RASPBERRY PI HIGH AVAILABILITY CLUSTER SERVER DENGAN METODE *SYNCHRONOUS REPLICATION*, *REDUNDANCY* DAN *FAILOVER*

Imam Kurniawan[1], Edhy Sutanta[2], Joko Triyono[3]
Program Studi Teknik Informatika, Fakultas Teknologi Industri
Institut Sains & Teknologi AKPRIND Yogyakarta
Email : [1]ximam86@skynetter.com, [2] edhy_sst@yahoo.com, [3]jack@akprind.ac.id

ABSTRACT

Information needs now require services that have high availability technology to handle server downtime disruption, so as to maintain the stability of the server in serving the high demand. Server downtime can occur due to damage to hardware or software components that cause crashes, or upgrade and maintenance on the server.

The technology used to handle server downtime is high availability cluster using synchronous replication, redundancy and failover methods. The devices that become cluster servers are two Raspberries Pi, namely as a master node in active status and slave node in passive status. The master node serves create, read, update, and delete data requests, then replicated to the slave node using synchronous replication method, so the data processing on each node runs simultaneously. When the master node fails, a failover occurs, the slave node becomes active and serves the request for data processing.

Implementation of synchronous replication and redundancy methods successfully tested by replicating files and databases between each cluster node. In testing the failover method, when the master node is down, the slave node manages to take over automatically and becomes the master node. The conclusion obtained from this research is the implementation of high availability cluster can reduce the time of server downtime, thus minimizing the risk of data loss or damage.

Keywords*: High Availability, Cluster, Server, Synchronous Replication, Redundancy, Failover, Raspberry Pi.*

INTISARI

Kebutuhan informasi sekarang ini membutuhkan layanan yang memiliki teknologi *high availability* untuk menangani gangguan *downtime server*, sehingga dapat menjaga kestabilan *server* dalam melayani permintaan yang tinggi. *Downtime server* dapat terjadi karena adanya kerusakan pada komponen *hardware* maupun *software* yang menyebabkan *crash*, atau perlu dilakukan *upgrade* dan *maintenance* pada *server*.

Teknologi yang digunakan untuk menangani *downtime server* adalah *high availability cluster* dengan metode *synchronous replication, redundancy,* dan *failover*. Perangkat yang menjadi *cluster server* adalah dua buah *Raspberry Pi*, yaitu sebagai *master node* dalam status aktif dan *slave node* dalam status pasif. *Master node* melayani permintan *create, read, update,* dan *delete* data, kemudian direplikasi ke *slave node* dengan metode *synchronous replication*, sehingga proses pengolahan data pada setiap *node* berjalan secara bersamaan. Ketika *master node* mengalami kegagalan, maka terjadi *failover*, yaitu *slave node* menjadi aktif dan melayani permintaan untuk pengolahan data.

Implementasi metode *synchronous replication* dan *redundancy* berhasil diujikan dengan melakukan replikasi *file* dan *database* antara setiap *cluster node*. Pada pengujian metode *failover*, ketika *master node* mengalami *down*, *slave node* berhasil melakukan *take over* secara otomatis dan menjadi *master node*. Kesimpulan yang diperoleh dari penelitian ini adalah implementasi *high availability cluster* dapat mengurangi waktu *downtime server*, sehingga meminimalkan risiko kehilangan atau kerusakan data.

Kata Kunci: *High Availability, Cluster, Server, Synchronous Replication, Redundancy, Failover, Raspberry Pi.*

PENDAHULUAN

Menghadapi banyaknya permintaan kebutuhan informasi pada lingkungan global seperti sekarang ini, diperlukan analisa terhadap kebutuhan dan *high availability* (ketersediaan tinggi) yang dimiliki untuk menangani banyaknya permintaan tersebut. Perusahaan tidak akan mampu kehilangan pelanggannya ketika layanan yang ditawarkan menjadi terbatas, terutama karena *downtime server* yang hampir pasti akan terjadi dan mengakibatkan kerugian dengan biaya yang tinggi. Faktanya *server* tidak berjalan selamanya. Permasalahan dapat terjadi karena adanya kerusakan pada komponen-komponen *hardware* (perangkat keras) di komputer *server*, *software* (perangkat lunak) yang menyebabkan *crash* (kerusakan), atau pun sistem operasi pada komputer *server* yang perlu *shutdown* (dimatikan) untuk melakukan *upgrade* (peningkatan) dan proses *maintenance* (pemeliharaan). Oleh karena itu, dibutuhkan teknologi yang bertujuan untuk menjaga kestabilan *server* dalam melayani permintaan yang tinggi dan menjaga kelancaran lalu lintas pertukaran data, khususnya yang melalui *internet*. Teknologi ini dikenal dengan *high availability cluster*, yang juga sering disebut sebagai *failover cluster*.

Pendekatan umum untuk meningkatkan ketersediaan sistem adalah menggunakan metode *synchronous replication*, *redundancy* (redundansi), dan *failover*. Metode *synchronous replication* adalah proses penyimpanan data pada lebih dari satu *server* di saat yang bersamaan, yang umumnya terdiri dari *master* dan *slave node*. Proses pengolahan data yang meliputi perubahan, penambahan, atau pengurangan data pada *master node*, juga dilakukan secara langsung di *slave node*, meski proses pengolahan data tersebut belum pada tahap *complete* (selesai), sehingga elemen *cluster* akan bekerja dengan memiliki *redundancy node*, yang kemudian digunakan untuk menyediakan layanan saat salah satu elemen *cluster* mengalami kegagalan.

