenario 3

Pengujian Ke	Hasil Pengujian (detik)	Rata-rata (detik)
1	1	1
2	1	
3	1	
4	1	
5	1	
6	1	
7	1	
8	1	
9	1	
10	1	
11	1	
12	1	

Setelah selesai melaksanakan pengujian ini, diketahui bahwa pada pengujian skenario 3 ini terjadi proses *failback*. Hal ini dibuktikan dari hasil pengujian dan dilakukan percobaan mengakses alamat IP virtual Keepalived setelah *service* perangkat lunak Keepalived pada Web Server 1 kembali dihidupkan.

Hasil semua pengujian *failback* memiliki hasil 1 detik. *Packet loss* terjadi karena adanya proses *failback* yang memindahkan *traffic* dari Web Server 2 menuju Web Server 1 sehingga untuk sementara *resource* tidak dapat diakses. Hasil rata-rata waktu *failback* pada pengujian ini sebesar 1 detik. Hasil dari pengujian mengakses alamat IP virtual Heartbeat setelah proses *failback* ditampilkan pada Gambar 6.20.



Gambar 6.20 Mengakses Alamat IP Virtual Setelah Proses Failback

Dari Gambar 6.20, dapat disimpulkan bahwa proses *failback* terjadi, karena pada percobaan sebelumnya Ketika mengakses alamat IP virtual Keepalived saat *service* Keepalived pada Web Server 1 sedang dalam kondisi mati,konten yang tampil merupakan konten dari Web Server 2, seperti yang ditampilkan pada Gambar 6.12Gambar 6.11.

6.2 Pengujian Topologi Sistem Failover Untuk Kebutuhan High Availability Load Balancing

6.2.1 Pengujian Fungsional

Pengujian fungsional dilakukan untuk mengetahui apakah sistem yang diimplementasi telah berjalan dengan baik dan benar sesuai fungsinya. Pada Tabel 6.17 dan Tabel 6.18 akan ditampilkan hasil dari pengujian fungsional.

Tabel 6.17 Pengujian Fungsional Topologi High Availability Load Balancing Perangkat Lunak Heartbeat

No	Pengujian	Keterangan
1.	Konten Wordpress pada Web Server 1 dapat diakses oleh pengguna	Valid
2.	Konten Wordpress pada Web Server 2 dapat diakses oleh pengguna	Valid
3.	Haproxy pada server Loadbalancer1 dapat membagi *traffic* yang masuk menuju Web Server 1 dan Web Server 2	Valid
4.	Haproxy pada server Loadbalancer2 dapat membagi *traffic* yang masuk menuju Web Server 1 dan Web Server 2	Valid
5.	Ketika server Loadbalancer1 dalam kondisi mati, pengguna masih dapat mengakses alamat virtual IP karena ada *server* Loadbalancer2 ada implementasi *failover* yang mengalihkan *traffic* menuju *server backup*, yaitu Loadbalancer2.	Valid
6.	Ketika *server* Loadbalancer2 dalam kondisi mati, pengguna tidak dapat mengakses website karena tidak ada server aktif untuk menangani *traffic* yang masuk.	Valid

Tabel 6.18 Pengujian Fungsional Topologi High Availability Load Balancing Perangkat Lunak Keepalived

No	Pengujian	Keterangan
1.	Konten Wordpress pada Web Server 1 dapat diakses oleh pengguna	Valid
2.	Konten Wordpress pada Web Server 2 dapat diakses oleh pengguna	Valid
3.	Haproxy pada server Loadbalancer1 dapat membagi *traffic* yang masuk menuju Web Server 1 dan Web Server 2	Valid
4.	Haproxy pada server Loadbalancer2 dapat membagi *traffic* yang masuk menuju Web Server 1 dan Web Server 2	Valid
5.	Ketika server Loadbalancer1 dalam kondisi mati, pengguna masih dapat mengakses alamat virtual IP karena ada *server* Loadbalancer2 ada implementasi *failover* yang mengalihkan *traffic* menuju *server backup*, yaitu Loadbalancer2.	Valid
6.	Ketika *server* Loadbalancer2 dalam kondisi mati, pengguna tidak dapat mengakses website karena tidak ada server aktif untuk menangani *traffic* yang masuk.	Valid

6.2.2 Pengujian Downtime

Downtime adalah waktu dimana sebuah *resource* untuk sementara tidak dapat diakses ketika sedang terjadi proses *failover*. Pengujian *downtime* dilaksanakan untuk mengetahui berapa lama waktu proses perpindahan *failover* dari server utama yang mengalami kegagalan menuju ke server cadangan. *Resource* akan tetap bisa diakses setelah proses *failover* selesai, karena ada komponen lain yang menjadi cadangan ketika komponen utama mengalami kegagalan.

Pengujian *downtime* ini memiliki tujuan untuk mengetahui berapa lama proses yang dibutuhkan perangkat lunak *failover* untuk memindahkan *traffic* dari *server master* menuju *server backup*. *Server master* dalam pengujian ini adalah Web Server 1, sedangkan yang menjadi *server backup* adalah Web Server 2.

Pengujian *downtime* akan dilaksanakan dalam 3 skenario untuk mengetahui apakah ada perbedaan waktu *failover* antar skenario. Pengujian *downtime* dilakukan dengan melakukan ping ke alamat IP virtual Heartbeat 192.168.200.25 dan alamat IP virtual Keepalived 192.168.200.26 . Untuk dapat mengetahui waktu *downtime* dapat dilakukan dengan perintah seperti pada Tabel 6.19Tabel 6.3 untuk mengirimkan paket ping dengan interval tiap 0.2 detik.

Tabel 6.19 Perintah Ping

1	$ ping 192.168.200.59 -i 0.2

Setelah melakukan ping seperti pada Tabel 6.3**Error! Reference source not found.** maka akan menampilkan hasil seperti pada Gambar 6.21.

```
64 bytes from 192.168.200.59: icmp_seq=259 ttl=64 time=2.41 ms
64 bytes from 192.168.200.59: icmp_seq=260 ttl=64 time=1.86 ms
64 bytes from 192.168.200.59: icmp_seq=261 ttl=64 time=1.68 ms
64 bytes from 192.168.200.59: icmp_seq=262 ttl=64 time=1.95 ms
64 bytes from 192.168.200.59: icmp_seq=263 ttl=64 time=1.18 ms
64 bytes from 192.168.200.59: icmp_seq=264 ttl=64 time=2.00 ms
64 bytes from 192.168.200.59: icmp_seq=265 ttl=64 time=1.45 ms
64 bytes from 192.168.200.59: icmp_seq=266 ttl=64 time=1.61 ms
64 bytes from 192.168.200.59: icmp_seq=267 ttl=64 time=1.36 ms
64 bytes from 192.168.200.59: icmp_seq=268 ttl=64 time=1.58 ms
64 bytes from 192.168.200.59: icmp_seq=269 ttl=64 time=1.61 ms
64 bytes from 192.168.200.59: icmp_seq=270 ttl=64 time=1.93 ms
64 bytes from 192.168.200.59: icmp_seq=271 ttl=64 time=2.78 ms
^C
--- 192.168.200.59 ping statistics ---
271 packets transmitted, 253 received, 6% packet loss, time 55277ms
rtt min/avg/max/mdev = 0.570/3.393/37.154/5.654 ms
dian@server222:~$
```

Gambar 6.21 Hasil Perintah Ping

Pada Gambar 6.21**Error! Reference source not found.** dapat dilihat bahwa dari 271 paket yang dikirimkan, hanya ada 253 paket yang diterima, jadi ada 18 paket yang hilang. Untuk menghitung waktu *downtime*, jumlah paket yang hilang dikali 0.2 detik, sehingga waktu *downtime*nya adalah 3.6 detik.

6.2.2.1 Pengujian Downtime Topologi High Availability Load Balancing Perangkat Lunak Heartbeat Skenario 1

Pengujian *downtime* dilakukan dengan melakukan ping ke alamat IP virtual Heartbeat 192.168.200.25. Pada pengujian *downtime* skenario 1, *server* virtual Loadbalancer 1 akan dimatikan secara sengaja, kemudian akan dihitung waktu yang dibutuhkan untuk proses *failover* dari *server* Loadbalancer1 menuju Loadbalancer2. Hasil dari pengujian *downtime* skenario 1 dijelaskan pada Tabel 6.20.

Tabel 6.20 Pengujian Downtime Topologi Load Balancing Perangkat Lunak Heartbeat Skenario 1

Pengujian Ke	Hasil Pengujian (detik)	Rata-rata (detik)
1	3,8	4,78
2	4	
3	4,6	
4	4,6	
5	5,2	

6	4,8	
7	5,4	
8	4,6	
9	4	
10	5,8	
11	5,4	
12	5,2	

Setelah selesai melaksanakan pengujian ini, diketahui bahwa pada pengujian topologi skenario 1 ini terjadi proses *failover*. Hal ini dibuktikan dari hasil pengujian *downtime* dan percobaan mengakses alamat IP *virtual* Heartbeat setelah *server virtual* Loadbalancer1 dimatikan.

Hasil pengujian downtimememiliki hasil *packet loss* berkisar antara 3,8 detik hingga 5,8 detik. Waktu d*owntime* terjadi karena adanya proses *failover* yang memindahkan *traffic* dari Loadbalancer1 menuju Loadbalancer2 sehingga untuk sementara *resource* tidak dapat diakses. Hasil rata-rata waktu *downtime* pada pengujian ini adalah 4,78 detik. Gambar dari dan percobaan mengakses alamat IP *virtual* Heartbeat setelah *server virtual* Loadbalancer1 dimatikan ditampilkan pada Gambar 6.22.

