PERFORMA ALGORITMA *LOAD BALANCE* PADA SERVER WEB APACHE DAN NGINX DENGAN DATABASE POSTGRESQL

SKRIPSI

Diajukan kepada Fakultas Teknik
Universitas Negeri Yogyakarta
Untuk Memenuhi Sebagian Persyaratan
Guna Memperoleh Gelar Sarjana Pendidikan Teknik

Disusun Oleh:
Omar Muhammad Altoumi Alsyaibani
NIM. 09520244085

PROGRAM STUDI PENDIDIKAN TEKNIK INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS NEGERI YOGYAKARTA
2013

PERSETUJUAN

Skripsi yang berjudul:

PERFORMA ALGORITMA *LOAD BALANCE* PADA SERVER WEB APACHE DAN NGINX DENGAN DATABASE POSTGRESQL

Oleh:

OMAR MUHAMMAD ALTOUMI ALSYAIBANI

NIM. 09520244085

Telah diperiksa dan disetujui oleh pembimbing untuk diujikan

Yogyakarta, 20 Agustus 2013

Menyetujui,
Dosen Pembimbing

Drs. Totok Sukardiyono, M. T.
NIP. 19670930 199303 1 005

PERNYATAAN

Saya yang bertanda tangan di bawah ini:

Nama	: Omar Muhammad Altoumi Alsyaibani
NIM	: 09520244085
Program Studi	: Pendidikan Teknik Informatika
Jurusan	: Pendidikan Teknik Elektronika
Judul Penelitian	: Performa Algoritma *Load Balance* Pada Server Web Apache Dan Nginx Dengan Database Postgresql

Dengan ini saya menyatakan bahwa dalam Tugas Akhir Skripsi ini tidak terdapat karya yang pernah diajukan untuk memperoleh gelar Sarjana Pendidikan atau gelar lainnya di suatu Perguruan Tinggi dan sepengetahuan saya juga tidak terdapat karya atau pendapat yang pernah ditulis oleh orang lain, kecuali bagian-bagian tertentu yang saya ambil sebagai acuan dengan mengikuti kaidah karya tulis ilmiah yang benar.

Demikian pernyataan ini dibuat dalam keadaan sadar dan tidak dipaksakan untuk digunakan sebagaimana mestinya.

Yogyakarta, 20 Agustus 2013

Yang menyatakan,

Omar Muhammad A. A.

NIM. 09520244085

PENGESAHAN

Skripsi yang berjudul "Performa Algoritma *Load Balance* Pada Server Web Apache Dan Nginx Dengan Database Postgresql" yang disusun oleh Omar Muhammad Altoumi Alsyaibani, NIM 09520244085 ini telah dipertahankan di depan Dewan Penguji pada tanggal 30 Agustus 2013 dan dinyatakan lulus.

DEWAN PENGUJI

Nama	Jabatan	Tanda Tangan	Tanggal
Drs. Totok Sukardiyono, M.T.	Ketua Penguji	.....	24-9-2013
Dra. Umi Rochayati, M. T.	Sekretaris Penguji	.....	24-9-2013
Drs. Achmad Fatch, M. Pd.	Penguji Utama	.....	11-9-2013

Yogyakarta, 25 September 2013

Dekan Fakultas Teknik

Dr. Moch. Bruri Triyono
NIP. 19560216 198603 1 003

MOTTO

Manusia dan Waktu itu saling berkejaran (Indonesia)

Kukui witang ada witus, surung jawu jangan pagat (Dayak Maanyan)

Kambang kada sakaki, kumbang kada saikung, alam kada batawing (Banjar)

A candle loses nothing by lighting another candle (Jan Groft)

Sugih tanpa bandha, digdaya tanpa aji, nglurug tanpa bala, menang tanpa ngasorake (Jawa)

Man jadda wajada (Arab)

BELILAH MASA DEPAN DENGAN HARGA SEKARANG

PERSEMBAHAN

Segala puji bagi Allah SWT. Tuhan semesta Alam. Dengan izin-NYa jualah skripsi dapat terselesaikan. Laporan tugas akhir skripsi ini saya persembahkan kepada :

- Kedua Orang Tuaku (Drs. Saimi dan Hj. Sufiati Zaenah) tercinta, terima kasih akan kasih sayangnya, dukungan dan doanya disetiap waktu.
- Semua Dosen UNY terutama jurusan Pendidikan Teknik Elektronika yang sudah banyak membimbing saya. Saat saya pertama datang ke Yogyakarta, saya tidak mengerti apapun. Terima kasih atas pengajarannya yang arif.
- Keluarga besar Limuny yang telah memberi banyak pelajaran berharga dalam waktu yang panjang. Terima kasih juga buat teman-teman Limuny yang seperjuangan saat mengerjakan skripsi atas semangat yang kalian bagikan.
- Keluarga besar HIMANIKA yang sudah menanamkan pondasi mental yang luar biasa di kepalaku. Kita ada karena Kontribusi!
- Teman-teman kelas Gembel yang telah memaparkan keragaman luar biasa bagiku di awal kuliah. Banyak hal yang kupelajari dari kalian teman.
- Rekan-rekan UKM Rekayasa Teknologi terutama Divisi IT. Teruslah tumbuh! Waktu berkembangmu masih panjang.
- Saudara-saudaraku yang ada di kos Pak Bambang. Meski kos kita memang berantakan, tapi berada di tengah-tengah kalian membuat penat dan letihku hilang.

PERFORMA ALGORITMA *LOAD BALANCE* PADA SERVER WEB APACHE DAN NGINX DENGAN DATABASE POSTGRESQL

Oleh:
Omar Muhammad Altoumi Alsyaibani
NIM. 09520244085

ABSTRAK

Penelitian ini bertujuan untuk mengetahui kinerja algoritma *load balance*. Algoritma ini digunakan untuk membagi kerja server *load balance* ke beberapa server web yang menggunakan Apache2 dan Nginx. Database server yang digunakan adalah Postgresql. Sistem operasi yang digunakan adalah Ubuntu 12.4 LTS.

Penelitian ini menggunakan pendekatan penelitian deskriptif. Dalam penelitian ini dikumpulkan data *throughput, response time, request,* dan *reply* pada masing-masing skenario dengan 10 kali perulangan untuk setiap algoritma. Ada 6 algoritma yang dibandingkan, yaitu *Round Robin, Least Connection, Shortest Expected Delay, Never Queue, Weighted Round Robin, dan Weighted Least Connection*. Pengambilan data dilakukan dengan dua skenario. Pada masing-masing skenario digunakan server web Apache dan Nginx. Pengambilan data dilakukan dengan menggunakan perangkat lunak Httperf dan dibantu dengan *script* tambahan. Terdapat sembilan komputer yang digunakan untuk percobaan algoritma *load balance*, dengan rincinan: tiga komputer untuk server web, tiga komputer untuk server database, komputer tester, *load balancer web* dan server replikasi database. Khusus untuk tiga server yang berfungsi sebagai server web dikondisikan mempunyai spesifikasi fisik yang berbeda.

Hasil penelitian ini menunjukkan bahwa keluaran yang dihasilkan oleh algoritma-algoritma *load balance* yang diimpelementasikan untuk server web Nginx hampir sama pada semua aspek. Sedangkan saat diimplementasikan untuk server web Apache, algoritma-algoritma *load balance* ini menghasilkan keluaran yang bervariasi. Rata-rata throughput tertinggi dihasilkan oleh algoritma SED, WRR, dan WLC serta throughput terkecil dihasilkan oleh algoritma RR. Response time yang terkecil dihasilkan oleh algoritma NQ dan yang terbesar dihasilkan oleh algoritma RR. Request dan reply tertinggi dihasilkan oleh algoritma WLC dan WRR.

Kata kunci: *load balance,* algoritma, server web.

KATA PENGANTAR

Alhamdulillaahirrabbilaalaamiin. Dengan izin Tuhan semesta alamlah maka laporan tugas akhir skripsi ini dapat terselesaikan. Selain untuk memenuhi syarat untuk mencapai gelar sarjana pendidikan pada Pendidikan Teknik Informatika di Universitas Negeri Yogyakarta, penulis juga berharap tugas akhir skripsi ini juga dapat memberikan sumbangan kecil dalam hal keilmuan dan penelitian yang topiknya berkaitan di Universitas Negeri Yogyakarta.

Banyak pihak yang telah membantu saya dalam menyelesaikan skripsi ini, baik secara langsung maupun tidak langsung. Oleh karena itu, penulis ingin mengucapkan terima kasih yang sebesar-besarnya kepada:

1. Bapak Prof. Dr. H. Rochmat Wahab, M.Pd., M.A. selaku Rektor Universitas Negeri Yogyakarta.
2. Bapak Dr. Moch. Bruri Triyono selaku Dekan Fakultas Teknik Universitas Negeri Yogyakarta.
3. Bapak Muhammad Munir, M.Pd. selaku Ketua Jurusan Pendidikan Teknik Elektronika.
4. Bapak Adi Dewanto, M. Kom. selaku dosen Pembimbing Akademik.
5. Bapak Handaru Jati, Ph.D. selaku Koordinator Tugas Akhir Skripsi.
6. Bapak Totok Sukardiyono, M. T. selaku dosen pembimbing tugas akhir skripsi yang telah memberikan bimbingan, arahan, dan dukungan dengan sabar dalam penyusunan skripsi ini.
7. Mas Nandar selaku Koordinator Pelaksana Limuny yang telah memberikan izin pemakaian komputer dan tempat untuk penelitian ini.
8. Semua pihak yang telah membantu penulis selama pembuatan skripsi ini.

Penulis menyadari bahwa skripsi ini masih jauh dari sempurna. Oleh karena itu, penulis mengharapkan kritik dan saran yang sifatnya membangun. Akhir kata penulis berharap skripsi ini bermanfaat bagi kita untuk memperkarya ilmu dan pengetahuan di masa sekarang dan masa mendatang.

Yogyakarta, 20 Agustus 2013

Penulis

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

BAB I
PENDAHULUAN

A. Latar Belakang

Internet saat ini sudah menjadi kebutuhan yang sangat penting bagi sebagian orang, terutama masyarakat perkotaan yang sudah melek teknologi. Setiap hari orang akan mengakses internet dalam bentuk apapun. Menurut statistik internet dunia (Juni 2012), Indonesia menduduki posisi pengakses internet terbesar kedelapan di dunia dengan perkiraan jumlah pengguna mencapai 55 juta pada tahun 2012. Hal ini sudah mulai disadari oleh berbagai pihak, salah satunya adalah pemerintah. Sebagai contoh, pemerintah Jakarta seperti yang sudah disorot oleh media mulai melakukan instalasi wifi di sepanjang jalan utama pusat kota. Instalasi tersebut dilakukan dengan tujuan untuk memenuhi kebutuhan masyarakat akan akses informasi yang cepat.

Melihat tingkat aktivitas dari pengguna yang begitu tinggi, tentu saja hal ini akan berdampak pada penyedia informasi. Kinerja server web dan database sebagai media penyedia konten selalu diharapkan dapat memenuhi semua kebutuhan dari pengguna. Jika tidak ditanggapi dengan serius, ini bisa saja berakibat pada server-server yang kelebihan beban permintaan (*request)* dari pengguna. Hal ini disebabkan permintaan dari pengguna lebih besar dari kemampuan server untuk memberikan layanan. Dampak ini tentu tidak diinginkan oleh beberapa instansi yang semua aktivitasnya sudah ketergantungan dengan jaringan komputer. Oleh karena itu, instansi-instansi tersebut tidak ragu lagi untuk mengalokasikan dananya untuk membeli perangkat server khusus dengan kemampuan yang tinggi. Sayangnya, setiap hari permintaan layanan dari pengguna selalu meningkat. Hal ini tentu saja berhubungan dengan semakin banyaknya perangkat-perangkat yang dapat menggunakan fasilitas internet seperti komputer, laptop, netbook, smartphone, tablet, dan perangkat lainnya.

Untuk mengatasi permasalahan di atas, pada tahun 1998 dimulailah sebuah *project* yang bertujuan untuk meningkatkan performasi layanan server menggunakan teknologi *clustering*. Salah satu jenis perangkat lunak (*software*) yang paling banyak digunakan untuk layanan server web adalah *Linux Virtual Server (LVS)* yang menerapkan metodologi *load balance* untuk membagi pekerjaan kepada beberapa komputer. *Software* ini terus dikembangkan dengan berbagai algoritma pembagian pekerjaan. Pembuatan berbagai algoritma ini tentu saja mempunyai tujuan dan karakteristik yang berbeda-beda pula.

Faktanya, server-server web yang digunakan oleh sebuah instansi tentu saja tidak selalu memiliki spesifikasi yang sama. *Load balancing server* sebagai pembagi kerja bagi server-server tersebut tentu saja harus dapat membagi pekerjaan ini sesuai dengan kemampuan masing-masing server agar dapat melayani pengguna secara optimal. Oleh karena itu, perlu dicari algoritma yang paling tepat untuk kasus server *load balance* di atas.

Di sisi lain, server database mengalami permasalahan yang sedikit berbeda. Secara umum kegiatan database server meliputi SELECT, INSERT, UPDATE, dan DELETE. Kegiatan SELECT pada banyak server database dapat ditangani oleh server *load balance,* tetapi untuk kegiatan lain diperlukan fasilitas replikasi agar data pada semua server database tetap sama.

Saat ini terdapat banyak sekali *engine* server web dan database. Diantara *engine* server web yang paling banyak digunakan adalah Apache dan Nginx karena secara default performa kedua *engine* web server ini sangat handal. Kemudian *engine* database server yang paling banyak digunakan adalah Mysql, Oracle, dan Postgresql. Pada penelitian ini, penulis hanya menggunakan Postgresql sebagai database karena untuk implementasi replikasi dan *load balance* nya cukup sederhana.

Dari beberapa permasalahan di atas, maka ada beberapa hal yang ingin diteliti oleh penulis, yaitu mengenai kinerja *engine* untuk server web,

mengenai *load balance* dan replikasi, serta penggunaan algoritma *load balance* pada server web.

B. Identifikasi Masalah

Dari fakta-fakta pada latar belakang, dapat diidentifikasi beberapa masalah, diantaranya:

1. Pengguna internet meningkat seiring dengan jumlah *device* yang terus meningkat, sehingga kinerja server dituntut meningkat pula.
2. Server yang mempunyai performa tinggi mempunyai harga yang tinggi, sehingga perlu dicari solusi membuat server dengan layanan yang tinggi namun dengan biaya yang minim.
3. Spesifikasi server-server web yang digunakan dalam pembuatan *cluster* tidak selalu sama yang tentu saja mempunyai tingkat kinerja yang berbeda-beda pula, sehingga perlu dicari algoritma yang cocok untuk membagikan pekerjaan dengan seimbang pada keadaan cluster yang seperti ini.
4. Untuk mensinkronkan beberapa server database, diperlukan server khusus yang bertugas untuk mereplikasi data.

C. Batasan Masalah

Penelitian ini akan dibatasi pada beberapa hal, diantaranya:

1. Perangkat lunak server web yang akan digunakan adalah Apache dan Nginx. Kedua perangkat lunak ini dipilih karena keduanya merupakan perangkat lunak server web yang paling banyak digunakan untuk melayani berbagai macam aplikasi, dari yang sederhana hingga yang *enterprise*. Selain itu, keduanya bersifat gratis dan *opensource*.
2. Perangkat lunak untuk server database yang akan digunakan adalah Postgresql. Postgresql merupakan perangkat lunak database yang ditujukan untuk aplikasi *enterprise* dan bersifat gratis serta *opensource*. Selain itu, dalam penelitian ini, telah ditemukan cara untuk *load balance* dan replikasi beberapa server Postgesql.

3. Sistem operasi yang akan digunakan pada penelitian ini adalah Ubuntu 12.4 LTS Server dan Desktop. Sistem operasi ini dipilih karena penggunaannya sederhana, dan banyak dokumentasinya di internet sehingga akan mempermudah proses penelitian. Selain itu, Ubuntu merupakan sistem operasi Linux dengan pengguna dan komunitas terbesar di dunia.

Terdapat sembilan komputer yang akan digunakan dalam penelitian ini. Tiga komputer diantaranya berfungsi sebagai server web, tiga komputer berfungsi sebagai server database yang databasenya akan disinkronkan, satu komputer sebagai *load balancing server*, satu komputer berfungsi sebagai server replikasi database dan satu komputer lagi digunakan untuk menguji kinerja server (*tester*). Penelitian ini akan berfokus pada kinerja algoritma *load balance,* baik pada server web Nginx maupun Apache. Pada teknik *load balance* terdapat banyak algoritma yang dapat digunakan, tetapi pada penelitian ini penulis batasi hanya menggunakan 6 algoritma saja yaitu: *Round Robin*, *Least Connection*, *Shortest Expected Delay*, *Never Queue*, *Weighted Round Robin*, dan *Weighted Least Connection*.

D. Rumusan Masalah

Bagaimanakah kinerja algoritma-algoritma *load balance* yang diimplementasikan pada server web Apache2 dan Nginx dengan database Postgresql?

E. Tujuan

Mengetahui kinerja algoritma-algoritma *load balance* yang diimplementasikan pada server web Apache2 dan Nginx dengan database Postgresql.

F. Manfaat

1. Manfaat praktis
   a. Bagi instansi/lembaga, penelitian ini dapat menjadi referensi dalam mengembangkan server *load balance* dan replikasi serta dalam pemilihan *engine* server web dan database pada instansi masing-masing.
   b. Bagi peneliti, penelitian ini dapat meningkatkan pengetahuan penulis dalam bidang server *load balance* dan replikasi, *engine* server web dan database.
2. Manfaat teoritis

   Hasil penelitian ini dapat menjadi referensi bagi penelitian selanjutnya yang topiknya berkaitan.

BAB II
KAJIAN PUSTAKA

A. Dasar Teori

1. Cluster Computer

Dalam *Longman Dictionary of Contemporary English* edisi kelima, *cluster* didefinisikan sebagai sebuah grup atau kumpulan orang atau benda yang berjenis sama dan sangat berdekatan. Kemudian pada kamus Merriam Webster *cluster* didefinisikan menjadi sejumlah benda yang mirip dan terjadi bersamaan.

Thomas Sterling (2001:14) dalam bukunya *Beowulf Cluster Computer with Linux* menyatakan bahwa *clustering* adalah konsep dan teknik yang *powerful* untuk mendapatkan kapabilitas yang lebih besar dari sekumpulan (*class*) komponen atau benda. *Cluster* merupakan mekanisme yang mendasar untuk membuat kompleksitas dan keberagaman melalui pengumpulan dan penggabungan dari elemen-elemen dasar yang sederhana.

Jadi, *cluster* dalam konteks komputer adalah sekumpulan komputer yang identik dan bekerja bersama untuk mendapatkan kapabilitas yang besar dengan cara menggabungkan atau mengkombinasikan komputer-komputer yang sederhana.

Di bidang sistem komputasi, *clustering* digunakan untuk membuat struktur sistem baru dari elemen komputasi yang ada untuk menghasilkan kapabilitas komputasi yang 10 kali lipat lebih murah dari pendekatan lain (Thomas Sterling, 2001: 14). Kemudian Thomas juga menyebutkan bahwa beberapa komputer terbesar di dunia adalah sistem *cluster*. Sistem *cluster* dapat melakukan pekerjaan dengan skala medium, namun dengan biaya yang rendah karena menggunakan teknologi yang berbasis PC.

Sistem ini menjadi populer karena menghasilkan harga yang murah dan kinerja yang tinggi, fleksibilitas konfigurasi dan *upgrade*, kemampuan untuk mengembangkan *tool-tool* yang baik, serta membuka kesempatan yang besar untuk pengembangan aplikasi komputasi baru.