Ukuran yang paling umum dari konsep *high availability cluster* adalah menggunakan dua *node (cluster server)*, yang merupakan syarat minimum untuk melakukan *redundancy*. Implementasi *cluster* dengan metode *failover* juga akan menggunakan *redundancy* terhadap komponen *cluster* untuk menghilangkan titik tunggal kegagalan (*Single Point of Failure*). Penerapan *failover* untuk bekerja secara otomatis (*auto failover*) pada infrastruktur *high availability cluster* akan mengurangi waktu *downtime server*, karena jika salah satu *server* mengalami *down* maka *server* yang lain akan melakukan *take over* (mengambil alih tugas) secara otomatis, sehingga *user* (pengguna) dan aplikasi yang sedang berjalan tidak akan terlalu merasakan jika terjadi perpindahan terhadap *server* yang sedang aktif digunakan. Total *redundancy* dari beberapa *server* akan dapat membantu untuk mencapai *uptime* sistem yang lebih besar.

Perkembangan teknologi *hardware* telah melahirkan beberapa perangkat yang cukup memadai untuk digunakan sebagai elemen *clustering*, bahkan dapat menekan banyaknya biaya yang dibutuhkan untuk menerapkan teknologi *high availability cluster*. Perangkat yang dapat digunakan adalah *Raspberry Pi*, yaitu sebuah perangkat elektronik sederhana dengan biaya yang terjangkau, namun memberikan keleluasan dalam pengembangan dan kostumisasi sebuah perancangan *cluster*. *Raspberry Pi* dapat digunakan sebagai komputer *server* dalam elemen *clustering,* dengan memanfaatkan semua kelebihan yang ada pada perangkat komputer, namun berukuran kecil seperti kartu kredit sehingga banyak menghemat konsumsi daya listrik dan kebutuhan tempat, serta memberikan banyak fleksibilitas.

TINJAUAN PUSTAKA

Penelitian ini tidak terlepas dari hasil penelitian-penelitian terdahulu yang pernah dilakukan sebagai bahan perbandingan dan kajian. Beberapa penelitian terdahulu yang mendasari, dapat dijadikan sebagai acuan dan pendukung dalam melakukan penelitian dan sebagai perbandingan adalah sebagai berikut:

1. Implementasi dan Analisa Sistem *Failover Virtual Computer Cluster* (Febriani, T.R., 2011).
   Penelitian ini membahas tentang sistem *failover virtual computer cluster*, dengan menganalisa perbandingan performa antara sistem *cluster* dan sistem yang tidak di-*cluster*. Pada penelitian tersebut, sistem *cluster* diimplementasikan menggunakan sistem operasi *Windows Server 2008 R2 x64* dan *platform Hyper-V* untuk simulasi. Analisa performa dilakukan pada lima parameter *benchmark*, yaitu *processor arithmetic*, *.NET arithmetic*, *memory bandwidth*, *memory latency, cache* dan *memory*. Hasil *benchmark* performa sistem *failover virtual computer cluster* untuk kelima parameter ini tidak lebih baik dibandingkan dengan sistem yang tidak di-*cluster*, dimana terjadi penurunan rata-rata keseluruhan sebesar 41% untuk skenario pertama dan 54% untuk skenario kedua. Pada segi *availability*, sistem *failover virtual computer cluster* lebih unggul dibandingkan sistem yang tidak di-*cluster*.
2. Implementasi dan Analisa Redundansi dan *High Availability* dalam *Server* untuk *Diskless Thin Client* Berbasis *Storage Area Network* (Hidayat, F., 2012).
   Penelitian ini membahas tentang redundansi dan *high availability* pada *server* dengan metode *Storage Area Network (SAN)* menggunakan *DRBD* dan *Heartbeat* menggunakan sistem operasi *Linux Ubuntu Server* 10.04. Hasil pengujian menunjukkan bahwa redundansi atau duplikasi data memiliki kecepatan rata-rata 40 *MB*/s dan memiliki tingkat ketersediaan yang tinggi rata-rata mencapai 99,99%.
3. Pengembangan Sistem Pemantau dan Pengendali Kendaraan Menggunakan *Raspberry Pi* dan *Firebase* (Susanti, E., dan Joko Triyono, 2016).
   Penelitian ini mengenai sistem pemantau dan pengendali kendaraan bermotor jarak jauh menggunakan teknologi web *realtime* yang telah berhasil dikembangkan dengan menggunakan teknologi *AngularFire* (*AngularJS* dan *Firebase*). Selanjutnya sistem dioperasikan untuk melakukan pemantauan terhadap kendaraan yang telah dipasang dengan perangkat *IoT* yang terdiri dari sebuah *Raspberry Pi* 3 Model B, modul *GPS Ublox Neo-6M*, *Relay DC 2 Channel*, *modem GSM* dan di dalamnya telah dilengkapi dengan program koneksi ke *server*. Pada penelitian tersebut *Raspberry Pi* digunakan sebagai alat pelacak yang terhuubung dengan internet, kemudian datanya akan dikirim ke *server.*

Konsep *Cluster*

Definisi sederhana dari *cluster* di dalam ilmu jaringan komputer adalah sebuah komputer (*cluster computer*) yang terhubung dengan komputer lainnya dan bekerja secara bersama-sama, sehingga hal tersebut dapat dilihat sebagai sebuah sistem yang tunggal (Kopper, K., 2005). Proses untuk menghubungkan beberapa komputer agar dapat bekerja secara bersama-sama seperti itu dinamakan dengan *clustering.* Komputer yang berfungsi sebagai *server* dan tergabung di dalam jaringan *clustering* disebut sebagai *cluster node*. Setiap komponen pada sistem *cluster node* tersebut biasanya saling terhubung melalui jaringan lokal (*LAN*/*Local Area Network*) atau jaringan secara luas (*WAN*/*Wide Area Network*).

