TESIS – TE142599

KAJIAN WEB LOAD BALANCING BERBASIS ROUND ROBIN DAN IP HASH UNTUK PENINGKATAN KINERJA LAYANAN SERVER

JANANTA PERMATA PUTRA
07111550060005

DOSEN PEMBIMBING
Dr. Supeno Mardi Susiki Nugroho, ST., MT.
Dr. Istas Pratomo, S.T., M.T.

PROGRAM MAGISTER
BIDANG KEAHLIAN TELEMATIKA
DEPARTEMEN TEKNIK ELEKTRO
FAKULTAS TEKNOLOGI ELEKTRO
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA

2018

TESIS – TE142599

KAJIAN WEB LOAD BALANCING BERBASIS ROUND ROBIN DAN IP HASH UNTUK PENINGKATAN KINERJA LAYANAN SERVER

JANANTA PERMATA PUTRA
07111550060005

DOSEN PEMBIMBING
Dr. Supeno Mardi Susiki Nugroho, ST., MT.
Dr. Istas Pratomo, S.T., M.T.

PROGRAM MAGISTER
BIDANG KEAHLIAN TELEMATIKA
DEPARTEMEN TEKNIK ELEKTRO
FAKULTAS TEKNOLOGI ELEKTRO
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2018

LEMBAR PENGESAHAN

Tesis disusun untuk memenuhi salah satu syarat memperoleh gelar
Magister Teknik (M.T)
di
Institut Teknologi Sepuluh Nopember

oleh:

Jananta Permata Putra
NRP. 07111550060005

Tanggal Ujian : 6 Juli 2018
Periode Wisuda : September 2018

Disetujui oleh:

1. Dr. Supeno Mardi Susiki Nugroho, ST., MT. (Pembimbing I)
   NIP: 197003131995121001
2. Dr. Istas Pratomo, S.T., M.T. (Pembimbing II)
   NIP: 197903252003121001
3. Dr. Ir. Achmad Affandi, DEA (Penguji)
   NIP: 196510241990021001
4. Eko Setijadi, ST., MT., Ph.D. (Penguji)
   NIP: 197210012003121002
5. Dr. Trihastuti Agustinah, ST., MT. (Penguji)
   NIP: 196808121994032001

Dekan Fakultas Teknologi Elektro

Dr. Ir. Arief Sardjono, S.T., M.T.
NIP. 197002121995121001

Halaman ini sengaja dikosongkan

PERNYATAAN KEASLIAN TESIS

Dengan ini saya menyatakan bahwa isi keseluruhan Tesis saya dengan judul "**KAJIAN WEB LOAD BALANCING BERBASIS ROUND ROBIN DAN IP HASH UNTUK PENINGKATAN KINERJA LAYANAN SERVER**" adalah benar-benar hasil karya intelektual mandiri, diselesaikan tanpa menggunakan bahan-bahan yang tidak diizinkan dan bukan merupakan karya pihak lain yang saya akui sebagai karya sendiri.

Semua referensi yang dikutip maupun dirujuk telah ditulis secara lengkap pada daftar pustaka. Apabila ternyata pernyataan ini tidak benar, saya bersedia menerima sanksi sesuai peraturan yang berlaku.

Surabaya, Juli 2018

Jananta Permata Putra
NRP. 07111550060005

Halaman ini sengaja dikosongkan

KAJIAN WEB LOAD BALANCING BERBASIS ROUND ROBIN DAN IP HASH UNTUK PENINGKATAN KINERJA LAYANAN SERVER

Nama mahasiswa : Jananta Permata Putra
NRP : 07111550060005
Pembimbing : 1. Dr. Supeno Mardi Susiki Nugroho, ST., MT.
2. Dr. Istas Pratomo, ST., MT.

ABSTRAK

Saat ini penggunaan akses terhadap layanan yang ada di Internet sangatlah penting. Salah satu dari layanan tersebut adalah layanan sistem informasi berupa web. Jika layanan tersebut tidak dapat diakses maka informasi yang dibutuhkan seseorang tidak akan didapatkan. Oleh karena itu server yang digunakan untuk aplikasi tersebut harus tetap nyala. Namun, dalam prakteknya sering terjadi kegagalan layanan tersebut karena server bermasalah, dalam proses perbaikan, maupun mendapatkan beban berlebih. Salah satu cara untuk mengatasinya adalah memasukkan aplikasi ke dalam beberapa server sekaligus, sehingga bila salah satu server bermasalah maka masih dapat dilayani oleh server lain. Digunakannya beberapa server juga membuat beban server menjadi ringan karena permintaan akses ke server dibagi ke beberapa server sekaligus. Hal tersebut biasanya disebut dengan penyeimbang beban web server. Di dalam penyeimbang beban web server juga terdapat metode-metode untuk menyeimbangkan beban server.

Dalam penelitian ini dibuat manajemen sistem berupa penyeimbang beban web server untuk meningkatkan ketersediaan yang tinggi. Cara yang digunakan adalah dengan menggunakan beberapa server untuk melayani layanan web yang didapatkan dan membagi beban ke beberapa server tersebut. Dalam penelitian ini juga didapatkan metode penyeimbang beban web server terbaik pada kondisi tertentu. Seperti pada kondisi spesifikasi server *upstream* yang sama didapatkan metode round robin mempunyai kesuksesan terbaik yaitu sebesar 99.905 persen. Sedangkan pada kondisi spesifikasi server yang berbeda didapatkan prosentase kesuksesan terbaik ada pada metode IP Hash yaitu sebesar 99,95 persen.

Kata kunci: Server Web, Penyeimbang beban.

Halaman ini sengaja dikosongkan

STUDY OF WEB LOAD BALANCING BASED ON ROUND ROBIN AND IP HASH FOR IMPROVING SERVER SERVICE PERFORMANCE

By	: Jananta Permata Putra
Student Identity Number	: 07111550060005
Supervisors	: 1. Dr. Supeno Mardi Susiki N., ST., MT.
	2. Dr. Istas Pratomo, ST., MT.

ABSTRACT

Nowadays, the access to Internet services is very important. One of these internet services is web service information system. If the service is inaccessible, then the required information will not be obtained. Therefore, the server for that application must remain on. However, in practice there is often a failure of the service because the server is in trouble, in the process of repair, or get excessive load. One way to overcome this, is to make the web application run in multiple servers at once, if one of the server is get problem then it can still be running in another server. The use of several servers also makes the server load becomes light because access to the server is divided into several servers at once. This is usually called a web server load balancer. In the web server load balancer there are also methods to balance the server load.

In this research is made management system based on web server load balancing to increase the high availability. The way it is used is to use multiple servers to serve web service request and share the load to those servers. In this study also found the best method to balance the web server load under certain conditions. As in the condition with same upstream server spesification, we obtained round robin method has the best success reply that is 99.905 percent. While in different server spesification condition, we obtained the best percentage of success is on the IP Hash metho that is 99.95 percent.

Key words: Web Server, Load Balancing

Halaman ini sengaja dikosongkan

KATA PENGANTAR

Segala puja dan puji syukur kepada Allah SWT atas segala rahmat dan karunia yang telah dilimpahkan kepada penulis sehingga penulisan tesis dengan judul :

KAJIAN WEB LOAD BALANCING BERBASIS ROUND ROBIN DAN IP HASH UNTUK PENINGKATAN KINERJA LAYANAN SERVER

Dapat diselesaikan dengan baik. Buku tesis ini disusun untuk memenuhi salah satu syarat memperoleh gelar magister pada program studi teknik elektro dengan bidang keahlian Telematika, Institut Teknologi Sepuluh Nopember.

Pada kesempatan ini penulis ingin menyampaikan ucapan terima kasih sedalam-dalamnya kepada :

1. Kedua orang tua yang telah mendidik dan merawat penulis sampai bisa berada pada posisi ini.
2. Istri penulis yang selalu mendukung dan memberi semangat penulis dalam melakukan tugas yang ada.
3. Bapak Supeno Mardi Susiki Nugroho dan Bapak Istas Pratomo atas bimbingan serta kesabarannya terhadap penulis sehingga bisa menyelesaikan tesis ini.
4. Bapak dan Ibu dosen telematika yang telah memberi banyak pengetahuan baru bagi penulis selama masa perkuliahan.
5. Rekan-rekan S2 Teknik Telematika dan teman-teman di lab B301 atas bantuan serta kerjasamanya selama penulis menempuh studi pascasarjan.

Surabaya, Juli 2018

Penulis

Halaman ini sengaja dikosongkan

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

Halaman ini sengaja dikosongkan

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Saat ini penggunaan akses terhadap layanan yang ada di Internet sangatlah penting. Salah satu layanan Internet yang sering digunakan adalah layanan sistem informasi. Jika layanan tersebut tidak dapat diakses, maka orang yang menggunakan layanan tersebut tidak akan dapat mengetahui informasi terbaru yang disediakan oleh sistem informasi tersebut. Layanan sistem informasi tersebut pasti membutuhkan server untuk menjalankan aplikasi sistem informasi. Bila mana server yang digunakan tersebut mati maka otomatis layanan yang disediakan juga akan berhenti. Sehingga sekecil apapun waktu layanan yang disediakan berhenti maka akan berdampak terhadap orang yang membutuhkannya. Penyebab layanan tersebut berhenti bisa disebabkan karena server mengalami beban yang berlebih sehingga menyebabkan sumber daya dari server tersebut penuh.

Salah satu cara untuk mengatasi sumber daya yang habis yaitu dengan menambah sumber daya lain untuk memproses layanan yang disediakan. Cara menambahnya sumber daya tersebut bisa dengan meng-*upgrade* server maupun menambahkan server. Salah satu cara terbaik untuk mengatasinya adalah dengan menambahkan server membagi beban ke beberapa server tersebut. Salah satu keunggulannya yaitu jika satu server mati maka layanan masih tetap berjalan. Penyeimbang beban pada layanan web biasanya disebut dengan HTTP *Load Balancer*. Penyeimbang beban pada web adalah suatu aplikasi yang berfungsi untuk membagi beban berupa pada permintaan HTTP dari klien ke beberapa server di bawahnya sehingga permintaan tersebut dapat dikerjakan bersama-sama.

Pada penelitian ini akan dibuat manajemen sistem untuk menyeimbangkan beban web server ke beberapa server sehingga didapatkan layanan sistem informasi yang ketersediaannya tinggi tanpa harus membuat server mempunyai beban berlebih.

1.2 Rumusan Masalah

Untuk memperbesar performansi dari sebuah layanan server web maka server harus meningkatkan kesuksesan pemberian jawaban dari permintaan HTTP yang diberikan. Jika server sedang mengalami masalah memberikan jawaban dari permintaan tersebut atau dalam proses perbaikan, beban server web harus dipindahkan ke tempat lain atau yang disebut juga penyeimbang beban HTTP. Dalam penyeimbang beban HTTP, beban yang ditanggung oleh server akan dibagi secara merata sesuai dengan metode pembagi beban yang digunakan.

1.3 Tujuan

Tujuan dari tesis ini adalah didapatkannya peningkatan performansi suatu server yang mempunyai layanan informasi berbasis web. Dengan menyeimbangkan beban ke beberapa server sehingga layanan informasi tersebut masih dapat berjalan dengan beban berlebih. Selain itu didapatkan metode penyeimbang beban terbaik dengan kondisi tertentu.

1.4 Batasan Masalah

1. Penyeimbang beban yang digunakan adalah dengan menggunakan perangkat lunak.
2. Sistem operasi yang digunakan adalah sistem operasi yang berbasis pada sumber terbuka.
3. Aplikasi penyedia layanan web adalah perangkat lunak HTTP Server.
4. Metode penyeimbang beban yang digunakan adalah *round-robin, ip hash*, *least connection*, dan *weight*.

1.5 Kontribusi

Kontribusi yang diharapkan dari hasil penelitian ini adalah didapatkannya manajemen sistem yang dapat menyeimbangkan beban web dari beberapa server sehingga layanan yang diberikan server tersebut masih dapat berjalan dengan beban yang berlebih.

1.6 Metodologi Penelitian

Metodologi penelitian yang dipakai untuk penelitian ini adalah:

1. Studi Literatur, yaitu dengan membaca dan memahami teori-teori yang berkaitan dengan penelitian yang dilakukan yang terdiri dari buku-buku referensi dan jurnal.
2. Perancangan sistem, yaitu dengan cara melakukan perancangan untuk membangun server web dengan menggunakan pembagi beban dalam jaringan lokal.
3. Implementasi, yaitu penerapan dari hasil rancangan desain yang sudah dibuat.
4. Pengujian dan analisis, yaitu melakukan pengujian berupa klien yang melakukan permintaan HTTP ke web server melalui pembagi beban dengan beberapa motodenya. Setelah dilakukan pengujian maka didapatkan hasil untuk dianalisis.

Halaman ini sengaja dikosongkan

BAB 2
KAJIAN PUSTAKA

2.1 Jaringan Komputer

2.1.1 Server

Komputer server adalah sebuah komputer yang bertugas untuk melayani permintaan komputer klien dengan menyediakan berbagai sumber daya seperti memori yang lebih besar, hardisk dengan kapasitas tinggi, printer yang bisa digunakan bersama dan lain-lainnya.

Beberapa fungsi dari komputer server antara lain:

- Bertugas melayani permintaan komputer klien.
- Menyediakan sumber daya untuk digunakan bersama, baik berupa perangkat keras ataupun berupa aplikasi agar dapat di gunakan di semua komputer klien di dalam jaringan.
- Mengatur lalu lintas data.
- Menyimpan file atau data untuk di akses bersama menggunakan file sharing.
- Mengatur level hak akses di dalam jaringan, sehingga tidak semua klien bisa membuka data yang tersimpan di komputer server.
- Menyediakan basis data atau aplikasi yang dapat di jalankan di semua komputer.
- Melindungi komputer klien dengan memasang firewall atau anti malware di komputer server.

Jenis komputer server ada bermacam-macam, misalnya server lokal server yang di gunakan khusus di jaringan lokal, atau server hosting server biasanya kita gunakan di internet.

Jenis-jenis server berdasarkan kegunaan antara lain:

1. Web Server adalah sebuah perangkat lunak yang dipasang pada server yang berfungsi untuk menyediakan layanan permintaan data dengan protocol https atau http yang dapat diakses dengan menggunakan browser. Cara

kerjanya secara sederhana adalah web server akan merespon permintaan yang ada dengan mengirimkan konten tersebut kembali dalam bentuk gambar, tulisan atau bentuk lainnya. Kemudian akan ditampilkan pada browser.