Mark Barker dan Rajkumar Buyya dalam *paper* mereka yang berjudul *Cluster Computing at a Glance* mengungkapkan bahwa *Cluster Computing* termasuk dalam kategori *Scalable Parallel Computer Architecture*. Selain *cluster*, sistem lain yang mereka sebutkan adalah *Massively Parallel Processors (MPP)*, *Symmetric Multiprocessors (SM), Cache-Coherent Nonuniform Memory Access (CC-NUMA), dan Distributed System*. Kemudian mereka membandingkan karakteristik masing-masing sistem. Perbandingan karakteristik ini dapat dilihat pada tabel di bawah ini:

Tabel 1. Perbandingan Karateristik Sistem Arsitektur

Characteristics	**MPP**	**SMP CC-NUMA**	***Cluster***	***Distributed***
Number of Nodes	100-1000	10-100	100 or less	10-1000
Node Complexity	*Fine grain or medium*	*Medium or craise grained*	*Medium grained*	*Wide range*
Internode Communication	*Message passing/shared variables for distributed shared memory*	*Centralized and distributed shared memory (DSM)*	*Message passing*	*Shared files, RPC, message passing and IPC*
Job Schedulling	*Single run queue on*	*Single run queue mostly*	*Multiple queue but*	*Independent queue*

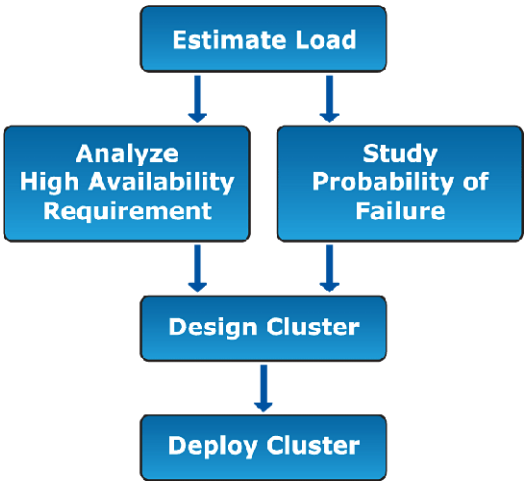
	host		*coordinate*	
SSI Support	*Partially*	*Always in SMP and some NUMA*	*Desired*	*No*
Node OS copies and type	*N micro-kernels monolithic or layered Oss*	*One monolithic SMP and many for NUMA*	*N OS platform – homogeneous or micro-kernel*	*N OS platform homogeneous*
Address Space	*Multiple-single for DSM*	*Single*	*Multiple or single*	*Multiple*
Internode security	*Unnecessary*	*Unnecessary*	*Required if exposed*	*Required*
Ownership	*One organization*	*One organization*	*One or more organizations*	*Many organizations*

Mark B. dan Rajkumar B. menyebutkan empat fitur yang diberikan oleh sistem *cluster*, diantaranya: *High Performance, Expandability and Scalability, High Throughput, and High Availability*. Hal ini juga sejalan dengan yang diterangkan oleh tim Savvion Bussiness Manager 6.5 (2006:10) dalam buku *Clustering Guide* bahwa tujuan dibuatnya *clustering* adalah:

a. Menghasilkan ketersediaan (sistem) yang tinggi (*High-Availability*) dengan cara menyeimbangkan permintaan sumberdaya (*resources*).
b. Memastikan kehandalan sistem melalui pengurangan kegagalan sistem (*system failure*) dengan mengelola *failover*.
c. Memberikan skalabilitas dengan membebaskan penambahan sumber daya secara dinamis untuk meningkatkan kinerja sistem.

Kemudian tim Savvion menerangkan juga bahwa untuk merancang sebuah sistem *cluster*, ada beberapa hal yang kiranya menjadi pertimbangan, diantaranya:

a. Banyaknya kebutuhan *high availability*.
b. Pengguna dan besarnya transaksi yang dilakukan oleh pengguna.
c. Rancangan dan kapasitas jaringan.
d. Rancangan dan ketergantungan database.
e. Pengelolaan sistem.



Gambar 1. *Cluster Design Process* (*Clustering Guide*, 2006:12)

Richard S. Morrison (2003:27) dalam tulisannya yang berjudul *Cluster Computing: Architectures, Operating Systems, Parallel Processing & Programming Languages* selain juga menjelaskan mengenai kelebihan *cluster computing*, dia juga menyebutkan beberapa kelemahan dari *cluster computing*, diantaranya:

a. Terbatasnya perangkat lunak yang mendukung sistem *cluster*.
b. Kemungkinan terganggunya komunikasi di jaringan yang disebabkan oleh ketidakreliabilitasan jaringan itu sendiri.
c. Timbulnya permasalahan keamanan.

Secara umum terdapat dua bagian utama pada sistem *cluster* yaitu komputer yang berperan sebagai *cluster manager* atau *master* dan

komputer lainnya yang berperan sebagai *cluster node* atau *slave*. *Master* bertugas untuk mendistribusikan pekerjaan kepada komputer lainnya yang dikonfigurasikan sebagai *slave* tergantung pada jenis *clustering* yang digunakan. (Fachrurrozi. as, 2012:31).

Kemudian Fachrurrozi juga menguraikan beberapa jenis *cluster* yang sering digunakan, diantaranya:

a. *Shared Processing*

Kemampuan komputasi satu komputer digabungkan dengan sejumlah komputer lainnya untuk memperoleh sistem dengan kemampuan komputasi yang lebih besar. Biasanya sistem ini digunakan pada superkomputer seperti *Beowulf clustering*.

b. *Load Balancing*

Sistem ini terdiri dari dua bagian yaitu *load balancer (cluster manager)* dan *cluster node (backend servers). Request* dari *client* diterima oleh komputer yang menjadi *load balancer*. Kemudian *load balancer* meneruskan request tersebut ke salah satu *cluster node* untuk diproses oleh *cluster node*. Lalu *cluster node* memberikan balasan bahwa request telah diproses kepada *cluster manager*.

c. *Fail-Over*

Sistem ini cara kerjanya hampir mirip dengan *load balancing*, hanya saja manager tidak mendistribusikan pekerjaan ke seluruh *cluster node* melainkan hanya ke suatu *node* tertentu. Jika *node* tersebut down atau tidak mampu memproses pekerjaan yang dibebankan manager, maka *node* lainnya akan mengambil alih pekerjaan *node* tersebut.

d. *High availability*

Dalam metode *high availability*, seluruh fasilitas komputasi yang digunakan sistem harus dijaga agar tetap bekerja secara kontinyu dan ketika dibutuhkan *maintenance*, ada komputer lain yang bisa mengambil alih proses komputasi. Teknik *load balancing* dan *failover*

sebenarnya merupakan teknik yang dapat mendukung sistem untuk mencapai ketersediaan sistem tinggi.

2. *Linux Virtual Server*

Pada situs resminya Linux Virtual Server (LVS) dideskripsikan sebagai *highly scalable server* dan *highly available server*, dibangun menggunakan *cluster* beberapa *real server* dengan menjalankan *load balance software* pada *load balancing server*. Arsitektur *cluster server* bersifat transparan terhadap pengguna, sehingga pengguna seakan-akan berinteraksi dengan server yang mempunyai performa tinggi.

Sekarang, kebanyakan kerja dari *LVS project* adalah mengembangkan IP load balancing (IPVS), software *load balancing* yang berjalan pada level aplikasi (KTCPVS), dan komponen-komponen *cluster management.*

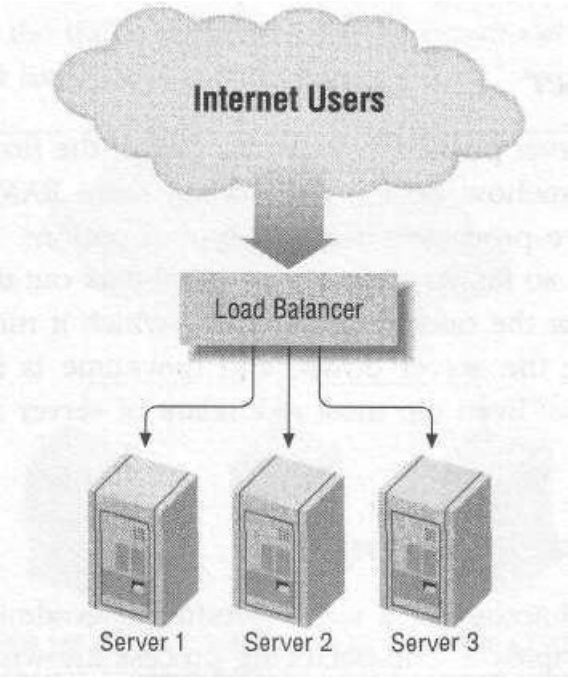
3. *Load Balancing Server*

Seperti yang sudah diuraikan oleh Fachrurrozi bahwa *Load Balance* merupakan salah satu jenis atau fitur yang dikembangkan dari arsitektur *cluster*. Sistem *load balance* bertujuan untuk menyeimbangkan pembagian kerja di antara *node-node* yang ada di dalam sebuah *cluster*.

Hal di atas tidak jauh berbeda dari yang dijelaskan oleh Tony Bourke (2001), *load balance* merupakan sebuah proses dan teknologi yang menyalurkan lalu lintas (*traffic*) diantara beberapa server menggunakan perangkat berbasis jaringan. Perangkat ini (*load balancing server*) menahan/menangkap *traffic* yang bertujuan kepada sebuah alamat kemudian me-*redirect traffic* tersebut kepada banyak server. Proses *load balance* ini bersifat transparan terhadap pengguna yang melakukan *request* ke server *load balance*. Ada puluhan bahkan ratusan server yang beroperasi dibalik sebuah alamat.

Kemudian Tony Bourke menjelaskan bahwa terdapat lima fungsi dari sebuah *load balancer*, yaitu:

a. Mencegat *traffic* berbasis jaringan (*network-based traffic*) seperti *web traffic* yang ditujukan pada sebuah alamat.
b. Membagi *traffic* menjadi satuan *request* dan memutuskan server mana yang akan menerima *request* tersebut.
c. *Monitoring server*, memastikan bahwa server-server tersebut dapat merespon *traffic.* Jika salah satu server tidak dapat merespon permintaan, maka ia akan dikeluarkan dari daftar *node*.
d. Menyediakan *redundancy* (server ganda) dengan menggunakan lebih dari satu skenario *fail-over*.
e. Memberikan perhatian/pengawasan pada distribusi konten (di dalam *traffic* jaringan), seperti membaca URL, mencegat *cookies*, dan *XML parsing*.



Gambar 2. Load Balancing Server yang sederhana (Tony Bourke, 2001:3)

Dari evolusinya, *load balancing* dapat dikatakan sebagai salah satu hasil evolusi dari *clustering*. Pada *clustering* terdapat integrasi yang

kuat antara server-server, secara otomatis perangkat lunak khususpun menjadi diperlukan. Di sisi lain, *support* yang diberikan oleh beberapa vendor sistem operasi hanya mendukung beberapa *platform*. Kemudian protokol yang didukung pun juga terbatas. Berbeda dengan *load balancing server*, yang bersifat netral terhadap semua sistem operasi dan dapat bekerja selama ada jaringan komputer. Jadi, jika Anda mempunyai sekumpulan server dengan OS yang berbeda-beda, *server load balancing* dapat menyeimbangkan jumlah *request* diantara server-server tersebut. Di samping itu, *load balancing server* juga mendukung berbagai protokol jaringan, bahkan hampir semua protokol yang berbasis TCP atau UDP. Hal-hal inilah yang membuat *load balancing server* menjadi lebih fleksibel dan mempunyai rancangan teknologi yang lebih sederhana, tidak ada interaksi di antara *node-node* dan penggambaran fungsi yang jelas. Rancangan *load balancing server* sangat sederhana dan luwes sebagaimana fungsionalnya.

4. Metode *Load Balance*

Menurut Simon Horman dkk (2004:3) dalam *paper*-nya yang berjudul *Linux Virtual Server Tutorial*, terdapat tiga metode dalam mengimplementasikan *load balancing server*.

a. *Network Address Translation (NAT)*

Metode ini memanipulasi sumber dan atau tujuan serta port dan atau alamat dari sebuah paket. Teknik *forwarding* yang paling umum digunakan adalah IP masquerading yang mengaktifkan jaringan *private* untuk mengakses Internet. Paket yang berasal dari pengguna diubah alamat asal dan atau portnya pada *load balancing server* untuk dikirimkan kepada web server. Paket kembalian dari server web yang melewati *load balancing server* diubah lagi IP Addressnya sehingga seakan-akan paket dikembalikan dari *load balancing server.*

b. *Direct Routing*

Pada metode ini paket balasan dari *server web* dikirimkan langsung kepada pengguna. Alamat asal pada paket tidak diubah. Paket

dikirimkan tidak melalui *load balancing server*, sehingga alamat pada paket tidak perlu diubah.

c. *IP Encapsulation (Tunneling)*

Metode yang digunakan mirip dengan metode pada *direct routing*, kecuali ketika paket di arahkan, paket dienkapsulasi dalam sebuah alamat. Keuntungan metode ini adalah server web dapat berada pada jaringan yang berbeda dengan *load balancing server*.

Dari ketiga metode di atas, metode pertama merupakan metode yang paling mudah diimplementasikan. Selain itu, resiko mengalami masalah karena perubahan IP Address yang berada pada paket kecil, seperti masalah otentikasi. Kelemahannya adalah, server *load balance* akan mendapat yang sangat berat karena semua paket harus melewati server terlebih dahulu. Dalam penelitian ini, akan digunakan metode pertama.

5. Algoritma *Load Balance*

Deskripsi masing-masing algoritma *load balance* di bawah ini didasarkan pada situs resmi linux virtual server (1998) dan penjelasan dari Peter Membrey dkk (2012:160-162).

a. *Round Robin* (RR)

Round Robin merupakan salah satu algoritma penjadwalan proses paling sederhana pada sistem operasi. Algoritma penjadwalan *Round Robin* mengirimkan setiap permintaan yang masuk ke server setelahnya di dalam daftar tanpa prioritas (dikenal dengan istilah *cyclic executive*). Jadi dalam tiga cluster server (server A, B dan C) permintaan 1 akan diberikan ke server A, permintaan 2 akan diberikan ke server B, permintaan 3 akan diberikan ke server C, dan permintaan 4 akan diberikan ke server A lagi. Algoritma *Round Robin* memperlakukan semua server sama terlepas dari jumlah koneksi masuk atau waktu respon setiap server. Algoritma ini sangat sederhana dan mudah untuk diimplementasikan pada berbagai bidang. Algoritma ini pun diadaptasi pada penjadwalan LVS, khususnya

perangkat lunak Ipvsadm. Pada perangkat lunak Ipvsadm algoritma RR merupakan algoritma *default.*

Kelebihan: mudah dimengerti dan bekerja dengan efektif.

Kekurangan: tidak mempertimbangkan *server load*, sangat mungkin salah satu server akan *overload* karena tingginya jumlah *request* dan rendahnya kapasitas pemrosesan, sedangkan server yang mempunyai kapasitas lebih besar tidak melakukan apapun.

b. *Weighted Round Robin* (WRR)

Penjadwalan *Weighted Round Robin* dirancang untuk penanganan yang lebih baik untuk server-server yang berbeda kapasitas pemrosesannya. Setiap server dapat diberi bobot berupa integer yang mengindikasikan kapasitas pemrosesannya. Server yang mempunyai bobot lebih tinggi akan menerima koneksi lebih dulu dari pada server yang mempunyai bobot lebih rendah dan server yang mempunyai bobot lebih akan mendapatkan permintaan koneksi lebih banyak dibandingkan dengan server yang mempunyai bobot lebih kecil.

Contoh, terdapat tiga server yaitu A, B, dan C yang masing-masing mempunyai bobot 4, 3, dan 2. Maka urutan penjadwalan yang baik untuk kasus ini adalah AABABCABC. Dalam implementasi penjadwalan weighted round robin, urutan penjadwalan akan dihasilkan berdasarkan bobot server setelah konfigurasi virtual server dibuat. Koneksi jaringan akan diteruskan ke server-server yang berbeda berdasarkan urutan penjadwalan yang digunakan *Round Robin*. Singkatnya, penjadwalan round robin adalah penjadwalan *Weighted Round Robin* yang menganggap semua bobot server sama.

Penjadwalan *Weighted Round Robin* lebih baik dari penjadwalan round robin ketika kapasitas pemrosesan masing-masing server node berbeda. Hal ini akan mengarah pada ketidakseimbangan *load* yang dinamis diantara server-server jika *request* yang datang sangat bervariasi. Singkatnya, ada kemungkinan sebagian besar

request membutuhkan *response* yang besar akan diarahkan pada *server node* yang sama (yang bobotnya dibuat paling besar).

Kelebihan: menggunakan metode yang sama sederhananya dengan *Round Robin* dengan kemampuan mengarahkan *load* lebih banyak ke server tertentu.

Kekurangan: yang menentukan bobot masing-masing server adalah admin, dimana hal ini harus dilakukan secara manual.

Algoritma ini sangat baik digunakan jika server menangani kompleks *request* yang sama dan administrator mengetahui kapasitas masing-masing server sehingga dapat menetapkan bobot yang tepat untuk masing-masing server. Jika administrator salah menentukan bobot, maka akan mengakibatkan ketidakseimbangan pembagian kerja. Semakin baik administrator menentukan bobot masing-masing server, semakin baik kinerja yang dihasilkan oleh algoritma ini.

c. *Least Connection* (LC)

Algoritma penjadwalan *Least Connection* mengarahkan koneksi dari jaringan ke server yang mempunyai jumlah koneksi aktif paling sedikit. Ini merupakan salah satu algoritma penjadwalan yang dinamis karena ia harus menghitung jumlah koneksi yang aktif pada setiap server secara dinamis. Untuk virtual server yang menangani banyak server dengan performa yang mirip, penjadwalan *Least Connection* cukup baik untuk memperhalus distribusi koneksi saat *request* sangat beragam. Virtual server akan mengarahkan *request* ke server yang mempunyai koneksi aktif paling sedikit. Jika algoritma RR menghitung jumlah koneksi saat membagikan koneksi, maka algoritma LC memperhitungkan jumlah koneksi yang aktif yang berarti hal ini berhubungan dengan waktu pemrosesan. Setiap satuan waktu tertentu algoritma LC akan mengecek jumlah koneksi yang sedang aktif pada masing-masing server dan kemudian membuat urutan prioritas pembagian kerja dengan server yang mempunyai koneksi aktif paling sedikit yang menjadi urutan pertama. Jika

terdapat dua atau lebih server yang mempunyai jumlah koneksi aktif yang sama, maka LC akan membagikan kerja sebagaimana cara yang dilakukan oleh algoritma RR.

Sekilas memang sepertinya penjadwalan least connection juga akan berjalan baik walaupun terdapat perbedaan kapasitas pemrosesan di antara server-server yang ada di dalam cluster, karena server yang mempunyai kapasitas pemrosesan yang lebih besar akan mendapatkan koneksi jaringan lebih banyak. Kenyataanya, hal ini tidak berjalan baik karena TIME_WAIT state milik TCP. TIME_WAIT TCP biasanya 2 menit. Selama waktu ini sebuah website yang sibuk sering kali menerima ribuan koneksi. Contoh, server A dua kali lebih *powerful* dari server B. server A memproses ribuan *request* dan menyimpannya di dalam TIME_WAIT state-nya TCP, tetapi server B harus merangkak untuk menyelesaikan ribuan koneksi tersebut. Jadi, penjadwalan *Least Connection* tidak dapat berjalan baik pada beberapa server yang berbeda kapasitas pemrosesannya.

Kelebihan: keseimbangan pembagian kerja didasarkan pada jumlah koneksi yang aktif pada masing-masing server. Dengan demikian, lama tidaknya koneksi aktif dari pengguna juga diperhitungkan.

Kekurangan: tidak mempertimbangkan kapabilitas pemrosesan masing-masing server. Selain itu, algoritma ini secara teknik terhalang oleh salah satu prosedur TCP yaitu TIME_WAIT.

Algoritma ini cocok digunakan jika server menangani koneksi yang bervariasi.

d. *Weighted Least Connection* (WLC)

Penjadwalan ini merupakan perbaikan dari penjadwalan *Least Connection*, dimana kita dapat memberi bobot performa pada setiap server. Server yang mempunyai bobot lebih tinggi akan menerima koneksi aktif dengan persentase lebih banyak dalam satu waktu. Pada saat konfigurasi virtual server, kita dapat memberikan bobot (kapasitas pemrosesan) pada setiap server dan kemudian koneksi

jaringan akan dijadwalkan pada setiap server dengan persentase jumlah koneksi aktif sesuai dengan rasio bobot kapasitas pemrosesannya. Bobot defaultnya adalah 1. Pada perangkat lunak Ipvsadm, bobot masing-masing server node diatur menggunakan angka integer. Administrator dapat mendetailkan selisih dengan menggunakan angka-angka yang besar.