Terminologi *High Availability Cluster*

Komputer *cluster* adalah sekelompok komputer yang bergabung untuk bekerja sama guna menyediakan ketersediaan beberapa layanan yang tinggi. Layanan biasanya dibangun dengan sejumlah aplikasi yang beroperasi di *layer* aplikasi. Biasanya aplikasi mengalami *error* atau *bug*, dan berhenti merespon atau *crash*. Situasi seperti ini akan menyebabkan *downtime* layanan dan kemungkinan juga kerugian finansial, jadi perlu memberikan redundansi pada *layer* aplikasi. Inilah alasan mengapa perlu menerapkan solusi komputer *cluster* ke dalam desain sistem *high availability* (Resman, M., 2015).

Sebuah sistem dianggap *highly available* ketika terjadi kegagalan data tidak hilang dan sistem dapat pulih dalam jumlah waktu yang wajar. Ada kemungkinan terjadinya kegagalan sistem dan aplikasi dapat terus digunakan, misalnya aplikasi *client-server* yang dirancang sehingga entri data dapat berlanjut pada *client* jika terjadi kegagalan sistem pada *server* (Critchley, T., 2015).

Tujuan utama dari implementasi *high availability cluster* adalah untuk meningkatkan ketersediaan sistem terhadap permintaan layanan yang tinggi, serta harus tetap dan terus tersedia, tanpa adanya gangguan atau pun keterbatasan. Penerapan *high availability cluster* dapat mengatasi masalah kegagalan (*failure*) pada sistem, sehingga layanan dapat terus berjalan (Resman, M., 2015).

Failover

Setiap situasi kegagalan sistem yang terdeteksi dalam infrastruktur *high availability cluster* seperti *downtime* pada *server*, akan mengakibatkan terjadinya suatu peralihan layanan yang disediakan untuk mengarah kepada *cluster node* yang lain, proses ini yang disebut sebagai *failover*. Sebuah *failover* terjadi ketika *resource* (sumber daya) layanan bergerak dari satu *node* ke *node* yang lainnya saat terjadi kegagalan pada suatu *node*. Dengan konfigurasi *high availability cluster* dan *failover* yang tepat, maka akan menghilangkan titik tunggal kegagalan (*Single Point of Failure*). *Single Point of Failure* (*SPOF*) adalah istilah di mana bagian dari sebuah kegagalan sistem akan menyebabkan seluruh sistem berhenti bekerja (Kopper, K., 2005).

Penerapan *failover* untuk bekerja secara otomatis (*auto failover*) pada infrastruktur *high availability cluster* akan mengurangi waktu *downtime server*. Jika salah satu *server* mengalami *down*, maka *server* yang lain akan melakukan *take over* (mengambil alih tugas) secara otomatis, sehingga *user* dan aplikasi yang sedang berjalan tidak akan terlalu merasakan jika terjadi perpindahan terhadap *server* yang sedang aktif digunakan.

Failback

Secara garis besar, konsep *failover* menggambarkan apabila suatu *node down* maka *node* yang lainnya akan mengambil alih tugas *node* yang *down* tersebut. Dalam infrastruktur *high availability cluster* yang hanya terdiri dari dua *node*, hal ini dapat diistilahkan sebagai hubungan antara *master* (*primary*) dan *slave* (*secondary*). Jika pada *master node* mengalami kegagalan, maka *slave node* yang akan bertugas menggantikan tugas dari *master node* tersebut (Critchley, T., 2015).

Saat posisi *slave node* mengambil semua tugas dari *master node*, maka layanan yang disediakan akan beralih kepada *slave node*. Jika sistem pada *master node* sudah pulih, semua tugas dan layanan yang disediakan oleh *slave node* tersebut dapat berpindah kembali kepada *master node*, seperti posisinya semula sebelum terjadi proses *failover*. Proses untuk mengembalikan kembali sistem, komponen, atau layanan setelah terjadinya *failover* kepada *master node* inilah yang disebut sebagai *failback* (Critchley, T., 2015).

Penerapan *failback* untuk bekerja secara otomatis (*auto failback*) pada infrastruktur *high availability cluster* akan memberikan keuntungan yang sama seperti *auto failover*. Saat *server* yang mengalami *down* telah pulih kembali, maka akan melakukan *take over* (mengambil alih tugas) secara otomatis dari *server* yang menggantikan tugasnya pada saat *down*, sehingga *user* dan aplikasi yang sedang berjalan tidak akan terlalu merasakan jika terjadi perpindahan terhadap *server* yang sedang aktif digunakan.

Floating/Failover IP

Floating IP atau bisa disebut juga *failover IP*, adalah alamat *IP* yang dapat melekat dan berpindah-pindah antar *cluster node*. Gambaran sederhananya pada infrastruktur jaringan *high available cluster* adalah dari hubungan antara *master node* dan *slave node*, dimana *floating IP* layaknya *resource* di dalam jaringan *cluster*, dapat berpindah-pindah antar *node*, tergantung pada *node* mana yang menjadi *master*. Alamat *IP* inilah yang akan diakses secara langsung oleh *user*, jadi ketika terjadi proses *failover* atau *failback*, maka *user* tidak akan terlalu merasakan bahwa telah terjadi perpindahan terhadap *server* yang sedang aktif digunakan, karena *user* mengakses *alamat IP*, bukan

server secara langsung. Komponen ini menjadi salah satu hal yang penting di dalam keberhasilan implementasi infrastruktur *high availability cluster*.

