Jurnal Politeknik Caltex Riau
http://jurnal.pcr.ac.id

# Implementasi Multi Server Data Storage Pada Cloud Computing

**Jasnica H[1], Yuli Fitrisia[2] dan Muhammad Arif Fadhly Ridha [3]**

[1]Politeknik Caltex Riau, email: jasnica@alumni.pcr.ac.id
[2]Politeknik Caltex Riau, email: uli@pcr.ac.id
[3]Politeknik Caltex Riau, email: fadhly@pcr.ac.id

**Abstrak**

*Kebutuhan akan informasi yang selalu up-to-date dan bisa diakses dimana saja dan kapan saja semakin meningkat. Hal ini menyebabkan perlu adanya layanan cloud berupa data storage yang high availability. Dengan adanya high availability, dapat membantu mengurangi terjadinya kegagalan server. Salah satu cara untuk meningkatkan kualitas layanan tersebut, yaitu mengimplementasikan multi server data storage. Implementasi yang dilakukan menerapkan beberapa teknik seperti data replication, load balancing dan failover. Data replication menggunakan GlusterFS, load balancing menggunakan HAProxy dan failover menggunakan Heartbeat. Dari hasil penelitian ini didapatkan bahwa rata – rata data berhasil di unggah 84.5 % dan di unduh kembali 100% dari data storage. Hasil keberhasilan metode failover didapatkan rata – rata keberhasilan 66.67% dalam melakukan peralihan meskipun salah satu server mengalami kegagalan, lalu penyebaran request dibagi secara merata 50:50 oleh load balancing. Sementara itu, performansi server menunjukkan bahwa delay pada layanan dikategorikan dalam "Sangat Bagus" dan throughput dikategorikan dalam "Bagus".*

***Kata kunci:*** *data storage, high availability, data replication, load balancing, failover*

**Abstract**

*The need of information that is always up-to-date and accessible anywhere and anytime is increasing now. It causes the need of cloud services in the form of high availability data storage. With high availability, it can reduce the occurrence of server failure. One way to improve the quality of the service is to implement multi server data storage. The implementation needs some techniques such as data replication, load balancing and failover. Data replication uses GlusterFS, load balancing uses HAProxy and failover uses Heartbeat. The result of this research found that the average data successfully uploaded 84.5 % and downloaded 100% from data storage. The failover method obtained 66.67% an average success rate in making the switch even one of the server failed, then the spread of the request is split evenly 50:50 by load balancing. Meanwhile, server performance shows that service delays are categorized in "Very Good" and throughput is categorized in "Good".*

***Keywords:*** *data storage, high availability, data replication, load balancing, failover*

Dokumen diterima pada 30 Oktober, 2018
Dipublikasikan pada 16 November, 2018

## 1. Pendahuluan

Dengan semakin berkembangnya teknologi, tidak bisa dipungkiri lagi bahwa kebutuhan untuk mendapatkan informasi atau data saat ini cukup mudah. Berbagai informasi atau data seperti gambar, berita artikel, hiburan dan lain - lain dapat diakses dan disimpan. Data yang terus-menerus disimpan akan mengalami pertambahan yang memerlukan ruang penyimpanan kapasitasnya semakin besar.

Semakin besar kapasitas penyimpanannya akan membuat pengolahan dan pengelolaan data menjadi tidaklah mudah. Untuk mengelola data yang begitu banyak, perlu membangun server berupa *data storage* dengan teknologi *cloud computing*. *Data storage* akan dibangun menggunakan virtualisasi.

Meskipun telah dibangun hal tersebut, tidak menjamin bahwa layanan yang diberikan pada *data storage* dapat berjalan lancar, seperti server tiba-tiba mengalami gangguan atau tidak tersedia. Saat akses data tidak mampu memberikan respon yang baik, dapat menyebabkan layanan pada *client* ikut terganggu dan *client* harus menunggu sampai server dapat berjalan normal kembali. Gangguan tersebut perlu diminimalkan agar meningkatkan kualitas layanan dari server tersebut.