Gambar 6.22 Mengakses IP Virtual Heartbeat Ketika Loadbalancer1 Mati

6.2.2.2 Pengujian Downtime Topologi High Availability Load Balancing Perangkat Lunak Keepalived Skenario 1

Pengujian *downtime* dilakukan dengan melakukan ping ke alamat IP virtual Keepalived 192.168.200.26. Pada pengujian *downtime* skenario 1, server virtual Loadbalancer 1 akan dimatikan secara sengaja, kemudian akan dihitung waktu yang dibutuhkan untuk proses failover dari server Loadbalancer1 ke Loadbalancer2. Hasil dari pengujian *downtime* skenario 1 dijelaskan pada Tabel 6.21.

Tabel 6.21 Pengujian Downtime Topologi Load Balancing Perangkat Lunak Keepalived Skenario 1

Pengujian Ke	Hasil Pengujian (detik)	Rata-rata (detik)
1	3,8	4,05
2	3,8	
3	4	
4	4,4	
5	4,2	
6	4,4	
7	3,8	
8	3,8	
9	4	
10	4,4	
11	4,2	
12	3,8	

Setelah selesai melaksanakan pengujian ini, diketahui bahwa pada pengujian topologi skenario 1 ini terjadi proses *failover*. Hal ini dibuktikan dari hasil pengujian *downtime* dan dan percobaan mengakses alamat IP *virtual* Keepalived setelah *server virtual* Loadbalancer1 dimatikan.

Pengujian downtime memiliki hasil berkisar antara 3,8 detik hingga 4,4 detik. Waktu d*owntime* terjadi karena adanya proses *failover* yang memindahkan *traffic* dari Loadbalancer1 menuju Loadbalancer2 sehingga untuk sementara *resource* tidak dapat diakses. Hasil rata-rata waktu *downtime* pada pengujian ini adalah 4,05 detik. Gambar dari dan percobaan mengakses alamat IP *virtual* Keepalived setelah *server virtual* Loadbalancer1 dimatikan ditampilkan pada Gambar 6.23.

Gambar 6.23 Mengakses IP Virtual Keepalived Setelah Server Loadbalancer1 Mati

6.2.2.3 Pengujian Downtime Topologi High Availability Load Balancing Perangkat Lunak Heartbeat Skenario 2

Pengujian *downtime* dilakukan dengan melakukan ping ke alamat IP virtual Heartbeat 192.168.200.25. Pada pengujian *downtime* skenario 2, *service* Haproxy pada server Loadbalancer1 akan dimatikan. Kemudian akan dihitung waktu *downtime* untuk proses *failover* dari *server* Loadbalancer1 menuju *server* Loadbalancer2. Hasil dari pengujian *downtime* skenario 2 dijelaskan pada Tabel 6.22.**Error! Reference source not found.**

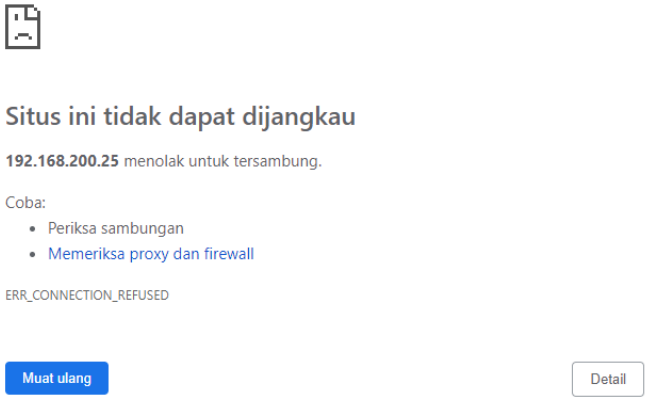
Tabel 6.22 Pengujian Downtime Topologi Load Balancing Perangkat Lunak Heartbeat Skenario 2

Pengujian Ke	Hasil Pengujian (detik)	Rata-rata (detik)
1	0	0
2	0	
3	0	
4	0	
5	0	
6	0	
7	0	
8	0	
9	0	
10	0	
11	0	
12	0	

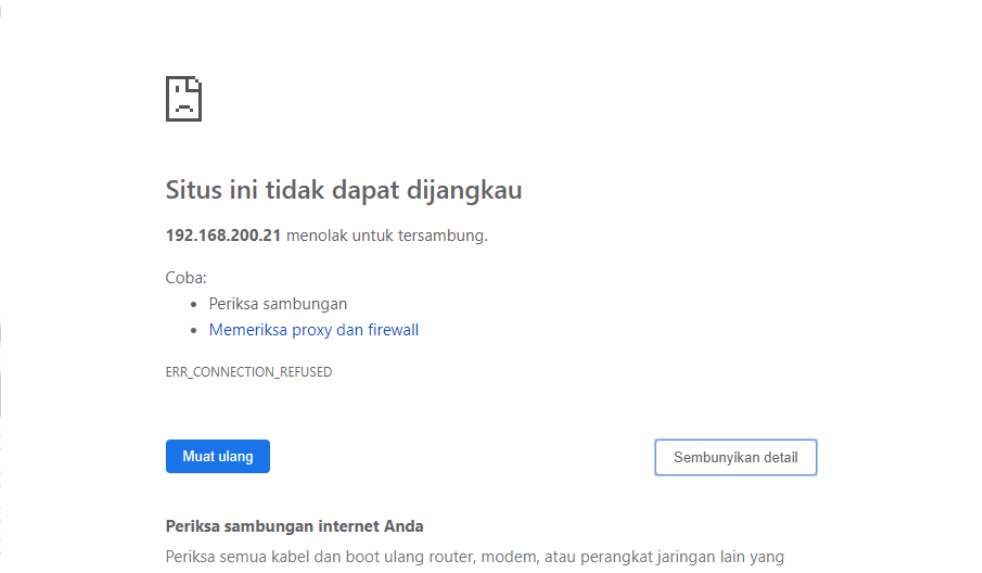
Setelah selesai melaksanakan pengujian ini, diketahui bahwa pada pengujian skenario 2 ini tidak terjadi proses *failover*. Hal ini dibuktikan dari hasil pengujian *downtime* dan percobaan mengakses alamat IP virtual setelah service Haproxy dimatikan. Pengujian ini memiliki hasil waktu *packet loss* sebesar 0 detik pada pengujian 1 hingga 12.

Proses *failover* tidak terjadi setelah *service* Haproxy dimatikan adalah karena Heartbeat tidak mengetahui bahwa *service* Haproxy pada loadbalancer1 telah mati. Heartbeat hanya mengetahui bahwa *server* Loadbalancer1 masih aktif, sehingga Heartbeat tidak memindahkan *traffic* dari server *master* loadbalancer1 menuju server *backup* loadbalancer2.

Percobaan mengakses alamat IP virtual Keepalived setelah service Haproxy pada loadbalancer1 dimatikan, ditampilkan pada Gambar 6.24



Gambar 6.24 Mengakses IP Virtual Heartbeat Setelah Service Haproxy Pada Loadbalancer1 Mati



Gambar 6.25 Mengakses IP Loadbalancer1 Setelah Service Haproxy Pada Loadbalancer1 Mati

Gambar 6.26 Mengakses IP Loadbalancer2 Setelah Service Haproxy Pada Loadbalancer1 Mati

Dari Gambar 6.25 dan Gambar 6.26, semakin menjelaskan bahwa proses failover tidak terjadi. Pada gambar Gambar 6.24 dan Gambar 6.25 menampilkan hasil yang sama, karena jika terjadi proses failover saat mengakses IP virtual Heartbeat maka akan menampilkan output konten Wordpress seperti pada Gambar 6.26.

6.2.2.4 Pengujian Downtime Topologi High Availability Load Balancing Perangkat Lunak Keepalived Skenario 2

Pengujian *downtime* dilakukan dengan melakukan ping ke alamat IP virtual Keepalived 192.168.200.26. Pada pengujian *downtime* skenario 2, service Haproxy pada server Loadbalancer1 akan dimatikan. Kemudian akan dihitung waktu *downtime* untuk proses *failover* dari *server* Loadbalancer1 menuju *server* Loadbalancer2. Hasil dari pengujian *downtime* skenario 2 dijelaskan pada Tabel 6.23.**Error! Reference source not found.**

Tabel 6.23 Pengujian Downtime Topologi Load Balancing Perangkat Lunak Keepalived Skenario 2

Pengujian Ke	Hasil Pengujian (detik)	Rata-rata (detik)
1	0	0
2	0	
3	0	
4	0	
5	0	
6	0	
7	0	

8	0	
9	0	
10	0	
11	0	
12	0	

Setelah selesai melaksanakan pengujian ini, diketahui bahwa pada pengujian skenario 2 ini tidak terjadi proses *failover*. Hal ini dibuktikan dari hasil pengujian *downtime* dan percobaan mengakses alamat IP virtual setelah *service* Haproxy dimatikan. Pengujian ini memiliki hasil *packet loss* sebesar 0 detik pada pengujian 1 hingga 12.

Proses *failover* tidak terjadi setelah *service* Haproxy dimatikan adalah karena Keepalived tidak mengetahui bahwa *service* Haproxy pada loadbalancer1 telah mati. Keepalived hanya mengetahui bahwa *server* Loadbalancer1 masih aktif, sehingga Keepalived tidak memindahkan *traffic* dari server *master* loadbalancer1 menuju server *backup* loadbalancer2.

6.2.2.5 Pengujian Downtime Topologi High Availability Load Balancing Perangkat Lunak Heartbeat Skenario 3

Pengujian *downtime* dilakukan dengan melakukan ping ke alamat IP virtual Heartbeat 192.168.200.25. Pada pengujian *downtime* skenario 3, *service* perangkat lunak *failover* Heartbeat pada server Loadbalancer1 akan dimatikan. Kemudian akan dihitung waktu *downtime* untuk proses *failover* dari server Loadbalancer1 menuju server Loadbalancer2. Hasil dari pengujian *downtime* skenario 3 dijelaskan pada Tabel 6.24.

