IMPLEMENTASI LOAD BALANCING SISTEM CLUSTER SERVER PADA WEB SERVER UNTUK JAMINAN KETERSEDIAAN LAYANAN TINGGI

Adnan Fauzi*)
Departemen Teknik Komputer Fakultas Teknik Universitas Diponegoro
Jalan Prof. Sudharto, Tembalang, Semarang, Indonesia
adnan@live.undip.ac.id

Abstract - Website services have become especially important thing and are the primary need of everyone's information needs. This High information needs require website services to be reliable so that it doesn't disrupt it's user. Problems often happened in the website services when the infrastructure supporting it was having system failure, such as power shortage, natural disaster, hardware failure, and internet network problem etc. To increase website services' system reliability and optimal service, infrastructure system that support server cluster schema is needed. The objective of infrastructure system that support server cluster schema is to increase website services' system reliability and minimize downtime. Server cluster allow website services to look undisturbed on the user's view even though there are some problems happening in the main server such as power shortage, because the website services is being serve by backup server using failover function. Infrastructure's requirement for implementation of cluster server system at least needs to be double from the main system used, so that when the main system had hardware or software problems the backup system can serve website services just as good as the main system. It is hoped that website services will always available and the services is undisturbed even though there are problems in the infrastructure.

Keywords: *Web, Server, Cluster, Downtime, Reliability, Infrastructure.*

I. PENDAHULUAN

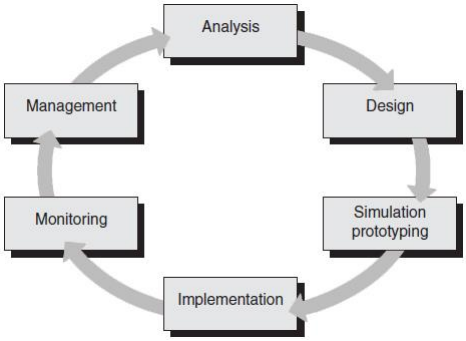
Perkembangan teknologi komputer dan jaringan diikuti juga oleh perkembangan layanan pada *server*. Saat ini banyak penyedia layanan teknologi informasi yang telah merevolusi sistem pelayanannya menggunakan teknologi informasi berbasis layanan *Web* dan *online services*. Namun, seringnya waktu banyak yang tidak memahami kebutuhan secara infrastruktur dan menyiapkan lingkungan yang baik bagi layanan yang berjalan diatasnya. Sehingga, saat terjadi lonjakan pengguna atau pengguna yang terus bertambah tiap harinya, banyak dari aplikasi *web* mengalami kegagalan layanan ataupun *downtime* karena *overload.* Layanan dan data yang disimpan pada mesin *server,* menjadi faktor dasar layanan ini tetap berjalanan sehingga membutuhkan pendukung ketersediaan layanan selain dari infrastruktur jaringan. Semakin banyak pengguna dan data yang perlu disimpan dan olah, diperlukan mesin *server* yang lebih handal dan hal tersebut dapat meningkatkan biaya juga kebutuhan ruang untuk operasionalnya[1]. Selain itu integritas dan ketersediaan layanan menjadi salah satu prioritas untuk meminilisir dampak kerugian saat terjadi kegagalan sistem atau fungsi.

Mesin *server* dapat mengalami kegagalan fungsi disebabkan bencana alam, kegagalan daya dan kerusakan komponen. Jika terjadi kegagalan fungsi, bisa jadi data-data yang besar dan penting dapat hilang atau rusak. Sehingga layanan yang berjalan di mesin akan terhenti dan menggangu operasional aplikasi *web* atau perusahaan hingga mengalami kerugian secara finansial. Sebagai langkah pencegahan dari kerusakan data dan meminimalkan *downtime* saat terjadi kerusakan ataupun pemeliharaan perangkat keras dan perangkat lunak pada mesin *server*, dibutuhkan sistem pada *server* virtualisasi yang dapat mendukung ketersediaan layanan tinggi [1].

Melalui penelitian ini, dilakukan perancangan sebuah sistem penyeimbang beban dengan menggunakan HAproxy, KeepAlived , ProxySQL, dan *Database Cluster* menggunakan MariaDB Galera. Dengan mengunakan HAproxy sebagai penyeimbang beban dan *failover services* dapat digunakan beberapa *web server* sehingga dapat melayani lebih banyak proses untuk melayani jumlah permintaan dari klien yang semakin meningkat. KeepAlived digunakan sebagai teknologi *failover* apabila salah satu mesin penyeimbang beban mengalami kegagalan yang menyebabkan layanan situs *web* menjadi tidak berfungsi. Dan ProxySQL digunakan sebagai penyeimbang beban *database* sekaligus *query routing.*

II. METODE PENELITIAN

Metode pengembangan yang digunakan mengacu pada kerangka kerja metode *Network Development Life Cycle* (NDLC). NDLC terdiri dari enam tahap yaitu analisis, desain, simulasi, implementasi, *monitoring*, dan manajemen. Sepeti ditunjukkan pada Gambar 1.