2. Fax Server. Sesuai dengan namanya, server ini digunakan untuk melayani kebutuhan Fax bagi client. Fax server ini akan membuat semua sistem penerimaan dan pengiriman fax akan melaluinya. Selain itu, biasanya sebuah fax server telah dilengkapi dengan modem untuk mendukung fax server ini.
3. FTP Server adalah Pengertian dan Macam-Macam Jenis Server di Dunia Komputer yang ketiga. Server ini memiliki protocol FTP yang dapat dilakukan sebagai protocol untuk transfer data.
4. Mail Server. Sesuai dengan namanya, mail server ini memiliki fungsi untuk melayani client khususnya dalam hal berkirim surat. Surat surat yang ada pun akan disimpan di dalam server mail tersebut pula. Selain menyediakan layanan untuk berkirim surat, mail server ini juga menyediakan layanan pelengkap lainnya seperti web interface dimana layanan ini dapat memudahkan client untuk mengorganisir atau menulis surat yang dimiliki oleh client.
5. File server dapat diartikan sebagai sebuah komputer yang berfungsi untuk menampun sejumlah data yng dimiliki oleh client yang bersangkutan. Biasanya, kapasitas yang dimiliki oleh server ini juga bergantung pada HDD yang ada pada server tersebut.
6. Game Server server merupakan server yang digunakan untuk pusat untuk menghubungkan antar pemain (client) dengan pemain yang lainnya. Game server ini selain merupakan server tersendiri, bisa juga didirikan dari komputer client yang bermain game tersebut.
7. DNS Server, Server ini memiliki fungsi untuk menerjemahkan informasi nama host atau domain menjadi sebuah alamat IP.
8. Proxy Server merupakan sebuah server yang dapat berfungsi sebagai komputer lainnya untuk melakukan permintaan untuk content dari sebuah intranet atau internet.

9. Database Server melayani klien yang membutuhkan sebuah layanan untuk menyimpan database. port yang digunakan untuk Database Server biasanya 3306 (Mysql) dan 5432 (PgSQL).
10. Print Server merupakan sebuah pusat layanan untuk kegiatan percetakan atau print untuk client. Print Server Client ini merupakan salah satu Pengertian dan Macam-Macam Jenis Server di Dunia Komputer yang ada.
11. Server aplikasi bertugas menjalankan aplikasi tertentu yang menyediakan sumber dayanya untuk dapat diproses komputer lainnya di jaringan. Yang termasuk dalam kategori jenis server ini adalah server-server yang menjalankan perangkat lunak yang dibuat khusus untuk program-program tertentu seperti program yang berhubungan dengan fungsi accounting, penjualan, dan lain sebagainya.
12. Streaming Media Server melayani servis streaming data media seperti musik, video. Servis ini memungkinkan kita dapat mengakses sebuah konten tanpa kita mendownloadnya terlebih dahulu.

Sedangkan jenis komputer server berdasarkan bentukya adalah tower server, rak server, dan blade server. Tower Server kerap dijadikan sebuah pilihan umum bagi perusahaan. Dikarenakan harganya yang lebih murah dan setara dengan PC, bentuk fisiknya juga terbilang cukup efisien dan dapat diletakkan di berbagai ruangan sempit. server jenis ini dapat melakukan upgrade secara perlahan mengikuti perkembangan. Bentuk jenis server tersebut dapat dilihat pada Gambar 2.1.

Gambar 2.1 Komputer server dengan jenis tower server[15]

Rak Server memungkinkan beberapa server ditumpuk menjadi satu di dalam rak seperti pada Gambar 2.2. Server jenis ini memungkin untuk lebih menghemat ruangan tanpa mengurangi performa dan kinerja dari server tersebut. Selain itu, penggunaan rak server juga dapat memadukan atau menggabungkan beberapa server, sehingga lebih fleksibel. Keuntungan lain dari rak server ini adalah dalam penggunaan jangka panjang dapat dijadikan sebuah data centre.

Gambar 2.2 Komputer server dengan jenis rak server[16]

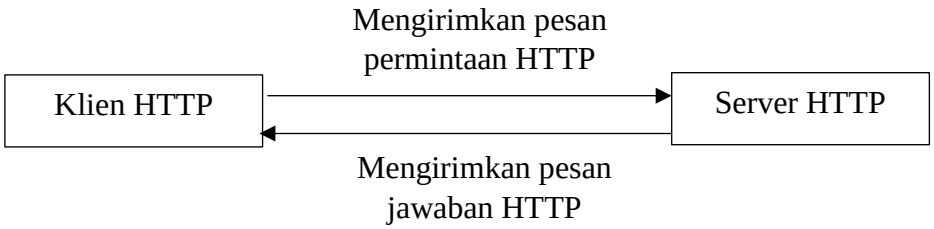
Blade server juga dikenal dengan istilah Ultrathin Server karena memiliki bentuk yang kecil namun dapat memuat data yang besar. Umumnya, Blade server ini diletakkan secara vertikal pada sebuah komponen perangkat keras seperti power supply seperti yang terlihat pada Gambar 2.3. Salah satu yang menjadi alasan utama perusahaan memilih Blade server ini adalah karena lebih menghemat ruang, menghemat tenaga serta daya, dan menghemat waktu.

Gambar 2.3 Komputer server dengan jenis blade server[17]

2.1.1.1 HTTP Server

Hyper Text Transfer Protocol (HTTP) adalah suatu protokol aplikasi jaringan yang berfungsi untuk mengkolaborasikan, mendistribusikan, dan menampilkan tulisan atau media lain di sistem informasi. Hypertext adalah teks yang terstruktur yang menggunakan tautan secara logika atau yang biasa disebut dengan hyperlinks.

Fungsi HTTP server sebagai jawaban dari permintaan di dalam model komputer klien server biasanya juga disebut web server. Alurnya dapat dilihat sperti pada Gambar 2.4. Web Server adalah suatu perangkat lunak yang berfungsi untuk memberikan jawaban dari permintaan HTTP (HyperText Transfer Protocol) atau HTTPS (HyperText Transfer Protocol Secure). Biasanya perangkat lunak yang permintaan HTTP tersebut dikenal peramban atau agen pengguna. Jenis jenis peramban yaitu Mozilla Firefox, Google Chrome, Opera, Safari, dan yang lainnya. Beberapa Contoh perangkat lunak webserver adalah Apache HTTP Server, Lighttpd, Nginx, Microsoft IIS.Cara kerja web server adalah ketika melakukan permintaan HTTP maka server akan menyediakan sumber daya seperti berkas HTML.



Gambar 2.4 Alur Permintaan dan Jawaban dari Protokol HTTP

Apache HTTP Server adalah sebuah proyek pengembangan perangkat lunak yang dikerjakan bersama sama. Tujuan dari pengembangan tersebut adalah untuk implementasi web (HTTP) yang handal, berkerja sekelas software komersil, banyak fitur, dan sumber kode yang mudah didapatkan[9].

NGINX adalah perangkat lunak sumber terbuka yang berfungsi untuk melayani web, reverse proxy, *caching*, penyeimbangan beban, streaming media, dan banyak lagi. Nginx pertama kali dibuat sebagai server web yang dirancang

untuk kinerja dan stabilitas maksimal. Selain kemampuan server HTTP-nya, NGINX juga dapat berfungsi sebagai server proxy untuk email (IMAP, POP3, dan SMTP) dan proxy reverse dan load balancer untuk server HTTP, TCP, dan UDP[11].

Pencipta Nginx yaitu Igor Sysoev awalnya menulis NGINX untuk memecahkan masalah C10K, istilah yang diciptakan pada tahun 1999 untuk menggambarkan kesulitan yang dialami server web yang ada dalam menangani koneksi bersamaan (C) yang berangka besar (10K). Dengan arsitektur *asynchronous* yang digerakkan oleh *event*, nginx merevolusi bagaimana server beroperasi dalam konteks kinerja tinggi dan menjadi server web tercepat yang tersedia. Tujuan di balik pembuatan nginx adalah untuk menjadi server web tercepat yang ada, dan mempertahankan keunggulannya masih merupakan tujuan utama proyek pembuatannya. Nginx secara konsisten mengalahkan Apache dan web server lain dalam tolok ukur yang mengukur kinerja server web. Sejak rilis pertama kali, situs web telah berkembang dari halaman HTML sederhana menjadi konten dinamis dan multifaset. NGINX telah tumbuh berbarengan dan sekarang mendukung semua komponen Web modern, termasuk WebSocket, HTTP/2, dan *streaming* berbagai format video (HDS, HLS, RTMP, dan lainnya).

Meskipun NGINX terkenal sebagai server web server tercepat, arsitektur yang digunakan telah terbukti secara ideal untuk menangani beban web di luar melayani konten. Karena ini dapat menangani koneksi dengan volume tinggi, NGINX umumnya digunakan sebagai proxy terbalik dan penyeimbang beban untuk mengelola lalu lintas masuk dan mendistribusikannya ke server hulu yang lebih lambat - apa pun dari server basis data lawas hingga layanan kecil. NGINX juga sering ditempatkan di antara klien dan server web kedua, untuk melayani sebagai terminator SSL/TLS atau akselerator web. Nginx yang bertindak sebagai perantara secara efisien menangani tugas yang mungkin memperlambat server web, seperti menegosiasikan SSL/TLS atau mengompresi dan konten tembolok untuk meningkatkan kinerja. Situs dinamis, dibangun menggunakan apa pun dari Node.js ke PHP, biasanya menggunakan NGINX sebagai tembolok konten dan reverse proxy untuk mengurangi beban pada server aplikasi dan memanfaatkan penggunaan perangkat keras yang paling efektif secara efektif.

Lightttpd adalah sebuah aplikasi web server yang efisien dan dioptimalkan untuk lingkungan berkinerja tinggi. Dengan penggunaan memori yang lebih kecil dibandingkan dengan web-server lain, manajemen CPU yang lebih efektif dan mempunyai beberapa fitur lanjutan (FastCGI, SCGI, Auth, Output-Compression, URL-Rewriting dan banyak lagi). lighttpd adalah solusi sempurna untuk setiap server yang mengalami masalah beban pada server. Lighttpd juga bersumber terbuka di bawah lisensi BSD yang direvisi. Lighttpd mendukung antarmuka FastCGI, SCGI dan CGI ke program eksternal, yang memungkinkan aplikasi web yang ditulis dalam bahasa pemrograman apapun dapat digunakan dengan didalam server. PHP sebagai bahasa pemrograman yang sangat populer mendapat perhatian khusus. FastCGI Lighttpd dapat dikonfigurasi untuk mendukung PHP dengan tembolok opcode (seperti APC) dengan baik dan efisien. Selain itu, lighttpd telah menerima perhatian dari komunitas Python, Perl, Ruby, dan Lua. Lighttpd juga mendukung WebDNA, sistem database in-memory yang tangguh yang dirancang untuk membangun situs web berbasis database. Ini adalah server web populer untuk kerangka kerja web Catalyst dan Ruby on Rails. Lighttpd tidak mendukung ISAPI.

Internet Information Services (IIS, sebelumnya *Internet Information Server*) adalah perangkat lunak server web yang dikembangkan oleh Microsoft untuk digunakan dengan keluarga Windows NT. IIS mendukung HTTP, HTTP/2, HTTPS, FTP, FTPS, SMTP dan NNTP. Sejarah terbentuknya IIS adalah saat Server web Microsoft yang pertama adalah proyek penelitian di Pusat Microsoft Windows NT Akademik Eropa (EMWAC), bagian dari Universitas Edinburgh di Skotlandia, dan didistribusikan sebagai perangkat lunak gratis. Karena server EMWAC tidak dapat menangani besarnya volume lalu lintas ke server Microsoft.com, maka Microsoft terpaksa mengembangkan server webnya sendiri yang selanjutnya bernama IIS. Hampir setiap versi dari IIS muncul berbarengan ketika Microsoft juga merilis Microsoft Windows. IIS versi 6.0 dan lebih tinggi mendukung mekanisme beberapa otentikasi yaitu Otentikasi anonim, Otentikasi akses dasar, Otentikasi akses Digest, Autentikasi Windows Terpadu, Otentikasi UNC,.NET Paspor Otentikasi (Dihapus di Windows Server 2008 dan IIS 7.0) dan Otentikasi sertifikat. IIS 7.0 memiliki arsitektur modular. Modul ini juga disebut dengan ekstensi, yang dapat ditambahkan atau dihapus secara individual sehingga hanya

modul yang diperlukan untuk fungsi tertentu yang harus terpasang. Modul tersebut antara lain berupa modul keamanan, modul konten, modul kompresi, modul tembolok, modul log, dan modul diagnostik. Sedangkan pada versi 7.5 mempunyai tambahan keamanan yang berupa pemetaan sertifikat klien, keamanan IP, penyaringan permintaan, dan otorisasi URL. Pada IIS versi 8.0 menawarkan performansi dan administrasi yang lebih mudah. Sedangkan pada IIS 8.5 menawarkan performansi yang lebih baik pada skenario yang lebih besar.