Data Replication

Data replication (replikasi data) adalah sebuah teknologi yang serupa dengan *mirroring*, namun mampu bekerja dengan sukses pada kondisi jaringan yang berjarak jauh. Replikasi data memungkinkan pembaruan data penting untuk disalin dalam jangka *realtime*, melintasi jarak yang terlalu jauh untuk dilakukan *mirroring*. Bila digabungkan dengan teknik *clustering* yang disesuaikan untuk berfungsi secara jarak jauh, maka replikasi data memungkinkan sebuah organisasi untuk bertahan dalam *disaster* (bencana) yang melumpuhkan seluruh *data center* atau area geografis (Barker, R., 2002).

Sistem replikasi data pada jaringan *cluster* menggunakan metode *redundancy*, atau biasa disebut sebagai *redundancy data*. Pengertian sederhana dari *redundancy data* adalah menyediakan penyimpanan data secara terpisah, baik lokasinya maupun secara fisik (Charles, B., dkk, 2014). Ketika data di suatu *server* terjadi kerusakan, maka risiko kehilangan dan kerusakan data tersebut tidak akan terjadi, karena dengan *redundancy* telah menyediakan cadangan data di lokasi lainnya.

Dengan menerapkan teknik *data replication* dan *redundancy* pada jaringan *cluster*, maka dapat dipastikan untuk konsistensi, ketersediaan, dan aksesibilitas data di antara masing-masing *node* dapat terjaga dengan baik. Dalam kasus *failover*, *node* yang menggantikan tugas dari *node* yang *down* akan tetap memiliki data yang sama seperti *node* yang *down* tersebut.

Fungsi dan manfaat dari *data replication* dan *redundancy* adalah sebagai berikut (Charles, B., dkk, 2014):

1. Mengurangi potensi kerusakan dan kehilangan data. Dengan memanfaatkan beberapa *server*, jika terjadi kerusakan data pada salah satu *server*, maka masih ada salinan data pada *server* yang lain.
2. Kegagalan proses pengolahan data di *server* tidak akan mempengaruhi dalam memenuhi ketersediaan data dan aplikasi, karena masih ada data yang direplikasi di *server* lain sebagai penggantinya untuk menangani proses tersebut.

Raspberry Pi

Raspberry Pi adalah komputer seukuran kartu kredit yang dikembangkan oleh *Raspberry Pi Foundation* di Inggris dengan tujuan mempromosikan pengajaran ilmu komputer dasar di sekolah-sekolah. *Raspberry Pi* saat ini telah digunakan secara luas di semua bidang, termasuk dalam proses pembuatan super komputer dan operasi robotika yang canggih. *Raspberry Pi* terlahir dari keinginan beberapa mahasiswa di Laboratorium Komputer *University of Cambridge* untuk dapat melakukan pemrograman dengan komputer yang murah. Setelah menghabiskan beberapa tahun merancang prototipe, Eben Upton, yang sebelumnya lulusan dari Universitas dan sekarang bekerja sebagai desainer *chip Broadcom*, bergabung dengan rekan-rekan Universitas lamanya, yaitu Pete Lomas desainer *hardware* perusahaan *Norcott Technologies*, dan David Braben *progammer game Elite* dari *BBC Micro*, untuk membentuk *Raspberry Foundation*. Pada tahun 2012, *Raspberry Pi* kemudian diproduksi secara massal dengan nama model B, dan kemudian diluncurkan versi yang berkapasitas *Random Access Memory (RAM)* lebih rendah tapi lebih murah, yaitu model A. Jajaran produk *Raspberry Pi* terus dikembangkan dengan nama model A+ dan B+, dan kemudian dengan spesifikasi perangkat komputer yang lebih tinggi yaitu *Raspberry Pi* 2 dan 3 model B.

Raspbian

Raspbian adalah nama sistem operasi yang diberikan pada varian distribusi *Linux Debian* yang populer. *Debian* adalah salah satu distribusi *Linux* yang telah berjalan lama, dan berkonsentrasi pada kompatibilitas tinggi dan kinerja yang sangat baik bahkan pada perangkat keras sederhana, yang menjadikannya mitra hebat untuk *Raspberry Pi* (Upton, dkk, 2016). *Raspbian* mengambil *Debian* sebagai basisnya, atau distribusi

induknya, dan menambahkan beberapa perangkat lunak khusus agar menggunakan *Raspberry Pi* menjadi semudah mungkin.

Nginx Web Server

Web server adalah salah satu layanan yang akan berdiri di atas infrastruktur *cluster* untuk keperluan aplikasi *website*. Salah satu aplikasi *web server* yang populer saat ini adalah *Nginx*. *Nginx* adalah *web server* yang ringan digunakan dan memiliki beberapa modul untuk digunakan bekerja sebagai *reverse proxy* dan *mail server* tanpa menginstal dependensi lainnya. *Raspberry Pi* tidak memiliki terlalu banyak *RAM* untuk digunakan, dan *Nginx* dirancang khusus untuk menggunakan sedikit *resource* (sumber daya) namun memberikan konten pada kecepatan yang lebih tinggi. (Kula, P.J., 2014).