Kelebihan: dapat menyeimbangkan pembagian kerja berdasarkan jumlah koneksi yang aktif pada masing-masing server dan jumlah koneksi yang aktif diatur berdasarkan bobot yang sudah ditentukan oleh administrator. Ini merupakan versi perbaikan dari algoritma *Least Connection*.

Kekurangan: seperti juga algoritma *Weighted Least Connection*, algoritma ini membutuhkan pemberian bobot secara manual oleh administrator. Algoritma ini sangat cocok digunakan jika server menangani request yang kompleks dan administrator mengetahui kapasitas pemrosesan masing-masing server. Algoritma WLC tidak akan efektif bekerja dan bahkan bisa mengacaukan pembagian kerja jika administrator salah dalam memberikan bobot pada masing-masing server node. Oleh karena itu, ada baiknya dilakukan beberapa kali uji coba untuk menentukan bobot masing-masing server dengan tepat.

e. *Locality Based Least Connection* (LBLC)

Algoritma penjadwalan *Locality Based Least Connection* (least connection berdasarkan lokasi) merupakan algoritma *load balance* yang menggunakan tujuan IP Address. Ini biasanya digunakan pada *cache cluster*. *Load balancing server* akan memberikan pekerjaan yang ditujukan pada IP Address yang sama dengan server (node) jika server tersebut sedang tidak *overload* dan tersedia. Jika server tidak tersedia, pekerjaan akan diberikan kepada server yang mempunyai pekerjaan paling sedikit (dari sisi koneksi) dan menyimpannya sebagai referensi pada *request* berikutnya. Pemilihan server yang

mempunyai koneksi paling sedikit ini menggunakan algoritma *Wighted Least Connection*. Pemilihan IP Address tujuan akan didasarkan pada IP Address asal pengguna.

Algoritma ini sering digunakan oleh website yang mempunyai banyak server di berbagai negara dan daerah. Masing-masing server akan melayani pengguna yang berada pada negara atau daerah tertentu. Selain itu, jika hal hal ini diimplementasikan bersama dengan *cache server* atau proxy, maka ini akan meningkatkan kinerja dan respon server.

Kelebihan: menghindarkan user agar tidak mendapatkan respon dari server yang posisinya jauh yang menyebabkan lamanya waktu tunggu. Selain itu, dengan ini juga diharapkan dapat mengoptimalkan penggunaan cache pada server-server tertentu yang berdekatan.

Kekurangan: suatu saat bisa saja ada pengguna dalam jumlah yang banyak dengan alamat asal yang sama (misal dalam satu negara) melakukan request secara bersamaan dan server yang berada di dekat daerah tersebut tidak mampu menanganinya. Jika hal ini terjadi, maka terjadi ketidakseimbangan pembagian kerja. Permasalahan ini kemudian diperbaiki pada algoritma *Locality Based Least Connection with Replication.*

f. *Locality Based Least Connection with Replication* (LBLCR)

Algoritma penjadwalan ini sebenarnya juga algoritma *load balance* dengan tujuan IP Address. Perbedaannya dari LBLC adalah paket diteruskan ke sekumpulan *server node* yang akan melayani satu tujuan IP Address (*replication*). Sekumpulan server ini akan bekerja sama melayani permintaan pengguna. Pemilihan server mana yang akan melayani pengguna pada sekumpulan server ini dipilih menggunakan algoritma least connection. Jika semua server yang melayani suatu tujuan IP Address *overload*, *load balancing server* akan menggunakan algoritma *Least Connection* lagi untuk memilih server yang lain yang akan dimasukkan ke dalam kumpulan server

tersebut. Jika server set belum berubah sampai waktu yang ditentukan, server yang paling banyak memproses load (pada saat itu) akan dikeluarkan dari server set untuk menghindari tingginya tingkat replikasi.

Sama seperti LBLC, algortima penjadwalan LBLCR biasanya digunakan bersama untuk *cache server*. Jika salah satu *cache server* overload, maka algoritma LBLCR akan memilih salah satu *cache server* yang lain untuk membantu. Pemilihan ini menggunakan algoritma WLC. Jika *cache server* baru ini overload lagi, maka akan dilakukan lagi prosedur yang sama. LBLCR dapat memetakan layanan-layanan yang paling sering dikunjungi ke sekumpulan *cache server*. Ketika permintaan terhadap layanan meningkat, maka akan ditambahkan *cache server* baru secara bertahap. Begitu juga saat suatu layanan berkurang pengunjungnya, maka sedikit demi sedikit cache server akan dikurangi. Pengurangan ini menggunakan algoritma reverse LC. Server dengan jumlah koneksi paling banyak akan dikeluarkan dari *server set*.

g. *Destination Hashing* (DH)

Algoritma penjadwalan *Destination Hashing* merupakan algortima penjadwalan statik yang dapat meneruskan *request* dari *client* kepada satu *real server* tertentu sesuai dengan layanan yang diminta. Terdapat suatu tabel hash berisi alamat tujuan dari masing-masing *real server* beserta layanan yang tersedia pada setiap *real server*. Jadi, permintaan pengguna akan diarahkan sesuai dengan layanan yang diminta. Sistem kerja yang dilakukan sebenarnya agak mirip dengan LBLC, tetapi tidak ketergantungan dengan IP Address asal tetapi tergantung dengan layanan apa yang diminta oleh pengguna. Algoritma ini sangat berguna jika suatu IP Address mempunyai banyak layanan, dan masing-masing layanan memerlukan proses yang sangat besar atau mempunyai pengguna yang sangat

banyak. Algoritma ini cukup baik jika dikombinasikan dengan sistem *cluster* dimana setiap layanan ditangani oleh minimal satu *cluster*.

h. *Source Hashing* (SH)

Algoritma ini hampir sama dengan metode *Destination Hashing* tetapi pada metode ini tabel berisi mengenai informasi alamat asal paket yang dikirimkan oleh client. Jadi, pengarahan permintaan pengguna diarahkan sesuai dengan alamat asal. Algoritma ini banyak digunakan untuk layanan jual beli online. Layanan seperti *shopping cart* ini sangat sensitif sekali terhadap *cookies* dan *session*, sehingga diharapkan pengguna tidak diarahkan pada beberapa server yang berbeda pada suatu sesi tertentu. Baik algoritma DH maupun SH sangat kurang sekali informasi maupun manualnya.

i. *Shortest Expected Delay* (SED)

Algoritma ini membagikan pekerjaan ke server dengan waktu tunda paling pendek. Waktu tunggu (*delay*) yang akan dialami oleh pekerjaan = (Ci + 1) / Ui, jika sebuah koneksi dikirim ke server ke-i. Ci adalah jumlah koneksi pada server ke-i dan Ui adalah *service rate* (atau *weight*) dari server ke-i. Server *load balance* akan mengecek lamanya *delay* masing-masing server tiap satuan waktu dan membuat urutan prioritas *server node*. Algoritma ini termasuk yang algoritma yang sangat dinamis karena bobot masing-masing server node bisa berubah dengan cepat. Server node dengan waktu tunggu paling kecil akan mendapat prioritas pertama untuk mendapatkan pekerjaan. Sebaliknya, server node dengan waktu tunggu paling besar, akan mendapat prioritas terakhir dalam mendapatkan pekerjaan.

j. *Never Queue* (NQ)

Algoritma penjadwalan ini mengadopsi dua model kecepatan. Jika ada server yang tidak melakukan apapun, pekerjaan akan dikirimkan kepadanya daripada harus menunggu antrian server yang lebih cepat. Ketika tidak ada server yang *idle*, pekerjaan akan diberikan kepada server yang paling kecil waktu tunggunya. Untuk

mengukur hal ini digunakan algoritma SED. Algoritma ini juga termasuk algoritma yang cukup dinamis dan membagi kerja kepada beberapa server node. Baik algoritma SED maupun NQ mempunyai referensi yang cukup minim jika dibandingkan dengan algoritma yang lain. Bahkan manual dari Ipvsadm pun hanya memuat sedikit informasi mengenai kedua algoritma ini.

6. Memilih Algoritma Untuk Pengujian

Dari 10 macam algoritma yang disediakan oleh Ipvsadm, terdapat 6 macam algoritma yang instalasi perangkat kerasnya cukup sederhana, yaitu: *Round Robin*, *Least Connection*, *Shortest Expected Delay*, *Never Queue*, *Weighted Round Robin*, dan *Weighted Least Connection*. Keenam algoritma ini dapat dijalankan pada instalasi perangkat keras yang standar. Selain itu, keenam algoritma di atas dapat digunakan pada instalasi perangkat keras yang sama sehingga akan mempermudah penelitian. Untuk empat algoritma yang lain, dibuat dengan tipikal instalasi perangkat keras tertentu dan dibuat dengan tujuan tertentu. Misal, LBLC dan LBLCR digunakan untuk pembagian kerja yang berbasis lokasi. Instalasi perangkat keras semacam ini cukup sulit untuk dipersiapkan. Begitu juga dengan algoritma DH dan SH yang berbasis pada IP Address tujuan dan alamat asal paket, kedua algoritma ini tidak bisa dibandingkan begitu saja dengan algoritma yang lain karena kedua algoritma ini ditujukan untuk jaringan dengan karakteristik yang berbeda. Oleh karena itu, hanya dipilih enam algoritma saja yang digunakan dalam pengujian pada penelitian ini.

7. Ubuntu

William Von Hagen dalam bukunya *Ubuntu Linux Bible* (2007:xxii) menyebutkan bahwa Ubuntu berarti "kemanusiaan kepada sesama". Nama distribusi ubuntu ini merupakan komitmen sosial dan bisnis bagi semua pengguna ubuntu. Ubuntu dirilis setiap enam bulan sekali. S*upport* dan *update* untuk setiap rilis ubuntu tersedia minimal selama 18 bulan setelah rilis.

Pada bagian lain William Von Hagen (2007:6) juga menyebutkan bahwa Ubuntu Linux adalah distribusi Linux yang didirikan pada tahun 2004 dan berfokus pada kebutuhan *end-user*. Ubuntu Linux adalah produk dari proyek yang disponsori oleh Canonical, Ltd. (www.canonical.com), yaitu perusahaan yang didirikan oleh Mark Shuttleworth. Mark Shuttleworth adalah seorang pengusaha sukses di Afrika Selatan, pengembang Linux Debian sejak lama, dan advokat umum komunitas *open source*. Ubuntu adalah distribusi Linux berbasis Debian yang menggunakan antarmuka (GUI) GNOME dan UNITY (mulai rilis 11.4) sebagai *desktop environment*. Proyek Ubuntu lain diantaranya Kubuntu (versi Ubuntu yang menggunakan *desktop environment* KDE), Xubuntu (versi Ubuntu yang menggunakan *desktop environment* Xfce yang lebih ringan), dan Edubuntu (versi Ubuntu yang berfokus pada aplikasi pendidikan dan mempopulerkan penggunaan Linux di sekolah).

8. Software Server Web dan Database

a. Server Web

Nancy J. Yeagar dan Robert E. McGrath dalam buku *Web Server Technology* (1996:20) mengungkapkan bahwa *web server* merupakan sebuah komputer dengan koneksi internet, mempunyai software untuk menjalankan komputer dan terkoneksi dengan sistem lain di internet. Komputer ini mempunyai komponen yang penting yaitu perangkat lunak untuk server web. Lebih jauh lagi server web harus mempunyai layanan. Layanan inilah yang menjadi alasan kenapa web server harus ada.

Kemudian Leon Shklar dan Richard Rosen (2003:65) menjelaskan bahwa web server memungkinkan kita untuk melakukan akses HTTP (*Hyper Text Transfer Protocol*) ke sebuah situs yang sederhananya berisi dokumen-dokumen dan informasi lain yang diatur menjadi sebuah struktur *tree*, yang sebenarnya lebih mirip dengan *filesystem* pada komputer. Server web yang modern mengimplemen-

tasikan berbagai *protocol* untuk melewatkan *request* ke perangkat lunak tambahan yang menyediakan konten dinamis. Seperti kita ketahui bersama bahwa perangkat lunak dinamis ini seperti PHP, Java, dan lain-lain.

1. Apache

   Pada masa awal pengembangan web, *National Center for Super Computing Applications (NCSA)* membuat sebuah perangkat lunak server web nomor satu pada awal tahun 1995. Selang beberapa waktu para pengembang utama di NCSA mulai meninggalkan perusahaan tersebut yang menyebabkan melambatnya pengembangan perangkat lunak server web ini. Pada waktu yang bersamaan orang-orang yang menggunakan perangkat lunak server web NCSA mulai bertukar *patch* untuk server web tersebut yang kemudian menyadari bahwa forum untuk mengatur patch ini diperlukan. Hal ini memicu lahirnya Apache Group, yang digunakan untuk bertukar *patch*. Kemudian Apache Group menggunakan inti perangkat lunak milik NCSA yang ditambahi *patch* dari komunitas tadi yang kemudian menjadi perangkat lunak yang diberi nama Apache. Dalam waktu tiga tahun, Apache sudah merajai pasar perangkat lunak server web.

   Versi pertama Apache (0.6.2) didistribusikan secara resmi pada bulan April 1995, sedangkan versi 1.0 dirilis pada tanggal 1 Desember 1995. Apache Group kemudian menjadi grup non-profit yang beroperasi menggunakan internet. Pengembangan Apache tidak terbatas pada komunitas saja, tetapi juga siapapun yang mengetahui cara ikut mengembangkan Apache sangat diperbolehkan. Hal ini memberikan kesempatan kepada siapapun untuk ingin memberikan *patch*, *bug-fixed*, dan pengembangan lain yang nantinya akan dikumpulkan oleh Apache Group. Apache Group kemudian akan meninjau ulang, melakukan *testing*, dan melaksanakan *quality control* terhadap kode-kode dari

pengembang. Jika mereka puas, kode akan diintegrasikan distribusi utama Apache.

Berikut adalah fasilitas utama Apache:

- Dukungan terhadap protokol HTTP 1.1 yang terbaru
- Sederhana, namun menggunakan konfigurasi berbasis file yang *powerful*.
- Dukungan terhadap CGI (*Common Gateway Interface*).
- Dukungan terhadap FastCGI
- Dukungan terhadap Virtualhost
- Dukungan terhadap otentikasi HTTP
- Perl yang sudah terintegrasi
- Dukungan terhadap kode PHP
- Dukungan terhadap Java Servlet
- Proxy server yang sudah terintegrasi
- Dukungan terhadap *Server-Side Includes (SSI)*
- Dukungan terhadap *Secured Socket Layer (SSL)*

Versi terbaru dari Apache adalah versi 2.0. pada versi ini Apache sudah lebih fleksibel, lebih *portable*, dan lebih *scalable*. Apache 2.0 menggunakan *multiprocessing modules (MPMs)*.

2. Nginx

Nginx merupakan salah satu perangkat lunak server web yang *powerful*. Para administrator sangat menyukai perangkat lunak ini karena dianggap sangat cepat dan sederhana. Dimitri Aivaliotis (2013:7) mengungkapkan bahwa Nginx merupakan server web yang mempunyai performa tinggi yang didesain untuk server bersumber daya kecil. Nginx pertama kali disusun untuk menjadi server HTTP. Nginx dirancang untuk menangani sebuah masalah yang disebut C10K yang dijelaskan oleh Daniel Kegel (http://www.kegel.com/c10k.html). Yaitu penanganan terhadap 10000 koneksi yang bersamaan. Nginx dapat manangani hal ini menggunakan mekanisme *event-based connection-handling*, dan

menggunakan sistem operasi yang sesuai (*support*) untuk menggunakan mekanisme tersebut.

Dalam *paper* yang dirilis oleh Nginx, Inc. diungkapkan bahwa Nginx menyediakan kombinasi yang unik dari *web server*, *caching proxy*, dan solusi *load balance* untuk website yang diharapkan efisien dan konsisten. Karena desain dan arsitekturnya, Nginx memberikan kinerja yang lebih, skalabilitas, keandalan dan keamanan untuk banyak organisasi di seluruh dunia. Sekarang, Nginx adalah *web server* nomor dua paling terkenal di internet.

Nginx dibuat oleh Igor Sysoev, seorang *engineer* sistem dan perangkat lunak dari Rusia. Igor mulai membawa Nginx ke komunitas open source pada tahun 2004, dan sejak saat itu Nginx menjadi komponen internet penting. Nginx menggunakan sangat sedikit memori dan mengoptimalkan penggunakan CPU, namun tetap mempunyai performa yang maksimal. Nginx tidak menuntut perangkat keras yang tinggi, yang tentu saja akan berdampak pada sisi ekonomi perusahaan yang menggunakannya. Nginx mampu menangani banyak koneksi, namun tetap ramah memori.

b. Server Database

1. Postgresql

Neil Matthew dan Richard Stones (2005:12) dalam buku mereka *Beginning Databases with PostgreSQL* menceritakan tentang awal mula lahirnya Postgresql di Universitas Barkeley California (UCB) pada tahun 1977. Sebuah *database relasional* yang bernama "Ingres" dikembangkan mulai tahun 1977 hingga 1985. Ingres pada awalnya banyak digunakan untuk pendidikan dan penelitian. Untuk melayani pangsa pasar, Ingres diambil alih oleh Relational Technologies/Ingres Corporation dan menjadi salah satu RDBMS komersial pertama.

Sementara itu, UCB mengembangkan sebuah *server database relasional* yang diberi nama Postgres mulai tahun 1986 hingga

1994. Lagi-lagi, produk ini diambil oleh pihak komersil untuk dijual. Sekitar tahun 1994, fitur SQL ditambahkan ke dalam Postgres dan namanya berubah menjadi Postgres95.

Pada tahun 1996, Postgres menjadi sangat terkenal, dan pengembang pun memutuskan untuk membuka pengembangannya ke milis. Inilah yang mengawali suksesnya kolaborasi para sukarelawan dalam pengembangan Postgres. Postgres pun mengubah namanya menjadi PostgreSQL yang menggambarkan bahwa Postgres sudah menggunakan bahasa *query* yang standard dan Posstgresql pun lahir.

9. *Load Balance* dan Replikasi Database Postgresql

Secara umum, terdapat dua skenario dalam replikasi database, yaitu master-master dan master-slave. Skenario master-master yaitu skenario dimana masing-masing server database bisa menjadi server master yang menerima permintaan dari pengguna, kemudian mengabari server yang lain jika ada perubahan. Skenario master-slave yaitu skenario dimana terdapat satu server yang bertugas menjadi master, atau dengan kata lain sistem terpusat. Server master ini yang bertugas memberikan update kepada server-server slave ketika terjadi perubahan.

Kelebihan skenario master-master adalah tidak ada ketergantungan satu sama lain. Pengguna akan melakukan query langsung ke salah satu server. Jika query pengguna mengubah data, maka setelah selesai proses query server yang menerima query akan mengabari server yang lain. Skenario ini sangat bagus jika ingin digunakan untuk membuat *backup* server database. Kekurangannya adalah sulitnya implementasi jika terdapat lebih dari dua server database. Terdapat dua perangkat lunak yang paling banyak digunakan untuk skenario ini, yaitu Rubyrep dan Bucardo. Diantara keduanya, Rubyrep adalah yang paling handal dan mudah digunakan. Terdapat tiga komponen utama Rubyrep, yaitu pembandingan (*compare*), sinkronisasi (*sync*), dan replikasi (*replication*). Rubyrep dibuat menggunakan bahasa Ruby (Ruby Replication). Di dalam

forum-forum praktisi database, Rubyrep hanya direkomendasikan untuk dua database saja, yang kedua database tersebut diungkapkan dengan nama *left* dan *right*.

Berikut adalah fitur-fitur utama Rubyrep:

- Konfigurasi yang sederhana, bisa diselesaikan hanya dalam satu file konfigurasi.
- Instalasi yang mudah. Ketika pada server sudah terdapat JVM, kita hanya perlu men-*download* dan menjalankannya langsung.
- *Multiplatform*, karena berjalan di atas JVM.