Adapun cara untuk meningkatkan kualitas layanan server, yaitu membuat multi server *data storage* dengan menerapkan *data replication, load balancing,* dan *failover* sehingga *data storage* memiliki tingkat ketersediaan yang tinggi (*high availability).* Setiap penerapan memiliki perannya masing – masing dalam membantu server meningkatkan kualitas layanannya.

Penerapan *data replication* berfungsi untuk melakukan penyalinan data dan pendistribusian data dari satu *storage* ke *storage* lain, serta melakukan sinkronisasi antar *storage* sehingga konsistensi dari data tersebut dapat terjamin. Penerapan *load balancing berfungsi* untuk membantu mendistribusikan *request* antar *storage* secara merata agar proses *transfer* data berjalan dengan stabil. Selain itu, penerapan *failover* berfungsi untuk mengalihkan layanan server ke server lain (komponen cadangan, elemen, atau operasi). Jika server pertama sedang tidak tersedia, maka server sekunder langsung menggantikan hingga server pertama atau utama hidup.

Dengan adanya penerapan tersebut, membantu mengurangi beban server dan ketersediaan layanan & data juga terjamin sehingga terbentuklah sebuah layanan *cloud computing data storage* yang memiliki tingkat performasi yang baik, handal, dan terjaga ketersediaannya.

## 2. Tinjauan Pustaka

### 2.1 Penelitian Terdahulu

Dikarenakan penelitian ini merupakan pengembangan dari beberapa penelitian sebelumnya, maka sebagai bahan perbandingan berikut ini dicantumkan beberapa penelitian terdahulu yang relevan.

Penelitian yang dilakukan Nanda [1] berisi tentang bagaimana mengatasi kegagalan yang terjadi karena *single point of failure* dengan menggunakan teknologi *Automatic File Replication Cluster High-Availability Storage* agar kegagalan sistem dapat diminimalkan sehingga server tetap dapat memberikan layanan kepada *client*.

Penelitian yang dilakukan oleh Randy [2] menyajikan perancangan sebuah server sebagai *data storage*, dengan mendistribusikan *file* ke dalam setiap server yang tergabung dalam satu jaringan *cluster* menggunakan GlusterFS. Sementara itu, penelitian yang dilakukan oleh Irfani [3] berisi tentang perancangan sistem *failover virtual computer cluster* sebagai salah satu solusi untuk mengatasi kegagalan fungsi server.

Penelitian yang dilakukan oleh Prasetyo dkk. [4] membahas tentang bagaimana menerapkan penyalinan data ke *storage* cadangan agar server menjadi *high availability* menggunakan DRBD. Selanjutnya penelitian Habibah dkk. [5] membahas tentang bagaimana membangun *datacenter* yang memiliki tingkat ketersediaan tinggi pada layanan *cloud* menggunakan *Heartbeat* dan *Rsync.*

## 2.2 *Cloud Computing*

*Cloud computing* adalah sebuah model yang memungkinkan penggunaan sumber daya (server, jaringan, *storage*, aplikasi, layanan, dll) secara bersama-sama yang dapat dikonfigurasi dengan mudah dan meminimalisir interaksi dengan penyedia layanan (*provider*) [6].

## 2.3 *Datacenter*

*Datacenter* adalah sebuah fasilitas khusus yang di dalamnya termasuk menyimpan, mengelola, dan mendukung sumberdaya komputasi yang diperuntukkan bagi satu atau lebih organisasi. Sebuah infrastruktur *datacenter* yang lengkap meliputi struktur bangunan, *power backup*, sistem pendingin, ruangan dengan peruntukan khusus (misal jaringan telekomunikasi), lemari peralatan, perkabelan, perangkat jaringan, sistem penyimpanan, komputer *mainframe*, aplikasi perangkat lunak, sistem keamanan fisik, pusat *monitoring*, dan beberapa sistem pendukung lainnya. Seluruh sumberdaya dikelola secara langsung maupun secara *remote* oleh personel khusus [7].