Tabel 6.24 Pengujian Downtime Topologi Load Balancing Perangkat Lunak Heartbeat Skenario 3

Pengujian Ke	Hasil Pengujian (detik)	Rata-rata (detik)
1	0,6	
2	1,6	
3	0,6	
4	0,4	
5	1,6	0,78
6	0,8	
7	1,6	
8	0,6	
9	0,4	

10	0,2	
11	0,6	
12	0,4	

Setelah selesai melaksanakan pengujian ini, diketahui bahwa pada pengujian topologi *high availability load balancing* skenario 3 ini terjadi proses *failover*. Hal ini dibuktikan dari hasil pengujian *downtime* dan percobaan mengakses alamat IP virtual Heartbeat setelah *service* Heartbeat pada Loadbalancer1 dimatikan.

Hasil pengujian ini memiliki hasil rata-rata waktu *downtime* sebesar 0,78 detik. Nilai *downtime* berkisar antara 0,2 detik hingga 1,6 detik. Adanya *downtime* terjadi karena proses *failover* yang memindahkan *traffic* dari Loadbalancer1 menuju Loadbalancer2 akibat *service* Heartbeat pada Loadbalancer1 mati sehingga menyebabkan *resource* tidak dapat diakses dalam waktu beberapa detik. Hasil dari percobaan mengakses alamat IP virtual Keepalived setelah *service* Keepalived pada Loadbalancer1 dimatikan ditampilkan pada Gambar 6.27.

Gambar 6.27 Mengakses Virtual IP Heartbeat Ketika Service Heartbeat Pada Loadbalancer1 Mati

6.2.2.6 Pengujian Downtime Topologi High Availability Load Balancing Aplikasi Keepalived Skenario 3

Pengujian *downtime* dilakukan dengan melakukan ping ke alamat IP virtual Keepalived 192.168.200.26. Pada pengujian *downtime* skenario 3, *service* perangkat lunak *failover* Keepalived pada server Loadbalancer1 dimatikan. Kemudian akan dihitung waktu *downtime* untuk proses *failover* dari *server* Loadbalancer1 menuju server Loadbalancer2. Hasil dari pengujian *downtime* skenario 3 dijelaskan pada Tabel 6.25.

Tabel 6.25 Pengujian Downtime Topologi Load Balancing Perangkat Lunak Keepalived Skenario 3

Pengujian Ke	Hasil Pengujian (detik)	Rata-rata (detik)
1	1,6	1,53
2	1,6	
3	1,6	
4	1,6	
5	1,4	
6	1,4	
7	1,6	
8	1,6	
9	1,6	
10	1,4	
11	1,4	
12	1,6	

Setelah selesai melaksanakan pengujian ini, diketahui bahwa pada pengujian topologi *high availability load balancing* skenario 3 ini terjadi proses *failover*. Hal ini dibuktikan dari hasil pengujian *downtime* dan percobaan mengakses alamat IP virtual Keepalived setelah *service* Keepalived pada Loadbalancer1 dimatikan.

Hasil pengujian ini memiliki hasil rata-rata waktu *downtime* sebesar 1,53 detik. Nilai *packet loss* berkisar antara 1,4 detik hingga 1,6 detik. Adanya *downtime* terjadi karena proses *failover* yang memindahkan *traffic* dari Loadbalancer1 menuju Loadbalancer2 akibat *service* Keepalived pada Loadbalancer1 mati sehingga menyebabkan *resource* tidak dapat diakses dalam waktu beberapa detik.

6.2.3 Pengujian Failback

Failback adalah sebuah proses kembalinya server *master* kedalam sebuah klaster, yang sebelumnya server utama tersebut mengalami kegagalan sehingga yang menangani permintaan untuk sementara adalah *server backup*. Dalam perancangan topologi ini, yg berperan sebagai server *master* adalah Loadbalancer1 dan yang berperan sebagai server *backup* adalah Loadbalancer2.

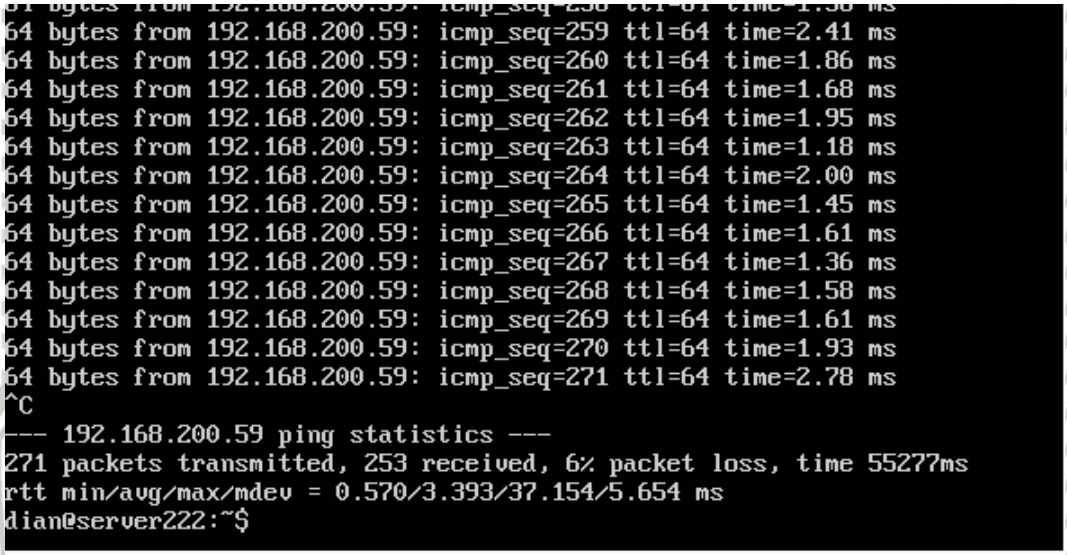
Pengujian *failback* dilakukan untuk mengetahui berapa lama waktu proses yang diperlukan untuk proses perpindahan *traffic* dari Loadbalancer2 kembali ke Loadbalancer1 yang baru saja aktif kembali. Pengujian *failback* akan dilakukan pada skenario 1, 2 dan 3 untuk mengetahui apakah ada perbedaan waktu *failback* antar skenario. Pengujian *failback* dilakukan dengan melakukan ping ke alamat IP virtual Heartbeat 192.168.200.25 dan alamat IP virtual Keepalived 192.168.200.26. Untuk dapat mengetahui waktu *failback* dapat dilakukan dengan

perintah seperti pada Tabel 6.26Tabel 6.3 untuk mengirimkan paket ping dengan interval tiap 0.2 detik.

Tabel 6.26 Perintah Ping

1	$ ping 192.168.200.59 -i 0.2

Setelah melakukan ping seperti pada Tabel 6.10**Error! Reference source not found.** maka akan menampilkan hasil seperti pada Gambar 6.28.



Gambar 6.28 Hasil Perintah Ping

Pada Gambar 6.28**Error! Reference source not found. Error! Reference source not found.** dapat dilihat bahwa dari 271 paket yang dikirimkan, hanya ada 253 paket yang diterima, jadi ada 18 paket yang hilang. Untuk menghitung waktu *failback*, jumlah paket yang hilang dikali 0.2 detik, sehingga waktu failbacknya adalah 3.6 detik.

6.2.3.1 Pengujian Failback Topologi High Availability Load Balancing Perangkat Lunak Heartbeat Skenario 1

Pengujian *failback* dilakukan dengan melakukan ping ke alamat IP virtual Heartbeat 192.168.200.25. Pada pengujian *failback* skenario 1, *server virtual* Loadbalancer1 akan dihidupkan kembali pada saat *server virtual* Loadbalancer2 sedang menangani *traffic* yang masuk akibat server Loadbalancer1 dalam kondisi mati, kemudian akan dihitung waktu proses *failback* dari Loadbalancer2 ke Loadbalancer1. Hasil dari pengujian *failback* skenario 1 dijelaskan pada Tabel 6.27.

Tabel 6.27 Pengujian Failback Topologi Load Balancing Aplikasi Heartbeat Skenario 1

Pengujian Ke	Hasil Pengujian (detik)	Rata-rata (detik)
1	0,4	0,48

2	0,6	
3	0,2	
4	0,8	
5	0,4	
6	0,4	
7	0,6	
8	0,4	
9	0,8	
10	0,6	
11	0,2	
12	0,4	

Setelah selesai melaksanakan pengujian ini, diketahui bahwa pada pengujian skenario1 terjadi proses *failback*. Hal ini dapat dibuktikan dari hasil pengujian *failback* dan percobaan mengakses alamat IP *virtual* Heartbeat setelah *server* Loadbalancer1 sudah kembali aktif.

Proses *failback* terjadi karena perpindahan *traffic* dari Loadbalancer2 ke Loadbalancer1, karena pada saat server Loadbalancer1 tidak aktif yang menangani *traffic* adalah Loadbalancer2. Saat proses *failback* sedang berlangsung, *resource* untuk sementara tidak dapat diakses. Hal ini dibuktikan dari hasil pengujian *failback* yang memiliki *packet loss* berkisar antara 0,2 detik hingga 0,8 detik. Hasil rata-rata waktu *failback* dari pengujian ini adalah 0,48 detik.

6.2.3.2 Pengujian Failback Topologi High Availability Load Balancing Perangkat Lunak Keepalived Skenario 1

Pengujian *failback* dilakukan dengan melakukan ping ke alamat IP virtual Keepalived 192.168.200.26. Pada pengujian *failback* skenario 1, *server virtual* Loadbalancer1 akan dihidupkan kembali pada saat *server virtual* Loadbalancer2 sedang menangani *traffic* yang masuk akibat server Loadbalancer1 dalam kondisi mati, kemudian akan dihitung waktu proses *failback* dari Loadbalancer2 ke Loadbalancer1. Hasil dari pengujian *failback* skenario 1 dijelaskan pada Tabel 6.28.

