



TUGAS AKHIR - KI141502

**RANCANG BANGUN SISTEM PENYEIMBANG BEBAN PADA
KLASTER SERVER DENGAN PRIORITAS BERBASIS KONTEN
DAN KONTROL KETERSEDIAAN LAYANAN**

BAHRUL HALIMI
NRP 5111100014

Dosen Pembimbing
Royyana Muslim Ijtihadie, S.Kom, M.Kom, PhD
Baskoro Adi P, S.Kom, M.Kom

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2015

Halaman ini sengaja dikosongkan



TUGAS AKHIR - KI141502

**RANCANG BANGUN SISTEM PENYEIMBANG BEBAN PADA
KLASTER SERVER DENGAN PRIORITAS BERBASIS KONTEN
DAN KONTROL KETERSEDIAAN LAYANAN**

BAHRUL HALIMI
NRP 5111100014

Dosen Pembimbing
Royyana Muslim Ijtihadie, S.Kom, M.Kom, PhD
Baskoro Adi P, S.Kom, M.Kom

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2015

Halaman ini sengaja dikosongkan



UNDERGRADUATE THESIS - KI141502

DESIGN AND IMPLEMENTATION OF LOAD BALANCING SYSTEM WITH CONTENT PRIORITY AND AVAILABILITY CONTROL

BAHRUL HALIMI
NRP 5111100014

Supervisor
Royyana Muslim Ijtihadie, S.Kom, M.Kom, PhD

Baskoro Adi P, S.Kom, M.Kom

Department of INFORMATICS
Faculty of Information Technology
Institut Teknologi Sepuluh Nopember
Surabaya, 2015

Halaman ini sengaja dikosongkan

**RANCANG BANGUN SISTEM PENYEIMBANG BEBAN
PADA KLASTER SERVER DENGAN PRIORITAS
BERBASIS KONTEN DAN KONTROL KETERSEDIAAN
LAYANAN**

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 :

Bahrul Halimi

NRP: 5111100014

Disetujui oleh Dosen Pembimbing Tugas Akhir :

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

Baskoro Adi P, S.Kom, M.Kom
NIP: 198702182014041001 (Pembimbing 2)

**SURABAYA
Desember 2015**

Halaman ini sengaja dikosongkan

RANCANG BANGUN SISTEM PENYEIMBANG BEBAN PADA KLASTER SERVER DENGAN PRIORITAS BERBASIS KONTEN DAN KONTROL KETERSEDIAAN LAYANAN

Nama	:	BAHRUL HALIMI
NRP	:	5111100014
Jurusan	:	Teknik Informatika FTIf
Pembimbing I	:	Royyana Muslim Ijtihadie, S.Kom, M.Kom, PhD
Pembimbing II	:	Baskoro Adi P, S.Kom, M.Kom

Abstrak

Aplikasi berbasis web semakin diminati untuk berbagai proses bisnis yang ada di lingkungan kita. Salah satu yang menjadi sorotan adalah penerimaan peserta didik baru yang dilaksanakan secara *online*. Dengan adanya aplikasi ini pengguna akan dihadapkan pada dua jenis halaman yaitu halaman pengisian informasi untuk daftar dan halaman untuk menampilkan informasi. Tentu dengan pengaturan yang biasa, server akan melayani dua jenis halaman ini secara bersamaan. Kondisi ini akan mengakibatkan *bottle neck* atau penumpukan permintaan.

Muncul gagasan untuk membagi beban kerja ke komputer lain agar setiap permintaan yang masuk dapat dilayani. Gagasan ini sudah biasa dilakukan dengan menggunakan Nginx sebagai balancer. Namun dengan beragamnya tipe permintaan pengguna, waktu yang dibutuhkan untuk melayani menjadi tidak stabil.

Gagasan lain muncul dengan adanya pembagian beban kerja berdasarkan konten yang diakses pengguna. Konten ini dapat diartikan sebagai dua jenis halaman sebelumnya. NodeJS akan bekerja sebagai balancer dan membaca setiap permintaan pengguna dan mengarahkan permintaan kepada *worker* yang sesuai untuk setiap URL yang diakses pengguna. Dibantu dengan MongoDB sebagai basis data, NodeJS akan bekerja lebih konsisten terhadap data masuk ke sistem.

Hasil menunjukkan dengan NodeJS dan prioritas berbasis konten,

waktu respon meningkat seiring dengan bertambahnya permintaan yang masuk. Namun dengan terfokusnya kerja *worker* untuk melayani satu jenis halaman, membuat setiap permintaan dapat terlayani hingga permintaan selesai.

Kata-Kunci: Load Balance, Prioritas Berbasis Konten, NodeJS, MongoDB.

DESIGN AND IMPLEMENTATION OF LOAD BALANCING SYSTEM WITH CONTENT PRIORITY AND AVAILABILITY CONTROL

Name : BAHRUL HALIMI
NRP : 5111100014
Major : Informatics FTIf
Supervisor I : Royyana Muslim Ijtihadie, S.Kom, M.Kom, PhD
Supervisor II : Baskoro Adi P, S.Kom, M.Kom

Abstract

Web-based application become viral in our living society nowadays. One of those application is an on line application for student enrollment. In this kind of application user will access two type of page, the first one is a page with information from database or not and the second one is a page with an insert action to database. With a basic installation of server, server will serve this two type of page and make a bottle neck condition for high access.

There is an idea to load balance every request to another computer, so every request can be handled. Usually, system administrator will use Nginx as a load balancer. But with multiple kind of request, response time to make every request handled, become unstable.

Another idea appear, every request will be load balance with content based priority. Content is represented as two type of page before. Node JS will be used as load balancer system and read every header of request and redirect request to specific worker. With MongoDB as a database of system, NodeJS will work consistently for every data that come to the system.

The result show that NodeJS and content-based priority will increase response time for increasing number of access from user. But the mechanism to make a worker focus on one type of page, make every request will be server well.

Kata-Kunci: Load Balance, Content Based Priority, NodeJS, MongoDB.

Halaman ini sengaja dikosongkan

KATA PENGANTAR

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

Alhamdulillahirabbil'alamin, segala puji bagi Allah SWT, yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul **RANCANG BANGUN SISTEM PENYEIMBANG BEBAN PADA KLASTER SERVER DENGAN PRIORITAS BERBASIS KONTEN DAN KONTROL KETERSEDIAAN LAYANAN**. 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 atas anugerahnya yang tidak terkira kepada penulis dan Nabi Muhammad SAW.
2. Bapak, Ibu, Mbak Nova 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 terselesaiannya Tugas Akhir ini.
4. Bapak Baskoro Adi P, S.Kom, M.Kom selaku pembimbing II yang juga telah membantu, membimbing, dan memotivasi penulis mulai dari pengerjaan proposal hingga terselesaiannya Tugas Akhir ini.

5. Ibu Dr. Eng. Nanik Suciati, S.Kom., M.Kom., selaku Kepala Jurusan Teknik Informatika ITS pada masa penggerjaan Tugas Akhir, 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, dan segenap dosen Teknik Informatika yang telah memberikan ilmu dan pengalamannya.
6. Teman-teman Laboratorium AJK, samihd, romen, vivi, harum, dimas, thiar, agus, surya, pur, eva, nisa, uul, wicak, zaza, daniel, nindy, rizma, asbun, oing, fatih, syukron yang selalu menghibur dan mendukung penulis dalam penggerjaan Tugas Akhir ini.
7. Yang selalu memberikan semangat, memasakkan makanan ketika penulis sakit, memarahi ketika penulis lupa dengan Tugas Akhir, dan memberikan hiburan ketika penulis terhenti pada penggerjaan Tugas Akhir.
8. Teman-teman Kidnapper PARESMAPA angkatan 21 yang selalu mendukung penulis untuk tidak segera lulus.
9. Teman-teman di Dinas Pendidikan Kota Surabaya yang memberikan semangat untuk segera menyelesaikan Tugas Akhir.
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, Desember 2015

Bahrul Halimi

DAFTAR ISI

ABSTRAK	ix
ABSTRACT	xi
Kata Pengantar	xiii
DAFTAR ISI	xv
DAFTAR TABEL	xix
DAFTAR GAMBAR	xxi
DAFTAR KODE SUMBER	xxiii
1 PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	3
1.3 Batasan Masalah	3
1.4 Tujuan	4
1.5 Manfaat	4
2 LANDASAN TEORI	5
2.1 Prioritas Berbasis Konten	5
2.2 Node JS	5
2.3 Angular JS	6
2.4 MongoDB	7
2.5 Apache JMeter	7
3 DESAIN DAN PERANCANGAN	9
3.1 Kasus Penggunaan	9
3.2 Arsitektur Sistem	11
3.2.1 Desain Umum Sistem	11
3.2.2 Desain Balancer	12

3.2.3	Desain Worker	13
3.2.4	Desain Halaman Admin	15
3.2.5	Desain Server Basis Data	16
4	Implementasi	17
4.1	Lingkungan Implementasi	17
4.2	Rincian Implementasi Balancer	17
4.2.1	Instalasi Paket untuk NodeJS	18
4.2.2	Koneksi ke Basis Data	20
4.2.3	Implementasi Balancer	21
4.3	Rincian Implementasi Worker	25
4.4	Rincian Implementasi Server Basis Data	25
4.5	Implementasi Halaman Admin	27
4.5.1	Pengendali /	27
4.5.2	Pengendali /cluster	28
4.5.3	Pengendali /path	29
5	Pengujian dan Evaluasi	31
5.1	Lingkungan Uji Coba	31
5.2	Skenario Uji Coba	33
5.2.1	Skenario Uji Fungsionalitas	34
5.2.2	Skenario Uji Performa	37
5.3	Hasil Uji Coba dan Evaluasi	37
5.3.1	Uji Fungsionalitas	38
5.3.2	Uji Performa	41
6	Penutup	49
6.1	Kesimpulan	49
6.2	Saran	50
DAFTAR PUSTAKA		51

A Instalasi Perangkat Lunak	53
A.1 Instalasi Node JS	53
A.2 Installasi MongoDB	53
B Kode Sumber	55
B.1 Load Balancer	55
B.2 Aktivitas Model	60
B.3 ClusterInsert Model	61
B.4 ClusterView Model	63
B.5 Path Model	66
B.6 Setting Model	67
C Grafik Uji Coba dengan berbagai Kombinasi	69
BIODATA PENULIS	73

Halaman ini sengaja dikosongkan

DAFTAR TABEL

3.1	Daftar Kode Kasus Penggunaan	10
3.1	Daftar Kode Kasus Penggunaan	11
3.2	Rute pada Halaman Admin	15
3.2	Rute pada Halaman Admin	16
4.1	Implementasi Pengendali /	27
4.1	Implementasi Pengendali /	28
4.2	Implementasi Pengendali /cluster	28
4.2	Implementasi Pengendali /cluster	29
4.3	Implementasi Pengendali /path	29
5.1	Kategori URL dan Klaster Worker	34
5.2	Implementasi Uji Fungsionalitas Halaman Admin	35
5.2	Implementasi Uji Fungsionalitas Halaman Admin	36
5.3	Daftar Akses URL dan Worker yang Melayani	38
5.3	Daftar Akses URL dan Worker yang Melayani	39
5.3	Daftar Akses URL dan Worker yang Melayani	40
5.4	Daftar Layani Worker	40
5.5	Uji Fungsionalitas Halaman Admin	41
5.6	Kombinasi Jumlah Worker dalam Klaster untuk Uji Coba	43
5.6	Kombinasi Jumlah Worker dalam Klaster untuk Uji Coba	44
5.7	Daftar Nilai Galat pada Akses Halaman	45
5.7	Daftar Nilai Galat pada Akses Halaman	46

Halaman ini sengaja dikosongkan

DAFTAR GAMBAR

2.1	Contoh Penggunaan NodeJS sebagai Web Server	6
2.2	Contoh Penggunaan Angular JS pada Halaman Sederhana	6
2.3	Contoh Bentuk Penyimpanan Data MongoDB	7
2.4	Contoh Antar Muka Apache JMeter	8
3.1	Digram Kasus Penggunaan	9
3.2	Desain Sistem Secara Umum	12
3.3	Desain Arsitektur Balancer	13
3.4	Diagram Interaksi antara Pengguna, Balancer dan Worker	14
3.5	Diagram Arsitektur Worker	14
3.6	Diagram Arsitektur Halaman Admin	15
3.7	Diagram Arsitektur Server Basis Data	16
5.1	Arsitektur Uji Coba	31
5.2	Arsitektur Jaringan PPDB Surabaya 2015	37
5.3	Perbandingan Penggunaan CPU pada 1500 thread	42
5.4	Penggunaan Memori pada 1500 Thread	43
5.5	Perbandingan Waktu Respon Nginx dan NodeJS	45
5.6	Distribusi Beban pada Klaster View	46
5.7	Perbandingan Distribusi Beban Cookie-Based dan Cookie-Based dan Round-Robin	47
A.1	Contoh Node JS Siap Digunakan	53
A.2	Contoh MongoDB dan Daftar Database	54
C.1	Kombinasi 1 Insert (bagus) dan 3 View	69
C.2	Kombinasi 1 Insert (jelek) dan 3 View	69
C.3	Kombinasi 2 Insert dan 2 View (campur)	70
C.4	Kombinasi 2 Insert (bagus) dan 2 View	70
C.5	Kombinasi 2 View (bagus) dan 2 Insert	71
C.6	Kombinasi 3 Insert dan 1 View (bagus)	71

C.7 Kombinasi 3 Insert dan 1 View (jelek)	72
---	----

DAFTAR KODE SUMBER

4.1	Isi Package.Json	18
4.2	Paket untuk Balancer	21
4.3	Koneksi Balancer ke MongoDB	21
4.4	Pengambilan Cookie Pengguna	22
4.5	Worker Default Melayani Cookie Kosong	23
4.6	Mekanisme Pencarian Worker Aktif	23
4.7	Konfigurasi MySQL pada Alamat IP AJK	26
4.8	Konfigurasi MySQL untuk Server Basis Data	26
B.1	Kode Sumber Lengkap Load Balancer	55
B.2	Aktivitas Model untuk Koleksi Aktivitas	60
B.3	ClusterInsert Model untuk Koleksi ClusterInsert	61
B.4	ClusterView Model untuk Koleksi ClusterView	63
B.5	Path Model untuk Koleksi URL	66
B.6	Setting Model untuk Koleksi Setting	67

Halaman ini sengaja dikosongkan

BAB 1

PENDAHULUAN

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

1.1 Latar Belakang

Semakin berkembangnya internet di masyarakat membuat penggunaan aplikasi berbasis web semakin diminati. Pengguna aplikasi berbasis web ini dapat dijumpai diberbagai aktivitas harian masyarakat, diantaranya bank *online*, *e-commerce*, reservasi tempat secara *online*, bahkan pendaftaran peserta didik baru secara *online* (PPDB Surabaya). Hal ini membuat penyedia layanan aplikasi berbasis web harus menyediakan servis yang layak sehingga aplikasi tetap berjalan dengan baik walaupun pengguna semakin bertambah.