## 2.4 *High Availability*

Bookman [8] mengatakan bahwa *high availability* pada dasarnya menempatkan satu atau lebih server cadangan dalam modus siaga, yang bisa *online* dalam beberapa saat hanya setelah mereka menemukan kegagalan pada sistem utama.

## 2.5 *Load Balancing*

*Load balancing* adalah suatu proses dan teknologi yang dapat mendistribusikan trafik ke beberapa server dengan menggunakan komputer atau perangkat jaringan. Proses ini mampu mengurangi beban kerja setiap server, serta memungkinkan server untuk menggunakan *bandwith* yang tersedia secara lebih efektif [9].

## 2.6 *Heartbeat*

*Heartbeat* merupakan teknologi multi *clustering node* yang mendukung *failover*, *failback*, dan *migration (load balancing)* dari sumberdaya *cluster* yang dikelola secara mandiri [10]. Berikut ini merupakan peran dari *Heartbeat*:

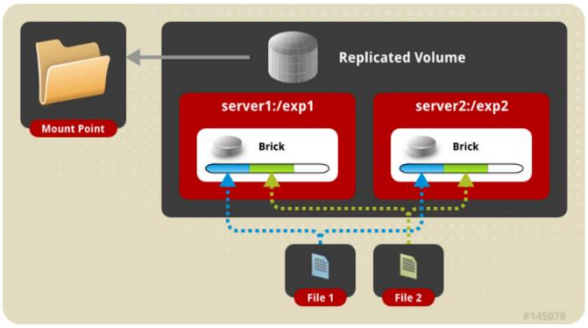
- Memonitor server *online* (bekerja atau tidak), bisa menggunakan *ethernet*, dan juga menggunakan *serial port*.
- Melakukan *failover* dimana *master server* akan dialihkan ke *slave server* jika terjadi kegagalan pada *master server*.

## 2.7 *HAProxy*

*HAProxy* adalah produk *open source* yang menyediakan solusi untuk menciptakan sistem *load balancing* dan *failover* dari aplikasi yang berbasis TCP dan HTTP. Perangkat lunak ini sangat cocok digunakan untuk *website* yang trafik hariannya tinggi sementara itu diperlukan keteguhan dan kekuatan dari pemrosesan pada *layer* 7. *HAProxy* dipasang pada server *front-end*. *Front-end server* umumnya adalah server yang memiliki IP statis teregistrasi dengan DNS [11].

## 2.8 *GlusterFS*

*GlusterFS* adalah file sistem Linux yang terdistribusi dan dirancang untuk mendapatkan skalabilitas yang tinggi dan mampu melayani tugas-tugas intensif data seperti penyimpanan *cloud* dan *streaming* media [12]

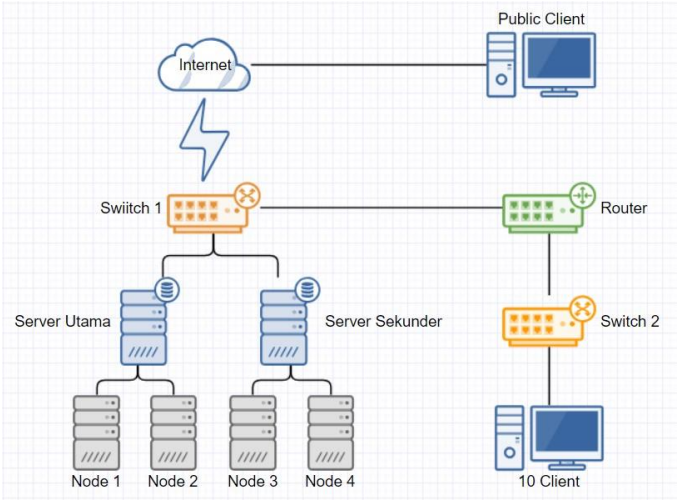



**Gambar 1. Mode *Replicated Volume* [13]**

Skema *GlusterFS* pada gambar 1 adalah *replicated*. Skema ini cocok difokuskan untuk *high availability* dan *high reliability* karena *replicated* menduplikasi *file* yang disimpan oleh *client* dan menyebarkannya ke semua *node* server pada jaringan *cluster* [13].