Tabel 6.28 Pengujian Failback Topologi Load Balancing Perangkat Lunak Keepalived Skenario 1

Pengujian Ke	Hasil Pengujian (detik)	Rata-rata (detik)
1	1	1,01
2	1	

3	1	
4	1	
5	1	
6	1	
7	1	
8	1	
9	1	
10	1,2	
11	1	
12	1	

Setelah selesai melaksanakan pengujian ini, diketahui bahwa pada pengujian skenario1 terjadi proses *failback*. Hal ini dapat dibuktikan dari hasil pengujian *failback* dan percobaan mengakses alamat IP virtual Keepalived setelah server Loadbalancer1 sudah kembali aktif.

Proses *failback* terjadi karena perpindahan *traffic* dari Loadbalancer2 ke Loadbalancer1, karena pada saat server Loadbalancer1 tidak aktif yang menangani *traffic* adalah Loadbalancer2. Saat proses *failback* sedang berlangsung, *resource* untuk sementara tidak dapat diakses. Hal ini dibuktikan dari hasil pengujian *failback* yang memiliki *failback* berkisar antara 1 detik hingga 1,2 detik. Hasil rata-rata waktu *failback* dari pengujian ini adalah 1,01 detik.

6.2.3.3 Pengujian Failback Topologi High Availability Load Balancing Perangkat Lunak Heartbeat Skenario 2

Pengujian *failback* dilakukan dengan melakukan ping ke alamat IP virtual Heartbeat 192.168.200.25. Pada pengujian *failback* skenario 2, *service* Haproxy pada Loadbalancer1 akan dihidupkan kembali. Kemudian akan dihitung waktu jika ada proses *failback* dari Loadbalancer2 ke Loadbalancer1. Hasil dari pengujian *failback* skenario 2 dijelaskan pada Tabel 6.29.

Tabel 6.29 Pengujian Failback Topologi Load Balancing Perangkat Lunak Heartbeat Skenario 2

Pengujian Ke	Hasil Pengujian (detik)	Rata-rata (detik)
1	0	
2	0	
3	0	0
4	0	
5	0	

6	0	
7	0	
8	0	
9	0	
10	0	
11	0	
12	0	

Setelah selesai melaksanakan pengujian skenario 2 ini, diketahui bahwa pada pengujian ini tidak terjadi proses *failback*. *Failback* tidak terjadi karena dari pengujian *downtime* skenario 2 yang telah dilakukan sebelumnya, tidak terjadi proses *failover*. Heartbeat tidak mendeteksi bahwa *service* Haproxy pada Loadbalancer1 telah mati, sehingga *traffic* tidak berpindah menuju *server backup*. Hal tersebut dapat dibuktikan bahwa dalam pengujian ini memiliki hasil sebesar 0 detik pada 12 kali percobaan pengujian.

6.2.3.4 Pengujian Failback Topologi High Availability Load Balancing Perangkat Lunak Keepalived Skenario 2

Pengujian *failback* dilakukan dengan melakukan ping ke alamat IP virtual Keepalived 192.168.200.26. Pada pengujian *failback* skenario 2, *service* Haproxy pada Loadbalancer1 akan dihidupkan kembali. Kemudian akan dihitung waktu jika ada proses *failback* dari Loadbalancer2 ke Loadbalancer1. Hasil dari pengujian *failback* skenario 2 dijelaskan pada Tabel 6.30.

Tabel 6.30 Pengujian Failback Topologi Load Balancing Perangkat Lunak Keepalived Skenario 2

Pengujian Ke	Hasil Pengujian (detik)	Rata-rata (detik)
1	0	
2	0	
3	0	
4	0	
5	0	
6	0	0
7	0	
8	0	
9	0	
10	0	

11	0	
12	0	

Setelah selesai melaksanakan pengujian skenario 2 ini, diketahui bahwa pada pengujian ini tidak terjadi proses *failback*. *Failback* tidak terjadi karena dari pengujian *failover* skenario 2 yang telah dilakukan sebelumnya, tidak terjadi proses *failover*. Keepalived tidak mendeteksi bahwa *service* Haproxy pada Loadbalancer1 telah mati, sehingga *traffic* tidak berpindah menuju *server backup*. Hal tersebut dapat dibuktikan bahwa dalam pengujian ini memiliki hasil sebesar 0 detik pada 12 kali percobaan pengujian.

6.2.3.5 Pengujian Failback Topologi High Availability Load Balancing Perangkat Lunak Heartbeat Skenario 3

Pengujian *failback* dilakukan dengan melakukan ping ke alamat IP virtual Heartbeat 192.168.200.25. Pada pengujian *failback* skenario 3, service Heartbeat pada Loadbalancer1 akan dihidupkan kembali pada saat *server virtual* Loadbalancer2 sedang menangani *traffic* yang masuk akibat *service* Heartbeat pada Loadbalancer1 dalam kondisi mati. Kemudian akan dihitung waktu proses *failback* dari Loadbalancer2 ke Loadbalancer1. Hasil dari pengujian *failback* skenario 3 dijelaskan pada Tabel 6.31.

Tabel 6.31 Pengujian Failback Topologi Load Balancing Perangkat Lunak Heartbeat Skenario 3

Pengujian Ke	Hasil Pengujian (detik)	Rata-rata (detik)
1	0,4	0,53
2	0,6	
3	0,6	
4	0,6	
5	0,2	
6	0,8	
7	0,8	
8	0,6	
9	0,4	
10	0,4	
11	0,6	
12	0,4	

Setelah selesai melaksanakan pengujian ini, diketahui bahwa pada pengujian skenario 3 terjadi proses *failback*. Hal ini dapat dibuktikan dari hasil pengujian *failback*.

Proses *failback* terjadi karena perpindahan *traffic* dari Loadbalancer2 ke Loadbalancer1, karena pada saat *service* Heartbeat pada Loadbalancer1 tidak aktif ,yang menangani *traffic* adalah Loadbalancer2. Saat proses *failback* sedang berlangsung, *resource* untuk sementara tidak dapat diakses. Hal ini dibuktikan dari hasil pengujian *failback* yang memiliki *packet loss* berkisar antara 0,2 detik hingga 0,8 detik. Hasil rata-rata waktu *failback* dari pengujian ini adalah 0,53 detik.

6.2.3.6 Pengujian Failback Topologi High Availability Load Balancing Perangkat Lunak Keepalived Skenario 3

Pengujian *failback* dilakukan dengan melakukan ping ke alamat IP virtual Keepalived 192.168.200.25. Pada pengujian *failback* skenario 3, service Keepalived pada Loadbalancer1 akan dihidupkan kembali pada saat *server virtual* Loadbalancer2 sedang menangani *traffic* yang masuk akibat *service* Keepalived pada Loadbalancer1 dalam kondisi mati. Kemudian akan dihitung waktu proses *failback* dari Loadbalancer2 ke Loadbalancer1. Hasil dari pengujian *failback* skenario 3 dijelaskan pada Tabel 6.32.

Tabel 6.32 Pengujian Failback Topologi Load Balancing Perangkat Lunak Keepalived Skenario 3

Pengujian Ke	Hasil Pengujian (detik)	Rata-rata (detik)
1	1	0,96
2	1	
3	1	
4	1	
5	0,8	
6	1	
7	1	
8	1	
9	1	
10	1	
11	1	
12	0,8	

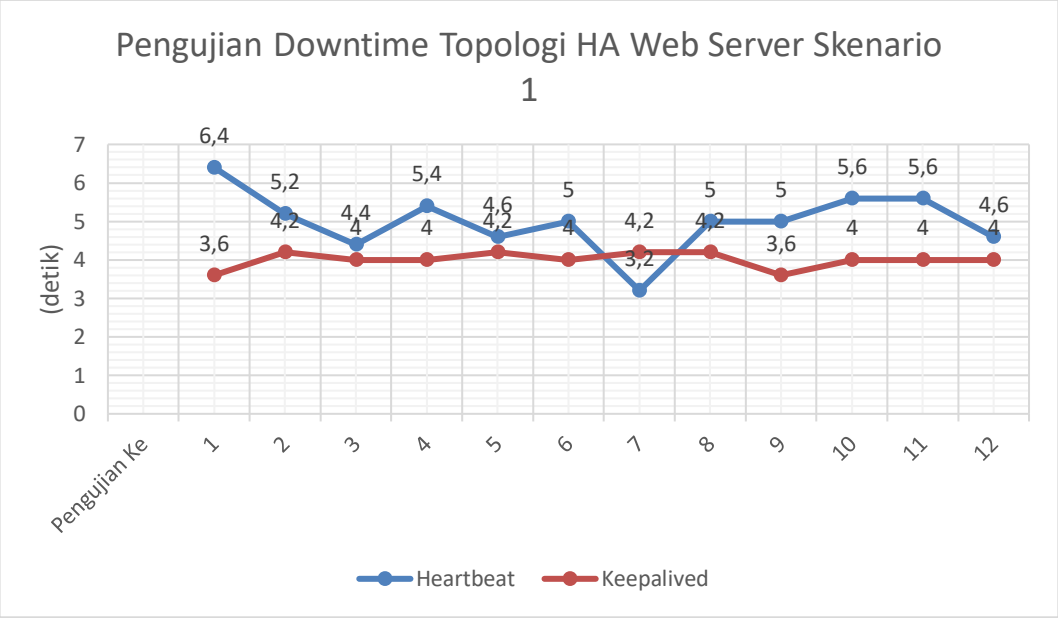
Setelah selesai melaksanakan pengujian ini, diketahui bahwa pada pengujian skenario 3 terjadi proses *failback*. Hal ini dapat dibuktikan dari hasil pengujian *failback*.

Proses *failback* terjadi karena perpindahan *traffic* dari Loadbalancer2 ke Loadbalancer1, karena pada saat *service* Keepalived pada Loadbalancer1 tidak aktif ,yang menangani *traffic* adalah Loadbalancer2. Saat proses *failback* sedang berlangsung, *resource* untuk sementara tidak dapat diakses. Hal ini dibuktikan dari hasil pengujian *failback* yang memiliki waktu berkisar antara 0,8 detik hingga 1 detik. Hasil rata-rata waktu *failback* dari pengujian ini adalah 0,96 detik.

