ALGORITMA: Jurnal Ilmu Komputer dan Informatika
Volume: 04, Number: 01, April 2020 ISSN 2598-6341 (online)

Rancang Bangun *Server HAproxy Load Balancing Master to Master MySQL (Replication)* Berbasis *Cloud Computing*

Eka Pandu Cynthia[1], Iwan Iskandar[2], Anwar Alfaruqi Sipayung[3]
[1,2,3]Jurusan Teknik Informatika, Fakultas Sains dan Teknologi, UIN Sultan Syarif Kasim Riau
Email: [1]eka.pandu.cynthia@uin-suska.ac.id*, [2]iwan.iskandar@uin-suska.ac.id,
[3]anwar.alfaruqi.sipayung@student.uin-suska.ac.id.

Abstrak

Saat ini, perkembangan teknologi *cloud computing* mengalami perkembangan yang pesat. Selaras pula dengan peningkatan akan kebutuhan dan penggunaan teknologi *cloud computing* itu sendiri. Peningkatan yang sangat pesat dan tinggi ini dapat mengakibatkan peningkatan beban untuk *server* yang ada, sehingga dapat mengakibatkan terjadinya *overload* pada *traffic* jalur koneksi. Untuk menghindari dan mengoptimalkan manajemen dalam *server*, diperlukan adanya pembagian beban jaringan mengunakan berbagai solusi yang ada pada metode *load balancing*. Salah satu solusinya adalah menggunakan perangkat lunak *HAProxy*. *HAProxy* menjembatani minimal 2 *server* untuk mengaplikasikan *load balancing* dan mengunakan algoritma penjadwalan *Round Robin*. Pada pengelolaan *database* dari dua *server*, dapat digunakan replikasi *Master to Master* dengan mengunakan *MySQL* sebagai media manajemen basis data yang ada. Dari hasil pengujian yang dilakukan pada penelitian ini, *server HAProxy* dinilai mampu menangani permasalahan dibandingkan dengan *server* tunggal. Replikasi yang diuji juga dinilai dapat mejadi solusi untuk menjembatani perubahan data yang ada pada *server HAProxy*.

Kata kunci: *Cloud Computing, HAProxy, Load Balancing, MySQL,* Replikasi *Master to Master.*

Abstract

At present, the development of cloud computing technology is experiencing rapid development. Also in line with the increase in the need and use of cloud computing technology itself. This rapid and high increase can cause an increase in load for the existing server so that it can result in overload on the connection path traffic. To avoid and optimize management on the server, it is necessary to share network load using various solutions available in the load balancing method. One solution is to use HAProxy software. HAProxy bridges at least 2 servers to apply load balancing and use the Round Robin scheduling algorithm. In the database management of two servers, replication Master to Master can be used by using MySQL as an existing database management media. From the results of tests conducted in this study, the HAProxy server is considered capable of handling problems compared to a single server. A replication that was tested was also considered to be a solution to bridge changes in existing data on the HAProxy server.

Keywords: *Cloud Computing, HAProxy, Load Balancing, MySQL, Master to Master Replication.*

1. PENDAHULUAN

Penerapan teknologi informasi dalam suatu perusahaan ataupun organisasi merupakan hal yang sangat dibutuhkan untuk saat ini. Pada tahun 2017 penggunaan internet di indonesia menduduki peringkat ke-6 di dunia. Menurut Asosiasi Penyelenggara Internet Indonesia (APJII) dengan 143,26 juta pengguna internet atau setara dengan 54,68% dari total jumlah penduduk indonesia. Dari tingginya jumlah penggunaan internet ini tentunya membuka peluang bisnis yang dapat berhubungan dengan teknologi informasi. Saat ini sebuah perusahaan atau organisasi yang menggunakan teknologi yang mengaplikasi layanan sistem informasi baik itu digunakan untuk kepentingan *internal* perusahaan atau organisasi tersebut atau untuk hubungan dengan *eksternal* perusahaan atau organisasi pasti memiliki *server* baik yang memiliki secara fisik atau pun dengan menyewa *server cloud*. Kebutuhan dan kemampuan dari *server* yang dimiliki baik itu *server* fisik maupun dengan menyewa *cloud computing* tentu juga berpengaruh dalam usaha perusahaan. Berbagai cara digunakan untuk mengoptimalkan *server* yang dimiliki.

Saat ini sebuah perusahaan atau organisasi yang menggunakan teknologi yang mengaplikasi layanan sistem informasi baik itu digunakan untuk kepentingan *internal* perusahaan atau organisasi tersebut atau untuk hubungan dengan *eksternal* perusahaan atau organisasi pasti memiliki *server* baik yang memiliki secara fisik atau pun dengan menyewa *server cloud* . Kebutuhan dan kemampuan dari *server* yang dimiliki baik itu *server* fisik maupun dengan menyewa *cloud computing* tentu juga berpengaruh dalam usaha perusahaan. Berbagai cara digunakan untuk

mengoptimalkan *server* yang dimiliki.

Berbagai cara dilakukan agar pelayanan yang dimiliki oleh sistem yang dimiliki berjalan dengan optimal. Sistem yang handal tentunya akan dapat menanngani masalah yang ada. Sebagai contoh saat sistem berjalan, meningkatnya *request* / permintaan pengaksesan menyebabkan *server* tidak dapat menangani permintaan dari pengguna sehingga menyebabkan *overload*. Untuk menangani masalah tersebut diperlukan adanya arsitektur multi*server* dengan menerapkan teknologi *load balancing* pada sebuah *web server*. *Load balancing* adalah teknik untuk menyalurkan beban *traffic* jaringan pada dua jalur atau lebih secara sama rata. Dengan penggunaan *load balancing* sebuah sistem atau *web server* dapat mengoptimalkan performa *resource* menjadi lebih efisien.

