ITS
Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR - KI141502

PERANCANGAN SISTEM PEMBAGIAN BEBAN PADA BASIS DATA *MULTI-MASTER* TERDISTRIBUSI UNTUK *MASSIVE DATA TRANSACTION*

MUHAMMAD SYAIFUL JIHAD AMRULLOH
NRP 5113 100 022

Dosen Pembimbing I
Royyana Muslim Ijtihadie, S.Kom, M.Kom, Ph.D

Dosen Pembimbing II
Bagus Jati Santoso, S.Kom, Ph.D

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2017

(Halaman ini sengaja dikosongkan)

TUGAS AKHIR - KI141502

PERANCANGAN SISTEM PEMBAGIAN BEBAN PADA BASIS DATA *MULTI-MASTER* TERDISTRIBUSI UNTUK *MASSIVE DATA TRANSACTION*

MUHAMMAD SYAIFUL JIHAD AMRULLOH
NRP 5113 100 022

Dosen Pembimbing I
Royyana Muslim Ijtihadie, S.Kom, M.Kom, Ph.D

Dosen Pembimbing II
Bagus Jati Santoso, S.Kom, Ph.D

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2017

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESIS - KI141502

DESIGNING LOAD BALANCER ON DISTRIBUTED MULTI-MASTER DATABASE SYSTEM FOR MASSIVE DATA TRANSACTION

MUHAMMAD SYAIFUL JIHAD AMRULLOH
NRP 5113 100 022

Supervisor I
Royyana Muslim Ijtihadie, S.Kom, M.Kom, Ph.D

Supervisor II
Bagus Jati Santoso, S.Kom, Ph.D

Department of INFORMATICS
Faculty of Information Technology
Institut Teknologi Sepuluh Nopember
Surabaya, 2017

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

PERANCANGAN SISTEM PEMBAGIAN BEBAN PADA BASIS DATA *MULTI-MASTER* TERDISTRIBUSI UNTUK *MASSIVE DATA TRANSACTION*

TUGAS AKHIR
Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Arsitektur dan Jaringan Komputer
Program Studi S1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh :
MUHAMMAD SYAIFUL JIHAD AMRULLOH
NRP: 5113 100 022

Disetujui oleh Dosen Pembimbing Tugas Akhir :

Royyana Muslim Ijtihadie, S.Kom, M.Kom, Ph.D ..... (Pembimbing 1)
NIP: 197708242006041001

Bagus Jati Santoso, S.Kom, Ph.D ..... (Pembimbing 2)
NIP: 051100116

SURABAYA
JUNI 2017

(Halaman ini sengaja dikosongkan)

PERANCANGAN SISTEM PEMBAGIAN BEBAN PADA BASIS DATA *MULTI-MASTER* TERDISTRIBUSI UNTUK *MASSIVE DATA TRANSACTION*

Nama	:	**MUHAMMAD SYAIFUL JIHAD AMRULLOH**
NRP	:	**5113 100 022**
Jurusan	:	**Teknik Informatika FTIf**
Pembimbing I	:	**Royyana Muslim Ijtihadie, S.Kom, M.Kom, Ph.D**
Pembimbing II	:	**Bagus Jati Santoso, S.Kom, Ph.D**

Abstrak

Semakin berkembangnya teknologi informasi menuntut semakin banyaknya penggunaan perangkat berupa komputer atau perangkat jaringan. Salah satu pemanfaatan dari teknologi tersebut adalah penggunaan website untuk menangani perekrutan anggota baru. Proses tersebut memerlukan kemampuan penulisan dan pengambilan data yang cepat dan dapat mengangani banyak permintaan dalam satu waktu.

Metode yang saat banyak digunakan untuk melakukan penyimpanan data adalah master-slave. Metode tersebut menggunakan sebuah master node yang memungkinkan untuk melakukan penulisan data. Sedangkan beberapa Slave node akan menyalin data dari master node dan melayani pembacaan data.

Kelemahan dari metode master-slave adalah saat terjadi penulisan data yang besar secara bersamaan. Hal tersebut mengakibatkan bottleneck-effect pada master node. Untuk menanggulangi masalah tersebut maka digunakan sebuah metode master-master dimana semua node memiliki kemampuan untuk menulis dan membaca data. Setiap node akan melakukan

sinkronisasi agar data setiap node tetap sama.

*Topologi yang digunakan untuk mengimplementasikan metode master-master membutuhkan lebih dari satu node agar bekerja dengan baik. Oleh karena itu dibutuhkan sebuah sekema pembagian beban yang bertujuan menentukan node mana yang menerima proses. Algoritma yang dipakai untuk menentukan tujuan pembagian tersebut menggunakan top-*k*. Algoritma tersebut menggunaan prosessor dan memory sebagai pertimbangan untuk menentukan node tujuan.*

***Kata-Kunci**: multi-master, pembagian beban, terdistribusi, algoritma top-k.*

DESIGNING LOAD BALANCER ON DISTRIBUTED MULTI-MASTER DATABASE SYSTEM FOR MASSIVE DATA TRANSACTION

Name	:	**MUHAMMAD SYAIFUL JIHAD AMRULLOH**
NRP	:	**5113 100 022**
Major	:	**Informatics FTIf**
Supervisor I	:	**Royyana Muslim Ijtihadie, S.Kom, M.Kom, Ph.D**
Supervisor II	:	**Bagus Jati Santoso, S.Kom, Ph.D**

Abstract

The rapid development of information technology leads to the considerable number of devices usage, both in computers or network devices. One of the utilization of such technology is the utilization of website to handle the recruitment of new members. The process requires fast writing and retrieval capabilities along with multiple request handling capabiliy at a time.

The most widely used method of data storage is master-slave. The method uses a master node that makes it possible to perform data writing, while some slave nodes will copy data from the master node and serve data readings.

The disadvantage of the master-slave method is when there is large data writing tasks simultaneously, they will cause bottleneck effect on the master node. To overcome this problem, master-master method are proposed, where all nodes have the ability to write and read data. Each node will synchronize the data for each node to remain the same state of the data.

In this work, we implement master-master method that requires more than one node to work properly. Therefore it takes a load-sharing scheme that aims to determine which nodes are

accept process. This work employs top-k technique to handle the division task around jobs. Furthermore, processor and memory information are considered by the algorithm in determining the destination node.
***Keywords**: multi-master, load distribution, distributed computing, top-k algorithm*

KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Alhamdulillahirabbil'alamin, segala puji bagi Allah SWT, yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul **Perancangan Sistem Pembagian Beban pada Basis Data *multi-master* Terdistribusi untuk Massive Data Transaction**. Pengerjaan Tugas Akhir ini merupakan suatu kesempatan yang sangat baik bagi penulis. Dengan pengerjaan Tugas Akhir ini, penulis bisa belajar lebih banyak untuk memperdalam dan meningkatkan apa yang telah didapatkan penulis selama menempuh perkuliahan di Teknik Informatika ITS. Dengan Tugas Akhir ini penulis juga dapat menghasilkan suatu implementasi dari apa yang telah penulis pelajari. Selesainya Tugas Akhir ini tidak lepas dari bantuan dan dukungan beberapa pihak. Sehingga pada kesempatan ini penulis mengucapkan syukur dan terima kasih kepada:

1. Allah SWT yang telah memberikan rahmat, pertolongan, dan anugerah yang tidak terkira serta Nabi Muhammad SAW selaku suri tauladan yang baik bagi manusia.
2. Ayah, Ibu, dan adik penulis yang telah memberikan dukungan moral dan material serta doa yang tak terhingga untuk penulis. Serta selalu memberi semangat dan dorongan untuk segera menyelesaikan pengerjaan Tugas Akhir ini.
3. Bapak Royyana Muslim Ijtihadie, S.Kom, M.Kom, Ph.D. selaku pembimbing I yang telah membantu, membimbing, dan memotivasi penulis mulai dari pengerjaan proposal hingga terselesaikannya Tugas Akhir ini.
4. Bagus Jati Santoso, S.Kom., Ph.D. selaku pembimbing II yang juga telah membantu, membimbing, dan memotivasi penulis mulai dari pengerjaan proposal hingga

terselesaikannya Tugas Akhir ini.

5. Bapak Darlis Herumurti, S.Kom, M.Kom, selaku Kepala Jurusan Teknik Informatika ITS saat ini, Bapak Radityo Anggoro, S.Kom, M.Sc, selaku koordinator TA, Bapak Imam Kuswardayan, S.Kom, MT, dan segenap dosen Teknik Informatika yang telah memberikan ilmu dan pengalamannya.
6. Teman-teman Laboratorium AJK Mas Thiar, Wicak, Zaza, Nindy, Daniel, Setiyo, Risma, Oing, Syukron, Fatih, Afif, Vivi, Bebet, Fuad, Awan, Satria yang selalu menghibur dan mendukung penulis dalam pengerjaan Tugas Akhir ini.
7. Segenap Keluarga Mahasiswa Klaten di Surabaya yang selalu ada dan memberikan motivasi kepada penulis.
8. EMAPAL XXV, Nia, Navi, Dita, Novelia, Aan, Danang, Evan, Dhika, Juan, Toni, Faris, Vando, Nopal, Seluruh anggota EMAPAL dan EMAPAL CORSICA yang telah memberikan banyak pelajaran hidup kepada penulis.
9. Teman-teman *God Bless You* yang selalu memberikan hiburan, canda, dan tawa kepada penulis.
10. Serta semua pihak yang telah turut membantu penulis dalam menyelesaikan Tugas Akhir ini.

Penulis menyadari bahwa Tugas Akhir ini masih memiliki banyak kekurangan. Sehingga dengan kerendahan hati, penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan ke depannya.

Surabaya, Juni 2017

Muhammad Syaiful Jihad Amrulloh

DAFTAR ISI

(Halaman ini sengaja dikosongkan)

DAFTAR TABEL

(Halaman ini sengaja dikosongkan)

DAFTAR GAMBAR

(Halaman ini sengaja dikosongkan)

DAFTAR KODE SUMBER

(Halaman ini sengaja dikosongkan)

BAB 1

PENDAHULUAN

Pada bab ini akan dipaparkan mengenai garis besar Tugas Akhir yang meliputi latar belakang, tujuan, rumusan masalah, batasan permasalahan, metodologi pembuatan Tugas Akhir, dan sistematika penulisan.

1.1 Latar Belakang

Semakin berkembangnya teknologi informasi menuntut semakin banyaknya penggunaan perangkat berupa komputer atau perangkat jaringan. Salah satu pemanfaatan dari teknologi tersebut adalah penggunaan *website* untuk menangani perekrutan anggota baru. Proses tersebut memerlukan kemampuan penulisan dan pengambilan data yang cepat dan dapat mengangani banyak permintaan dalam satu waktu.

Metode yang saat banyak digunakan untuk melakukan penyimpanan data adalah *master-slave*. Metode tersebut menggunakan sebuah *master node* yang memungkinkan untuk melakukan penulisan data. Sedangkan beberapa *slave node* akan menyalin data dari *master node* dan melayani pembacaan data.

Kelemahan dari metode *master-slave* adalah saat terjadi penulisan data yang besar secara bersamaan. Hal tersebut mengakibatkan *bottleneck-effect* pada *master node*. Untuk menanggulangi masalah tersebut maka digunakan sebuah metode *master-master* dimana semua *node* memiliki kemampuan untuk menulis dan membaca data. Setiap *node* akan melakukan sinkronisasi agar data setiap *node* tetap sama.

1.2 Rumusan Masalah

Berikut beberapa hal yang menjadi rumusan masalah dalam tugas akhir ini:

1. Bagaimana mengimplementasikan metode *multi-master* dengan algoritma top-*k* sebagai pembagi beban ?
2. Bagaimana performa algoritma top-*k* dengan dibandingkan dengan algoritma yang lain ?
3. Bagaimana pengaruh perbandingan rasio CPU dan *memory* serta jumlah *worker* terhadap kinerja sistem ?
4. Bagaimana perbandingan CPU dan *memory* dari sistem yang baru dengan sistem yang telah ada ?

1.3 Batasan Masalah

Dari permasalahan yang telah diuraikan di atas, terdapat beberapa batasan masalah pada tugas akhir ini, yaitu:

1. Database yang digunakan menggunakan RDBMS (*Relational Database Management System*) dengan MariaDB.
2. Penggunaan rasio perbandingan CPU dan *memory* ditentukan secara manual oleh pengguna.
3. Semua server *worker* memiliki spesifikasi yang identik.
4. Sistem akan diuji coba dengan menggunakan aplikasi PPDB Surabaya tahun 2015 dan dibandingkan dengan Nginx sebagai *load balancer*.

1.4 Tujuan

Tugas akhir dibuat dengan beberapa tujuan. Berikut beberapa tujuan dari pembuatan tugas akhir:

1. Mengimplementasikan arsitektur sistem dengan metode *multi-master* dengan pembagian beban untuk menangani transaksi yang besar.
2. Membandingkan algoritma pembagian beban dengan algoritma top-*k* dengan algoritma yang sudah ada.

1.5 Manfaat

Dengan dibangunnya sistem penyimpanan data *multi-master* terdistribusi dengan pembagian beban ini diharapkan dapat menanggulangi masalah penulisan data yang terpusat pada metode *master-slave*. Selain itu sistem juga diharapkan dapat membagi beban dengan lebih baik dari algoritma yang sudah ada.

