JITEL (Jurnal Ilmiah Telekomunikasi, Elektronika, dan Listrik Tenaga) p-ISSN: 2774-7972
Vol. 2, No. 2, September 2022, pp. 129-138 e-ISSN: 2775-6696
DOI: https://doi.org/10.35313/jitel.v2.i2.2022.129-138

Analisis performansi *high availability cluster server* menggunakan *heartbeat* pada *private cloud*

Nanda Iryani[1*], Kholifah Dyah Ayatri[2], Reni Dyah Wahyuningrum[3]
[1,2,3]Jurusan Teknik Telekomunikasi, Institut Teknologi Telkom Purwokerto
Jl. D.I Panjaitan No. 128 Purwokerto 53147, Indonesia
[1*]nanda@ittelkom-pwt.ac.id, [2]kholifah@gmail.com, [3]reni@ittelkom-pwt.ac.id

ABSTRAK

Semakin tinggi sebuah permintaan pada *web server* maka beban *web server* akan semakin bertambah dan *server* akan mengalami *down*. Jika menggunakan sebuah *server* saja, tidak dapat disebut handal dikarenakan jika *server* tersebut mati maka sistem akan mati dan tidak dapat melayani sebuah permintaan tanpa adanya *server* cadangan. Pada penelitian ini, akan diterapkan *high availability cluster server* dengan metode *heartbeat* untuk memastikan *failover clustering* disaat *server* utama mati. *Heartbeat* dibangun dalam beberapa *web server* yang terpasang pada *private cloud. Private cloud* digunakan karena memiliki banyak manfaat seperti kendali penuh dapat menyesuaikan penggunaan sumber daya dengan kebutuhan *server* dengan lebih baik. Faktor penelitian berdasarkan dari nilai *downtime* dan nilai *availability* yang menjadikan sebuah parameter *server cluster* dapat berjalan dengan baik. Pengujian dilakukan melalui lima variasi jumlah koneksi, yaitu 200 koneksi, 400 koneksi, 600 koneksi, 800 koneksi, dan 1000 koneksi dengan laju 100 permintaan/detik. Hasil pengujian menunjukkan nilai rata-rata *throughput* semua variasi sebesar 5,254 Mbps dengan tertinggi pada 600 koneksi yaitu 6,233 Mbps. *Delay* menunjukkan kategori sangat baik berdasarkan standarisasi TIPHON yaitu kurang dari 150 ms. Persentasi *packet loss* juga menunjukkan kategori sangat baik hingga 1000 koneksi semua paket diloloskan. Hasil pengujian menunjukkan bahwa *high availability cluster server* menggunakan *heartbeat* pada *private cloud* menunjukan hasil yang sangat baik dimana sistem masih optimal hingga beban 1000 koneksi.

Kata kunci: *web server*, *high availability*, *heartbeat*, *private cloud*

ABSTRACT

The higher a request on the web server, the load on the web server will increase and the server will experience down. If you only use a server, it cannot be called reliable because if the server dies, the system will die and cannot serve a request without a backup server. In this study, a high availability cluster server will be implemented with the heartbeat method to ensure failover clustering when the main server dies. Heartbeat is built on several web servers installed in a private cloud. Private cloud is used because it has many benefits such as full control can better adjust the use of resources to the needs of the server. The research factor is based on the downtime value and availability value which makes a server cluster parameter run well. The test was carried out through five variations of the number of connections, namely 200 connections, 400 connections, 600 connections, 800 connections, and 1000 connections with a rate of 100 requests/second. The test results show the average throughput of all variations is 5.254 Mbps with the highest at 600 connections, which is 6.233 Mbps. The delay shows the very good category based on the TIPHON standard, which is less than 150 ms. The packet loss percentage also shows the very good category up to 1000 connections all packets are passed. The test results show that the high availability cluster server using heartbeat in the private cloud shows very good results where the system is still optimal up to a load of 1000 connections.

Keywords: *web server, high availability, heartbeat, private cloud*

1. PENDAHULUAN

Pada era *new normal* pengguna internet akan bertambah karena untuk pembelajaran daring, kerja daring, dan untuk kegiatan lainnya yang menggunakan internet. Permasalahan dimulai dari kenaikan jumlah *request* pengguna untuk mengakses *web server* sehingga *server* akan terputus dan membutuhkan waktu untuk mengkonfigurasi ulang *server* untuk dapat melayani permintaan pengguna internet. Berdasarkan data statistik laporan survei internet yang dilakukan oleh Asosiasi

Naskah diterima tanggal 26 April 2022, disetujui tanggal 23 Agustus 2022
*E-mail korespondensi

Penyelenggaraan Jasa Internet Indonesia atau APJII total pengguna internet 196,71 juta jiwa dari total popolasi 266,91 juta jiwa penduduk Indonesia. Penetrasi internet mencapai 73,7% pada tahun 2019-2020. Sedangkan survei padatahun 2018 sebanyak 171,17 juta jiwa pengguna internet dari total populasi 264,16 juta jiwa dengan penetrasi internet 64,8%. Peningkatan mencapai 8,9% dari tahun sebelumnya [1].