Dari penjabaran diatas, maka akan dilakukan sebuah penelitian tentang bagaimana *response time*, *throughput* dan pengujian pembagian beban dari *server* yang di *load balancing*, serta pengujian pada sinkronisasi setiap *database*. Pengujian sinkronisasi *database*, dilakukan untuk mengetahui perubahan data pada satu *server* di *database server* lain dimana pengukuran menghitung proses perubahan data dan waktu yang diperlukan. Pada penelitian ini juga akan digunakan dengan menggunakan algoritma penjadwalan yaitu *round robin.*

2. LANDASAN TEORI

2.1 *Cloud Computing*

Cloud computing atau komputasi awan dapat diartikan sebagai sebuah model yang memungkinkan pengguna untuk dapat mengakses sumber daya seperti *processor*, *storage* (media penyimpan), *network*, *software* (perangkat lunak) secara abstrak dan diberikan sebagai layanan di jaringan/internet (Purbo, 2011). Secara umum arsitektur *cloud computing* terbagi menjadi 3 bagian yaitu infrastruktur, *platform* dan aplikasi. Setiap layanan yang berada dalam lingkup *cloud computing* dapat diakses hanya dengan menghubungkan perangkat ke internet tanpa perlu meng-*install* layanan yang di inginkan. Untuk dapat melakukan akses terhadap layanan *cloud computing* hanya dibutuhkan *web browser* atau antarmuka program.

2.2 *Web Server*

Server merupakan sebuah sistem komputer dimana difungsikan untuk memberikan layanan, membatasi dan juga mengontrol akses pada komputer klien yang berada dalam sebuah jaringan komputer. Komputer yang bertindak sebagai *server* ini menyediakan *resource* yang dapat digunakan untuk komputer lain yang bertindak sebagai komputer klien. *Web server* adalah sebuah perangkat lunak yang dipasang pada *server* yang berfungsi untuk menyediakan layanan permintaan data dengan protocol HTTPS atau HTTP yang dapat diakses dengan menggunakan browser. Cara kerjanya secara sederhana adalah *web server* akan merespon permintaan yang ada dengan mengirimkan konten tersebut kembali dalam bentuk gambar, tulisan atau bentuk lainnya. Kemudian akan ditampilkan pada *browser* (Yeager & McGrath, 1996).

2.3 *Load Balancing*

Dalam terjemahannya *load balancing* dapat diartikan sebagai peyeimbang beban. Jika dimasukkan kedalam jaringan komputer, *load balancing* merupakan metode yang digunakan sebagai penyeimbang beban bagi jaringan yang diharapkan mampu mendistribusikan beban yang ada menjadi sama rata sehingga beban kerja menjadi lebih ringan. Sebuah *load balancing* mendistribusikan beban yang dimiliki jaringan dalam mengakases beberapa *server* untuk mendapatkan cara yang paling efisien, dengan memngirimkan permintaan hanya kepada *server* yang sedang *online* dan mampu memenuhi permintaan tersebut sehinga dapat diandalkan. *Load balancing* berada diantara internet dan aplikasi *web servers*, dimana *load balancer* mengatur alur arah lalu lintas paket yang di akan dikirim maupun di terima.

2.4 Algoritma *Round Robin*

Round robin merupakan salah satu algoritma penjadwalan sederhana dalam suatu proses sistem operasi. Algoritma *round robin* dijalankan dengan membagi waktu setiap proses yang ada pada porsi yang sama dan dalam urutan melingkar, menjalankan semua proses tanpa prioritas (Sony, Andono., & Pratama, 2013). Penjadwalan *round robin* tidak hanya digunakan dalam penjawalan sistem operasi saja, namun dapat juga digunakan untuk penjadawalan lainya seperti penjadwalan paket data dalam jaringan komputer. *Round robin* dirancang untuk sistem *time sharing*. Dalam algoritma *round robin,* antrian yang siap diperlakukan atau dianggap sebagai antrian sirkular. CPU akan mengalirkan antrian yang sudah siap dan mengalokasikan masing-masing proses dengan interval waktu tertentu sampai satu *time slice quantum.* Algoritma ini berjalan dengan menggilir proses yang ada pada antrian. Setiap Proses akan mendapat jatah sebesar *time* quantum. Jika *time* quantum-nya habis atau proses sudah selesai, CPU akan dialihkan ke proses yang selanjutnya. Dengan proses ini memberikan keadilan pada setiap proses, karena tak ada proses yang diprioritaskan, semua proses mendapat jatah waktu yang sama dari CPU yaitu (1/n), dan tak akan menunggu lebih lama dari (n-1)q dengan q adalah lama 1 *quantum.*

2.5 *Virtual Private Server*

Virtual private server (VPS) adalah sebuah metode untuk membagi sumber daya atau *resource* dari komputer *server* menjadi beberapa *server virtual*, dimana *server virtual* yang telah dibagi-bagi tersebut dapat berjalan secara mandiri. *Virtual private server* dapat memiliki kemampuan layaknya komputer *server*, dimana *virtual private server* mampu menjalankan sistem operasi sendiri dan menjalankan berbagai aplikasi yang diperlukan sebuah *server* asli (Idcloudhost.com, 2016). Setiap *virtual private server* dapat dihapus, ditambahkan, dan diubah semua *file* yang ada didalamnya termasuk *file* yang ada di dalam *root*, serta dapat meng-*install software* aplikasi sendiri atau mengkonfigurasi root *application software*. Setiap *virtual private server* bekerja seperti sebuah *server* yang terpisah, dimana setiap *virtual private server* mempunyai *ip address*, *port number*, *tables*, *filtering* dan *routing rules* sendiri. Sumber daya (*resource*) ataupun spesifikasi sebuah *virtual private server* baik *CPU Core*, *CPU Usage*, *RAM*, dan media penyimpan (*storage*) tergantung dari spesifikasi atupun sumber daya yang dimiliki oleh penyedia layanan. *Virtual private server* haruslah selalu terhubung dengan internet dengan kecepatan tinggi agar setiap pengguna bisa dengan mudah mengaksesnya.