Pada skenario master-slave terdapat fitur yang tidak bisa didapatkan di skenario master-master, yaitu: *load balance*, *parallel query,* dan *connection pooling*. *Load balance* pada database prinsipnya sama dengan *load balance* pada server web. Server master akan memilih server slave mana yang akan melayani permintaan pengguna. *Parallel query* merupakan fitur dimana beberapa server slave dapat bekerja sama mengerjakan query yang memerlukan sumber daya yang besar. Hal ini terjadi pada database yang mempunyai data yang besar, atau pada kasus-kasus *data-mining*. *Connection pooling* adalah fitur dimana server master dapat membuatkan antrian *query* permintaan dari pengguna agar server-server slave tidak kelebihan beban.

Perangkat lunak yang paling handal dan banyak digunakan adalah Pgpool, yang sekarang sudah mencapai Pgpool versi II (Pgpool II). Cukup sulit memang mencari buku yang khusus membahas Pgpool secara komprehensif, namun Pgpool ini membuat dokumentasi yang cukup lengkap. Pada user manualnya, disebutkan bahwa Pgpool merupakan *Middleware* yang berada diantara server Postgresql dan pengguna. Dengan menggunakan Pgpool, pengguna hanya dapat melihat satu server Postgresql saja, dengan kata lain server-server *slave* Postgresql bersifat transparan terhadap pengguna dan bahkan terhadap server Postgresql sendiri. Pengguna melihat Pgpool merupakan server Postgresql dan server Postgresql melihat Pgpool sebagai pengguna. Sistem operasi yang

didukung oleh Pgpool adalah Linux, Solaris, FreeBSD, dan hampir semua sistem yang berarsitektur mirip UNIX. Pgpool dapat digunakan pada Postgresql versi 6.4 ke atas. Untuk menggunakan fitur *parallel query*, Postgresql yang digunakan harus versi 7.4 ke atas. Untuk menghindari kesalahan komunikasi diantara server-server *slave*, disarankan server-server *slave* menggunakan versi Postgresql yang sama.

Default-nya, Pgpool menggunakan konfigurasi yang berbasis file seperti perangkat lunak lain pada Linux dan UNIX. Namun, sekarang sudah terdapat perangkat lunak berbasis web yang dapat kita gunakan untuk mengkonfigurasi Pgpool dengan mudah, yaitu PgpoolAdmin. PgpooAdmin perlu diinstal terlebih dahulu agar file-file konfigurasi yang diperlukan terpenuhi.

10. Performansi Jaringan dan Server

a. *Throughput*

Throughput adalah jumlah bit yang diterima dengan sukses perdetik melalui sebuah sistem atau media komunikasi dalam selang waktu pengamatan tertentu. Umumnya *throughput* direpresentasikan dalam satuan bit per second (bps). Aspek utama *throughput* yaitu berkisar pada ketersediaan *bandwidth* yang cukup untuk menjalankan aplikasi. Hal ini menentukan besarnya trafik yang dapat diperoleh suatu aplikasi saat melewati jaringan. Aspek penting lainnya adalah *error* (pada umumnya berhubungan dengan *link error rate*) dan *losses* (pada umumnya berhubungan dengan kapasitas *buffer*). *Throughput* tergantung pada faktor-faktor berikut ini:

1. Karakteristik link: *bandwidth*, *error rate*.
2. Karakteristik node: kapasitas *buffer*, daya pemrosesan.

Adapun perbandingannya dengan *bandwidth*, *bandwidth* adalah jumlah bit yang dapat dikirimkan dalam satu detik. Berikut adalah rumus dari *bandwidth*:

$$Bandwith = \frac{\sum \text{bits}}{s} \ldots (1)$$

Sedangkan *throughput* walau pun memiliki satuan dan rumus yang sama dengan *bandwidth*, tetapi *throughput* lebih pada menggambarkan *bandwidth* yang sebenarnya (aktual) pada suatu waktu tertentu dan pada kondisi dan jaringan internet tertentu yang digunakan untuk mengunduh suatu file dengan ukuran tertentu. Berikut adalah formula pembanding *throughput* dengan *bandwidth*:

$$Throughput = \frac{Jumlah\ paket\ diterima}{Waktu\ Pengukuran} x\ panjang\ paket \ldots (2)$$

Keterangan:

Throughput	= rata-rata bit per sekon (bps)
Waktu pengukuran	= detik (s)
Jumlah paket	= Bytes

b. *Response Time*

Response time adalah selang waktu antara user memasukkan suatu perintah ke dalam sistem hingga mendapat balasan selengkapnya dari sistem. Pada ITU-T G.1030 11/2005 (2005:4), dijelaskan bahwa *response time* memiliki *range* yang dapat dibagi ke dalam beberapa kategori:

1. *Instantaneous experience* (0,1 detik)

   0,1 detik adalah perkiraan batasan waktu seorang pengguna merasakan bahwa sistem langsung bereaksi (instan) setelah memasukkan input atau perintah ke sistem, contohnya pada chatting.

2. *Uninterrupted experience* (1,0 detik)

   1,0 detik adalah perkiraan batasan waktu seorang pengguna tetap tidak terganggu, meskipun pengguna mulai merasa bahwa sistem tidak merespon input secara instan, contohnya pada gaming.

3. *Loss of attention* (10 detik)

   10 detik adalah perkiraan batasan waktu seorang pengguna tetap fokus memperhatikan dialog. Pada *delay* yang lebih lama, pengguna umumnya akan melakukan pekerjaan lain sembari menunggu komputer selesai merespon, sehingga pengguna harus diberikan *feedback* bahwa komputer akan segera memberikan respon.

Response time dapat diuraikan lagi menjadi beberapa faktor, yaitu *latency*, *processing time*, dan *think time*. *Think time* merupakan waktu yang dibutuhkan server untuk membaca sebuah permintaan dan memasukkan perintah tersebut ke dalam antrian. *Processing time* merupakan waktu yang diperlukan oleh server untuk memproses sebuah permintaan. Di dalam pengukuran server web, *latency* adalah waktu yang dibutuhkan oleh permintaan untuk dikirim dari *client* sampai server memproses permintaan tersebut ditambah waktu kirim kembali dari server ke *client*.

Response time = latency + processing time + think time … (3)

Latency kemudian dapat diuraikan kembali menjadi *transfer time* ditambah *queue time*. *Transfer time* merupakan waktu yang diperlukan untuk mengirim dan menerima paket. *Queue time* adalah waktu yang dihabiskan oleh sebuah paket untuk menunggu di dalam antrian sebelum diproses.

Latency = transfer time + queue time … (4)

Waktu memproses tentu saja dipengaruhi oleh *processor*, *memory* dan kemampuan *cache* sebuah server. Semakin tinggi kecepatan prosesor dan kapasitas RAM, semakin tinggi pula kecepatan proses sebuah server yang akan berimbas pada semakin rendahnya waktu pemrosesan. Kemampuan *cache* sangat penting pada saat server

harus memberikan layanan yang sama dalam jumlah yang banyak. Waktu pemrosesan sebuah server bisa berubah-ubah sesuai banyaknya jumlah permintaan dari *client*. Perubahan ini sering juga disebut degradasi. Salah satu contoh degradasi adalah saat permintaan *client* melebihi kapasitas pemrosesan server atau melebihi kapasitas memori penyimpanan server sehingga server mengalami *overload*.

Waktu antri (*queue time*) dipengaruhi oleh banyaknya jumlah permintaan dari *client*, jumlah *thread* yang memproses permintaan pada engine server web, dan besarnya memori maksimal untuk masing-masing *thread*.

c. *Reply Connection Client*

Merupakan jumlah *request* yang diterima oleh server, kemudian server memberikan pengiriman pemberitahuan kepada *client* bahwa *request* telah diterima. Selain berpengaruh terhadap *latency, reply connection* juga berpengaruh terhadap tinggi *throughput*. Dalam praktiknya, *throughput* merupakan rata-rata besarnya *content* (jumlah bit) yang diterima dikalikan jumlah permintaan *content* yang dibalas (*replied connection*), kemudian dibagi dengan waktu pengukuran (*test durations*).

d. *Error Connection Client*

Merupakan jumlah *request* dari *client* yang ditolak atau tidak mampu terjawab oleh server. Jika terdapat 10 permintaan koneksi ke server dan *reply connection*-nya ada 8, maka *error connection*-nya sama dengan 2.

Error Connection Client = demand request – replied request … (5)

Waktu pengukuran (*test durations*) secara tidak langsung dipengaruhi oleh *response time* masing-masing permintaan. Semakin tinggi *response time* masing-masing permintaan, maka akan semakin lama pengukuran berlangsung. *Response time* tidak berpengaruh secara

langsung terhadap waktu pengukuran karena waktu pengukuran juga dipengaruhi oleh jumlah permintaan dan jumlah *thread*.

B. Penelitian yang Relevan

1. Hasil penelitian Yu Shengsheng, Yang Lihui, Lu Song, dan Zhou Jingli (Januari 2003) dalam analisa mereka mengenai algoritma *Least Connection* berdasarkan variabel *weight* pada transmisi multimedia di *College of Computer Science* China menunjukkan bahwa penggunaan algoritma ini meningkatkan kinerja server dari sebelumnya.
2. Hasil penelitian yang dilakukan oleh Fahrurrozi (2012) mengenai penggunaan *load balance* menunjukkan bahwa:
   - Pemilihan algoritma penjadwalan pada *load balance* yang tepat menentukan performasi server dan performasi jaringan.
   - Performasi server *cluster* IPTV dan *server cluster* LMS berbasis *load balance* pada penelitian ini diperoleh algoritma penjadwalan *Weighted Least Connection* (wlc) dan *Least Connection* (lc) sebagai penjadwalan yang terbaik untuk diterapkan pada *load balancer* karena bekerja dinamis dengan mengarahkan beban jaringan ke server aktif yang memiliki jumlah beban jaringan paling sedikit.
   - Peningkatan *request* klien yang terlalu besar akan menghasilkan *error connection* besar, jika tidak diikuti dengan penambahan server web.
   - Peningkatan *bandwidth* akan mempengaruhi *throughput* yang didapatkan *client*, semakin besar *bandwidth* maka *throughput* juga akan semakin besar, sehingga performasi jaringan bisa dimaksimalkan.
3. Hasil penelitian Ambia Rachman Haryadi (Maret 2013) mengenai penggunaan algoritma load balance pada sistem di PT. Bank Mega Syariah mengungkapkan bahwa *load balance* merupakan solusi terbaik karena cukup tangguh untuk dapat diterapkan pada lingkungan

jaringan yang padat akan lalu lintas data dan layanan-layanan di Internet/intranet misalkan Perbankan.

4. Penelitian Yoppi Lisyadi Oktavianus di Universitas Andalas pada sistem cloud computing yang dibuatnya menggunakan teknik *Linux Virtual Server* (Maret 2013) membandingkan lima algoritma *load balance*, yaitu LC, WLC, SH, SED, dan NQ. Dari hasil penelitiannya diketahui bahwa algoritma NQ merupakan algoritma yang mempunyai rata-rata *response time* terkecil dan rata-rata *throughput* terbesar.

C. Kerangka Berpikir

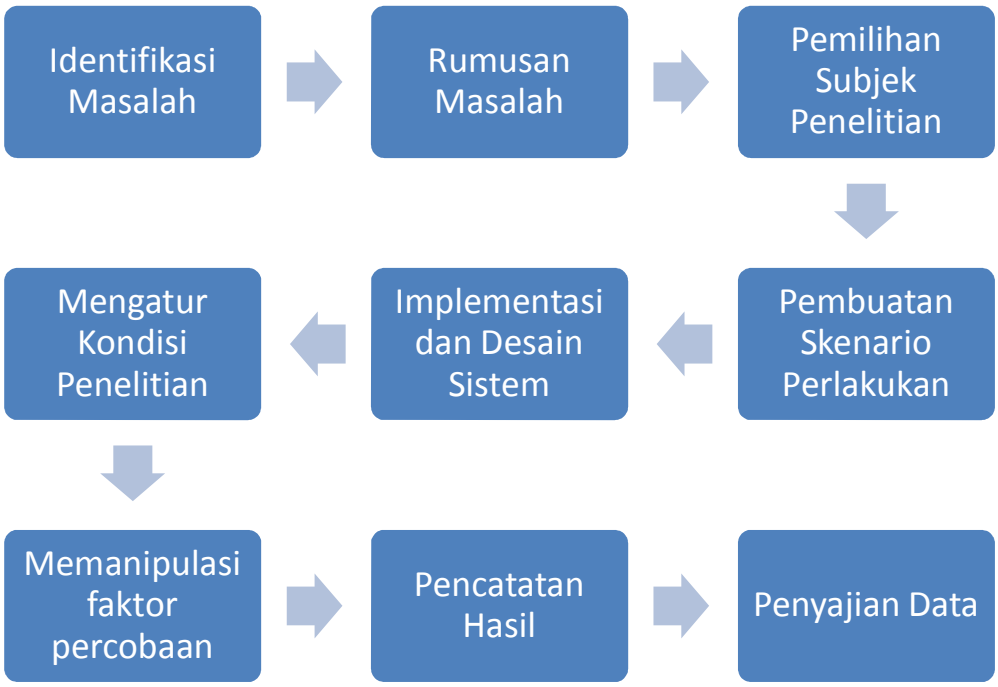
Penelitian ini didasari peningkatan penggunaan internet di masyarakat yang berdampak pada meningkatnya tuntutan terhadap penyedia informasi. Salah satu tuntutan yang diharapkan meningkat adalah kapabilitas server-server yang bertugas memberikan informasi yang diminta oleh pengguna. Peningkatan kapabilitas ini tentu saja menuntut meningkatnya spesifikasi server yang tinggi. Disisi lain, server-server yang berkemampuan tinggi itu harganya mahal.

Dengan memanfaatkan cluster, beberapa server sederhana dapat bekerjasama melayani satu layanan besar yang sama. Dalam pembagian pekerjaan, perlu diperhatikan kemampuan masing-masing node yang ada di dalam cluster. Dalam teknik *load balancing*, terdapat beberapa algoritma yang diperuntukkan bagi kondisi cluster tertentu. Pada penelitian ini disimulasikan tiga server web yang mempunyai spesifikasi fisik berbeda-beda, yang kemudian dicari algoritma mana yang paling cocok untuk keadaan tersebut.

Dalam penelitian ini dilakukan percobaan terhadap enam algoritma penjadwalan atau pembagian kerja pada sistem operasi Ubuntu. Web server yang digunakan adalah Apache2 dan Nginx, sedangkan database server yang digunakan adalah Postgresql. Dari penelitian ini akan diketahui bagaimana performa masing-masing algoritma pada kedua skenario. Hasil performa ini kemudian dibahas dan dapat menjadi salah

satu bahan pertimbangan dalam implementasi ataupun untuk dipelajari kembali.

Penelitian ini terdiri dari beberapa langkah, yaitu: identifikasi dan perumusan masalah, pemilihan subjek penelitian, pembuatan skenario perlakuan, desain dan implementasi sistem yang akan diuji, mengatur kondisi penelitian, mengatur faktor-faktor percobaan, pencatatan hasil, dan penyajian data. Berikut adalah bagan alur berpikir penelitian ini.



Gambar 3. Alur Penelitian

D. Pertanyaan Penelitian

Bagaimanakah kinerja algoritma-algoritma *load balance* yang diimplementasikan pada server web Apache2 dan Nginx serta database Postgresql?

BAB III
METODOLOGI PENELITIAN

A. Pendekatan Penelitian

Penelitian ini dilakukan dengan menggunakan pendekatan deskriptif. Statistik deskriptif adalah statistik yang berfungsi untuk mendeskripsikan atau memberi gambaran terhadap obyek yang diteliti melalui data sampel atau populasi sebagaimana adanya, tanpa melakukan analisis dan membuat kesimpulan yang berlaku untuk umum. (Sugiyono: 29).

B. Waktu dan Tempat Penelitian

Penelitian ini dilakukan mulai tanggal 2 Februari hingga 2 Mei 2013 di LIMUNY UPT. PUSKOM Universitas Negeri Yogyakarta lantai 2 blok Q.

C. Perangkat Keras dan Perangkat Lunak

1. Perangkat Keras

Dalam penelitian ini menggunakan beberapa perangkat keras berikut:

a. Sembilan buah perangkat komputer lengkap dengan spesifikasi sebagai berikut:

Tabel 2. Spesifikasi komputer

No.	Fungsi PC	Prosesor	Memory (RAM)
1.	Komputer Tester	P4 3,06 GHz	1 GB
2.	Server *load balance* web	P4 3,1 GHz	512 MB
3.	Server *load balance* dan replikasi database	P4 3,0 GHz	512 MB
4.	Server web 1	Dual Core 1,8 GHz	2 GB
5.	Server web 2	P4 2,8 GHz	1 GB
6.	Server web 3	P4 3,0 GHz	512 MB
7.	Server database 1	P4 2,4 GHz	512 MB
8.	Server database 2	P4 3,0 GHz	256 MB
9.	Server database 3	P4 3,0 GHz	512 MB

b. Instalasi jaringan lokal lengkap dan koneksi internet

2. Perangkat Lunak

a. Sistem operasi yang digunakan adalah Ubuntu 12.4. Ubuntu Desktop untuk komputer tester dan Ubuntu Server untuk lainnya.
b. *Web server* yang digunakan adalah Apache2 dan Nginx.
c. *Database server* yang digunakan adalah Postgresql.
d. Perangkat lunak PgAdmin untuk mengecek keberhasilan replikasi.
e. Perangkat lunak untuk *load balance* website adalah Ipvsadm.
f. Perangkat lunak untuk *load balance* dan replikasi Postgresql adalah Pgpool yang dilengkapi dengan PgpoolAdmin.
g. Perangkat lunak yang digunakan untuk pengukuran adalah Httperf.
h. *Script* sederhana untuk mempermudah pengujian.

D. Prosedur Penelitian

Penelitian ini terdiri dari beberapa langkah, yaitu: identifikasi dan perumusan masalah, pemilihan subjek penelitian, pembuatan skenario

perlakuan, desain dan implementasi sistem yang akan diuji, mengatur kondisi penelitian, mengatur faktor-faktor percobaan, pencatatan hasil, dan penyajian data.

1. Skenario Perlakuan

Dalam eksperiman ini dibuat dua skenario sebagai berikut:

a. Skenario 1

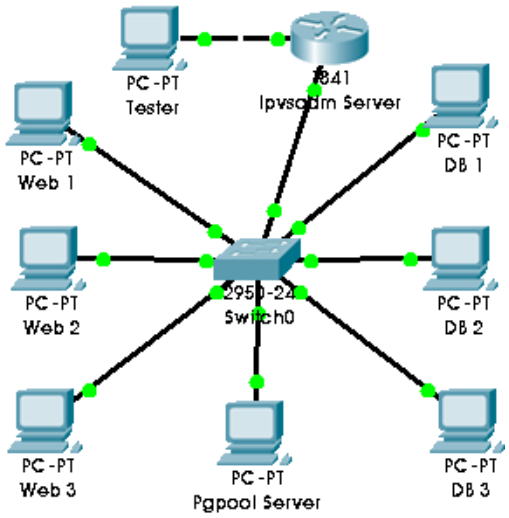
Pada skenario ini dilakukan percobaan terhadap tiga server web yang menjalankan Apache2 dan sudah di *load balance* menggunakan server khusus. Server khusus ini menjalankan perangkat lunak Ipvsadm. Di sini dilakukan percobaan penggantian algoritma *load balance* pada Ipvsadm. Server database Postgresql yang digunakan berjumlah 3 server yang sudah di *load balance* dan replikasi menggunakan Pgpool dan PgpoolAdmin.

b. Skenario 2

Pada prinsipnya skenario 2 hampir sama dengan skenario 1, perbedaannya adalah perangkat lunak server web yang digunakan adalah Nginx.