Menurut IANA, Kode status dalam HTTP yaitu:

- 1xx: Informasi - Request received, continuing process
- 2xx: Sukses - The action was successfully received, understood, and accepted
- 3xx: Pengalihan - Further action must be taken in order to complete the request
- 4xx: Galat klien - The request contains bad syntax or cannot be fulfilled
- 5xx: Galat Server - The server failed to fulfill an apparently valid request

Tabel 2.1 Kode status dalam HTTP

Nilai	Deskripsi	Referensi
100	Continue	[RFC7231, Section 6.2.1]
101	Switching Protocols	[RFC7231, Section 6.2.2]
102	Processing	[RFC2518]
103	Early Hints	[RFC8297]
104-199	Unassigned	
200	OK	[RFC7231, Section 6.3.1]
201	Created	[RFC7231, Section 6.3.2]
202	Accepted	[RFC7231, Section 6.3.3]
203	Non-Authoritative Information	[RFC7231, Section 6.3.4]
204	No Content	[RFC7231, Section 6.3.5]
205	Reset Content	[RFC7231, Section 6.3.6]
206	Partial Content	[RFC7233, Section 4.1]
207	Multi-Status	[RFC4918]
208	Already Reported	[RFC5842]
209-225	Unassigned	
226	IM Used	[RFC3229]
227-299	Unassigned	
300	Multiple Choices	[RFC7231, Section 6.4.1]
301	Moved Permanently	[RFC7231, Section 6.4.2]
302	Found	[RFC7231, Section 6.4.3]

Tabel 2.1 Kode Status dalam HTTP (Lanjutan)

Nilai	Deskripsi	Referensi
303	See Other	[RFC7231, Section 6.4.4]
304	Not Modified	[RFC7232, Section 4.1]
305	Use Proxy	[RFC7231, Section 6.4.5]
306	(Unused)	[RFC7231, Section 6.4.6]
307	Temporary Redirect	[RFC7231, Section 6.4.7]
308	Permanent Redirect	[RFC7538]
309-399	Unassigned	
400	Bad Request	[RFC7231, Section 6.5.1]
401	Unauthorized	[RFC7235, Section 3.1]
402	Payment Required	[RFC7231, Section 6.5.2]
403	Forbidden	[RFC7231, Section 6.5.3]
404	Not Found	[RFC7231, Section 6.5.4]
405	Method Not Allowed	[RFC7231, Section 6.5.5]
406	Not Acceptable	[RFC7231, Section 6.5.6]
407	Proxy Authentication Required	[RFC7235, Section 3.2]
408	Request Timeout	[RFC7231, Section 6.5.7]
409	Conflict	[RFC7231, Section 6.5.8]
410	Gone	[RFC7231, Section 6.5.9]
411	Length Required	[RFC7231, Section 6.5.10]
412	Precondition Failed	[RFC7232, Section 4.2][RFC8144, Section 3.2]
413	Payload Too Large	[RFC7231, Section 6.5.11]
414	URI Too Long	[RFC7231, Section 6.5.12]
415	Unsupported Media Type	[RFC7231, Section 6.5.13][RFC7694, Section 3]
416	Range Not Satisfiable	[RFC7233, Section 4.4]
417	Expectation Failed	[RFC7231, Section 6.5.14]
418-420	Unassigned	
421	Misdirected Request	[RFC7540, Section 9.1.2]
422	Unprocessable Entity	[RFC4918]
423	Locked	[RFC4918]
424	Failed Dependency	[RFC4918]
425	Unassigned	
426	Upgrade Required	[RFC7231, Section 6.5.15]
427	Unassigned	
428	Precondition Required	[RFC6585]
429	Too Many Requests	[RFC6585]
430	Unassigned	
431	Request Header Fields Too Large	[RFC6585]
432-450	Unassigned	
451	Unavailable For Legal Reasons	[RFC7725]

Tabel 2.1 Kode Status dalam HTTP (Lanjutan)

Nilai	Deskripsi	Referensi
452-499	Unassigned	
500	Internal Server Error	[RFC7231, Section 6.6.1]
501	Not Implemented	[RFC7231, Section 6.6.2]
502	Bad Gateway	[RFC7231, Section 6.6.3]
503	Service Unavailable	[RFC7231, Section 6.6.4]
504	Gateway Timeout	[RFC7231, Section 6.6.5]
505	HTTP Version Not Supported	[RFC7231, Section 6.6.6]
506	Variant Also Negotiates	[RFC2295]
507	Insufficient Storage	[RFC4918]
508	Loop Detected	[RFC5842]
509	Unassigned	
510	Not Extended	[RFC2774]
511	Network Authentication Required	[RFC6585]
512-599	Unassigned	

2.1.1.2 DNS Server

Server DNS adalah server komputer yang berisi basis data alamat IP dan hostname terkait, dan dalam banyak kasus, berfungsi untuk menyelesaikan, atau menerjemahkan, nama-nama umum ke alamat IP seperti yang diminta. Server DNS menjalankan perangkat lunak khusus dan berkomunikasi satu sama lain menggunakan protokol khusus. Untuk mudah memahaminya, server DNS di internet adalah perangkat yang menerjemahkan www.its.ac.id yang diketik di browser ke 103.94.189.5 alamat IP yang sebenarnya.

Tujuan adanya DNS server adalah karena manusia lebih mudah mengingat nama daripada mengingat angka. Ketika kita ingin membuka web www.its.ac.id yang perlu kita ingat adalah www.its.ac.id, tidak perlu mengingat alamat IP-nya. Begitu pula jika kita ingin membuka situs web lain, misalnya google.co.id, integra.its.ac.id, ataupun yang lainnya.

DNS secara umum menggunakan User Datagram Protocol (UDP) pada nomor port 53 untuk melayani permintaan. Permintaan DNS terdiri dari permintaan UDP tunggal dari klien diikuti oleh balasan UDP tunggal dari server. Ketika

panjang jawaban melebihi 512 byte dan kedua klien dan dukungan server EDNS, paket UDP yang lebih besar digunakan. Jika tidak, permintaan dikirim lagi menggunakan Transmission Control Protocol (TCP). TCP juga digunakan untuk tugas-tugas seperti transfer zona. Beberapa implementasi resolver menggunakan TCP untuk semua pertanyaan.

Sistem Nama Domain menentukan satu set berbagai jenis catatan sumber daya (RR), yang merupakan elemen informasi dasar dari sistem nama domain. Setiap catatan memiliki tipe (nama dan nomor), waktu kedaluwarsa, kelas, dan data jenis-spesifik. Catatan sumber daya dari jenis yang sama dijelaskan sebagai satu set catatan sumber daya (RRset) seperti pada tabel 2.2. Urutan catatan sumber daya dalam satu set, yang dikembalikan oleh resolver ke aplikasi, tidak terdefinisi, tetapi sering server menerapkan perintah round-robin untuk mencapai load balancing.

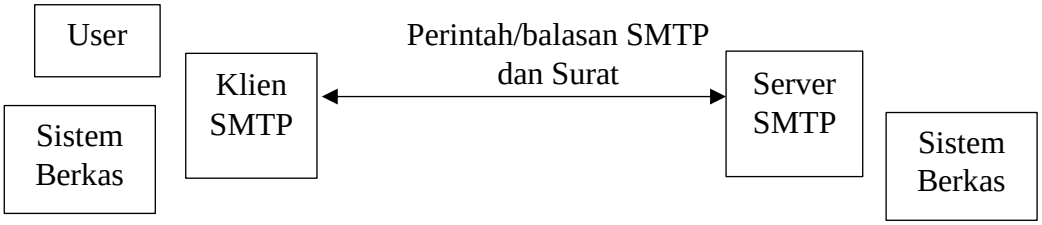
Tabel 2.2 Tipe RR pada DNS server

Tipe	value	kegunaan
A	1	Alamat Host
NS	2	Nama server yang berwenang
MD	3	Tujuan Email (Obsolete - gunakan MX)
MF	4	Forwarder email (Obsolete - gunakan MX)
CNAME	5	alias
SOA	6	Tanda mulai zona yang berwenang
MB	7	Nama domain mailbox (EXPERIMENTAL)
MG	8	Anggota grup email (EXPERIMENTAL)
MR	9	Nama domain mail rename (EXPERIMENTAL)
NULL	10	null RR (EXPERIMENTAL)
WKS	11	Deskripsi layanan yang diketahui.
PTR	12	Pointer nama domain
HINFO	13	Informasi ost
MINFO	14	Informasi mailbox atau daftar mail
MX	15	Penukaran email
TXT	16	text strings

Ketika dikirim melalui jaringan Protokol Internet, semua catatan menggunakan format umum yang ditentukan dalam RFC 1035.

2.1.1.3 Mail Server

Mail server atau Simple Mail Transfer Protocol (SMTP) adalah sebuah Internet standard untuk pengiriman surat elektronik (surel).



Gambar 2.5 Alur Permintaan dan Jawaban dari Protokol SMTP

Pada gambar 2.5 ketika klien SMTP memiliki pesan untuk mengirim, itu menetapkan saluran transmisi dua arah ke server SMTP. Tanggung jawab klien SMTP adalah mentransfer pesan email ke satu atau lebih server SMTP, atau melaporkan kegagalannya untuk melakukannya. Klien SMTP yang mentransfer semua lalu lintas terlepas dari domain target yang terkait dengan pesan individu, atau yang tidak mempertahankan antrean untuk mencoba transmisi pesan yang awalnya tidak dapat diselesaikan, mungkin jika tidak sesuai dengan spesifikasi ini tetapi tidak dianggap berkemampuan penuh. Implementasi SMTP sepenuhnya-mampu, termasuk relay yang digunakan oleh yang kurang mampu ini, dan tujuan mereka, diharapkan untuk mendukung semua fungsi-fungsi alamat antrian, mencoba kembali, dan alternatif yang dibahas dalam spesifikasi ini. Dalam banyak situasi dan konfigurasi, klien yang kurang mampu yang dibahas di atas harus menggunakan protokol submission daripada SMTP.

2.1.2 Network Switch

Network Switch jaringan atau yang disebut juga switching hub, bridging hub adalah perangkat jaringan komputer yang menghubungkan perangkat secara bersamaan di jaringan komputer dengan menggunakan packet switching untuk

menerima, memproses, dan meneruskan data ke perangkat tujuan. Slah satu bentuk dari Network Switch dapat dilihat seperti pada Gambar 2.6.

Gambar 2.6 Bentuk Switch Cisco 3560G[18]

Network Switch adalah jembatan jaringan multiport yang menggunakan alamat perangkat keras untuk memproses dan meneruskan data pada lapisan data link (layer 2) dari model OSI. Beberapa switch juga dapat mengolah data pada layer jaringan (layer 3) dengan tambahan fungsionalitas routing. Switch tersebut umumnya dikenal sebagai switch layer-3 atau switch multilayer.

Switch untuk Ethernet adalah bentuk switch jaringan yang paling umum. Switch Ethernet pertama diperkenalkan oleh Kalpana pada tahun 1990. Switch juga terdapat jenis jaringan lain misalnya Fibre Channel, Asynchronous Transfer Mode, dan InfiniBand.

Jenis switch berdasarkan konfigurasi, dibagi menjadi dua, yaittu unmanaged switch dan managed switch.

Unmanaged switch adalah switch yang tidak memiliki antarmuka konfigurasi atau pilihan. Mereka adalah plug and play. Mereka biasanya merupakan switch yang paling murah, dan oleh karena itu sering digunakan di lingkungan kantor / kantor kecil. Sakelar yang tidak terkelola bisa dipasang di desktop atau rak.

Sedangkan managed switch adalah switch yang memiliki satu atau lebih metode untuk memodifikasi pengoperasian sakelar. Metode manajemen umum meliputi: antarmuka baris perintah (CLI) yang diakses melalui konsol serial, telnet atau Secure Shell, agen Simple Network Management Protocol (SNMP) yang tertanam yang memungkinkan pengelolaan dari konsol jarak jauh atau stasiun manajemen, atau antarmuka web untuk manajemen dari sebuah browser web Contoh perubahan konfigurasi yang dapat dilakukan seseorang dari switch yang dikelola meliputi: mengaktifkan fitur seperti Spanning Tree Protocol atau port

mirroring, menetapkan bandwidth port, membuat atau memodifikasi LAN virtual (VLAN), dll. Dua subkelas switch terkelola dipasarkan hari ini. :

Switch cerdas (atau cerdas) - ini dikelola dengan serangkaian fitur manajemen yang terbatas. Demikian juga switch "web-managed" adalah switch yang jatuh ke dalam ceruk pasar antara tidak terkelola dan dikelola. Untuk harga yang jauh lebih rendah daripada switch yang dikelola sepenuhnya, mereka menyediakan antarmuka web (dan biasanya tidak ada akses CLI) dan memungkinkan konfigurasi pengaturan dasar, seperti VLAN, bandwidth port dan dupleks.

Switch yang dikelola oleh perusahaan (atau dikelola sepenuhnya) - ini memiliki serangkaian fitur manajemen lengkap, termasuk CLI, agen SNMP, dan antarmuka web. Mereka mungkin memiliki fitur tambahan untuk memanipulasi konfigurasi, seperti kemampuan untuk menampilkan, memodifikasi, membuat cadangan dan mengembalikan konfigurasi. Dibandingkan dengan smart switch, switch enterprise memiliki lebih banyak fitur yang bisa disesuaikan atau dioptimalkan, dan umumnya lebih mahal daripada smart switch. Switch perusahaan biasanya ditemukan di jaringan dengan jumlah switch dan koneksi yang lebih banyak, di mana manajemen terpusat adalah penghematan yang signifikan dalam waktu dan usaha administratif. Switch yang dapat ditumpuk adalah versi switch yang dikelola perusahaan.

Jenis switch berdasarkan layer adalah sebagai berikut:

- Switch Layer 1
- Switch Layer 2
- Switch Layer 3

Layer 1 Hub jaringan, atau pengulang, adalah perangkat jaringan sederhana yang tidak mengelola lalu lintas yang melewatinya. Setiap paket yang masuk ke port dibanjiri atau "diulang" pada setiap port lainnya, kecuali port masuk. Secara khusus, setiap bit atau simbol diulang saat mengalir (dengan penundaan minimum untuk antarmuka garis). Karena ini, hub pengulang hanya bisa menerima dan meneruskan dengan kecepatan tunggal. Karena setiap paket diulang pada setiap port lainnya, tabrakan paket mempengaruhi keseluruhan jaringan, sehingga

membatasi kapasitas keseluruhannya. Ada aplikasi khusus di mana hub jaringan dapat berguna, seperti menyalin lalu lintas ke beberapa sensor jaringan. Switch jaringan high-end biasanya memiliki fitur yang disebut port mirroring yang menyediakan fungsionalitas yang sama. Sebuah switch jaringan menciptakan lapisan 1 koneksi end-to-end hanya secara virtual, padahal awalnya itu wajib. Fungsi bridging switch menggunakan informasi yang diambil dari layer 2 untuk memilih setiap paket port tertentu yang harus diteruskan, menghilangkan persyaratan bahwa setiap node diberi semua lalu lintas. Akibatnya, jalur koneksi tidak "dialihkan" secara harfiah, malah hanya terlihat seperti pada tingkat paket.

Layer 2, Sebuah jembatan jaringan, yang beroperasi pada lapisan data link, dapat menghubungkan sejumlah kecil perangkat di rumah atau kantor. Ini adalah kasus remeh yang menjembatani, di mana jembatan mempelajari alamat MAC dari setiap perangkat yang terhubung. Jembatan juga menyangga paket masuk dan menyesuaikan kecepatan transmisi dengan port keluar. Jembatan klasik juga dapat saling berhubungan menggunakan protokol spanning tree yang menonaktifkan tautan sehingga jaringan area lokal yang dihasilkan adalah pohon tanpa loop. Berbeda dengan router, jembatan pohon rentang harus memiliki topologi dengan hanya satu jalur aktif antara dua titik. Protokol pohon yang lebih tua IEEE 802.1D bisa sangat lambat, dengan penerusan berhenti selama 30 detik sementara pohon spanning reconverged. Sebuah Rapid Spanning Tree Protocol diperkenalkan sebagai IEEE 802.1w. Standar terbaru Shortest path bridging (IEEE 802.1aq) adalah pengembangan logis berikutnya dan menggabungkan semua Protokol Spanning Tree yang lebih tua (IEEE 802.1D STP, IEEE 802.1w RSTP, IEEE 802.1s MSTP) yang memblokir lalu lintas di semua kecuali satu jalur alternatif. IEEE 802.1aq (Shortest Path Bridging SPB) memungkinkan semua jalur untuk aktif dengan banyak jalur biaya yang sama, memberikan topologi 2 lapisan yang jauh lebih besar (sampai 16 juta dibandingkan dengan batas VLAN 4096), konvergensi yang lebih cepat, dan memperbaiki penggunaannya. dari topologi mesh melalui peningkatan bandwidth dan redundansi antara semua perangkat dengan memungkinkan lalu lintas memuat saham di semua jalur jaringan mesh. Sementara lapisan 2 beralih tetap merupakan istilah pemasaran daripada istilah teknis, Produk yang diperkenalkan sebagai "switch" cenderung menggunakan mikrosegmentasi

dan full duplex untuk mencegah tabrakan antar perangkat yang terhubung ke Ethernet. Dengan menggunakan pesawat forwarding internal jauh lebih cepat daripada antarmuka apapun, mereka memberi kesan jalan simultan di antara beberapa perangkat. Perangkat 'Non-blocking' menggunakan pesawat forwarding atau metode setara cukup cepat untuk memungkinkan lalu lintas dupleks penuh untuk setiap port secara bersamaan.