2.6 *MySQL*

MySQL merupakan sebuah perankat lunak (*software*) sistem manajemen basis data (*database management system*) yang mengunakan *SQL* (*Structured Query Language*) yang sangat cepat, multithreaded, multi-pengguna, dan kuat. *MySQL Server* ditujukan untuk penggunaan sistem produksi *mission-critical*, dengan beban berat serta untuk dimasukkan ke dalam pengembangan perangkat lunak yang digunakan secara missal (Oracle, 2019a). Perangkat lunak *MySQL* memiliki dua Lisensi, yaitu berbayar dan umum. Pengguna dapat memilih untuk menggunakan *MySQL* sebagai produk *Open Source* di bawah ketentuan GNU *(General Public License)* atau dapat membeli lisensi komersial standar dari Oracle.

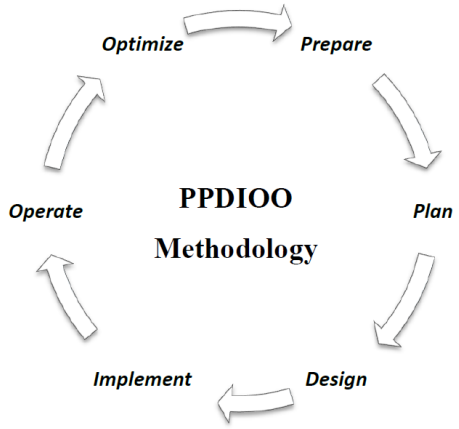
2.6.1 *MySQL Replication*

Dalam *MySQL* dimungkinkan adanya replikasi data dari satu *server database MySQL* untuk disalin ke satu atau lebih *server database MySQL*. *Database* sumber disebut dengan *master* sedangkan *database* yang lain ini disebut sebagai *slave*. Replikasi secara *default* bersifat *asynchronous*, di mana *database slave* tidak selalu harus terhubung dengan *database master* secara terus-menerus untuk mendapatkan *update*. Berdasarkan pada konfigurasi, pengguna dapat mereplikasi semua *database*, salah satu *database* yang dipilih, atau bahkan tabel yang dipilih dalam *database* (Oracle, 2019b). Berdasarkan pada konfigurasi *default, MySQL Replication* yang digunakan bersifat asynchronous, untuk dapat berubah menjadi *synchronous* maka akan menghasilkan NDB *(Network Database)* Cluster. NDB *Cluster* tidak mendukung replikasi menggunakan GTID, replikasi semisinkron juga tidak didukung oleh mesin penyimpanan NDB.

NDB Cluster disediakan oleh *MySQL* sebagai replikasi *multi-master* adalah metode replikasi *database* yang memungkinkan data disimpan oleh sekelompok komputer, dan data dapat diperbarui oleh setiap anggota. Seluruh angota kelompok dapat menangani permintaan data klien. Sistem replikasi multi-master bertanggung jawab untuk menyebarkan perubahan data yang dibuat oleh masing-masing anggota ke anggota grup lainnya, dan menyelesaikan setiap konflik yang mungkin timbul antara perubahan bersamaan yang dilakukan oleh anggota yang berbeda. Replikasi *multi-master* dapat dibedakan dengan replikasi *master-slave*. Pada *multi-master*, anggota grup ditunjuk sebagai "*master*" untuk bagian data tertentu dan merupakan satu-satunya *node* yang diizinkan untuk memodifikasi item data tersebut. Anggota lain yang ingin memodifikasi item data harus menghubungi *node master* terlebih dahulu. Replikasi *multi-master* juga dapat dibedakan dengan *failover clustering*, di mana *server slave* yang pasif mereplikasi data *maste*r untuk mempersiapkan pengambilalihan jika *master* berhenti berfungsi. *Master* adalah satu-satunya *server* yang aktif untuk interaksi *client*. Tujuan utama replikasi *multi-master* adalah untuk peningkatan *availability* dan *response time server* yang lebih cepat.

3. METODE PENELITIAN

Metodeologi penelitian merupakan acuan dalam melaksanaan sebuah penelitian. Metodologi penelitian berisi rencana urutan kerja beraturan,sehingga didapatkan hasil yang sesuai dengan yang diharapkan. Penelitian dilakukan dengan berdasarkan pada metodologi PPDIOO (*Prepare, Plan, Design, Implement, Operate, Optimize*). PPDIOO merupakan standart pengembangan siklus hidup dalam pengelolaan jaringan yang digunakan oleh CISCO.



Gambar 1. Metodologi Penelitian

Dalam PPDIOO terdapat 6 fase (Cisco, 2010), yaitu:

1. Fase *prepare* atau persiapan merupakan tahapan penetapan kebutuhan dari organisasi dang mengembangkan strategi jaringan serta mengusulkan konsep tertinggi yang paling dapat mendukung arsitektur.
2. Fase *plan* atau perencanaan mengidentifikasikan persyaratan awal berdasarkan tujuan, fasilitas, kebutuhan pengguna, dan sebagainya. Fase ini menjelaskan karakterisasi serta menilai jaringan yang ada dan melakukan analisis untuk menentukan apakah infrastruktur sistem yang ada, lokasi, dan lingkungan operasional dapat mendukung sistem yang diusulkan. Penentuan sistem baru berdasarkan pada permasalahan yang dikemukakan pada fase *prepare*.
3. Fase *design* atau desain merupakan fase perencanaan dari persyaratan yang telah dibuat pada fase perencanaan. Spesifikasi desain jaringan didesain secara terperinci dan komprehensif dimana telah memenuhi persyaratan dari fase perencanaan. Fase desain adalah dasar untuk kegiatan implementasi.
4. Fase *Implement* adalah fase dimana jaringan yang dibangun dan komponen tambahan dimasukkan sesuai dengan spesifikasi desain.
5. Fase *Operate* adalah tes akhir dari kesesuaian desain. Fase operasional melibatkan pemeliharaan kesehatan jaringan melalui operasi sehari-hari. Deteksi kesalahan, koreksi, dan pemantauan kinerja yang terjadi dalam operasi sehari-hari memberikan data awal untuk fase optimasi.
6. Fase *Optimize* atau optimaliasi merupakan fase yang dapat mendorong perancangan ulang jaringan jika terdapat terlalu banyak masalah dan kesalahan jaringan muncul, jika kinerja tidak memenuhi harapan, atau jika aplikasi baru diidentifikasi untuk mendukung persyaratan organisasi dan teknis.