2. Rancangan sistem

Rancangan jaringan yang digunakan untuk percobaan menggunakan jaringan lokal sederhana seperti yang terlihat pada gambar di bawah ini:



Gambar 4. Rancangan Sistem

Berikut adalah detail konfigurasi dasar masing-masing komputer:

Tabel 3. Detail konfigurasi masing-masing komputer

Nama Komputer	IP Address	Keterangan
Tester	Eth0 : 10.129.0.133/24	Komputer untuk melakukan pengujian
Ipvsadm Server	Eth0 : 10.129.0.134/24 Eth1 : 11.12.13.1/24	Server *load balance* sekaligus Router
Web 1	Eth0 : 11.12.13.10/24	Server web
Web 2	Eth0 : 11.12.13.20/24	Server web
Web 3	Eth0 : 11.12.13.30/24	Server web
Pgpool Server	Eth0 : 11.12.13.100/24	Server *load balance* dan replikasi Postgresql
DB 1	Eth0 : 11.12.13.11/24	Server database
DB 2	Eth0 : 11.12.13.12/24	Server database
DB 3	Eth0 : 11.12.13.13/24	Server database

3. Implementasi

a. Setting IP Address

Untuk setting IP Address pada Ubuntu Server, file yang perlu di edit adalah file "/etc/network/interfaces". Berikut adalah contoh konfigurasinya:

```
#nano /etc/network/interfaces
auto lo
iface lo inet loopback
auto eth0
iface eth0 inet static
        address 11.12.13.10
        netmask 255.255.255.0
        gateway 11.12.13.1
```

Sedangkan file yang diubah untuk setting alamat DNS adalah file "/etc/resolv.conf". Berikut adalah contoh konfigurasinya:

```
#nano /etc/resolv.conf
nameserver 10.129.0.254
```

atau bisa juga menggunakan perintah:

```
#echo "nameserver 10.129.0.254" > /etc/resolv.conf
```

b. Setting NAT pada Router

Untuk membuat Ubuntu Server menjadi sebuah Router, file konfigurasi yang perlu diedit adalah "/etc/sysctl.conf". Pada file ini akan ditambahkan perintah untuk melewatkan paket IP.

```
#echo "net/ipv4/ip_forward = 1" > /etc/sysctl.conf
#sysctl –p (untuk mengecek)
```

Kemudian pada server perlu ditambahkan perintah "iptables" untuk melewatkan paket dari jaringan lokal ke luar melalui router.

```
#iptables –t nat –A POSTROUTING –s 11.12.13.0/24 –d 0/0 MASQUERADE
```

Agar perintah tersebut dijalankan setiap startup server, perintahnya perlu ditambahkan ke dalam file "/etc/rc.local". Untuk mengecek konfigurasi iptables, digunakan perintah:

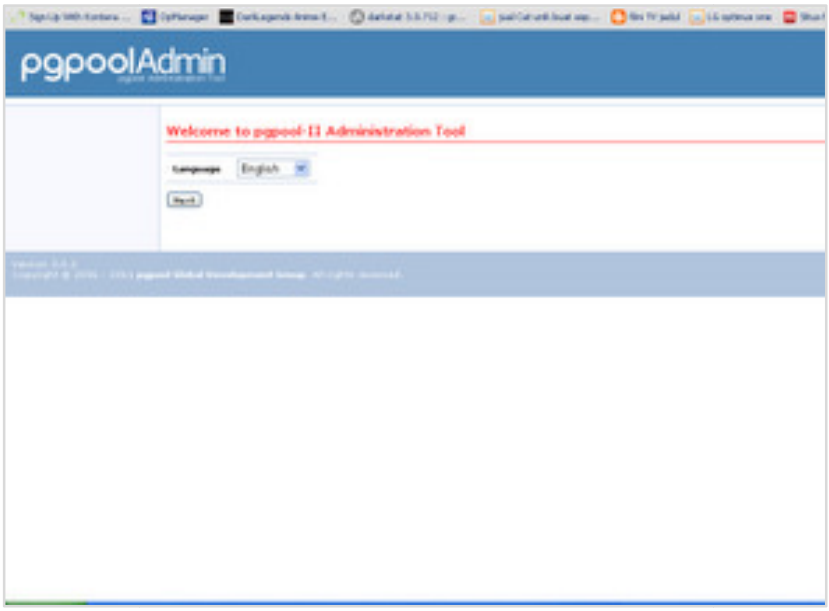
```
#iptables –L –t nat
```

c. Konfigurasi Pgpool

File konfigurasi utama Pgpool terdapat pada file "/etc/pgpool2/ pgpool.conf". Selain itu, terdapat juga file pool_passwd yang berisi password berbentuk hash md5. Pgpool juga mempunyai file pool_hba.conf yang mirip dengan file pg_hba.conf pada Postgresql. Isi konfigurasinya pun sama. Isi file-file konfigurasi file di atas terlampir.

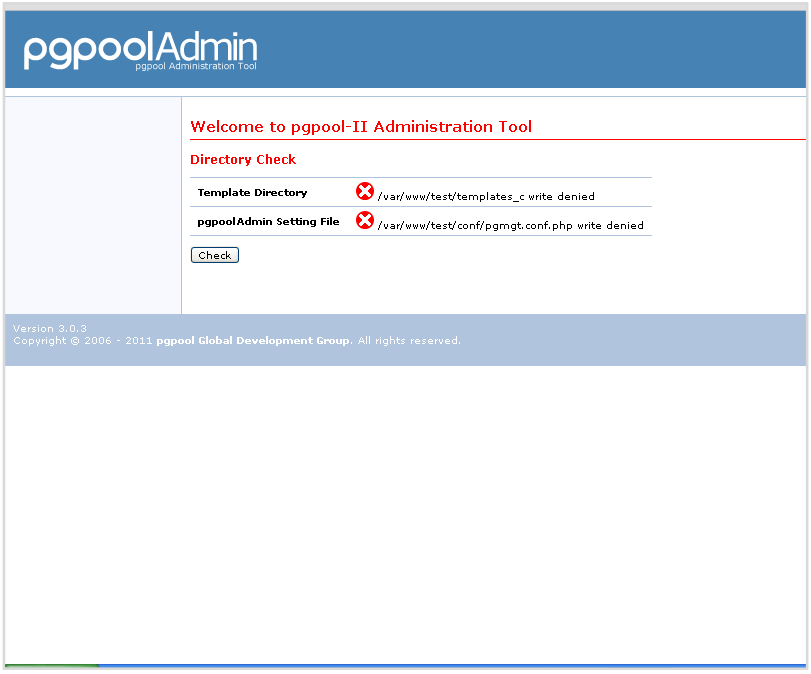
d. Konfigurasi PgpoolAdmin

Konfigurasi PgpoolAdmin cukup sederhana, layaknya menggunakan *Content Management System (CMS)* pada umumnya. Setelah diekstrak, kita perlu melakukan instalasi PgpoolAdmin via website.



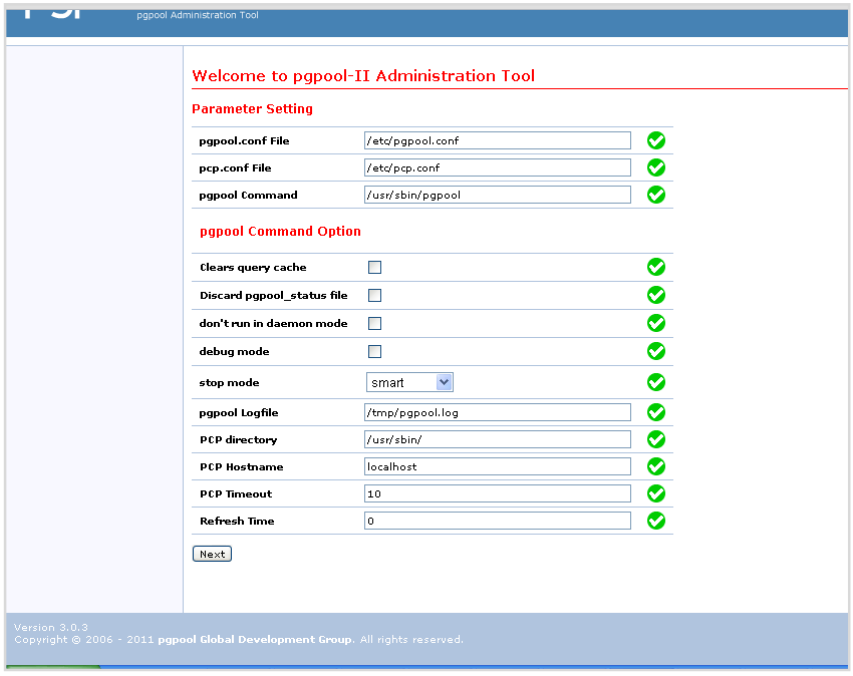
Gambar 5. Tampilan awal instalasi PgpoolAdmin

Dalam proses instalasi, PgpoolAdmin akan mengecek beberapa hal yang diperlukan oleh PgpoolAdmin untuk beroperasi seperti letak file-file konfigurasi Pgpool dan pengaturan *permission* terhadap direktori Pgpool.



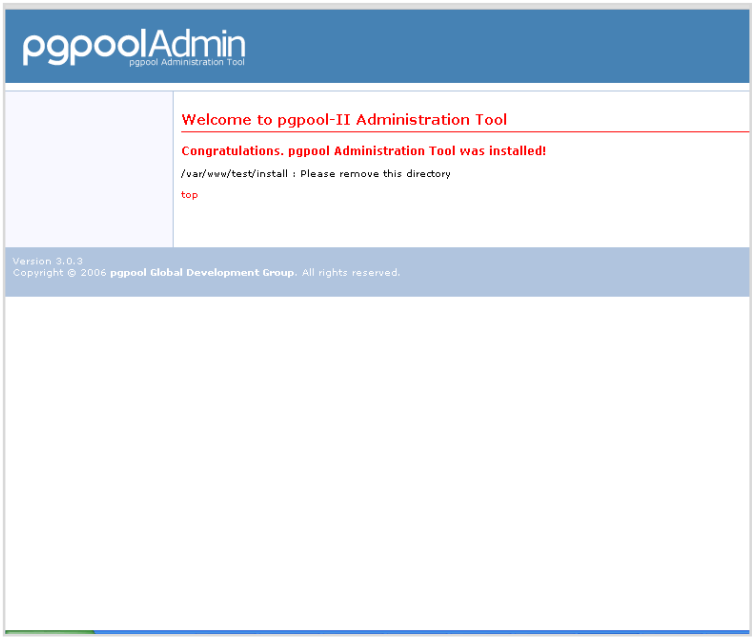
Gambar 6. Pengecekan *permission* direktori

Pada beberapa kasus kita perlu mengarahkan PgpoolAdmin ke lokasi file-file konfigurasi yang ia perlukan, sebab terdapat sedikit perbedaan lokasi hasil instalasi aplikasi pada beberapa jenis sistem operasi Linux dan Unix.



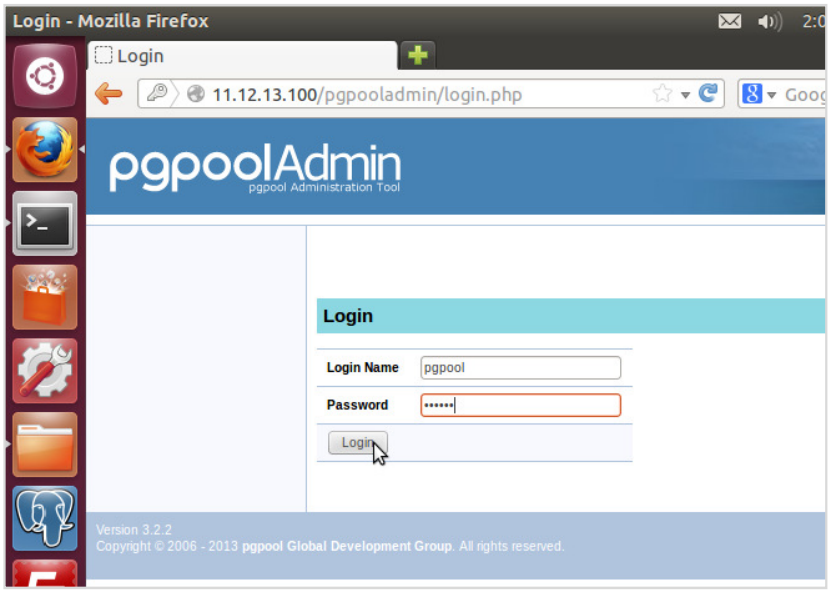
Gambar 7. Pengecekan lokasi file-file konfigurasi

Jika instalasi telah berhasil, akan muncul tampilan seperti di bawah ini:



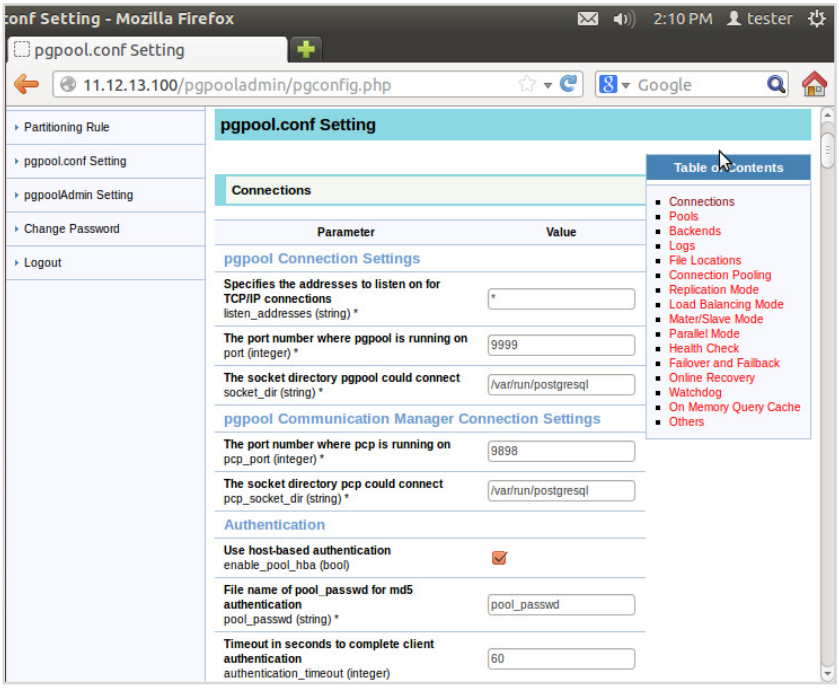
Gambar 8. PgpoolAdmin berhasil diinstal

Setelah instalasi PgpoolAdmin selesai dilakukan, kita dapat mengkonfigurasi Pgpool menggunakan fasilitas PgpoolAdmin dengan lebih mudah. Setiap kita ingin menggunakan PgpoolAdmin, kita terlebih dahulu diminta untuk login menggunakan username dan password yang tertulis pada file pcp.conf di direktori Pgpool.

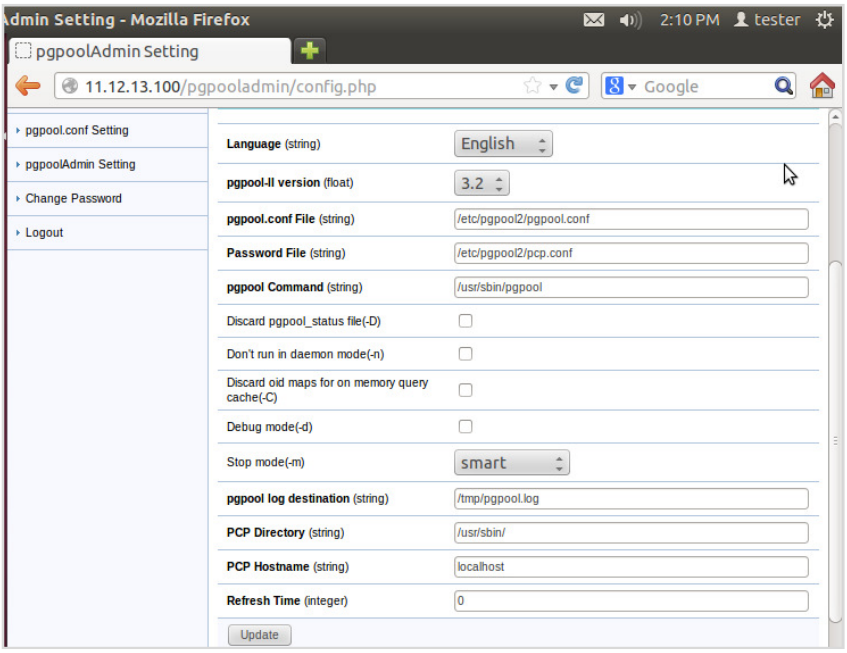


Gambar 9. Tampilan saat login ke PgpoolAdmin

Terdapat dua bagian penting dalam konfigurasi Pgpool via PgpoolAdmin, yaitu pengaturan Pgpool dan pengaturan PgpoolAdmin itu sendiri.

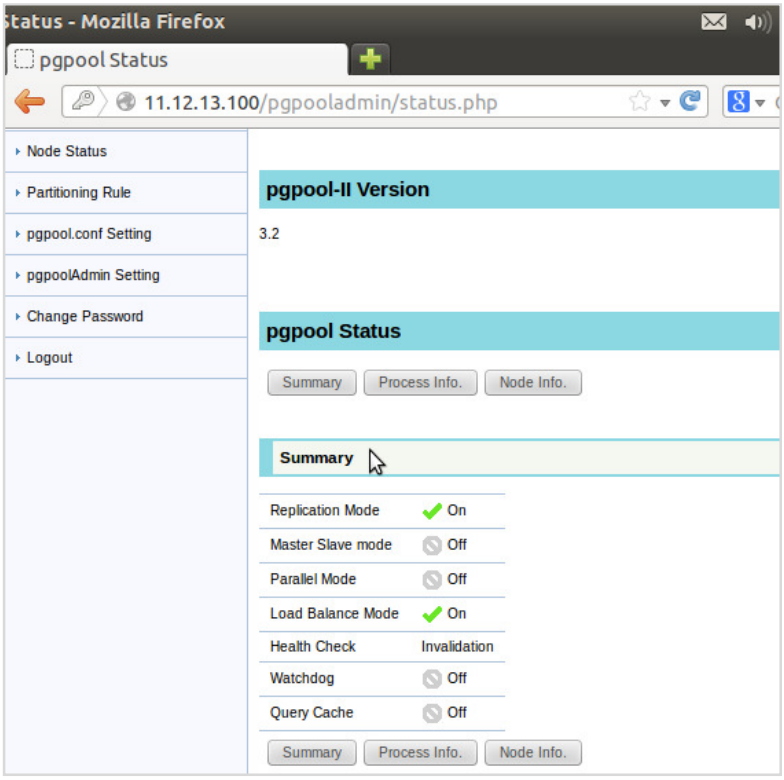


Gambar 10. Pengaturan Pgpool via PgpoolAdmin



Gambar 11. Pengaturan PgpoolAdmin

Selain itu, kita juga dapat melakukan pengamatan terhadap kondisi Pgpool dan fasilitas yang sedang dijalankan oleh Pgpool via PgpoolAdmin.



Gambar 12. Status *service* yang dijalankan Pgpool

4. Mengatur kondisi penelitian

a. Kondisi Jaringan

Jaringan lokal yang digunakan penulis kondisikan saat sepi dan tidak ada trafik jaringan. Untuk itu, penulis selalu melaksanakan percobaan pada malam hari, karena pada malam hari pengguna di Limuny tidak mencapai lantai 2.

b. Jumlah *request* perkoneksi.