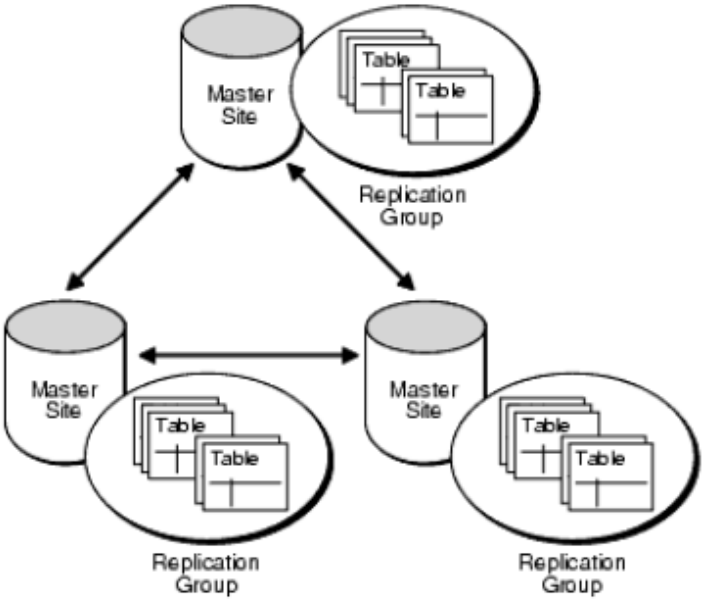
(Halaman ini sengaja dikosongkan)

BAB 2

LANDASAN TEORI

2.1 *Multi-master Replication*

Multi-master merupakan sebuah metode replikasi dimana setiap *node* bertindak sebagai *master* dan juga replika pada saat yang bersamaan. Setiap *node* pada klaster tersebut memiliki kemampuan untuk menulis dan membaca data secara bersamaan. Setiap *node* pada klaster memiliki state yang sama untuk mengontrol dan melakukan penyalinan data. Metode tersebut juga memungkinkan setiap *node* melakukan sinkronisasi antar *node*. Proses sinkronisasi dilakukan dengan membuat sebuat hubungan antara *node* satu dengan *node* lain seperti pada Gambar 2.1 sehingga didapat data yang konsisten di semua *node* [1][2].



Gambar 2.1: Replikasi Multi-Master [11]

2.2 Top-*k*

Merupakan algoritma yang digunakan untuk menentukan dominasi data pada suatu dataset. Algoritma tersebut akan melakukan perhitungan dari suatu variabel dengan bobot tertentu. Setelah dilakukan perhitungan maka data tersebut diurutkan sesuai perhitungan yang telah didapat. Hasil dari algoritma tersebut adalah data yang mendominasi sejumlah *k*. Dataset akan diperbarui dengan kurun waktu tertentu dan dilakukan penghitugan ulang. [3].

Algoritma ini akan digunakan sebagai penentu *node* yang akan melakukan pekerjaan. *Worker* dengan beban kerja yang paling ringan akan dipilih untuk melayani permintaan yang ada. Parameter yang digunakan dalam rangcang bangun ini adalah persentase beban CPU dan persentase beban *memory*.

2.3 MariaDB

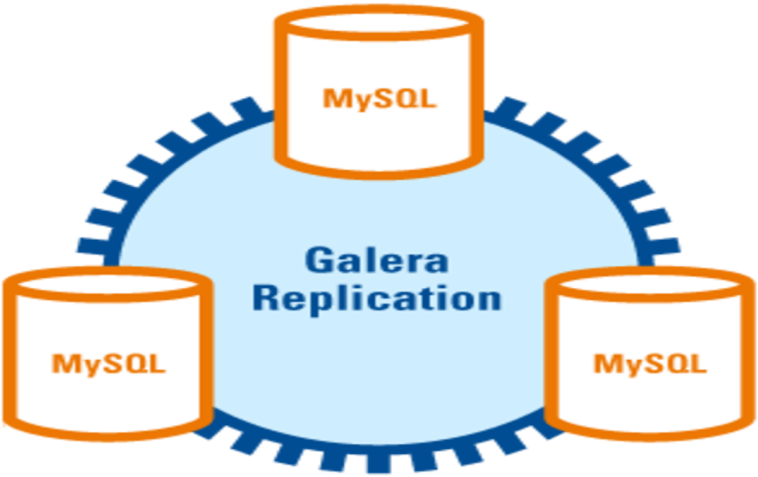
MariaDB merupakan sebuah aplikasi basis data sumber terbuka yang dikembangankan oleh pengembang asli dari MySQL. Aplikasi tersebut menggunakan tipe *Relational Database Management System* (RDBMS). Basis data tersebut menggunakan tabel dengan baris dan kolom untuk menyimpan data. Setiap tabel dapat berhubungan dengan tabel yang lain tanpa harus mengganti susunan data dari tabel yang sudah ada. RDBMS biasanya menggunakan *Structured Query Language* (SQL) untuk melakukan penyimpanan, pengambilan, dan penyuntingan data. Pada aplikasi MariaDB, data yang disimpan diubah dalam bentuk *wide-array* dan dapat digunakan untuk menyimpan berbagai macam informasi [4].

MariaDB memiliki kemampuan yang baik dalam menangani penyimpanan data dan mudah digunakan karena didukung dengan berbagai macam aplikasi manajemen basis data. Selain

itu aplikasi tersebut sudah mendukung replikasi dengan metode *master-master* seperti yang sudah dijelaskan pada subbab 2.1.

2.4 Galera Cluster

Galera Cluster merupakan sebuah *plugin opensource* aplikasi basis data yang mendukung replikasi *synchronous* untuk basis data bertipe *multi-master*. Galera Cluster memiliki kelebihan mudah digunakan, ketersediaan tinggi, dan juga konsistensi data untuk pengembangan lebih lanjut.



Gambar 2.2: Cluster Galera Cluster [5]

Galera Cluster membantu dalam menyediakan implementasi basis data terdistribusi dengan metode *multi-master* dengan *wsrep API* seperti Gambar 2.2. Selain itu Galera Cluster juga sudah memiliki dukungan penuh pada aplikasi MariaDB sehingga mudah untuk dikonfirugasi [5]. Beberapa fitur yang diunggulkan oleh Galera Cluster antara lain:

1. *True multi-master*
   Memungkinkan semua *node* melakukan penulisan dan pembacaan data pada waktu yang bersamaan.
2. *Synchronous replication*
   Tidak ada keterlambatan propagasi sehingga tidak ada kehilangan data saat terjadi kesalahan pada *node*.
3. *Tightly coupled*
   Semua *node* memiliki keadaan yang sama, sehingga tidak terjadi penyimpangan data yang terjadi antar *node*.
4. *Automatic node joining*
   Memudahkan penambahan *node* pada *cluster*.
5. *True parallel replication*
   Replikasi yang dijalankan berada pada *row-level* sehingga setiap row yang ada dan juga konfigurasi datanya terjaga.
6. *Support InnoDB*
   Mendukung aplikasi basis data yang berjalan dengan mesin InnoDB.

2.5 CollectD

CollectD merupakan aplikasi yang dapat mengumpulkan data *metric* dari sistem operasi, aplikasi, *logfile*, dan juga perangkat eksternal yang digunakan. Aplikasi tersebut membantu dalam memantau kinerja pada perangkat server. Aplikasi CollectD mempunyai berbagai macam plugin yang dapat digunakan untuk melakukan *monitoring* sebuah server. Salah satu plugin yang didukung oleh CollectD adalah pengiriman data melalui jaringan. Data yang telah didapat selanjutnya disimpan pada aplikasi InfluxDB. CollectD berada dibawah lisensi sumber terbuka dan dapat digunakan tanpa biaya. Aplikasi tersebut ditulis dengan bahasa C, sehingga memberikan performa yang baik serta mendukung berbagai sistem operasi tanpa perlu ada *scripting* untuk beroperasi dengan baik [6].

2.6 InfluxDB

InfluxDB merupakan sebuah aplikasi yang berada dibawah lisensi kode sumber terbuka sehingga dapat digunakan dan dimodifikasi dengan bebas. Aplikasi tersebut merupakan sebuah basis data yang berbasis *time series*, dimana penyimpanan data didasarkan pada sebuah satuan waktu yang kontinyu. Basis data tersebut memiliki performa yang baik untuk menyimpan data yang besar dan *realtime*, seperti data sensor, data operasi monitoring, dan juga *real-time analytics*. InfluxDB ditulis dengan bahasa Go dan mendukung penyimpanan data dalam jaringan [7].

2.7 Golang

Golang adalah bahasa pemrogramman dengan lisensi sumber terbuka yang dikembangkan oleh Google. Golang termasuk bahasa *statically typed* seperti C. Golang akan di kompilasi menjadi bahasa sistem terlebih dahulu, sehingga performa bahasa Golang lebih baik dan cepat dari pada bahasa yang menggunakan *interpreter* seperti php, ruby dan python. Walaupun Golang termasuk bahasa *statically typed*, Golang sudah mendukung fitur-fitur seperti pada bahasa pemgrogramman yang lebih maju. Golang mendukung fitur-fitur antara lain *garbage collection*, *memory safety*, dan *concurrent programming*. Selain itu Golang memiliki dukungan komunitas yang besar dan memiliki banyak *plugin* yang dapat digunakan dengan mudah [8].

2.8 Apache JMeter

Menjadi salah satu alat bantu untuk melakukan pengujian beban dan mengukur performa aplikasi, salah satunya berbasis web. Aplikasi Apache JMeter mampu melakukan pengujian pada berbagai macam protokol diantaranya Web (HTTP), FTP, Basis

Data, dan Mail (SMTP,IMAP,POP3). Apache JMeter dibangun berbasis Java dengan cara kerja meniru sebuah peramban web [9].

2.9 Grafana

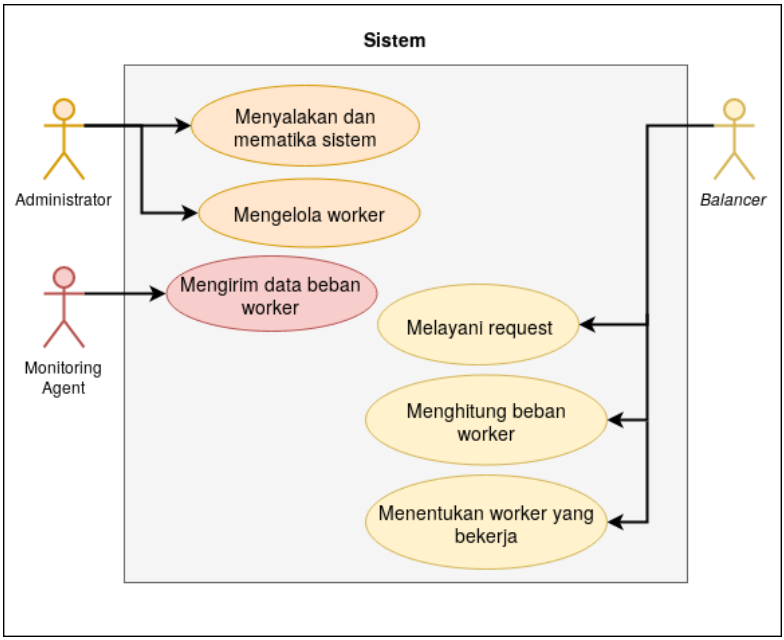
Grafana merupakan sebuah aplikasi sumber terbuka yang digunakan untuk visualisasi data dan analisis matrik. Grafana biasanya digunakan untuk menampilkan data dengan basis waktu seperti data beban sebuah server, data produksi industri, dan juga data sensor. Grafana dapat menampilkan menampilkan data dari sebuah basis data, contohnya InfluxDB [10].

BAB 3

DESAIN DAN PERANCANGAN

3.1 Kasus Penggunaan

Terdapat tiga aktor dalam sistem yaitu Administrator, *Monitoring agent*, dan *Balancer*. Administrator adalah aktor yang mengoperasikan sistem dan mengelola *worker*, *Monitoring agent* adalah pengambil data dari keadaan beban yang ada di *worker*, sedangkan *Balancer* adalah aktor yang menghitung beban worker dan menentukan *worker* mana saja yang akan menangani permintaan. Diagram kasus penggunaan menggambarkan kebutuhan-kebutuhan yang harus dipenuhi sistem. Diagram kasus penggunaan digambarkan pada Gambar 3.1.



Gambar 3.1: Digram Kasus Penggunaan

Digram kasus penggunaan pada Gambar 3.1 dideskripsikan masing-masing pada Tabel 3.1.

Tabel 3.1: Daftar Kode Kasus Penggunaan

Kode Kasus Penggunaan	**Nama Kasus Penggunaan**	**Keterangan**
UC-0001	Menyalakan dan mematikan sistem.	Administrator memiliki kendali atas aktif atau tidaknya sistem.
UC-0002	Mengelola *worker*.	Administrator dapat menambahkan dan mengurangi jumlah *worker* dalam klaster.
UC-0003	Mengirim data beban *worker*.	*Monitoring agent* mengirim data beban CPU dan beban *memory* pada basis data.
UC-0004	Melayani *request*.	*Balancer* akan meneruskan permintaan kepada *worker* terpilih dan mengembalikan balasan.
UC-0005	Menghitung beban *worker*.	*Balancer* akan menghitung beban dari semua *worker* dalam klaster.

Tabel 3.1: Daftar Kode Kasus Penggunaan

Kode Kasus Penggunaan	Nama Kasus Penggunaan	Keterangan
UC-0006	Menentukan *worker* yang bekerja.	*Balancer* akan melakukan pengurutan beban dan memilih *worker* mana saja yang akan bekerja.

3.2 Arsitektur Sistem

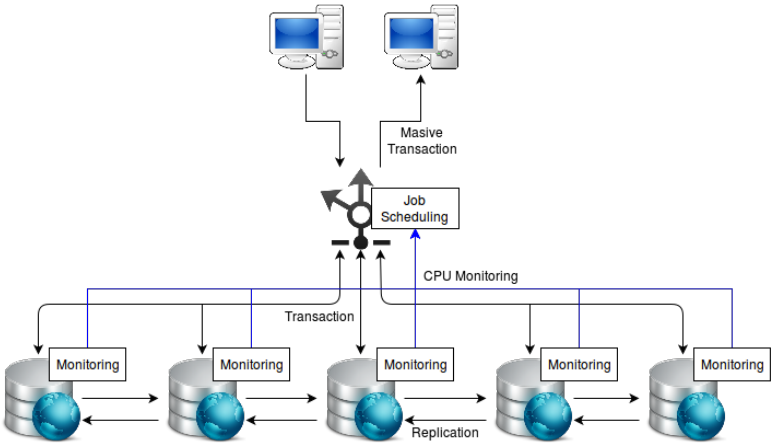
Pada sub-bab ini, dibahas mengenai tahap analisis dan desain dari sistem yang akan dibangun. Pembahasan akan dilakukan pada tiap-tiap komponen pembangun sistem yaitu: *balancer*, *monitoring agent*, penyimpanan data.