Terjadinya *bottleneck* (penumpukan permintaan) menjadi tantangan tersendiri ketika pengembang tidak memperhatikan sumber dayanya dan berujung pada gagalnya permintaan pengguna [1]. Muncul gagasan awal dengan penggunaan kelompok server yang akan menangani permintaan ini. Kelompok server ini akan secara bergantian melayani setiap permintaan terhadap aplikasi berbasis web ini. Dengan adanya tugas bergantian ini dibutuhkan sebuah komputer yang bertugas membagi beban kerja kelompok server. Komputer ini biasa disebut pembagi muat atau *load balancer*. Sistem kerja dari *load balancer* ini menggunakan sebuah algoritma yang sudah ditanam untuk kemudian digunakan untuk memilih komputer mana yang harus melayani permintaan pengguna. Pada titik ini, banyak penyedia layanan web memutuskan menggunakan Nginx sebagai *load balancer*, dengan alasan kemampuannya menangani banyak permintaan, dibanginkan Apache [9][10].

Di sisi lain sebuah aplikasi berbasis web memiliki dua jenis ha-

laman yang mungkin di akses. Yang pertama adalah halaman berisi informasi, baik hasil *query* basis data maupun tidak, selanjutnya disebut halaman informasi dan yang kedua adalah halaman yang digunakan untuk mengirimkan data ke server, dalam hal ini berupa form pengisian informasi, selanjutnya disebut halaman daftar.

Dua jenis halaman ini memiliki kebutuhan yang berbeda. Untuk halaman informasi, pengguna mengharapkan akses yang cepat sedangkan untuk halaman daftar, pengguna mengharapkan data yang dimasukkan dapat diproses dengan aman. Padahal di dalam penggunaan mekanisme sebelumnya dan dengan teknologi yang ada, *load balancer* tidak dapat memisahkan dua jenis permintaan ini. Mekanisme yang ada sebelumnya hanya memisahkan banyak permintaan sesuai dengan ketersediaan server melayani pengguna. Padahal ketika proses pemasukan data di dalam halaman daftar, seharusnya bisa digunakan untuk melayani permintaan pada halaman informasi.

Muncullah gagasan lain mengenai pengelompokan permintaan berdasarkan konten yang diinginkan oleh pengguna. Pengelompokan ini didasarkan pada dua halaman sebelumnya, yakni halaman informasi dan halaman daftar. Tujuannya untuk mengatur penggunaan sumber daya yang digunakan. Dua kelompok server terpisah akan melayani masing-masing permintaan yang berbeda. Dengan permintaan satu tipe dalam satu kelompok server, membuat kerja server menjadi lebih terpusat dan mengurangi beban yang besar.

Berbeda dengan yang terjadi saat ini, sebuah server atau bahkan dalam sebuah kelompok server, harus melayani berbagai bentuk permintaan dari pengguna, sehingga menyebabkan beban kerja server meningkat. Bahkan waktu dalam penyelesaian suatu permintaan tidak dapat diukur dalam satuan waktu yang sama karena berbedanya bentuk permintaan pengguna.

Sementara itu di dalam kelompok server yang bekerja bergantian melayani permintaan, ada kalanya sebuah server mengalami

gangguan dan sama sekali tidak dapat melayani setiap permintaan pengguna. Padahal setiap permintaan yang ada masih diteruskan oleh load balancer pada server tersebut. Tidak adanya mekanisme untuk memindahkan permintaan dari server mati ke server yang masih aktif membuat akses ke sebuah web menjadi tidak maksimal.

Oleh karena itu dibangunlah sistem ini. Dengan adanya sistem load balancing menggunakan algoritma berbasis konten yang memisahkan antara halaman informasi dan halaman daftar diharapkan dapat meningkatkan jumlah pengguna suatu halaman web dengan banyaknya bentuk permintaan dari pengguna. Sistem ini juga akan dibandingkan dengan sistem yang sudah berjalan sebelumnya dengan studi kasus sistem Pendaftaran Peserta Didik Baru Surabaya tahun 2015 dengan Nginx sebagai *load balancer*.

1.2 Rumusan Masalah

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

1. Bagaimana membagi beban kerja server berdasarkan konten permintaan pengguna ?
2. Bagaimana menentukan pengelompokan server berdasarkan konten permintaan pengguna ?
3. Bagaimana meningkatkan jumlah pengakses pada halaman informasi dengan terpisahnya akses antara halaman informasi dan halaman daftar ?
4. Bagaimana menjaga pengguna tetap dilayani kelompok server yang tersedia hingga permintaan selesai ?
5. Bagaimana perbandingan penggunaan CPU dan memori antara sistem baru dan yang sudah berjalan sebelumnya ?

1.3 Batasan Masalah

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

1. Konten permintaan pengguna dilihat dari URL yang diakses.
2. Pendefinisian kelompok konten permintaan pengguna dilakukan manual oleh manusia.
3. Sistem pembagi beban kerja diimplementasikan untuk aplikasi berbasis web.
4. Kelompok server yang bekerja dibedakan dengan besar memori yang digunakan.
5. Sistem yang akan dibandingkan adalah sistem untuk PPDB Surabaya tahun 2015 dengan Nginx sebagai *load balancer*.

1.4 Tujuan

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

1. Mampu mengategorikan permintaan pengguna terhadap suatu web berdasarkan halaman yang diakses pengguna.
2. Mampu melayani banyaknya permintaan pengguna dengan mengandalkan pengelompokan komputer.
3. Mampu meningkatkan jumlah pengakses yang dilayani dengan berhasil oleh aplikasi dibandingkan dengan akses tanpa pemisahan jenis halaman yang diakses.

1.5 Manfaat

Dengan dibangunnya *load balancer* ini diharapkan jumlah pengakses yang mampu dilayani oleh kelompok server untuk halaman informasi menjadi lebih banyak dibandingkan dengan penggunaan algoritma dan teknologi *load balancing* yang sudah ada.

BAB 2

LANDASAN TEORI

2.1 Prioritas Berbasis Konten

Munculnya mekanisme ini didasarkan pada beberapa jenis permintaan pengguna yang mengakses suatu halaman web. Sebuah server melayani berbagai jenis permintaan akan memberikan waktu balasan yang beragam pula. Hal ini akan meningkatkan beban kerja server.

Dengan adanya pemisahan pengguna berdasarkan konten, kelompok server akan melayani setiap permintaan yang memang ditujukan untuknya server tersebut. Bentuk permintaan akan selalu sama sehingga waktu untuk melayani permintaan menjadi sama dan lebih terkontrol. Beban kerja server akan lebih ringan dengan adanya pembagian beban berdasarkan algoritma ini [1].

2.2 Node JS

Merupakan sebuah platform yang dibangun di atas Chrome's JavaScript runtime dengan teknologi V8 yang mendukung proses server yang bersifat long-running. Tidak seperti platform modern yang mengandalkan multithreading, NodeJS memilih menggunakan asynchronous I/O eventing. Karena inilah NodeJS mampu bekerja dengan konsumsi memori rendah [2][3].

Teknologi yang tidak memanfaatkan multi-thread ini memudahkan pengembang yang terkadang kesulitan mengatur sumberdaya yang digunakan thread. Karena tidak mungkin ada sumberdaya yang terkunci karena thread yang berjalan. Akhirnya banyak yang memanfaatkan kemampuan dasar NodeJS sebagai web server.

Dengan adanya callback untuk setiap penggunaan fungsi, memungkinkan setiap pemanggilan fungsi yang tidak menghasilkan apapun, NodeJS akan *sleep*.

```

const http = require('http');

const hostname = '127.0.0.1';
const port = 1337;

http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/plain' });
  res.end('Hello World\n');
}).listen(port, hostname, () => {
  console.log(`Server running at http://:${hostname}:${port}/`);
});

```

Gambar 2.1: Contoh Penggunaan NodeJS sebagai Web Server

2.3 Angular JS

Angular JS membantu dalam pembangunan halaman HTML menjadi lebih dinamis. Merupakan sebuah kumpulan alat bantu yang mampu bekerja baik dengan pustaka lainnya. Setiap fitur dapat di-modifikasi sesuai dengan kebutuhan aplikasi. Menjadi salah satu kerangka kerja yang memfasilitasi pembangunan aplikasi kompleks yang terorganisir dan mudah dirawat. Angular JS lebih dikenal pada kemampuannya melayani sebuah situs web yang menggunakan halaman tunggal untuk penyajian data yang beragam [4][5].

```

<!DOCTYPE html>
<html lang="en-US">
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>

<div ng-app="">
  <p>Name : <input type="text" ng-model="name"></p>
  <h1>Hello {{name}}</h1>
</div>

</body>
</html>

```

Gambar 2.2: Contoh Penggunaan Angular JS pada Halaman Sederhana

Dengan menggunakan berkas JavaScript Angular yang didapatkan dari CDN (*Content Delivery Network*) atau media lain, pe-

ngembang dapat dengan mudah menggunakan fitur yang ditawarkan Angular JS.

2.4 MongoDB

Merupakan salah satu NoSQL (Not only SQL) terkenal yang dirancang untuk mengelola polimorfik, obyek, dan struktur data yang terus berkembang. MongoDB adalah basis data open-source yang memungkinkan mengubah skema dengan cepat sementara fungsi yang diharapkan dari basis data tradisional masih berjalan [6][7].

```
{
  name: "sue",           ← field: value
  age: 26,               ← field: value
  status: "A",           ← field: value
  groups: [ "news", "sports" ] ← field: value
}
```

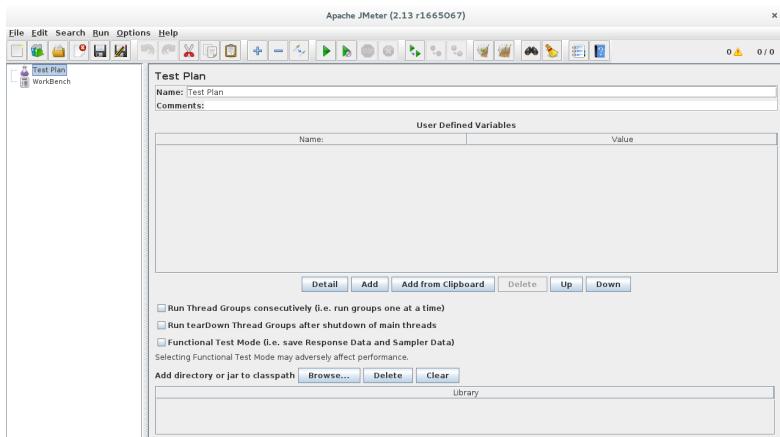
Gambar 2.3: Contoh Bentuk Penyimpanan Data MongoDB

Model penyimpanan yang menyerupai JSON membuat pengguna dapat dengan mudah mengakses dan mengubah data yang ada. Karena penyimpanannya yang menyerupai JSON, membuat struktur penyimpanan dapat berubah sewaktu-waktu tanpa mengubah konfigurasi sebelumnya.

2.5 Apache JMeter

Menjadi salah satu alat bantu untuk melakukan tes muat dan mengukur performa aplikasi, salah satunya berbasis web. Mampu melakukan pengujian pada berbagai protokol diantaranya Web, FTP, basis data, Mail (SMTP, POP3, IMAP), serta MongoDB [8].

Apache JMeter berbasis Java. Cara kerjanya yang menyerupai sebuah browser, karena desain awal memang ditujukan untuk meng-



Gambar 2.4: Contoh Antar Muka Apache JMeter

uji aplikasi berbasis web, mampu mengakses halaman website yang memiliki sumber daya statis maupun dinamis. Namun ada beberapa fitur browser yang tidak ditiru oleh Apache JMeter.

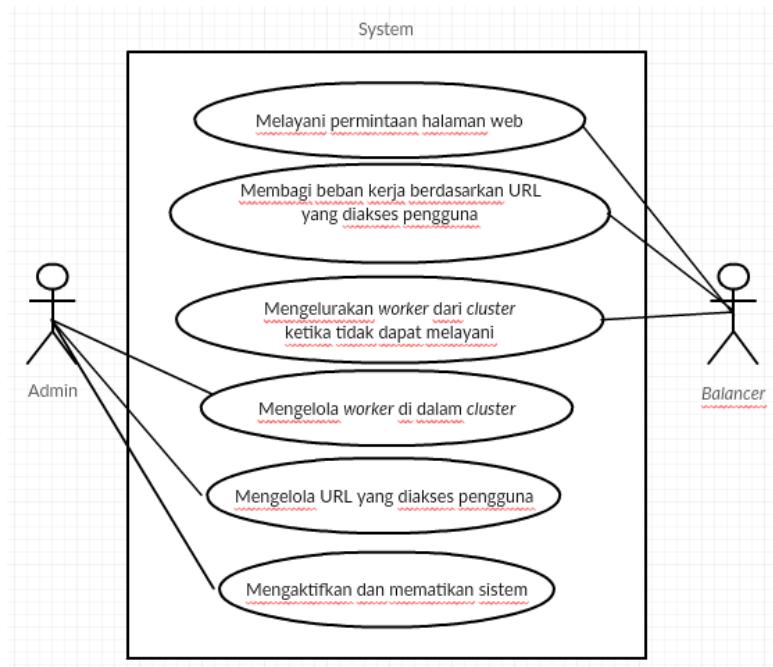
BAB 3

DESAIN DAN PERANCANGAN

Pada bab ini dibahas mengenai analisis dan perancangan sistem.