Gambar 1 Metode pengembangan.

Setiap tahapan merupakan sebuah proses yang menghasilkan keluaran, dimana keluaran tersebut menjadi dasar untuk tahapan selanjutnya. Tahap analisis mencakup analisis kebutuhan fungsional, kebutuhan perangkat keras, dan kebutuhan perangkat lunak. Tahap desain mencakup desain jaringan dan desain sistem berdasarkan spesifikasi kebutuhan yang dibuat. Tahap simulasi dan atau *prototyping* mencakup pembuatan sistem dalam lingkungan percobaan atau perangkat lunak simulasi. Tahap *monitoring* dan manajemen merupakan proses untuk memastikan kelancaran sistem setelah implementasi [4]

A. Analisis Kebutuhan Perangkat Virtual Server

Perangkat *virtual server* yang digunakan untuk membangun sistem *cluster server* pada layanan pada penelitian ini yaitu :

1. Dua unit *virtual server* untuk *load balancer server* dengan spesifikasi masing-masing *server* seperti pada Tabel 1.

Tabel 1 Spesifikasi *Load Balancer Virtual Server*

Komponen	**Server**	
	Virtualisasi 1	**Virtualisasi 2**
Prosesor	4 Core - Intel(R) Xeon(R) Silver 4114 CPU @ 2.20GHz	4 Core - Intel(R) Xeon(R) Silver 4114 CPU @ 2.20GHz
RAM	4 GB	4 GB
Storage	20 GB ZFS SSD Storage	20 GB ZFS SSD Storage
NIC	2 *Virtual Ethernet 10 Gbps*	2 *Virtual Ethernet 10 Gbps*

2. Dua unit *virtual server* untuk *web server* dengan spesifikasi masing-masing *server* ditunjukkan pada Tabel 2.

Tabel 2 Spesifikasi *Web Virtual Server*

Komponen	**Server**	
	Virtualisasi 1	**Virtualisasi 2**
Prosesor	16 Core - Intel(R) Xeon(R) Silver 4114 CPU @ 2.20GHz	16 Core - Intel(R) Xeon(R) Silver 4114 CPU @ 2.20GHz
RAM	16 GB	16 GB
Storage	500 GB ZFS SSD Storage	500 GB ZFS SSD Storage
NIC	1 *Virtual Ethernet 10 Gbps*	1 *Virtual Ethernet 10 Gbps*

3. Tiga unit *virtual server* untuk *database server* dengan spesifikasi masing-masing *server* ditunjukkan pada Tabel 3.

Tabel 3 Spesifikasi *Database Cluster Virtual Server*

Komponen	**Server**		
	Virtualisasi 1	**Virtualisasi 2**	**Virtualisasi 3**
Prosesor	16 Core - Intel(R) Xeon(R) Silver 4114 CPU @ 2.20GHz	16 Core - Intel(R) Xeon(R) Silver 4114 CPU @ 2.20GHz	16 Core - Intel(R) Xeon(R) Silver 4114 CPU @ 2.20GHz
RAM	16 GB	16 GB	16 GB
Storage	200 GB ZFS SSD Storage	200 GB ZFS SSD Storage	200 GB ZFS SSD Storage
NIC	2 *Virtual Ethernet 10 Gbps*	2 *Virtual Ethernet 10 Gbps*	2 *Virtual Ethernet 10 Gbps*

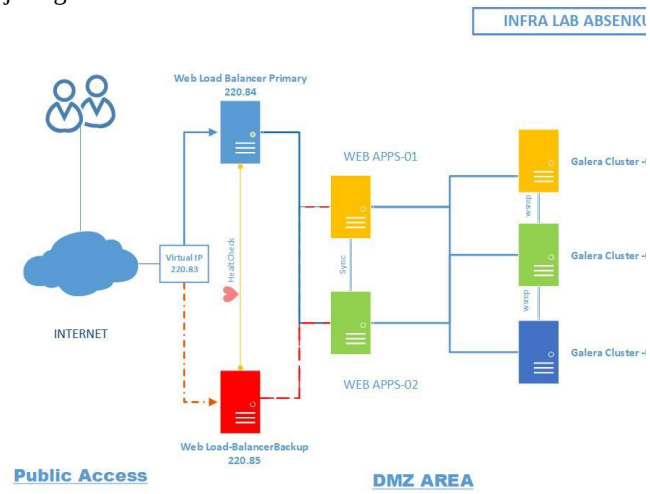
B. Analisis Kebutuhan Perangkat Lunak

Perangkat lunak yang digunakan untuk membangun sistem *cluster server* yang mendukung ketersediaan tinggi di atas lingkungan virtual dan perangkat lunak pendukung lainnya untuk penelitian ini yaitu :