3.2.1 Desain Umum Sistem

Sistem pembagian beban yang akan dibangun mempunyai bebearapa bagian besar yang menjadi pembangun utama dari sistem tersebut. Bagian tersebut antara lain:

1. *Worker*
   Worker berguna sebagai pekerja yang melayani seluruh permintaan dari pengguna yang diteruskan oleh *balancer*. Didalam setiap *worker* terdapat basis data dan web server.
2. *Monitoring agent*
   Monitoring agent adalah aplikasi yang mengambil dan mengumpulkan beban kerja CPU dan *memory* pada tiap-tiap *worker*.
3. *Balancer*
   Balancer akan membagi tugas operasi terhadap basis data dari hasil perhitungan algoritma top-*k*.

Arsitektur sistem secara umum digambarkan pada diagram arsitektur 3.2.



Gambar 3.2: Desain Sistem Secara Umum

3.2.2 Desain *Worker*

Worker bekerja sebagai penyedia layanan. Menunggu permintaan dari balancer dan kemudian akan dikembalikan lagi pada *balancer*. Aplikasi akan ditempatakan pada worker. Pada worker terdapat beberapa komponen sebagai berikut:

1. Penyimpanan data

   Penyimpanan data merupakan klaster basis data yang akan digunakan oleh aplikasi yang berjalan pada *worker* untuk menyimpan data. Penyimpanan data yang digunakan merupakan sebuah cluster MariaDB yang telah terintegrasi dengan bantuan Galera Cluster. Klaster tersebut menggunakan replikasi *multi-master* sehingga setiap *node* memiliki salinan data yang sama. Ketika diperlukan *node* tambahan untuk memenuhi kebutuhan maka *node* tersebut akan mengkopi semua konfigurasi dari klaster. Transaksi

penyimpanan dan pengambilan data menggunakan SQL (*Structured Query Language*) yang sering digunakan untuk basis data. Digram arsitektur Penyimpanan Data tertera pada Gambar 3.3.

Gambar 3.3: Desain Arsitektur Penyimpanan Data

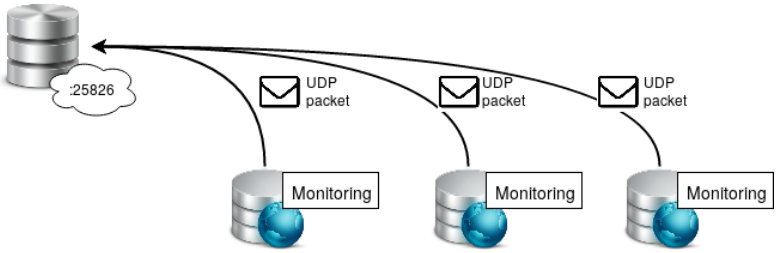
2. *Web server*

   Web server adalah penyedia layanan dari aplikasi yang berbasis web. Permintaan dari pengguna akan diterima dan diolah oleh *web server*. Pada sistem ini aplikasi yang digunakan sebagai *web server* adalah Nginx.

3.2.3 Desain *Monitoring Agent*

Monitoring agent adalah sebuah aplikasi yang digunakan untuk membaca penggunaan sebuah sumber daya pada suatu *worker*. Data yang didapat dari aplikasi tersebut dapat merepresentasikan seberapa sibuk sebuah *worker*. Aplikasi tersebut akan membaca persentase penggunaan CPU dan *memory* dari setiap *worker* lalu menyimpan data tersebut pada penyimpanan data.

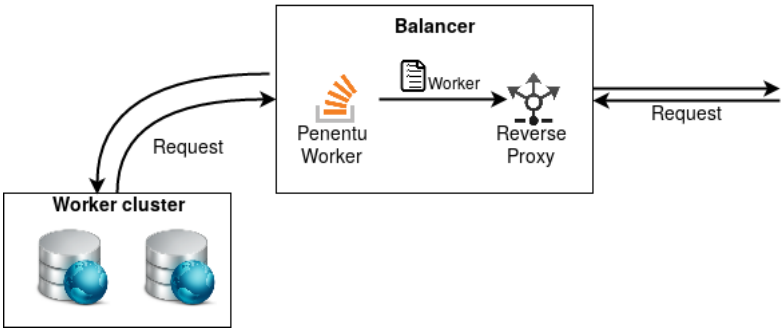
Pada sistem ini *monitoring agent* akan dibangun dengan menggunakan aplikasi CollectD. Aplikasi tersebut mendukung pengiriman data lewat jaringan dengan cara mengirimkan paket UDP. Pada sistem ini paket tersebut akan dikirim kepada basis data InfluxDB melalui *port* 25826. Diagram arsitektur *monitoring agent* tertera pada gambar 3.4.



Gambar 3.4: Desain Arsitektur *Monitoring Agent*

3.2.4 Desain *Balancer*

Balancer berperan penting dalam sistem. Setiap permintaan yang masuk akan diproses oleh balancer dan dikembalikan ke pengguna setelah mendapat balasan dari *worker*. Pada sistem ini balancer akan dibangun dengan bahasa Go. Terdapat dua komponen utama penyusun *balancer*, yaitu: *reverse proxy* dan penentu *worker*. Diagram arsitektur *balancer* tertera pada gambar 3.5.



Gambar 3.5: Desain Arsitektur Penyeimbang Beban

Reverse proxy akan melayani permintaan yang berasal dari client dan meneruskannya ke *worker*. Balasan dari *worker*

kemudian akan dikembalikan lagi ke client. *Reverse proxy* akan berjalan pada *port* 80.

Penentu *worker* akan menentukan *worker* mana yang akan memproses permintaan. Data yang akan diambil dari sebuah aplikasi InfluxDB yang menyimpan seluruh data beben *worker*. Parameter yang digunakan menentukan adalah beban CPU dan *memory* dari tiap *worker*. Data tersebut ditentukan dengan menggunakan algoritma top-*k*. Algoritma tersebut akan berjalan secara periodik dan melakukan penghitungan ulang untuk menentukan *worker* yang selanjutnya.

(Halaman ini sengaja dikosongkan)

BAB 4

IMPLEMENTASI

Bab ini membahas implementasi sistem pembagian beban secara rinci. Pembahasan dilakukan secara rinci untuk setiap komponen yang ada yaitu: *balancer*, penyimpanan data, dan *monitoring agent*. Implentasi dilakukan pada lingkungan komputer fisik dan virtual. Penjelasan lingkungan implementasi akan dijelaskan kemudian.

4.1 Lingkungan Implementasi

Lingkungan implementasi dan pengembangan dilakukan menggunakan virtualisasi Vagrant yang berjalan pada sebuah server dengan spesifikasi Intel(R) Xeon(R) CPU E3-1220 V2 @ 3.10GHz dengan memori 16 GB dan sebuah server Proxmox yang berjalan pada komputer *host* dengan spesifikasi Intel(R) Core(TM) i3-2120 CPU @ 3.20GHz dengan memori 8 GB. Semua server berada di Laboratorium Arsitektur dan Jaringan Komputer, Teknik Informatika ITS dan terhubung dengan jaringan tersebut. Perangkat lunak yang digunakan dalam pengembangan adalah sebagai berikut :

- Sistem Operasi Proxmox versi 4.4-1/eb2d6f1e
- Vagrant versi 1.8.1
- VirtualBox
- Sistem Operasi Linux Ubuntu Server 14.04.2 LTS (*Worker*)
- Sistem Operasi Linux Ubuntu Server 16.04.1 LTS (*Balancer*)
- Golang versi 1.8.1
- Git versi 2.21.2 untuk pengolahan versi program
- MariaDB Galera Cluster versi 10.1.30
- Apache JMeter untuk melakukan pengujian sistem
- Postman Chrome Plugin untuk melakukan pengujian fungsional sistem.

4.2 Implementasi *Worker*

Untuk melayani semua permintaan digunakan sebuah aplikasi web server Nginx yang berjalan dalam sistem operasi Ubuntu Server 14.04. Agar dapat melayani halaman dinamis berbasis PHP, maka diperlukan aplikasi lain yaitu php-fpm. Selain itu pada worker terdapat sebuah aplikasi CollectD yang dapat mengambil penggunaan sumber daya untuk disimpan.

4.2.1 Instalasi dan Konfigurasi Web Server

Untuk melakukan instalasi dan konfigurasi web server Nginx terdapat langkah-langkah sebagai berikut:

- Instalasi
  `sudo apt-get install nginx`
  Perintah diatas akan melakukan instalasi web server keladam sistem operasi Ubuntu.
- Konfigurasi
  Berkas konfigurasi dari Nginx dapat ditemukan pada direktori `/etc/nginx/sites-available/default`. Agar dapat melayani permintaan web dinamis seperti PHP maka Nginx harus ditambahkan baris konfigurasi seperti dibawah:

```
1  server {
2    ...
3    index index.php index.html index.htm;
4
5    location / {
6      ...
7      try_files $uri $uri/ /index.php;
8      fastcgi_split_path_info ^(.+\.php)(/.+)$;
9      fastcgi_param SCRIPT_FILENAME
           $document_root$fastcgi_script_name;
10     fastcgi_pass unix:/var/run/php5-fpm.sock;
11     fastcgi_index index.php;
12     include fastcgi_params;
```

```
13      ...
14    }
15    ...
16  }
```

Kode Sumber 4.1: Konfigurasi Web Server Nginx

4.2.2 Instalasi dan Konfigurasi php-fpm

PHP-fpm dibutuhkan agar *worker* dapat menjalankan aplikasi yang berbasis php. Langkah-langkah untuk melakukan instalasi dan konfigurasi php-fpm adalah sebagai berikut:

- Installasi
  `sudo apt-get install php5-fpm php5-mysql`
  Perintah diatas akan melakukan instalasi php-fpm kedalam sistem operasi. Selain itu diperlukan sebuah aplikasi tambahan berupa php5-mysql agar aplikasi berbasis php dapat berkomunikasi dengan basis data MariaDB.
- Konfigurasi
  Berkas konfigurasi dari php-fpm terdapat pada direktori `/etc/php5/fpm/php.ini`. Agar aplikasi tersebut dapat dijalankan oleh web server maka perlu mengubah baris `;cgi.fix_pathinfo=1` menjadi `cgi.fix_pathinfo=0`.

4.2.3 Instalasi dan Konfigurasi Basis Data

Agar *worker* dapat menyimpan data dari aplikasi, maka diperlukan sebuah aplikasi basis data. Pada sistem digunakan basis data MariaDB versi 10.1.30. Untuk melakukan sinkronisasi data antar basis data, maka pada sistem ini diperlukan juga aplikasi Galera Cluster. Untuk melakukan instalasi dan konfigurasi aplikasi tersebut, langkah-langkahnya adalah sebagai berikut:

- Instalasi
  `sudo add-apt-repository 'deb`

`[arch=amd64,i386] http://mariadb.biz.net.id/repo/10.1/ubuntu trusty main'`

Perintah diatas digunakan untuk menambahkan repository dari MariaDB versi 10.1.30 kedalam daftar repository sistem operasi.

`apt-get install mariadb-server`

Perintah tersebut untuk melakukan instalasi dari aplikasi MariaDB versi 10.1.30.

- Konfigurasi

  Berkas konfigurasi dari aplikasi MariaDB versi 10.1.30 berada pada direktori `/etc/mysql/conf.d`. MariaDB versi 10.1.30 sudah terpasang aplikasi Galera Cluster sehingga kita hanya perlu melakukan konfigurasi dengan menambahkan sebuah file `galera.cnf`. Konfigurasi dapat dilakukan seperti dibawah ini:

```
1  [mysqld]
2    binlog_format=ROW
3    default-storage-engine=innodb
4    innodb_autoinc_lock_mode=2
5    bind-address=0.0.0.0
6
7    wsrep_on=ON
8    wsrep_provider=/usr/lib/galera/libgalera_smm.so
9
10   wsrep_cluster_name="test_cluster"
11   wsrep_cluster_address="gcomm://10.151.36.61,
         10.151.36.62, 10.151.36.63, 10.151.36.65,
         10.151.36.66, 10.151.36.67, 10.151.36.68,
         10.151.36.69, 10.151.36.70, 10.151.36.71"
12
13   wsrep_sst_method=rsync
14   wsrep_node_address="10.151.36.61"
15   wsrep_node_name="mnode-1"
```

Kode Sumber 4.2: Konfigurasi MariaDB Galera Cluster

4.2.4 Instalasi dan Implementasi *Monitoring Agent*

Monitoring agent bertugas untuk melakukan pengawasan terhadap sumberdaya dari *worker*. Pada sistem ini aplikasi yang digunakan adalah CollectD. Untuk melakukan instalasi dan konfigurasi aplikasi tersebut, langkah-langkahnya adalah sebagai berikut:

- Instalasi
  `sudo add-apt-repository ppa:collectd/collectd-5.5`
  Perintah diatas digunakan untuk menambahkan repository CollectD kedalam daftar repository sistem operasi. `sudo apt-get install -y collectd collectd-utils`
  Perintah diatas digunakan untuk melakukan pemasangan aplikasi CollectD kedalam sistem operasi. Dibutuhkan aplikasi tambahan colledctd-utils agar aplikasi tersebut dapat berjalan dengan baik.
- Konfigurasi
  Berkas konfigurasi dari aplikasi berada pada `/etc/collectd/collectd.conf`. Agar dapat membaca persentase CPU, *memory*, dan dapat mengirimkan data melalui jaringan, maka diperlukan konfigurasi seperti dibawah:

```
1  Hostname "mnode-1"
2  FQDNLookup true
3
4  Interval 3
5
6  LoadPlugin syslog
7  LoadPlugin cpu
8  LoadPlugin memory
9  LoadPlugin network
10
11 <Plugin syslog>
12   LogLevel info
13 </Plugin>
```

```
<Plugin cpu>
  ReportByCpu false
  ReportByState false
  ValuesPercentage true
</Plugin>
<Plugin memory>
  ValuesAbsolute false
  ValuesPercentage true
</Plugin>
<Plugin network>
  Server "10.151.36.64" "25826"
</Plugin>

<Include "/etc/collectd/collectd.conf.d">
  Filter "*.conf"
</Include>
```

Kode Sumber 4.3: Konfigurasi CollectD

Aplikasi tersebut akan mengirimkan data beban *worker* melalui jaringan menuju alamat IP `10.151.36.64`. Paket yang dikirimkan berupa paket UPD dengan *port* `25826`. Paket tersebut akan diterima dan disimpan pada basis data InfluxDB untuk pengolahan data yang selanjutnya.