Dalam penelitian ini, 6 fase PPDIOO digunakan sebagai model atau kerangka kerja. Dimana tidak semua fase dalam PPDIOO akan dijalankan pada penelitian ini, serta ada berberapa tambahan yang akan disesuaikan dengan penelitian ini.

4. HASIL DAN PEMBAHASAN

4.1 Analisa Sistem Lama

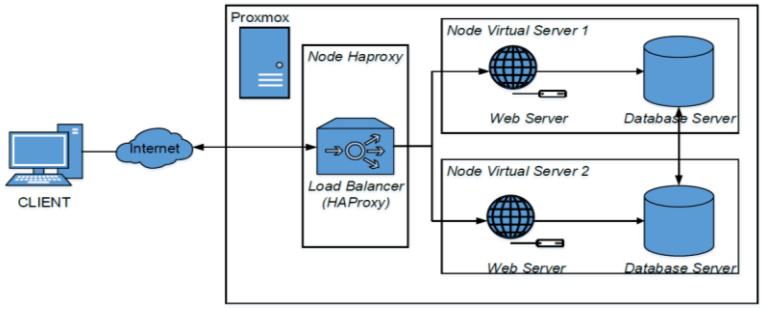
Sistem lama pada penelitian ini disimulasikan menggunakan *server* tunggal, dimana pada *server* tunggal tersebut hanya memiliki 1 *web server* dan 1 *database server*. Akses *server* tunggal hanya memilik 1 arah saja, hal ini tentunya akan berpengaruh dalam performa pengaksesan jika *web server* diakses oleh oleh banyak *client* pada saat bersamaan. *Server web* tunggal bisa saja mengalami penurunan performa atau bahkan kelebihan kapasitas atau *overload* jika permintaan akses melebih kemampuan *server* itu sendiri. Dari pernyataan tersebut, didapatlah peluang untuk menangani permalahan tersebut yaitu dengan membangun *server load balancer* yang dapat menangani 2 buah bahkan lebih *server* dalam satu jaringan yang sama sehingga mampu melayani lebih banyak permintaan dari pada *server* tunggal.

4.2 Perancangan *Server*

Solusi yang diberikan adalah dengan mengunakan dua buah *node server* yang dihubungkan dalam satu jaringan dengan menggunakan *HAProxy* sebagai *load balancer* dari kedua *node server*.

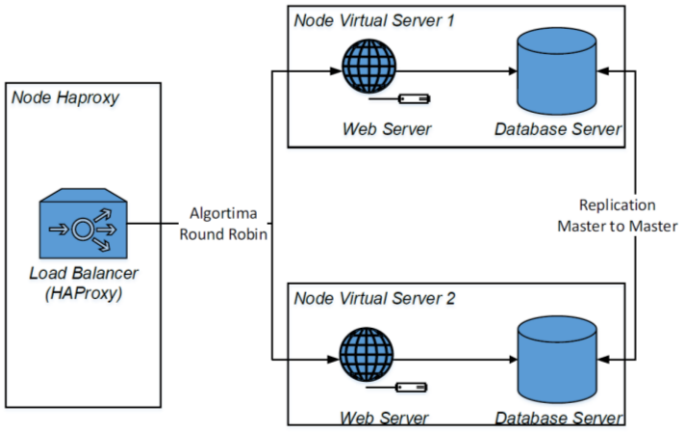
4.2.1 Topologi Jaringan

Pada penelitian ini dibangun sebuah layanan berbasis *cloud computing* akan menggunakan *platform Proxmox* dan Centos 6. *Proxmox* menggunakan antar muka (*interface*) berbasis *web*. Dalam sebuah *server* fisik yang pada penelitian ini akan dipasangkan dengan sistem operasi *Proxmox*, didalamnya akan dibuat 3 *node* yang menggunakan sistem operasi CentOS. Dari 3 *node* tersebut, 1 *node* nantinya akan digunakan sebagai *load balancer* dari 2 *node* lainnya. *Node load balancer* akan di-*install*-kan *HAProxy* untuk media *gateway load balancing* untuk membagi beban *request* yang akan menuju 2 *node* lainya. Saat pengguna meminta akses (*request*) untuk mengakses *server*, pengguna akan dialihkan menuju ke *server* yang sudah ter-*install HAProxy* sebagai *load balancing* kemudian akan di teruskan ke salah satu *server*. Berikut adalah topologi dari jaringan yang akan dibangun pada penelitian ini :



Gambar 2. Topologi Jaringan Baru Yang Dibangun

Gambar 2. merupakan gambaran dari topologi jaringan yang akan di buat dalam penelitian ini, *request* dari *client* akan diteruskan oleh *load balancer* (*HAProxy*) menuju salah satu *node virtual server*. Untuk dapat mengatur penjadwalan dari kedua *node virtual server*, digunakan algoritma *round robin*.



Gambar 3. Sistem Kerja *Server* Yang Dibangun

Gambar 3. menjelaskan alur dari sistem kerja dari *load balancer HAProxy* dan *database replication master to master*, dimana jika terjadi perubahan pada salah *database* di *node virtual*, *database* yang berada di *node virtual* lainya akan mengikuti perubahan yang ada.