1. Proxmox VE 5.2, digunakan untuk mengimplementasikan dan mengelola mesin *virtual*. Proxmox VE merupakan *distro* Linux *server* dengan kode sumber terbuka untuk lingkungan virtualisasi berbasis distribusi Linux Debian [2].
2. Debian 9.12 Server x64, digunakan sebagai sistem operasi pada mesin virtual. Debian 9.12 Server x64 akan didukung hingga tahun 2022. Pembaruan akan meliputi fitur baru perangkat keras komputer, dan pembaruan keamanan.
3. Xshell 6, digunakan untuk melakukan *remote access* ke *server* virtualisasi dan *switch*. Putty adalah perangkat lunak *remote console*/terminal yang digunakan untuk melakukan *remote access* ke komputer dan perangkat jaringan dengan menggunakan SSH, Telnet, atau serial.
4. Fping, digunakan untuk mengukur *downtime* pada *server*. Fping merupakan program *ping* sederhana yang memiliki kemampuan melakukan *ping* dengan interval lebih cepat dibanding *ping* standar.
5. Rsync, digunakan untuk sinkronisasi file dan folder antara kedua *web server* yang digunakan.
6. Haproxy, digunakan sebagai aplikasi penyeimbang beban *web server*.
7. KeepAlived, digunakan sebagai aplikasi *failover* pada dua *load balancer server*.
8. ProxySQL, digunakan sebagai aplikasi penyeimbang beban pada layanan *database* sekaligus menjalankan *query routing*.
9. Mozilla Firefox sebagai *web browser* untuk mengakses *web management* Proxmox VE.
10. Windows 10 Pro, sistem operasi yang terpasang pada laptop untuk menjalankan Xshell dan *web browser* serta sebagai sistem operasi klien.
11. Microsoft Visio 2016, digunakan dalam perancangan sistem untuk membuat diagram alir dan topologi jaringan.

C. Desain Topologi Jaringan Sistem Cluster Server

Topologi jaringan secara fisik menggambarkan bagaimana perangkat jaringan / *server* terhubung satu sama lain [5]. Desain topologi fisik (*virtual)* sistem ini membutuhkan total tujuh unit *server*, dimana dua unit digunakan sebagai *load balancer*, dua unit digunakan sebagai *web cluster server*, dan tiga unit digunakan sebagai *database cluster server*. Seluruh *virtual server* tersebut berjalan pada mesin fisik yang berbeda-beda untuk mengurangi resiko terjadinya kegagalan sistem pada *hardware case.* Karena menggunakan mesin *virtual* maka media komunikasi yang digunakan juga *virtual* sehingga tidak membutuhkan media kabel seperti UTP ataupun *Fiber Optic.* Gambar 2 menjelaskan desain topologi jaringan sistem *cluster server.*



Gambar 2 Desain topologi jaringan *cluster server*

Desain topologi sistem *cluster server* ini menggunakan alamat *IP Public* dan *IP Private* yang berbeda untuk masing-masing segmen jaringan. Tabel 4 menunjukkan alamat IP yang digunakan dalam desain topologi logis sistem *cluster server*.