3.1 Kasus Penggunaan

Terdapat dua aktor dengan masing-masing tiga aktivitas dalam sistem yang digambarkan pada Gambar 3.1.



Gambar 3.1: Diagram 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	Melayani permintaan halaman web	Setiap permintaan halaman web akan mengarah ke sistem dan <i>balancer</i> akan melayani dengan cara meneruskan permintaan ke <i>worker</i> dan mengeruskan balasan ke pengguna
UC-0002	Membagi beban kerja berdasarkan URL yang diakses pengguna	Dibelakang sistem terdapat beberapa <i>worker</i> yang bekerja melayani permintaan terusan dari sistem, di sini lah <i>balancer</i> membagi beban kerja tersebut
UC-0003	Mengeluarkan <i>worker</i> dari klaster ketika tidak dapat melayani	Jika terdapat <i>worker</i> yang tidak dapat melayani permintaan, <i>balancer</i> akan mengeluarkan sementara dari tugas melayani hingga <i>worker</i> mampu memberikan layanan
UC-0004	Mengelola <i>worker</i> di dalam klaster	Admin dapat menambahkan dan mengurangi <i>worker</i> di dalam daftar klaster
UC-0005	Mengelola URL yang diakses pengguna	Admin dapat menambahkan dan mengurangi daftar URL ke dalam sistem

Tabel 3.1: Daftar Kode Kasus Penggunaan

Kode Kasus Penggunaan	Nama Kasus Penggunaan	Keterangan
UC-0006	Mengaktifkan dan mematikan sistem	Admin memiliki kendali atas aktif dan tidaknya sistem

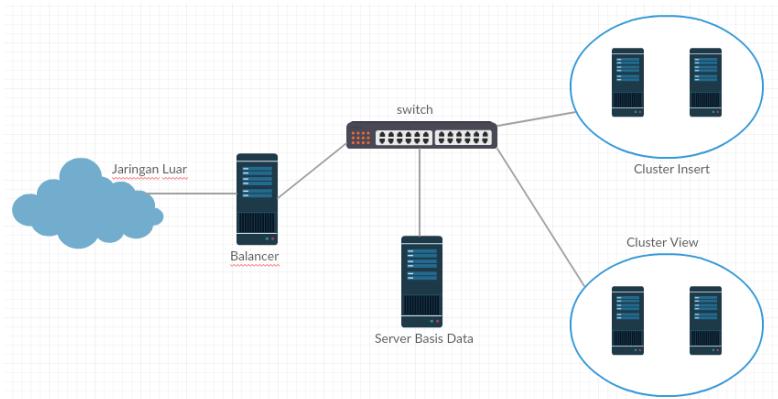
3.2 Arsitektur Sistem

Pada sub-bab ini, dibahas mengenai tahap analisis dan kebutuhan bisnis dan desain dari sistem yang akan dibangun.

3.2.1 Desain Umum Sistem

Sistem load balancing yang akan dibangun menggunakan teknologi Node JS dengan JavaScript sebagai bahasa pemrograman. Sistem ini akan berjalan pada port 80 dan menjadi target utama ketika pengguna akan menggunakan aplikasi berbasis web yang dikembangkan dengan menggunakan load balancer ini. Akan terdapat beberapa komputer pembantu, selanjutnya disebut *worker*, yang menerima perintah untuk melayani permintaan halaman website dari load balancer. *Worker* akan digolongkan menjadi dua cluster untuk melayani permintaan sesuai dengan URL yang diakses pengguna. Di sinilah algoritma berbasis konten diterapkan. Secara visual, desain sistem secara umum digambarkan pada Gambar 3.2.

Load balancer memiliki daftar *worker* dan URL yang akan digunakan dalam operasional load balancing. Daftar ini disimpan dalam MongoDB agar daftar yang dimiliki konsiste. MongoDB menyimpan data dalam bentuk JSON sehingga koneksi untuk mendapatkan daftar ini cepat.



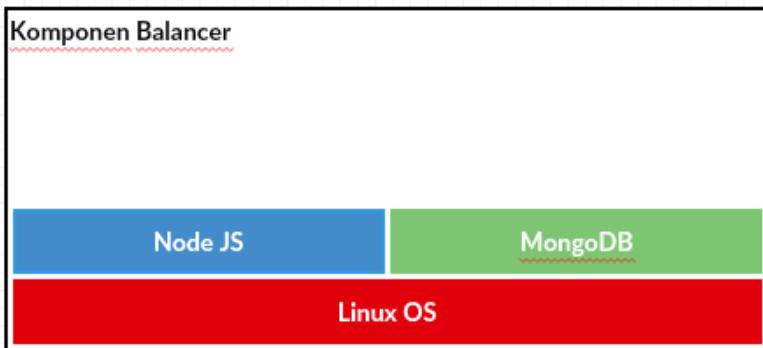
Gambar 3.2: Desain Sistem Secara Umum

Untuk memudahkan administrator sistem dalam mengkonfigurasikan *worker* dan menambahkan daftar URL yang ada, disediakan sebuah halaman admin yang berjalan pada port 3000. Selain dengan halaman admin ini, administrator sistem sudah dibantu untuk mengeluarkan *worker* yang tidak aktif dari pekerjaannya dalam proses load balancing oleh sistem secara otomatis dalam rentang waktu tertentu.

3.2.2 Desain Balancer

Balancer berperan penting dalam sistem. Setiap permintaan yang masuk ke dalam sistem akan diolah oleh balancer dan dikembalikan ke pengguna setelah mendapat balasan dari *worker*. Ada beberapa aplikasi yang sudah menyediakan fitur *balancing*, namun dalam Tugas Akhir ini digunakan NodeJS dan MongoDB untuk membangun fitur *balancing*. Diagram arsitektur Balancer tertera pada Gambar 3.3 dan diagram proses pengguna mengakses halaman web tertera pada Gambar 3.4.

NodeJS akan menyediakan servis web yang akan menangkap setiap permintaan pengguna. Sebelum diteruskan ke *worker*, *ba-*



Gambar 3.3: Desain Arsitektur Balancer

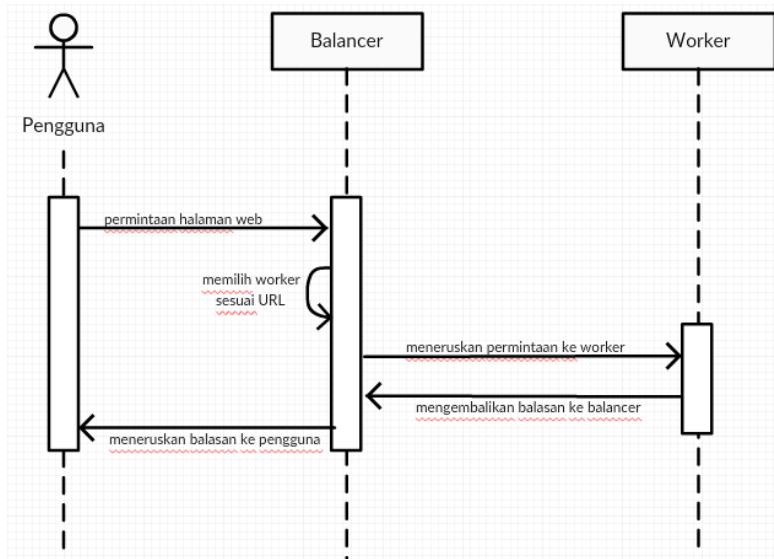
lancer akan membaca data tentang klaster di dalam MongoDB. Data di dalam database berupa *worker* yang siap melayani permintaan berdasarkan URL yang ada. Setelah *balancer* mendapatkan satu *worker* yang siap, permintaan diteruskan untuk mendapatkan balasan yang sesuai. Setelah mendapatkan balasan dari *worker*, balasan dikembalikan ke pengguna.

Setiap aktivitas yang terjadi di *balancer* akan dicatat dan disimpan pada MongoDB. Semua data yang dibutuhkan *balancer* tersimpan pada MongoDB dan disajikan pada halaman admin.

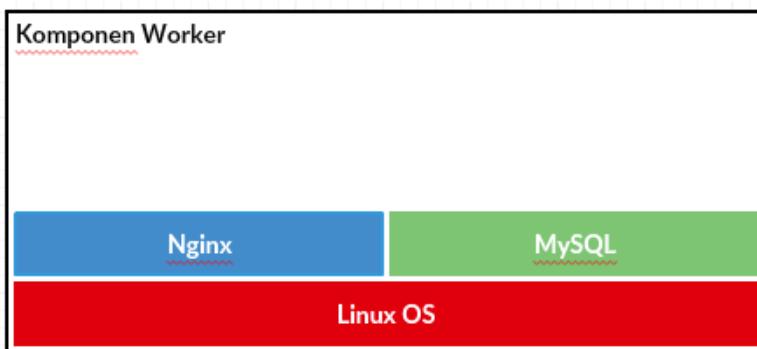
3.2.3 Desain Worker

Worker bekerja sebagai pemberi layanan. Menunggu permintaan yang diteruskan dari balancer dan mengembalikan lagi ke balancer. Aplikasi berbasis web yang akan diakses pengguna dijalankan pada *worker*. Diagram arsitektur *worker* tertera pada Gambar 3.5

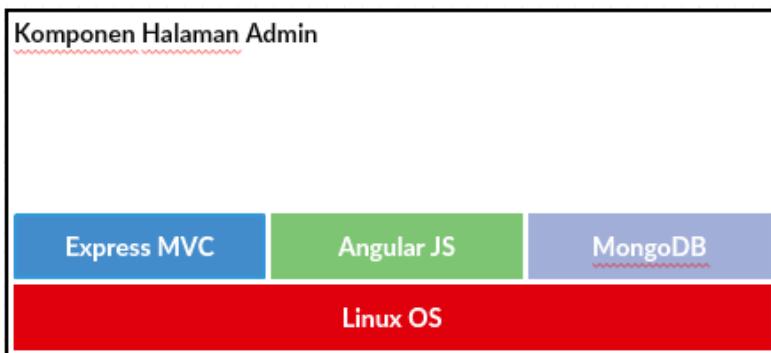
Di dalam sistem yang dibangun, peran *worker* sudah digambarkan pada diagram proses yang tertera pada Gambar 3.4. Pengguna tidak dapat mengakses *worker* secara langsung. Semua akses harus melalui *balancer*. Walaupun sebenarnya semua aplikasi berada pada *worker*.



Gambar 3.4: Diagram Interaksi antara Pengguna, Balancer dan Worker



Gambar 3.5: Diagram Arsitektur Worker



Gambar 3.6: Diagram Arsitektur Halaman Admin

3.2.4 Desain Halaman Admin

Halaman ini hanya akan diakses oleh administrator sistem. Adanya halaman ini ditujukan untuk memudahkan admin dalam mengelola cluster dan mengelola URL yang tersimpan di dalam basis data. Diagram arsitektur halaman admin tertera pada Gambar 3.6.

Express JS berperan dalam menangkap setiap rute yang diminta admin. Sementara Angular JS digunakan untuk menampilkan data sesuai dengan rute yang diminta admin. MongoDB yang digunakan dalam halaman admin sama dengan MongoDB yang digunakan oleh *balancer*. Daftar rute yang dapat diakses admin tertera pada Tabel 3.2.

Tabel 3.2: Rute pada Halaman Admin

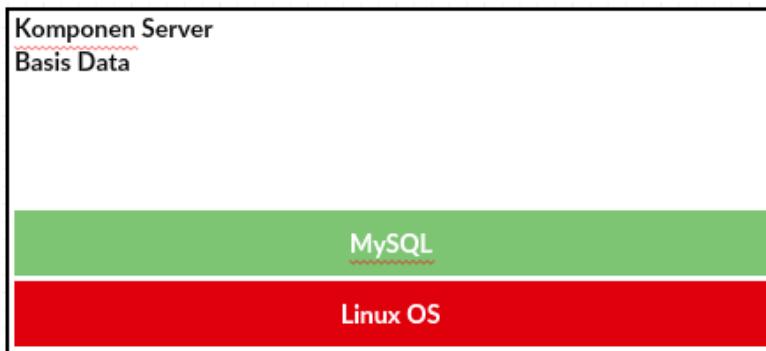
No	Rute	Metode	Aksi
1	/	GET, POST	Menampilkan status balancer dan operasi mengaktifkan/mematikan <i>balancer</i>
2	/cluster	GET, POST	Mengelola <i>worker</i> di dalam klaster

Tabel 3.2: Rute pada Halaman Admin

No	Rute	Metode	Aksi
3	/path	GET, POST	Mengelola rute pada aplikasi sesuai dengan klaster

3.2.5 Desain Server Basis Data

Dalam menjalankan sebuah aplikasi yang menuntut kemampuan akses hingga banyak pengguna, banyak pengembang yang memisahkan antara server aplikasi dan server basis data. Aplikasi yang berjalan dalam server ini hanya MySQL sebagai basis data seperti yang tertera pada 3.7.

**Gambar 3.7:** Diagram Arsitektur Server Basis Data

BAB 4

IMPLEMENTASI

Bab ini membahas implementasi sistem Load Balancing secara rinci. Pembahasan dilakukan secara rinci untuk setiap komponen yang ada yaitu: Balancer, Worker, Halaman Admin, dan Server Basis Data.