## 3. Hasil dan Pembahasan

### 3.1 Perancangan

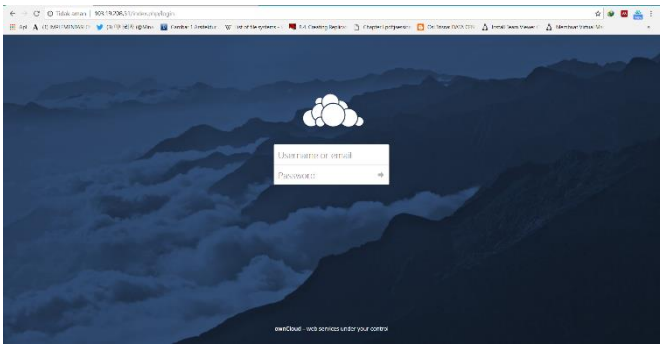



**Gambar 2. Topologi Jaringan**

Pada gambar 2 terdapat dua buah *physical server* yaitu server utama dan server sekunder. Setiap server dibangun sebanyak dua *node* dengan bantuan media virtualisasi KVM, agar *data storage* pada layanan *cloud* terbentuk. Selain itu, *tools Heartbeat* dan *load balancing* juga di pasangkan di tiap *physical server* untuk menjaga keterserdiaan layanannya. *Heartbeat* yang membantu proses peralihan layanan server utama ke server sekunder jika terjadi kegagalan atau *down server*. Sedangkan *load balancing yang* membantu membagi *request* yang diterima server kedalam masing – masing *node*. Semua *node* di *install tools GlusterFS* untuk replikasi data sehingga semuanya memiliki data yang sama. Data yang di replikasi adalah data *owncloud.*

### 3.2 Antar muka *Datacenter*

*Owncloud* digunakan sebagai *interface* antara *client* dengan server. *Client* dapat mengakses *owncloud* menggunakan *web browser* dengan alamat *url* http://103.19.208.51/ seperti pada gambar 3.



**Gambar 3. Tampilan *Login Owncloud***

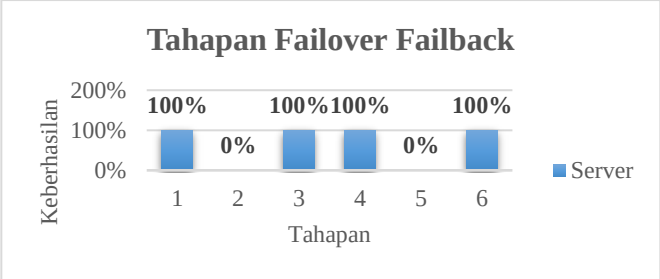
*Client* dapat menambah *folder* baru, melakukan *upload* dan *download file*, mengubah nama *file* atau *folder*, menghapus *file* atau *folder*, serta membagikan *file* atau *folder*.

### 3.3 Pengujian dan Analisa

Untuk mengetahui kelayakan penerapan teknik – teknik yang diterapkan pada server, dilakukan pengujian yang menguji keberhasilan *failover* dan *failback*.