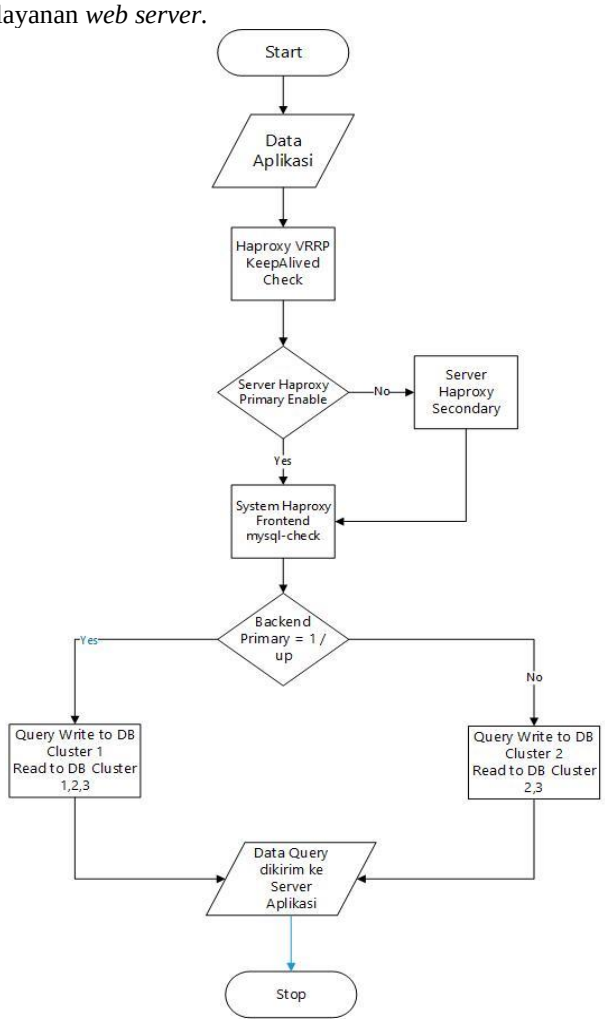
Tabel 4 Alamat IP sistem *cluster server*.

Node / Server Name	***Interface***	**Alamat IP**
Loadbalancer 1	Ens18	103.30.180.234 vIP : 103.30.180.235
	Ens19	172.22.77.14
Loadbalancer 2	Ens18	103.30.180.236 vIP : 103.30.180.235
	Ens19	172.22.77.15
Web Server 1	Ens18	157.119.220.154
	Ens19	172.22.77.16
Web Server 2	Ens18	157.119.220.153
	Ens19	172.22.77.17
Database Cluster 1	Ens18	172.22.77.21
Database Cluster 2	Ens18	172.22.77.22
Database Cluster 3	Ens18	172.22.77.23

D. Cara Kerja Sistem

Sistem *cluster server* yang dibuat memiliki cara kerja yang banyak ditentukan oleh server layanan *proxy load balancer* untuk mengatur setiap *request* yang masuk. Setiap kebutuhan terhadap layanan *website* akan masuk melewati *virtual* IP yang terpasang hanya pada *server load balancer* yang aktif saja karena sudah menerapkan *redundancy* menggunakan *protocol* VRRP (*Virtual Router Redundancy Protocol*). Dimana untuk dapat menggunakan *protocol* ini maka dibutuhkan paling tidak minimal dua server yang saling terhubung menggunakan IP yang masih berada dalam satu *segment*. Alamat IP yang digunakan sebanyak tiga buah untuk kebutuhan alamat server utama (*main*), server cadangan (*backup)* dan satu lagi untuk alamat *virtual* yang akan pindah ke server cadangan jika server utama mengalami kegagalan sistem. Setelah dilakukan proses pada *proxy load balancer*, maka semua *request* layanan akan diteruskan ke server *backend* dengan menerapkan *request and session load balancing.*

Server *backend* adalah server layanan sesungguhnya yang bertugas untuk melayani permintaan layanan. Pada contoh kasus ini server *backend* adalah *web server* yang menggunakan *engine* apache *server* yang terimplementasi pada dua buah server yang saling aktif (mode *active – active).* Semua permintaan layanan yang telah diproses oleh server *load balancer* akan dibagi rata tugasnya kedalam dua *web server*. Namun jika salah satu web server mengalami kegagalan fungsi maka server *load balancer* akan mengalihkan semua layanan hanya ke satu *web server* yang masih dalam keadaan baik. Pada Gambar 3 menunjukkan cara kerja dari sistem *cluster server* pada layanan *web server.*



Gambar 3 Diagram alir cara kerja sistem *cluster server*.

III. IMPLEMENTASI SISTEM

Tahap pertama dalam implementasi pada penelitian ini adalah dengan melakukan sistem operasi pada ketujuh *server* yang akan digunakan untuk kebutuhan sistem *cluster*. Ketujuh *virtual server* tersebut menggunakan sistem operasi Debian 9.11 x64 Server dan menggunakan *share storage* dengan teknologi ZFS SSD *Storage.*

Tahap kedua yaitu konfigurasi jaringan. Konfigurasi jaringan meliputi konfigurasi alamat IP di semua *node*, konfigurasi VLAN, konfigurasi *Domain Name System* (DNS), dan konfigurasi *network* VRRP pada *load balancer server*. Tujuan dari konfigurasi jaringan adalah untuk menghubungkan antar *node* secara logis dengan menggunakan protokol jaringan dan membangun redundansi jalur konektivitas jaringan.