4.1 Lingkungan Implementasi

Lingkungan implementasi dan pengembangan dilakukan menggunakan komputer dengan spesifikasi Intel(R) Core(TM) i3-2120 CPU @ 3.30GHz dengan memori 8 GB di Laboratorium Arsitektur dan Jaringan Komputer, Teknik Informatika ITS. Perangkat lunak yang digunakan dalam pengembangan adalah sebagai berikut :

- Sistem Operasi Linux Ubuntu Server 14.04.01 LTS
- Desktop xfce4
- Editor teks vim
- Editor teks Sublime Text 2
- git versi 1.9.1 untuk pengolahan versi program
- NodeJS versi 4.2.3 untuk pengembangan aplikasi
- MongoDB versi 3.2.0 untuk basis data
- Express versi 4.13.1 untuk web server halaman admin
- Angular JS versi 1.5.0-beta.2 untuk pengolahan data pada halaman admin
- Nginx versi 1.4.6 untuk implementasi sistem PPDB Surabaya tahun 2015
- Paket L^AT_EX untuk pembuatan buku tugas akhir
- Peramban web Mozilla Firefox

4.2 Rincian Implementasi Balancer

Balancer dibangun dengan menggunakan Node JS, MongoDB dan beberapa paket yang diinstall secara terpisah. Paket-paket ini di-

install dengan menggunakan npm (<https://www.npmjs.com/>) yang memang digunakan sebagai *command line* oleh Node JS. Pada sub bab ini akan dijelaskan implementasi balancer hingga *balancer* siap digunakan.

4.2.1 Instalasi Paket untuk NodeJS

Seperti yang sudah dijelaskan sebelumnya, instalasi paket yang dibutuhkan menggunakan npm. Cara kerja npm ini hanya mengunduh paket untuk kebutuhan sistem, hanya saja beberapa paket bisa diinstall pada sistem operasi dan dapat digunakan pada *command line*. Untuk mengunduh paket yang dibutuhkan secara bersamaan, digunakan perintah `npm install`. Perintah ini akan membaca file `package.json` yang berisi daftar paket yang akan digunakan. Isi `package.json` untuk Tugas Akhir ini seperti pada Kode Sumber 4.1

Kode Sumber 4.1: Isi Package.Json

```
{  
  "name": "tugasakhir",  
  "version": "1.0.0",  
  "description": "Tugas Akhir ini berisi  
    tentang load balancer dengan node JS",  
  "dependencies": {  
    "body-parser": "~1.13.2",  
    "cookie-parser": "~1.3.5",  
    "debug": "~2.2.0",  
    "express": "~4.13.1",  
    "express-session": "^1.7.6",  
    "express-socket.io-session": "^1.3.1",  
    "hbs": "~3.1.0",  
    "helmet": "^0.10.0",  
    "http-proxy": "^1.12.0",  
    "moment": "~2.x.x",  
  }  
}
```

```
"mongoose": "^4.3.1",
"morgan": "~1.6.1",
"mysql": "~2.x.x",
"net-ping": "^1.1.12",
"node-sass-middleware": "0.8.0",
"request": "^2.67.0",
"serve-favicon": "~2.3.0",
"socket.io": "~1.x.x"
},
"devDependencies": {},
"scripts": {
  "test": "echo \\\"Error: no test specified\\\" && exit 1"
},
"repository": {
  "type": "git",
  "url": "git+https://github.com/bahrulhalimi/tugasakhir.git"
},
"keywords": [
  "balancer",
  "TA",
  "nodeJS"
],
"author": "Bahrul Halimi",
"license": "ISC",
"bugs": {
  "url": "https://github.com/bahrulhalimi/tugasakhir/issues"
},
"homepage": "https://github.com/bahrulhalimi/tugasakhir#readme"
}
```

Selain paket yang diinstall melalui perintah `npm install`, ada beberapa paket yang harus di install ke dalam sistem untuk bisa dipanggil melalui *command line*. Salah satu paket yang digunakan dalam Tugas Akhir ini adalah `forever`. Paket ini harus di install ke komputer, sehingga dapat digunakan melalui *command line*. Untuk menginstallnya digunakan perintah `sudo npm install forever -g`.

4.2.2 Koneksi ke Basis Data

Pada Tugas Akhir ini digunakan paket yang membantu untuk menghubungkan aplikasi dengan basis data yaitu paket `mongoose`. Untuk dapat terhubung dengan basis data, digunakan perintah `mongoose.connect('mongodb://127.0.0.1/tugasakhir')`

Karena basis data yang digunakan MongoDB dan bersifat NoSQL, maka struktur tabel menjadi tidak wajib di deklarasikan di awal. Untuk selanjutnya dibuat model untuk mendeklarasikan setiap koneksi ke basis data, sesuai dengan tabel yang digunakan. Dalam MongoDB istilah tabel menjadi koleksi (*collection*). Terdapat empat koleksi yang dibuat yaitu:

- `aktivitas_model`, digunakan untuk mencatat setiap permintaan ke sistem.
- `clusterinsert_model`, digunakan untuk mencatat daftar *worker* di dalam *cluster insert*.
- `clusterview_model`, digunakan untuk mencatat daftar *worker* di dalam *cluster view*.
- `path_model`, digunakan untuk mencatat daftar URL dan digolongkan ke dalam dua jenis, yaitu `insert` dan `view`.
- `setting_model`, digunakan untuk mencatat konfigurasi *balancer* terkait *worker default* dan waktu untuk melakukan pengecekan terhadap *worker* yang tidak aktif.

Kode program untuk setiap model tertera pada lampiran.

4.2.3 Implementasi Balancer

Mekanisme prioritas berbasis konten diterapkan untuk menjalankan balancer. Mekanisme ini hanya akan memilih *worker* sesuai dengan klaster yang aktif dan siap melayani permintaan. Daftar *worker* yang dapat digunakan didapatkan dari basis data yang diakses dengan menggunakan model yang sudah di bahas pada sub-bab sebelumnya. Selanjutnya setiap permintaan akan dilempar ke *worker* terpilih dan dilayani hingga permintaan selesai. Daftar paket yang akan digunakan tertera pada Kode Sumber 4.2.

Kode Sumber 4.2: Paket untuk Balancer

```
var http = require('http'),
    httpProxy = require('http-proxy'),
    proxy = httpProxy.createProxyServer({}),
    url = require('url'),
    mongoose = require('mongoose');
```

Balancer membutuhkan data yang berhubungan dengan *worker* yang aktif dan kategori URL yang akan diakses pengguna. Data ini didapatkan dengan melakukan koneksi ke MongoDB dengan menggunakan model yang dijelaskan sebelumnya. Kode program untuk melakukan koneksi ini tertera pada Kode Sumber 4.3.

Kode Sumber 4.3: Koneksi Balancer ke MongoDB

```
var clusterview_model = require('../model/
    clusterview_model')
var clusterinsert_model = require('../model/
    clusterinsert_model')
var aktivitas_model = require('../model/
    aktivitas_model')
var path_model = require('../model/path_model')
```

```
mongoose.connect('mongodb://127.0.0.1/
tugasakhir');
```

Balancer seakan-akan menjadi server yang menyediakan aplikasi berbasis web dan berjalan pada port 80. Untuk membuat servis ini, NodeJS perlu menjalankan servis sebagai web server dengan perintah `http.createServer(function(req, res) {})`. Sehingga semua permintaan pengguna dapat masuk ke dalam *balancer* dan diolah *balancer*.

Balancer akan memberikan *worker* yang aktif untuk melayani setiap pengguna yang masuk. Pasangan pengguna dan *worker* ini akan disimpan pada koleksi aktivitas sesuai dengan jenis halaman yang diakses. Untuk membedakan setiap pengguna, *balancer* akan membaca cookie dari pengguna dan menggunakan cookie ini sebagai id setiap pengguna. Pengambilan cookie tertera pada Kode Sumber 4.4

Kode Sumber 4.4: Pengambilan Cookie Pengguna

```
var cookie = req.headers.cookie;
var pengguna;
if (!cookie) cookie='kosong';
var awal = cookie.indexOf('csrf_cookie_name
');
var akhir = cookie.indexOf(';', awal)
if (akhir > awal) {
    pengguna = cookie.substring(awal, akhir)
}
else {
    pengguna = cookie.substring(awal, cookie.
        length)
}
```

Ketika pengguna yang masuk tidak memiliki cookie, cookie pengguna diisi dengan nilai 'kosong', kemudian pengguna akan dilayani

oleh *worker* default. *Worker* ini dipilih manual oleh administrator dan dikelola pada halaman admin. Tanpa melihat kategori URL yang diakses, pengguna akan dilayani oleh *worker* default. Kode Program 4.5 menjelaskan *worker* default melayani pengguna dengan cookie kosong.

Kode Sumber 4.5: Worker Default Melayani Cookie Kosong

```
if (cookie=='kosong') {
    setting_model.setting.findOne({ setting :"default" }, 'pakehurst', function(error,
        resul){
        if (error) console.log(error)
        petugas = resul.pakehurst
        proxy.web(req,res,{ target : 'http://'+
            petugas },function(ee,aa){
            console.log("ini error juga", ee)
        });
    })
}
```

Dalam Kode Program 4.5 muncul perintah `proxy.web()`. Fungsi ini adalah fungsi bawaan dari paket `http-proxy` yang akan menjalankan sistem proxy dimana permintaan pengguna dikirimkan ke komputer target, dalam kasus ini komputer target adalah *worker*. Fungsi ini akan digunakan setiap kali pengguna masuk dan dilayani *worker*.

Jika pengguna yang masuk memiliki cookie, cookie dan URL yang diakses pengguna akan dicek pada koleksi `aktivitas`. Jika ditemukan, *worker* yang melayani sesuai dengan yang ada pada data dari MongoDB. Namun ketika tidak ditemukan data, balancer akan memilihkan *worker* aktif dan yang siap melayani sesuai dengan kategori URL yang diakses. Pada Kode Program 4.6 tertera mekanisme untuk mendapatkan *worker* aktif.

Kode Sumber 4.6: Mekanisme Pencarian Worker Aktif

```

if (result.aksi == 'view') {
    clusterview_model.findOneClusterView(
        function(salah, hasil){
            if (salah) return console.log(salah)
            petugas = hasil.ip;
            clusterview_model.updateLayani(hasil.ip,
                function(wa, we){
                    if (wa) return console.log(wa)
                    console.log(we);
                })
            console.log(result.aksi)
            console.log(pathname[1])
            console.log("petugas : ", petugas);
            // tambah pengguna baru ke dalam basis
            data
            var kumpul = {cookie: pengguna, ip:
                petugas, aksi: result.aksi}
            aktivitas_model.tambahAktivitas(kumpul,
                function(errr, ress){
                    if (errr) return handleError(errr)
                    proxy.web(req, res, {target : 'http
                        ://'+petugas}, function(ee, aa){
                            if(ee) console.log(ee)
                        });
                })
            })
        }
    }
}

```

Kode Program 4.6 adalah contoh pada klaster view. Mekanisme tersebut juga dilakukan pada klaster insert untuk mendapatkan *worker*. Setelah mendapatkan *worker*, pasangan *worker*, cookie, dan kategori URL akan ditambahkan pada koleksi aksi yang nan-

tinya akan digunakan pada iterasi selanjutnya ketika ada pengguna masuk ke dalam sistem.

4.3 Rincian Implementasi Worker

Untuk melayani setiap permintaan dari pengguna, digunakan Nginx sebagai web server dengan sistem operasi Ubuntu Server 14.04.03. *Balancer* tidak menjalankan fungsi sebagai web server secara utuh karena semua permintaan akan dilayani *worker*. Untuk melakukan installasi kedua aplikasi tersebut digunakan perintah sebagai berikut:

- `sudo apt-get install nginx`. Perintah ini digunakan untuk instalasi Nginx ke dalam sistem operasi. Untuk dapat melayani halaman dinamis seperti PHP, diperlukan tambahan aplikasi lain yaitu `php5-fpm`. Untuk menginstalnya dijalankan perintah `sudo apt-get install php5-fpm`.
- Berhubungan dengan `php5-fpm`, karena aplikasi yang akan dijalankan menggunakan kerangka kerja CodeIgniter, dibutuhkan paket lain yang dapat diinstall dengan perintah `sudo apt-get install php5-fpm php5-cgi php5-cli php5-mysql php5-curl php5-gd php5-idn php-pear php5-imagick php5-imap php5-mcrypt php5-memcache php5-mhash php5-pspell php5-recode php5-sqlite php5-tidy php5-xmlrpc php5-xsl (https://bayultimate.wordpress.com/2012/05/08/daftar-paket-php-untuk-nginx/)`

4.4 Rincian Implementasi Server Basis Data

Hanya ada satu aplikasi yang berjalan yaitu MySQL yang memberikan servis basis data kepada *worker*. Konfigurasi lanjut dibutuhkan agar setiap *worker* dapat menggunakan basis data yang ada. Salah satunya adalah alamat IP yang digunakan server harus dikenali lingkungan implementasi, dalam hal ini lingkungan di Laboratori-

um AJK. Konfigurasi basis data agar berjalan dengan menggunakan IP tertera pada Kode Sumber 4.7

Kode Sumber 4.7: Konfigurasi MySQL pada Alamat IP AJK

bind-address	= 10.151.36.205
--------------	-----------------

Worker harus diberi akses ke basis data dengan menggunakan user yang ditentukan. Pada Tugas Akhir ini karena tidak dibutuhkan keamanan yang tinggi maka perintah untuk menambahkan akses diberikan untuk semua alamat yang akan mengakses basis data. Perintah yang dijalankan ada pada baris perintah MySQL, sehingga perlu masuk ke MySQL sebagai user root atau user lain yang memiliki hak akses untuk menambahkan pengguna ke dalam basis data. Berikut adalah urutan perintah untuk menambahkan akses.