4.3 Rincian Implementasi *Balancer*

Balancer akan dibangun dengan bahasa pemrograman Go, aplikasi InfluxDB, dan algoritma top-k. Aplikasi InfluxDB akan menampung data yang akan diolah dengan algorimta top-k untuk menentukan *worker* tujuan. Data yang didapat kemudian diolah dengan algoritma top-k untuk menentukan *worker* dengan beban kerja paling ringan. Daftar *worker* sejumlah k akan digunakan *balancer* untuk melayani permintaan pengguna.

4.3.1 Instalasi dan Konfigurasi Go Lang

Aplikasi *balancer* yang akan dibangun menggunakan bahasa pemrograman Go. Agar bahasa pemrograman tersebut dapat berjalan dengan baik maka perlu dilakukan instalasi paket Go. Langkah untuk melakukan instalasi dan konfigurasi adalah sebagai berikut:

- Unduh paket Go versi 1.8.1 dari alamat web `https://golang.org/dl/`
- Lakukan ekstraksi paket yang sudah diunduh, kemudian pindahkan berkas hasil ekstraksi kedalam direktori `/usr/local` dengan perintah: `tar -C /usr/local -xzf go1.8.1.linux-amd64.tar.gz`
- Lakukan pengaturan tempat kerja dari bahasa Go dengan mengatur variable GOPATH dengan perintah: `export GOPATH=$HOME/golang`
- Atur *environtment variable* dari sistem operasi agar berkas *binary* dari bahasa Go dapat diakses dari seluruh sistem dengan perintah `export PATH=$PATH:/usr/local/go/bin:$GOPATH/bin`
- Setelah semua konfigurasi selesai, konfirmasi keberhasilan pemasangan dengan perintah `go version`

4.3.2 Instalasi dan Konfigurasi InfluxDB

Sebelum data beban CPU dan *memory* diolah oleh penentu *worker*, maka data tersebut harus disimpan kedalam sebuah basis data. Basis data yang digunakan untuk menyimpan data tersebut adalah InfluxDB. Langkah untuk melakukan instalasi aplikasi tersebut adalah sebagai berikut:

- Tambahkan repository InfluxDB dengan menjalankan perintah: `curl -sL https://repos.influxdata.com/influxdb.key | sudo apt-key add -`

```
source /etc/lsb-release
echo "deb
https://repos.influxdata.com/${DISTRIB_ID,,}
${DISTRIB_CODENAME} stable" | sudo tee
/etc/apt/sources.list.d/influxdb.list
```

- Lakukan instalasi InfluxDB kedalam sistem operasi dengan perintah: `sudo apt-get install influxdb`
- Agar aplikasi InfluxDB dapat menerima data yang dikirim dari CollectD melalui jaringan, maka perlu ditambahkan baris konfigurasi pada berkas `/etc/influxdb/influxdb.conf`. Cari baris yang berisi `[[collectd]]` agar seperti ini:

```
1 [[collectd]]
2   enabled = true
3   bind-address = "0.0.0.0:25826"
4   database = "resource"
5   retention-policy = ""
6   batch-size = 5000
7   batch-pending = 10
8   batch-timeout = "10s"
9   typesdb = "/usr/local/opt/collctd/types.db"
```

Kode Sumber 4.4: Konfigurasi InfluxDB

Baris konfigurasi tersebut akan melakukan *listen* pada *port* `25826` dan menyimpan data pada *database* `resource`

- Buka *command shell* aplikasi InfluxDB dengan perintah `influx`. Kemudian buat sebuah *database* dengan cara `create database resource`.
- Konfigurasi awal dari InfluxDB tidak menyertakan berkas `types.db`. Isi dari berkas tersebut dapat dilihat pada lampiran A.1.

4.3.3 Implementasi Pengambilan Data

Penentuan *worker* yang bekerja pada sistem ini ditentukan oleh data CPU dan *memory* yang disimpan pada basis data InfluxDB. Agar bahasa pemrograman Go dapat melakukan koneksi dan pengambilan data, dibutuhkan pustaka tambahan InfluxDB Client. Pengunduhan pustaka dapat dilakukan dengan perintah `go get github.com/influxdata/influxdb/client/v2`. Pustaka yang berhasil diunduh akan masuk kedalam direktori yang sudah diatur pada GOPATH. Pembangunan pengambilan dilakukan sesuai langkah-langkah berikut:

- Untuk membangun pengambilan data dari InfluxDB diperlukan pustaka InfluxDB Client dan beberapa penunjang yang lain, digunakan perintah *import* sesuai dengan kode sumber B.1.
- Membuat variabel konstan untuk keperluan koneksi dan pemilihan *database* dari aplikasi InfluxDB. Variabel tersebut berisi nama *database*, *username*, kata sandi, dan nomor *port* dari InfluxDB. Pembuatan variabel sesuai dengan kode sumber B.2.
- Untuk menampung data dari seluruh *worker* diperlukan sebuah variabel `Data` yang berupa *struct* untuk menampung alamat IP *worker*, nama *worker*, beban CPU, beban *memory*, dan total beban. Variabel tersebut disimpan dalam variabel `DataSlice` yang bertipe *array*. Pembuatan variabel sesuai dengan kode sumber B.3.
- Melakukan inisiasi koneksi dengan aplikasi InfluxDB dengan membuat fungsi `CreateClient`. Parameter masukan dari fungsi merupakan data dari variabel konstan. Fungsi tersebut akan mengembalikan koneksi yang terbuka setelah berhasil dan pesan kesalahan jika mengalami kegagalan. Kode fungsi sesuai dengan kode sumber B.4.
- Setelah inisiasi koneksi berhasil, untuk melakuakn

pengambilan data dari aplikasi InfluxDB diperlukan fungsi dengan parameter masukan variabel `DataSlice` dan hasil koneksi. Fungsi tersebut akan menjalankan *query* `SELECT last(value) from cpu_value, memory_value where type='percent' and host='[nama_host]'` untuk mengambil beban CPU dan *memory* dari tiap *worker*. Fungsi tersebut akan mengembalikan nilai beban CPU dan *memory* dengan tipe `float64`. Kode fungsi tersebut sesuai dengan kode sumber B.5.

4.3.4 Implementasi Algoritma Top-*k*

Algoritma top-*k* diperlukan untuk menentukan *worker* yang memiliki beban kerja paling sedikit. Parameter yang diperlukan untuk melakukan perhitungan beban *worker* adalah beban CPU dan *memory*. Penentuan beban dari tiap *worker* dihitung dari penjumlahan dari data CPU dan *memory* dikalikan dengan rasio dari beban tersebut. Proses penghitungan sesuai dengan *pseudocode* 4.1.

```
1 for worker range semuaWorker
2     worker.CPU, worker.Memory = Ambil data dari InfluxDB
3     worker.Beban = rasioCPU*worker.CPU + rasioMemory*worker.Memory
4
5 sort(worker) by worker.Beban
6 return worker sejumlah k
```

Gambar 4.1: Pseudocode penghitungan dengan algoritma Top-K

Langkah-langkah proses penghitungan beban sebagai berikut:

- Setiap iterasi, sistem melakukan pengambilan data sesuai dengan kode sumber B.5 hingga semua data dari tiap *worker* disimpan kedalam sebuah *array*.
- Lakukan penghitungan beban tiap *worker*. Penghitungan beban dilakukan dengan melakukan perkalian data CPU dan *memory* dengan masukan rasio beban dari masukan

administrator sistem.

- Setelah didapat semua beban *worker*, maka dilakukan pengurutan *worker* yang paling ringan dengan parameter total beban.
- *Worker* yang sudah diurutkan kemudian diambil sejumlah *k* teratas sebagai nilai kembalian dari fungsi top-*k*.

Implementasi top-*k* secara lengkap sesuai dengan kode sumber B.6.

4.3.5 Implementasi *Reverse Proxy*

Permintaan yang datang dari pengguna akan diteruskan kepada *worker* untuk diproses lebih lanjut. Proses tersebut membutuhkan *reverse proxy* agar dapat berjalan dengan baik. Program akan dibangun dengan bahasa Go dengan tambahan pustaka yang sudah tersedia. Pembangunan *reverse proxy* sesuai dengan langkah-langkah berikut:

- Agar *reverse proxy* dapat meneruskan permintaan pengguna, maka diperlukan pustaka penunjang. Digunakan perintah *import* sesuai dengan kode sumber B.7.
- Setelah pustaka berhasil di impor, diperlukan fungsi untuk meneruskan permintaan dari pengguna. Fungsi tesebut memerlukan parameter masukan antara lain: *array* dari seluruh *worker*, koneksi dengan InfluxDB B.4, rasio CPU, rasio *memory*, dan jumlah *k* dan jumlah round (r). Didalam fungsi tersebut akan menjalankan algoritma top-*k* untuk menentukan *worker*, dan mengembalikan nilai kembalian daftar *worker* yang akan bekerja dan diteruskan ke *director* untuk meneruskan permintaan pengguna ke *worker*. Pembuatan fungsi *reverse proxy* sesuai dengan kode sumber B.8.
- Untuk mendaftarkan semua *worker* yang akan menerima permintaan dari pengguna, semua *worker* didaftarkan

kedalam sebuah *array*. *Array* tersebut akan menyimpan *host*, alamat IP, dan data awal *worker*. Inisiasi daftar *worker* sesuai dengan kode sumber B.9.

4.3.6 Menjalankan *Service Load Balancer*

Sebelum *balancer* dapat digunakan, maka perlu menjalankan layanan yang diperlukan. Perintah untuk menljalankan layanan adalah `./balancer -k [worker] -mem [memory_ratio] -cpu [cpu_ratio] -r [roundtrip]`. Penjelasan dari perintah tersebut:

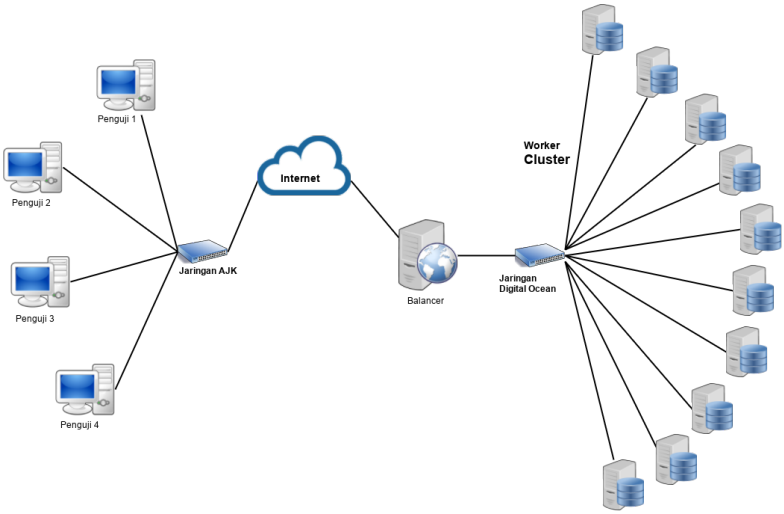
- `-k [worker]`. Adalah jumlah *worker* yang diperlukan untuk melayani permintaan. Nilai awal jika tidak ada parameter adalah 1.
- `-mem [memory_ratio]`. Adalah rasio penggunaan *memory* sebagai perhitungan beban kerja *worker*. Nilai awal jika pengguna tidak menentukan adalah 0.5 (50%).
- `-cpu [cpu_ratio]`. Adalah rasio penggunaan CPU sebagai perhitungan beban kerja *worker*. Nilai awal jika pengguna tidak menentukan adalah 0.5 (50%).
- `-r [roundtrip]`. Adalah jumlah dari permintaan yang diterima setiap *worker* sebelum dilakukan perhitungan ulang dengan algoritma top-*k*. Nilai awal jika pengguna tidak menentukan adalah 10.

BAB 5

PENGUJIAN DAN EVALUASI

5.1 Lingkungan Uji Coba

Lingkungan pengujian menggunakan komponen-komponen yang terdiri dari : satu *droplet* DigitalOcean sebagai *balancer*, 10 *droplet* DigitalOcean sebagai *worker*, dan 4 buah komputer fisik sebagai penguji. Pengujian dilakukan di Laboratorium Arsitektur dan Jaringan Komputer Jurusan Teknik Informatika ITS. Desain arsitektur lingkungan pengujian tertera pada Gambar5.1.



Gambar 5.1: Arsitektur Pengujian *Balancer*

Spesifikasi untuk setiap komponen yang digunakan adalah sebagai berikut:

- *Balancer* :
  - Spesifikasi *droplet*:
    * 2 Core Processor
    * RAM 2 GB
    * 40 GB SSD Disk

    * 3TB Transfer Data
  - Perangkat lunak:
    * Sistem operasi Ubuntu 16.04.02 LTS 64 Bit
    * Nginx
    * Golang versi 1.8.1
    * InfluxDB versi 1.2.2
- *Worker*:
  - Spesifikasi *droplet*:
    * Single Core Processor
    * RAM 1 GB
    * 30 GB SSD Disk
    * 2 TB Transfer Data
  - Perangkat lunak:
    * Sistem operasi Ubuntu 16.04.02 LTS 64 Bit
    * Nginx Web Server
    * CollectD versi 5.5
    * MariaDB Galera Cluster versi 10.1.30
- Komputer Penguji Fungsionalitas:
  - Perangkat Keras:
    * Processor Intel(R) Core(TM) i3-3240 @ 3.40GHz
    * RAM 4096 MB
  - Perangkat Lunak:
    * Sistem operasi OpenSUSE Tumbleweed
    * Postman Chrome
- Komputer Penguji Performa 4 buah:
  - 4 Buah Komputer
    * Perangkat Keras:
      · Processor Intel(R) Core(TM) i3-3240 @ 3.40GHz
      · RAM 4096 MB
    * Perangkat Lunak:
      · Sistem operasi Windows 8 64 Bit
      · Java Version 8

· JMeter versi 3.1

Untuk akses ke masing-masing komponen, dibutuh pembagian alamat IP sesuai yaitu :

- *Balancer* memiliki alamat IP 128.199.114.161
- *Droplet* untuk worker
  - worker 1 memiliki alamat IP 139.59.125.75
  - worker 2 memiliki alamat IP 139.59.125.76
  - worker 3 memiliki alamat IP 139.59.125.76
  - worker 4 memiliki alamat IP 139.59.125.103
  - worker 5 memiliki alamat IP 139.59.125.89
  - worker 6 memiliki alamat IP 139.59.125.85
  - worker 7 memiliki alamat IP 139.59.125.101
  - worker 8 memiliki alamat IP 139.59.125.107
  - worker 9 memiliki alamat IP 139.59.125.95
  - worker 10 memiliki alamat IP 139.59.125.91
  - Grafana *monitoring* dengan alamat IP 139.59.117.72

5.2 Skenario Uji Coba

Uji coba akan dilakukan untuk mengetahui keberhasilan sistem yang telah dibangun. Aplikasi yang dijalankan pada *worker* adalah aplikasi Penerimaan Peserta Didik Baru Kota Surabaya tahun 2015. Skenario pengujian dibedakan menjadi 2 bagian yaitu :

- **Uji Fungsionalitas**.
  Pengujian ini didasarkan pada fungsionalitas yang disajikan sistem. Uji coba yang akan dilakukan adalah uji penentuan *worker* dan pelayanan transaksi dari aplikasi. Uji coba dilakukan untuk mengetahui sistem dapat menjalankan pembagian beban sesuai dengan fungsinya.
- **Uji Performa**.
  Pengujian ini untuk menguji ketahanan sistem terhadap sejumlah permintaan yang masuk. Pengujian dilakukan

dengan melakukan benchmark pada sistem.