Tahap ketiga yaitu konfigurasi *load balancing web* menggunakan aplikasi Haproxy pada kedua *load balancer server*. Karena secara penggunaan *web* direncanakan adalah penggunaan *web* / aplikasi yang dinamis serta membutuhkan integritas data dan *session* yang baik. Maka untuk penerapan *load balancing web* menggunakan algoritma *Round Robin with Sticky Session.*

Tahap keempat yaitu konfigurasi *web server*. Konfigurasi dilakukan setelah dilakukan installasi kebutuhan *web server* yaitu aplikasi Apache *web server*, PHP-FPM, dan modul – modul lainnya yang dibutuhkan untuk menunjang kebutuhan dari *website* / aplikasi. Selain itu konfigurasi juga dilakukan *tunning* pada konfigurasi PHP dan Apache sehingga dapat maksimal dalam penggunaan *resource* dan lebih handal.

Tahap kelima yaitu konfigurasi untuk sinkronisasi data file maupun folder antar *web server*. Sinkronisasi menggunakan program Rsync yang konfigurasinya disesuaikan kebutuhan dan proses bisnis dari aplikasi yang berjalan pada *web server*. Rsync hanya digunakan pada *directory* tertentu saja yang hanya dibutuhkan untuk disinkronisasikan.

Tahap keenam adalah installasi dan konfigurasi ProxySQL yang digunakan untuk penyeimbang beban dan *query routing* pada layanan *database.* ProxySQL dipasang pada *web server* sehingga nantinya ProxySQL akan menjadi *agent* ataupun perantara yang menghubungkan antara layanan *web* dan layanan *database.* Pada konfigurasi ProxySQL terdapat *query routing* untuk dapat memisahkan antara *query read* dan *query write. Query read* akan diteruskan pada *database cluster 1* dan *database cluster 2*, Sedangkan *query write* hanya akan diteruskan ke *database cluster 3*. Hal ini diimplementasikan untuk dapat mencegah terjadinya tabrakan pada penambahan data dan juga memisah beban *query read* yang biasa lebih dari 70% dari total *query* adalah *query read.*

Tahap ketujuh adalah installasi dan konfigurasi *database cluster*. *Database cluster server* menggunakan *environtment* MariaDB 10.1.4 dengan Galera Cluster 3. Penggunaan *clustering database* dengan metode *multimaster server* diimplementasikan untuk dapat digunakan sebagai *database server active – active* maupun *active – backup.* Layanan *database* yang berjalan akan terhubung ke *web server* melalui perantara dari ProxySQL yang bertugas membuat *connection pool* antara *web server* dan *database server.*

IV. PENGUJIAN SISTEM

Berdasarkan kebutuhan fungsional sistem dari aplikasi *web*. Sistem *cluster server* yang mendukung ketersediaan layanan tinggi, juga harus dapat meningkatkan kehandalan pada aplikasi. Layanan yang dimaksud adalah layanan *web service* dan *database*. Diinginkan saat terjadi kegagalan pada salah satu komponan dalam *cluster server*. Maka server lainnya yang memiliki layanan yang sama akan menggantikan perannya dan layanan tidak akan terganggu ataupun minimum *downtime.* Komponen pada sistem *cluster server* terdiri dari *server* Load Balancer 1, *server* Load Balancer 2, *server* WebApps 1, *sever* WebApps 2, *server* Database 1, *server* Database 2, dan *server* Database 3.

A. Pengujian Kegagalan Komponen Sistem Cluster Server

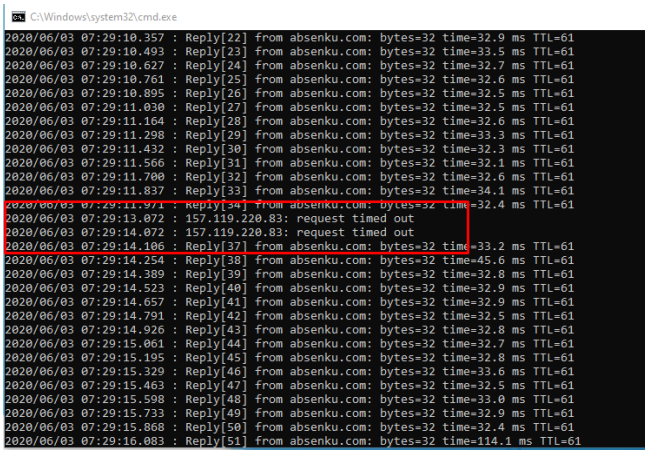
Kondisi layanan *web* / aplikasi saat terjadi kegagalan pada salah satu komponen diamati. Dalam kondisi normal *server* melayani layanan *web* / aplikasi hanya *server* Load Balancer 1, *server* WebApps 1, *sever* WebApps 2, *server* Database 1, *server* Database 2, dan *server* Database 3. Sedangkan *server* Load Balancer 2 hanya *standby* menjadi *backup* jika *server* Load Balancer 1 terjadi kegagalan fungsi / mati. Untuk WebApps 1 dan WebApps 2 berjalan *active – active* melayani *request* HTTP dan HTTPS.