- CREATE USER 'input'@'%' IDENTIFIED BY 'a';. Perintah ini akan menambahkan pengguna `input` dari semua alamat dengan password `a`.
- GRANT ALL PRIVILEGES ON ppdb.* TO 'input'@'%';. Perintah ini akan memberikan akses pengguna `input` ke semua tabel yang ada di basis data `ppdb`.

Konfigurasi lain yang dapat meningkatkan koneksi ke basis data juga diatur, dalam kasus ini variabel `max_connections` dapat diperbesar nilainya untuk mendapatkan koneksi yang lebih banyak. Konfigurasi yang diubah untuk mendapatkan hasil yang maksimal pada basis data tertera pada Kode Sumber 4.8

Kode Sumber 4.8: Konfigurasi MySQL untuk Server Basis Data

key_buffer	= 64M
max_allowed_packet	= 16M
max_connections	= 1024
query_cache_size	= 64M

4.5 Implementasi Halaman Admin

Implementasi Halaman Admin dibagi sesuai dengan rute yang dijelaskan pada Tabel 3.2. Untuk setiap rute dibuat dalam pengendali / controller terpisah.

4.5.1 Pengendali /

Pengendali ini berfungsi untuk aktivasi fungsi *balancer* dan memberikan nilai untuk dua pengaturan utama pada fungsi *balancer*. Dua pengaturan ini adalah *worker* default yang akan digunakan *balancer* ketika permintaan yang masuk tidak memiliki *cookie* dan waktu yang digunakan *balancer* untuk melakukan pengecekan anggota klaster yang digunakan. Penjelasan implementasi pengendali terdapat pada Tabel 4.1.

Tabel 4.1: Implementasi Pengendali /

No	Rute	Masukan	Luaran	Proses
1	GET /	-	status balancer, waktu, alamat IP	Data ditampilkan dengan teks dan ikon
2	POST /	-	status balancer	Kirim perintah aktifkan/matikan; Ubah tampilan status balancer

Tabel 4.1: Implementasi Pengendali /

No	Rute	Masukan	Luaran	Proses
3	POST /	alamat IP	alamat IP	Kirim IP <i>worker</i> default; Ubah tampilan <i>worker</i> default
4	POST /	waktu	waktu	Kirim waktu untuk pengecekan; Ubah waktu pengecekan

4.5.2 Pengendali /cluster

Pengendali ini berfungsi untuk mendaftar *worker* aktif dan tidak aktif di dalam klaster. Terdapat dua klaster yaitu klaster view dan klaster insert. Penjelasan implementasi pengendali terdapat pada Tabel 4.2.

Tabel 4.2: Implementasi Pengendali /cluster

No	Rute	Masukan	Luaran	Proses
1	GET /	-	Daftar <i>worker</i>	Daftar <i>worker</i> ditampilkan pada tabel, dipisahkan berdasarkan klaster

Tabel 4.2: Implementasi Pengendali /cluster

No	Rute	Masukan	Luaran	Proses
2	POST /	alamat IP	Daftar <i>worker</i>	Worker baru ditambahkan sesuai klaster. Daftar <i>worker</i> di update.

4.5.3 Pengendali /path

Pengendali ini berfungsi untuk mendaftarkan URL dan kategorinya. Dua kategori sesuai dengan dua klaster yang ada pada pengendali /cluster. Penjelasan implementasi pengendali /path tertera pada Tabel 4.3.

Tabel 4.3: Implementasi Pengendali /path

No	Rute	Masukan	Luaran	Proses
1	GET /	-	Daftar URL	Daftar URL dan kategori ditampilkan dalam tabel
2	POST /	URL, Kategori	Daftar URL	URL baru ditambahkan, daftar URL diperbarui

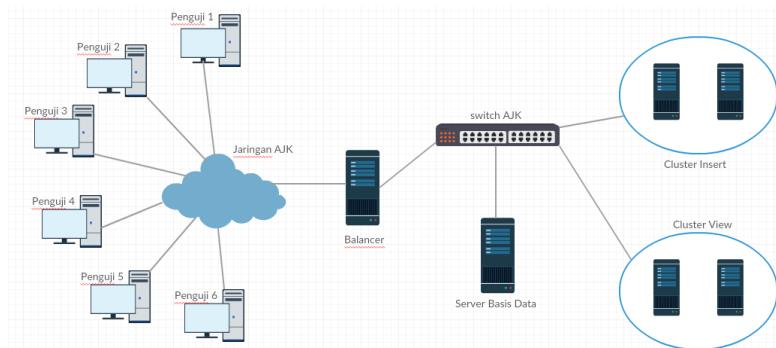
Halaman ini sengaja dikosongkan

BAB 5

PENGUJIAN DAN EVALUASI

5.1 Lingkungan Uji Coba

Lingkungan pengujian menggunakan empat komponen yang terdiri dari : satu komputer balancer, empat komputer *worker*, satu komputer basis data, dan enam komputer penguji. Semua komponen menggunakan komputer fisik dan tidak menggunakan virtualisasi. Arsitektur jaringan tertera pada Gambar 5.1. Pengujian dilakukan di Laboratorium Arsitektur dan Jaringan Komputer Jurusan Teknik Informatika ITS.



Gambar 5.1: Arsitektur Uji Coba

Spesifikasi untuk setiap komponen yang digunakan adalah sebagai berikut:

- Balancer:
 - Perangkat Keras:
 - * Processor Intel(R) Core(TM) i3-2120 CPU @ 3.30GHz
 - * RAM 8192 MB
 - * Hard disk 130 GB
 - Perangkat Lunak:
 - * Sistem operasi Ubuntu LTS 14.04.03

- * Nginx 1.4.6
 - * Node JS 4.2.3
 - * MongoDB 3.2.0
 - * PHP 5.5.9
 - * OpenSSH
- Worker:
 - Worker 1:
 - * Perangkat Keras:
 - Processor Intel(R) Core(TM)2 CPU 4400 @ 2.00GHz
 - RAM 2048 GB
 - Hard disk 80 GB
 - * Perangkat Lunak:
 - Sistem operasi Ubuntu LTS 14.04.03
 - Nginx 1.4.6
 - PHP 5.5.9
 - OpenSSH
 - Worker 2:
 - * Perangkat Keras:
 - Processor Intel(R) Core(TM)2 Duo CPU E7200 @ 2.53GHz
 - RAM 2048 GB
 - Hard disk 250 GB
 - * Perangkat Lunak:
 - Sistem operasi Ubuntu LTS 14.04.03
 - Nginx 1.4.6
 - PHP 5.5.9
 - OpenSSH
 - Worker 3 dan 4:
 - * Perangkat Keras:
 - Processor Intel(R) Core(TM) i3-2120 CPU @ 3.30GHz
 - RAM 4096 GB 33

- Hard disk 500 GB
- * Perangkat Lunak:
 - Sistem operasi Ubuntu LTS 14.04.03
 - Nginx 1.4.6
 - PHP 5.5.9
 - OpenSSH
- Server basis data:
 - Perangkat Keras:
 - * Processor Intel(R) Core(TM) i3-2120 CPU @ 3.30GHz
 - * RAM 4096 GB
 - * Hard disk 500 GB
 - Perangkat Lunak:
 - * Sistem operasi Ubuntu LTS 14.04.03
 - * MySQL 14.14 Distrib 5.5.44
- Komputer Penguji:
 - Perangkat Lunak:
 - * Sistem operasi Windows 7 dan Windows 8.1
 - * Apache Jmeter 2.13

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

- Balancer pada 10.151.36.37
- Worker 1 pada 10.151.36.21
- Worker 2 pada 10.151.36.34
- Worker 3 pada 10.151.36.201
- Worker 4 pada 10.151.36.203
- Server Basis Data pada 10.151.36.205
- Penguji pada 10.151.36.24, 10.151.36.25, 10.151.36.26, 10.151.36.27, 10.151.36.31 dan 10.151.36.32

5.2 Skenario Uji Coba

Pengujian menggunakan aplikasi berbasis web yang dijalankan pada *worker*. Aplikasi yang digunakan 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. Sistem sendiri memiliki 3 fitur utama yaitu pembagi beban, kontrol layanan dan halaman admin.
- **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

Dengan adanya tiga fitur pada balancer, sehingga pengujian akan dilakukan pada tiga fitur secara terpisah.

- **Pembagi Beban**

Sistem yang berjalan akan memisahkan permintaan yang menuju ke halaman insert dan halaman view. Pada pengujian ini diharapkan setiap permintaan diarahkan ke klaster yang sesuai. Permintaan dari pengguna dilihat dari URL yang diakses. Kategori URL dan *worker* yang melayani tertera pada Tabel 5.1.

Tabel 5.1: Kategori URL dan Klaster Worker

Cookie	Kategori URL	Alamat IP Worker
Ada	View	10.151.36.21
		10.151.36.34
Ada	Insert	10.151.36.201
		10.151.36.203
Tidak ada	View, Insert	10.151.36.201

- **Kontrol Layanan**

Kontrol Layanan akan mengeluarkan secara sementara *worker* yang sedang tidak aktif dari klasternya. Ketika *worker* su-

dah kembali pada fase aktif dan siap melayani, kontrol layanan akan kembali memasukkan *worker* ke dalam sistem. Pengujian ini akan memastikan *worker* yang sedang tidak aktif, benar-benar tidak mendapatkan aliran permintaan dari pengguna.

Pengujian dilakukan sebanyak dua kali dengan masing-masing pengujian sebanyak 300 thread. Pada pengujian pertama semua *worker* akan diaktifkan dan melayani setiap permintaan. Pada pengujian kedua salah satu *worker* akan dimatikan layannya tanpa mengubah konfigurasi pada *load balancer*. *Worker* yang tidak melayani tidak akan mengalami penambahan angka layani pada tampilan halaman admin.

- **Halaman Admin**

Pengujian dilakukan dengan pengaksesan halaman admin yang disediakan sistem dan disesuaikan dengan harapan luaran yang didapatkan. Beberapa hal yang akan diuji berdasarkan pengendali pada halaman admin seperti yang tertera pada Tabel 5.2.

Tabel 5.2: Implementasi Uji Fungsionalitas Halaman Admin

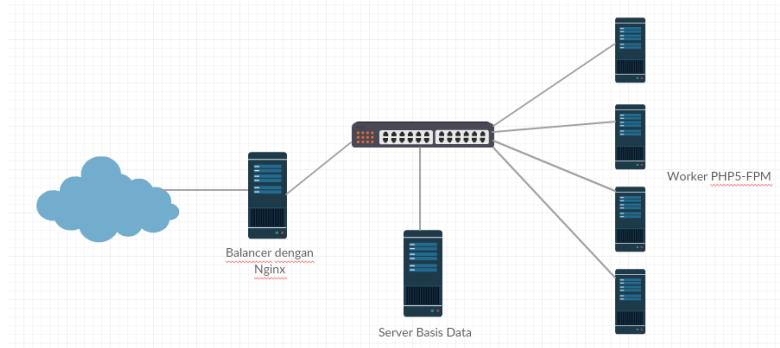
No	Pengendali	Uji Coba	Hasil Harapan
1	/index	Melihat status balancer, IP default, waktu cek	Status balancer (aktif atau tidak), IP worker default tampil, waktu cek tampil
		Mengaktifkan balancer	Status balancer aktif dan servis balancer berjalan

Tabel 5.2: Implementasi Uji Fungsionalitas Halaman Admin

No	Pengendali	Uji Coba	Hasil Harapan
		Mematikan balancer	Status balancer nonaktif dan servis balancer mati
		Mengganti IP worker default	IP worker berubah dan ditampilkan
		Mengganti waktu cek	Waktu cek berubah dan ditampilkan
2	/cluster	Melihat daftar <i>worker</i> pada klaster	Daftar <i>worker</i> dalam klaster
		Menambahkan <i>worker</i> ke dalam klaster	Worker dalam klaster bertambah
		Menghapus <i>worker</i> dari klaster	Worker dalam klaster berkurang
3	/path	Melihat daftar URL untuk setiap klaster	Daftar URL
		Menambahkan URL ke dalam klaster	URL dalam klaster bertambah
		Menghapus URL dari dalam klaster	URL dalam klaster berkurang

5.2.2 Skenario Uji Performa

Pada pengujian ini dilakukan benchmark dengan menggunakan aplikasi berbasis Java yaitu Apache JMeter (). Karena balancer melakukan pemisahan akses, pengujian akan berlangsung berupa insert dan view. Akses insert akan melakukan pendaftaran ke aplikasi PPDB Surabaya 2015 pada jenjang SMA Umum, sedangkan akses view akan menampilkan beberapa halaman pada aplikasi yang sama sekali tidak melakukan insert ke basis data. Apache JMeter akan membuat thread untuk setiap akses ke aplikasi. Pengujian akan berlangsung bertahap mulai dari 300, 600, 900, 1200 dan 1500 thread dalam satu detik. Dari hasil pengujian akan didapatkan waktu respon terhadap permintaan. Waktu ini akan dibandingkan dengan penggunaan arsitektur yang dibangun pada pelaksanaan PPDB Surabaya 2015. Arsitektur sistem PPDB Surabaya 2015 tertera pada Gambar 5.2.