4.2.2 Konfigurasi Perangkat

Rancangan konfigurasi yang digunakan pada penelitian ini meliputi jenis perangat, *IP Address*, *subnet mask* serta keterangan tambahan dari setiap perangkat.

Tabel 1. Konfigurasi Perangkat

Perangkat	IP Address	Subnet Mask	Keterangan
Server Fisik	192.168.137.117:8006	255.255.255.0	*Proxmox*
Load balancer (*HAProxy*)	192.168.137.184	255.255.255.0	CentOS
Node Virtual Server1	192.168.137.221	255.255.255.0	CentOS

Node Virtual Server2	192.168.137.150	255.255.255.0	CentOS

4.2.3 Kebutuhan Perangkat

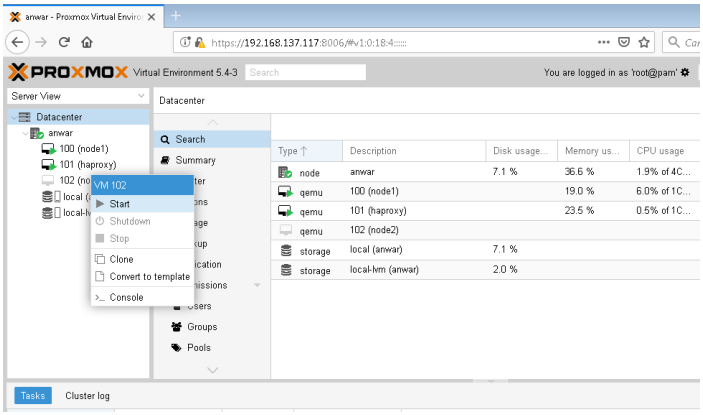
Kebutuhan perangkat baik itu perangkat lunak (*software)* dan perangkat keras (*hardware*) yang digunakan dalam penelitian ini akan dibagi dalam 2 bagian. Bagian pertama adalah kebutuhan dari *server* itu sendiri dan bagian kedua adalah kebutuhan dari *client.*

1. *Server Proxmox*
   Server yang akan digunakan dalam penelitian ini berbasis pada virtualisasi yaitu *Proxmox*. yang didalamnya terdapat *template* OS. Pada *template* ini lah akan di-*install* 3 *node* yang akan digunakan.
2. *Laod Balancer HAProxy*
   Pada penelitian ini terdapat *server* hasil virtualisasi yang digunakan sebagai *load balancer*.
3. *Node Virtual Server*
   Pada penelitian ini terdapat 2 *node server* virtual. *Server* tersebut merupakan kontainer untuk *web server* dan *database*.
4. PC Pengguna (*Client*)
   Pada peneliatan ini PC *client* digunakan untuk melakukan pengujian pada penelian ini.

4.3 Implementasi

4.3.1 Pembuatan *Node Virtual Server* dan *HAProxy*

Langkah awal dari rangkaian proses pembangunan *server HAProxy load balancing Master to Master MySQL(Replication)* pada penelitian ini dimulai dengan membuat *node virtual server* didalam *Proxmox*. Setalah mengakses *Proxmox* dengan *web browser* menggunakan IP yang telah di dapat dari pengaturan awal dari instalasi *Proxmox* yaitu 192.168.137.117:8006, langkah selanjutnnya adalah membuat 3 *node virtual server* yang masing-masing *node* akan menggunakan sistem operasi CentOS.



Gambar 4. Tampilan Awal Pengoperasian *Node* Pada *Proxmox*

Gambar 4. merupakan tampilan sebelum proses untuk menjalankan *node virtual server* yang telah dibuat. Dalam *Proxmox*, saat membuat *node virtual server* akan setiap *node* akan diberi ID yang berbeda. *Node virtual server* dengan ID 100 dan 102 akan digunakan sebagi *node virtual server* untuk *web server*, sedangkan *node* 101 akan digunakan sebagai *server HAProxy* untuk 2 *node* tersebut. Setelah pembuatan dan instalasi CentOS pada 3 *node virtual server* tersebut, langkah selanjutnya adalah dengan meng-*isntall* paket-paket yang dibutuh kan pada setiap *node virtual server*. Pada *node* 100 dan 102 akan di-*install* paket *sofware web server open source* yaitu "*Apache*". Pada *node* 101 yang digunakan untuk *server HAProxy* akan di-*install* paket aplikasi "*HAProxy*". Setalah melakukan instalasi paket-paket tersebut, dibuatlah *web* dasar sebagai pengujian dari keberhasilan instalasi *apache web server*. untuk mengetahui keberhasilan dari instalasi dan pembuatan *web* dasar *server*, dilakukan akses menuju *web* dasar dengan menggunakan komputer *client*.

Gambar 5. Web Sederhana Yang Diakses Dari *Browser Client*

4.3.2 Pengaturan *HAProxy*

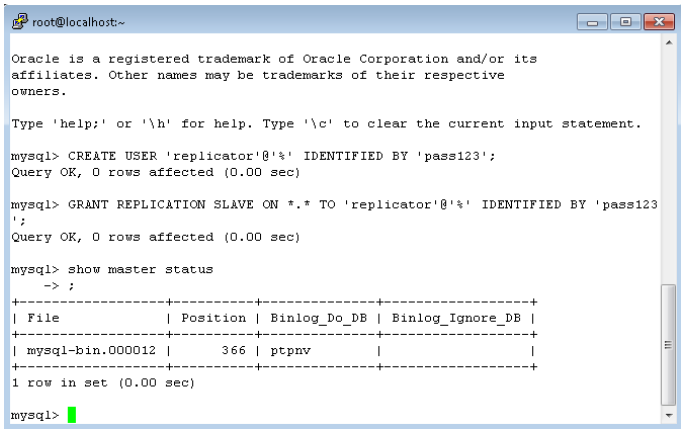
Setelah melakukan instalasi paket *HAProxy*, barulah konfigurasi dapat dilakukan. Untuk melakukan konfigurasi pada *HAProxy*, ada 2 tahap yang harus dilakukan. Yang pertama yaitu pengaturan yang ada dalam file "*HAProxy*.cfg" dimana ada beberapa hal yang harus di ubah dan ditambahkan. Hal hal yang perlu di ubah dan d tambahkan antara lain :