MariaDB Database Server

MariaDB merupakan aplikasi sistem manajemen basis data yang gratis untuk digunakan dengan fiturnya yang melimpah dan dapat memenuhi kebutuhan aplikasi yang berjalan pada jaringan *cluster*. *MariaDB* adalah hasil *forking* dari *MySQL*. *MariaDB* memiliki banyak fitur baru yang menarik, pengujian yang lebih baik, peningkatan kinerja, dan perbaikan *bug* yang tidak tersedia di *MySQL*. Beberapa optimalisasi untuk *MariaDB* datang dari perusahaan besar, seperti *Google*, *Facebook*, *Twitter*, dan sebagainya. Hal ini juga mencakup peningkatan ekstensi yang signifikan ke *database server*, seperti *dynamic columns*, *role-based access control*, dan dukungan terhadap *microsecond timestamp* (Mavro, P., 2014).

PHP

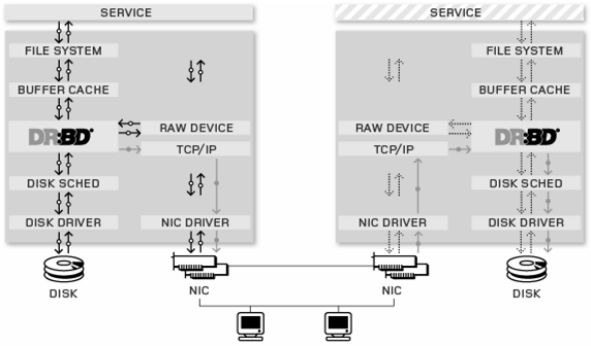
PHP (*PHP Hypertext Preprocessor*) adalah bahasa pemrograman *server-side scripting* yang menyatu dengan *HTML* untuk membuat halaman *website* yang dinamis. *PHP* adalah generator *markup HTML*, dimana pada halaman web tidak akan akan ditampilkan *source code* dari *PHP*, hanya ada *tag HTML* (MacIntyre, P., dkk, 2011).

PHP juga bisa ditulis dengan desain *Object Oriented Programming* (*OOP*), seperti *classes, properties, methods, inheritance, polymorphism,* dan *encapsulation*. Hal ini memudahkan dalam penggunaan pengulangan dari *source code* secara keseluruhan. Pendekatan *OOP* untuk pemrograman sudah lama ada di dunia teknologi, dan *PHP* telah mengadopsi dan memperluas integrasinya selama beberapa tahun terakhir (MacIntyre, P., dkk, 2011).

DRBD

DRBD (*Distributed Replicated Block Device*) merupakan aplikasi yang dirancang untuk membangun *high availability cluster*. Pada dasarnya *DRBD* ini mirip dengan teknologi *disk mirroring* dengan *RAID* 1. Pada *RAID* 1 *disk mirroring* dilakukan pada satu komputer yang sama, sedangkan *DRBD* mampu melakukan *disk mirroring* antar jaringan (Jones, M.T., 2010).

DRBD bekerja pada level *block device*. *Block device* adalah *device* (perangkat) yang menyimpan atau membawa data, seperti *hard disk drive*, *optical disk drive*, *Universal Serial Bus (USB) flash drive*, dan semacamnya. Ketika menggunakan aplikasi *DRBD*, *DRBD* akan menggunakan *block device* yang biasa disebut sebagai *DRBD device*, untuk media penyimpanan data yang akan direplikasi antar *cluster node*. Gambar 1 menampilkan proses dan alur kerja dari *DRBD* pada setiap konfigurasinya.



Gambar 1 Proses dan Alur Kerja *DRBD* (Hellman, B., dkk, 2012)

Corosync

Corosync adalah aplikasi yang bekerja di sistem *cluster* pada *layer* komunikasi. *Corosync* biasanya dikombinasikan dengan aplikasi *Pacemaker* sebagai *cluster resource manager*. Sebelum adanya kedua aplikasi ini, penerapan teknologi *cluster* awalnya menggunakan aplikasi *Heartbeat*. Seiring dengan perkembangan teknologi dan kebutuhan terhadap *infrastruktur high availability cluster*, maka penggunaan aplikasi pun berubah agar dapat mendukung fitur-fitur tambahan yang menerapkan ketersediaan tinggi dalam aplikasi (Vugt, S.V., 2014).

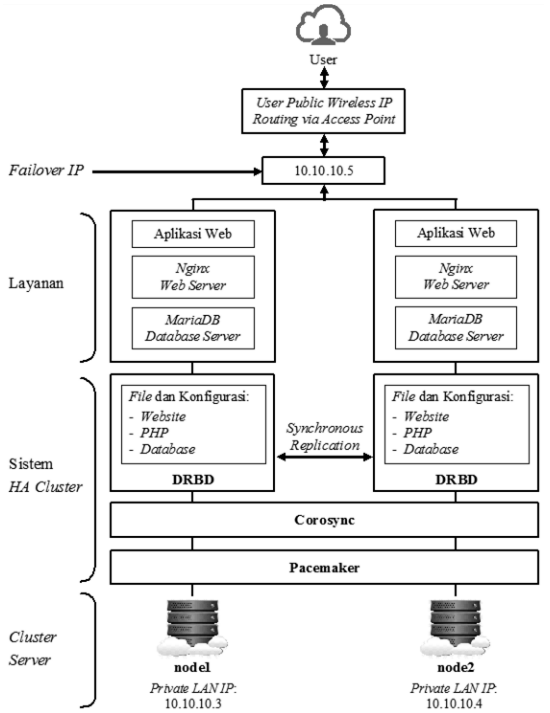
Pacemaker

Pacemaker adalah aplikasi yang berfungsi sebagai *cluster resource manager*. Aplikasi ini memiliki kemampuan untuk melakukan *monitoring* terhadap *node* atau *resource*, dengan memanfaatkan *layer* komunikasi yang disediakan oleh aplikasi *cluster* seperti *Corosync* (Vugt, S.V., 2014).