6.3 Analisis Pengujian Downtime

6.3.1 Analisis Pengujian Downtime Topologi High Availability Webserver Skenario 1

Dari hasil pengujian *downtime* topologi High Availability Web Server Skenario 1, diketahui perbandingan waktu *downtime* antara perangkat lunak *failover* Heartbeat dan perangkat lunak *failover* Keepalived.



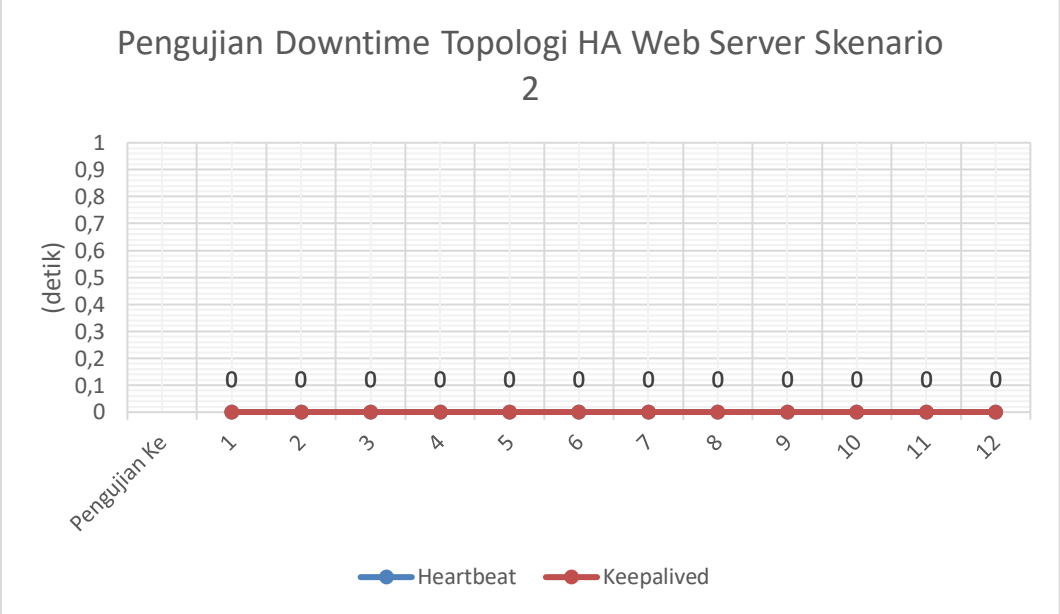
Gambar 6.29 Grafik Hasil Pengujian Downtime Topologi HA Web Server Skenario 1

Dari 12 kali pengujian *downtime* yang dilakukan pada skenario 1 ini, diketahui bahwa perangkat lunak Heartbeat memiliki hasil rata-rata waktu *downtime* sebesar 5 detik, sedangkan perangkat lunak Keepalived memiliki hasil rata-rata waktu *downtime* sebesar 4 detik. Dengan demikian pada skenario ini, Keepalived memiliki rata-rata waktu *downtime* yang lebih rendah.

6.3.2 Analisis Pengujian Downtime Topologi High Availability Webserver Skenario 2

Dari hasil pengujian *downtime* topologi High Availability Web Server Skenario 2, diketahui bahwa proses *failover* tidak terjadi. Hal ini disebabkan karena

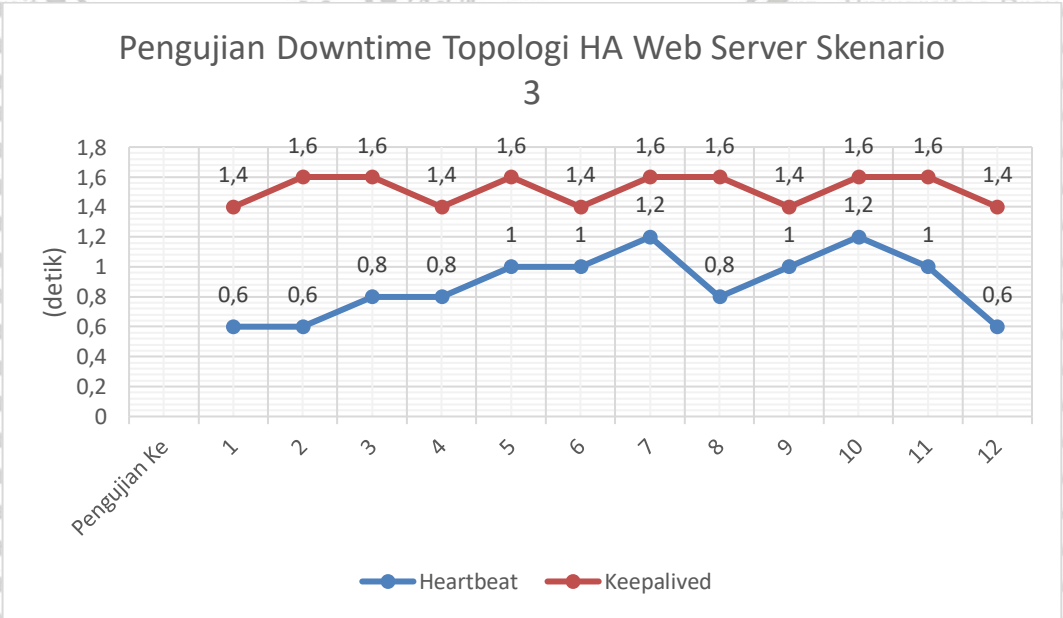
perangkat lunak *failover*, baik Heartbeat maupun Keepalived tidak dapat mendeteksi bahwa service perangkat lunak Apache2 berubah dari aktif menjadi tidak aktif.



Gambar 6.30 Grafik Hasil Pengujian Downtime Topologi HA Web Server Skenario 2

6.3.3 Analisis Pengujian Downtime Topologi High Availability Webserver Skenario 3

Dari hasil pengujian *downtime* topologi High Availability Web Server Skenario 3, diketahui perbandingan waktu *downtime* antara perangkat lunak *failover* Heartbeat dan perangkat lunak *failover* Keepalived.

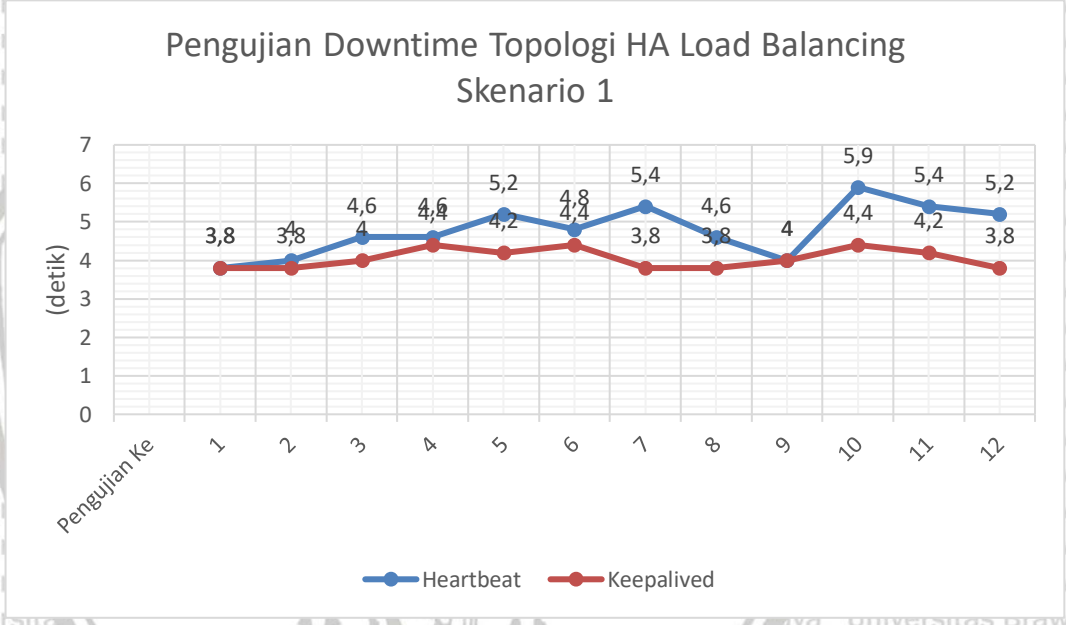


Gambar 6.31 Grafik Hasil Pengujian Downtime Topologi HA Web Server Skenario 3

Dari 12 kali pengujian *downtime* yang dilakukan pada skenario 3 ini, diketahui bahwa perangkat lunak Heartbeat memiliki hasil rata-rata waktu *downtime* sebesar 0,88 detik, sedangkan perangkat lunak Keepalived memiliki hasil rata-rata waktu *downtime* sebesar 1 detik. Dengan demikian pada skenario ini Heartbeat memiliki rata-rata waktu *downtime* yang lebih rendah.

6.3.4 Analisis Pengujian Downtime Topologi High Availability Load Balancing Skenario 1

Dari hasil pengujian *downtime* topologi High Availability Load balancing Skenario 1, diketahui perbandingan waktu *downtime* antara perangkat lunak *failover* Heartbeat dan perangkat lunak *failover* Keepalived.