Gambar 5.2: Arsitektur Jaringan PPDB Surabaya 2015

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 Pembagi Beban

Dilakukan pengujian acak untuk mengakses aplikasi PPDB Surabaya 2015 melalui balancer NodeJS. Hasil pengujian seperti tertera pada Tabel

Tabel 5.3: Daftar Akses URL dan Worker yang Melayani

Cookie	Kategori	URL	Worker	Hasil
kosong	insert	/pendaftaran/pilih_kk	10.151.36.201	OK
kosong	view	/umum/sambutan	10.151.36.201	OK
ada	view	/umum/ketentuan	10.151.36.21	OK
ada	insert	/pendaftaran	10.151.36.203	OK
ada	view	/umum/subrayon	10.151.36.21	OK
kosong	insert	/pendaftaran/pilih_kk	10.151.36.201	OK
kosong	view	/umum/sambutan	10.151.36.201	OK
ada	view	/umum/inklusif	10.151.36.21	OK
ada	view	/umum/jadwal	10.151.36.21	OK
ada	view	/umum/ketentuan	10.151.36.21	OK
ada	view	/rekap/lihat/sma/umum	10.151.36.21	OK
ada	insert	/pendaftaran	10.151.36.201	OK
ada	view	/umum/subrayon	10.151.36.21	OK
ada	insert	/pendaftaran/view	10.151.36.203	OK
ada	view	/umum/inklusif	10.151.36.21	OK
kosong	insert	/pendaftaran/pilih_kk	10.151.36.201	OK
kosong	view	/umum/sambutan	10.151.36.201	OK
ada	view	/umum/jadwal	10.151.36.21	OK
ada	view	/umum/ketentuan	10.151.36.21	OK
ada	view	/rekap/lihat/sma/umum	10.151.36.21	OK

Tabel 5.3: Daftar Akses URL dan Worker yang Melayani

Cookie	Kategori	URL	Worker	Hasil
ada	insert	/pendaftaran/submit	10.151.36.203	OK
ada	insert	/pendaftaran	10.151.36.203	OK
ada	view	/umum/subrayon	10.151.36.21	OK
ada	insert	/pendaftaran/view	10.151.36.201	OK
ada	view	/umum/inklusif	10.151.36.21	OK
ada	view	/umum/jadwal	10.151.36.21	OK
ada	view	/rekap/lihat/sma/umum	10.151.36.21	OK
ada	insert	/pendaftaran/submit	10.151.36.201	OK
ada	insert	/pendaftaran/submit	10.151.36.203	OK
ada	insert	/pendaftaran/view	10.151.36.203	OK
ada	insert	/pendaftaran/submit	10.151.36.201	OK
ada	insert	/pendaftaran/submit	10.151.36.203	OK
ada	insert	/pendaftaran/submit	10.151.36.203	OK
ada	insert	/pendaftaran/submit	10.151.36.203	OK
ada	insert	/pendaftaran/submit	10.151.36.203	OK
ada	insert	/pendaftaran/submit	10.151.36.203	OK
ada	insert	/pendaftaran/submit	10.151.36.203	OK
ada	insert	/pendaftaran/submit	10.151.36.203	OK
ada	insert	/pendaftaran/submit	10.151.36.203	OK
ada	insert	/pendaftaran/submit	10.151.36.203	OK
ada	insert	/pendaftaran/submit	10.151.36.203	OK
ada	insert	/pendaftaran/submit	10.151.36.203	OK
ada	insert	/pendaftaran/submit	10.151.36.203	OK
ada	view	/rekap/lihat/sma/umum	10.151.36.21	OK
ada	insert	/pendaftaran/submit	10.151.36.201	OK
ada	insert	/pendaftaran/submit	10.151.36.203	OK

Tabel 5.3: Daftar Akses URL dan Worker yang Melayani

Cookie	Kategori	URL	Worker	Hasil
ada	insert	/pendaftaran/submit	10.151.36.203	OK
ada	insert	/pendaftaran/submit	10.151.36.203	OK
ada	view	/rekap/lihat/sma/umum	10.151.36.21	OK
ada	view	/rekap/lihat/sma/umum	10.151.36.21	OK

Sesuai dengan skenario ujicoba yang diberikan pada Tabel 5.1, hasil ujicoba pada pembagian beban ke klaster yang sesuai dengan kategori 100% berhasil dilaksanakan. Cookie yang ada tidak ditampilkan untuk memberikan ruang pada penyajian data, karena panjang cookie setiap permintaan akan membuat penyajian data tidak mudah dibaca.

5.3.1.2 Kontrol Layanan

Sesuai dengan skenario pengujian yang akan dilakukan, pengujian dilakukan sebanyak dua kali. Servis yang dilayani worker untuk dua kali pengujian tertera pada Tabel 5.4.

Tabel 5.4: Daftar Layani Worker

Pengujian	Worker 1	Worker 2	Worker 3	Worker 4
1	75	75	150	149
2	0	150	150	150

Worker 1 tidak mendapatkan aliran permintaan dari pengguna karena worker 1 dimatikan pada pengujian kedua. Load balancer me-non-aktif-kan fungsi worker 1 sebagai pelayan. Sehingga tidak ada satupun permintaan yang dikirim ke worker 1. Kedua pengujian menghasilkan 100% sukses.

5.3.1.3 Halaman Admin

Pengujian dilakukan sesuai dengan skenario ujicoba. Hasil dan evaluasi yang didapatkan dari pengujian halaman admin tertera pada Tabel 5.5.

Tabel 5.5: Uji Fungsionalitas Halaman Admin

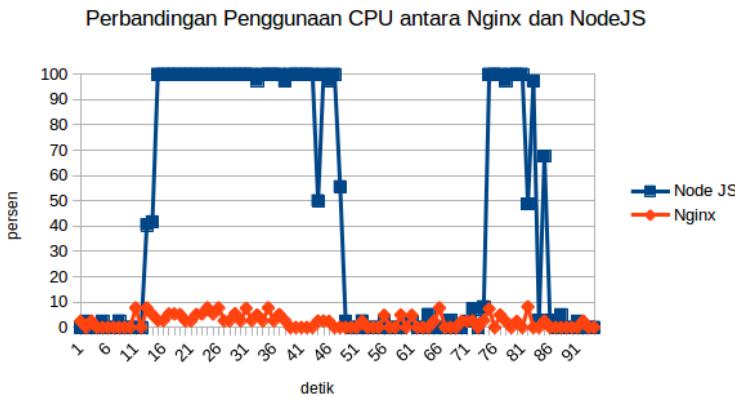
No	Pengendali	Uji Coba	Hasil
1	/index	Melihat status balancer, IP default, waktu cek	Sukses - 1190 ms
		Mengaktifkan balancer	Sukses - 354 ms
		Mematikan balancer	Sukses - 372 ms
		Mengganti IP worker default	Sukses - 17 ms
		Mengganti waktu cek	Sukses - 16 ms
2	/cluster	Melihat daftar <i>worker</i> pada klaster	Sukses - 1300 ms
		Menambahkan <i>worker</i> ke dalam klaster	Sukses - 105 ms
		Menghapus <i>worker</i> dari klaster	Sukses - 26 ms
3	/path	Melihat daftar URL untuk setiap klaster	Sukses - 1820 ms
		Menambahkan URL ke dalam klaster	Sukses - 38 ms
		Menghapus URL dari dalam klaster	Sukses - 48 ms

5.3.2 Uji Performa

Seperti yang sudah dijelaskan pada bab 5.2 pengujian performa dilakukan bertahap dengan menggunakan jumlah thread yang bertahap. Beberapa hal yang dibandingkan adalah sebagai berikut:

5.3.2.1 Penggunaan CPU dan Memori

Pengujian dilakukan hingga jumlah thread mencapai 1500. Hasil pengujian pada dua platform yaitu Nginx dan Node JS tertera pada Gambar 5.3.

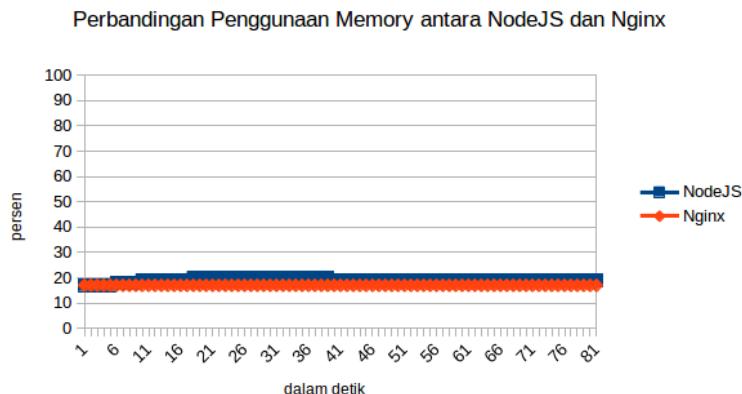


Gambar 5.3: Perbandingan Penggunaan CPU pada 1500 thread

Rata-rata penggunaan CPU ketika koneksi yang masuk berjumlah 1500 koneksi adalah 100% untuk NodeJS. Jika dibandingkan dengan penggunaan arsitektur lama pada aplikasi PPDB Surabaya 2015, balancer dengan NodeJS memiliki performas buruk dalam hal penggunaan CPU. Hal ini dikarenakan NodeJS akan membuat sebuah thread untuk setiap koneksi yang masuk ke sistem. Thread ini lah yang menjamin keberlangsungan proses yang berjalan secara asinkronus dapat tetap berjalan.

Sementara untuk penggunaan memori tidak ada perubahan yang signifikan. Ketika menggunakan thread berjumlah 1500 kenaikan penggunaan memori hanya mencapai 20% saja. Hasil pengujian penggunaan memori tertera pada Gambar 5.4.

Kedua platform memang tidak menggunakan banyak memori



Gambar 5.4: Penggunaan Memori pada 1500 Thread

untuk melayani permintaan dari pengguna. Nginx memanfaatkan multi-thread sementara NodeJS menggunakan single-thread. Mekanisme ini akan lebih memakan CPU dibandingkan memori.

5.3.2.2 Waktu Respon

Pengujian waktu respon dilakukan dengan melakukan insert dan view secara bersamaan ke dalam sistem. Waktu yang digunakan dicatat pada Apache JMeter dan disajikan pada tampilan grafik yang memudahkan untuk dibaca.

Pengujian dilakukan dengan kombinasi yang berbeda-beda pada setiap klaster. Kombinasi jumlah *worker* dalam klaster untuk pengujian tertera pada Tabel 5.6.

Tabel 5.6: Kombinasi Jumlah Worker dalam Klaster untuk Uji Coba

Klaster Insert	Klaster View
1 Worker (2 GB)	3 Worker
1 Worker (4 GB)	3 Worker

Tabel 5.6: Kombinasi Jumlah Worker dalam Klaster untuk Uji Coba

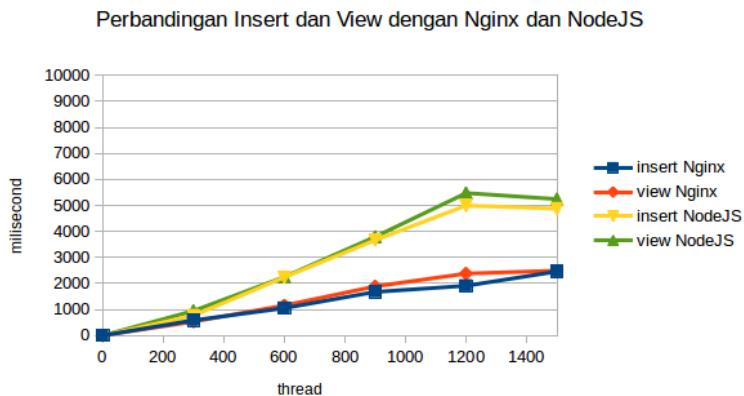
Klaster Insert	Klaster View
2 Worker (2 GB)	2 Worker (4 GB)
2 Worker (4 GB)	2 Worker (2 GB)
2 Worker (campur)	2 Worker (campur)
3 Worker	1 Worker (2 GB)
3 Worker	1 Worker (4 GB)
4 Worker	4 Worker

Untuk setiap kombinasi dilakukan pengujian dengan jumlah thread dari 300 hingga 1500 thread. Dari semua kombinasi yang dilakukan didapatkan hasil terbaik dengan waktu respon tercepat. Hasil waktu respon terbaik ini dibandingkan dengan hasil pengujian pada arsitektur sistem PPDB Surabaya 2015. Hasil yang didapatkan tertera pada Gambar 5.5.

Nginx bekerja lebih cepat dibandingkan dengan balancer dengan NodeJS. Pada kenyataannya yang digunakan Nginx adalah 4 worker untuk semua akses, sementara NodeJS membagi worker menjadi klaster tersendiri untuk melayani setiap permintaan. Jika NodeJS menggunakan setiap worker untuk semua akses, waktu respon yang didapatkan jauh lebih lama dibandingkan dengan worker yang dibagi. Hasil pengujiananya dapat dilihat pada lembar lampiran.

5.3.2.3 Permintaan Terlayani

Pada percobaan dengan 1500 thread, balancer Node JS dengan 4 *worker* pada klaster insert dan 1 *worker* pada klaster view memberikan ketersediaan layanan lebih baik dibandingkan dengan Nginx yang menggunakan 4 *worker* untuk semua aktivitas. Perbedaan ini didapatkan dari hasil benchmark yang kemudian di rata-rata nilai error yang muncul setelah benchmark selesai. Hasil yang didapat tertera pada Tabel 5.7.



Gambar 5.5: Perbandingan Waktu Respon Nginx dan NodeJS

Tabel 5.7: Daftar Nilai Galat pada Akses Halaman

Halaman	NodeJS	Nginx
Entry No UASBN	0,00%	21,89%
Entry PIN	0,00%	19,89%
Inklusif	0,00%	10,83%
Jadwal	0,00%	6,83%
Ketentuan	0,00%	17,17%
Open Pendaftaran	0,00%	9,00%
Pendaftaran Dalam Kota	0,00%	22,22%
Pilih Sekolah1	0,00%	18,34%
Pilih Sekolah2	0,00%	17,67%
Pilih Sub Rayon	0,00%	19,67%
Rekapsmaumum	0,00%	5,50%
Sambutan	0,00%	6,67%
Selesai	0,00%	16,56%
Simpan Permanen	0,00%	16,89%
Submit	0,00%	16,89%

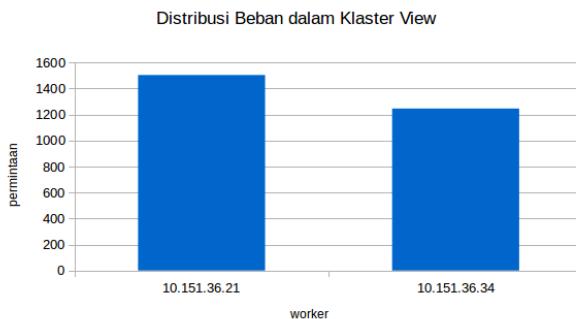
Tabel 5.7: Daftar Nilai Galat pada Akses Halaman

Halaman	NodeJS	Nginx
Subrayon	0,00%	12,17%
TOTAL	0,00%	15,60%

5.3.2.4 Distribusi Beban

Pada pengujian ini dilakukan benchmark dengan total 500 thread dimana 500 thread ini dipisah menjadi dua bagian. Bagian pertama menggunakan `cookie` yang teracak, dalam artian setiap thread yang berjalan akan menggunakan `cookie` yang berbeda-beda. Sementara bagian kedua adalah thread yang menggunakan 20 `cookie` yang sama dan berulang ketika setiap `cookie` sudah digunakan. Masing-masing bagian akan menjalankan 250 thread.