5.2.1 Skenario Uji Fungsionalitas

Uji fungsionalitas dibagi menjadi 3, yaitu uji penentuan worker dan uji fungsionalitas pembagian beban.

5.2.1.1 Uji Fungsionalitas Manajemen Sistem

Pengujian manajemen sistem dilakukan dengan menguji fungsionalitas dari sistem. Pengujian ditinjau dari sisi administrator sistem menangani kerja dari balancer tersebut. Pengujian yang dilakukan meliputi menjalankas layanan *balancer*, mengurangi dan menambahkan *worker*, dan mematikan sistem.

5.2.1.2 Uji Fungsionalitas *Monitoring Agent*

Monitoring agent akan melakukan *monitoring* beban kerja dari setiap *worker* dalam klaster. Pengujian dilakukan dengan melihat apakah monitoring agent dapat melakukan pembacaan beban dari *worker* dan kemudian mengirimkan data ke InfluxDB dalam interval tertentu.

5.2.1.3 Uji Fungsionalitas *Balancer*

Dalam sistem ini *balancer* akan melayani permintaan dari penggua dan meneruskannya pada *worker* yang telah ditentukan. Pengujian akan dilakukan dengan melihat apakah *balancer* dapat menghitung beban kerja dari setiap *worker*, menentukan *worker* yang bekerja dan menggantinya selama interval tertentu, dan juga merespon permintaan dari pengguna.

5.2.2 Skenario Uji Performa

Uji performa dilakukan untuk menguji ketahanan sistem dan kemampuan sistem dalam menangani permintaan dari pengguna. Hasil yang didapat dari pengujian adalah jumlah pengujian yang dapat terlayani dan waktu respon permintaan.

5.2.2.1 Skenario Uji Performa Sistem

Pada pengujian ini dilakukan benchmark dengan menggunakan aplikasi berbasis Java yaitu Apache JMeter. Akses insert akan melakukan pendaftaran ke aplikasi PPDB Surabaya 2015 pada jenjang SMA Umum. Apache JMeter akan membuat thread untuk setiap akses ke aplikasi. Pengujian akan berlangsung bertahap mulai dari 400, 600, 800, 1000 dan 1200 thread dalam satu detik. Dari hasil pengujian akan didapatkan waktu respon terhadap permintaan, persentase galat dan penggunaan *resource* pada aplikasi *balancer*. Parameter hasil pengujian tersebut menunjukan seberapa baik performa dari sistem yang telah dibangun. Pengujian tersebut akan dibandingkan dengan mengganti *balancer* dengan aplikasi Nginx yang menggunakan algoritma Round Robin, IP Hash, dan Least Connected.

5.2.2.2 Skenario Uji Skalabilitas Sistem

Skalabilitas akan dilakukan dengan cara mengganti parameter jumlah *worker* yang ada dalam sistem secara bertahap, mulai dari 2, 4, 6, dan 8 *worker*. Sistem akan diuji dengan melayani sejumlah 800 *thread* dalam satu detik dengan parameter perubahan 10 permintaan. Dari hasil pengujian akan didapat waktu respon terhadap permintaan dan jumlah permintaan yang berhasil ditangani oleh sistem. Hasil yang didapat akan dibandingkan dan melihat pengaruh dari jumlah *worker* terhadap kinerja sistem. Percobaan akan dilakukan

sebanyak tiga kali dan diambil nilai rata-rata dari ketiga percobaan tersebut.

5.2.2.3 Skenario Uji Perbandingan Rasio

Penentuan *worker* yang pada algoritma top-*k* menggunakan parameter persentase penggunaan CPU dan persentase penggunaan *memory*. Untuk melakukan penghitungan beban algoritma top-*k* membutuhkan rasio perbandingan dari kedua parameter tersebut dengan rumus `rasio_memory x beban_memory + rasio_CPU x beban_CPU`. Pada pengujian ini akan dilakukan perbandingan rasio CPU:*memory* mulai dari 30:70, 50:50, dan 70:30.

5.3 Hasil Uji Coba dan Evaluasi

Berikut dijelaskan hasil uji coba dan evaluasi berdasarkan skenario yang sudah dijelaskan pada bab 5.2.

5.3.1 Uji Fungsionalitas

Berikut dijelaskan hasil pengujian fungsionalitas pada sistem yang sudah dibangun.

5.3.1.1 Manajemen Sistem

Dilakukan pengujian pada halaman admin untuk melakukan manajemen pada sistem. Rincian pengujian dan hasil dapat dilihat pada tabel 5.1.

Tabel 5.1: Hasil Uji Coba Manajemen Sistem

No	Perintah	Uji Coba	Hasil
1	./balancer -k [worker] -mem [memory_ratio] -cpu [cpu_ratio] -r [roundtrip]	Menghidupkan layanan *balancer*	OK.
2	Sunting file worker.cfg	Menambahkan daftar IP *worker* yang akan digunakan	OK.

Sesuai dengan skenario ujicoba yang diberikan pada Tabel 5.1, hasil ujicoba menunjukkan semua fungsionalitas manajemen sistem berhasil ditangani.

5.3.1.2 Uji Fungsionalitas *Monitoring Agent*

Pada pengujian ini akan dilakukan pengujian pembacaan, pengiriman, dan penulisan data dari *monitoring agent*. Hasil uji coba tertera pada tabel 5.2.

Tabel 5.2: Hasil Uji Fungsionalitas *Monitoring Agent*

No	Menu	Uji Coba	Hasil
1	Membaca Data	Membaca data beban CPU dan *memory* dari setiap *worker*	OK
2	Mengirim Data	Mengirim data hasil pembacaan melalui jaringan.	OK

Tabel 5.2: Hasil Uji Fungsionalitas *Monitoring Agent*

No	Menu	Uji Coba	Hasil
3	Menulis Data	Menyimpan data yang sudah dikirim kedalam basis data	OK

Sesuai dengan hasil pengujian pada table 5.2 semua fungsionalitas dari *monitoring agent* dapat berjalan dengan baik.

5.3.1.3 Uji Fungsionalitas *Balancer*

Pengujian dilakukan dengan menguji fungsionalitas dari *balancer*. *Balancer* dapat melayani permintaan dari pengguna, menghitung beban *worker*, menentukan *worker*, penggantian *worker* dalam interval waktu tertentu. Hasil uji coba tertera pada tabel 5.3.

Tabel 5.3: Hasil Uji Fungsionalitas *Balancer*

No	Fungsional	Uji Coba	Hasil
1	Menghitung beban *worker*	Melakukan pengambilan data dari basis data dan melakukan penghitungan beban kerja *worker*	OK
2	Menentukan *worker*	Melakukan pengurutan *worker* dengan beban kerja paling ringan dan mengambil sesuai parameter masukan.	OK

Tabel 5.3: Hasil Uji Fungsionalitas *Balancer*

No	Fungsional	Uji Coba	Hasil
3	Melayani permintaan	Meneruskan permintaan dari balancer ke *worker*	OK
		Melayani permintaan pengguna yang mengakses aplikasi PPDB Surabaya 5.4	OK

Tabel 5.4: Hasil Uji Akses Apliksi PPDB Surabaya

Cookie	Kategori	URL	Hasil
Kosong	View	/umum/sambutan	OK
Kosong	View	/umum/ketentuan	OK
Kosong	View	/umum/subrayon	OK
Kosong	View	/umum/inklusif	OK
Kosong	View	/umum/jadwal	OK
Kosong	Insert	/pendaftaran/pilih_kk	OK
Ada	Insert	/pendaftaran	OK
Ada	Insert	/pendaftaran/submit	OK
Ada	Insert	/pendaftaran/submit	OK
Ada	Insert	/pendaftaran/submit	OK
Ada	Insert	/pendaftaran/submit	OK
Ada	Insert	/pendaftaran/submit	OK
Ada	Insert	/pendaftaran/submit	OK
Ada	Insert	/pendaftaran/submit	OK

5.3.2 Uji Performa

Seperti yang sudah dijelaskan pada bab 5.2 pengujian performa dilakukan menggunakan 4 komputer penguji. Pengujian dilakukan secara bertahap dengan menggunakan jumlah *thread* yang berbeda pada tiap komputer penguji. Pada masing-masing *thread* akan mengirimkan permintaan. Evaluasi akan dilakukan terhadap persentase jumlah permintaan yang dapat terlayani dan penggunaan CPU serta *memory* pada setiap pengujian. Persentase jumlah data yang hilang di dapatkan dengan perhitungan `permintaan yang dikirim / permintaan terlayani x 100%`. Selain itu pengujian dilakukan untuk mengetahui waktu respon rata-rata dari semua transaksi yang ditangani *balancer*.

5.3.2.1 Uji Pelayanan Permintaan

Uji pelayana dilakukan dengan mengirimkan data banyak dalam satu detik. Pengiriman akan dilakukan secara bertahap mulai dari 200, 400, 600, 800, 1000, dan 1200 *thread*. Hasil yang didapat adalah kemampuan sistem dalam menangani banyak permintaan. Pengujian akan dibandingkan dengan algoritma lain yang sudah dijelaskan pada subbab 5.2.2.1. Hasil pengujian adalah sebagai berikut:

- Pengujian dengan 200 permintaan:

Tabel 5.5: Hasil Uji Coba menggunakan 200 permintaan

Algoritma	**Persentase Kesalahan**	**Waktu Respon**	**Throughput**
Round Robin	0%	2504 ms	15.7/sec
IP Hash	64.40%	633 ms	44.4/sec

Tabel 5.5: Hasil Uji Coba menggunakan 200 permintaan

Algoritma	**Persentase Kesalahan**	**Waktu Respon**	**Throughput**
Least Connected	0.0%	2260 ms	17.2/sec
Top-*k*	0%	2112 ms	19.1/sec

- Pengujian dengan 400 permintaan:

Tabel 5.6: Hasil Uji Coba menggunakan 400 permintaan

Algoritma	**Persentase Kesalahan**	**Waktu Respon**	**Throughput**
Round Robin	0.7%	4845 ms	16.4/sec
IP Hash	86.9%	352 ms	96,3/sec
Least Connected	0.0%	4381 ms	17.0/sec
Top-*k*	0.0%	4164 ms	18.7/sec

- Pengujian dengan 600 permintaan:

Tabel 5.7: Hasil Uji Coba menggunakan 600 permintaan

Algoritma	**Persentase Kesalahan**	**Waktu Respon**	**Throughput**
Round Robin	5.34%	7642 ms	17.5/sec
IP Hash	93.47%	245 ms	158/sec
Least Connected	0.0%	7680 ms	16.7/sec
Top-*k*	1.27%	6545 ms	19.0/sec

- Pengujian dengan 800 permintaan:

Tabel 5.8: Hasil Uji Coba menggunakan 800 permintaan

Algoritma	**Persentase Kesalahan**	**Waktu Respon**	**Throughput**
Round Robin	8.0%	7011 ms	22.1/sec
IP Hash	97.6%	196 ms	239/sec
Least Connected	0.0%	9678 ms	14.7/sec
Top-*k*	7.0%	7596 ms	21.1/sec

- Pengujian dengan 1000 permintaan:

Tabel 5.9: Hasil Uji Coba menggunakan 1000 permintaan

Algoritma	**Persentase Kesalahan**	**Waktu Respon**	**Throughput**
Round Robin	51.0%	5000 ms	37.9/sec
IP Hash	96.7%	544 ms	254/sec
Least Connected	50%	4550 ms	36.2/sec
Top-*k*	40%	4885 ms	38.1/sec

- Pengujian dengan 1200 permintaan:

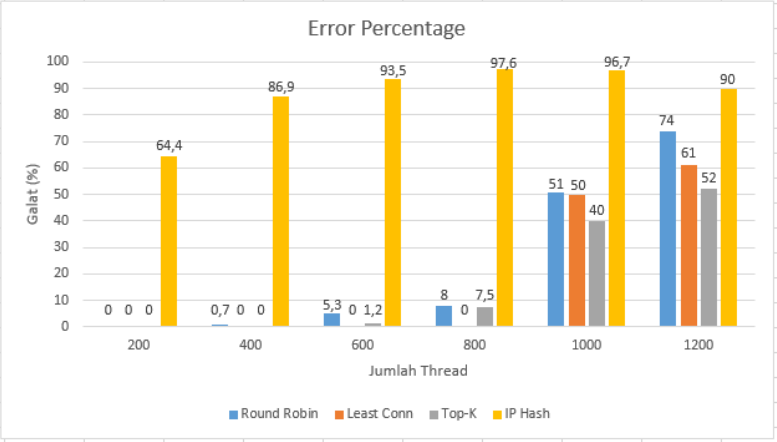
Tabel 5.10: Hasil Uji Coba menggunakan 1200 permintaan

Algoritma	**Persentase Kesalahan**	**Waktu Respon**	**Throughput**
Round Robin	74.0%	2954 ms	52.3/sec

Tabel 5.10: Hasil Uji Coba menggunakan 1200 permintaan

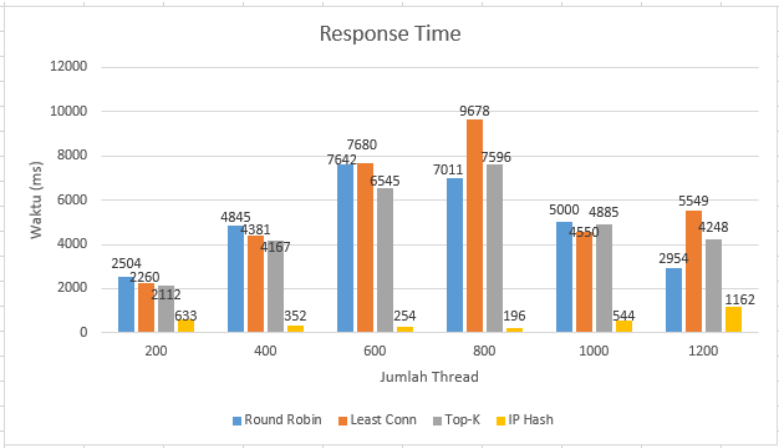
Algoritma	**Persentase Kesalahan**	**Waktu Respon**	**Throughput**
IP Hash	90.0%	1162 ms	91.9/sec
Least Connected	61.0%	5549 ms	25.8/sec
Top-*k*	52.0%	4248 ms	46.5/sec

Dari data pengujian, didapatkan persentase galat yang paling baik dengan menggunakan algoritma Least Connected. Algoritma tersebut memberikan persentase kesalahan 0% saat melayani 600 permintaan, setelah itu algoritma top-*k* dengan 400 permintaan, Round Robin dengan 200 permintaan, dan hasil paling buruk dengan IP Hash yang sudah memberikan 64.40% galat saat melayani 100 permintaan. Hal tersebut menjadikan algoritma Least Connected memberikan hasil yang paling baik untuk menangani permintaan dalam jumlah banyak. Data perbandingan persentase galat dapat dilihat pada Gambar 5.2.