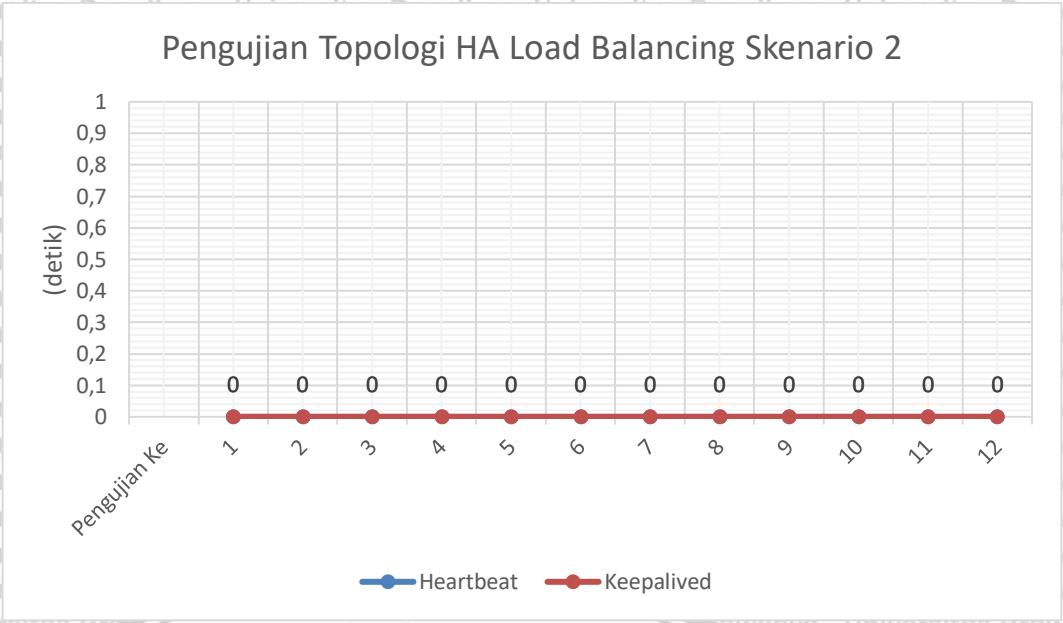


Gambar 6.32 Grafik Pengujian Downtime HA Load Balancing Skenario 1

Dari 12 kali pengujian *downtime* yang dilakukan pada skenario 1 ini, diketahui bahwa perangkat lunak Heartbeat memiliki hasil rata-rata waktu *downtime* sebesar 4,78 detik, sedangkan perangkat lunak Keepalived memiliki hasil rata-rata waktu *downtime* sebesar 4,05 detik. Dengan demikian pada skenario ini, Keepalived memiliki rata-rata waktu *downtime* lebih rendah.

6.3.5 Analisis Pengujian Downtime Topologi High Availability Load Balancing Skenario 2

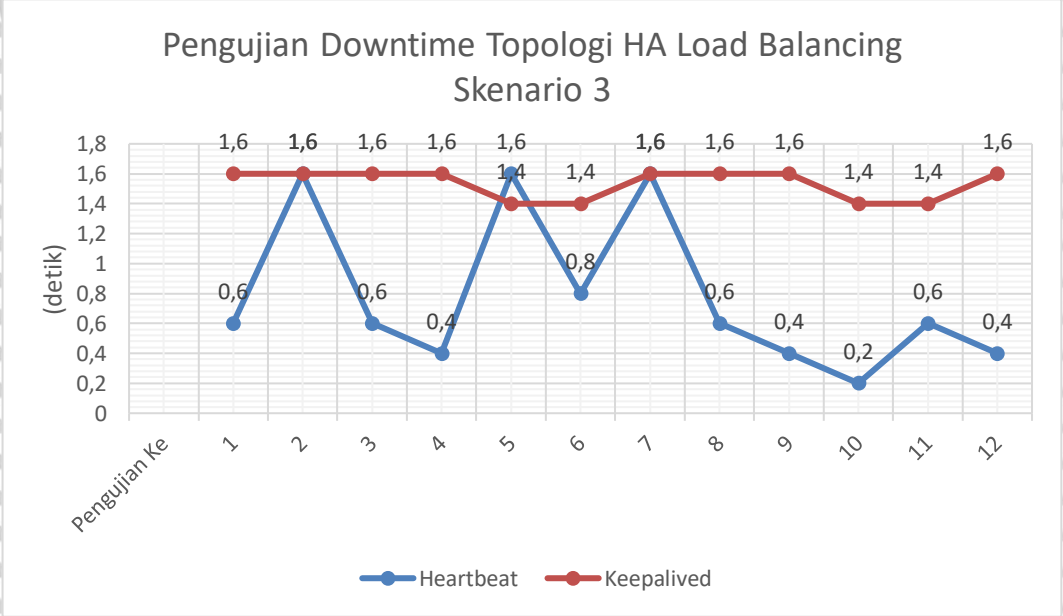
Dari hasil pengujian *downtime* topologi High Availability Load Balancing Skenario 2, diketahui bahwa proses *failover* tidak terjadi. Hal ini disebabkan karena perangkat lunak *failover*, baik Heartbeat maupun Keepalived tidak dapat mendeteksi bahwa status *service* perangkat lunak Haproxy berubah dari aktif menjadi tidak aktif.



Gambar 6.33 Grafik Pengujian Downtime Topologi HA Load Balancingd Skenario 2

6.3.6 Analisis Pengujian Downtime Topologi High Availability Load Balancing Skenario 3

Dari hasil pengujian *downtime* topologi High Availability Load Balancing Skenario 3, diketahui perbandingan waktu *downtime* antara perangkat lunak *failover* Heartbeat dan perangkat lunak *failover* Keepalived.



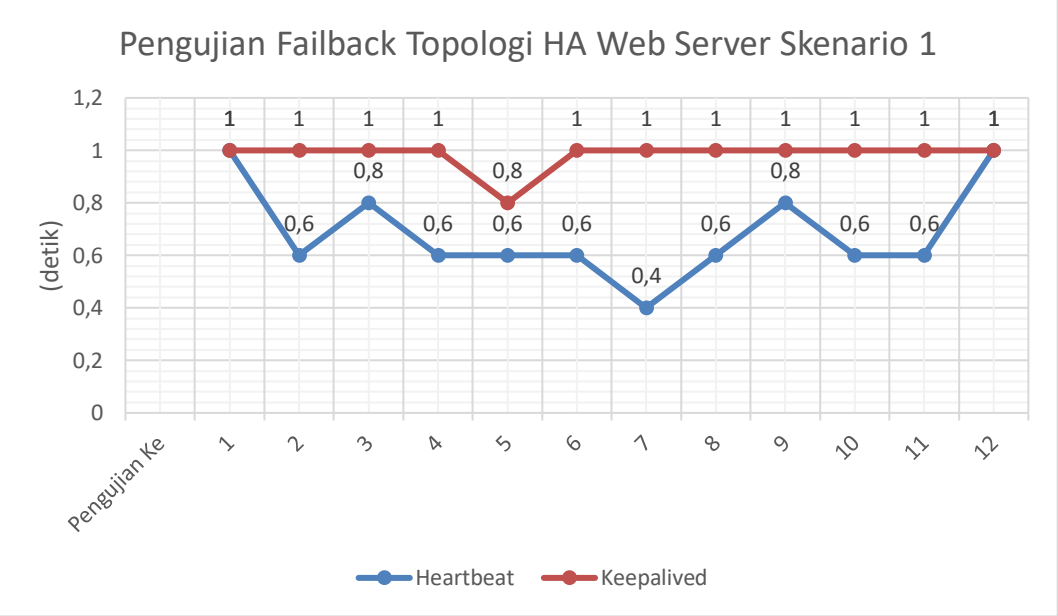
Gambar 6.34 Grafik Pengujian Downtime HA Load Balancing Skenario 3

Dari 12 kali pengujian *downtime* yang dilakukan pada skenario 3 ini, diketahui bahwa perangkat lunak Heartbeat memiliki hasil rata-rata waktu *downtime* sebesar 0,78 detik, sedangkan perangkat lunak Keepalived memiliki hasil rata-rata waktu *downtime* sebesar 1,53 detik. Dengan demikian pada skenario ini, Heartbeat memiliki rata-rata waktu *downtime* yang lebih rendah.

6.4 Analisis Pengujian Failback

6.4.1 Analisis Pengujian Failback Topologi High Availability Webserver Skenario 1

Dari hasil pengujian *failback* topologi High Availability Web Server Skenario 1, diketahui perbandingan waktu *failback* antara perangkat lunak *failover* Heartbeat dan perangkat lunak *failover* Keepalived.



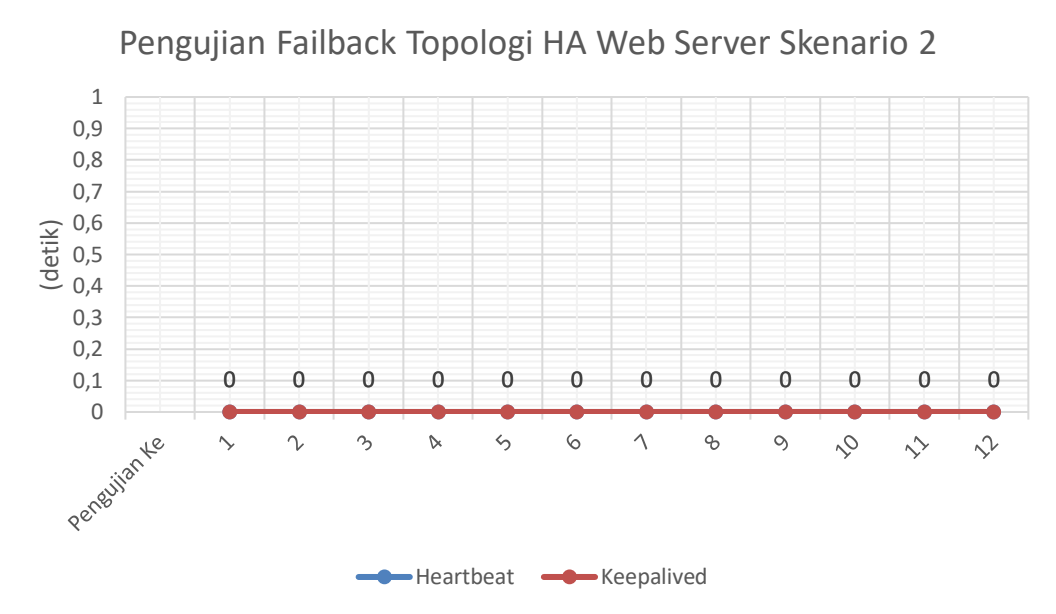
Gambar 6.35 Grafik Pengujian Failback Topologi HA Webserver Skenario 1

Dari 12 kali pengujian *failback* yang dilakukan pada skenario 1 ini, diketahui bahwa perangkat lunak Heartbeat memiliki hasil rata-rata waktu *failback* sebesar 0,68 detik, sedangkan perangkat lunak Keepalived memiliki hasil rata-rata waktu *failback* sebesar 0,88 detik. Dengan demikian pada skenario ini, dapat dikatakan Heartbeat memiliki rata-rata waktu *downtime* yang lebih rendah.