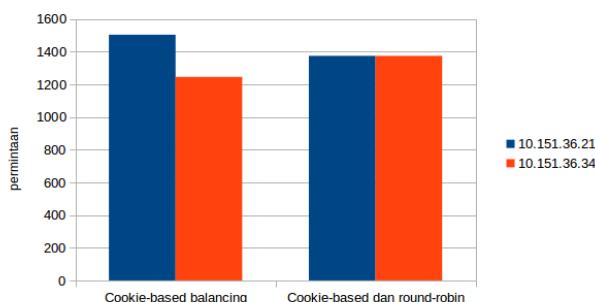
Pengujian hanya dilakukan pada salah satu klaster, yaitu klaster view. Pengujian ini tidak dilakukan pada dua klaster dan tidak dibandingkan karena memang beban untuk akses ke halaman informasi dan halaman insert berbeda. Hasil pengujian distribusi beban pada klaster view tertera pada Gambar 5.6.

**Gambar 5.6:** Distribusi Beban pada Klaster View

Dengan hasil yang tidak seimbang ini, kemudian dilakukan per-

ubahan mekanisme penyeimbang beban. Awal mekanisme yang digunakan adalah cookie-based dengan pengecekan apakah cookie sudah ada atau belum untuk mendapatkan worker yang akan melayani. Mekanisme ini memaksa pengguna yang sama dilayani oleh worker yang sama. Kemudian mekanisme diganti dengan menggunakan cookie-based dan round robin, dimana cookie tetap diambil namun *worker* yang melayani dipilih dengan menggunakan algoritma round robin.

Mekanisme baru diuji dengan melakukan benchmark yang sama seperti sebelumnya. Dibandingkan dengan mekanisme sebelumnya, hasil yang didapatkan tertera pada Gambar 5.7.



Gambar 5.7: Perbandingan Distribusi Beban Cookie-Based dan Cookie-Based dan Round-Robin

Halaman ini sengaja dikosongkan

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 kerja berdasarkan konten permintaan pengguna dilakukan dengan membaca URL yang diakses pengguna. Pembagian beban kerja berhasil dilakukan dengan pengujian beban kerja yang menghasilkan 100% sukses. Distribusi beban menjadi tidak merata ketika parameter yang digunakan untuk menjaga pengguna tetap dilayani oleh *worker* yang sama adalah `cookie`.
2. Berdasarkan hasil pengujian didapatkan penggunaan CPU yang meningkat sehingga menyebabkan waktu respon yang makin lama. Hal ini disebabkan NodeJS membuat sebuah thread untuk melayani setiap permintaan baru yang masuk ke sistem.
3. Kontrol ketersediaan layanan dapat dilakukan dengan melakukan pengecekan servis pada *worker*. Untuk mengeluarkan *worker* dari aktivitas melayani pengguna, tidak dibutuhkan mekanisme untuk merestart sistem. Mekanisme ini menggunakan basis data sebagai dasar balancer untuk meneruskan permintaan.
4. Dibandingkan dengan sistem yang sudah berjalan sebelumnya, dengan menggunakan Nginx sebagai balancer, NodeJS memberikan hasil penggunaan CPU yang buruk (100% untuk 1500 thread) sementara Nginx memberikan hasil yang memu-

askan (kurang dari 10% untuk 1500 thread).

6.2 Saran

Berikut beberapa saran yang diberikan untuk pengembangan lebih lanjut:

- 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*.
- Kontrol ketersediaan layanan perlu ditambah parameter untuk memastikan apakah *worker* benar-benar tidak bisa memberikan layanan ke pengguna.
- Diperlukan mekanisme lain untuk menjaga pengguna dilayani *worker* yang sama hingga akhir selain menggunakan `cooperative`. Salah satunya dengan round-robin, namun mekanisme ini tidak dapat memastikan *worker* yang sama melayani pengguna yang sama.

DAFTAR PUSTAKA

- [1] Saeed Sharifian, Seyed A. Motamed, Mohammad K. Akbari b, **A content-based load balancing algorithm with admission control for cluster web servers**, Future Generation Computer Systems, vol. 24, pp. 775-787, 2008
- [2] Joyent, Inc, **About Node.js**, [Online], <https://nodejs.org/en/about/>, diakses tanggal 01 April 2015
- [3] S. Tilkov and S. Vinoski, **Node.js: Using JavaScript to Build High-Performance Network Programs**, IEEE Computer Society Issue, vol. 6, pp. 80-83, 2010
- [4] Jeffrey D. Walker, Steven C. Chapra, **A client-side web application for interactive environmental simulation modeling**, Environmental Modelling & Software, vol. 55, pp. 49-60, 2014
- [5] Google, **Angular JS**, [Online], <https://angularjs.org/>, diakses tanggal 07 April 2015
- [6] Ming Zhang, Jing Zhang, Wei Zheng, Feiran Hu, Ge Zhuang, **A self-description data framework for Tokamak control system design**, Fusion Engineering and Design, p. 5, 2015
- [7] MongoDB, Inc, **MongoDB**, [Online], <https://www.mongodb.org/>, diakses tanggal 08 April 2015
- [8] Apache Software Foundation, **Apache JMeter**, [Online], <http://jmeter.apache.org/>, diakses tanggal 08 April 2015
- [9] Quora, **Why is nginx so efficient?**, [Online], <https://www.quora.com/Why-is-nginx-so-efficient>, diakses tanggal 03 Januari 2016
- [10] AOSABook, **nginx**, [Online], <http://www.aosabook.org/en/nginx.html>, diakses tanggal 03 Januari 2016

Halaman ini sengaja dikosongkan

LAMPIRAN A

INSTALASI PERANGKAT LUNAK

A.1 Instalasi Node JS

Ada banyak cara untuk menginstall NodeJS ke dalam komputer, salah satunya melalui kode sumber yang dapat diunduh pada halaman <https://nodejs.org/en/download/>. Setelah diunduh berkas perlu diekstrak untuk melanjutkan instalasi. Di dalam folder hasil ekstraksi, terdapat berkas README.md yang berisi langkah untuk menginstall NodeJS ke komputer. Semua langkah dilakukan melalui *command line* dan langkah installasinya adalah sebagai berikut

- Pindah ke folder dimana Node JS di ekstrak dengan perintah `cd {spathdownload}/node-v4.2.3/`.
- Di dalam folder jalankan perintah berikut secara berurutan `./configure, make, make install`.
- Jika terdapat pesan *error* mengenai beberapa pustaka yang belum terinstall sebelumnya, jalankan perintah `sudo apt-get install -f`.
- Semua perintah dilakukan pada hak akses *root*.

Untuk memastikan bahwa Node JS sudah berjalan dan siap digunakan, dijalankan perintah dan dihasilkan output seperti pada Gambar A.1.

```
uyung@laksmana:~$ node -e "console.log('Hello from nodejs ' + process.version)"  
Hello from nodejs v4.2.3  
uyung@laksmana:~$ █
```

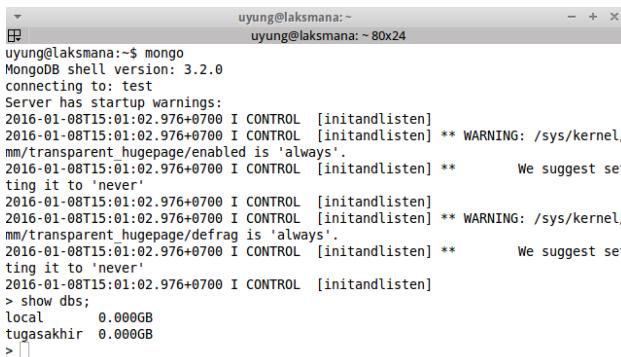
Gambar A.1: Contoh Node JS Siap Dugunakan

A.2 Installasi MongoDB

MongoDB menyediakan repositori tersendiri untuk mengunduh MongoDB. Setelah terkoneksi dengan repositori yang dimiliki Mongo-

DB, instalasi dapat berlanjut dengan menggunakan perintah apt-get. Berikut ini langkah-langkah instalasi MongoDB.

- Tambahkan *public key* yang digunakan untuk manajemen paket pada sistem operasi dengan perintah `sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10`.
- Buat file repositori untuk MongoDB pada sistem komputer dengan perintah `echo "deb http://repo.mongodb.org/apt/ubuntu trusty/mongodb-org/3.0 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.0.list`. Perintah ini disesuaikan dengan distro Linux yang digunakan.
- Perbaharui basis data manajemen paket dengan perintah `sudo apt-get update`.
- Install MongoDB ke komputer dengan perintah `sudo apt-get install -y mongodb-org`.
- Setelah instalasi selesai jalankan MongoDB dengan perintah `sudo service mongod start`.
- MongoDB siap digunakan dengan tampilan pada Gambar A.2.



```
uyung@laksmana:~$ mongo
MongoDB shell version: 3.2.0
connecting to: test
Server has startup warnings:
2016-01-08T15:01:02.976+0700 I CONTROL [initandlisten]
2016-01-08T15:01:02.976+0700 I CONTROL [initandlisten] ** WARNING: /sys/kernel/mm/transparent_hugepage/enabled is 'always'.
2016-01-08T15:01:02.976+0700 I CONTROL [initandlisten] **           We suggest setting it to 'never'
2016-01-08T15:01:02.976+0700 I CONTROL [initandlisten]
2016-01-08T15:01:02.976+0700 I CONTROL [initandlisten] ** WARNING: /sys/kernel/mm/transparent_hugepage/defrag is 'always'.
2016-01-08T15:01:02.976+0700 I CONTROL [initandlisten] **           We suggest setting it to 'never'
2016-01-08T15:01:02.976+0700 I CONTROL [initandlisten]
> show dbs;
local          0.000GB
tugasakhir   0.000GB
> 
```

Gambar A.2: Contoh MongoDB dan Daftar Database

LAMPIRAN B

KODE SUMBER

B.1 Load Balancer

Kode Sumber B.1: Kode Sumber Lengkap Load Balancer

```
var http = require('http'),
httpProxy = require('http-proxy'),
proxy = httpProxy.createProxyServer({}),
url = require('url'),
mongoose = require('mongoose');

var clusterview_model = require('../model/
    clusterview_model')
var clusterinsert_model = require('../model/
    clusterinsert_model')
var aktivitas_model = require('../model/
    aktivitas_model')
var path_model = require('../model/path_model
    ')
var setting_model = require('../model/
    setting_model')

mongoose.connect('mongodb://127.0.0.1/
    tugasakhir');

http.createServer(function(req, res) {

    var cookie = req.headers.cookie;
    var pengguna;
    if (!cookie) cookie='kosong';
    var awal = cookie.indexOf(
        csrf_cookie_name');
```

```
var akhir = cookie.indexOf(';', awal)
if (akhir > awal) {
    pengguna = cookie.substring(awal, akhir)
}
else {
    pengguna = cookie.substring(awal, cookie.length)
}

var fullpath = url.parse(req.url).pathname
;
var pathname = url.parse(req.url).pathname
.split('/');

var petugas;

if (pathname[1] == '') {pathname[1]='home'}

path_model.path.findOne({path: pathname[1]}, 'aksi', function(err, result){
    if (err) return handleError(err);
    console.log(result.aksi)
    if (cookie=='kosong') {
        setting_model.setting.findOne({setting :"default"}, 'pakehuruf', function(error, resul){
            if (error) console.log(error)
            petugas = resul.pakehuruf
            console.log(pengguna,";",result.aksi
            ,";",fullpath,";",petugas)
            proxy.web(req,res,{target : 'http
://'+ petugas},function(ee,aa){
```

```
        console.log("ini error juga", ee)
    });
})
}
else {
    var cari = {cookie: pengguna, aksi:
        result.aksi}
    aktivitas_model.findAktivitas(cari,
        function(erro, resu){
            if (erro) return console.log("kesalahan")
            if (!resu) {
                //dapatkan server yang dapat
                digunakan
                if (result.aksi == 'view') {
                    clusterview_model.
                        findOneClusterView(function(
                            salah, hasil){
                            if (salah) return console.log(
                                salah)
                            petugas = hasil.ip;
                            clusterview_model.updateLayani
                                (hasil.ip, function(wa, we)
                                    {
                                        if (wa) return console.log(
                                            wa)
                                        console.log(we);
                                    })
                }
                //tambah pengguna baru ke
                //dalam basis data
                var kumpul = {cookie: pengguna
                    , ip: petugas, aksi: result
                    }
```

```
    .aksi}
aktivitas_model.
tambahAktivitas(kumpul,
function(errr, ress){
if (errr) return handleError
(errr)
proxy.web(req, res, {target
: 'http://'+petugas},
function(ee,aa){
if(ee) console.log(ee)
}))
})
})
}
else {
clusterinsert_model.
findOneClusterInsert(function
(salah, hasil){
if (salah) return console.log(
salah)
petugas = hasil.ip;
clusterinsert_model.
updateLayani(hasil.ip,
function(wa, we){
if (wa) return console.log(
wa)
console.log(we);
})
// tambah pengguna baru ke
dalam basis data
var kumpul = {cookie: pengguna
, ip: petugas, aksi: result}
```

```
    .aksi}
aktivitas_model.
    tambahAktivitas(kumpul,
        function(errr, ress){
            if (errr) return handleError
                (errr)
            proxy.web(req, res, {target
                : 'http://'+petugas},
                function(ee, aa){
                    if(ee) console.log(ee)
                }));
            })
        })
    })
}
else {
    proxy.web(req, res, {target :
        'http://'+resu.ip}, function(ee,
            aa){
        if(ee) console.log("di sini error",
            ee)
    });
}
});
});
});
```