Gambar 5.2: Grafik Persentase Galat

Selain persentase galat, hasil pengujian juga menunjukkan waktu respon dari tiap algoritma pembagian beban. Grafik perbandingan waktu respon dari tiap algoritma dapat dilihat pada Gambar 5.3.

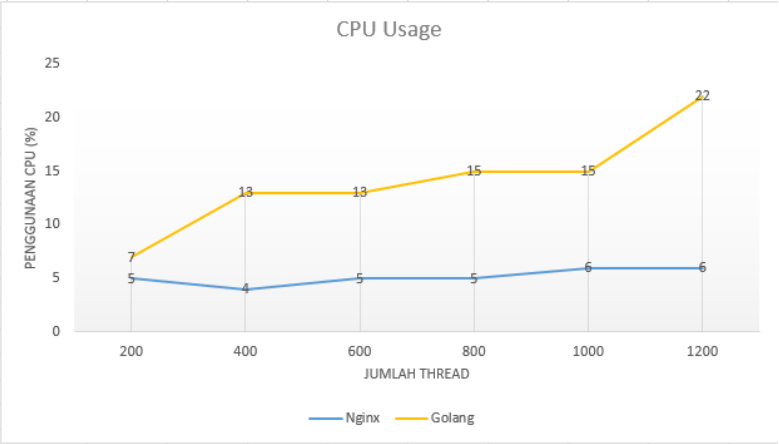


Gambar 5.3: Grafik Persentase Waktu Respon

Hasil paling baik dari waktu respon adalah algoritma IP Hash. Akan tetap hasil tersebut dikarena persentase galat yang besar sehingga permintaan diabaikan. Rata-rata waktu terbaik setelah IP Hash adalah top-*k* kemudian Round Robin. Perbedaan waktu respon kedua algoritma tersebut tidak terlalu signifikan. Hasil paling buruk dengan menggunakan algoritma Least Connected.

5.3.2.2 Penggunaan CPU dan Memory

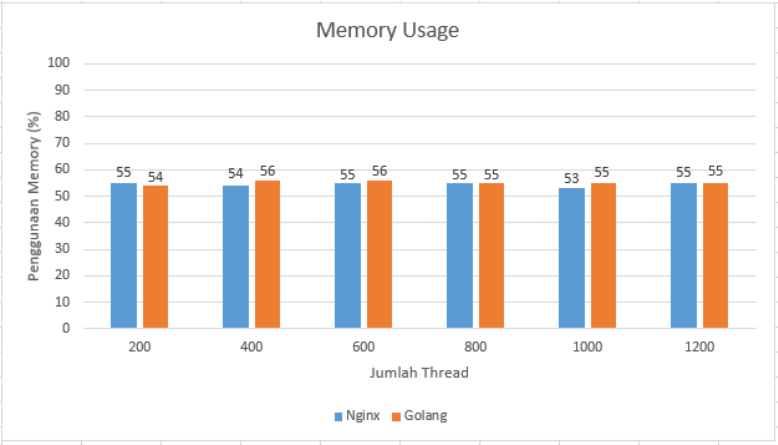
Pengujian dilakukan hingga jumlah *thread* mencapai 1200. Hasil pengujian penggunaan CPU pada dua platform Nginx dan GO Lang tertera pada Gambar 5.4.



Gambar 5.4: Grafik Penggunaan CPU pada Uji Coba Performa

Dari hasil pengujian didapatkan penggunaan CPU pada platform Go Lang membutuhkan persentase yang lebih tinggi. Pertambahan penggunaan CPU semakin meningkat dengan bertambahnya jumlah permintaan yang harus dilayani. Peningkatan penggunakan CPU pada platform Go Lang lebih signifikan dibandingkan dengan Nginx. Kenaikan rata-rata pada platform Go Lang sebesar 7.5% sedangkan kenaikan rata-rata penggunaan CPU pada platform Nginx sebesar 1%.

Sementara untuk penggunaan *memory* tidak terjadi perubahan yang signifikan. Ketika permintaan yang dikirimkan mencapai 1200, kedua platform hanya mengalami kenaikan penggunaan *memory* rata-rata sebesar 1%. Hal tersebut dikarenakan kedua platform menggunakan *multi-threading* sehingga lebih banyak menggunakan CPU dibandingkan *memory*. Hasil perbandingan penggunaan *memory* dapat dilihat pada Gambar 5.5.



Gambar 5.5: Grafik Penggunaan Memory pada Uji Coba Performa

5.3.3 Uji Skalabilitas Sistem

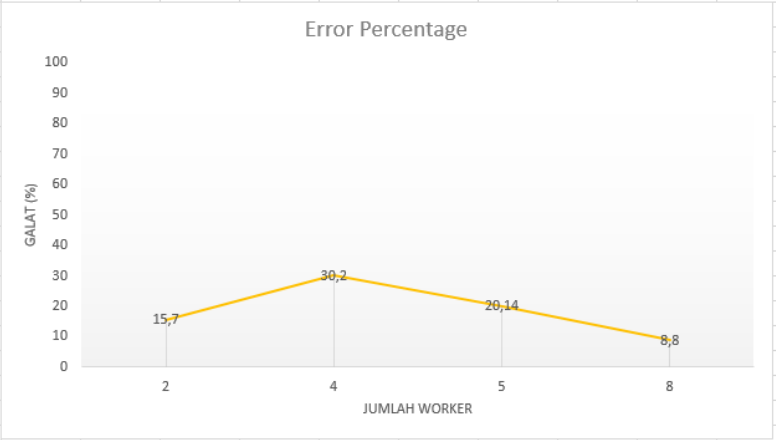
Uji skalabilitas dilakukan untuk membuktikan semakin banyak *worker* yang bekerja maka semakin banyak permintaan yang dapat dilayani. Pengujian akan dilakukan secara bertahap dari 2, 4, 6, dan 8 *worker* seperti pada subbab 5.2.2.2. Hasil pengujian terdapat pada tabel berikut:

Tabel 5.11: Hasil Uji Coba Skalabilitas

Jumlah *Worker*	**Persentase Kesalahan**	**Waktu Respon**	**Throughput**
2	15.7%	11331 ms	16.1/sec
4	30.2%	5568 ms	28.6/sec
6	20.14%	5376 ms	28.4/sec
8	8.8%	7757 ms	21.8/sec

Dari hasil uji coba dapat dilihat, semakin banyak pekerja yang digunakan maka sistem dapat menangani permintaan

dengan lebih baik. Selain itu hasil percobaan terbukti dengan menambahkan jumlah komputer pekerja dapat mengurangi kehilangan data permintaan. Hal ini terjadi karena beban di distribusikan secara merata kepada setiap komputer pekerja. Semakin menurunnya data yang hilang, sistem akan menjadi lebih handal dalam menangani permintaan. Grafik perbandingan persentase galat dari tiap worker dapat dilihat pada Gambar 5.6

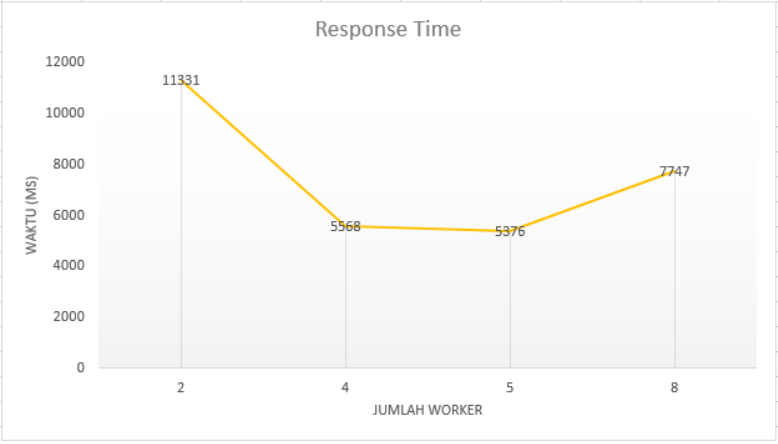


Gambar 5.6: Grafik Persentase Galat pada Uji Coba Skalabilitas

Selain itu dengan bertambahnya jumlah *worker* yang menangani permintaan, waktu yang dibutuhkan untuk menangani permintaan menunjukkan tren yang menurun. Hal tersebut dikarenakan semakin banyak pekerja yang digunakan maka pembagian kerja dari satu permintaan dapat dilayani dengan lebih cepat. Grafik perbandingan waktu respon dapat dilihat pada Gambar 5.7.

Pada hasil pengujian terdapat anomali pada saat menggunakan dua *worker*. Hal tersebut terjadi karena pembagian semua beban benar-benar dibagi rata sama banyak sehingga penganganan layanan lebih merata sehingga pelayanan

permintaan dapat diproses lebih optimal. Akan tetapi hal tersebut mengakibatkan peningkatan waktu respon, karena permintaan akan dilayane setelah permintaan yang sebelumnya selesai dikerjakan. Selain itu semakin sedikit jumlah *worker*, proses penghitungan beban *worker* semakin sering sehingga menyebabkan waktu respon meningkat secara signifikan.



Gambar 5.7: Grafik Waktu Respon pada Uji Coba Skalabilitas

5.3.4 Uji Perbandingan Rasio

Uji perbandingan rasio dilakukan untuk mengetahui pengaruh dari rasio beban CPU dan rasio beban *memory*. Pengujian dilakukan dengan mengirimkan 800 permintaan kepada sistem. Hasil pengujian terdapat pada tabel berikut:

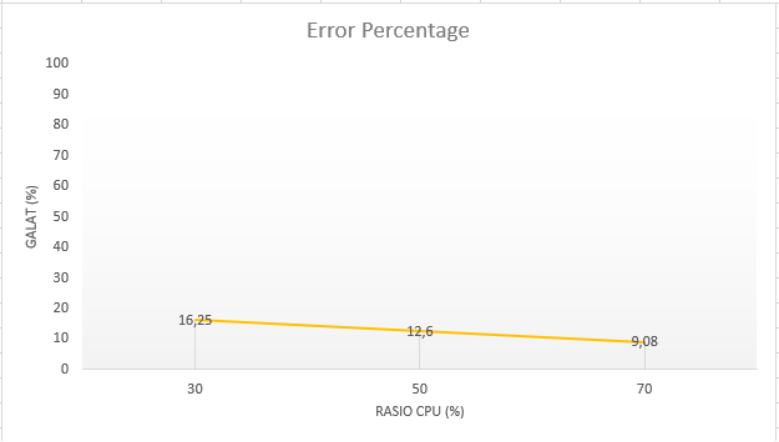
Tabel 5.12: Hasil Uji Coba Perbandingan Rasio

Rasio CPU:Memory	Persentase Kesalahan	Waktu Respon	Throughput
30:70	16.25%	8661 ms	17.50/sec

Tabel 5.12: Hasil Uji Coba Perbandingan Rasio

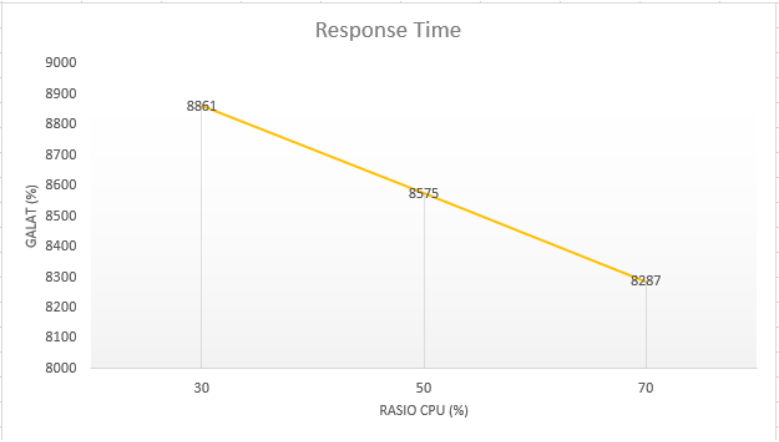
Rasio CPU:Memory	Persentase Kesalahan	Waktu Respon	Throughput
50:50	12.60%	8575 ms	18.8/sec
70:30	9.80%	8287 ms	19.5/sec

Dari hasil pengujian diatas didapatkan semakin besar rasio CPU untuk menentukan *worker* pada algoritma top-*k* didapatkan persentase galat yang semakin rendah. Grafik perbandingan galat dapat dilihat pada Gambar 5.8.