High availability failover clustering menggunakan metode *heartbeat* menjadi jawaban dengan tetap melayani pengguna internet ketika salah satu *server* mengalami gangguan [2]. Penelitian [3] membuktikan dengan menggunakan *failover clustering* menunjukkan hasil yang baik dari segi *availability* saat *server* mengalami kegagalan. Solusi selanjutnya adalah dengan menggunakan *private cloud.* Karena *private cloud* merupakan metode pemulihan sumber daya *virtual* yang cepat dan handal contohnya pada Openstack saat *server* fisik atau virtual mesin mati. Target utama Openstack adalah menyediakan API *control primitive* dari sumber daya *virtual*. Secara khusus Openstack tidak mendukung file pemulihan sumber daya *virtual* ketika *server* fisik atau mesin *virtual* mati dan penyedia layanan mengalami gangguan untuk memulihkannya. Mekanisme *high availability* dari beberapa sumber jaringan *virtual* telah dibahas dalam komunitas Openstack [4].

Penelitian lainnya menganalisis *availability* dengan metode *failover cluster* komputer. Penelitian ini bertujuan untuk mengantisipasi kerusakan dan kegagalan *server* yangmengganggu kinerja jaringan *server.* Data yang dikirim adalah 120 B per data. Berdasarkan hasil pengujian, sistem dapat melayani transaksi dengan mengirimkan1000 paket data berdasarkan lama waktu 5 menit untuk setiap 10 kali percobaan dan menghasilkan 99% paket data yang diterima dan 1% paket yang gagal [5]. Penelitian [6] menganalisis *failover cluster* untuk pemulihan sistem. Penelitian ini bertujuan untuk menyediakan layanan *web server* untuk membangun sistem dengan *high availability.* Hasil pengujian nilai *availability* dengan *failover cluster* yang dibangun menggunakan fitur *vmware fault tolerance* bekerja dengan baik sesuai konsep kerja dengan nilai *availability* tertinggi adalah 94,44% pada *guest* OS dengan waktu *downtime* 15 menit. Kemudian, penelitian [7] mengoptimalkan *web server* menggunakan *system failover clustering* berbasis *cloud computing.* Penelitian bertujuan untuk mengimplementasikan dan mendesain *system failover clustering* menggunakan *cloud computing* pada *web server.* Pengujian yang dilakukan menggunakan teknik *black box testing* bertujuan untuk mengetahui semua fungsi *software* sudah berjalan dengan baik. Analisa data dengan metode *network development life cycle* dengan parameter yang diukur yaitu nilai *availability, downtime, CPU utilization* dan *throughput.* Hasil yang didapatkan cukup baik dari segi skalabilitas dan *availability* dalam hal pelayanan dan keamanan data serta dapat menangani masalah *downtime.*

Selanjutnya, penelitian [8] mengimplementasilan *cluster server* menggunakan metode *high availability* pada *cloud computing* berbasis Proxmox VE. Hasil dari implementasi sistem *cluster* yang menggunakan metode *high availability* yang didukung DRBD mampu memberikan waktu migrasidan *downtime* yang sangat rendah pada *server* 2 dan 3. Tetapi pada *server* 1 waktu migrasi dan *downtime tinggi* yaitu dengan rata-rata 9,7 s untuk waktu migrasi dan 0,58 ms untuk waktu *downtime*. *Resource* yang tinggi, mampu mengurangi *downtime* yang cukup lama pada *server*. Waktu migrasi dari *server* 1 ke *server* lainnya sebanyak rata-rata 9,7 *second* pada *node*1 ke *node*2. Selanjutnya diperoleh rata-rata waktu migrasi 3,7 s dari *node* 2 ke *node* 3 dan rata-rata 3 *second* dari *node* 3 kembali ke *node* 1. Waktu *downtime* 0,58 ms pada *node* 1, 0,02 ms pada *node* 2 dan 0,02 ms pada *node* 3. Penelitian [9] mengimplementasikan dan menganalisis *high availability network file system pada server cluster.* Penelitian ini bertujuan untuk menganalisis dan mengimplementasikan file *server* jaringan yang memiliki *high availability.* Pengujian dilakukan pada beberapa parameter seperti *throughput*, *latency*, nilai *availability*, serta sinkronisasi data. Hasil dari pengujian nilai akurasi *availability* sebesar 99%. Waktu sinkronisasi data bergantung pada besar data jika data besar waktu yang diperlukan sikronisasi antar node akan semakin lama. Nilai *throughput* dan *latency* lebih besar *node master* daripada *node slave* dikarenakan *server node master* menjadi *server* utama yang diakses. Penelitian [10] mengimplementasikan *failover clustering server* untuk mengurangi resiko *downtime* pada *web server.* Penelitian ini bertujuan untuk menerapkan teknik *failover* pada *high availability clustering web server.* Pengujian yang dilakukan pengujian *failover clustering* dengan pengujian sikronisasi sistem data. Hasil yang didapatkan penerapan *failover clustering server* mengurangi resiko *downtime server.* Penerapan *rsync* menjadikandata dapat rersinkronisasi dari *master server* ke *slave server.*