Layer 3, dalam batas-batas lapisan fisik Ethernet, switch layer-3 dapat melakukan beberapa atau semua fungsi yang biasanya dilakukan oleh router. Kemampuan layer-3 yang paling umum adalah kesadaran multicast IP melalui pengintaian IGMP. Dengan kesadaran ini, switch layer-3 dapat meningkatkan efisiensi dengan memberikan lalu lintas grup multicast hanya ke port tempat perangkat yang terpasang memberi isyarat bahwa ia ingin mendengarkan grup tersebut.

2.2 Scalable Web Server

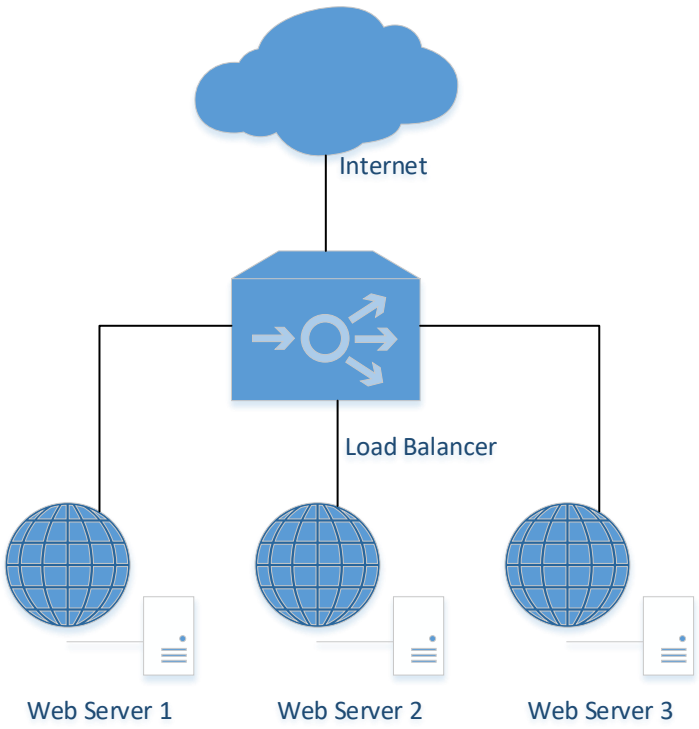
Pada penelitian yang berjudul A Comparison of Load Balancing Techniques for Scalable Web Servers oleh Espen Klovning and Oivind Kure bertujuan untuk membandingkan dan mengevaluasi algoritma load balancing yang berbeda untuk server Web yang dapat dikembangkan. Kontribusi utama dari penelitian mereka adalah untuk membandingkan beberapa solusi yang berbeda. Berbasis pada studi simulasi untuk membandingkan kinerja dan evaluasi kinerja dari implementasi prototipe. Ketika beban Transfer Hypertext Protocol (HTTP) meningkat, akan ada titik ketika server Web harus ditingkatkan atau beban dibagi ke beberapa server. Dengan menghubungkan banyak mesin yang lebih kecil. keunggulan harga dari mesin kelas desktop dan workstation dapat digunakan untuk membangun server yang dapat dikembangkan dengan biaya lebih murah. Kumpulan Server juga dapat menyediakan pembagian beban yang lebih baik, algoritma pembagian beban yang tepat dapat memfasilitasi ketahanan kesalahan dengan penurunan kinerja yang luwes saat mesin meninggalkan kluster karena kegagalan atau pemeliharaan preventif. Kumpulan Server juga memungkinkan untuk menambahkan mesin baru tanpa layanan interupsi. Hasil yang didapatkan dari penelitian tersebut adalah, penggunaan DNS server yang bergantian menghasilkan kinerja load balancing yang masuk akal, tetapi tembolok DNS untuk

nilai TTL yang paling umum (1 jam) menyebabkan beban yang miring pada kumpulan server dengan rata-rata 240 persen dari total beban. Hasil ini valid di server Web dengan tingkat beban yang berbeda. Elemen pemetaan ulang jaringan juga dapat meningkatkan load balancing di kumpulan server. Elemen jaringan ini dapat disediakan oleh vendor router yang dapat dengan mudah menggabungkan modifikasi algoritma routing yang diuraikan dan menggunakan salah satu algoritma load balancing yang diusulkan. mereka merekomendasikan algoritma load balancing round-robin untuk implikasi sederhana dan pembagian beban yang baik, dengan algoritma round-trip sebagai alternatif kedua. Jika algoritma RoundTrip ditingkatkan dengan prediksi yang lebih baik dari beban saat ini, algoritma ini memiliki potensi untuk mendistribusikan beban lebih baik daripada pendekatan round-robin, terutama dalam kondisi beban tinggi pada saat tertentu[3].

2.3 Teknik Load Balancing

2.3.1 HTTP Load Balancing

Load balancer atau penyeimbang beban adalah suatu aplikasi yang mendistribusikan permintaan klien yang masuk ke beberapa kumpulan server dan memberikan jawaban dari permintaan yang didapatkan ke klien. Topologi dari load balancer dapat dilihat seperti pada Gambar 2.7. Load balancer paling sering digunakan ketika server membutuhkan banyak server karena jumlah permintaan terlalu banyak untuk ditangani oleh satu server saja. Memakai penyeimbang beban ke beberapa server juga mengurangi jumlah kegagalan menjawab permintaan, sehingga membuat situs web lebih dapat diandalkan. Secara umum, aplikasi pada semua server host mempunyai konten yang sama, dan tugas load balancer adalah untuk mendistribusikan beban kerja dengan cara mengoptimalkan setiap kapasitas server, mencegah kelebihan beban pada server apa pun, dan hasil dalam respon tercepat yang mungkin untuk klien.



Gambar 2.7 Topologi Load Balancer

Load balancer juga dapat meningkatkan pengalaman pengguna dengan mengurangi jumlah respons kesalahan yang didapat oleh klien. Load balancer melakukan ini dengan mendeteksi ketika server mati maka akan mengalihkan permintaan ke server lain yang masih dalam grup. Dalam penerapan yang paling sederhana, penyeimbang beban mendeteksi kesehatan server dengan cara mencegat respons kesalahan terhadap permintaan biasa. Pemeriksaan kesehatan aplikasi adalah metode yang lebih fleksibel dan canggih di mana penyeimbang beban mengirim permintaan pemeriksaan kesehatan yang terpisah dan membutuhkan jenis respons tertentu untuk mempertimbangkan server yang baik.

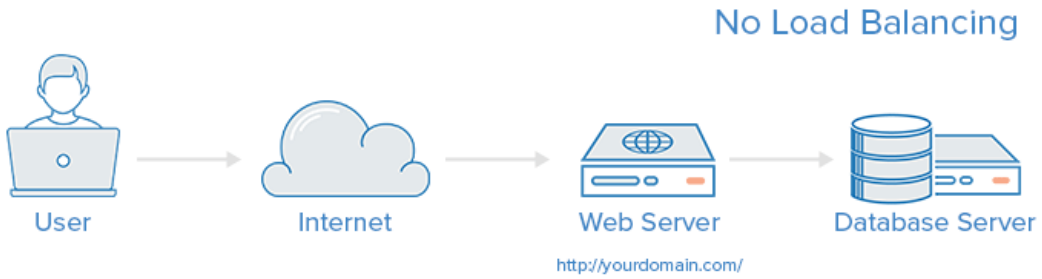
Fungsi lain yang berguna disediakan oleh beberapa load balancers adalah persistensi sesi, yang berarti mengirim semua permintaan dari klien tertentu ke server yang sama. Meskipun secara teori HTTP bersifat *stateless*, banyak aplikasi yang harus menyimpan informasi status hanya untuk menyediakan fungsi utama mereka. Aplikasi seperti underperform atau load even balancer mendistribusikan

permintaan dalam sesi pengguna ke server yang berbeda daripada mengarahkan semuanya ke server yang merespons permintaan awal.

Terdapat tiga tipe load balancing yaitu: [13]

1. Tanpa Load Balancing

Pengguna terhubung langsung ke server web Anda, di domainAnda.com dan tidak ada penyeimbangan beban seperti pada gambar 2.8. Jika server web tunggal Anda turun, pengguna tidak akan lagi dapat mengakses server web Anda. Selain itu, jika banyak pengguna mencoba mengakses server Anda secara bersamaan dan tidak dapat menangani beban, mereka mungkin memiliki pengalaman yang lambat atau mereka mungkin tidak dapat terhubung sama sekali.

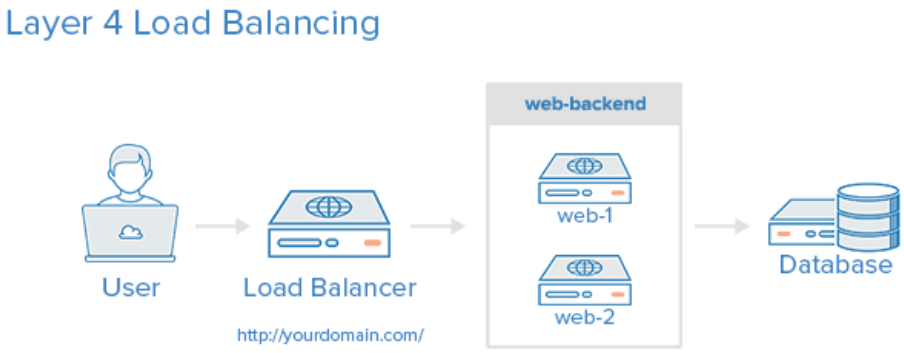


Gambar 2.8 Topologi Tanpa Load Balancer[13]

2. Layer 4 Load Balancing

Cara termudah untuk memuat traffic jaringan balance ke beberapa server adalah dengan menggunakan load balancing layer 4 (transport layer). Load balancing dengan cara ini akan meneruskan lalu lintas pengguna berdasarkan jangkauan IP dan port (yaitu jika permintaan masuk untuk http://integra.its.ac.id/apasaja, lalu lintas akan diteruskan ke backend yang menangani semua permintaan untuk integra.its.ac.id di port 80).

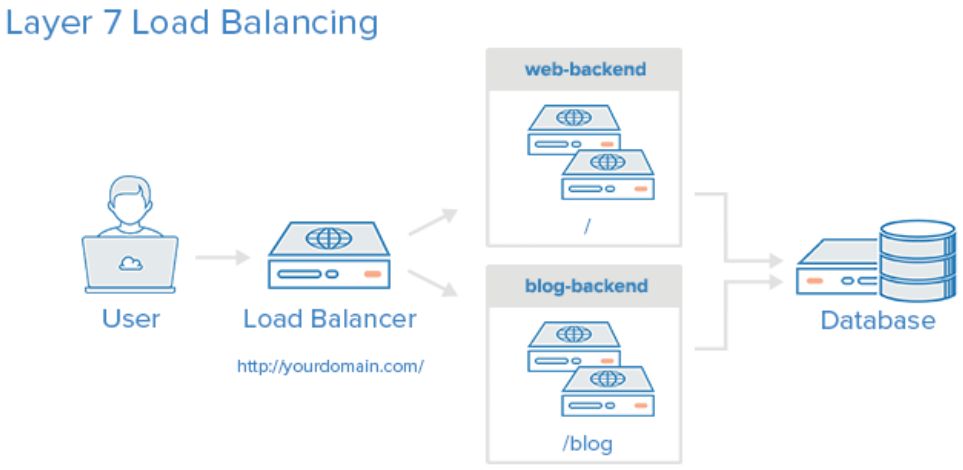
Pengguna mengakses load balancer, yang meneruskan permintaan pengguna ke grup backend web-backend server. Server backend mana saja yang dipilih akan merespon langsung permintaan pengguna seperti pada gambar 2.9. Secara umum, semua server di web-backend harus menyajikan konten yang identik - jika tidak, pengguna mungkin menerima konten yang tidak konsisten. Perhatikan bahwa kedua server web terhubung ke server database yang sama.



Gambar 2.9 Topologi Layer 4 Load Balancer[13]

3. Layer 7 Load Balancing

Layer 7 Load Balancing memungkinkan load balancer untuk meneruskan permintaan ke server backend yang berbeda berdasarkan konten permintaan pengguna. Mode load balancing ini memungkinkan Anda menjalankan beberapa server aplikasi web di bawah domain dan port yang sama.



Gambar 2.10 Topologi layer 7 Load Balancer[13]

Dilihat dari gambar 2.10 jika pengguna meminta yourdomain.com/blog, mereka diteruskan ke backend blog, yang merupakan sekumpulan server yang menjalankan aplikasi blog. Permintaan lain diteruskan ke web-backend, yang

mungkin menjalankan aplikasi lain. Kedua backend menggunakan server database yang sama.

Terdapat beberapa macam aplikasi penyeimbang beban yang banyak digunakan baik aplikasi secara perangkat keras maupun berupa perangkat lunak. Merek penyeimbang beban berupa perangkat lunak misalnya f5 tahu Barracuda Load Balancer. Sedangkan aplikasi penyeimbang beban berupa perangkat lunak yaitu:

2.3.1.1 HAProxy

HAProxy, yang merupakan singkatan dari High Availability Proxy, adalah perangkat lunak open source populer TCP / HTTP Load Balancer dan solusi proxy yang dapat dijalankan di Linux, Solaris, dan FreeBSD[10]. Penggunaannya yang paling umum adalah untuk meningkatkan kinerja dan keandalan lingkungan server dengan mendistribusikan beban kerja di beberapa server (misalnya web, aplikasi, database). Ini digunakan di banyak beberapa web server yang terkenal termasuk: GitHub, Imgur, Instagram, dan Twitter.