Selain jumlah koneksi, terdapat juga jumlah *request* perkoneksi. Pada dokumentasi httperf koneksi diibaratkan pengguna yang membuka koneksi ke server, sedangkan *request* digambarkan sebagai aktivitas yang dilakukan oleh pengguna. Angka 10 dipilih karena

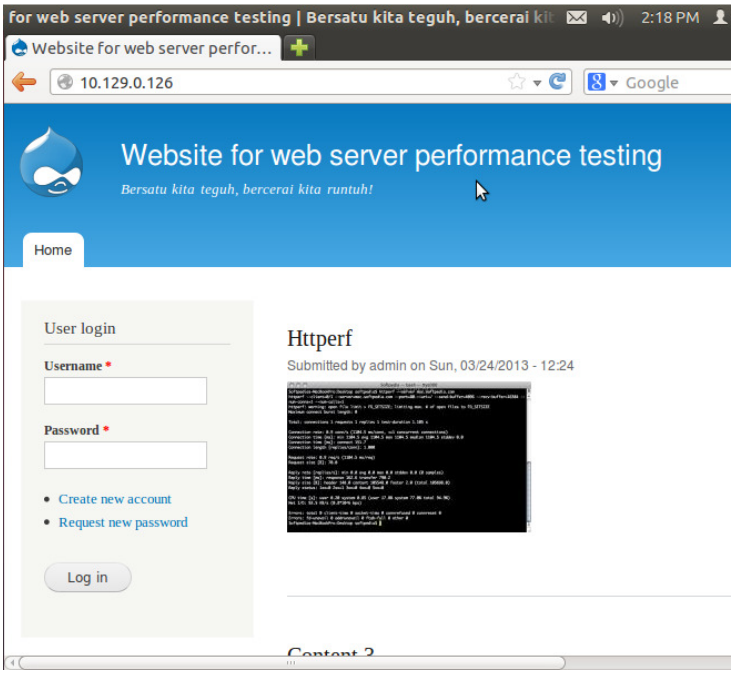
sesedikitnya aktivitas pengguna pada sebuah website adalah 10 aktivitas. Angka ini pun merupakan angka yang paling banyak digunakan pada beberapa percobaan lain yang sama yang ada pada forum-forum komunitas.

c. *Timeout*.

Pada ITU-T G.1030 11/2005 diungkapkan bahwa waktu 10 detik merupakan batas dari hilangnya perhatian pengguna terhadap sebuah website (*loss of attention*). Rata-rata pada kondisi jaringan yang normal, seorang tidak mau menunggu lebih dari 10 detik. Bahkan pada beberapa penelitian yang dilakukan di luar negeri diungkapkan bahwa seorang pengguna tidak mau menunggu lebih dari 8 detik.

d. Website yang digunakan untuk pengujian.

Dalam penelitian ini, penulis membuat sebuah website sederhana menggunakan Drupal 7 yang halaman depannya berisi tiga artikel dan satu file gambar.



Gambar 13. Tampilan website yang digunakan untuk eksperimen

e. Konfigurasi

Konfigurasi yang dimaksud di sini adalah konfigurasi yang ada pada server web dan server database. Penulis tidak mengubah parameter-parameter yang digunakan untuk optimasi. Hanya parameter-parameter yang digunakan untuk utilitas dan fungsional yang penulis ubah.

f. Spesifikasi komputer

Spesifikasi komputer tidak penulis ubah sama sekali pada setiap skenario percobaan. Komputer yang digunakan sama.

g. Jenis server database

Jenis server database yang digunakan pada kedua skenario sama, yaitu Postgresql dan di *load balance* serta replikasi menggunakan Pgpool.

h. Jumlah server web

Jumlah server web yang digunakan berjumlah tiga server dan di *load balance* menggunakan perangkat lunak Ipvsadm.

i. Jumlah server database

Seperti juga halnya jumlah server web, server database juga berjumlah tiga server dan di *load balance* menggunakan Pgpool.

5. Mengatur faktor-faktor Penelitian

a. Jenis *Web Server*

Untuk memanipulasi variabel ini hanya dilakukan penggantian perangkat lunak server web dari Apache2 ke Nginx. Hal ini terlihat dari perbedaan skenario 1 dan skenario 2.

b. Jumlah koneksi perdetik

Pada masing-masing skenario, penulis melakukan beberapa kali percobaan untuk mencari stabilitas server yang optimal dengan mengubah jumlah koneksi perdetik. Jika sudah ditemukan jumlah koneksi maksimal yang dapat ditangani oleh server, jumlah koneksi inilah yang digunakan untuk mengambil data percobaan.

Jumlah koneksi perdetik diatur pada perintah yang digunakan oleh httperf seperti pada contoh perintah di bawah ini.

```
httperf --server hostname \
  --port 80 --uri /test.html \
  --rate 150 --num-conn 100 \
  --num-call 1 --timeout 5
```

c. Algoritma load balance untuk server web

Dilakukan beberapa kali percobaan menggunakan algoritma yang berbeda-beda untuk setiap skenario dan kemudian diambil data hasil percobaannya. Untuk menambahkan algoritma *load balance* ke server Ipvsadm digunakan perintah berikut:

```
#ipvsadm -A -t 11.12.13.1:80 -s wrr
```

Perintah di atas berarti bahwa server akan menjalankan *service* nya di interface yang menggunakan IP Address 10.11.12.1 pada port 80. Algoritma yang digunakan adalah wrr (weighted round robin). Berikutnya kita perlu mendefinisikan IP Address atau server mana yang akan menerima *request* dari pengguna yang diteruskan oleh server Ipvsadm.

```
# ipvsadm -a -t 11.12.13.1:80 -r 11.12.13.10:80 -m -w 100
```

Perintah di atas memerintahkan server mengalihkan *request* yang masuk ke IP Address 11.12.13.1 pada port 80 ke IP Address 11.12.13.10 port 80 juga. Pengalihan ini menggunakan opsi MASQUERADE (-m). Parameter –w 100 berarti server ini akan diberi nilai (weight) 100. Pemberian nilai ini menggunakan bilangan decimal (integer). Pemberian parameter ini multak dilakukan karena jika algoritma yang digunakan adalah wrr dan wlc.

E. Teknik Pengumpulan Data

Teknik pengumpulan data yang digunakan pada penelitian ini adalah observasi terstruktur. Sugiyono (2011:145) mengungkapkan bahwa observasi tidak terbatas pada orang, tetapi juga objek-objek alam yang lain. Teknik

pengumpulan data dengan observasi digunakan bila penelitian berkenaan dengan perilaku manusia, proses kerja, gejala-gejala alam dan bila responden yang diamati tidak terlalu besar.

Instrumen yang digunakan untuk mengumpulkan data pada penelitian adalah perangkat lunak. Pemilihan instrumen perangkat lunak mutlak harus dilakukan karena variabel yang diukur tidak dapat diukur menggunakan panca indera manusia. Jadi, pengumpulan data ini dilakukan dengan pengukuran tidak langsung.

Perangkat lunak yang digunakan adalah Httperf. Httperf sudah banyak digunakan untuk pengukurang performa server web pada beberapa penelitian. Rob Orsini (2007:385) dalam bukunya *Rails Cookbook* memberikan Httperf sebagai solusi dalam pengukuran performa server web. Selain itu, Julian T J Midgley (2001) juga mencontohkan pengukuran performa server web menggunakan Httperf. Httperf dipilih karena mempunyai utilitas yang baik dan lengkap dalam pengukuran performa server web yang statis. Selain hal-hal di atas, penulis memilih Httperf karena keluaran yang dihasilkan oleh Httperf cukup lengkap mencakup variabel-variabel terikat yang ingin dicari oleh penulis.

Setiap alat ukur tentu mempunyai keterbatasan dan kemungkinan kesalahan pengukuran. Oleh karena itu, pengukuran akan dilakukan sebanyak 10 kali pada masing-masing satuan skenario untuk menghasilkan data yang mendekati benar.

Berikut adalah contoh penggunaan perintah Httperf dalam satu kali pengukuran:

```
#httperf --server 10.129.0.126 --port 80 --uri /index.php --rate 95 --num-conn 95 --num-call 10 --timeout 10
```

Pada contoh di atas terlihat ada beberapa parameter yang digunakan. Masing-masing parameter akan dijelaskan berikut ini:

1. Server: merupakan parameter yang digunakan untuk mendefinisikan alamat server yang akan diuji. Parameter dapat berupa hostname, ip address, atau domain.

2. Port: adalah parameter dimana server web menjalankan *service*. *Default*-nya server web akan menjalankan *service* pada port 80. Sehingga jika parameter ini tidak didefinisikan, maka Httperf akan menggunakan port 80 sebagai parameter.
3. Uri: akan digunakan jika file tujuan yang digunakan untuk pengukuran tidak berada tepat di bawah root dari *Home Directory* server web. Parameter ini berguna untuk menspesifikkan direktori dan nama file yang akan diuji.
4. Num-conn: pada parameter ini akan ditetapkan jumlah koneksi yang akan dibuat oleh Httperf dalam satu kali pengukuran.
5. Rate: merupakan parameter yang digunakan untuk mendefinisikan jumlah koneksi yang akan dikirim ke server setiap detiknya. Jadi, jika pada sebuah pengukuran didefinisikan bahwa jumlah koneksi yang akan dibuat sebanyak 40 koneksi dan kemudian ditetapkan tinggi *rate*-nya adalah 10, maka Httperf akan membuat 10 koneksi perdetik sebanyak 4 kali.
6. Num-call: dengan parameter ini dapat didefinisikan jumlah *request* yang dibuat oleh Httperf untuk setiap koneksinya.
7. Timeout: jika server web tidak dapat merespon lebih dari waktu yang sudah ditetapkan pada parameter ini (menggunakan satuan detik), maka koneksi dinyatakan gagal dibuat.

Selain menggunakan Httperf, penulis juga menggunakan *script* sederhana dalam pengukuran. *Script* ini dibuat karena melihat pada teknik percobaan awal dan pengukuran hasil ini terjadi perulangan yang teratur. Ada dua buah *script* yang dibuat, yaitu *script* untuk melakukan percobaan awal yang jumlah koneksi dapat dinaikkan secara teratur dan *script* untuk melakukan pengukuran sebenarnya.

Berikut adalah *script* pertama yang digunakan untuk *pressing* kinerja server web:

#nano pressing.sh

```
#!/bin/sh
a=10
waktu=50
for i in `seq 1 10`
```

```
do
    koneksi=$(($i*$a))
    httperf --server 10.129.0.126 --port 80 --uri /index.php
    --rate $koneksi --num-conn $koneksi --num-call 10
    --timeout 10
    echo percobaan ke-$i
    sleep $(($waktu*$i))
done
```

Pada *script* di atas terlihat ada dua variabel yang dapat didefinisikan, yaitu variabel a dan variabel waktu. Variabel a adalah variabel yang digunakan untuk banyaknya pertambahan jumlah koneksi setiap kali pengukuran. Variabel waktu adalah variabel yang digunakan untuk memberikan jeda untuk saturasi kepada server hingga keadaan normal. Hal ini harus dilakukan karena pengukuran ini termasuk pengukuran statis. Jeda waktu ini berbeda pada setiap skenario pengukuran tergantung cepat lambatnya proses saturasi ini oleh server. Variabel a dan waktu akan bertambah setiap kali pengukuran.

Sedangkan *script* yang digunakan untuk mengukur kinerja server web dapat dilihat di bawah ini:

```
#!/bin/sh
for i in `seq 1 10`
do
  httperf --server 10.129.0.126 --port 80 --uri /index.php
     --rate 95 --num-conn 95 --num-call 10 --timeout 10
  echo "percobaan ke-$i"
  sleep 250
done
```

F. Teknik Penyajian Data

Penelitian ini merupakan penelitian deskriptif. Penelitian ini tidak dimaksudkan untuk menguji hipotesis tertentu, tetapi hanya menggambarkan apa adanya tentang kinerja algoritma *load balance*. Dalam penelitian ini akan disajikan beberapa jenis data, yaitu data mentah dan data gabungan. Data mentah akan disajikan dalam bentuk tabel, sedangkan data gabungan akan disajikan dalam bentuk grafik sederhana. Analisis data dilakukan menggunakan deskriptif kualitatif.

BAB IV
HASIL PENELITIAN DAN PEMBAHASAN

A. Hasil Penelitian

Berikut adalah hasil pengukuran yang dilakukan sesuai dengan skenario yang sudah dirancang. Hasil pengukuran disajikan ke dalam dua tabel, yaitu tabel percobaan awal dan tabel hasil pengukuran. Tabel percobaan awal memuat hasil percobaan dalam rangka mengukur jumlah koneksi yang mampu ditampung oleh skenario tersebut. Jumlah koneksi ini kemudian digunakan dalam pengukuran sebenarnya yang hasilnya disajikan pada tabel hasil pengukuran.

1. Skenario 1

a. Menggunakan Algoritma *Round Robin*

Tabel 4. Percobaan awal skenario 1 menggunakan algoritma *RR*

Koneksi	*Throughput (KB/s)*	*Response Time (s)*	*Request*	*Reply*
25	115.2	1798.2	169	160
50	131.0	3632.2	340	340
100	50.2	5697.7	167	101
100	56.2	5465.8	159	92
70	132.5	4864.5	461	460
65	131.3	4615.8	430	430
50	130.0	3554.1	330	330
55	124.0	3903.6	370	370
60	112.5	4312.9	400	400
65	131.8	4530.9	430	430
70	132.0	5020.7	470	470
75	132.8	5040.4	482	480
75	133.1	5282.0	500	500
75	131.6	7306.6	682	674
75	127.2	7043.2	654	642
80	125.0	6374.9	585	560
80	124.7	6492.1	596	572
80	124.2	6691.2	600	577
77	129.5	6601.1	620	603
77	128.8	6451.1	601	581
77	125.5	5834.0	542	515

Jumah koneksi yang digunakan adalah 77 koneksi.

Tabel 5. Hasil pengukuran skenario 1 menggunakan algoritma *RR*

Pengukuran ke	*Throughput (KB/s)*	*Response Time (ms)*	*Request*	*Reply*
1	127.6	6125.5	571	548
2	129.9	6769.2	628	612
3	128.8	6751.9	623	606
4	128.4	6627.2	615	597
5	127.6	6216.9	574	551
6	129.0	6772.7	625	608
7	129.9	6717.5	626	609
8	126.8	6381.0	588	566
9	124.7	5877.8	535	507
10	128.7	7039.7	647	633
Rata-rata	**128.14**	**6527.94**	**603.2**	**583.7**

b. Menggunakan Algoritma *Least Connection*

Tabel 6. Percobaan awal skenario 1 menggunakan algoritma *LC*

Koneksi	*Throughput (KB/s)*	*Response Time (ms)*	*Request*	*Reply*
10	128.2	926.9	100	100
20	161.1	1524.5	200	200
30	161.4	2283.9	300	300
40	159.9	3005.9	400	400
50	160.8	3770.8	500	500
60	165.0	4525.0	600	600
70	173.2	4793.3	655	650
80	166.4	5819.4	764	760
90	179.4	5625.0	738	720
100	79.1	5346.5	239	139
90	160.3	6000.0	760	744
85	171.6	5873.6	769	760
80	175.3	5328.7	719	710
80	177.1	5497.4	719	710
85	181.6	5630.6	742	730
85	170.1	5815.3	760	750

Saat koneksi ditingkatkan hingga 90 koneksi, salah satu server web mengalami *overload*. Jumlah koneksi yang digunakan adalah 85 koneksi.

Tabel 7. Hasil pengukuran skenario 1 menggunakan algoritma *LC*

Pengukuran ke	*Throughput (KB/s)*	*Response Time (ms)*	*Request*	*Reply*
1	168.1	5978.4	769	760
2	172.2	5892.9	769	760
3	174.8	5487.1	715	700
4	176.9	5594.5	733	720
5	168.8	5992.0	778	770
6	170.4	5870.6	769	760
7	177.1	5648.9	751	740
8	177.3	5673.9	742	730
9	177.1	5630.6	733	720
10	177.7	5707.1	751	740
Rata-rata	**174.04**	**5747.6**	**751**	**740**

c. Menggunakan Algoritma *Shortest Expected Delay*

Tabel 8. Percobaan awal skenario 1 menggunakan algoritma *SED*

Koneksi	*Throughput (KB/s)*	*Response Time (ms)*	*Request*	*Reply*
10	134.5	895.1	100	100
20	157.1	1536.0	200	200
30	27.5	429.8	20	20
40	163.2	2965.7	400	400
50	166.7	3695.7	500	500
60	164.9	4487.9	600	600
70	175.8	4766.9	646	640
80	173.9	5406.3	719	710
90	156.8	5037.8	608	573
100	155.9	5575.8	666	626
110	174.2	5639.6	729	687
120	134.2	6616.2	693	631
80	165.1	5117.2	684	670
85	176.1	5552.4	724	710
90	162.8	4933.8	616	582
90	133.7	6048.4	608	573
85	167.3	5223.0	697	680
82	179.8	5243.8	694	680
82	179.3	5615.4	730	720
82	178.8	5614.0	739	730

Jumlah koneksi yang digunakan adalah 82 koneksi.

Tabel 9. Hasil pengukuran skenario 1 menggunakan algoritma *SED*

Pengukuran ke	*Throughput (KB/s)*	*Response Time (ms)*	*Request*	*Reply*
1	181.1	5536.1	730	720
2	175.2	5763.2	766	760
3	179.0	5553.5	739	730
4	178.7	5532.2	748	740
5	183.3	5407.9	730	720
6	183.1	5442.3	730	720
7	179.7	5441.7	714	702
8	181.6	5408.7	712	700
9	175.8	5776.1	766	760
10	182.1	5472.7	730	720
Rata-rata	**179.96**	**5533.44**	**736.5**	**727.2**

d. Menggunakan Algoritma *Never Queue*

Tabel 10. Percobaan awal skenario 1 menggunakan algoritma *NQ*

Koneksi	*Throughput (KB/s)*	*Response Time (ms)*	*Request*	*Reply*
10	125.2	951.2	100	100
20	154.1	1572.9	200	200
30	156.1	2226.4	300	300
40	159.1	2980.0	400	400
50	159.3	3765.0	500	500
60	178.6	3863.1	528	520
70	169.6	4848.5	655	650
80	174.8	5504.5	728	720
90	147.0	5779.5	640	608
100	91.2	5749.4	281	181
80	174.7	5108.3	675	661
85	177.1	5656.1	742	730
90	176.6	5417.4	709	687
80	170.9	5643.2	746	740
80	175.1	5650.3	755	750

Jumlah koneksi yang digunakan adalah 80 koneksi.

Tabel 11. Hasil Pengukuran skenario 1 menggunakan agoritma *NQ*

Pengukuran ke	*Throughput (KB/s)*	*Response Time (ms)*	*Request*	*Reply*
1	176.0	5494.8	737	730
2	182.3	5281.3	710	700
3	177.2	5380.0	719	710

4	176.7	5412.2	737	730
5	180.2	5308.6	719	710
6	178.0	5440.9	737	730
7	172.2	5418.6	737	730
8	171.4	4481.1	610	583
9	167.2	5948.3	782	780
10	180.2	5305.2	710	700
Rata-rata	**176.14**	**5347.1**	**719.8**	**710.3**

e. Menggunakan Algoritma *Weighted Round Robin*

Tabel 12. Percobaan awal skenario 1 menggunakan algoritma *WRR*

Koneksi	*Throughput (KB/s)*	*Response Time (ms)*	*Request*	*Reply*
10	87.7	981.4	91	87
20	177.6	1540.5	200	200
30	188.8	2217.3	300	300
40	185.6	3007.0	400	400
50	189.8	3739.5	500	500
60	188.2	4500.1	600	600
70	186.6	5246.5	700	700
80	185.2	5888.6	773	770
90	184.7	6074.5	801	790
100	181.9	6211.3	811	790
95	168.0	5184.7	683	653
95	183.1	6519.0	860	850
95	180.4	6126.6	797	780
95	182.0	5988.8	788	770

Jumlah koneksi yang digunakan adalah 95 koneksi.