Penelitian [11] mengimplementasikan *high availability web server* pada *cloud computing* menggunakan Pacemaker. Penelitian ini bertujuan untuk menerapkanmetode *high availability* dengan

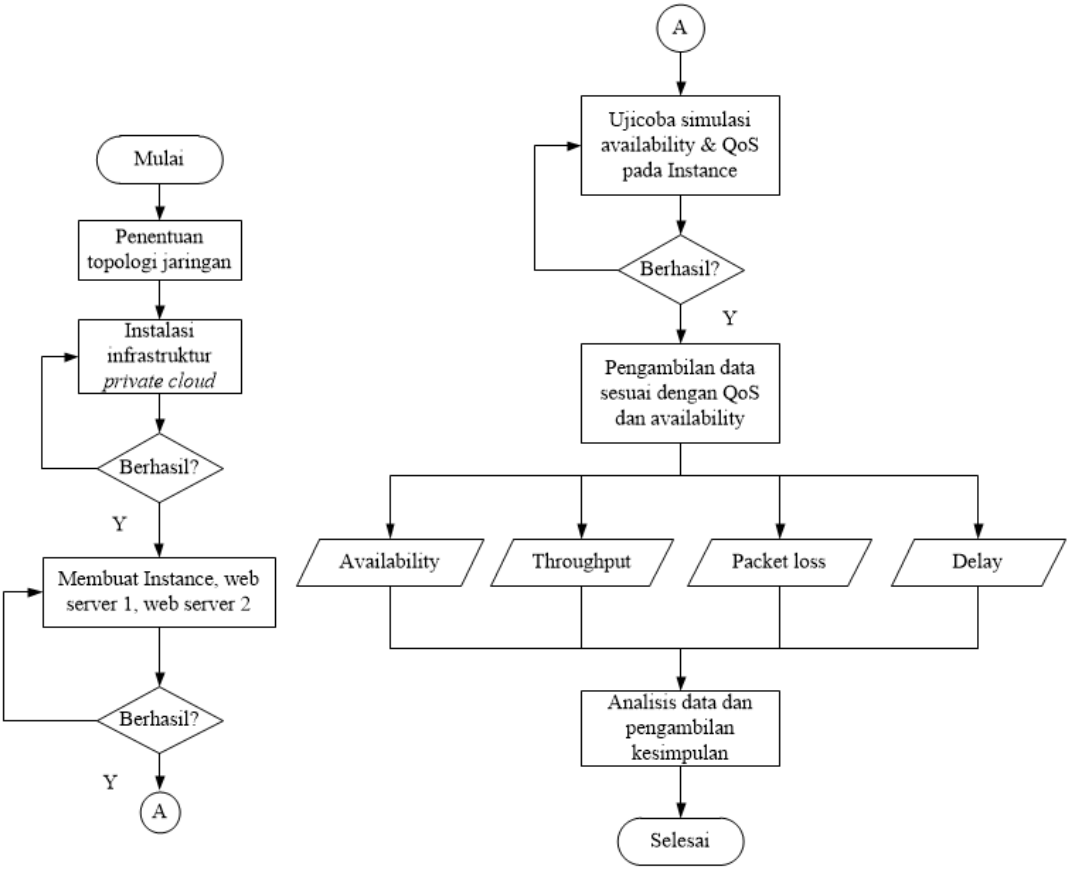
pacemaker sehingga dapat melakukan *backup server* saat *server* utama mati. Pengujian yang dilakukan pada penelitian ini adalah pengujian nilai *availability* dan pengujian penggunaan CPU. Hasil dari pengujian *failover failback* dapat berjalan dengan baik. Nilai *availability* pada tiap *server* sebesar 99,8% waktu yang dibutuhkan pada saat *failover* sebesar 3,6 detik pada *node* 2 untuk beralih ke *node* 1 dan nilai *failback* sebesar 2,8 detik pada *node* 1 untuk beralih ke *node* 2. Hasil dari pengujian CPU dilihat dari presentasi penggunaan CPU client mengakses *web server* memiliki keberhasilan 100% saat 30 *client* mengakses *web,* 97% untuk 60 *client* dan 95% untuk 100 *client.* Terakhir, penelitian [12] mengimplementasikan *service high availability* pada *native server* dan *virtual server* menggunakan Proxmox VE. Tujuan dari penelitian ini adalah menganalisis kinerja dari *high availability* pada dua struktur *server* yang berbeda yakni *native server* dan *virtual server* menggunakan sistem *failover* dan *failback*. Hasil dari penelitian kinerja *high availability* menunjukan bahwasistem *virtual server* memiliki tingkat kinerja lebih rendah dari rata-rata 4,15% dibandingkan dengan sistem *native server*. Tetapi, pada sistem *virtual server* memiliki keunggulan dapat membuat 2 atau lebih mesin *virtual* pada satu mesin *native server*. Sehingga lebih efektif dalam pengelolaan sistem manajemen serverdan lebih ekonomis dari sisi anggaran.