Di dalam HAProxy terdapat istilah-istilah yang dipakai dalam mengonfigurasi penyeimbang beban, misalnya ACL, backend, maupun frontend. ACL atau Access Control List di HAProxy digunakan untuk menguji beberapa kondisi dan melakukan tindakan (misalnya memilih server, atau memblokir permintaan) berdasarkan hasil pengujian. Penggunaan ACL memungkinkan pengaturan lalu lintas jaringan menjadi fleksibel berdasarkan berbagai faktor seperti pencocokan pola dan jumlah koneksi ke backend. Backend adalah sekumpulan server yang menerima permintaan dari permintaan yang diteruskan. Backend didefinisikan di bagian backend konfigurasi HAProxy. Backend dapat didefinisikan menjadi algoritma penyeimbang beban yang dipakai dan kumpulan server beserta port-nya. Frontend pada HAProxy adalah sekumpulan alamat IP dan port yang mendefinisikan bagaimana permintaan diteruskan ke backend.

Pada HAProxy Algoritma penyeimbang beban sangat banyak. Tetapi yang populer digunakan yaitu round robin, leastconn, dan source. Pada algoritma round robin permintaan akan dibagi secara bergantian ke backend. Algoritma ini merupakan algoritma asali pada HAProxy. Pada algoritma leastconn, frontend akan memilih server dengan jumlah koneksi paling sedikit. Algoritma ini

direkomendasikan untuk web yang mempunyai sesi yang lebih lama. Server di backend yang sama juga dipakai bergantian seperti pada mode round-robin. Pada Source, frontend memilih server mana yang akan digunakan berdasarkan hash dari alamat IP sumber yaitu alamat IP pengguna. Algoritma ini adalah salah satu metode untuk memastikan bahwa pengguna akan terhubung ke server yang sama.

Berikut adalah penjelasan algoritma load balancing yang didukung oleh HAProxy[19]:

1. Round-Robin

Setiap server digunakan bergantian, sesuai dengan bobotnya. Ini adalah algoritma paling halus dan paling adil ketika waktu pemrosesan server tetap merata. Algoritma ini bersifat dinamis, yang berarti bahwa bobot server dapat disesuaikan dengan cepat untuk memulai dengan lambat misalnya. Ini dibatasi oleh desain hingga 4095 server aktif per backend. Perhatikan bahwa di beberapa peternakan besar, ketika server menjadi naik setelah turun untuk waktu yang sangat singkat, kadang-kadang diperlukan beberapa ratus permintaan untuk itu untuk diintegrasikan kembali ke kelompok server dan mulai menerima lalu lintas. Ini normal, meskipun sangat jarang.

2. Static RR

Setiap server digunakan bergantian, sesuai dengan bobotnya. Algoritma ini sama dengan roundrobin kecuali bersifat statis, yang berarti bahwa mengubah berat server dengan cepat tidak akan berpengaruh. Di sisi lain, ia tidak memiliki batasan desain pada jumlah server, dan ketika server naik, itu selalu segera diperkenalkan kembali ke kelompok, setelah pemetaan selesai dikomputasi ulang. Algoritma ini menggunakan lebih sedikit CPU untuk dijalankan (sekitar -1%).

3. Leastconn

Server dengan jumlah koneksi terendah menerima koneksi. Round-robin dilakukan dalam kelompok server dengan beban yang sama untuk memastikan bahwa semua server akan digunakan. Penggunaan algoritma ini direkomendasikan di mana sesi yang sangat panjang diharapkan, seperti LDAP, SQL, TSE, dll ... tetapi tidak sangat cocok untuk protokol menggunakan sesi pendek seperti HTTP. Algoritma ini bersifat dinamis, yang berarti bahwa bobot server dapat disesuaikan dengan cepat untuk memulai dengan lambat misalnya.

4. First

Server pertama dengan slot koneksi yang tersedia menerima koneksi. Server dipilih dari nomor identitas terendah ke yang tertinggi (lihat parameter server "id"), yang asal ke posisi server di kelompok server. Setelah server mencapai nilai koneksi maksimalnya, maka server akan digunakan. Tidak mungkin untuk menggunakan algoritma ini tanpa pengaturan koneksi maksimal. Tujuan dari algoritma ini adalah selalu menggunakan jumlah server terkecil sehingga server tambahan dapat dimatikan selama waktu non-intensif. Algoritma ini mengabaikan berat server, dan membawa manfaat lebih banyak untuk sesi panjang seperti RDP atau IMAP daripada HTTP, meskipun dapat berguna di sana juga. Untuk menggunakan algoritma ini secara efisien, disarankan agar pengontrol cloud secara teratur memeriksa penggunaan server untuk mematikannya ketika tidak digunakan, dan secara teratur memeriksa antrean backend untuk mengaktifkan server baru ketika antrean mengembang. Alternatif lain yaitu menggunakan "http-check send-state" dapat menginformasikan server tentang beban.

5. Source

Alamat IP sumber di-hash dan dibagi dengan total berat server yang sedang berjalan untuk menunjuk server mana yang akan menerima permintaan tersebut. Ini memastikan bahwa alamat IP klien yang sama akan selalu mencapai server yang sama selama tidak ada server yang turun atau naik. Jika hasil hash berubah karena jumlah server yang berjalan berubah, banyak klien akan diarahkan ke server yang berbeda. Algoritma ini umumnya digunakan dalam mode TCP di mana tidak ada cookie yang dimasukkan. Ini juga dapat digunakan di Internet untuk memberikan kelekatan terbaik bagi klien yang menolak cookie sesi. Algoritma ini secara default berarti, yang berarti bahwa mengubah berat server dengan cepat tidak akan berpengaruh, tetapi ini dapat diubah menggunakan "hash-type".

6. Uri

Algoritma ini melakukan hash bagian kiri dari URI (sebelum tanda tanya) atau seluruh URI (jika parameter "keseluruhan" ada) dan membagi nilai hash dengan total berat server yang sedang berjalan. Hasilnya menunjukkan server mana yang akan menerima permintaan. Ini memastikan bahwa URI yang sama akan selalu diarahkan ke server yang sama selama tidak ada server yang naik atau turun.

Ini digunakan dengan cache proksi dan proxy anti-virus untuk memaksimalkan rasio klik cache. Perhatikan bahwa algoritma ini hanya dapat digunakan di backend HTTP. Algoritma ini secara default bersifat statis, yang berarti bahwa mengubah berat server dengan cepat tidak akan berpengaruh, tetapi ini dapat diubah menggunakan "hash-type".

Algoritma ini mendukung dua parameter opsional "len" dan "depth", keduanya diikuti oleh bilangan bulat positif. Opsi-opsi ini dapat membantu ketika diperlukan untuk menyeimbangkan server berdasarkan pada awal URI saja. Parameter "len" menunjukkan bahwa algoritma seharusnya hanya mempertimbangkan banyak karakter di awal URI untuk menghitung hash. Perhatikan bahwa memiliki "len" diatur ke 1 jarang masuk akal karena sebagian besar URI mulai dengan "/" didepan.

Parameter "depth" menunjukkan kedalaman direktori maksimum yang akan digunakan untuk menghitung hash. Satu level dihitung untuk setiap karakter slash dalam permintaan. Jika kedua parameter ditentukan, evaluasi akan berhenti ketika tercapai.

7. URL Param

Parameter URL yang ditentukan dalam argumen akan dicari dalam string kueri setiap permintaan HTTP GET. Jika pengubah "check_post" digunakan, maka entitas permintaan HTTP POST akan dicari untuk argumen parameter, ketika tidak ditemukan dalam string kueri setelah tanda tanya ('?') Di URL. Badan pesan hanya akan mulai dianalisis setelah jumlah data yang diiklankan telah diterima atau buffer permintaan penuh. Dalam hal yang tidak mungkin bahwa pengkodean chunked digunakan, hanya potongan pertama yang dipindai. Nilai-nilai parameter yang dipisahkan oleh batas potongan, dapat secara acak seimbang jika sama sekali. Kata kunci ini digunakan untuk mendukung parameter opsional <max_wait> yang sekarang diabaikan.

Jika parameter ditemukan diikuti oleh tanda yang sama ('=') dan nilai, maka nilainya di-hash dan dibagi dengan total berat server yang sedang berjalan. Hasilnya menunjukkan server mana yang akan menerima permintaan. Ini digunakan untuk melacak pengenal pengguna dalam permintaan dan memastikan bahwa ID pengguna yang sama akan selalu dikirim ke server yang sama selama

tidak ada server yang naik atau turun. Jika tidak ada nilai yang ditemukan atau jika parameter tidak ditemukan, maka algoritma round robin diterapkan. Perhatikan bahwa algoritma ini hanya dapat digunakan di backend HTTP. Algoritma ini statis secara default, yang berarti bahwa mengubah berat server dengan cepat tidak akan berpengaruh, tetapi ini dapat diubah menggunakan "hash-type".

8. Hdr

Header HTTP <name> akan dicari di setiap permintaan HTTP. Sama seperti dengan fungsi ACL 'hdr ()', nama header dalam kurung tidak peka huruf besar kecil. Jika header tidak ada atau jika tidak mengandung nilai apa pun, maka diterapkan algoritma roundrobin. Parameter opsional 'use_domain_only' tersedia, untuk mengurangi algoritma hash ke bagian domain utama dengan beberapa header tertentu seperti 'Host'. Misalnya, dalam nilai host "integra.its.ac.id", hanya "integra" yang akan dipertimbangkan. Algoritma ini secara asali bersifat statis, yang berarti bahwa mengubah berat server dengan cepat tidak akan berpengaruh, tetapi ini dapat diubah menggunakan "hash-type".

9. RDP-Cookie

RDP Cookie <nama> (atau "mstshash" jika dihilangkan) akan dicari dan dicirikan untuk setiap permintaan TCP yang masuk. Seperti halnya fungsi ACL 'req_rdp_cookie ()', tidak peka huruf besar kecil. Mekanisme ini berguna sebagai mode persistensi terdegradasi, seperti yang digunakan sebagai gantinya. Perhatikan bahwa agar ini berfungsi, frontend harus memastikan bahwa cookie RDP sudah ada di buffer permintaan. Untuk ini, Anda harus menggunakan aturan 'tcp-request content accept' yang dikombinasikan dengan 'req_rdp_cookie_cnt' ACL. Algoritma ini statis secara default, yang berarti bahwa mengubah berat server dengan cepat tidak akan berpengaruh, tetapi ini dapat diubah menggunakan "hash-type".

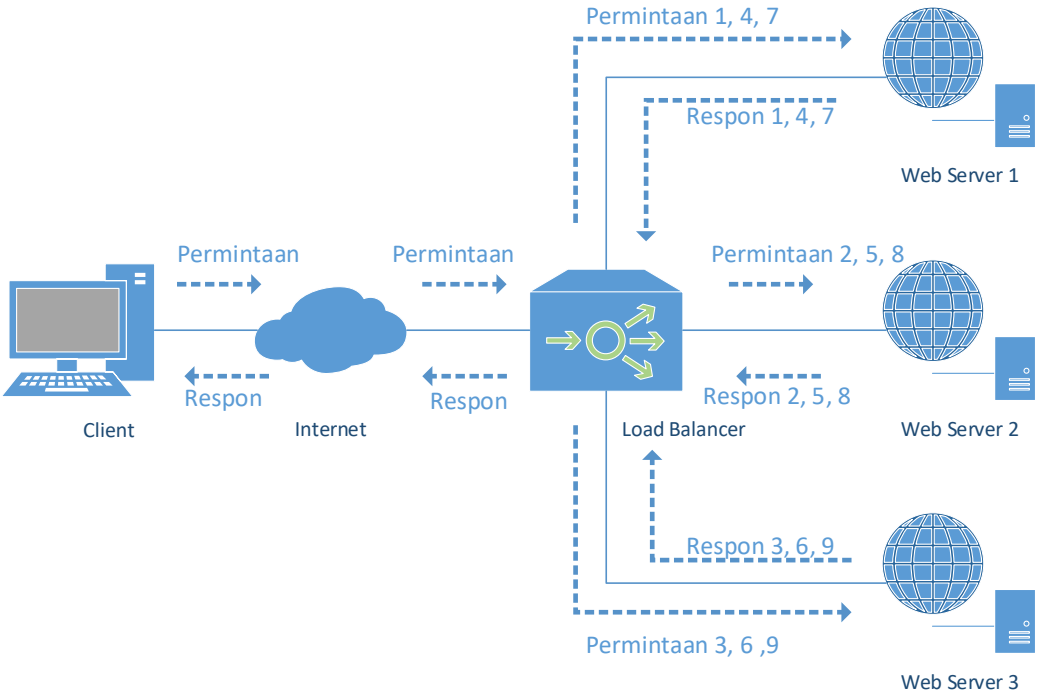
2.3.1.2 NGINX

Selain menjadi web server, NGINX juga dapat menjadi load balancer. Metode yang didukung oleh NGINX antara lain: [14]

2.3.1.2.1 Round Robin

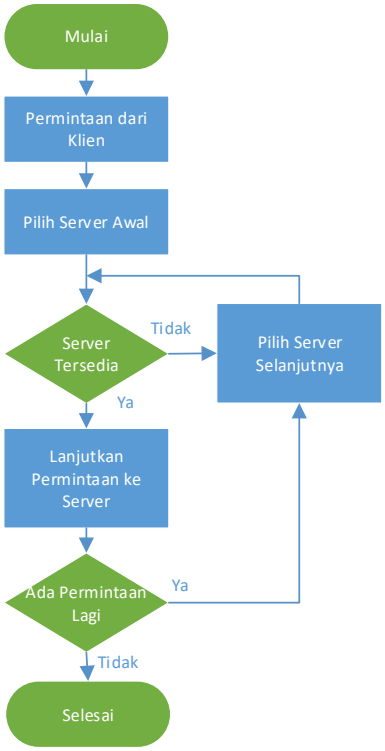
Round Robin adalah teknik load balancing standar untuk NGINX Plus dan NGINX. Load balancer berjalan melalui daftar server upstream secara berurutan,

menugaskan permintaan koneksi berikutnya ke masing-masing secara bergantian. Alur permintaan pada metode Round Robin dapat dilihat pada gambar 2.11.



Gambar 2.11 Alur Permintaan dan Respon Metode Round-robin

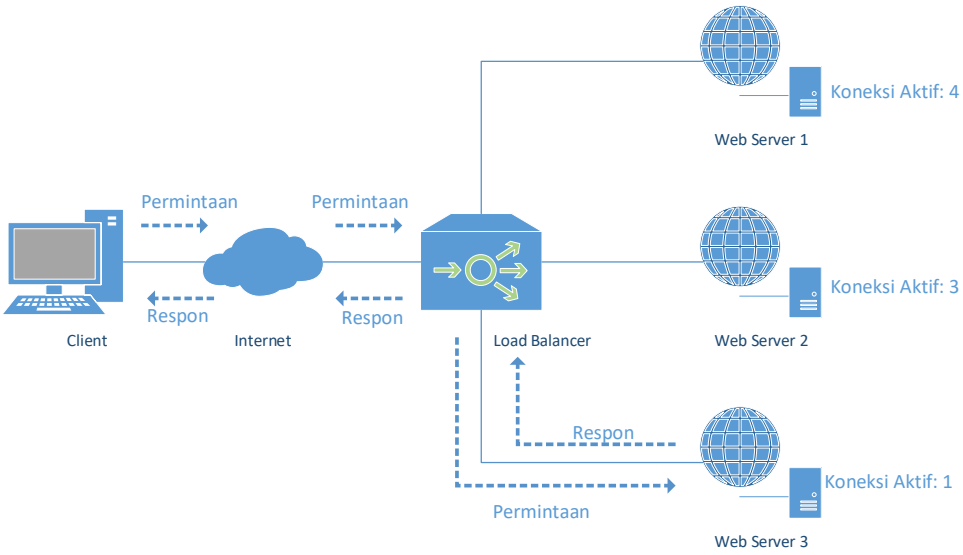
Sedangkan diagram alur metode round robin dapat dilihat seperti pada gambar 2.12.