Dalam memenuhi kebutuhan implementasi *high availability cluster*, *Pacemaker* menyediakan berbagai fungsi dalam hal konfigurasi dan pengelolaan *cluster resource*. *Pacemaker* memiliki kemampuan untuk mendeteksi kegagalan pada suatu *node* atau adanya *resource* yang tidak berjalan sebagaimana mestinya, sehingga perlu untuk dilakukan *failover* (Gabriel, C., 2016).

PEMBAHASAN

Konfigurasi jaringan *cluster* dengan masing-masing alamat *IP* dan layanan yang disediakan ditunjukkan pada Gambar 2. Dua buah *cluster server* memiliki masing-masing alamat *IP* dengan satu buah *failover IP* di dalam jaringan *private LAN* untuk berinteraksi dengan pengguna aplikasi web di jaringan *public wireless,* yang telah diberikan pengaturan *routing* melalui *wireless router access point*.



Gambar 2 Konfigurasi Jaringan dan Layanan *High Availability Cluster*

Pengujian terhadap replikasi dan sinkronisasi data dilakukan dengan membuat beberapa *file* dalam ukuran tertentu, yang selanjutnya memantau aktifitas replikasi data pada *cluster server* di dalam *log file*. Parameter pengujian yang digunakan adalah ukuran *file* dan waktu yang digunakan untuk melakukan replikasi dan sinkronisasi data. Data pengujian yang didapatkan kemudian dijadikan bahan perbandingan untuk mencari nilai rata-rata, selisih waktu, dan kecepatan proses sinkronisasi dan replikasi data, yang kemudian dapat menjadi ukuran seberapa baik implementasi metode *synchronous replication* pada *cluster server*, seperti yang ditunjukkan pada Tabel 1 dan dalam bentuk grafik pada Gambar 3.

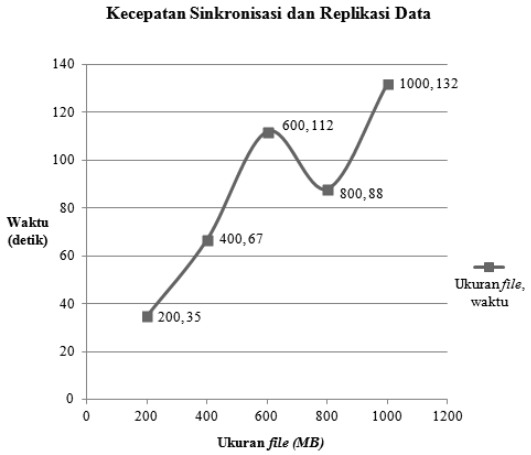
Tabel 1 Hasil Perbandingan Pembuatan *File* dan Replikasi Antar *Cluster Server*

No.	Ukuran *File*	*Node1*			*Node2*			Selisih Waktu Dibuat di Node1 dan Notifikasi di Node2 (detik)	Selisih Waktu Selesai Node1 dan Node2 (detik)
		Dibuat (jam)	Jangka Waktu Pembuatan (detik)	Selesai (jam)	Notifikasi (jam)	Mulai (jam)	Selesai (jam)		
1	200 *MB*	00:06:39	16	00:06:55	00:06:39	00:06:47	00:07:14	0	19
2	400 *MB*	00:13:44	30	00:14:14	00:13:44	00:13:48	00:14:51	0	37
3	600 *MB*	00:30:46	45	00:31:31	00:30:51	00:30:56	00:32:27	5	56
4	800 *MB*	00:42:43	67	00:43:50	00:42:45	00:42:48	00:44:11	2	21
5	1 *GB*	00:52:45	81	00:54:06	00:52:50	00:52:52	00:54:57	5	51
Rata-Rata Selisih Waktu dengan Total Ukuran *File* 3 *GB*								2,4	36,8

Hasil yang didapatkan dari pengujian replikasi *file* berukuran 800 *MB* lebih cepat dibandingkan dengan yang berukuran 600 *MB*. Hal tersebut dikarenakan pada saat replikasi di *node2* untuk *file* berukuran 600 *MB* sempat terhenti, yang disebabkan oleh beberapa faktor sebagai berikut:

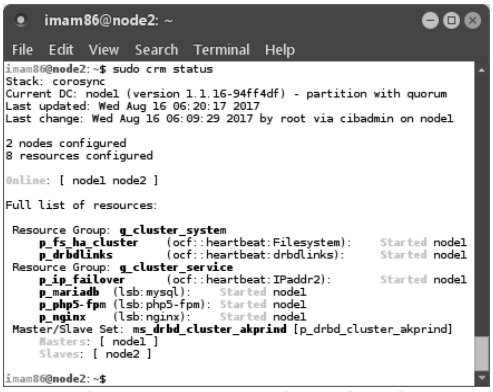
1. Kondisi *traffic* jaringan *cluster* yang padat pada saat pengujian.
2. Beberapa proses yang berjalan pada *node1* menghambat proses replikasi ke *node2*, seperti *web server*, *database server*, dan layanan *cluster* lainnya yang mempengaruhi *processing time* pada *CPU* dan *memory*, serta *I/O latency* pada media penyimpanan di *node1*.

Pada kondisi sinkronisasi dan replikasi *file* yang sempat terhenti, dilakukan pengecekan kembali di *node2*, untuk memastikan integritas dan konsistensi data yang akan direplikasi. Setelah proses pengecekan selesai dilakukan, maka kemudian proses replikasi dari *node1* ke *node2* dapat dilanjutkan dan hasilnya berjalan dengan lancar.