1. Menambahkan "*local2*" pada *log* dibaris ke-26, menambahkan local2 berfungsi untuk mengaktifkan log dari *HAProxy*.
2. *Timeout client* pada baris ke-53 diubah dari 1m menjadi 30s.
3. *Timeout server* pada baris ke-54 diubah dari 1m menjadi 30s.
4. Pada baris ke-62 *frontend http-in* ditambahkan port 80 dengan mengetikkan *bind:*80*.
5. Pada baris ke-67 *default_backend* adalah membuat pengaturan baru pada *backend* dengan cara mengubah nama menjadi *backend_servers*.
6. Pada baris ke-81 pengaturan *backend*, diubah namanya sesuai dengan pengaturan yang kita buat pada baris ke-67 yaitu *backend_servers.*
7. Pada bari ke-82 *balance*, diatur menjadi algoritma *round robin*.
8. Pada bari ke-83 *server*, dimasukkan alamat IP dari *node* 100, dan tambahkan pada baris berikutnya alamat IP dari *node* 102.

4.3.3 Pengaturan *Replication Master to Master MySQL*

Untuk melakukan replikasi pada kedua *database* langkah-langkah dalam melakukan replikasi ini, akan dijabarkan sebagai berikut:

1. Mengubah file yang konfigurasi *MySQL* dengan mengakases file "*my.cnf*", didalam file tersebut akan ditambahkan *id server* dan *binary log* dari *database*. Konfigurasi ini akan dilakukan pada kedua *server*. pada *server* 1 akan diberi id "1" dan "2" pada *server* 2.
2. Langkah selanjutnya adalah membuat hak akses pengguna yang akan digunakan untuk dapat memberi akses ke *database* sehingga dapat dilakukan perubahan dan mereplikasi *database*. Untuk membuat hak akses antara kedua *server*, digunakan perintah "***GRANT REPLICATION SLAVE ON *.* TO 'anwar1'@'%' IDENTIFIED BY 'password123'; FLUSH PRIVILEGES;***". Pada perintah diatas, "anwar1" merupakan nama akun dan "password123" merupakan kata sandinya. Langkah ini dilakukan pada kedua *server*.
3. Setelah membuat pengaturan untuk penggunaan hak akses *database*, langkah selanjutnya adalah mengecek status "*master*" dari *database* setiap *server* yang akan terhubung. Pengecekan dilakukan untuk mengetahui bianry log dari *database*, hasil dari perintah ini dapat dilihat pada Gambar 6. Konfigurasi ini dilakukan pada kedua *server*.



Gambar 6. Status *Master Server* 1

Gambar 6. merupakan hasil dari perintah "***show master status;***" pada *server node* 1. Dari perintah tersebut akan menghasilkan sebuah tabel dimana nama kolom tabel "***file***" dan "***position***" isi dari baris tersebut nantinya berperan penting untuk tahap konfigurasi selanjutnya.

4. Langkah konfigurasi selanjutnya adalah mematikan slave dari *database* dan mengubah master pada setiap *server*. Pada perintah pengubahan master sertiap *database*, "*master_host*", dimasukkan alamat IP dari *server* 1, pada perintah yang digunakan pada *server node* 2, begitupun sebaliknya. Pada kedua perintah tersebut ***master user*** dan ***master password*** diisi sesuai dengan akun pengguna yang telah dibuat sebelumnya. Pada

master_log_file dan ***master_log_pos*** merupakan data pada tabel "***file***" dan "***position***" pada saat penggunaan perintah "***show master status***" sebelumnya.

4.4 Pengujian Kinerja

Pengujian kinerja (*performance testing*) adalah proses menentukan kecepatan, respon, dan stabilitas dari suatu komputer, jaringan, *software program*, atau perangkat memiliki beban kerja. Pengujian kinerja dapat melibatkan tes kuantitatif yang dilakukan pada lingkungan kerja dalam skenario terbatas. Pada penelitian ini, terdapat beberapa skenario pengujian. Pengujian-pengujian yang dilakukan adalah pengujian *response time*, *throughput*, dan pengujian pembagian beban dari *server* yang di *load balancing*, serta pengujian pada sinkronisasi setiap *database*.

Dalam penelitian ini, pengujian *response time* dan *throughput*, dilakukan dengan menggunakan *tool* yang berupa aplikasi yaitu "Wireshark". Wireshark merupakan aplikasi penganalisa protokol jaringan yang memungkinkan untuk dapat melihat apa yang terjadi di jaringan pada tingkat mikroskopis. Wireshark merupakan standar yang banyak digunakan oleh perusahaan komersial dan nirlaba, lembaga pemerintah, dan lembaga pendidikan. Versi wireshark yang digunakan dalam penelitian ini adalah versi 2.6.10.