6.4.2 Analisis Pengujian Failback Topologi High Availability Webserver Skenario 2

Dari hasil pengujian *failback* topologi High Availability Web Server Skenario 2, diketahui bahwa proses *failback* tidak terjadi. Hal ini disebabkan karena perangkat lunak *failover*, baik Heartbeat maupun Keepalived tidak dapat mendeteksi bahwa

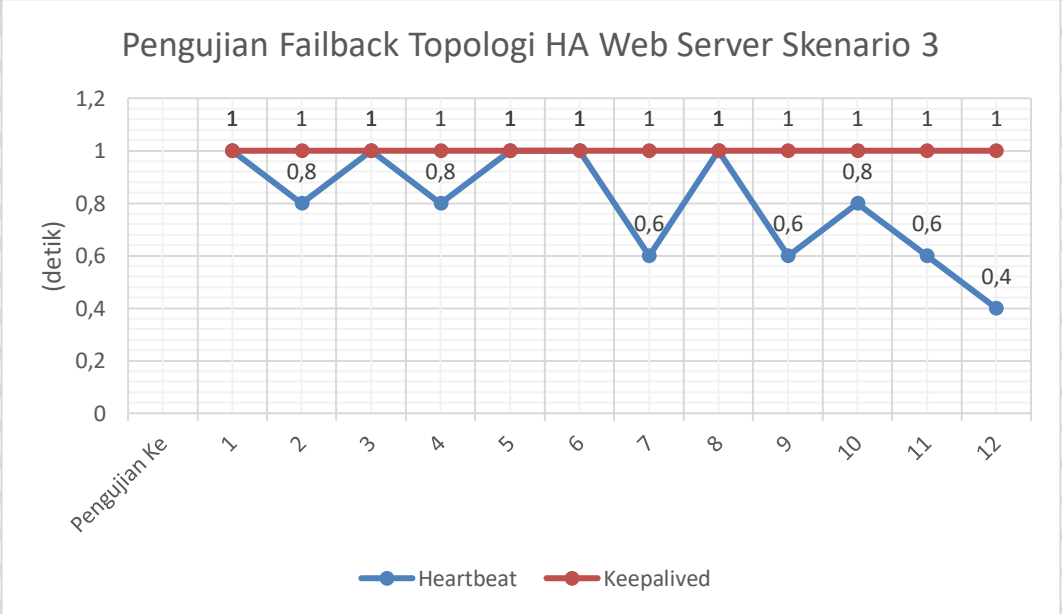
service perangkat lunak Apache2 pada Web Server 1 berubah dari tidak aktif menjadi aktif kembali.



Gambar 6.36 Grafik Pengujian Failback Topologi HA Web Server Skenario 2

6.4.3 Analisis Pengujian Failback Topologi High Availability Webserver Skenario 3

Dari hasil pengujian *failback* topologi High Availability Web Server Skenario 3, diketahui perbandingan waktu *failback* antara perangkat lunak *failover* Heartbeat dan perangkat lunak *failover* Keepalived.

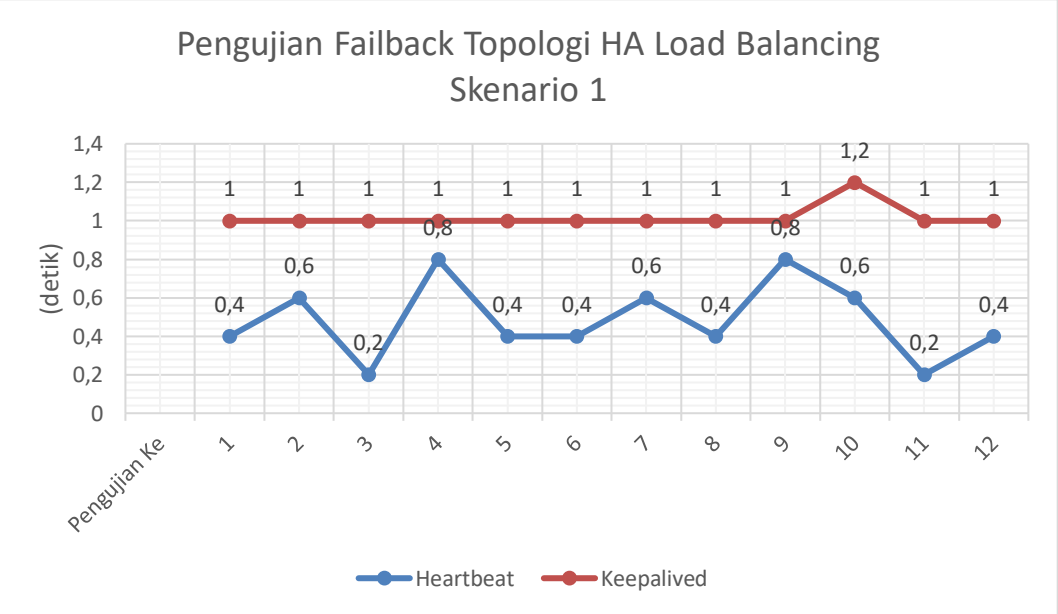


Gambar 6.37 Grafik Pengujian Failback Topologi HA Web Server Skenario 3

Dari 12 kali pengujian *failback* yang dilakukan pada skenario 3 ini, diketahui bahwa perangkat lunak Heartbeat memiliki hasil rata-rata waktu *failback* sebesar 0,88 detik, sedangkan perangkat lunak Keepalived memiliki hasil rata-rata waktu *failback* sebesar 1 detik. Dengan demikian pada skenario ini, Heartbeat memiliki rata-rata waktu *failback* yang lebih rendah.

6.4.4 Analisis Pengujian Failback Topologi High Availability Load Balancing Skenario 1

Dari hasil pengujian *failback* topologi High Availability Load Balancing Skenario 1, diketahui perbandingan waktu *failback* antara perangkat lunak *failover* Heartbeat dan perangkat lunak *failover* Keepalived.

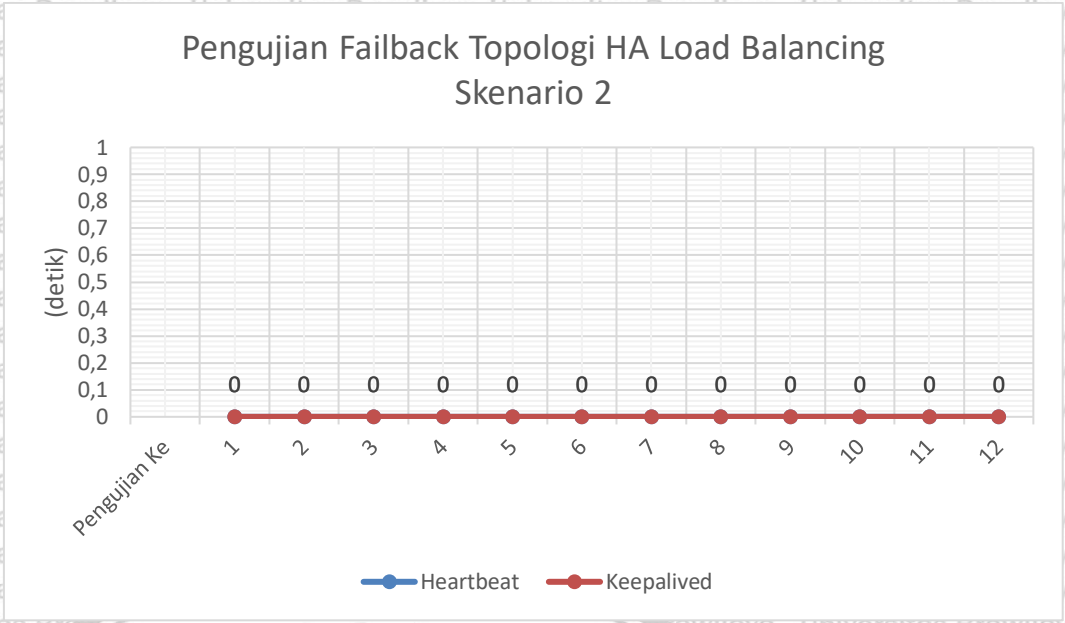


Gambar 6.38 Grafik Pengujian Failback Topologi Load Balancing Skenario 1

Dari 12 kali pengujian *failback* yang dilakukan pada skenario 1 ini, diketahui bahwa perangkat lunak Heartbeat memiliki hasil rata-rata waktu *failback* sebesar 0,48 detik, sedangkan perangkat lunak Keepalived memiliki hasil rata-rata waktu *failback* sebesar 1,01 detik. Dengan demikian pada skenario ini, dapat dikatakan Heartbeat memiliki rata-rata waktu *failback* yang lebih rendah.