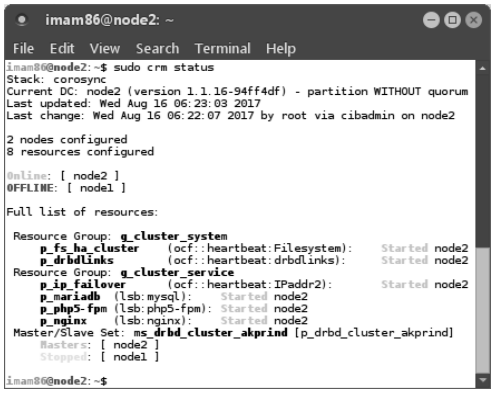


Gambar 3 Grafik Kecepatan Sinkronisasi dan Replikasi Data

Pengujian *failover* akan dilakukan dengan menonaktifkan *node1* (*poweroff*). Tujuannya adalah untuk melihat apabila salah satu *server* mengalami *down*, maka *server* yang lainnya akan melakukan *take over* (mengambil alih tugas) secara otomatis. Hasilnya akan terlihat seperti yang ditunjukkan pada Gambar 4 sebelum terjadinya *failover* dan Gambar 5 setelah terjadinya *failover*. *Node2* sukses melakukan *take over*, yaitu mengambil alih tugas secara otomatis. *Output* dari *cluster resource manager* juga terlihat bahwa status *node1* sekarang adalah *offline*, dan *node2* yang bertugas menjadi *master* pada sistem *cluster*.



Gambar 4 Status *Node* pada *Cluster Resource Manager* sebelum Terjadi *Failover*

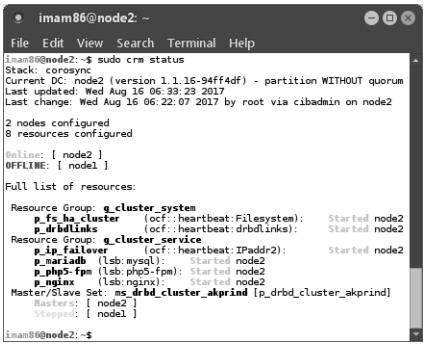


Gambar 5 Status *Node* pada *Cluster Resource Manager* setelah Terjadi *Failover*

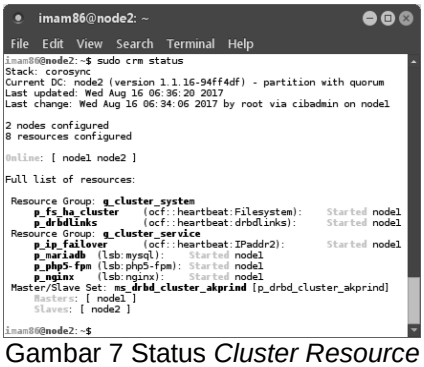
Pengujian *failback* akan dilakukan dengan mengaktifkan kembali *node1* yang statusnya *offline* untuk menjadi *online* kembali. Skenario *failback* yang akan dilakukan cukup sederhana, yaitu dengan mengaktifkan *node1* kembali, dan melihat status *cluster resource manager* pada *node2* yang sedang dalam posisi *master* dan *take over*. Ketika *node1* telah aktif kembali, maka seharusnya dapat terlihat bahwa status pada *cluster resource manager* pun akan berubah.

Pada Gambar 6 menunjukkan saat sebelum terjadinya *failback*, *node2* masih menjadi *master*. Ketika *node1* diaktifkan kembali, maka *node1* sukses melakukan *take over* dari *node2*, yaitu mengambil kembali tugasnya secara otomatis, seperti yang

ditunjukkan pada Gambar 7. *Cluster resource manager* juga melaporkan bahwa status *node1* sekarang adalah *online* dan bertugas menjadi *master* pada sistem *cluster*, sedangkan *node2* kembali bertugas menjadi *slave*.

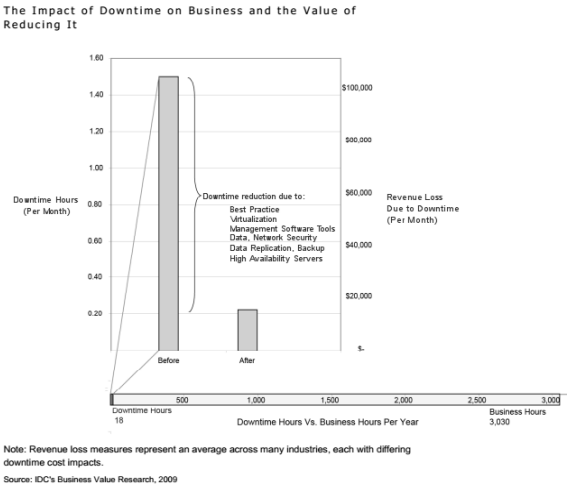


Gambar 6 Status *Cluster Resource Manager* sebelum Terjadi *Failback*



Gambar 7 Status *Cluster Resource Manager* Berubah saat Kondisi *Failback*

Studi berbasis pelanggan *IDC* menunjukkan bahwa setiap tahun, perusahaan berskala menengah rata-rata mengalami 16-20 jam kerja *downtime* pada jaringan, sistem, atau aplikasi, yang mengakibatkan kerugian pendapatan per jam rata-rata sebesar 70.000 *dollar* untuk satu kelompok studi (IDC, 2009). Penerapan beberapa praktik terbaik telah memungkinkan perusahaan berskala menengah untuk mengurangi waktu *downtime* secara signifikan, dengan rata-rata pengurangan *downtime* sebesar 25% per tahun selama tiga tahun terakhir. Pada Gambar 8 menunjukkan penelitian *ROI (Return on Investment)* dari *IDC* telah menemukan beberapa solusi terbaik untuk mengurangi waktu *downtime* dan kerugian yang ditimbulkan akibat *downtime*. *Failover clustering* adalah salah satu cara untuk mengurangi *downtime* hingga sebesar 43 persen (IDC, 2009).