**Tabel 1. Skenario *Failback* dan *Failover***

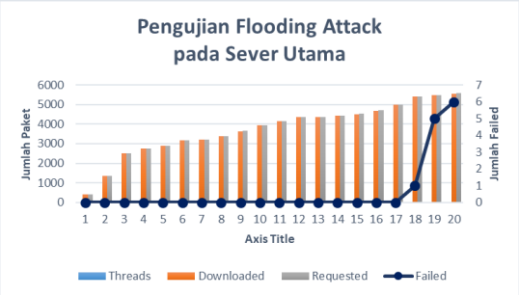
| No | Tahapan |
|---|---|
| 1 | Server Utama dimatikan.<br>(Apakah Server Sekunder mengambil alih?) |
| 2 | Server Utama dimatikan.<br>(Apakah proses *transfer* data berlanjut saat server dialihkan?) |
| 3 | Server Utama dimatikan.<br>(Apakah layanan pada Server Sekunder berjalan?) |
| 4 | Server Utama dihidupkan kembali.<br>(Apakah layanan berpindah kembali?) |
| 5 | Server Utama dihidupkan kembali.<br>(Apakah proses *transfer* data berlanjut saat server dialihkan?) |
| 6 | Server Utama dihidupkan kembali.<br>(Apakah layanan pada Server Sekunder berhenti?) |



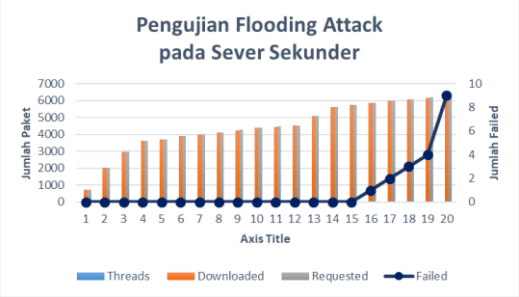

**Gambar 4. Grafik Keberhasilan *Failover* & *Failback***

Skenario *failover* dan *failback* (1,3,4,6) memiliki keberhasilan 100%, tetapi keberhasilan skenario proses *transfer data* (2,5) yaitu 0% ketika terjadi perpindahan layanan server. Hal yang menyebabkan skenario 2 dan 5 gagal adalah perpindahan layanan yang terjadi antar *physical server* karena lokasi *storage* berada di dua *disk physical* yang berbeda sehingga aktivitas terkait *transfer data* akan terhenti sendiri saat peralihan.

Pembanjiran *request* juga dilakukan untuk mengetahui kelayakan server dalam merespon. Pengujian ini dilakukan dengan *flooding attack* menggunakan LOIC yakni mengirim paket protokol *http* dengan jumlah *threads* secara bertahap dari 1 – 20 *threads*. *Flooding attack* melakukan permintaan *request* terus – menerus sehingga server tidak dapat lagi menangani *request* yang diminta.




**Gambar 5. Pembanjiran *Request* Server Utama**



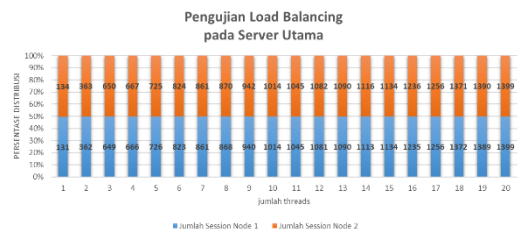

**Gambar 6. Pembanjiran *Request* Server Sekunder**

Selama pengujian dilakukan, terlihat bahwa semakin banyak jumlah *threads* maka semakin banyak jumlah paket yang diminta. Namun hal itu bergantung kembali pada kemampuan server dalam menangani seberapa banyak *request*.

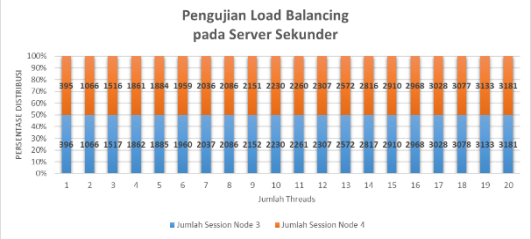
Pada gambar 5 dan 6 memperlihatkan jumlah paket yang *failed* semakin lama semakin meningkat mulai dari titik 16 -20 *threads*. Hal ini disebabkan pembanjiran *request* mencoba menghabiskan *resource* dari server sekunder sehingga berhenti untuk menerima *request*.

Ketika paket *failed* telah muncul, pembanjiran *request* pada server dikatakan berhasil karena server sudah mulai mengalami *down*. Server utama berhasil me-*reply* paket yang di-*request* sebanyak 5556 paket, sedangkan server sekunder sebanyak 6244 paket.