4.4.1 *Response Time Utilization*

Pengujian *response time* (*Response Time Utilization*) merupakan pengujian yang lakukan untuk mengukur waktu yang dibutuhkan dari saat *request* terkirim hingga *response* diterima. Pada pengujian ini yang dilihat adalah besaran waktu yang dibutuhkan untuk melakukan *service*. Pengujian ini dilakukan dengan melihat besaran waktu saat *service* dijalankan dan diakses oleh *client* secara bersamaan. Pada penelitian ini, untuk mengukur *response time* akan digunakan aplikasi wireshark sebagai *tool* untuk dapat melihat waktu *respon*se yang diperlukan untuk dapat mengakses *server*. Akan dilakukan 7 kali uji *request* pada 2 keadaan, dimana keadaan pertama adalah *server* yang diaktifkan hanya *node virtual server* 1 dan diuji *respon time* dari *server* tersebut. Skenario kedua adalah dimana *node virtual server HAProxy* dijalankan, keadaan ini bertujuan untuk menguji *respon time* dari *HAProxy*. Dari kedua pengujian tersebut didapatlah perhitungan waktu sebagai berikut:

Tabel 2. Hasil Uji *Request* Skenario 1

Source	***Destination***	***Protocol***	***Length***	***Time Since Request***	***Info***
192.168.137.221	192.168.137.1	*HTTP*	204	0.001754	*HTTP/1.1 304 Not Modified*
192.168.137.221	192.168.137.1	*HTTP*	524	0.00129	*HTTP/1.1 404 Not Found (text/html)*
192.168.137.221	192.168.137.1	*HTTP*	204	0.001799	*HTTP/1.1 304 Not Modified*
192.168.137.221	192.168.137.1	*HTTP*	204	0.001315	*HTTP/1.1 304 Not Modified*
192.168.137.221	192.168.137.1	*HTTP*	204	0.001589	*HTTP/1.1 304 Not Modified*
192.168.137.221	192.168.137.1	*HTTP*	204	0.001048	*HTTP/1.1 304 Not Modified*
192.168.137.221	192.168.137.1	*HTTP*	204	0.001196	*HTTP/1.1 304 Not Modified*

Dari data tabel hasil uji request sebanyak 7 kali diatas, didapatlah rata-rata **0.001427s** sebagai hasil dari pengujian dari sekenario 1. Untuk sekernario 2 dapat dilihat pada tabel 3. dibawah ini :

Tabel 3. Hasil Uji *Request* Skenario 2

Source	***Destination***	***Protocol***	***Length***	***Time Since Request***	***Info***
192.168.137.184	192.168.137.1	*HTTP*	443	0.001754	*HTTP/1.1 200 OK (text/html)*
192.168.137.184	192.168.137.1	*HTTP*	439	0.00129	*HTTP/1.1 200 OK (text/html)*
192.168.137.184	192.168.137.1	*HTTP*	443	0.001799	*HTTP/1.1 200 OK (text/html)*
192.168.137.184	192.168.137.1	*HTTP*	439	0.001315	*HTTP/1.1 200 OK (text/html)*
192.168.137.184	192.168.137.1	*HTTP*	443	0.001589	*HTTP/1.1 200 OK (text/html)*
192.168.137.184	192.168.137.1	*HTTP*	*439*	0.001048	*HTTP/1.1 200 OK (text/html)*
192.168.137.184	192.168.137.1	*HTTP*	443	0.001196	*HTTP/1.1 200 OK (text/html)*

Dari data tabel hasil uji request sebanyak 7 kali diatas, didapatlah rata-rata **0.001628s** sebagai hasil dari pengujian,

4.4.2 *Throughput Utilization*

Pengujian *throughput* (*Throughput Utilization*) merupakan pengujian yang lakukan untuk mengukur kecepatan rata-rata data yang diterima oleh *node server* dalam jangka waktu pengamatan tertentu (*European Telecommunications Standards Institute* (ETSI), 2012). Untuk dapat mengukur nilai *throughput*, rumus yang digunakan adalah:

$$\boldsymbol{Throughput = \frac{Jumlah\ Data\ Yang\ Dikirim}{Waktu\ Penerimaan\ Data}}$$

Dalam pengujian ini wireshark dijalankan selama 2 menit 58 detik. hasil dari pengujian yang dilakukan oleh aplikasi wireshark dimana "*Time Span*" adalah waktu pengiriman data, dan "*Byte*" adalah satuan jumlah paket data yang dikirim. Total *time span* adalah 178,127 detik (*second*) dan *Byte* adalah sebesar 289333 Byte. Dengan menggunakan rumus *throughput* di atas, diperoleh nilai *throughput* sebesar 162.4290534Bps

4.4.3 Pembagian Beban Dari Server HAproxy

Skenario dari pengujian pembagian beban dari *server HAProxy* adalah dimana salah satu *node server* dari rangkaian *server load balancing* akan dimatikan. Dari proses tersebut akan dilihat apakah layanan dari *HAProxy* masih tetap berjalan. Dan dari hasil pengujian yang dilakukan adalah : service dari *HAProxy* masih tetap dapat berjalan dengan baik, *HAProxy* berjalan dengan cara menampilkan *node server* yang masih hidup.

4.4.4 Sinkronisasi Database

Pada pengujian sinkronisasi *database* ini, skenario yang akan digunakan adalah dengan mengubah (*update*) salah satu isi dari salah satu tabel yang ada. Pengujian dilakukan dengan mengubah isi dari tabel kurs dari *server* 1, dimana nilai dari "mata_uang" ringgit diubah yang semula 7400 menjadi 7500. Dari pegujian yang telah dilakukan, hasilnya adalah data tabel pada *server* 2 berubah mengikuti data yang telah di ubah pada tabel *server* 1.

4.5 Pengujian Kompatibilitas

Pengujian kompatibilitas adalah jenis pengujian perangkat lunak yang digunakan untuk memastikan kompatibilitas lingkungan sistem yang dibangun, seperti *browser compatibility*, *OS compatibility*, *database compatibility*. Pada penelitian ini kompabilitas pengujian yang dilakukan adalah *browser compability*. Skenario pengujian dilakukan dengan membuka halaman *web server* yang telah *load balancing* di dua *browser* yaitu Mozilla Firefox dan Google Chrome. Hasil dari pengujian tersebut adalah *web server load balancing* dapat berjalan pada kedua *browser* tersebut.