Tabel 5 Pengujian kegagalan salah satu komponen *cluster*.

Komponen Gagal	**Hasil**
server Load Balancer 1	Layanan Web tetap berjalan dengan baik, fungsi *load balancing* langsung ditangani oleh *server* Load Balancer 2.
server Load Balancer 2	Layanan Web tetap berjalan dengan baik, karena *server* Load Balancer 2 hanya sebagai *backup* dari *server* Load balancer 1. Saat kondisi normal,*server* Load Balancer 2 tidak melayani.
server WebApps 1	Layanan Web tetap berjalan dengan baik, karena langsung digantikan perannya oleh serber WebApps 2
sever WebApps 2	Layanan Web tetap berjalan dengan baik, karena langsung digantikan perannya oleh serber WebApps 1
server Database 1	*Query Read :* Layanan web tetap berjalan dengan baik, karena *query read* tetap dilayani oleh *server* Database 2
server Database 2	*Query Read :* Layanan web tetap berjalan dengan baik, karena *query read* tetap dilayani oleh *server* Database 1
server Database 3	*Query Write :* Layanan web tetap berjalan dengan baik, karena *query read* tetap dilayani oleh *server* Database 2, karena memiliki IP yang besarnya dibawah IP *server* Database 3

B. Pengukuran Downtime Saat Jalur Aktif Terputus

Pengukuran *downtime* menggunakan aplikasi Fping, dimana aplikasi tersebut dikonfigurasi untuk melakukan ping ke *server* secara terus menerus dengan interval waktu 1 milisekon / 0,001 sekon . Ukuran paket yang dikirim adalah 32 *byte*. Untuk menghitung waktu *downtime* dapat dilakukan dengan melihat id paket yang hilang ketika proses ping berjalan. Waktu dimulainya *downtime* hingga hidup kembali dihitung sebagai waktu *downtime* yang *real*. Pada Gambar 4 disimulasikan terjadi *downtime* pada *server balancer* yang aktif dan dilakukan *failover* layanan *Virtual IP* secara otomatis ke *server standby*.



Gambar 4. Pengukuran *failover* dan waktu *downtime*

Pada gambar tersebut dapat dilihat terjadi *connection timeout* sebanyak 1 kali dengan total waktu *downtime* kurang lebih 1 sekon. Dan pada Tabel 3 dapat dilihat ringkasan data pengujian sebanyak 50 kali dan hasil rata-rata waktu yang dibutuhkan untuk *failover* layanan ke *server balancer* lainnya adalah 1787 *miliseconds.*

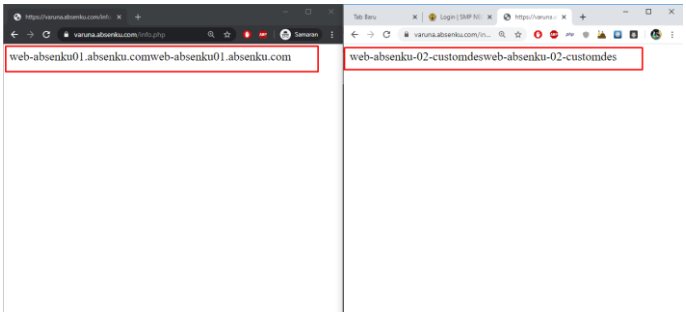
Tabel 3 Pengukuran *Downtime Server Balancer*

Ukur ke-	***Downtime Server* (ms)**	**Keterangan**
1	1013	Layanan *Web* Berjalan Normal Setelah *Downtime*
5	1184	Layanan *Web* Berjalan Normal Setelah *Downtime*
10	2014	Layanan *Web* Berjalan Normal Setelah *Downtime*
20	1300	Layanan *Web* Berjalan Normal Setelah *Downtime*
35	2017	Layanan *Web* Berjalan Normal Setelah *Downtime*
40	2012	Layanan *Web* Berjalan Normal Setelah *Downtime*
50	2012	Layanan *Web* Berjalan Normal Setelah *Downtime*
Rerata 50 x	**1787 ms /** 1,78 Detik	Layanan *Web* Berjalan Normal Setelah *Downtime*

Dapat disimpulkan dari pengujian *downtime* pada sistem *failover* pada *server load balancer* yang dilakukan dalam beberapa kali percobaan didapatkan hasil rata-rata *downtime* adalah 1.787 *miliseconds*.