Gambar 5.8: Grafik Galat pada Uji Coba Perbandingan Rasio

Selain itu semakin besar rasio CPU juga mempersingkat waktu respon permintaan. Data tersebut menenjukkan penentuan *worker* dengan algoritma top-*k* lebih efektif dengan rasio CPU yang lebih tinggi. Hal tersebut dikarenakan pada sistem ini perubahan CPU lebih signifikan dibandingkan dengan perubahan *memory* saat melayani permintaan dari pengguna. Grafik perbandingan waktu respon dapat dilihat pada Gambar 5.9.



Gambar 5.9: Grafik Waktu Respon pada Uji Coba Perbandingan Rasio

BAB 6

PENUTUP

Bab ini membahas kesimpulan yang dapat diambil dari tujuan pembuatan sistem dan hubungannya dengan hasil uji coba dan evaluasi yang telah dilakukan. Selain itu, terdapat beberapa saran yang bisa dijadikan acuan untuk melakukan pengembangan dan penelitian lebih lanjut.

6.1 Kesimpulan

Dari proses perancangan, implementasi dan pengujian terhadap sistem, dapat diambil beberapa kesimpulan berikut:

1. Pembagian beban dengan algoritma top-*k* pada basis data *multi-master* berhasil dilakukan dengan tingkat keberhasilan mencapai 100%.
2. Algoritma top-*k* memiliki hasil penanganan layanan yang lebih buruk 16% dari Least Connected, tetapi memiliki waktu respon rata-rata 758ms lebih cepat.
3. Semakin besar perbandingan rasio CPU dan jumlah *worker*, kinerja sistem pembagi beban semakin meningkat sebesar 13%.
4. Penggunaan CPU rata-rata pada sistem yang dibuat lebih tinggi 16% dari aplikasi Nginx.

6.2 Saran

Berikut beberapa saran yang diberikan untuk pengembangan lebih lanjut:

- Mekanisme penanganan permintaan perlu ditingkatkan untuk memastikan semua layanan dapat terpenuhi.
- Mekanisme load balancing yang sudah dirancang perlu ditambah dengan mekanisme pengecekan ketersediaan worker. Ketersediaan dapat berupa penggunaan CPU dan

memori serta batas open file jika memang proses pada worker dibatasi open file.

- Perlu digunakan mekanisme yang terpisah untuk melakukan penghitungan kerja *worker* sehingga dapat berjalan secara konkuren agar tidak mengganggu kinerja pembagi beban.

DAFTAR PUSTAKA

[1] severalnines **Learn the difference between Multi-Master and Multi-Source replication**, [Online], https://severalnines.com/blog/learn-difference-between-multi-master-\and-multi-source-replication, diakses tanggal 28 Februari 2017

[2] H. Paci, E. Kajo, I. Tafa and A. Xhuvani, **Adding A New Site In An Existing Oracle Multimaster Replication Without Quiescing The Replication** International Journal of Database Management Systems (IJDMS), Vol.3, No.3, August 2011.

[3] B. J. Santoso, G. M. Chiu, **Close Dominance Graph: An Efficient Framework for Answering Continuous Top-k Dominating Queries** IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 26, NO. 8, AUGUST 2014.

[4] The MariaDB Foundation **About MariaDB**, [Online], https://mariadb.org/about/, diakses tanggal 28 Februari 2017

[5] Galera Cluster **Galera Cluster Documemtation**, [Online], http://galeracluster.com/documentation-webpages/, diakses tanggal 28 Februari 2017

[6] Collectd – The system statistics collection daemon **CollectD**, [Online], https://collectd.org/, diakses tanggal 28 Februari 2017

[7] Influx Data **InfluxDB Open Source Platform**, [Online], https://www.influxdata.com/, diakses tanggal 28 Februari 2017

[8] The Go Programming Language **The Go Programming Language**, [Online], https://golang.org/doc/, diakses tanggal 28 Februari 2017

[9] Apache Software Foundation **Apache JMeter**, [Online], http://jmeter.apache.org/, diakses tanggal 28 Februari 2017

[10] Grafana Labs **Grafana The Analytics Platform**, [Online], https://grafana.com/, diakses tanggal 28 Februari 2017

[11] Oracle Documentation **Oracle**, [Online], https://docs.oracle.com/cd/B28359_01/server.111/b28326/repmaster.htm#i43908, diakses tanggal 28 Februari 2017

LAMPIRAN A

BERKAS KONFIGURASI

A.1 types.db

Dalam instalasi aplikasi InfluxDB terdapat berkas `types.db` yang diperlukan agar aplikasi berjalan dengan benar. Berkas konfigurasi dari `types.db` dapat dilihat seperti dibawah ini:

```
absolute                  value:ABSOLUTE:0:U
apache_bytes              value:DERIVE:0:U
apache_connections        value:GAUGE:0:65535
apache_idle_workers       value:GAUGE:0:65535
apache_requests           value:DERIVE:0:U
apache_scoreboard         value:GAUGE:0:65535
ath_nodes                 value:GAUGE:0:65535
ath_stat                  value:DERIVE:0:U
backends                  value:GAUGE:0:65535
bitrate                   value:GAUGE:0:4294967295
blocked_clients           value:GAUGE:0:U
bucket                    value:GAUGE:0:U
bytes                     value:GAUGE:0:U
cache_eviction            value:DERIVE:0:U
cache_operation           value:DERIVE:0:U
cache_ratio               value:GAUGE:0:100
cache_result              value:DERIVE:0:U
cache_size                value:GAUGE:0:1125899906842623
capacity                  value:GAUGE:0:U
ceph_bytes                value:GAUGE:U:U
ceph_latency              value:GAUGE:U:U
ceph_rate                 value:DERIVE:0:U
changes_since_last_save value:GAUGE:0:U
charge                    value:GAUGE:0:U
clock_last_meas           value:GAUGE:0:U
clock_last_update         value:GAUGE:U:U
clock_mode                value:GAUGE:0:U
clock_reachability        value:GAUGE:0:U
clock_skew_ppm            value:GAUGE:0:1000000
clock_state               value:GAUGE:0:U
clock_stratum             value:GAUGE:0:U
compression               uncompressed:DERIVE:0:U, compressed:
    DERIVE:0:U
```

```
33 compression_ratio       value:GAUGE:0:2
34 connections             value:DERIVE:0:U
35 conntrack               value:GAUGE:0:4294967295
36 contextswitch           value:DERIVE:0:U
37 count                   value:GAUGE:0:U
38 counter                 value:COUNTER:U:U
39 cpu                     value:DERIVE:0:U
40 cpu_affinity            value:GAUGE:0:1
41 cpufreq                 value:GAUGE:0:U
42 current                 value:GAUGE:U:U
43 current_connections     value:GAUGE:0:U
44 current_sessions        value:GAUGE:0:U
45 delay                   value:GAUGE:-1000000:1000000
46 derive                  value:DERIVE:0:U
47 df                      used:GAUGE:0:1125899906842623, free:
       GAUGE:0:1125899906842623
48 df_complex              value:GAUGE:0:U
49 df_inodes               value:GAUGE:0:U
50 dilution_of_precision   value:GAUGE:0:U
51 disk_error              value:GAUGE:0:U
52 disk_io_time            io_time:DERIVE:0:U, weighted_io_time
       :DERIVE:0:U
53 disk_latency            read:GAUGE:0:U, write:GAUGE:0:U
54 disk_merged             read:DERIVE:0:U, write:DERIVE:0:U
55 disk_octets             read:DERIVE:0:U, write:DERIVE:0:U
56 disk_ops                read:DERIVE:0:U, write:DERIVE:0:U
57 disk_ops_complex        value:DERIVE:0:U
58 disk_time               read:DERIVE:0:U, write:DERIVE:0:U
59 dns_answer              value:DERIVE:0:U
60 dns_notify              value:DERIVE:0:U
61 dns_octets              queries:DERIVE:0:U, responses:DERIVE
       :0:U
62 dns_opcode              value:DERIVE:0:U
63 dns_qtype               value:DERIVE:0:U
64 dns_qtype_cached        value:GAUGE:0:4294967295
65 dns_query               value:DERIVE:0:U
66 dns_question            value:DERIVE:0:U
67 dns_rcode               value:DERIVE:0:U
68 dns_reject              value:DERIVE:0:U
69 dns_request             value:DERIVE:0:U
70 dns_resolver            value:DERIVE:0:U
```

```
dns_response            value:DERIVE:0:U
dns_transfer            value:DERIVE:0:U
dns_update              value:DERIVE:0:U
dns_zops                value:DERIVE:0:U
domain_state            state:GAUGE:0:U, reason:GAUGE:0:U
drbd_resource           value:DERIVE:0:U
duration                seconds:GAUGE:0:U
email_check             value:GAUGE:0:U
email_count             value:GAUGE:0:U
email_size              value:GAUGE:0:U
energy                  value:GAUGE:U:U
energy_wh               value:GAUGE:U:U
entropy                 value:GAUGE:0:4294967295
errors                  value:DERIVE:0:U
evicted_keys            value:DERIVE:0:U
expired_keys            value:DERIVE:0:U
fanspeed                value:GAUGE:0:U
file_handles            value:GAUGE:0:U
file_size               value:GAUGE:0:U
files                   value:GAUGE:0:U
filter_result           value:DERIVE:0:U
flow                    value:GAUGE:0:U
fork_rate               value:DERIVE:0:U
frequency               value:GAUGE:0:U
frequency_error         value:GAUGE:-1000000:1000000
frequency_offset        value:GAUGE:-1000000:1000000
fscache_stat            value:DERIVE:0:U
gauge                   value:GAUGE:U:U
hash_collisions         value:DERIVE:0:U
http_request_methods    value:DERIVE:0:U
http_requests           value:DERIVE:0:U
http_response_codes     value:DERIVE:0:U
humidity                value:GAUGE:0:100
if_collisions           value:DERIVE:0:U
if_dropped              rx:DERIVE:0:U, tx:DERIVE:0:U
if_errors               rx:DERIVE:0:U, tx:DERIVE:0:U
if_multicast            value:DERIVE:0:U
if_octets               rx:DERIVE:0:U, tx:DERIVE:0:U
if_packets              rx:DERIVE:0:U, tx:DERIVE:0:U
if_rx_dropped           value:DERIVE:0:U
if_rx_errors            value:DERIVE:0:U
```

```
112 if_rx_octets            value:DERIVE:0:U
113 if_rx_packets           value:DERIVE:0:U
114 if_tx_dropped           value:DERIVE:0:U
115 if_tx_errors            value:DERIVE:0:U
116 if_tx_octets            value:DERIVE:0:U
117 if_tx_packets           value:DERIVE:0:U
118 invocations             value:DERIVE:0:U
119 io_octets               rx:DERIVE:0:U, tx:DERIVE:0:U
120 io_packets              rx:DERIVE:0:U, tx:DERIVE:0:U
121 ipc                     value:GAUGE:0:U
122 ipt_bytes               value:DERIVE:0:U
123 ipt_packets             value:DERIVE:0:U
124 irq                     value:DERIVE:0:U
125 job_stats               value:DERIVE:0:U
126 latency                 value:GAUGE:0:U
127 links                   value:GAUGE:0:U
128 load                    shortterm:GAUGE:0:5000, midterm:
        GAUGE:0:5000, longterm:GAUGE:0:5000
129 memory_bandwidth        value:DERIVE:0:U
130 md_disks                value:GAUGE:0:U
131 memcached_command       value:DERIVE:0:U
132 memcached_connections   value:GAUGE:0:U
133 memcached_items         value:GAUGE:0:U
134 memcached_octets        rx:DERIVE:0:U, tx:DERIVE:0:U
135 memcached_ops           value:DERIVE:0:U
136 memory                  value:GAUGE:0:281474976710656
137 memory_lua              value:GAUGE:0:281474976710656
138 memory_throttle_count   value:DERIVE:0:U
139 multimeter              value:GAUGE:U:U
140 mutex_operations        value:DERIVE:0:U
141 mysql_bpool_bytes       value:GAUGE:0:U
142 mysql_bpool_counters    value:DERIVE:0:U
143 mysql_bpool_pages       value:GAUGE:0:U
144 mysql_commands          value:DERIVE:0:U
145 mysql_handler           value:DERIVE:0:U
146 mysql_innodb_data       value:DERIVE:0:U
147 mysql_innodb_dblwr      value:DERIVE:0:U
148 mysql_innodb_log        value:DERIVE:0:U
149 mysql_innodb_pages      value:DERIVE:0:U
150 mysql_innodb_row_lock   value:DERIVE:0:U
151 mysql_innodb_rows       value:DERIVE:0:U
```

```
mysql_locks             value:DERIVE:0:U
mysql_log_position      value:DERIVE:0:U
mysql_octets            rx:DERIVE:0:U, tx:DERIVE:0:U
mysql_select            value:DERIVE:0:U
mysql_sort              value:DERIVE:0:U
mysql_sort_merge_passes value:DERIVE:0:U
mysql_sort_rows         value:DERIVE:0:U
mysql_slow_queries      value:DERIVE:0:U
nfs_procedure           value:DERIVE:0:U
nginx_connections       value:GAUGE:0:U
nginx_requests          value:DERIVE:0:U
node_octets             rx:DERIVE:0:U, tx:DERIVE:0:U
node_rssi               value:GAUGE:0:255
node_stat               value:DERIVE:0:U
node_tx_rate            value:GAUGE:0:127
objects                 value:GAUGE:0:U
operations              value:DERIVE:0:U
operations_per_second   value:GAUGE:0:U
packets                 value:DERIVE:0:U
pending_operations      value:GAUGE:0:U
percent                 value:GAUGE:0:100.1
percent_bytes           value:GAUGE:0:100.1
percent_inodes          value:GAUGE:0:100.1
perf                    value:DERIVE:0:U
pf_counters             value:DERIVE:0:U
pf_limits               value:DERIVE:0:U
pf_source               value:DERIVE:0:U
pf_state                value:DERIVE:0:U
pf_states               value:GAUGE:0:U
pg_blks                 value:DERIVE:0:U
pg_db_size              value:GAUGE:0:U
pg_n_tup_c              value:DERIVE:0:U
pg_n_tup_g              value:GAUGE:0:U
pg_numbackends          value:GAUGE:0:U
pg_scan                 value:DERIVE:0:U
pg_xact                 value:DERIVE:0:U
ping                    value:GAUGE:0:65535
ping_droprate           value:GAUGE:0:100
ping_stddev             value:GAUGE:0:65535
players                 value:GAUGE:0:1000000
power                   value:GAUGE:U:U
```

```
pressure                value:GAUGE:0:U
protocol_counter        value:DERIVE:0:U
ps_code                 value:GAUGE:0:9223372036854775807
ps_count                processes:GAUGE:0:1000000, threads:
    GAUGE:0:1000000
ps_cputime              user:DERIVE:0:U, syst:DERIVE:0:U
ps_data                 value:GAUGE:0:9223372036854775807
ps_disk_octets          read:DERIVE:0:U, write:DERIVE:0:U
ps_disk_ops             read:DERIVE:0:U, write:DERIVE:0:U
ps_pagefaults           minflt:DERIVE:0:U, majflt:DERIVE:0:U
ps_rss                  value:GAUGE:0:9223372036854775807
ps_stacksize            value:GAUGE:0:9223372036854775807
ps_state                value:GAUGE:0:65535
ps_vm                   value:GAUGE:0:9223372036854775807
pubsub                  value:GAUGE:0:U
queue_length            value:GAUGE:0:U
records                 value:GAUGE:0:U
requests                value:GAUGE:0:U
response_code           value:GAUGE:0:U
response_time           value:GAUGE:0:U
root_delay              value:GAUGE:U:U
root_dispersion         value:GAUGE:U:U
route_etx               value:GAUGE:0:U
route_metric            value:GAUGE:0:U
routes                  value:GAUGE:0:U
satellites              value:GAUGE:0:U
segments                value:GAUGE:0:65535
serial_octets           rx:DERIVE:0:U, tx:DERIVE:0:U
signal_noise            value:GAUGE:U:0
signal_power            value:GAUGE:U:0
signal_quality          value:GAUGE:0:U
smart_attribute         current:GAUGE:0:255, worst:GAUGE
    :0:255, threshold:GAUGE:0:255, pretty:GAUGE:0:U
smart_badsectors        value:GAUGE:0:U
smart_powercycles       value:GAUGE:0:U
smart_poweron           value:GAUGE:0:U
smart_temperature       value:GAUGE:-300:300
snr                     value:GAUGE:0:U
spam_check              value:GAUGE:0:U
spam_score              value:GAUGE:U:U
spl                     value:GAUGE:U:U
```

```
swap                    value:GAUGE:0:1099511627776
swap_io                 value:DERIVE:0:U
tcp_connections         value:GAUGE:0:4294967295
temperature             value:GAUGE:U:U
threads                 value:GAUGE:0:U
time_dispersion         value:GAUGE:-1000000:1000000
time_offset             value:GAUGE:-1000000:1000000
time_offset_ntp         value:GAUGE:-1000000:1000000
time_offset_rms         value:GAUGE:-1000000:1000000
time_ref                value:GAUGE:0:U
timeleft                value:GAUGE:0:U
total_bytes             value:DERIVE:0:U
total_connections       value:DERIVE:0:U
total_objects           value:DERIVE:0:U
total_operations        value:DERIVE:0:U
total_requests          value:DERIVE:0:U
total_sessions          value:DERIVE:0:U
total_threads           value:DERIVE:0:U
total_time_in_ms        value:DERIVE:0:U
total_values            value:DERIVE:0:U
uptime                  value:GAUGE:0:4294967295
users                   value:GAUGE:0:65535
vcl                     value:GAUGE:0:65535
vcpu                    value:GAUGE:0:U
virt_cpu_total          value:DERIVE:0:U
virt_vcpu               value:DERIVE:0:U
vmpage_action           value:DERIVE:0:U
vmpage_faults           minflt:DERIVE:0:U, majflt:DERIVE:0:U
vmpage_io               in:DERIVE:0:U, out:DERIVE:0:U
vmpage_number           value:GAUGE:0:4294967295
volatile_changes        value:GAUGE:0:U
voltage                 value:GAUGE:U:U
voltage_threshold       value:GAUGE:U:U, threshold:GAUGE:U:U
vs_memory               value:GAUGE:0:9223372036854775807
vs_processes            value:GAUGE:0:65535
vs_threads              value:GAUGE:0:65535

#
# Legacy types
# (required for the v5 upgrade target)
#
```

```
273 arc_counts              demand_data:COUNTER:0:U,
        demand_metadata:COUNTER:0:U, prefetch_data:COUNTER:0:U,
        prefetch_metadata:COUNTER:0:U
274 arc_l2_bytes            read:COUNTER:0:U, write:COUNTER:0:U
275 arc_l2_size             value:GAUGE:0:U
276 arc_ratio               value:GAUGE:0:U
277 arc_size                current:GAUGE:0:U, target:GAUGE:0:U,
         minlimit:GAUGE:0:U, maxlimit:GAUGE:0:U
278 mysql_qcache            hits:COUNTER:0:U, inserts:COUNTER:0:
        U, not_cached:COUNTER:0:U, lowmem_prunes:COUNTER:0:U,
        queries_in_cache:GAUGE:0:U
279 mysql_threads           running:GAUGE:0:U, connected:GAUGE
        :0:U, cached:GAUGE:0:U, created:COUNTER:0:U
```