Kemudian kelayakan server dalam mendistribusikan paket secara merata juga di uji, paket yang masuk melalui pembanjiran *request* ternyata dapat dibagi secara seimbang ke masing – masing *node* dalam server seperti pada gambar 7 dan 8.

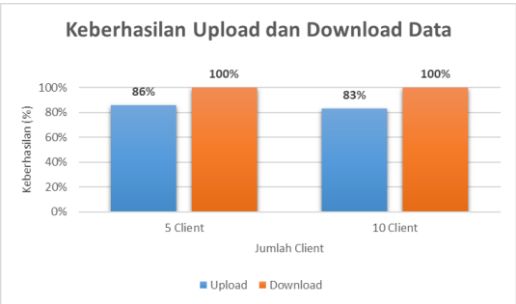



**Gambar 7. Grafik *Load Balancing* Server Utama**




**Gambar 8. Grafik *Load Balancing* Server Sekunder**

Penerapan server yang dibangun bersifat *data storage*, tentunya ada kegiatan yang di lakukan oleh *client* seperti *upload* dan *download file*. Kelayakan server dalam menangani kegiatan *upload* dan *download* juga di uji.

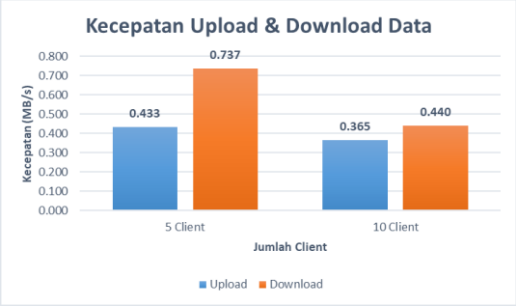



**Gambar 9. Grafik Keberhasilan *Upload* dan *Download***

Berdasarkan pengujian yang dilakukan, semakin banyak jumlah *client* yang meng-*upload* secara bersamaan, maka keberhasilan persentase *upload* ikut berkurang seperti pada gambar 9.

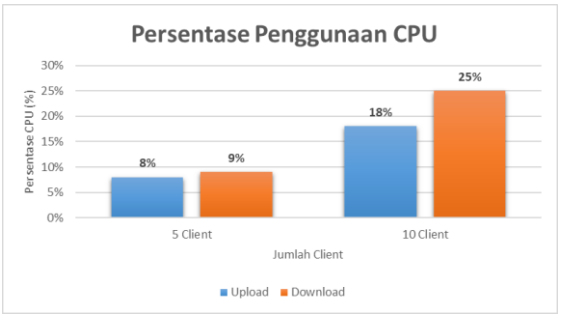
Ukuran *file* yang di *upload* juga ikut mempengaruhi persentase keberhasilan *upload*. Jika *file* yang di-*upload* secara bersamaan lebih besar dari *speed upload*-nya, maka proses *upload* memakan waktu lama hingga *timeout* yang berarti gagal. Sedangkan persentase *download* memiliki keberhasilan 100%.

Kecepatan *transfer* data saat proses *upload* dan *download* juga mengalami penurunan kecepatan karena banyaknya *client* yang melakukan kegiatan tersebut secara bersamaan.