C. Pengujian *Load Balancing Web Server*

Hasil pengujian pembagi beban pada sistem *cluster server* seperti pada Gambar 8, menunjukkan pembagian beban permintaan layanan web diproses secara seimbang sesuai beban yang sudah diberikan dan dengan respon yang diterima dari masing-masing mesin *server web* (*backend)* yang berbeda. Sistem pada Haproxy akan dengan sendirinya mengetahui beban tertinggi dan terendah pada *server backend* yang bekerja melayani dibelakangnya. Sehingga jika ada permintaan baru masuk, maka akan diteruskan ke mesin *server* yang memiliki beban paling rendah.

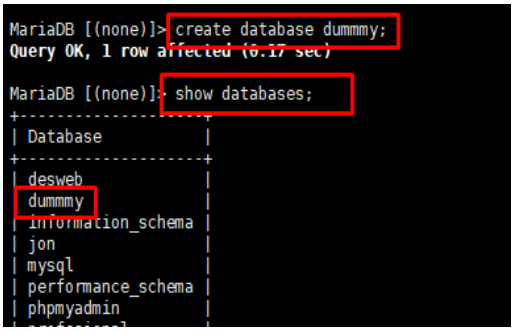


Gambar 4. Pengujian *Load Balancing* pada *Web Server*

Pada gambar diatas, menunjukkan penggunaan algoritma *roundrobin* membuktikan walaupun dalam satu perangkat yang sama dengan IP yang sama, permintaan layanan *web* pada *client* bisa ditangani oleh *server* layanan *web* yang berbeda-beda dan tidak tergantung pada waktu, alamat IP, *network*, maupun perangkat.

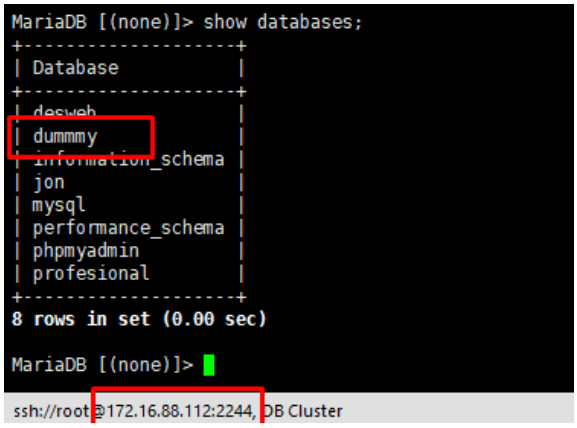
D. Pengujian Replikasi *Database Cluster*

Pada pengujian ini dilakukan dengan menambahkan *database* baru, dengan kondisi awalan *database* yang ada pada setiap *server* sama. Contoh Pengujian tersebut dilakukan pada setiap *server database*, yakni *server database* 1, *server database* 2, *server database* 3. Dengan cara melakukan penambahan *database* baru pada *server database* 3 seperti terlihat pada Gambar 5 dengan hasil replikasi di *server database* 1 dan 3 dapat dilihat pada Gambar 6 dan Gambar 7.

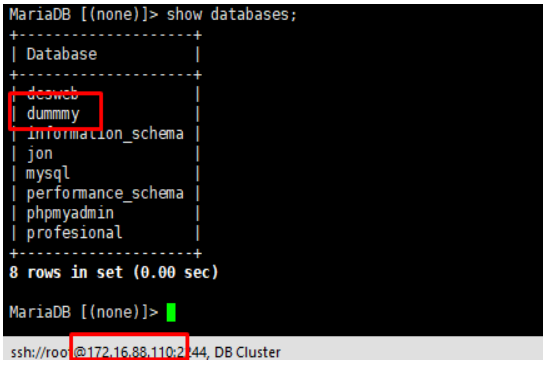


Gambar 5. Membuat *Database* Baru pada *Server Database* 2

Pada Gambar 5 dibuat *database* baru dengan nama dummy dan dapat dilihat pada *server* yang sama *database* tersebut sudah terbentuk. Kemudian dilanjutkan pada *server* 3 dan *server* 1 pada Gambar 6 dan Gambar 7. Terlihat pada daftar *database* , terdapat *database* baru dengan nama dummy.