Penelitian ini bertujuan menerapkan *high availability cluster server* dengan metode *heartbeat* untuk memastikan *failover clustering* disaat *server* utama mati. Metode yang digunakan pada penelitian ini menggunakan metode eksperimental dengan melakukan penelitian terhadap hubungan sebab akibat melalui uji coba yang dikendalikan oleh peneliti. Kemudian menggunakan pendekatan kuantitatif yang bersifat objektif dengan data numerik dan statistik. Pendekatan ini mengimplementasikan pola analisis data deduktif dimana analisis data beroperasi pada tahap akhir setelah melakukan percobaan.

2. METODE PENELITIAN

2.1 Konfigurasi Perangkat

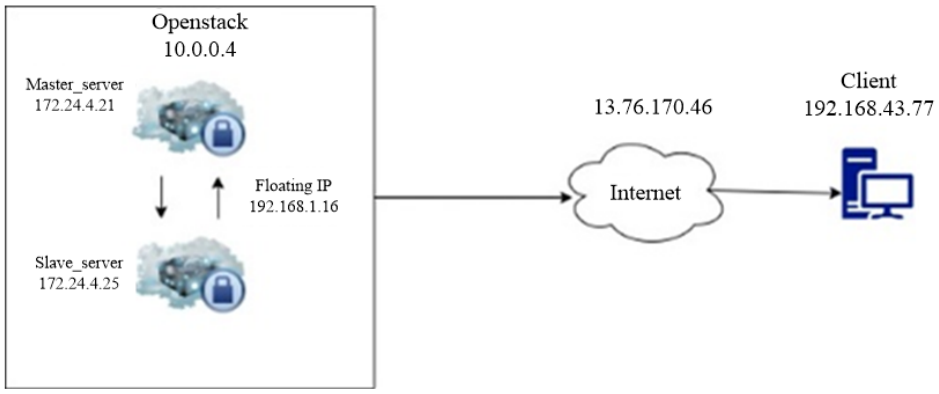
Tahapan-tahapan penelitian digambarkan melalui diagram alir pada Gambar 1.



Gambar 1. Diagram alir penelitian

Pada diagram alir menunjukkan perencangan sistem agar hasil penelitian berhasil dilakukan. Berawal dari penentuan topologi jaringan sebagai dasar menjalankan *failover clustering* pada *private cloud*. Selanjutnya dilakukan instalasi *private cloud* menggunakan Openstack Devstack yang bekerja pada *node* Openstack *server* dengan sistem operasi Ubuntu Server 18.04. Apabila berhasil, maka dilanjutkan dengan membuat dua *instance web server*. Setelah itu dilakukan percobaan terhadap sistem *failover clustering* yang telah dikonfigurasi. Apabila berhasil, kemudian melakukan pengujian beberapa parameter yang telah ditentukan. Setelah itu, dilakukan pengambilan data hasil pengujian berupa parameter *availability, downtime, respons time, QoS throughput, delay*, dan *packet loss.*

Perancangan topologi jaringan pada penelitian ini diperlihatkan pada Gambar 2 yang terdiri dari satu *client* yang digunakan untuk melakukan simulasi pengujian, satu *server* untuk memasang infrastruktur layanan *private cloud* menggunakan Openstack, kemudian terdapat *virtual switch* agar terhubung dengan jaringan *internal* untuk kedua *web server. Master server* dan *slave server* akan saling bergantian kerja jika salah satu *server* ada yang mengalami gangguan. Dari sisi *client* akan mengakses *floating* IP yang terhubung dengan kedua *server*. Adapun spesifikasi perangkat yang digunakan dapat dilihat pada Tabel 1.



Gambar 2. Topologi jaringan

Tabel 1. Spesfikasi dan IP *address* perangkat

Perangkat	Spesifikasi	Keterangan
Client	Sistem operasi	ElementaryOS 5.1.6 Hera
	RAM	8 GB
	Hard disk	150 GB
	Alamat IP	192.168.43.77
Openstack *server*	Sistem operasi	Ubuntu Server 18.04
	RAM	14 GB
	Hard disk	64 GB
	Alamat IP	10.0.0.4/24
Web server 1	Sistem operasi	Ubuntu Server 18.04
	RAM	2 GB
	Hard disk	20 GB
	Alamat IP	172.24.4.21/24
Web server 2	Sistem operasi	Ubuntu Server 18.04
	RAM	2 GB
	Hard disk	20 GB
	Alamat IP	172.24.4.25/24

2.2 Instalasi dan Konfigurasi Sistem *Heartbeat* pada *Private Cloud*

Uji coba dilakukan untuk memastikan sebuah *web server* akan terus menjalankan perintah dari *client* walaupun *server* mengalami gangguan. Sistem ini yaitu *failover clustering* menggunakan *heartbeat* yang berjalan dengan baik pada *private cloud*. Sebelum membuat sistem *failover clustering*, peneliti terlebih dahulu melakukan beberapa konfigurasi.