Gambar 2.12 Diagram Alur Metode Round-robin

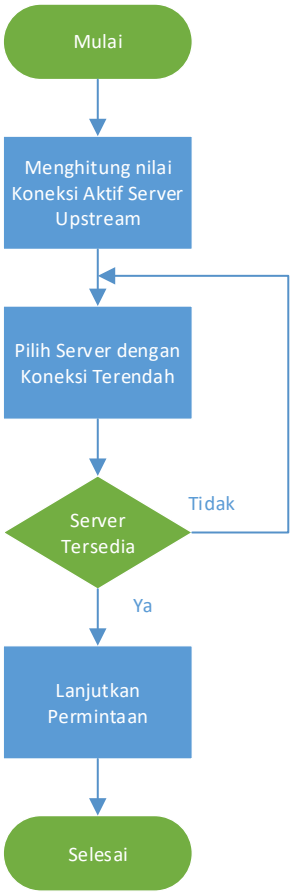
2.3.1.2.2 Least Connection

Dengan metode Least Connections, load balancer membandingkan jumlah koneksi aktif saat ini yang dimiliki setiap server, dan mengirim permintaan ke server dengan koneksi paling sedikit.



Gambar 2.13 Alur Permintaan dan Respon Metode Least Connection

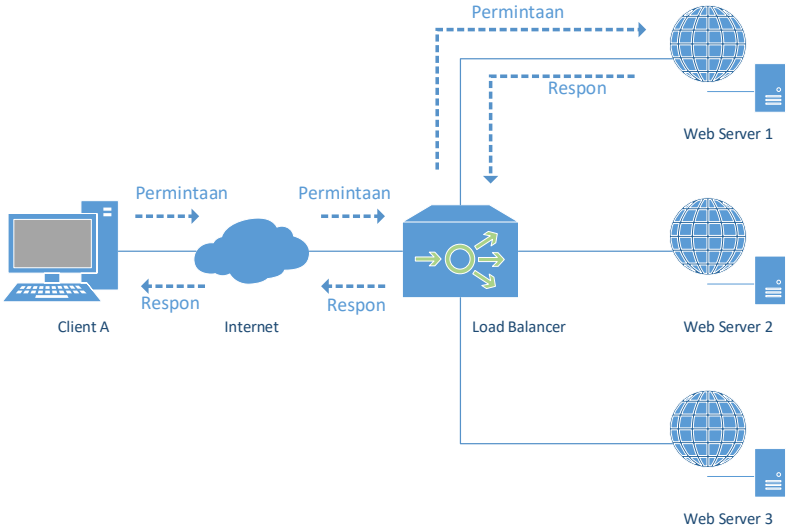
Pada Gambar 2.13 dilihat server 1 mempunyai koneksi aktif sebesar 4, server 2 mempunyai koneksi aktif sebesar 3, sedangkan server 3 mempunyai koneksi aktif sebesar 1. Karena koneksi aktif server 3 mempunyai nilai yang paling kecil, maka permintaan dari klien akan diarahkan ke server 3. Untuk diagram alur metode Leat Connection dapat dilihat seperti pada gambar 2.14.



Gambar 2.14 Daigram Alur Metode Least Connection

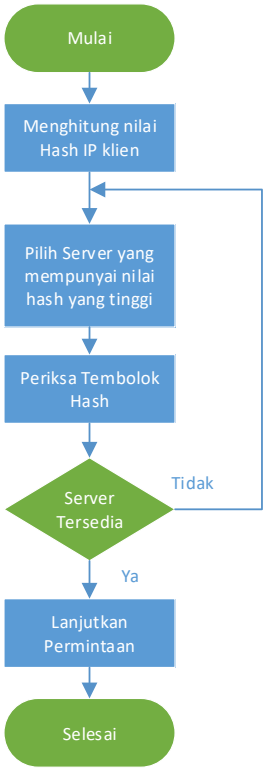
2.3.1.2.3 IP Hash

Server tempat permintaan dikirim ditentukan dari alamat IP klien. Dalam hal ini, baik tiga oktet pertama dari alamat IPv4 atau alamat IPv6 keseluruhan digunakan untuk menghitung nilai hash. Metode ini menjamin bahwa permintaan dari alamat yang sama sampai ke server yang sama kecuali jika tidak tersedia.



Gambar 2.15 Alur Permintaan dan Respon Metode IP Hash

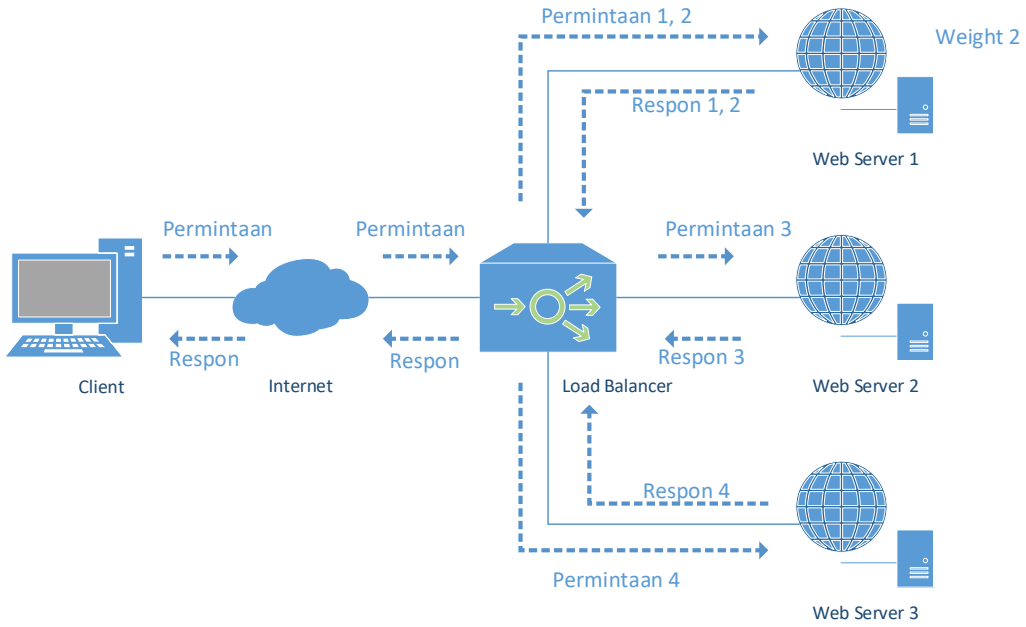
Dilihat dari gambar 2.15 Load Balancer akan menghitung nilai Hash dari alamat IP klien A kemudian menentukan server mana yang akan dipilih yaitu server1. Server tersebut akan terus menangani semua permintaan dari Klien A selamanya atau sampai server tersebut menghilang. Sedangkan untuk digram alur metode IP Hash dapat dilihat sperti pada gambar 2.16.



Gambar 2.16 Diagram Alur Permintaan Metode IP Hash

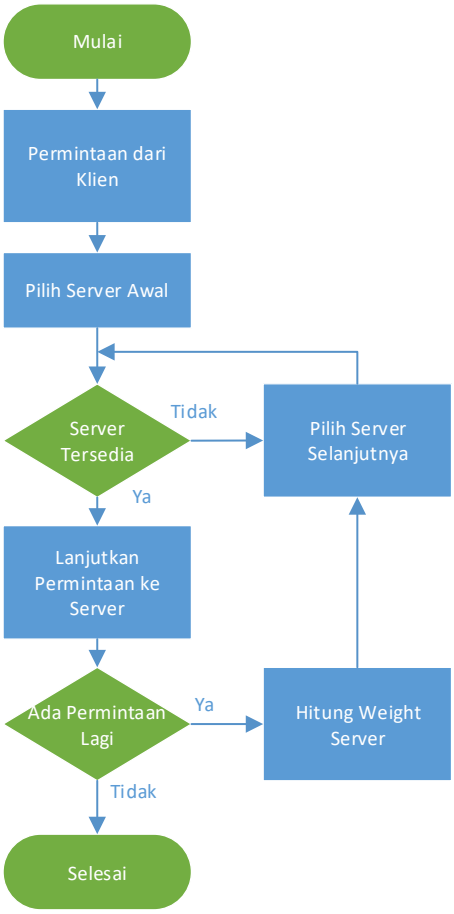
2.3.1.2.4 Weight

Secara default, NGINX mendistribusikan permintaan di antara server dalam kelompok sesuai dengan bobot mereka menggunakan metode Round Robin. Parameter weight juga dapat diberikan ke salah satu server. Secara default nilai weight adalah 1.



Gambar 2.17 Alur Permintaan dan Respon Metode Weight

Pada Gambar 2.17, diberikan contoh permintaan sebanyak 4. Karena web server 1 mempunyai weight sebesar 2 maka webserver1 akan manangani 2 permintaan sedangkan webserver 2 dan webserver 3 akan menangani masing masing satu permintaan. Sedangkan untuk diagram alur metode Weight dapat dilihat seperti pada gambar 2.18.



Gambar 2.18 Diagram Alur Permintaan Metode Weigth

2.3.1.2.5 Least Time

Dengan metode Least Time (hanya tersedia di NGINX Plus), penyeimbang beban secara matematis menggabungkan dua metrik untuk setiap server - jumlah koneksi aktif saat ini dan rata-rata waktu respons sebelumnya dan mengirim permintaan ke server dengan yang mempunyai nilai terendah. Least Time sangat cocok ketika server upstream memiliki waktu respons rata-rata yang sangat berbeda.

Web Server Load Balancing juga dapat mengefisiensi energi yang digunakan seperti pada penelitian yang dilakukan oleh Jörg Lenhardt, Kai Chen, dan Wolfram Schiffmann yang berjudul *Energy-Efficient Web Server Load Balancing*. Mereka melakukan penelitian tersebut karena peningkatan biaya energi dan dampak negatif lingkungan yang dihasilkan dari pembangkitan listrik, terutama

ketika mengandalkan bahan bakar fosil, meningkatkan kebutuhan komputasi hemat energi[6]. Selain optimalisasi perangkat keras, penggunaan solusi perangkat lunak canggih untuk mengurangi konsumsi energi merupakan bidang penting dalam penelitian saat ini. Mendistribusikan beban di antara server untuk optimalisasi kinerja adalah area yang diteliti dengan baik. Ada potensi besar untuk mengurangi konsumsi energi dengan teknik serupa. Dalam makalah tersebut, mereka menguji penerapan strategi load-balancing (atau ketidakseimbangan) hemat energi untuk permintaan server web yang dikirim ke perkumpulan server web. Permintaan tunggal biasanya mengarah ke fraksi kecil dari kebutuhan kinerja. Mereka sangat cocok untuk strategi yang mengandalkan asumsi beban yang sepenuhnya terbagi-bagi. Mereka menunjukkan bahwa dapat mengurangi konsumsi daya secara terus menerus. Akhirnya, mereka dapat mengurangi konsumsi energi di peternakan server web, secara signifikan.

Pada Penelitian lain yang dilakukan oleh Shuo Di dan Weimin Zheng dengan judul *Request dispatching algorithms for web server clusters based on load balancing* melakukan *Request dispatching* atau permintaan pengiriman yang secara terpusat menerima semua permintaan HTTP yang masuk dan mengirimkannya ke server di cluster untuk mencapai paralelisme[7]. Penelitian tersebut menunjukkan bahwa efisiensi pengiriman permintaan ditentukan oleh apakah beban server seimbang selama pengiriman. Penelitian tersbut juga memunculkan karakteristik beban yang dihasilkan oleh permintaan HTTP, memberikan metode pengambilan sampel yang efisien, dan menyajikan algoritma pengiriman permintaan baru yang dapat memprediksi beban dan menyeimbangkannya. Metode ini dapat digunakan untuk membangun cluster server Web dengan perangkat server yang berbeda-beda.

Selain melakukan penelitian di Load Balancingnya, juga terdapat penelitian yang melakukan pengelolaan performansi dari kluster web server. Seperti yang dilakukan oleh G. Pacifici, M. Spreitzer, A.N. Tantawi, dan A. Youssef dengan judul *Performance management for cluster-based web services*. Mereka menyajikan arsitektur dan implementasi prototipe dari sistem manajemen kinerja untuk layanan web berbasis kluster. Sistem tersebut mendukung banyak kelas lalu lintas layanan web dan mengalokasikan sumber daya server secara dinamis sehingga memaksimalkan nilai yang diharapkan dari fungsi utilitas kluster

yang diberikan dalam menghadapi beban yang berfluktuasi. Utilitas kluster adalah fungsi dari kinerja yang disampaikan ke berbagai kelas, dan ini mengarah ke layanan yang berbeda. Mereka menggunakan waktu respon rata-rata sebagai metrik kinerja. Sistem manajemen transparan yang tidak memerlukan perubahan dalam kode klien, kode server, atau antarmuka jaringan di antara mereka. Sistem tersebut melakukan tiga tugas manajemen kinerja yaitu alokasi sumber daya, load balancing, dan perlindungan server yang berlebihan. Mereka menggunakan dua tingkat manajemen yang berlapis. Lapisan terdalam berpusat pada antrian dan penjadwalan pesan permintaan. Sedangkan lapisan luar adalah loop kontrol umpan balik yang secara berkala menyesuaikan bobot penjadwalan dan alokasi server dari level dalam. Pengontrol umpan balik didasarkan pada model prinsip pertama perkiraan dari sistem, dengan parameter yang berasal dari pemantauan berkelanjutan. Mereka melaporkan bahwa hasil eksperimen yang dilakukan menunjukkan perilaku dinamis dari sistem[8].

2.3.2 Network Load Balancing

Network Load Balancing adalah kemampuan untuk menyeimbangkan lalu lintas di dua tautan WAN tanpa menggunakan protokol routing kompleks seperti BGP.