5. KESIMPULAN

Dari hasil tahapan-tahan penelitian serta serangkaian pengujian yg telah dilakukan, diperoleh beberapa kesimpulan sebagai hasil dari penelitian ini. Adapun kesimpulan dari penelitian rancang bangun *server HAProxy load balancing master to master* berbasis *cloud computing* ini adalah sebagai berikut:

1. *Server* dengan menggunakan sistem *load balancing* dapat menambah kehandalan dari segi ketersediaan layanan karena dapat menghubungkan banyak *server* sehingga jika salah satu *server* ada yang mati, layanan tersebut masih dapat diakses.
2. Nilai *Troughput* yang dihasilkan oleh *server load balancing* pada penelitian ini adalah sebesar 162.4290534Bps.
3. Rata-rata *response time* dari *server HAProxy* yang dihasilkan dari 7 kali uji request adalah 0.001628s, dengan nilai *response time* terlama adalah 0.001526s dan response time tercepat adalah 0.001873s.
5. 4. Sinkronisasi *database* yang berjalan menggunakan replikasi master to master dapat berjalan dengan baik. Dimana setiap perubahan yang terjadi pada salah satu *database* akan replikasikan pada *database* lainya. Hal ini dibuktikan dengan percobaan yang dilakukan pada bab sebelumnya dimana dari hasil pengujian, data yang ada pada *database node server* 2 mengikuti perubahan yang dilakukan pada *database node server* 1.

DAFTAR PUSTAKA

CISCO (2010). *An Introduction to the Cisco Lifecycle Services Approach*. Retrieved from https://www.cisco.com/web/partners/services/promos/accelerate/downloads/lifecycle_services_sg.pdf

ERNAWATI, T., & ZULFIAJI, A. H. (2013). Analisis dan Pembangunan Infrastruktur Cloud Computing, *1*(2), 17–23.

EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE (ETSI). (2012). ETSI Technical Committee Speech and multimedia Transmission Quality (STQ). In *ETSI EG 203 165 Speech and multimedia Transmission Quality (STQ); Throughput Measurement Guidelines* (Vol. 1, pp. 1–30).

IDCLOUDHOST.COM. (2016). Mengenal Virtual Private Server atau VPS. Retrieved February 5, 2019, from https://idcloudhost.com/mengenal-virtual-private-server-atau-vps/

LUKITASARI, D., & OKLILAS, A. F. (2010). Analisis Perbandingan Load Balancing Web Server Tunggal Dengan Web server Cluster Menggunakan Linux Virtual Server, *5*(2), 31–34.

MELL, P., & GRANCE, T. (2011). The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology. *Nist Special Publication*, *145*, 7. https://doi.org/10.1136/emj.2010.096966

MOLYNEAUX, I. (2014). The Art of Application Performance Testing From Strategy to Tools.

MSN, F., & RAHMATULLOH, A. (2017). Implementasi Load Balancing Web Server menggunakan Haproxy dan Sinkronisasi File pada Sistem Informasi Akademik Universitas Siliwangi, *2*, 241–248.

MULIYANTORO, H. S. (2013). Penerapan Metode Load-Balancing Clusters Pada Database Server Guna Peningkatan Kinerja Pengaksesan Data, *IX*(1), 97–108.

NASSER, H., & WITONO, T. (2016). ANALISIS ALGORITMA ROUND ROBIN , LEAST CONNECTION , DAN RATIO PADA LOAD BALANCNG, *12*(1), 25–32.

NUGROHO, A., YAHYA, W., & AMRON, K. (2017). Analisis Perbandingan Performa Algoritma Round Robin dan Least Connection untuk Load Balancing pada Software Defined Network, *1*(12), 1568–1577.

OKTAVIANUS, Y. L. (2013). Membangun Sistem Cloud Computing Dengan Implementasi Load Balancing Dan Pengujian Algoritma Penjadwalan Linux Virtual Server Pada, (1), 25–30.

ORACLE. (2019a). General Information. Retrieved February 4, 2019, from https://dev.mysql.com/doc/refman/8.0/en/introduction.html

ORACLE. (2019b). MySQL. Retrieved April 20, 2019, from https://dev.mysql.com/doc/refman/5.7/en/replication.html

PURBO, O. W. (2011). Petunjuk Praktis Cloud Computing Mengunakan Open Source, 1–48.

RABUR, J. A., PURWADI, J., & RAHARJO, W. S. (2012). Implementasi Load Balancing Web Server Menggunakan Metode LVS-NAT, *8*(2).

RACHMAWAN, D., IRWAN, D., & ARGYAWATI, H. (2016). Penerapan Teknik Load Balancing Pada Web Server Lokal Dengan Metode Nth Menggunakan, *4*(2), 98–108.

SAM, J. (2009). No Title. Retrieved from https://commons.wikimedia.org/wiki/File:Cloud_computing.svg#/media/File:Cloud_computing.s vg

SAMAL, P., & MISHRA, P. (2013). Analysis of variants in Round Robin Algorithms for load balancing in Cloud Computing, *4*(3), 416–419.

SONY, T., ANDONO., & PRATAMA, P. N. (2013). Visualisasi Pembelajaran Algoritma Round Robin Pada Load Balancing. Retrieved from http://eprints.dinus.ac.id/id/eprint/4802

SUMARNO, E., & HASMORO, H. P. (2011). Implementasi Metode Load Balancing Dengan Dua Jalur (Study Kasus Jaringan Internet Smp Negeri 2 Karanganyar), 28–34.

SYAPUTRA, A. W., & ASSEGAFF, S. (2017). Analisis Dan Implementasi Load Balancing Dengan Metode Nth Pada Jaringan Dinas Pendidikan Provinsi Jambi, *2*(4).

YEAGER, N. J., & MCGRATH, R. E. (1996). *Web Server Technology*. Morgan Kaufmann.