Konfigurasi Openstack menggunakan Devstack dilakukan untuk merancang*private cloud* sebagai *cloud environment* untuk menerapkan sistem *failover cluster* menggunakan *heartbeat*. Konfigurasi utama yaitu menentukan *username* dan *password* serta *host* IP untuk mengakses *dashboard* Openstack. Konfigurasi berada pada *file local.conf* di direktori */home/devtack* dari *node* Openstack *server*. Gambar 3 menunjukkan konfigurasi Openstack.

```
[[locallocalrc]]

# Password untuk KeyStone, Database, RabbitMQ dan Service
ADMIN_PASSWORD=rahasia
DATABASE_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
SERVICE_PASSWORD=$ADMIN_PASSWORD
HOST_IP=10.0.0.4
```

Gambar 3. *File* konfigurasi Openstack

Sebelum melakukan konfigurasi *heartbeat*, peneliti terlebih dahulu melakukan konfigurasi IP *address* pada kedua *web server*, kemudian melakukan tes koneksi kepada kedua *web server* dengan menguji coba melakukan akses pada *browser* atau terminal pada PC *host*. Setelah itu, membangun *web server* dengan Apache2 dan memulai *service* Apache2 dan mengakses kedua *web server* melalui *browser*.

Konfigurasi *heartbeat* terdiri dari tiga konfigurasi yaitu *ha.cf*, *authkeys* dan *haresources*. *File* konfigurasi *heartbeat* disimpan dalam */etc/ha.d* dan dikonfigurasikan pada kedua *web server*. Gambar 4(a) merupakan konfigurasi global pada *heartbeat*. Kemudian Gambar 4(b) berisi keamanan yang memberikan *node* untuk mengautentikasi ke *cluster*. Terakhir pada Gambar 4(c) untuk menentukan layanan yang digunakan oleh *cluster* dan *node* yang merupakan pemilik layanan.

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

```
node master
ucast ens3 172.24.4.21
node slave
ucast ens3 172.24.4.25
```

(a)

```
vim /etc/ha.d/authkeys
```

```
auth 1
1 crc
```

```
chmod 600 /etc/ha.d/authkeys
```

(b)

```
nano /etc/ha.d/haresources
```

```
master 192.168.1.16
```

(c)

Gambar 4. Konfigurasi *heartbeat*: (a) konfigurasi *ha.cf*, (b) konfigurasi *authkeys*; (c) konfigurasi *haresource*

3. HASIL DAN PEMBAHASAN

Nilai dari setiap parameter *Quality of Service*s (QoS) didapatkan melalui lima variasi pengujian dengan jumlah koneksi yang berbeda-beda yaitu 200 koneksi, 400 koneksi, 600 koneksi, 800 koneksi dan 1000 koneksi. Permintaan *rate* rata-rata setiap koneksi adalah sebesar 100 permintaan per detik, dimana pengujian dilakukan berulang sebanyak 30 kali untuk mendapatkan hasil yang akurat serta mengurangi resiko kesalahan pengujian.

3.1 Proses Pengujian

Pengujian bertujuan untuk mencari nilai QoS saat *web server* dalam keadaan sepi pengujung dan saat ramai pengunjung. Dibuat 5 variasi pengujian jumlah koneksi yang berbeda-beda. Setiap variasi dilakukan uji coba sebanyak 30 kali, sehingga akan didapatkan data QoS rata-rata setiap parameter yang telah ditentukan. Pengujian *web server* juga dilakukan dengan menggunakan aplikasi *httperf*, kemudian *capture* jaringan menggunakan Wireshark untuk mendapatkan data yang akan diolah sesuai

dengan parameter QoS. Jumlah koneksi, request per detik, dan banyaknya pengujian ditunjukkan pada Tabel 2.

Tabel 2. Skenario pengujian

No.	Jumlah koneksi	Permintaan per detik
1	200	100
2	400	100
3	600	100
4	800	100
5	1000	100

Proses pengujian dilakukan untuk mendapatkan hasil kinerja *web server*. Dilakukan lima skenario pengujian dengan jumlah koneksi yang berbeda-beda. Setiap skenario didapatkan data QoS rata-rata per parameter yang telah ditentukan. Pengujian *web server* menggunakan *stress tool* HTTPerf dilakukan dengan memasukkan jumlah koneksi pada "num-conn" dan permintaan per detik pada "rate", selanjutnya diarahkan ke alamat *public* IP yang sebelumnya dibuat di Openstack. Paket data kemudian di-*capture* dan di-*filter* berdasarkan *transfer protocol* HTTP menggunakan Wireshark untuk mendapatkan data yang akan diolah sesuai dengan parameter QoS.