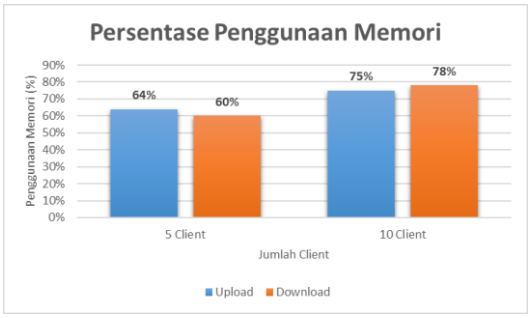



**Gambar 10. Grafik Kecepatan *Upload* dan *Download***

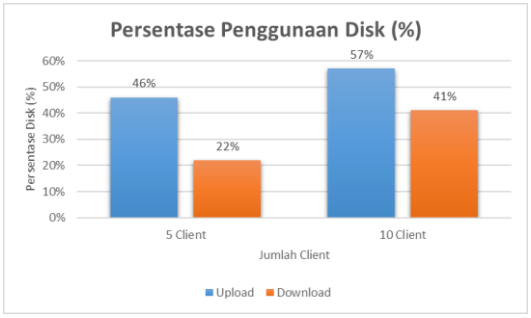
Selanjutnya perfomansi server seperti pengunaan CPU, memori dan *disk* juga diukur saat memberikan layanan kepada *client*. Berdasarkan hasil percobaan yang dilakukan, performansi server semakin meningkat ketika jumlah *client* yang mengakses bertambah. Untuk melihat seberapa besar penggunaan *resource* server dapat dilihat pada gambar 11 – 13.




**Gambar 11. Grafik Penggunaan CPU**




**Gambar 12. Grafik Penggunaan Memori**

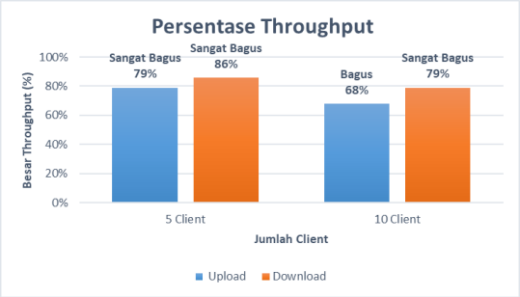



**Gambar 13. Grafik Penggunaan Disk**

Selain itu, *Quality of Service* pada server juga diukur untuk mengetahui kualitas server masuk dalam kategori yang baik atau belum. Parameter QoS yang digunakan adalah *throughput* dan *delay* dan diukur saat terjadi proses *upload* dan *download* bersamaan.

**Tabel 2. Kategori *Throughput* [14]**

| Kategori *Throughput* | *Throughput* (%) | Indeks |
|---|---|---|
| Sangat Bagus | 100 | 4 |
| Bagus | 75 | 3 |
| Sedang | 50 | 2 |
| Buruk | < 25 | 1 |




**Gambar 14. Grafik Persentase *Throughput***

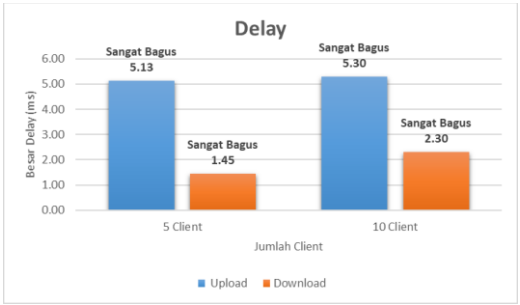
Kualitas *throughput* pada *upload dan download* cenderung mengalami penurunan pada gambar 14 dikarenakan jumlah pengguna yang semakin banyak mengakses secara bersamaaan.

Selain itu, penurunan juga dipengaruhi oleh lamanya waktu *upload* dan *download* yang bergantung pada kecepatan jaringan internet dan seberapa besar ukuran *file* yang di unggah atau di unduh secara bersamaan.

Walaupun mengalami penurunan, kategori rata – rata *throughput* untuk pengujian ini tergolong “bagus” sesuai dengan kategori standar *throughput* pada tabel 2.

**Tabel 3. Kategori *Delay* [15]**

| Kategori *Delay* | Besar *Delay* (ms) | Indeks |
|---|---|---|
| Sangat Bagus | < 150 | 4 |
| Bagus | 150 s/d 300 | 3 |
| Sedang | 300 s/d 450 | 2 |
| Buruk | > 450 | 1 |




**Gambar 15. Grafik Persentase *Delay***

Sedangkan untuk *delay* pada *upload dan download*, keduanya cenderung mengalami kenaikan mulai dari 5 sampai 10 *client*. Peningkatan *delay* disebabkan karena bertambahnya jumlah *client* yang mengakses secara bersamaan, walaupun begitu kategori *delay* untuk percobaan ini tergolong “sangat bagus” dan mencukupi standar *delay* dari TIPHON sesuai pada tabel 3.