6.4.5 Analisis Pengujian Failback Topologi High Availability Load Balancing Skenario 2

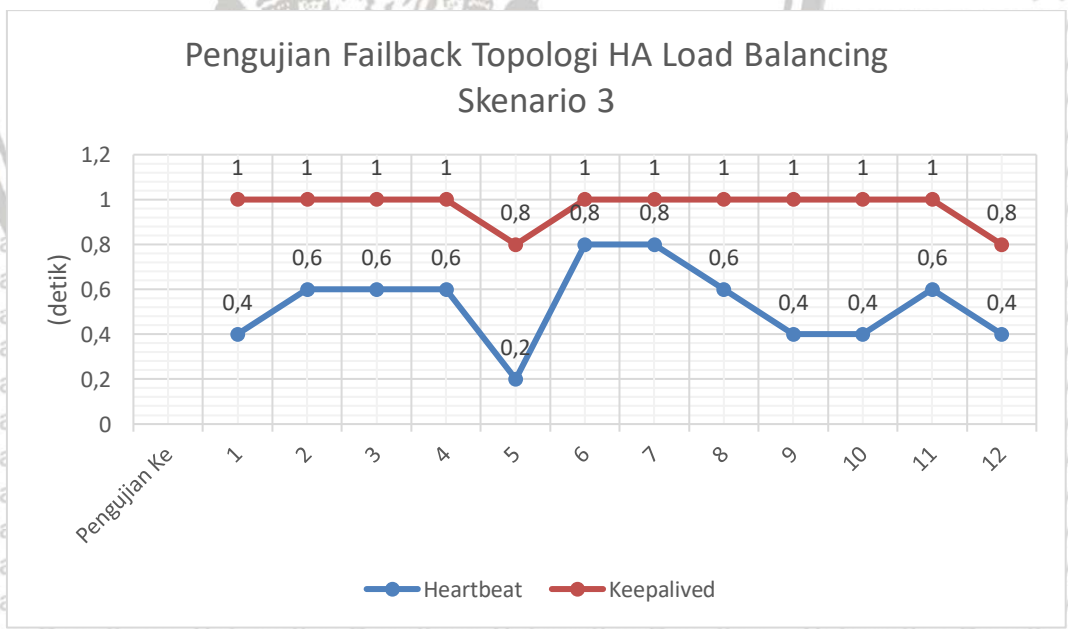
Dari hasil pengujian *failback* topologi High Availability Load Balancing Skenario 2, diketahui bahwa proses *failback* tidak terjadi. Hal ini disebabkan karena perangkat lunak *failover*, baik Heartbeat maupun Keepalived tidak dapat mendeteksi bahwa *service* perangkat lunak HaProxy pada Web Server 1 berubah dari tidak aktif menjadi aktif kembali.



Gambar 6.39 Grafik Pengujian Failback Topologi HA Load Balancing Skenario 2

6.4.6 Analisis Pengujian Failback Topologi High Availability Load Balancing Skenario 3

Dari hasil pengujian *failback* topologi High Availability Load Balancing Skenario 3, diketahui perbandingan waktu *failback* antara perangkat lunak *failover* Heartbeat dan perangkat lunak *failover* Keepalived.



Gambar 6.40 Grafik Pengujian Failback Topologi HA Load Balancing Skenario 3

Dari 12 kali pengujian *failback* yang dilakukan pada skenario 3 ini, diketahui bahwa perangkat lunak Heartbeat memiliki hasil rata-rata waktu *failback* sebesar 0,53 detik, sedangkan perangkat lunak Keepalived memiliki hasil rata-rata waktu

failback sebesar 0,96s detik. Dengan demikian pada skenario ini, Heartbeat memiliki rata-rata waktu *failback* yang lebih rendah.

BAB 7 PENUTUP

7.1 KESIMPULAN

Kesimpulan yang dapat diambil dari penelitian mengenai perbandingan kinerja teknologi failover berbasis klaster (heartbeat) dengan failover berbasis jaringan (keepalived) adalah sebagai berikut :

1. Perangkat lunak *failover* Heartbeat dan Keepalived dapat digunakan untuk implementasi *web server* dan *load balancing* yang bersifat ketersediaan tinggi.
2. Berdasarkan hasil pengujian yang dilakukan pada topologi high availability *web server*, hasil pengujian *downtime* skenario 1 rata-rata waktu heartbeat sebesar 5 detik dan keepalived 4 detik, pada skenario 2 tidak terjadi proses *failover* dan pada skenario 3 rata-rata waktu heartbeat sebesar 0,88 detik dan keepalived 1,51 detik. Hasil pengujian *failback* skenario 1 rata-rata waktu heartbeat sebesar 0,68 detik dan keepalived 0,88 detik, skenario 2 tidak terjadi proses *failback*, skenario 3 rata-rata waktu heartbeat sebesar 0,88 detik dan keepalived 1 detik.
3. Berdasarkan hasil pengujian yang dilakukan pada topologi high availability *load balancing*, hasil pengujian *downtime* skenario 1 rata-rata waktu heartbeat sebesar 4,78 detik dan keepalived 4,05 detik, pada skenario 2 tidak terjadi proses *failover* dan pada skenario 3 rata-rata waktu heartbeat sebesar 0,78 detik dan keepalived 1,53 detik. Hasil pengujian *failback* skenario 1 rata-rata waktu heartbeat sebesar 0,48 detik dan keepalived 1,01 detik, skenario 2 tidak terjadi proses *failback*, skenario 3 rata-rata waktu heartbeat sebesar 0,53 detik dan keepalived 0,96 detik.

7.2 SARAN

Saran bagi penelitian serupa yang akan datang antara lain :

1. Dilakukan pengujian tambahan untuk menghitung penggunaan *memory usage* dan *cpu usage*.
2. Mengimplementasikan rancangan ini ke server dengan kondisi yang nyata.

DAFTAR PUSTAKA

Bookman, C., 2002. Linux Clustering: Building and Maintaining Linux Clusters. 1st penyunt. United States of America: New Rider Publishing.

Cassen, A., 2002. Keepalived For LVS, s.l.: Free Software Foundation.

Cisco, 2018. ASDM Book 1: Cisco ASA Series General Operations ASDM Configuration Guide. [Online] Available at: https://www.cisco.com/c/en/us/td/docs/security/asa/asa96/asdm76/general/asdm-76-general-config.html [Diakses 26 12 2018].

Dani, R., Suryawan, F., 2017. Perancangan Dan Pengujian Load Balancing Dan Failover Menggunakan Nginx. Jurnal Khazanah Informatika, 3(1), p.43-50.

Digital Ocean, 2018. *How To Install Linux, Apache, MySQL, PHP (LAMP) stack on Ubuntu 18.04*. [Online] Tersedia di: https://www.digitalocean.com/community/tutorials/how-to-install- linux-apache-mysql-php-lamp-stack-ubuntu-18-04 [Diakses 10 Februari 2021]

Haproxy, 2018. *The Reliable, High Performance TCP/HTTP Load Balancer* [Online] Tersedia di : http://www.haproxy.org/ [Diakses 10 Februari 2021]

Heidi, E., 2016. What is High Availability?. [Online] Available at: https://www.digitalocean.com/community/tutorials/what-is-high-availability [Diakses 25 12 2018].

Hirt, A., 2009. Pro Sql Server 2008 Failover Clustering. [e-book]. Apress. Tersedia di: Dauodisamir < http://www.daoudisamir.com/references/sql_ebooks/sql_failover.pdf> [diakses 14 Februari 2018]

Irfani, Sulistyanto, Hernawan, 2015. IMPLEMENTASI HIGH AVAILABILITY SERVER DENGAN. [Online] Available at: http://eprints.ums.ac.id/35179/ [Diakses 18 Februari 2018].

Kurniawan, A., 2010. Windows Server 2008 R2 Failover Cluster. [e-book]. Ilmu Data Publisher. Tersedia melalui: Google Books < https://books.google.co.id/books?id=EZJQDwAAQBAJ> [Diakses 23 Mei 2018]

Lakhe, S., Shinde, A., Sukhthankar, N., & Reddy, C. (2016). Serverload Balancing Using Haproxy. www.ejurnal.aessangli.in/ASEEJournals/CE 166.pdf, 1-8

M, Tim Jones. 2012. “High Availability storage with linux and DRBD” Open Networking Foundation. “Software-Defined Networking. The New Norm for Networks,” ONF White Paper.

Purwiadi, M., Yahya, W. & Basuki, A., 2017. High Availability Controller Software Defined Network Menggunakan Heartbeat dan DRBD. Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer, 2(8), pp. 2297-2306.

Robertson, A., 2000. Linux-HA Heartbeat System Design. Atlanta, USENIx.

Ross, K. & Kurose, J., 2013. Computer Networking: a top-down approach. 6 penyunt. New Jersey: Pearson Eduaction, Inc.

Rumbaugh, J., Jacobson, I. & Booch, G., 2005. The Unified Modeling Language reference manual. 2nd ed. Boston: Addison-Wesley. 56 Berndtsson, M., Hansson, J., Olsson, B. & Lundell, B., 2008. Thesis projects: a guide for students in Computer Science and Information Systems. 2nd ed. London: Springer-Verlag London Limited.

Tarreau, W., Haproxy + Heartbeat. [Online] Tersedia di <http://www.formilux.org/archives/haproxy/1003/3259.html> [Diakses pada 16 April 2018]

Techopedia, 2018. *What is a Single Point of Failure (SPOF)*. [Online] Tersedia di: <https://www.techopedia.com/definition/4351/single- point-of-failure-spof> [Diakses 10 Februari 2021].

The Reliable, High Performance TCP/HTTP Load Balancer. [Online] Tersedia di : http://www.haproxy.org/ [Diakses 13 Februari 2018]

Ubuntu., 2018. The leading operating system for Pc, IoT Devices, servers and cloud, [online] tersedia di <https://www.ubuntu.com/> [diakses 16 februari 2016]

Virtualbox Org., 2018. Welcome to Virtualbox.org, [online] tersedia di <https://www.virtualbox.org/> [diakses 13 Februari 2018]

Vugt, S. V., 2014. Pro Linux High Availability Clustering. 1 penyunt. New York: Apress.

Wibowo, M., 2014. *Analisis dan Implementasi Quality of Service (QoS) Menggunakan Ipcop di SMK Muhammadiyah Imogiri*. Yogyakarta: AMIKOM YOGYAKARTA.