3.2 Pengukuran Nilai *Availability*

Availability diukur berlandaskan "nine", semakin banyak "nine" maka semakin bagus tingkat ketersediaan suatu sistem. Pengujian ini dilakukan dengan mengamati apakah services web yang berjalan pada *cluster* tetap dapat berjalan saat terjadi kegagalan. Adapun hasil pengukuran *availability* ditunjukkan pada Tabel 3.

Tabel 3. Hasil uji coba *availability*

Downtime		**Penyebab**	**MTBF (menit)**	**MTTR (menit)**	***Availability* (%)**
Tanggal	**Waktu**				
9 Februari 2022	22.30	Kedua *server* mati, *master server* dihidupkan	600	0,15	99,97
10 Februari 2022	14.30	Kedua *server* hidup, *master server* dimatikan	960	0,11	99,98
11 Februari 2022	20.00	*Master server restart*	1770	2,11	99,88
12 Februari 2022	05.00	*Master server* mati, *slave server* hidup, *master server* dihidupkan	540	0,15	99,97
13 Februari 2022	05.00	Kedua *server* mati, *slave server* dihidupkan	1440	0,13	99,99
Rata-rata					99,95

Pengujian *availability* menggunakan skenario *mean time to repair* (MTTR) merupakan berapa lama waktu yang diperlukan untuk mengembalikan layanan pada saat terjadi kegagalan pada *cluster*, sedangkan *mean time between fail* (MTBF) merupakan parameter yang menunjukan berapa lama waktu menit *uptime* terdapat 2,65 menit yang digunakan untuk proses pemulihan *web server* agar dapat diakses kembali. Rata-rata *availability* pada saat pengujian sebesar 99,95%. Melalui percobaan pada Tabel 3 waktu terlama untuk pemulihan web server agar dapat diakses kembali adalah pada saat melakukan *restart* pada *master server*. Layanan kembali dan dapat diakses setelah 2,11 menit

kemudian. Pada saat pengujian dengan *restart* dan *power off* pada salah satu *node* yang ada pada Openstack layanan *web server* dapat diakses *server* lain yang masih aktif sehingga waktu pemulihannya yang cepat.

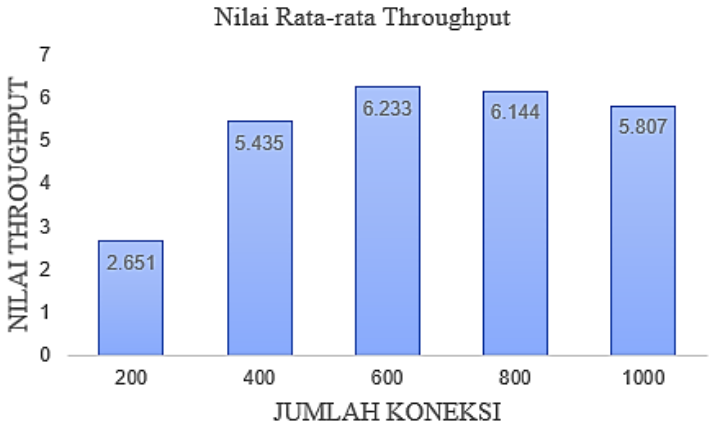
Tabel 4. Hasil uji coba availability dengan variasi koneksi

Jumlah koneksi	Hasil (%)	*Respons time* (s)
200	99,99	1,43
400	99,99	1,82
600	99,98	1,65
800	99,99	1,73
1000	99,99	1,70

Tabel 4 menunjukkan pengukuran *high availability* yang dilakukan untuk mengetahui *realibility server.* Nilai *availability* dan *respons time* didapatkan dengan menggunakan *stress tools siege*. Pada 200 koneksi, didapatkan nilai *availability* 99,99% dengan *respons time* 1,43 s. Selanjutnya 400 koneksi, didapatkan nilai *availability* 99,99% dengan *respons time* 1,82 s. Kemudian pada 600 koneksi, didapatkan nilai *availability* 99,98% dengan *respons time* 1,65 s. Pada 800 koneksi nilai *availability* 99,99% dengan *respons time* 1,73 s, dan 1000 koneksi didapatkan nilai *availability* 99,99% dengan *respons time* 1,70 s. Berdasarkan hasil tersebut, dapat disimpulkan bahwa *server* dapat berjalan dengan baik hingga 1000 koneksi.

3.3 Analisis Nilai *Throughput*

Pengukuran *throughput* dilakukan untuk mengukur kinerja jaringan sebenarnya pada saat melakukan transmisi data. Nilai *throughput* didapatkan dari pembagian jumlah paket data yang diterima dengan waktu pengiriman data. Menurut pengertian tersebut dapat disimpulkan bahwa semakin tinggi nilai *throughput*, maka semakin baik kualitas jaringannya. Wireshark telah memberikan nilai *throughput* dari paket data yang telah di-*capture* melalui menu Statistic. Adapun nilai rata-rata *throughput* yang didapatkan pada setiap variasi pengujian ditunjukan pada Gambar 5.