Tabel 13. Hasil pengukuran skenario 1 menggunakan algoritma *WRR*

Pengukuran ke	*Throughput (KB/s)*	*Response Time (ms)*	*Request*	*Reply*
1	179.5	6049.9	797	780
2	182.1	6532.5	860	850
3	178.2	6165.3	815	800
4	182.7	6326.4	833	820
5	185.8	6062.3	806	790
6	175.6	5842.3	753	731
7	183.0	6397.0	842	830
8	179.1	6145.0	797	780
9	182.4	6250.3	824	810
10	186.6	6189.1	833	820
Rata-rata	**181.5**	**6196.01**	**816**	**801.1**

f. Menggunakan Algoritma *Weighted Least Connection*

Tabel 14. Percobaan awal skenario 1 menggunakan algoritma *WLC*

Koneksi	*Throughput (KB/s)*	*Response Time (ms)*	*Request*	*Reply*
10	143.3	901.1	100	100
20	173.8	1539.1	200	200
30	182.9	2208.3	300	300
40	182.1	3026.3	400	400
50	181.1	3753.5	500	500
60	181.1	4560.2	600	600
70	189.1	5219.8	700	700
80	179.5	5914.6	755	750
90	182.4	6059.7	792	780
100	174.4	6079.3	771	745
95	179.6	6243.3	807	791
95	174.4	5699.7	743	720
95	177.1	6145.8	797	780
95	168.9	5300.7	691	658
95	179.8	6470.3	845	833
95	177.9	5937	770	750
95	184.0	6639.1	878	870
95	180.5	6486.3	842	830

Pgpool mulai kehabisan RAM saat diuji menggunakan koneksi di atas 95 koneksi. Dengan demikian, jumlah koneksi yang digunakan adalah 95 koneksi.

Tabel 15. Hasil pengukuran skenario 1 menggunakan algoritma *WLC*

Pengukuran ke	*Throughput (KB/s)*	*Response Time (ms)*	*Request*	*Reply*
1	180.4	6348.2	824	810
2	182.0	6352.8	833	820
3	182.1	6183.3	815	800
4	178.5	6640.2	856	845
5	180.5	6598.8	862	852
6	180.0	6364.7	824	810
7	183.3	6539.8	860	850
8	182.0	6319.9	833	820
9	180.3	6175.5	806	790
10	181.1	6212.0	806	790
Rata-rata	**181.02**	**6373.52**	**831.9**	**818.7**

2. Skenario 2

a. Menggunakan Algoritma *Round Robin*

Tabel 16. Percobaan awal skenario 2 menggunakan algoritma *RR*

Koneksi	*Throughput (KB/s)*	*Response Time (ms)*	*Request*	*Reply*
10	126.2	707.1	100	100
20	178.3	1514.7	200	200
30	189.0	2234.8	300	300
40	188.0	2921.7	400	400
50	187.6	3506.9	500	500
60	188.4	4241.3	600	600
70	188.3	4999.4	700	700
80	188.2	5595.8	791	790
90	187.5	6317.1	867	863
100	182.9	5662.7	763	734
95	182.3	5590.0	746	721
90	189.5	6078.7	833	825

Jumlah koneksi yang digunakan adalah 90 koneksi.

Tabel 17. Hasil Pengukuran skenario 2 menggunakan algoritma *RR*

Pengukuran ke	*Throughput (KB/s)*	*Response Time (ms)*	*Request*	*Reply*
1	189.2	6088.6	833	825
2	189.1	6246.7	857	852
3	184.0	5858.8	770	754
4	184.7	6209.8	819	809
5	187.8	6122.3	824	815
6	187.6	6163.2	834	825
7	188.8	6219.4	857	850
8	187.9	6151.1	842	835
9	187.2	6187.4	853	847
10	188.3	6198.4	852	846
Rata-rata	**187.46**	**6144.57**	**834.1**	**825.8**

b. Menggunakan Algoritma *Least Connection*

Tabel 18. Percobaan awal skenario 2 menggunakan algoritma *LC*

Koneksi	*Throughput (KB/s)*	*Response Time (ms)*	*Request*	*Reply*
10	134.1	713.3	100	100
20	184.5	1500.8	200	200
30	192.3	2182.1	300	300
40	181.2	2840.3	400	400

50	186.2	3518.7	500	500
60	191.0	4202.2	600	600
70	188.3	4921.5	700	700
80	186.8	5639.9	789	787
90	186.6	6134.0	809	798
100	179.8	5533.7	721	687
90	186.9	6282.7	854	848
90	190.2	6234.1	869	865

Jumlah koneksi yang digunakan adalah 90 koneksi.

Tabel 19. Hasil pengukuran skenario 2 menggunakan algoritma *LC*

Pengukuran ke	*Throughput (KB/s)*	*Response Time (ms)*	*Request*	*Reply*
1	186.2	5909.3	784	770
2	188.0	6071.2	805	793
3	188.7	6091.3	852	845
4	185.9	6078.0	801	789
5	187.2	6144.8	819	809
6	187.6	6248.8	853	847
7	189.1	6176.5	851	845
8	186.8	6222.2	836	828
9	185.1	5692.0	762	745
10	187.7	6220.6	851	845
Rata-rata	**187.23**	**6085.47**	**821.4**	**811.6**

c. Menggunakan Algoritma *Shortest Expected Delay*

Tabel 20. Percobaan awal skenario 2 menggunakan algoritma *SED*

Koneksi	*Throughput (KB/s)*	*Response Time (ms)*	*Request*	*Reply*
10	164.0	829.2	100	100
20	181.4	1535.1	200	200
30	187.8	2231.2	300	300
40	191.9	2806.7	400	400
50	187.7	3527.6	500	500
60	186.3	4249.4	600	600
70	186.2	4933.1	700	700
80	188.6	5602.3	792	791
90	188.1	6192.3	846	839
100	181.8	5591.2	742	710
90	188.3	5956.7	793	780
90	188.3	6153.3	843	836

Jumlah koneksi yang digunakan adalah 90 koneksi.

Tabel 21. Hasil pengukuran skenario 2 menggunakan algoritma *SED*

Pengukuran ke	*Throughput (KB/s)*	*Response Time (ms)*	*Request*	*Reply*
1	188.8	6243.1	861	856
2	189.3	6193.4	858	852
3	187.5	6194.4	831	822
4	185.9	5935.1	785	771
5	187.9	6144.3	827	818
6	187.3	6118.1	810	799
7	186.3	6231.0	844	837
8	185.9	5893.5	784	770
9	188.6	6220.2	860	855
10	188.1	6103.9	814	804
Rata-rata	**187.56**	**6127.7**	**827.4**	**818.4**

d. Menggunakan Algoritma *Never Queue*

Tabel 22. Percobaan awal skenario 2 menggunakan algoritma *NQ*

Koneksi	*Throughput (KB/s)*	*Response Time (ms)*	*Request*	*Reply*
10	134.2	726.5	100	100
20	183.3	1513.9	200	200
30	188.1	2244.1	300	300
40	188.8	2902.2	400	400
50	187.7	3579.6	500	500
60	189.0	4225.8	600	600
70	190.3	4892.6	700	700
80	188.6	5702.0	800	800
90	189.0	6140.9	841	834
100	184.9	6251.8	846	827
95	188.1	6627.1	899	892
95	189.6	6340.1	851	839
90	190.3	6157.7	857	852
90	190.2	6186.0	866	862

Jumlah koneksi yang digunakan adalah 90 koneksi.

Tabel 23. Hasil pengujian skenario 2 menggunakan algoritma *NQ*

Pengukuran ke	*Throughput (KB/s)*	*Response Time (ms)*	*Request*	*Reply*
1	187.0	5906.3	785	771

2	189.2	6085.3	824	815
3	188.5	6113.4	818	808
4	187.8	6202.4	850	844
5	186.5	6258.3	843	836
6	187.3	6298.1	863	858
7	186.6	6164.5	858	853
8	187.9	6172.1	831	822
9	189.0	6210.1	857	852
10	186.9	6124.0	808	797
Rata-rata	**187.67**	**6153.45**	**833.7**	**825.6**

e. Menggunakan Algoritma *Weighted Round Robin*

Tabel 24. Percobaan awal skenario 2 menggunakan algoritma *WRR*

Koneksi	*Throughput (KB/s)*	*Response Time (ms)*	*Request*	*Reply*
10	164.1	833.8	100	100
20	183	1491.2	200	200
30	189.2	2246.2	300	300
40	188.9	2823	400	400
50	185.8	3621.7	500	500
60	180.2	4288.6	592	590
70	188.1	4899.3	700	700
80	188	5735.5	800	800
90	186	6251.2	854	838
100	181.9	5624.6	746	715

Jumlah koneksi yang digunakan adalah 90 koneksi.

Tabel 25. Hasil pengukuran skenario 4 menggunakan algoritma *WRR*

Pengukuran ke	*Throughput (KB/s)*	*Response Time (ms)*	*Request*	*Reply*
1	185.2	6213.9	834	825
2	186.1	6279.7	855	849
3	186.9	6178.0	821	811
4	186.1	6191.4	827	818
5	185.9	8277.8	841	834
6	186.4	6224.4	836	828
7	186.3	5719.1	768	752
8	185.2	5959.1	782	768
9	186.8	5983.7	794	781
10	189.2	6144.5	843	836
Rata-rata	**186.41**	**6317.16**	**820.1**	**810.2**

f. Menggunakan Algoritma *Weighted Least Connection*

Tabel 26. Percobaan awal skenario 4 menggunakan algoritma *WLC*

Koneksi	*Throughput (KB/s)*	*Response Time (ms)*	*Request*	*Reply*
10	130.6	758.7	100	100
20	183.7	1497	200	200
30	186.9	2267.6	300	300
40	189.1	2895.4	400	400
50	187.1	3600.2	500	500
60	185.8	4291.4	600	600
70	187.3	5000.8	700	700
80	187.8	5721.8	800	800
90	186.7	6212.8	834	826
100	184.6	5921.4	802	778
95	183.2	6264.8	825	810
90	187.4	6137.8	832	824
90	184.8	6166.4	802	790

Jumlah koneksi yang digunakan adalah 90 koneksi.

Tabel 27. Hasil pengukuran skenario 4 menggunakan algoritma *WLC*

Pengukuran ke	*Throughput (KB/s)*	*Response Time (ms)*	*Request*	*Reply*
1	188.4	6224.1	860	855
2	189.0	6173.7	859	854
3	186.3	6292.0	859	854
4	186.2	6199.7	818	808
5	186.7	6308.1	857	852
6	184.9	5691.9	759	742
7	187.0	6144.9	810	799
8	187.3	6199.3	833	825
9	187.9	6053.2	809	798
10	186.4	6146.5	818	808
Rata-rata	**187.01**	**6143.34**	**828.2**	**819.5**

B. Pembahasan

Sebelum membahas hasil penelitian, ada baiknya untuk terlebih dahulu mengetahui penyebab hal-hal yang menyebabkan terhambatnya pelayanan server terhadap permintaan *client*. Hal-hal yang menjadi penghambat layanan server pada penelitian ini, yaitu kapasitas *memory*, kecepatan prosesor, dan

kecepatan kartu jaringan. Karena hal ini berhubungan dengan peralatan eksperimen yang terbatas, maka hal-hal tersebut tidak dapat dihindari.

Secara teori, prosesor pada server web banyak digunakan oleh *engine* server web seperti Apache2 dan Nginx untuk memproses kode-kode HTML dan kode-kode dinamis lainnya seperti PHP atau CGI. Kemudian memory pada server web banyak dihabiskan oleh engine server database untuk keperluan pemrosesan query. Banyaknya memory yang digunakan bergantung dengan kompleksitas query dan banyaknya jumlah query dalam satu satuan waktu. Hal ini dapat terlihat pada percobaan awal skenario 1 yang menggunakan algoritma WLC. Overload terjadi pada Pgpool yang bertugas membagi query tersebut kepada ketiga server database. Selain itu, Pgpool juga bertugas mengembalikan hasil query kepada server web.

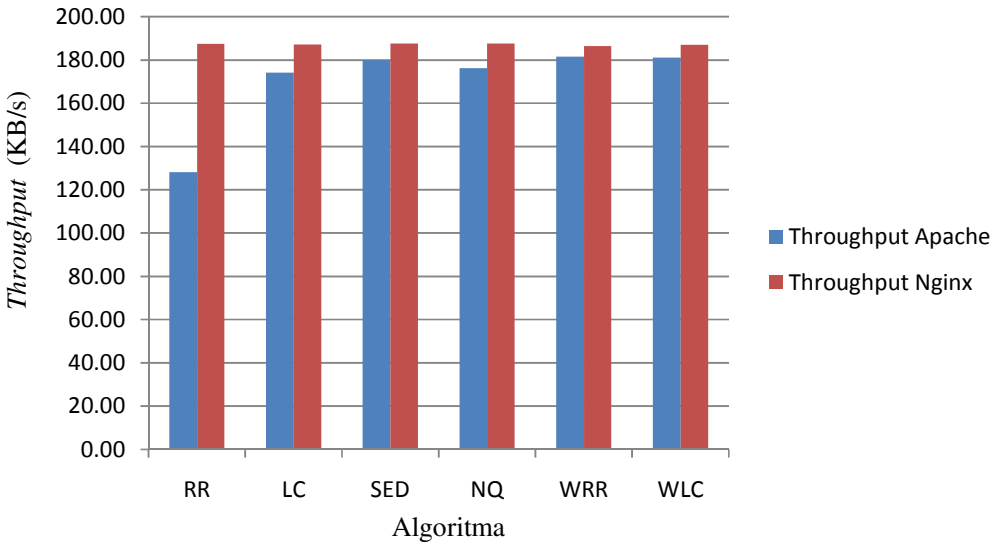
Linux Ubuntu server menyediakan fasilitas untuk memonitor pemrosesan yang terjadi, yaitu menggunakan tool Top. Dengan menggunakan tool ini, dapat diketahui jumlah memori, swap, dan prosesor yang sedang digunakan, yang sedang tidak digunakan, yang digunakan oleh sistem itu sendiri dan lain-lain. Selain itu, terdapat juga fasilitas untuk memonitor penggunaan sumber daya oleh masing-masing perangkat lunak yang sedang berjalan.

Pada skenario 1, dilakukan percobaan pada algoritma-algoritma *load balance* menggunakan server web Apache2 dan server database Postgresql. Begitu juga pada skenario 2 dilakukan percobaan pada algoritma-algoritma *load balance* menggunakan server web Nginx dan server database Postgresql. Dalam pengaturan algoritma pada Ipvsadm, algoritma *Round Robin* sering dianggap menjadi algoritma *default*.

Pada skenario 1 saat menggunakan algoritma *Round Robin* dan *Least Connection* pada bagian percobaan awal terlihat bahwa salah satu server web kekurangan sumber daya. Hal ini adalah akibat dari tidak sesuainya pembagian kerja yang dilakukan oleh Ipvsadm terhadap server-server yang berbeda spesifikasinya ini. Begitu pula saat menggunakan algoritma *Weighted Least Connection*, perlu dilakukan beberapa kali pengaturan ulang pemberian

bobot masing-masing server hingga didapatkan pemberian bobot yang benar-benar sesuai.

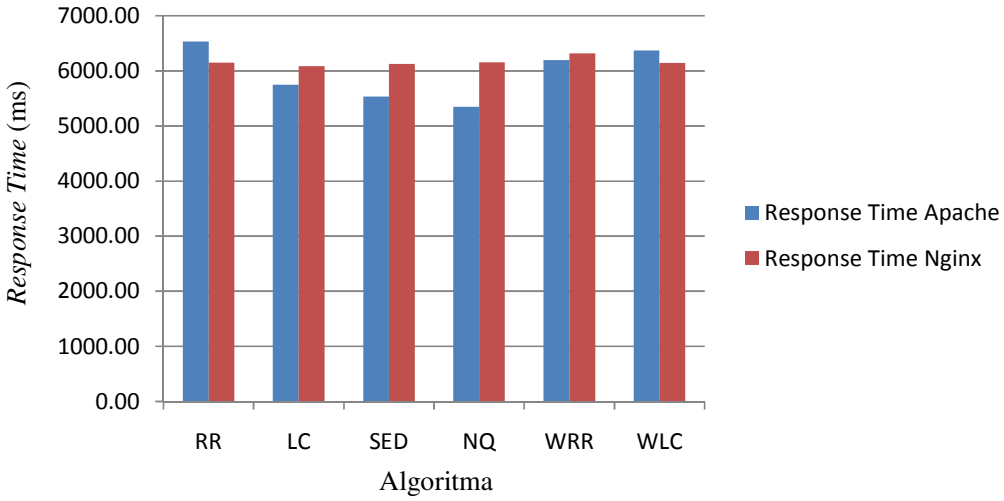
Berikut adalah rata-rata *throughput* masing-masing algoritma pada kedua skenario di atas.



Gambar 14. Perbandingan rata-rata *throughput*

Dari grafik di atas terlihat bahwa terdapat perbedaan throughput yang dihasilkan oleh web server Apache2, sedangkat pada *throughput* yang dihasilkan oleh Nginx tidak. Rata-rata throughput yang dihasilkan oleh Apache2 paling tinggi adalah menggunakan algoritma SED, WRR, dan WLC. Rata-rata throughput ketiga algoritma ini hampir sama. Sedangkan pada throughput yang dihasilkan oleh Nginx tidak terlihat perbedaan rata-rata *throughput* yang signifikan. Throughput terkecil dihasilkan oleh algoritma RR saat menggunakan web server Apache2.

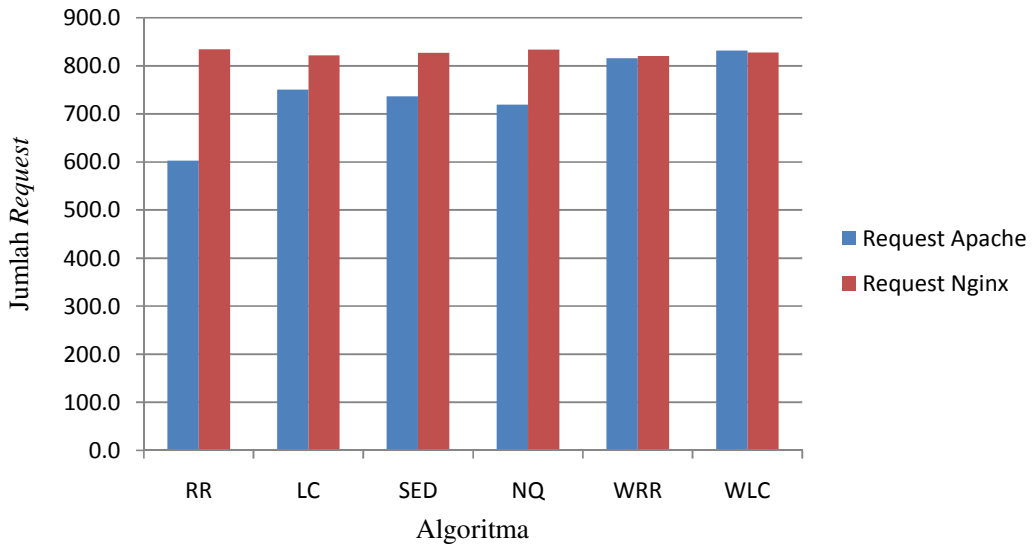
Berikut ini adalah rata-rata *response time* masing-masing algoritma pada kedua skenario.



Gambar 15. Perbandingan rata-rata *response time*

Pada grafik diatas terlihat bahwa response time yang dihasilkan oleh keenam algoritma saat menggunakan web server Nginx relatif sama. Hal ini dikarenakan throughput yang dihasilkan juga relatif sama. Sedangkan response time yang dihasilkan oleh web server Apache2 mengalami perbedaan pada masing-masing algoritma. Response time terkecil dihasilkan oleh algoritma NQ, sedangkan response time yang terbesar dihasilkan oleh RR. Padahal algoritma RR menghasilkan throughput paling kecil akan tetapi ia menghasilkan rata-rata response time yang paling besar. Hal ini menggambarkan ketidakseimbangan pembagian kinerja server.