Gambar 6. *Database* Bertambah pada *Server Database* 3



Gambar 7. Database Bertambah pada *Server Database* 2

Hasil pengujian replikasi MySQL pada setiap *server database* seperti ditunjukkan pada Gambar 6 dan 7 menunjukkan sistem *cluster server database* yang dibangun berhasil melakukan replikasi MySQL secara *multi-master*. Hasil pengujian secara keseluruhan dapat dilihat pada Tabel 1 yang menunjukkan replikasi *multi-master* pada seluruh node yang tergabung dalam sistem *cluster*. Disimulasikan pada setiap *node* dilakukan penambahan *database* baru. Dan dilihat pada *node* lainnya replikasi berjalan sesuai.

Tabel 1 Pengujian Replikasi di *Database Cluster*

Master	**Slave**		**Keterangan**
Server Database 1	*Server Database* 2	*Server Database* 3	Berhasil tersinkronisasi
Server Database 2	*Server Database* 1	*Server Database* 3	Berhasil tersinkronisasi
Server Database 3	*Server Database* 1	*Server Database* 2	Berhasil tersinkronisasi

Dari tabel diatas, didapatkan kesimpulan seluruh *server database* termasuk *server database* 1 dan *server database* 3 dengan menggunakan skenario yang sama berhasil melakukan penambahan data dan langsung tersinkronisasi ke server-server *database* lainnya.

V. KESIMPULAN DAN SARAN

A. Kesimpulan

Setelah melalukan rancang bangun sistem *cluster* server untuk web dan database di lingkungan virtual, maka disimpulkan sistem penyeimbang beban menggunakan Haproxy sudah berjalan sesuai dengan yang direncanakan, dapat menyeimbangkan dan meratakan permintaan *client* ke dalam tiga *server* layanan *web* menggunakan algoritma *roundrobin.*

Kemudian Sistem *cluster* ini sudah mendukung *high availability* atau skema *faillover* pada *service database*, sehingga 1 atau 2 server *database* mati / *down* layanan akan tetap berjalan normal dengan rata-rata *downtime* sebesar 1787 *ms* / 1.78 detik dan untuk replikasi *multi-master* pada *database* server berfungsi secara *realtime* dengan baik menggunakan skema *database cluster* menggunakan galera *cluster*, setiap server dapat menambahkan data dan akan tereplikasi secara *realtime synchronus* ke setiap *server* lainnya pada sistem *cluster.*

B. *Saran*

Berdasarkan kesimpulan yang telah dikemukakan, rancang bangun sistem *cluster* ini masih terdapat beberapa kekurangan untuk nanti diperbaiki antara lain sehingga ada beberapa saran yang dapat disampaikan. Sistem penyeimbang beban ini dapat dikembangkan lagi dengan jumlah web dan *database cluster* yang lebih banyak sesuai kebutuhan dari aplikasi dan banyaknya pengguna.

Pengembangan aplikasi juga harus melihat kebutuhan infrastruktur sebagai salah satu komponen utama dalam menjalankan layanan aplikasi tersebut, sehingga perlu memperhatikan beban-beban yang ditimbulkan dari sisi aplikasi saat layanan sudah digunakan pelanggan / pengguna.

DAFTAR PUSTAKA

[1] Ahmed, W. (2015). Proxmox Cookbook, Birmingham: Packt Publishing

[2] Arfiandi, A,. (2012), Perancangan, Implementasi, dan Analisis Kinerja Virtualisasi Server Menggunakan Proxmox, VMWare ESX, dan Openstack, Jurnal Teknologi, vol.5, no.2, pp. 182-191.

[3] Adi, (2016). Perancangan Sistem Cluster Server untuk Jaminan Ketersediaan Layanan Tinggi pada Lingkungan Virtual, Jurnal Nasional Teknik Elektro dan Teknologi Informasi (JNTETI).

[4] Bhujbal, P. (2014). Anonymity Based Attack on TOR, Global Journal Of Engineering Science And Researches, vol. 1, no. 7, pp. 8–14

[5] Cheng, S. M.C. (2014). Proxmox High Availability. Birmingham: Packt Publishing Ltd.

[6] Dines, R. (2011). The State Of Disaster Recovery Preparedness, Disaster Recovery Journal, vol. 24, no. 1, pp. 1–3.

[7] Hertzog, R. (2012). The Debian Administrator 's Handbook

[8] Irfani, 2015, Implementasi High Availability Server Dengan Teknik Failover Virtual Computer Cluster, Skripsi S-1, Surakarta: Universitas Muhammadiyah Surakarta.

[9] Sofana, I. (2012). Cisco CCNA & Jaringan Komputer, Bandung: Informatika