Kode Sumber A.1: Isi dari berkas types.db

LAMPIRAN B

KODE SUMBER

B.1 Kode Sumber Go Lang

B.1.1 Import Pustaka Pengambilan Data

Beberapa pustaka harus di impor untuk mendukung pengambilan data pada basis data InfluxDB. Pustaka-pustaka yang diperlukan yang diperlukan antara lain pustaka log, fmt, encoding/json, sort, dan Influx Client.

```
1 import (
2   "log"
3   "fmt"
4   "encoding/json"
5   "sort"
6
7   "github.com/influxdata/influxdb/client/v2"
8 )
```

Kode Sumber B.1: Kode Sumber Import Pustaka Pengambilan Data

B.1.2 Konstanta Koneksi InfluxDB

Variabel konstanta berisi nama *database*, *username*, kata sandi, alamat IP, dan nomor *port* dari aplikasi InfluxDB.

```
1 const (
2   MyDB = "resource"
3   username = "fairfax"
4   password = "password"
5   address = "http://10.151.36.64:8086"
6 )
```

Kode Sumber B.2: Kode Sumber Pembuatan Variable Konstanta Koneksi InfluxDB

B.1.3 Variable Penampung Beban *Worker*

Variabel yang berupa *struct* akan menampung alamat IP *worker*, nama *worker*, beban CPU, *memory*, dan penghitungan beban.

```
type DataSlice []*Data

type Data struct {
  Host    string
  Ip      string
  Cpu     float64
  Mem     float64
  Weight  float64
}
```

Kode Sumber B.3: Kode Sumber Pembuatan Variable Konstanta Koneksi InfluxDB

B.1.4 Fungsi Koneksi InfluxDB

Fungsi tersebut akan melakukan inisiasi koneksi dengan aplikasi InfluxDB. Fungsi tersebut memerlukan parameter dari konstanta B.2.

```
func CreateClient() client.Client{
  c, err := client.NewHTTPClient(client.HTTPConfig{
    Addr:     address,
    Username: username,
    Password: password,
  })
  if err != nil {
    log.Fatalln("Error: ", err)
  }
  return c
}
```

Kode Sumber B.4: Kode Sumber Pembuatan Fungsi Koneksi InfluxDB

B.1.5 Fungsi *Query* InfluxDB

Fungsi tersebut akan melakukan pengambilan data dari aplikasi InfluxDB dan mengembalikan nilai berupa beban CPU dan *memory*.

```
1  func GetData(clnt client.Client, node string) (cpu float64,
       mem float64) {
2    q := client.Query{
3      Command:  fmt.Sprintf("SELECT last(value) from cpu_value
           , memory_value where type='percent' and host='%s'",
           node),
4      Database: MyDB,
5    }
6    if response, err := clnt.Query(q); err == nil {
7      res := response.Results
8      cpu, err = res[0].Series[0].Values[0][1].(json.Number).
           Float64()
9      mem, err = res[0].Series[1].Values[0][1].(json.Number).
           Float64()
10   }else {
11     return cpu, mem
12   }
13   return cpu, mem
14 }
```

Kode Sumber B.5: Kode Sumber Pembuatan Fungsi Query Koneksi InfluxDB

B.1.6 Fungsi Top-*k*

Algoritma top-*k* digunakan sebagai penentu *worker* dengan beban yang paling ringan. Parameter masukan dari fungsi adalah *array worker*, jumlah *k*, rasio CPU, rasio *memory*.

```
1  func (d DataSlice) Len() int {
2    return len(d)
3  }
4
5  func (d DataSlice) Swap(i, j int) {
6    d[i], d[j] = d[j], d[i]
```

```
7  }
8
9  func (d DataSlice) Less(i, j int) bool {
10   return d[i].Weight < d[j].Weight
11 }
12
13 func TopK(s DataSlice, c client.Client, perMem float64,
       perCPU float64, k int) []string {
14   for _, d := range s {
15     d.Cpu, d.Mem = GetData(c, d.Host)
16     d.Weight = d.Cpu*perCPU + d.Mem*perMem
17   }
18   fmt.Println("========================================")
19   sort.Sort(s)
20   var ret []string
21   for i:=1; i<=k; i++ {
22     fmt.Printf("Host: %v | CPU: %v | Memory: %v | Total: %v\
         n",s[i].Host, s[i].Cpu, s[i].Mem, s[i].Weight)
23     ret = append(ret, s[i].Ip)
24   }
25   return ret
26 }
```

Kode Sumber B.6: Kode Sumber Fungsi Top-K

B.1.7 Import Pustaka Pembangun *Reverse Proxy*

Beberapa pustaka yang harus di impor untuk mendukung berjalannya *reverse proxy*. Pustaka-pustaka yang diperlukan antara lain net dan sync.

```
1  import (
2    "net/http"
3    "net/http/httputil"
4    "log"
5    "sync"
6    "fmt"
7    "flag"
8
9    "data"
```

```
10    "github.com/influxdata/influxdb/client/v2"
11 )
```

Kode Sumber B.7: Kode Sumber Import Pustaka Pembangun Reverse Proxy

B.1.8 Fungsi Reverse Proxy

Fungsi *reverse proxy* akan meneruskan permintaan dari pengguna kepada *worker*, dan mengirimkan hasil pengolahan kembali kepada pengguna.

```
1  var node []string
2
3  func NewMultipleReverseProxy(s data.DataSlice, c client.
       Client, perMem float64, perCpu float64, k int) *httputil
       .ReverseProxy {
4    var mutex = &sync.Mutex{}
5    sum := 0
6    index := 0
7    for _, d := range s {
8      d.Cpu, d.Mem = data.GetData(c, d.Host)
9    }
10   node = data.TopK(s, c, perMem, perCpu, k)
11   director := func(req *http.Request) {
12     mutex.Lock()
13     if index == len(node) {
14       index = 0
15     }
16     if sum == (10*len(node)) {
17       for _, d := range s {
18         d.Cpu, d.Mem = data.GetData(c, d.Host)
19       }
20       node = data.TopK(s, c, perMem, perCpu, k)
21       sum = 0
22     }
23     req.URL.Scheme = "http"
24     req.URL.Host = node[index]
25     index++
26     sum++
27     mutex.Unlock()
```

```
28    }
29    return &httputil.ReverseProxy{
30      Director: director,
31    }
32  }
```

Kode Sumber B.8: Kode Sumber Fungsi Reverse Proxy

B.1.9 Inisiasi *Worker*

Semua *worker* yang siap menerima permintaan dari pengguna didaftarkan pada sebuar *array*.

```
1  m := map[int]*data.Data {
2    1: {"mnode-1", "10.151.36.61:80", 999.9, 999.9, 999.9},
3    2: {"mnode-2", "10.151.36.62:80", 999.9, 999.9, 999.9},
4    3: {"mnode-3", "10.151.36.63:80", 999.9, 999.9, 999.9},
5    4: {"mnode-4", "10.151.36.65:80", 999.9, 999.9, 999.9},
6    5: {"mnode-5", "10.151.36.66:80", 999.9, 999.9, 999.9},
7    6: {"mnode-6", "10.151.36.67:80", 999.9, 999.9, 999.9},
8    7: {"mnode-7", "10.151.36.68:80", 999.9, 999.9, 999.9},
9    8: {"mnode-8", "10.151.36.69:80", 999.9, 999.9, 999.9},
10   9: {"mnode-9", "10.151.36.70:80", 999.9, 999.9, 999.9},
11   10: {"mnode-10", "10.151.36.71:80", 999.9, 999.9, 999.9},
12 }
13 s := make(data.DataSlice, 0, len(m))
14 for _, d := range m {
15   s = append(s, d)
16 }
```

Kode Sumber B.9: Kode Sumber Inisiasi *Worker*

B.1.10 Membuka Koneksi dan Layanan *Balancer*

Agar dapat diakses oleh pengguna, maka diperlukan sebuah layanan yang berjalan pada *balancer*.

```
1 c := data.CreateClient()
2 proxy := NewMultipleReverseProxy(s, c, *perMem, *perCpu, *k)
```

```
3 fmt.Println("balancer ready on port 80")
4 log.Fatal(http.ListenAndServe(":80", proxy))
```

Kode Sumber B.10: Kode Sumber Koneksi dan Layanan

(Halaman ini sengaja dikosongkan)

BIODATA PENULIS

Muhammad Syaiful Jihad Amrulloh, Akrab dipanggil Uul lahir di Klaten pada tanggal 16 Januari 1995. Penulis yang memiliki hobi bermain sepak bola ini adalah anak pertama dari dua bersaudara. Penulis menempuh pendidikan formal di SDN 1 Tempursari Klaten, SMPN 1 Klaten, SMAN 1 Klaten dan melanjutkan kuliah S1 Teknik Informatika FTIF ITS. Penulis pernah menjadi asisten Lab pada Laboratorium Arsitektur dan Jaringan Komputer pada bidang Maintenance. Selama menjadi asisten Lab penulis beberapa kali menjadi asisten dosen dan praktikum pada mata kuliah sistem operasi, jaringan komputer, dan teknologi antar jaringan. Penulis mudah memahami hal baru dan tertarik pada bidang Infrastruktur Jaringan dan Komputasi Awan. Penulis dapat dihubungi melalui surat elektronik `jihad.amrulloh@gmail.com`.