Gambar 7 Kerugian Akibat *Downtime* pada Sektor Bisnis dan Nilai Pengurangannya (IDC, 2009)

KESIMPULAN

Perancangan dan implementasi infrastruktur jaringan *high availability cluster* yang direncanakan telah sukses dilakukan secara nyata, dengan menggunakan *Raspberry Pi* dan aplikasi *open source*. Penggunaan aplikasi *DRBD* sebagai fungsinya untuk *synchronous replication* dan *data redundancy* dapat meningkatkan ketersediaan sistem terhadap data yang dibutuhkan. Apabila terjadi permasalahan di *server*, solusi untuk meminimalkan atau menghilangkan risiko kehilangan data telah berhasil dilakukan.

Implementasi *failover* dan *failback* pada sistem *cluster* akan mengurangi waktu *downtime server*, karena jika salah satu *server* mengalami *down* maka *server* yang lain akan melakukan *take over* (mengambil alih tugas) secara otomatis. Kombinasi antara penggunaan aplikasi *Corosync* sebagai *layer* komunikasi pada sistem *cluster*, dan *Pacemaker* sebagai aplikasi yang berfungsi untuk *cluster resource management*, mendukung tercapainya tujuan untuk membuat infrastruktur jaringan *high availability cluster*.

Konsep *high availability cluster* wajib untuk tidak memiliki *Single Point of Failure*. Hal ini juga termasuk bagaimana caranya untuk secara otomatis dapat mendeteksi jika suatu *node* mengalami kegagalan. Dalam mencapai tujuan infrastruktur jaringan *cluster* harus diuji secara menyeluruh dan tidak cepat merasa puas, hingga siap untuk menjadikannya ke dalam level jaringan *server* yang siap produksi. Pengujian terhadap berbagai macam konfigurasi *cluster* wajib dilakukan agar dapat mengetahui bagaimana masing-masing *cluster node* akan berperilaku ketika terjadi suatu kegagalan dan menanganinya dengan benar. Tentunya untuk mengoptimalkan jaringan *cluster* sangat membutuhkan kerja keras dan kesabaran.

Implementasi *high availability cluster* dapat digunakan untuk menjaga kestabilan *server* dan menyediakan ketersediaan tinggi akan permintaan yang dapat menambah beban pada *server*. Dengan total *redundancy* dari beberapa *node*, maka sistem *cluster* dapat meminimalkan kemungkinan terjadinya *downtime* dan membantu untuk mencapai *uptime* sistem yang lebih besar.

DAFTAR PUSTAKA

Barker, R., 2002, *Storage Area Network Essentials*, Wiley, New York

Charles, B., dkk. 2014. *MySQL High Availability*, Edisi Kedua, O'Reilly Media, Sebastopol

Critchley, T., 2015, *High Availability IT Services*, CRC Press, Boca Raton

Febriani, T.R., 2011, Implementasi dan Analisa Sistem *Failover Cluster*, Skripsi, Program Studi Teknik Komputer, Fakultas Teknik, Universitas Indonesia, Depok

Gabriel, C., 2016, *CentOS High Performance*, Packt Publishing, Birmingham

Hellman, B., dkk, 2012, *The DRBD User's Guide,* https://www.linbit.com/en/resources/documentation, diakses tangggal 17 Juli 2017

Hidayat, F., 2012, Implementasi dan Analisa Redundansi dan *High Availability* dalam *Server* untuk *Diskless Thin Client* Berbasis *Storage Area Network*, Skripsi, Program Studi Teknik Komputer, Fakultas Teknik, Universitas Indonesia, Depok

IDC, 2009, *Reducing Downtime and Business Loss: Addressing Business Risk with Effective Technology*, *IDC's Business Value Research* 2009, http://www.compaq.net/hpinfo/newsroom/press_kits/2009/CompetitiveEdge/ReducingDowntime.pdf, diakses tanggal 11 Agustus 2017

Jones, M.T., 2010, *High availability with the Distributed Replicated BlockDevice*, https://www.ibm.com/developerworks/library/l-drbd/l-drbd-pdf.pdf, diakses tanggal 14 Juli 2017

Kopper, K, 2005, *The Linux Enterprise Cluster*. No Starch Press, San Francisco

Kula, P.J., 2014, *Raspberry Pi Server Essentials*, Packt Publishing, Birmingham

MacIntyre, P., dkk, 2011, *Pro PHP Programming*, Apress, New York

Mavro, P., 2014, *MariaDB High Performance*, Packt Publishing, Birmingham

Resman, M., 2015, *CentOS High Availability*, Packt Publishing, Birmingham

Susanti, E., dan Joko Triyono, 2016, Pengembangan Sistem Pemantau dan Pengendali Kendaraan Menggunakan *Raspberry Pi* dan *Firebase*, Konferensi Nasional Teknologi Informasi dan Komunikasi (KNASTIK 2016), UKDW, hal. 144-152, Lembaga Penelitian IST AKPRIND, Yogyakarta

Upton, E., dkk, 2016, *Raspberry Pi User Guide*, Edisi Keempat, Wiley, Chicester

Vugt, S.V., 2014, *Pro Linux High Availability Clustering*, Apress, New York