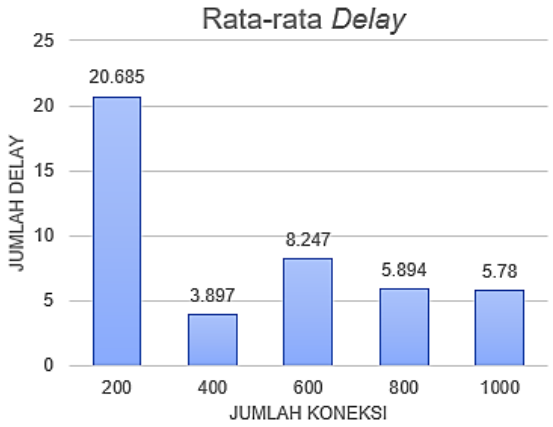
Gambar 5. Grafik *throughput*

Pada 200 koneksi, didapatkan nilai rata-rata *throughput* sebesar 2,651 Mbps yang nilainya meningkat pada 400 koneksi menjadi 5,435 Mbps dan kembali meningkat pada 600 koneksi menjadi 6,233 Mbps. Pada 800 koneksi terjadi penunurunan jumlah *throughput* menjadi 6,144 Mbps dan mengalami penurunan lagi pada 1000 koneksi menjadi 5,807 Mbps. Kenaikan nilai *throughput* karena penambahan jumlah koneksi secara otomatis meningkat jumlah permintaan terhadap *web server* dalam waktu pengiriman yang pendek, sehingga terjadi peningkatan performa *web server.* Kemudian mengalami penurunan nilai *throughput* disebabkan karena penambahan jumlah koneksi. Namun penurunan yang terjadi tidak terlalu banyak sehingga pada *web server* yang menggunakan *high*

availability cluster server menggunakan *heartbeat* masih dapat berjalan dengan baik pada 1000 koneksi.

3.4 Analisis Nilai *Delay*

Pengukuran *delay* dilakukan untuk mendapatkan informasi waktu yang diperlukan paket hingga sampai ke tujuan selama proses transmisi. Rata-rata *delay* dihasilkan dengan pembagian total *delay* keseluruhan dengan total paket yang diterima. Nilai-nilai tersebut terdapat pada menu Statistic Wireshark setelah *capture* paket. Berikutnya dilakukan perhitungan dengan pengkategorian berdasarkan standar *delay* dari THIPON, sehingga dihasilkan rata-rata *delay* yang ditunjukkan Gambar 6.

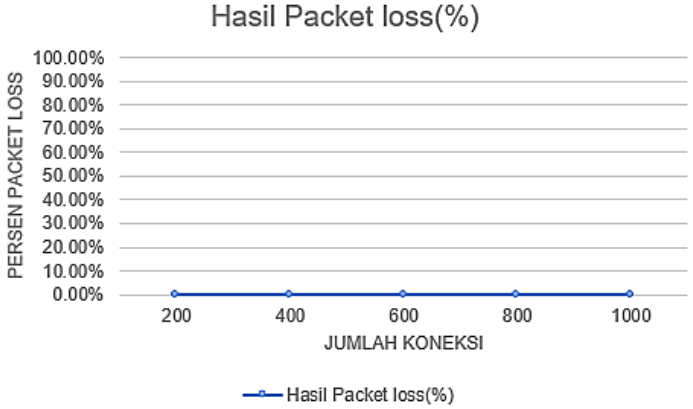


Gambar 6. Grafik *delay*

Pada 200 koneksi, didapatkan nilai *delay* sebesar 20,685 ms, kemudian *trend* rata-rata *delay* mengalami penurunan secara drastis pada 400 koneksi sebesar 3,897 ms dan mengalami peningkatan pada 600 koneksi menjadi 8,247 ms. Kemudian nilai *delay* mengalami penurunan pada 800 dan 1000 koneksi menjadi 5,894 ms dan 5,78 ms. Berdasarkan nilai rata-rata *delay* yang dihasilkan, sistem masih dapat bekerja optimal hingga 1000 koneksi.

3.5 Analisis Nilai *Packet Loss*

Pengukuran *packet loss* dilakukan untuk mendapatkan gambaran tentang jumlah total paket yang *error* selama proses transmisi. Presentasi rata-rata *packet loss* didapatkan dari hasil pembagian antara total *error* yang dihasilkan dengan jumlah permintaan yang dikirim, kemudian dikonversi dalam bentuk persen, Total *error* didapatkan dari *output* HTTPerf, sedangkan jumlah permintaan sama dengan jumlah koneksi, karena setiap koneksi mengirimkan 1 permintaan. Nilai presentasi *packet loss* setiap variasi pengujian ditunjukkan Gambar 7 dan dikategorikan berdasarkan standarisasi dari TIPHON untuk mendapatkan nilai kelayakan *packet loss*.