B.2 Aktivitas Model

Kode Sumber B.2: Aktivitas Model untuk Koleksi Aktivitas

```
var mongoose = require('mongoose');

var aktivitasSchema = new mongoose.Schema({
  cookie: String,
  ip: String,
  aksi: String
},{{
  collection : 'aktivitas'
}})

var aktivitas = mongoose.model('aktivitas',
  aktivitasSchema);

exports.aktivitas = aktivitas;

exports.findAktivitas = function(data,
  callback) {
  aktivitas.findOne({cookie: data.cookie,
    aksi: data.aksi}, 'aksi ip', function(
      err, resultIP){
    if(err) console.log(err)
    if(resultIP == null) {
      callback(err, null)
    }
    else {
      callback(err, resultIP)
    }
  })
}
```

```
exports.tambahAktivitas = function(data ,  
    callback){  
    aktivitas.create({cookie: data.cookie , ip:  
        data.ip , aksi: data.aksi} ,function(  
            errr , ress){  
            if (errr) return handleError(errr)  
            callback(errr , ress)  
        })  
}
```

B.3 ClusterInsert Model

Kode Sumber B.3: ClusterInsert Model untuk Koleksi ClusterInsert

```
var mongoose = require('mongoose');  
  
var clusterInsertSchema = new mongoose.  
    Schema({  
        ip : String ,  
        online : Boolean ,  
        layani : Number  
    },{  
        collection : 'clusterInsert'  
})  
  
var clusterInsert = mongoose.model(''  
    clusterInsert' , clusterInsertSchema);  
  
exports.clusterInsert = clusterInsert;  
  
exports.findClusterInsert = function(ip ,  
    callback) {
```

```
clusterInsert.findOne({ ip: ip }, function( err , resultIP ){
    if(err) console.log(err)
    if(resultIP == null) {
        callback(new Error("IP tidak ada"))
    }
    else {
        callback(err , resultIP)
    }
})

exports.findOneClusterInsert = function( callback ) {
    clusterInsert.findOne({ 'online': 1}, {ip
        online layani , {sort:{'layani':1},
        limit:1}, function(err , resultIP){
            if(err) console.log(err)
            if(resultIP == null) {
                callback(err , null)
            }
            else {
                callback(err , resultIP)
            }
        })
}

exports.updateLayani = function(ip , callback
    ) {
    clusterInsert.findOneAndUpdate({ ip: ip } , {
        $inc : {layani: 1}}, function(err ,
        result){
            if (err) return console.log(err)
        })
}
```

```
    callback(err , "sudah ditambahkan")
  })
}

exports.tambahClusterInsert = function(ip ,
  callback){
  exports.findClusterInsert(ip , function(err
    , resu){
    if(resu == null) {
      var newIP = new clusterInsert({ip: ip ,
        online: 1, layani: 0})
      newIP.save(function(errr){
        callback(errr ,newIP)
      })
    } else {
      callback(new Error("IP sudah ada"))
    }
  })
}

exports.hapusClusterInsert = function(ip ,
  callback){
  clusterInsert.findOneAndRemove({ip: ip} ,
    function(err , result){
    if (err) console.log(err)
    callback(err , result)
  })
}
```

B.4 ClusterView Model

Kode Sumber B.4: ClusterView Model untuk Koleksi ClusterView

```
var mongoose = require('mongoose');

var clusterViewSchema = new mongoose.Schema
  ({
    ip : String ,
    online : Boolean ,
    layani : Number
  },{
    collection : 'clusterView'
  })

var clusterView = mongoose.model(
  'clusterView' , clusterViewSchema);

exports.clusterView = clusterView;

exports.findClusterView = function(ip ,
  callback) {
  clusterView.findOne({ip: ip} , function(err ,
    resultIP){
    if(err) console.log(err)
    if(resultIP == null) {
      callback(new Error("IP tidak ada"))
    }
    else {
      callback(err , resultIP)
    }
  })
}

exports.findOneClusterView = function(
  callback) {
```

```
clusterView.findOne({ 'online' : 1}, 'ip
    online layani', {sort:{'layani':1},
    limit:1}, function(err, resultIP){
if(err) console.log(err)
if(resultIP == null) {
    callback(err, null)
}
else {
    callback(err, resultIP)
}
})
}

exports.updateLayani = function(ip, callback
) {
clusterView.findOneAndUpdate({ ip: ip }, {
    $inc : {layani: 1}}, function(err,
    result){
if (err) return console.log(err)
callback(err, "sudah ditambahkan")
})
}

exports.tambahClusterView = function(ip,
    callback){
exports.findClusterView(ip, function(err,
    resu){
if(resu == null) {
    var newIP = new clusterView({ip: ip,
        online: 1, layani: 0})
    newIP.save(function(errr){
        callback(errr, newIP)
    })
}
})
```

```

    } else {
      callback(new Error("IP sudah ada"))
    }
  })
}

exports.hapusClusterView = function(ip,
  callback){
  clusterView.findOneAndRemove({ip: ip},
    function(err, result){
      if (err) console.log(err)
      callback(err, result)
    })
}
}

```

B.5 Path Model

Kode Sumber B.5: Path Model untuk Koleksi URL

```

var mongoose = require('mongoose');

var pathSchema = new mongoose.Schema({
  path: String,
  aksi: String
},{ 
  collection : 'path'
})

var path = mongoose.model('path', pathSchema);

exports.path = path;

```

```

exports.simpanPathBaru = function(data,
    callback){
    var newPath = new pathI({path: data.path,
        aksi: data.aksi})
    newPath.save(function(errr){
        callback(errr,newPath)
    })
}

exports.hapusPath = function(path, callback)
{
    pathI.findOneAndRemove({path: path},
        function(err, result){
            if (err) console.log(err)
            callback(err, result)
    })
}

```

B.6 Setting Model

Kode Sumber B.6: Setting Model untuk Koleksi Setting

```

var mongoose = require('mongoose');

var pathSchema = new mongoose.Schema({
    setting: String,
    pakeangka: Number,
    pakehuruf: String
},{ 
    collection : 'setting'
})

```

```
var settingBalancer = mongoose.model('
  setting', pathSchema);

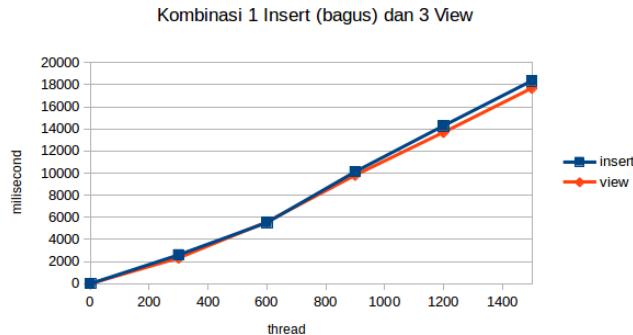
exports.setting = settingBalancer;

exports.ubahSettingIP = function(data,
  callback){
  settingBalancer.findOneAndUpdate({setting:
    data.setting}, {$set : {pakehuruf: data
      .ip}}, function(err, result){
    if (err) console.log(err)
    callback(err, result)
  })
}

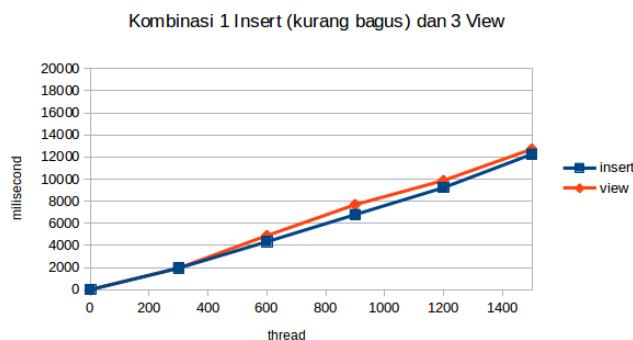
exports.ubahSettingWaktu = function(data,
  callback){
  settingBalancer.findOneAndUpdate({setting:
    data.setting}, {$set : {pakeangka: data
      .angka}}, function(err, result){
    if (err) console.log(err)
    callback(err, result)
  })
}
```

LAMPIRAN C

GRAFIK UJI COBA DENGAN BERBAGAI KOMBINASI

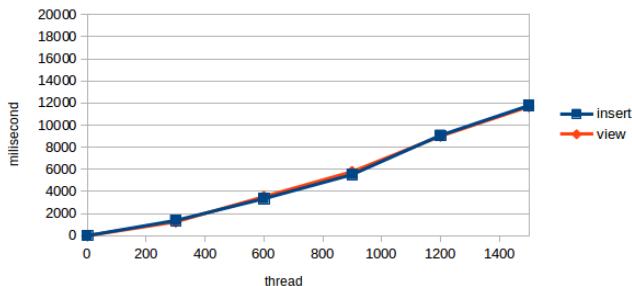


Gambar C.1: Kombinasi 1 Insert (bagus) dan 3 View



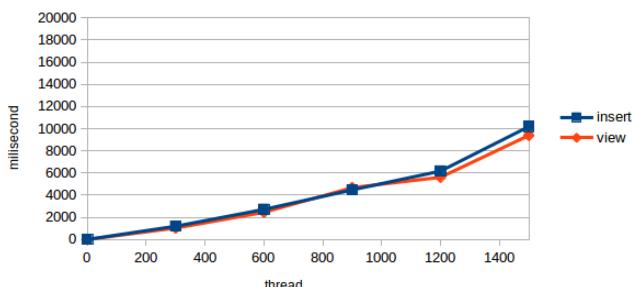
Gambar C.2: Kombinasi 1 Insert (jelek) dan 3 View

Kombinasi 2 Insert dan 2 View (campur)

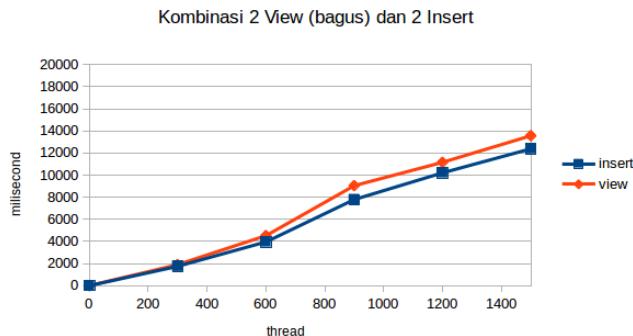


Gambar C.3: Kombinasi 2 Insert dan 2 View (campur)

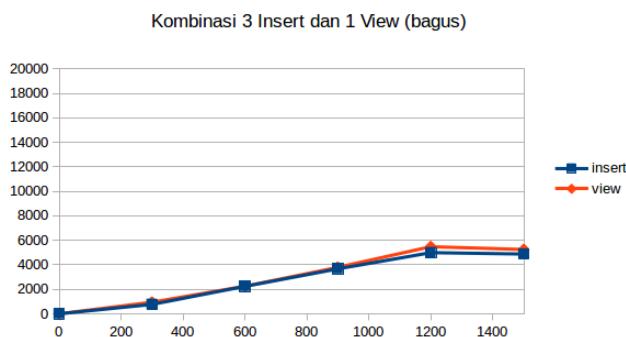
Kombinasi 2 Insert (bagus) dan 2 View



Gambar C.4: Kombinasi 2 Insert (bagus) dan 2 View

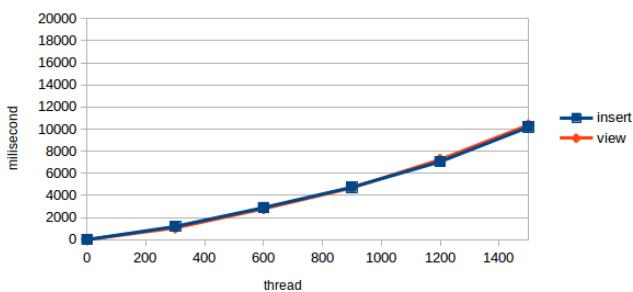


Gambar C.5: Kombinasi 2 View (bagus) dan 2 Insert



Gambar C.6: Kombinasi 3 Insert dan 1 View (bagus)

Waktu Respon Dengan Kombinasi 3 Insert 1 View

**Gambar C.7:** Kombinasi 3 Insert dan 1 View (jelek)

BIODATA PENULIS



Bahrul Halimi, seharusnya memiliki panggilan Rurul, namun karena hal lain menjadikan dia memiliki panggilan Uyung. Pada 20 November 1993 lahir di Pati, Jawa Tengah, kota kecil yang mulai terkenal dengan bisnis karaoke. 17 tahun tidak meninggalkan Pati dan akhirnya memutuskan untuk melanjutkan studi di Institut Teknologi Sepuluh Nopember Surabaya. Penulis mulai akrab dengan dunia komputer pada tingkat pendidikan Sekolah Dasar (SD), namun karena biaanya teknologi yang dipelajari tidak sesuai dengan perkembangan pada saat itu. Aktif di dunia koding pada tingkat pendidikan Sekolah Menengah Atas (SMA) dengan mengikuti Olimpiade Komputer, namun tidak pernah lolos pada tingkat provinsi. Hingga buku ini dikeluarkan, penulis masih mendalami ilmu di bidang jaringan pada sisi infrastruktur. Penulis dapat dihubungi melalui surel 31uyung10@gmail.com.