## 4. Kesimpulan

Dari hasil pengujian dan analisis yang telah dilakukan, maka dapat diambil beberapa kesimpulan sebagai berikut:

- Pemanfaatan *GlusterFS* dengan skema *mirroring* atau *replicate* dapat menjamin *data never loss*.
- Pemanfaatan teknologi *cloud* dan *Heartbeat* dalam menjaga ketersediaan informasi pada *data storage* memiliki tingkat keberhasilan *failover* dan *failback* sebesar 66.67%. Hal ini disebabkan perpindahan layanan terjadi pada dua server fisik berbeda yang masing-masing *physical disk*-nya berbeda.
- Pembagian *request* dibagi seimbang 50:50 dengan adanya *load balancer*.
- Seiring bertambahnya jumlah *client* saat mengakses bersama – sama, besaran *file* yang di unduh atau diunggah, dan kecepatan internet dapat mempengaruhi keberhasilan *transfer data*, kecepatan *transfer data*, performansi server dan performansi jaringan.

## Daftar Pustaka

[1] P. Nanda, "Automatic File Replication Cluster High-Availability Storage Dengan Menggunakan GlusterFS," 2014. .

[2] Randy, "Analisis Kinerja Layanan Penyimpanan File Terdistribusi Cluster High-Availability Storage dengan Menggunakan GlusterFS," 2014. .

[3] Irfani, "Implementasi High Availability Server Dengan Teknik Failover Virtual Computer Cluster," 2015.

[4] S. Prasetyo, M. A. Fadhly, dan S. Purwantoro, "Implementasi Data Replication Pada Cloud Computing Menggunakan Distributed Replicated Block Device (DRBD)," Pekanbaru, 2016.

[5] U. Habibah, Y. E. Putra, dan H. Rachmawati, "High Availability Datacenter Pada Cloud Computing Menggunakan Heartbeat," Pekanbaru, 2016.

[6] P. Mell dan T. Grance, "The NIST Definition of Cloud Computing," *NIST Spec. Publ. 800-145*, hal. 2–3, 2011.

[7] G. Santana, *Data Center Virtualization Fundamentals*. Indianapolis: Cisco Press, 2014.

[8] C. Bookman, *Linux Clustering : Building and Maintaining Linux Clusters*. Indianapolis: New Riders, 2003.

[9] T. Bourke, *Server Load Balancing*, 1 ed. Sebastopol: O'Reilly Media, Inc., 2001.

[10] T. Roth dan T. Schraitle, *SUSE Linux Enterprise High Availability Extension*. Provo: Novell, Inc, 2012.

[11] HAProxy Community Edition, "The Reliable, High Performance TCP/HTTP Load Balancer," 2017. [Daring]. Tersedia pada: http://www.haproxy.org/. [Diakses: 11-Agu-2018].

[12] Gluster Docs, "GlusterFS Documentation," 2018. [Daring]. Tersedia pada: https://docs.gluster.org/en/latest/. [Diakses: 11-Agu-2018].

[13] D. Muntimadugu, B. Mohanraj, dan A. S. Sriram, "Red Hat Storage 2.1 : Administration Guide," *Red Hat, Inc.*, 2014. [Daring]. Tersedia pada: https://access.redhat.com/documentation/en-US/Red_Hat_Storage/2.1/html/Administration_Guide/. [Diakses: 12-Jun-2017].

[14] W. Sugeng, J. E. K. O. Istiyanto, K. Mustofa, dan A. Ashari, "The Impact of QoS Changes towards Network Performance," *Int. J. Comput. Networks Commun. Secur.*, vol. 3, no. 2, hal. 48–53, 2015.

[15] TIPHON, "Telecommunications and Internet Protocol Harmonization Over Networks (TIPHON); General aspects of Quality of Service (QoS)," 1999. .