Gambar 7. Grafik *packet loss*

Nilai kelayakan *packet loss* untuk variasi pengujian 200 hingga 1000 koneksi menunjukkan kategori sangat baik yang berarti sistem dapat melayani semua permintaan dengan minimum paket hilang selama proses transmisi, yaitu 0,0%. Semua paket dapat diloloskan hingga 1000 koneksi. Berdasarkan presentasi rata-rata *packet loss,* system bekerja sangat baik hingga 1000 koneksi.

4. KESIMPULAN

Implementasi *high availability cluster server* menggunakan *heartbeat* pada *private cloud* menunjukkan hasil yang sangat baik, dimana sistem masih optimal hingga beban 1000 koneksi. Berdasarkan hasil data dan analisis yang telah dilakukan, maka didapatkan kesimpulan yaitu nilai rata-rata *high avalability* dari semua skenario mendapatkan hasil yang baik yaitu sebesar 99,95% dan 99,98% pada semua variasi koneksi, sehingga *server* dapat berjalan dengan baik walaupun salah satu *server* mengalami gangguan. Nilai rata-rata *throughput* yang dihasilkan pada setiap koneksi sebesar 5,254 Mbps dengan kualitas jaringan terbaik dari sisi *throughput* adalah ketika terdapat 600 *request* ke *web server* yaitu sebanyak 6,233 Mbps. Kualitas jaringan dari sisi *delay* menunjukkan nilai kelayakan sangat baik berdasarkan standarisasi TIPHON yaitu kurang dari 150 ms. Presentasi *packet loss* menunjukkan nilai kelayakan sangat baik berdasarkan standarisasi TIPHON dengan tidak ada paket yang hilang pada saat pengiriman hingga 1000 koneksi. Penelitian selanjutnya dapat digunakan DRBD *cluster* dan jumlah koneksi ditingkatkan lagi lebih dari 1000 disesuaikan dengan perangkat yang lebih baik.

REFERENSI

[1] (2019), Asosiasi Penyedia Jaringan Internet Indonesia, "Penetrasi & Profil Perilaku Pengguna Internet Indonesia 2018," [Online]. Available: www.apjii.or.id.

[2] S. Winarso and M. Edi, "Implementasi dan Analisis High Availability Server dengan Teknik Failover Clustering Menggunakan Heartbeat," *Artikel Ilmiah Universitas Kristen Satya Wacana Salatiga*, no. 672012133, pp. 1-26, 2017.

[3] Y. Pribadi, A. B. Putra Negara, and M. A. Irwansyah, "Analisis Penggunaan Metode Failover Clustering untuk Mencapai High Availability pada Web Server (Studi Kasus: Gedung Jurusan Informatika)," *J. Sist. dan Teknol. Inf.*, vol. 8, no. 2, pp. 218, 2020.

[4] Y. Yamato, Y. Nishizawa, S. Nagao, and K. Sato, "Fast and Reliable Restoration Method of Virtual Resources on OpenStack," *IEEE Trans. Cloud Comput.*, vol. 6, no. 2, pp. 572–583, 2018.

[5] R. Dewantara, S. Iskandar, and A. Fatwanto, "Availability Analysis with Failover Computer Cluster Method Case Study in Academic Information System of UIN Sunan Kalijaga," *IJID (International Journal on Informatics for Development)*, vol. 6, no. 2, pp. 46-50, 2017.

[6] H. E. Wahanani, "Performance Analysis of Failover Cluster For System Recovery," *Proceeding International Conference on Information Technology and Business*, 2017, pp. 88–92.

[7] M. Muhajirin. "Optimalisasi Web Server Menggunakan System Failover Clustering Berbasis Cloud Computing," *Jurnal Ilmiah Ilmu Komputer*, vol. 3, no. 2, pp. 35-42, 2017.

[8] N. D. N. Linda Apriliana and U. Darusala, "Pengembangan Laboratorium Virtual untuk Simulasi," *JOINTECS) J. Inf. Technol. Comput. Sci.*, vol. 4, no. 1, pp. 2541–3619, 2019.

[9] C. Umam, L. B. Handoko, and G. M. Rizqi, "Implementation And Analysis High Availability Network File System Based Server Cluster," *J. Transform.*, vol. 16, no. 1, p. 31, 2018.

[10] D. H. Abdul, "Impelementasi Failover Clustering Server untuk mengurangi Resiko Downtime Pada Web Server," *J. Chem. Inf. Model.*, vol. 53, no. 9, pp. 76–78, 2018.

[11] N. D. S. R. Suhatman and I. Muslim, "Implementasi High Availability Web Server pada Cloud Computing Menggunakan Pacemaker," *J. Sntiki-10*, no. November, pp. 268–275, 2018.

[12] D. Irwan, H. Sukoco, and S. Wahjuni, "Service High Availability Pada Native Server dan Virtual Server Menggunakan Proxmox VE," *J. Kaji. Ilm.*, vol. 20, no. 2, pp. 137–144, 2020.