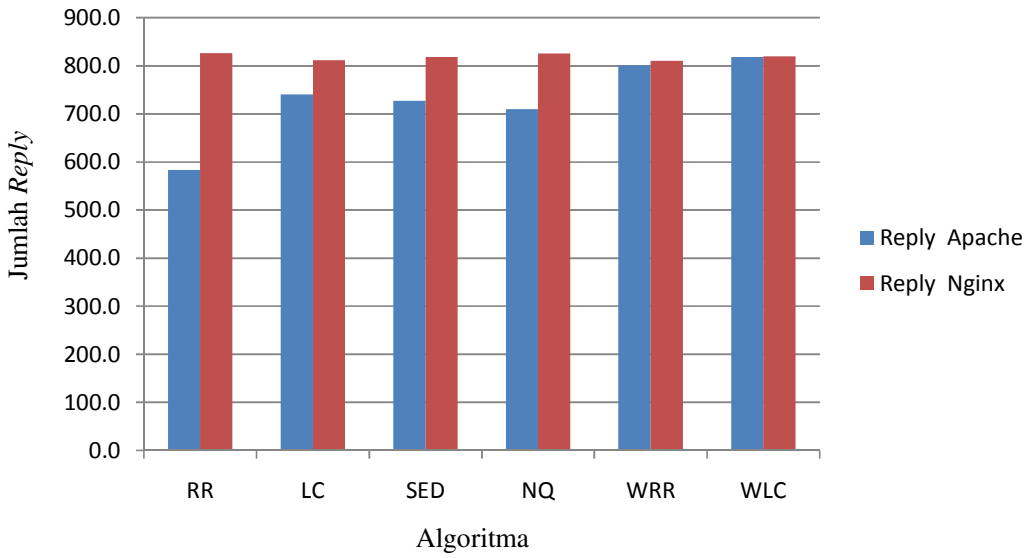
Berikut ini adalah rata-rata *request* masing-masing algoritma pada kedua skenario.



Gambar 16. Perbandingan rata-rata *Request*

Pada grafik di atas masih terlihat kesamaan hasil masing-masing algoritma saat menggunakan web server Nginx, dan sebaliknya saat menggunakan web server Apache2. Jumlah *request* paling kecil masih dihasilkan oleh algoritma RR sedangkan jumlah *request* paling besar dihasilkan oleh algoritma WRR dan WLC.

Berikut ini adalah rata-rata *reply* masing-masing algoritma pada kedua skenario.



Gambar 17. Perbandingan rata-rara reply

Web server Nginx masih menghasilkan keluaran yang sama seperti sebelumnya. Kemudian jumlah reply paling sedikit dihasilkan oleh algoritma RR. Untuk jumlah reply terbanyak dihasilkan oleh algoritma WLC yang kemudian diikuti oleh algoritma WRR pada posisi kedua. Jumlah reply ini tentu saja berkaitan dengan jumlah request yang dihasilkan, karena jumlah reply merupakan jumlah koneksi yang terlayani oleh server dari sekian banyak jumlah *request* yang dilemparkan oleh *client*.

Dari empat keadaan di atas dapat diketahui bahwa algoritma yang diimplementasikan pada server web Nginx menghasilkan keluaran yang relatif sama. Hal ini tidak terjadi saat algoritma-algoritma ini diimplementasikan pada server web Apache2. Jika dilihat dari konfigurasinya, Nginx mempunyai mempunyai variabel yang digunakan untuk memberikan batasan jumlah koneksi aktif pada waktu yang sama yaitu variabel “worker_connections”. Jika Nginx menerima koneksi melebihi batas yang ditentukan pada variabel ini, maka koneksi sisanya akan di*drop* oleh Nginx. Jika kita amati tabel-tabel

hasil percobaan keenam algoritma pada Nginx, tampak terlihat data-data yang dihasilkan sama. Ada kemungkinan sebenarnya Nginx masih mampu menangani permintaan yang lebih besar dari yang dihasilkan sekarang jika konfigurasi Nginx dioptimasi sesuai dengan spesifikasi komputer dan jenis layanan yang diberikan. Selain itu, adapula kemungkinan hal ini disebabkan oleh keterbatasan server *load balance* untuk server web atau untuk server database. Oleh karena itu, perlu kiranya diadakan penelitian lebih lanjut dalam optimasi kinerja masing-masing node yang ada di dalam sebuah *cluster* dan memahami lebih jauh bagaimana optimasi kinerja server *load balance* baik untuk server web maupun server database.

Algoritma RR saat diimplementasikan menggunakan web server Apache2 menghasilkan keluaran terburuk jika dibandingkan dengan algoritma-algoritma lain pada semua aspek. Di sisi *response time,* algoritma NQ menghasilkan keluaran terbaik. Kemudian pada sisi throughput, jumlah request dan jumlah reply tertinggi dihasilkan oleh algoritma WLC dan WRR. Jika dilihat kembali pada teorinya, kedua algoritma ini adalah algoritma yang memberikan bobot kepada setiap server nodenya. Pembagian kerja kemudian disesuaikan dengan bobot yang sudah diberikan. Hal ini cocok digunakan jika server node yang ada memang berbeda-beda spesifikasinya. Namun, untuk memberikan bobot pada server ini sebenarnya tidak boleh sembarangan. Sebaiknya ada model atau formula khusus yang digunakan berdasarkan spesifikasi server node tersebut.

Disisi lain, algoritma SED adalah algoritma yang terus mengkalkulasi waktu delay masing-masing server node dan kemudian memberikan bobot kepada masing-masing server node secara otomatis. Pada penelitian ini, algoritma SED juga tampak stabil dalam mengatur pembagian kerja server node. Ini dapat dilihat dari throughput yang hampir sama dengan throughput yang dihasilkan oleh WRR dan WLC.

BAB V
KESIMPULAN

A. Kesimpulan

Dari hasil pembahasan pada bab sebelumnya, diperoleh beberapa kesimpulan sebagai berikut:

1. Algoritma NQ menghasilkan rata-rata *response time* terendah saat diimplementasikan menggunakan server web Apache2.
2. Algoritma WRR dan WLC menghasilkan rata-rata *throughput*, jumlah *request* dan jumlah *reply* tertinggi saat diimplementasikan menggunakan server web Apache2.
3. Algoritma RR menghasilkan keluaran paling bawah pada setiap parameter saat diimplementasikan menggunakan server web Apache2.
4. Keenam algoritma *load balance* menghasilkan keluaran yang relatif sama pada setiap parameter pengukuran saat diimplementasikan menggunakan server web Nginx.

B. Rekomendasi

Dari hasil penelitian ini penulis merekomendasikan beberapa hal berikut:

1. Algoritma WRR dan WLC cukup baik digunakan pada implementasi *load balance* yang menggunakan server web Apache2
2. Jika spesifikasi server web pada suatu cluster *load balance* berbeda-beda, sebaiknya jangan gunakan algoritma RR.

C. Saran

1. Dalam membangun server load balance perlu dipertimbangkan mengenai spesifikasi perangkat keras server yang digunakan untuk *load balance* server web dan server yang digunakan untuk *load balance* dan replikasi database. Proses *load balance* dan replikasi sangat bergantung pada kemampuan perangkat keras server ini.
2. Jika ada penelitian lanjutan yang bertopik sama dan spesifikasi perangkat keras server yang digunakan untuk *load balance* kurang tinggi, bisa dicoba metode *load balance* yang kedua yaitu *direct routing* untuk server web. Secara teoritis, metode ini memberi beban lebih rendah kepada server *load balance* jika dibandingkan dengan metode pertama.
3. Perlu dilakukan juga penelitian lanjut yang mengarah ke salah satu server web seperti bagaimana optimalisasi server yang di *load balance* dan korelasi antara optimalisasi konfigurasi dengan spesifikasi server web.
4. Perlu diadakan penelitian lebih lanjut mengenai bagaimana cara menentukan rasio bobot server sesuai dengan spesifikasi perangkat keras server jika digunakan algoritma *load balance* yang menggunakan teknik pemberian bobot pada server seperti WLC dan WRR.
5. Perlu dilakukan penelitian lebih lanjut mengenai bagaimana mekanisme *fail-over* berlapis untuk menutupi kegagalan dari server *load balance*.
6. Perlu dilakukan penelitian lebih lanjut juga mengenai bagaimana cara *load balance* dan replikasi database menggunakan Mysql dan perbandingan kinerjanya dengan server database Postgresql yang sudah di *load balance* dan replikasi.
7. Jika ada penelitian lanjut yang menggunakan jumlah koneksi yang sangat besar, perlu diperhatikan bahwa secara *default* Linux membatasi jumlah *open descriptor* sebanyak 1024.

8. Jika dilakukan penelitan secara langsung di lapangan dengan trafik yang sebenarnya, maka perlu dilakukan sinkronisasi antar direktori server web sehingga *update* kode web dapat dilakukan secara otomatis

DAFTAR PUSTAKA

Aivaliotis, Dimitri. 2013. *Mastering NGINX*. Birmingham: Packt Publishing Ltd.

Anonym. 2012. *Jakarta Govt Provides Free Internet in Various Places*. http://www.jakarta.go.id/english/news/2012/08/-jakarta-govt-provides-free-internet-in-various-places. Diakses pada tanggal 2 Nopember 2012**.**

Anonym. *Linux Virtual Server*. http://en.wikipedia.org/wiki/Linux_Virtual_Server. Diakses akses pada tanggal 2 Nopember 2012.

Arikunto, Prof. Dr. Suharsini. 2010. *Prosedur Penelitian*. Jakarta: Rineka Cipta.

Baker, Mark & Buyya, Rajkumar. *Cluster Computing at a Glance*.

Bourke, Tony. 2001. *Server Load Balancing*. Sebastopol: O'Reilly & Associates, Inc.

Dr. Ir. Harinaldi, M. Eng. 2002. *Prinsip-Prinsip Statistika Untuk Teknik dan Sains*. Jakarta: Penerbit Erlangga.

Fachrurrozi. as. 2012. *Rancang Bangun Metoda Penjadwalan Load Balancing Layanan IPTV (Internet Protocol Television) Dan Layanan LMS (Learning Management System)*. :Surabaya: Fakultas Teknologi Industri, ITS.

Hardjodipuro, Dr. Siswojo. 1988. *Statistik Nonparametrik – Aplikasi dan Interpretasi*. Jakarta: P2LPTK.

Haryadi, Ambia Rachman. 2013. *Load Balancing Menggunakan Linux Virtual Server (Lvs) Via Network Address Translation (Nat) Pada Pt Bank Mega Syariah.* Jakarta: Fakultas Pasca Sarjana Universitas Gunadarma. Link publikasi: http://library.gunadarma.ac.id/repository/view/3752267/load-balancing-menggunakan-linux-virtual-server-lvs-via-network-address-translation-nat-pada-pt-bank-mega-syariah.html/.

Horman, Simon. 2004. *Linux Virtual Server Tutorial.* http://www.ultramonkey.org/papers/lvs_tutorial/stuff/lvs_tutorial.pdf. Diakses pada tanggal 15 Nopember 2012.

Internet World Statistics. 2012. Top 20 Countries – Internet Usage. http://www.internetworldstats.com/top20.htm. diakses pada tanggal 31 Agustus 2013.

ITU-T G.1030 11/2005. *Estimating end-to-end performance in IP networks for data applications.*

Kemp. *Load Balancing, Content Switching and SSL Acceleration*. http://www.kemptechnologies.com/us/navigation/faq.html. Diakses pada tanggal 20 Nopember 2012.

Lembaga Penelitian IKIP Yogyakarta. 1995. *Pedoman Penelitian*. Yogyakarta: Percetakan "KITA".

Matthew, Neil & Stones, Richard. 2005. *Beginning Databases with PostgreSQL.* United States of America: Apress.

Midgley, Julian T J. 2001. *The Linux HTTP Benchmarking.* http://www.xenoclast.org/doc/benchmark/HTTP-benchmarking-HOWTO/node1.html. Diakses pada tanggal 18 Mei 2013.

Morrison, Richard S. 2003. *Cluster Computing: Architectures, Operating Systems, Parallel Processing & Programming Languages*. GNU General Public Licence.

Office of Information Technology - The University of Tennessee Knoxville. *Selecting a load balancing method*. http://oit2.utk.edu/helpdesk/kb/entry/1699/. Diakses pada tanggal 21 Nopember 2012.

Oktavianus, Yoppi Lisyadi. 2013. *Membangun Sistem Cloud Computing dengan Implementasi Load Balancing dan Pengujian Algoritma Penjadwalan Linux Virtual Server pada FTP Server*. Padang: Fakultas Teknik Universitas Andalas.

Orsini, Rob. 2007. *Rails Cookbook*. Sebastopol. O'Reilly Media, Inc

Prof. Dr. Sugiyono. 2011. *Metode Penelitian Kuantitatif, Kualitatif, dan R & D.* Bandung: Penerbit Alfabeta.

Prof. Dr. Sugiyono. 2011. *Statistika Untuk Penelitian*. Bandung: Penerbit Alfabeta.

Rackspace Support. 2012. *Which Algorithm Should I Use?*. http://www.rackspace.com/knowledge_center/article/which-algorithm-should-i-use. Diakses pada tanggal 20 Nopember 2012 saat dokumen terakhir pada tanggal 23 Juli 2012.

Savvion Team. 2006. *Clustering Guide*. Santa Clara: Savvion Incorporated.

Shengsheng, Yu dkk. *Least-Connection Algorithm based on variable weight for multimedia transmission*. Wuhan: Huazhong University of Science & Technology.

Sterling, Thomas. 2001. *Beowulf Cluster Computing with Linux*. Cambridge: MIT Press.

Tahaghoghi, Saeid & Williams, Hugh E. 2007. *Learning MySQL.* Sebastopol: O'Reilly Media, Inc.

Von Hagen, William. 2007. *Ubuntu Linux Bible*. Indianapolis: Wiley Publishing, Inc..

Wensong. 1998. *Job Scheduling Algorithms in Linux Virtual Server*. http://www.linuxvirtualserver.org/docs/scheduling.html. diakses pada tanggal 20 Nopember 2012.

Wensong. 2011. *About LVS*. http://www.linuxvirtualserver.org/about.html. diakses pada tanggal 7 Maret 2013.

Yeager, Nancy J. & McGrath, Robert E. 1996. *Web Server Technology – The Advanced Guide for World Wide Web Information Providers*. San Fransisco: Morgan Kaufmann Publishers, Inc.

LAMPIRAN

Lampiran 1. Konfigurasi Pgpool

1. Pgpool.conf

```
listen_addresses = '*'
port = 9999
socket_dir = '/var/run/postgresql'
pcp_port = 9898
pcp_socket_dir = '/var/run/postgresql'
enable_pool_hba = on
authentication_timeout = 60
ssl = off
num_init_children = 32
max_pool = 4
child_life_time = 300
for this many seconds
child_max_connections = 0
that many connections
connection_life_time = 0
after being idle for this many seconds
client_idle_limit = 0
log_destination = 'syslog'
print_timestamp = on
log_hostname = on
log_statement = on
log_per_node_statement = on
informations
log_standby_delay = 'always'
syslog_facility = 'LOCAL0'
Default to LOCAL0
syslog_ident = 'pgpool'
debug_level = 4
pid_file_name = '/var/run/postgresql/pgpool.pid'
logdir = '/var/log/postgresql'
connection_cache = on
reset_query_list = 'ABORT; DISCARD ALL'
replication_mode = on
replicate_select = on
insert_lock = on
lobj_lock_table = ''
replication_stop_on_mismatch = off
failover_if_affected_tuples_mismatch = off
load_balance_mode = on
ignore_leading_white_space = on
white_function_list = ''
black_function_list = 'nextval,setval'
master_slave_mode = off
master_slave_sub_mode = 'slony'
sr_check_period = 0
sr_check_user = 'nobody'
sr_check_password = ''
replication check user
delay_threshold = 0
follow_master_command = ''
parallel_mode = off
enable_query_cache = off
```

```
pgpool2_hostname = ''
system_db_hostname = 'localhost'
system_db_port = 5432
system_db_dbname = 'pgpool'
system_db_schema = 'pgpool_catalog'
system_db_user = 'pgpool'
system_db_password = ''
health_check_period = 0
health_check_timeout = 20
health_check_user = 'nobody'
health_check_password = ''
failover_command = ''
recovery_user = 'nobody'
recovery_password = ''
recovery_1st_stage_command = ''
stage
recovery_2nd_stage_command = ''
stage
recovery_timeout = 90
relcache_expire = 0
pool_passwd = 'pool_passwd'
ssl_key = ''
ssl_cert = ''
ssl_ca_cert = ''
ssl_ca_cert_dir = ''
health_check_max_retries = 0
health_check_retry_delay = 1
use_watchdog = off
trusted_servers = ''
delegate_IP = ''
wd_hostname = '11.12.13.100'
wd_port = 9999
wd_interval = 10
ping_path = ''
ifconfig_path = ''
if_up_cmd = ''
if_down_cmd = ''
arping_path = ''
arping_cmd = ''
wd_life_point = 3
wd_lifecheck_query = 'SELECT 1'
memory_cache_enabled = off
memqcache_method = 'shmem'
memqcache_memcached_host = 'localhost'
memqcache_memcached_port = 11211
memqcache_total_size = 67108864
memqcache_max_num_cache = 1000000
memqcache_expire = 0
memqcache_auto_cache_invalidation = on
memqcache_maxcache = 409600
memqcache_cache_block_size = 1048576
memqcache_oiddir = '/var/log/pgpool/oiddir'
white_memqcache_table_list = ''
black_memqcache_table_list = ''
relcache_size = 256
```

```
check_temp_table = on
backend_socket_dir = '/tmp'
backend_hostname0 = '11.12.13.13'
backend_port0 = 5435
backend_weight0 = 1.0
backend_data_directory0 = ''
backend_flag0= 'ALLOW_TO_FAILOVER'
backend_hostname1 = '11.12.13.12'
backend_port1 = 5434
backend_weight1 = 1.0
backend_data_directory1 = ''
backend_flag1= 'ALLOW_TO_FAILOVER'
backend_hostname2 = '11.12.13.11'
backend_port2 = 5433
backend_weight2 = 1.0
backend_data_directory2 = ''
backend_flag2= 'ALLOW_TO_FAILOVER'
```

2. Pool_passwd

```
postgres:5b6b0a90084f9f4c49fd98f874f5835a
```

3. Pool_hba.conf

```
local      all         all                                     trust
host       all         all         127.0.0.1/32                trust
host all                all         11.12.13.1/24               trust
host all                all         11.12.13.200/24             trust
```

4. Pcp.conf

```
# USERID:MD5PASSWD
pgpool:ba777e4c2f15c11ea8ac3be7e0440a
```

KEMENTERIAN PENDIDIKAN DAN KEBUDAYAAN
UNIVERSITAS NEGERI YOGYAKARTA
UPT PUSAT KOMPUTER
Jalan Colombo Nomor 1 Karangmalang Yogyakarta 55281
Telepon (0274) 586168 ext. 228 Fax. (0274)586168 e-mail: puskom@uny.ac.id; web: http://puskom.uny.ac.id

Nomor : 71/UN.34.32/TU/2013
Lampiran : -
Hal : **Ijin Penelitian An. Omar Muhammad AA.**

Kepada : Yth. Wakil Dekan I Fakultas Teknik
Universitas Negeri Yogyakarta
di Yogyakarta

Dengan hormat,
Berdasarkan surat dari Wakil Dekan I FMIPA UNY Nomor : 2297/UN.34.13/PL/2013, Tentang : Permohonan ijin penelitian terkait dengan permintaan data untuk keperluan Tugas Akhir Skripsi Mahasiswa :

Nama	: Omar Muhammad Altoumi Alsyaibani
NIM	: 09520244085
Jurusan/Prodi	: Pendidikan Teknik Informatika Universitas Negeri Yogyakarta
Judul Skripsi	: **Performa Algoritma Load Balance Pada Server Web Apache dan Nginx Dengan Data Base.**
Area Penelitian	: Layanan Internet Mahasiswa UNY (LIMUNY)

Dengan ini kami **mengijinkan yang bersangkutan untuk melaksanakan penelitian**, pada Deskripsi dan Area Penelitian yang sudah ditentukan, dengan catatan agar tidak melakukan penelitian pengukuran performa Akses Sistem Informasi (SI) dalam waktu dekat ini, dikarenakan saat ini di lingkungan UNY sedang dilakukan pembenahan menyeluruh terhadap Sistem Informasi, sehingga pengukuran pada saat ini tidak akan mencerminkan performa yang sebenarnya.

Demikian surat ini agar dapat digunakan sebagaimana mestinya.

Yogyakarta, 19 Septemeber 2013
UPT Pusat Komputer
Kepala,

Dr. Eko Marpanaji.
NIP. 19670608 199303 1 001

Tembusan Yth. :
1. Wakil Rektor I (sebagai laporan)
2. Wakil Dekan I FT
3. Yang Bersangkutan