Kemampuan ini menyeimbangkan sesi jaringan melalui beberapa koneksi untuk menyebarkan jumlah bandwidth yang digunakan oleh setiap pengguna LAN, sehingga meningkatkan jumlah total bandwidth yang tersedia. Misalnya, pengguna memiliki satu koneksi WAN ke Internet yang beroperasi pada 2.5 Mbit/dtk. Mereka ingin menambahkan koneksi broadband (kabel, DSL, nirkabel, dll) kedua yang beroperasi pada 3.5 Mbit/dtk. Ini akan memberi mereka total 6 Mbit/dtk bandwidth saat menyeimbangkan sesi.

Sesi balancing tidak hanya itu, menyeimbangkan sesi di setiap tautan WAN. Ketika browser Web terhubung ke Internet, mereka biasanya membuka beberapa sesi, satu untuk teks, yang lain untuk gambar, yang lain untuk beberapa gambar lain, dll. Masing-masing sesi ini dapat diseimbangkan di seluruh koneksi yang tersedia. Aplikasi FTP hanya menggunakan satu sesi sehingga tidak seimbang; namun jika koneksi FTP sekunder dibuat, maka mungkin seimbang sehingga secara

keseluruhan, lalu lintas didistribusikan secara merata di berbagai koneksi dan dengan demikian memberikan peningkatan throughput secara keseluruhan.

Selain itu, load balancing jaringan umumnya digunakan untuk menyediakan redundansi jaringan sehingga dalam hal terjadi pemutusan tautan WAN, akses ke sumber daya jaringan masih tersedia melalui tautan sekunder. Redundansi adalah persyaratan utama untuk rencana kesinambungan bisnis dan umumnya digunakan bersama dengan aplikasi penting seperti VPN dan VoIP.

Akhirnya, kebanyakan sistem load balancing jaringan juga menggabungkan kemampuan untuk menyeimbangkan lalu lintas keluar dan masuk. Inbound load balancing umumnya dilakukan melalui DNS dinamis yang dapat dibangun ke dalam sistem, atau disediakan oleh layanan atau sistem eksternal. Memiliki layanan DNS dinamis dalam sistem umumnya dianggap lebih baik dari penghematan biaya dan sudut pandang kontrol secara keseluruhan.

Seperti riset yang dilakukan Minh-Tuan Thai, Ying-Dar Lin , Po-Ching Lin, Yuan-Cheng Lai dengan Judul *A joint network and server load balancing algorithm for chaining virtualized network functions* yaitu dengan menghubungkan fungsi jaringan virtualisasi (VNF) secara efektif untuk menyebarkan layanan jaringan di pusat data operator jaringan[4]. Dua masalah umum yang timbul dalam penyebaran tersebut adalah load balancing jaringan dan load balancing server. Dalam riset tersebut, disebabkan oleh argumen bahwa dua kekhawatiran tersebut harus diatasi bersama untuk secara efisien menjejalkan VNF di lingkungan pusat data, kami mengusulkan algoritma 2-fase, *Nearest First and Local-Global Transformation* (NF-LGT), yang secara bersamaan mendukung jaringan dan load balancing layanan. Algoritma ini pertama membangun rantai layanan dengan strategi tamak yang keduanya menganggap latensi jaringan dan latensi server. Kemudian teknik pencarian diterapkan untuk meningkatkan solusi. Kami telah mengimplementasikan algoritma menggunakan konsep jaringan yang ditentukan perangkat lunak (SDN)/OpenFlow. Hasil eksperimen menunjukkan bahwa, dibandingkan dengan pendekatan sekuensial, NF-LGT meningkatkan penggunaan bandwidth sistem hingga 45%.

Riset lain yang dilakukan oleh Minh-Tuan Thai, Ying-Dar Lin, Po-Ching Lin, Yuan-Cheng Lai dengan judul *Hash-based load balanced traffic steering on*

softswitches for chaining virtualized network functions membahas tentang Solusi load balancing sebelum untuk rantai fungsi jaringan virtual menyebabkan kontrol yang signifikan dan overhead data pada permintaan khusus di perangkat keras jaringan[5]. Dalam riset tersebut, juga menyajikan desain, implementasi, dan evaluasi Pengarahan Lalu Lintas Berbasis Hash pada Softswitches (HATS), mekanisme penyeimbang beban yang bertujuan untuk mengurangi kerugian tersebut. Metode tersebut memanfaatkan teknik flow hashing yang diimplementasikan pada softswitch untuk melakukan server dan load balancing jaringan tanpa memicu control plane. Mereka menerapkan desain ini menggunakan OpenDayLight controller dan Open vSwitch platform. Implementasi menunjukkan bahwa HATS dapat dengan mudah diimplementasikan dengan perangkat keras jaringan komoditas. Selain itu, hasil eksperimen menegaskan bahwa HATS dapat mengurangi jumlah entri aliran dan waktu chaining layanan hingga 85% dan 93%, masing-masing, bila dibandingkan dengan Least Load First (LLF), algoritma chaining layanan berbasis kontroler.

2.3.3 Aplikasi Pengujian Webserver

Ada banyak aplikasi pengujian pada webserver. Salah satu aplikasi yang sering digunakan adalah Apache JMeter. Aplikasi Apache JMeter adalah perangkat lunak open source, aplikasi Java 100% murni yang dirancang untuk memuat perilaku tes kinerja dan mengukur kinerja [12]. Awalnya dirancang untuk menguji Aplikasi Web tetapi sejak itu diperluas ke fungsi uji lainnya.

Apache JMeter dapat digunakan untuk menguji kinerja baik pada sumber daya statis dan dinamis. Aplikasi tersebut dapat digunakan untuk mensimulasikan beban berat pada server, sekelompok server, jaringan atau objek untuk menguji kekuatannya atau untuk menganalisis kinerja keseluruhan di bawah jenis beban yang berbeda. Fitur dari apche JMeter meliputi[12]:

1. Kemampuan memuat dan menguji kinerja banyak jenis aplikasi/ server/protokol yang berbeda:
    - Web - HTTP, HTTPS (Java, NodeJS, PHP, ASP.NET, …)
    - SOAP / REST Webservices
    - FTP

- Database via JDBC
- LDAP
- Message-oriented middleware (MOM) via JMS
- Mail - SMTP(S), POP3(S) and IMAP(S)
- Native commands or shell scripts
- TCP
- Java Objects

2. Uji IDE berfitur lengkap yang memungkinkan perekaman Rencana Uji cepat (dari Browser atau aplikasi asli), membuat dan debugging.
3. Mode Command-line (Non GUI/headless mode) untuk memuat tes dari OS yang kompatibel dengan Java (Linux, Windows, Mac OSX,…)
4. Laporan HTML dinamis yang lengkap dan siap disajikan
5. Korelasi mudah melalui kemampuan mengekstrak data dari format respons paling populer, HTML, JSON, XML, atau format teks apa pun
6. Lengkapi portabilitas dan kemurnian 100% Java.
7. Kerangka multi-threading penuh memungkinkan sampling konkuren oleh banyak utas dan pengambilan sampel secara simultan dari fungsi yang berbeda oleh kelompok utas yang terpisah.
8. Caching dan analisis offline/replaying hasil tes.
9. Inti Sangat Dapat Diperluas:
    - Pluggable Samplers memungkinkan kemampuan pengujian yang tidak terbatas.
    - Scriptable Samplers (bahasa yang kompatibel dengan JSR223 seperti Groovy dan BeanShell)
    - Beberapa statistik beban dapat dipilih dengan pengatur waktu yang dapat dilepas.
    - Analisis data dan visualisasi plugin memungkinkan ekstensibilitas yang luar biasa serta personalisasi.
    - Fungsi dapat digunakan untuk memberikan input dinamis ke tes atau menyediakan manipulasi data.

- Integrasi Berkelanjutan yang Mudah melalui perpustakaan Open Source pihak ketiga untuk Maven, Graddle dan Jenkins

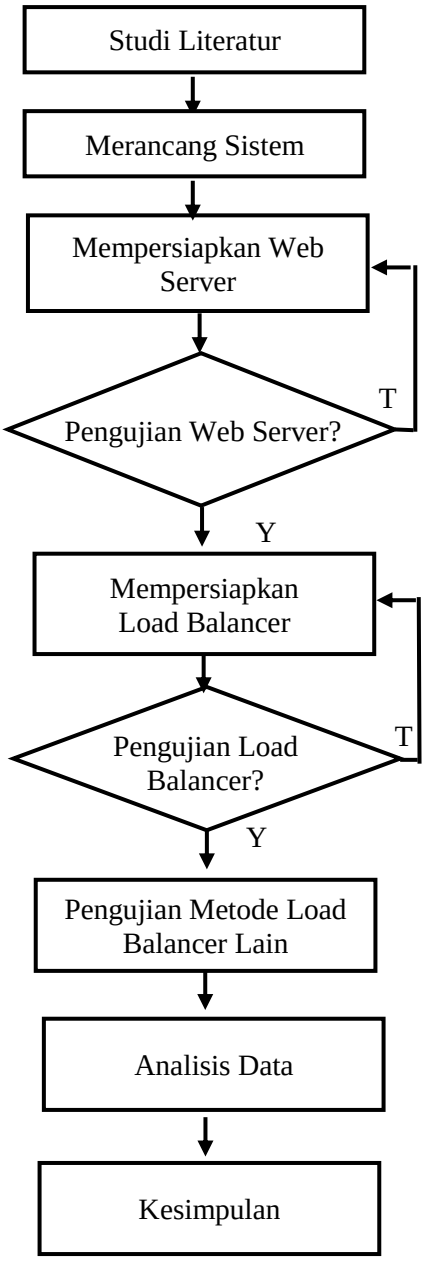
JMeter bukanlah browser melainkan sebua perangkat lunak yang berfungsi pada level protokol. Sejauh layanan web dan layanan jarak jauh yang ada, JMeter terlihat seperti browser (atau lebih tepatnya, beberapa browser); Namun JMeter tidak melakukan semua tindakan yang didukung oleh browser. Secara khusus, JMeter tidak mengeksekusi Javascript yang ditemukan di halaman HTML. Juga tidak membuat halaman HTML seperti yang browser lakukan (memungkin untuk melihat respon HTML dll, tetapi timing tidak termasuk dalam sampel, dan hanya satu sampel dalam satu thread yang ditampilkan pada suatu waktu).

Halaman ini sengaja dikosongkan

BAB 3
METODOLOGI PENELITIAN

3.1 Metode Penelitian

Pada bagian ini adalah metode yang digunakan untuk menjawab permasalahan penelitian yang dilakukan serta tahapan penelitian. Tahapan penelitian dapat dilihat seperti pada gambar 3.1.



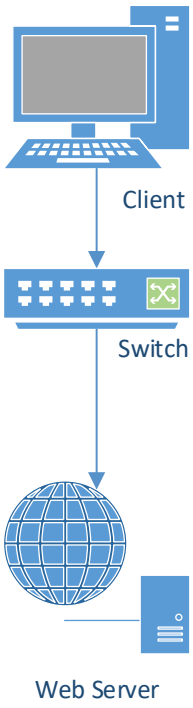
Gambar 3.1 Blok Diagram Metode Penelitian

3.2 Rancangan Sistem

Rancangan yang dibuat menggunakan web server tanpa menggunakan load balancer dan dengan menggunakan load balancer.

3.2.1 Web Server tanpa Menggunakan Load Balancer

Topologi yang digunakan pada web server yang tanpa menggunakan load balancer ditunjukkan seperti pada gambar 3.2. Pada topologi tersebut terdapat klien web dan server web dihubungkan oleh switch. Alokasi alamat IP antara server dengan klien dibuat dalam satu jaringan. Sehingga klien dapat langsung mengakses layanan yang diberikan oleh server tanpa melalui router terlebih dahulu.



Gambar 3.2 Topologi Pengujian Web Server tanpa Menggunakan Load Balancer

Spesifikasi hardware komputer klien adalah sebagai berikut:

Tabel 3.1 Spesifikasi Hardware Komputer Klien

Jenis	Spesifikasi
CPU	2 Core 4 Threads @ 2,5 GHz
Memori	8 GB
Penyimpanan	250GB SSD + 1 TB HDD
Koneksi	Kabel Gigabit LAN

Sedangkan untuk perangkat lunak yang digunakan pada klien adalah:

Tabel 3.2 Spesifikasi Software Komputer Klien

Jenis	Rincian
Sistem Operasi	Windows 10 Pro 64Bit
Aplikasi penguji	Apache JMeter versi 4

Untuk switch mempunyai spesifikasi berikut:

- 24 RJ-45 auto-negotiating 10/100/1000 ports
- 4 SFP 1000 Mbps ports
- 1U – Height

Sedangkan untuk web server mempunyai spesifikasi berikut:

Tabel 3.3 Spesifikasi Web Server

Jenis	Spesifikasi
vCPU	2 Core
Memori	256 MB
Penyimpanan	16 GB
Koneksi	Kabel Gigabit LAN
Sistem Operasi	Debian 8
Web server	Apache2 web server

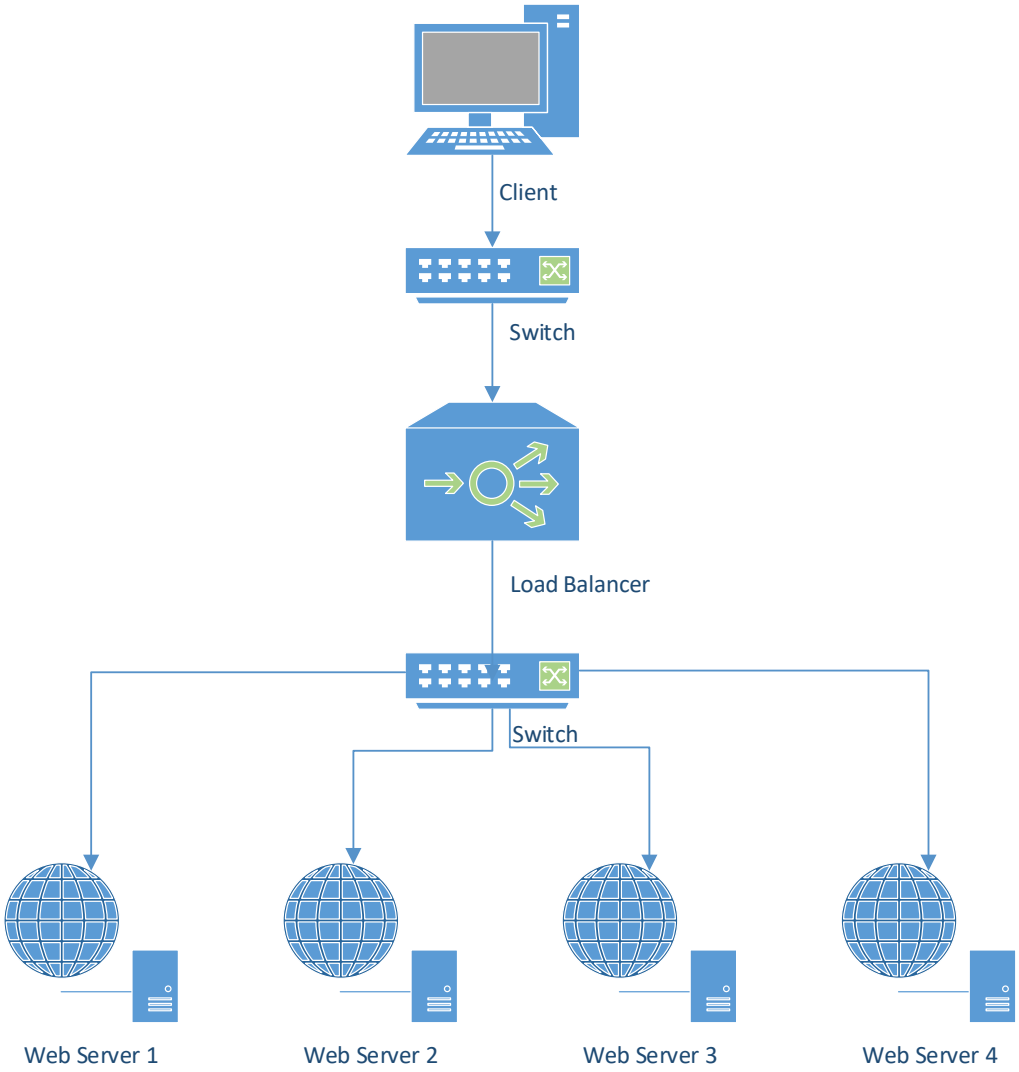
Alamat IP yang dipakai adalah sebagai berikut:

Tabel 3.4 alamat IP host di pengujian webserver tanpa load balancer

Host	Alamat IP
Klien	10.199.13.19
Switch	-
Webserver	10.199.13.9

3.2.2 Web Server dengan Menggunakan Load Balancer

Sedangkan Topologi server web yang menggunakan load balancer ditunjukkan pada gambar 3.3. Pada topologi tersebut klien secara tidak langsung terhubung dengan web server, melainkan terlebih dahulu melewati load balancer. Load balancer tersebut mempunyai dua kaki, yang pertama terhubung dengan jaringan klien dan yang kedua terhubung dengan jaringan server web. Untuk dapat mengakses layanan HTTP yang diberikan server, klien harus mengakses alamat IP dari load balancer. Load balancer tersebut kemudian akan meneruskan permintaan HTTP dari klien ke server web yang sebenarnya.



Gambar 3.3 Topologi Pengujian Web Server dengan Menggunakan Load Balancer

Untuk spesifikasi klien dan switch yang dipakai mempunyai spesifikasi yang sama dengan pengujian web server tanpa menggunakan Load Balancer seperti pada Tabel 3.3.

Alamat IP yang dipakai adalah sebagai berikut:

Tabel 3.5 alamat IP host di pengujian webserver tanpa load balancer

Host	Alamat IP
Klien	10.199.13.19
Switch	-
Load Balancer	10.199.13.9 dan 10.199.14.200
Webserver1	10.199.14.191
Webserver2	10.199.14.192
Webserver3	10.199.14.193
Webserver4	10.199.14.194
Webserver5	10.199.14.195
Webserver6	10.199.14.196
Webserver7	10.199.14.197
Webserver8	10.199.14.198

3.3 Metode Pengujian Permintaan HTTP ke Web Server tanpa Load Balancer

Pada pengujian permintaan HTTP digunakan digunakan server web yang menggunakan Apache2 web server dengan halaman asali. Sedangkan pada klien menggunakan aplikasi Apache JMeter. Pada aplikasi JMeter dibuatkan sebuah kategori Thread dengan jumlah thread sesuai kebutuhan yang mensimulasikan jumlah pengguna yang mengakses web server. Dalam thread tersebut juga diberikan durasi pengujian sesuai dengan kebutuhan. Didalam thread terdapat juga sampler berupa HTTP request yang berfungsi untuk meminta respon HTTP ke server. Untuk menyimpan hasil respon tersebut diberikan Listener berupa Graph Results. Graph Results tersebut dapat menyimpan hasil respon ke dalam berkas. Berkas terserbut berisi data berupa timestamp, elapsed, label responseCode, responseMessage,

threadName, dataType, success, failureMessages, bytes, sentBytes, grpThread, allThread, latency, IdleTime, dan Connect.

3.3.1 Pengujian dengan Hit yang Berbeda

Pengujian ini dilakukan untuk mengetahui hubungan antara beban server dengan ketersediaan layanan web dan latency dari jawaban server ke klien. Pada pengujian ini sumber daya yang digunakan adalah menggunakan Prosesor satu inti dengan memori 256 MB, sedangkan beban yang diberikan ke server adalah berupa permintaan HTTP yang dilakukan dari 100 sampai 1000 per detik dengan durasi pengujian selama 60 detik.

3.3.2 Pengujian dengan Jumlah Memori yang Berbeda

Pengujian ini dilakukan untuk mengetahui hubungan antara sumber daya yang ada pada server dengan ketersediaan dan latency dari jawaban yang diberikan server. Perbedaan sumber daya yang divariabelkan yaitu memori pada server. Menggunakan Prosesor satu inti dengan memori 128 – 1024 MB, Hit dilakukan dari 5000 per detik dengan durasi pengujian selama 60 detik.

3.4 Metode Pengujian Permintaan HTTP ke Web Server dengan Load Balancer

3.4.1 Pengujian dengan Hit yang Berbeda

Pengujian ini dilakukan untuk mengetahui hubungan antara beban server yang melalui penyeimbang beban dengan ketersediaan layanan web dan latency dari jawaban server ke klien. Pada pengujian ini diberikan sumber daya berupa dua buah server upstream. Sedangkan untuk beban diberikan permintaan HTTP yang dilakukan dari 100 -1000 per detik dengan durasi pengujian selama 60 detik. Selanjutnya hasil dari pengujian ini akan dibandingkan dengan web server yang tanpa melalui penyeimbang beban.

3.4.2 Pengujian dengan Jumlah Server Upstream yang Berbeda

Pengujian ini dilakukan untuk mengetahui hubungan antara jumlah upstream server dengan dengan ketersediaan layanan web dan latency dari jawaban server ke klien. Pada pengujian ini menggunakan satu sampai dengan delapan buah server upstream. Sedangkan beban server diberikan permintaan HTTP sebesar 5000 per detik dengan durasi pengujian selama 300 detik.

3.4.3 Pengujian Metode Load Balancing yang Berbeda dengan Upstream yang Identik

Pengujian ini dilakukan untuk mengetahui hubungan antara metode pembagi beban web server dengan dengan ketersediaan layanan web dan latency dari jawaban server ke klien. Pada pengujian ini digunakan empat buah server upstream dengan spesifikasi yang sama. Untuk spesifikasi semua server upstream ditunjukkan seperti pada Tabel 3.3. Beban yang diberikan adalah permintaan HTTP sebesar 1000 per detik dengan durasi pengujian selama 300 detik.

3.4.4 Pengujian Metode Load Balancing yang Berbeda dan Spesifikasi Server Upstream yang Berbeda

Pengujian ini dilakukan untuk mengetahui hubungan antara metode pembagi beban web server dengan dengan ketersediaan layanan web dan latency dari jawaban server ke klien. Pada pengujian ini digunakan empat buah server upstream dengan spesifikasi memori yang berbeda, Beban yang diberikan adalah permintaan HTTP sebesar 1000 per detik dengan durasi pengujian selama 300 detik.

Halaman ini sengaja dikosongkan

BAB 4
HASIL DAN PEMBAHASAN

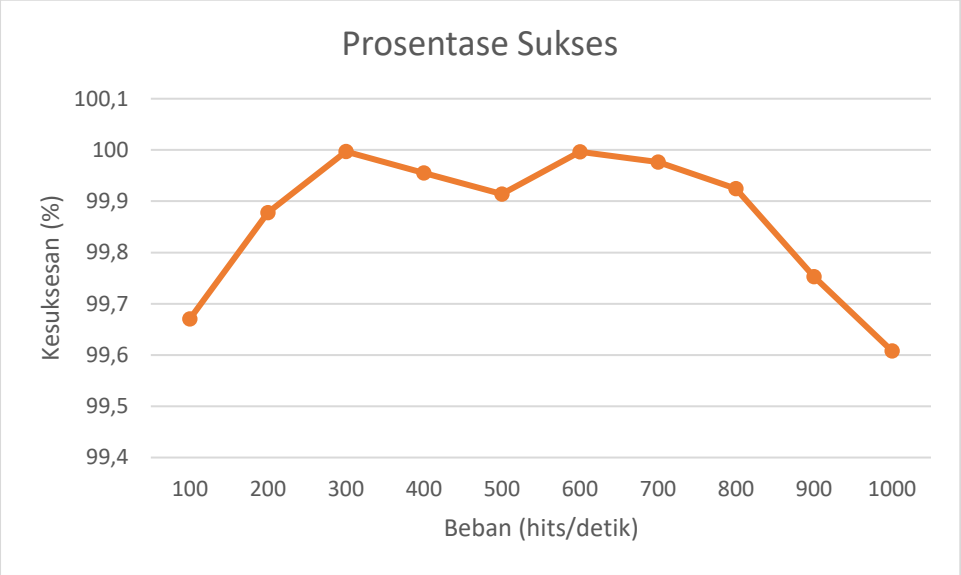
4.1 Data Hasil Pengujian

4.1.1 Pengujian tanpa Load Balancer

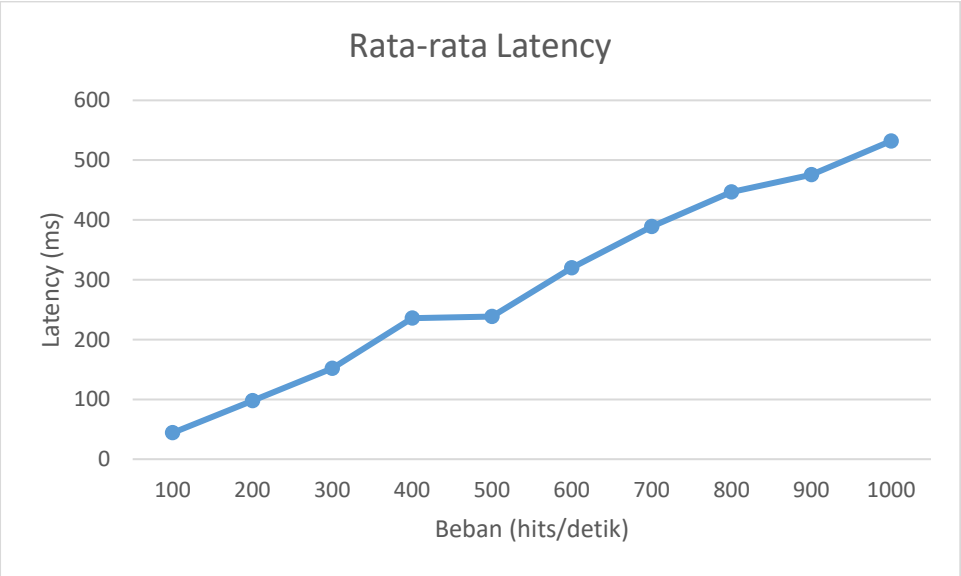
Pada Pengujian ini akan dilakukan permintaan koneksi HTTP tanpa menggunakan *Load Balancer* tetapi langsung ke server penyedia layanan web. Permintaan koneksi HTTP per detiknya akan dibuat bermacam macam. Dari perubahan tersebut akan dihitung sukses tidaknya permintaan yang direspon oleh server. Selain itu juga dihitung rata-rata *latency* selama dilakukan pengujian. Durasi menjalankan yaitu sebesar 60 detik.

Tabel 4.1 Hasil Pengujian Permintaan HTTP tanpa menggunakan *Load Balancer* dengan Jumlah Permintaan yang Berbeda

Hits (Permintaan/dtk)	Sukses	Tidak Sukses	Prosentase Sukses (%)	Rata-rata Latency (ms)
100	98303	325	99,67	44
200	94183	115	99,88	98
300	95348	3	100,00	152
400	84680	38	99,96	236
500	94988	82	99,91	239
600	91571	3	100,00	320
700	88335	21	99,98	389
800	90331	68	99,92	446
900	91212	226	99,75	476
1000	90472	356	99,61	532



Gambar 4.1 Grafik Prosentase kesuksesan dari Jawaban Server



Gambar 4.2 Grafik Latency dari Jawaban Server ke Klien tanpa Load Balancer dengan Permintaan yang berbeda..

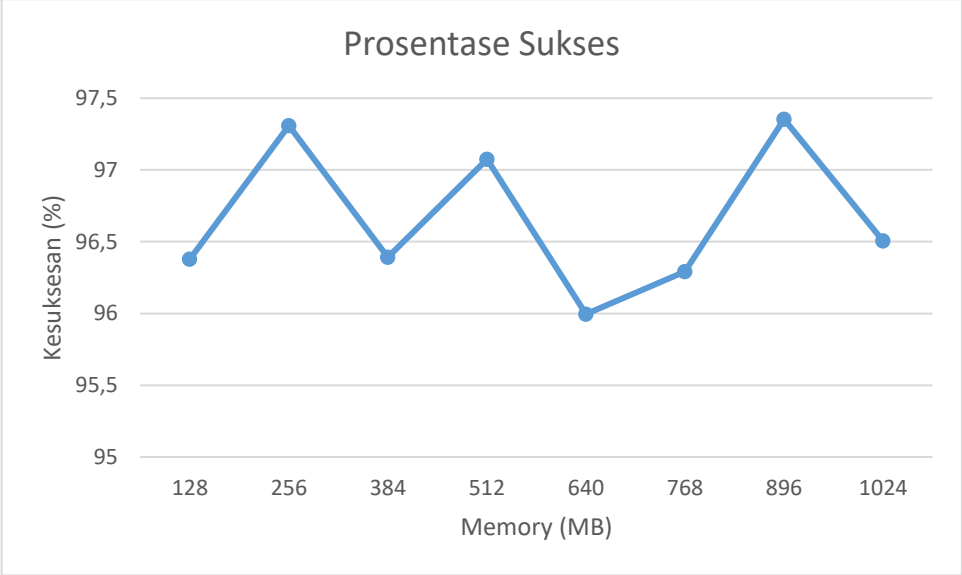
4.1.2 Pengujian dengan Jumlah Memori yang Berbeda tanpa load Balancer

Pada pengujian kali ini, dilakukan juga permintaan request HTTP ke server tanpa menggunakan Load Balancer. Tetapi kali ini menggunakan jumlah memori yang berbeda beda. Jumlah permintaan http ke server sebesar 1000 kali per detiknya dengan durasi pengujian selama 60 detik.

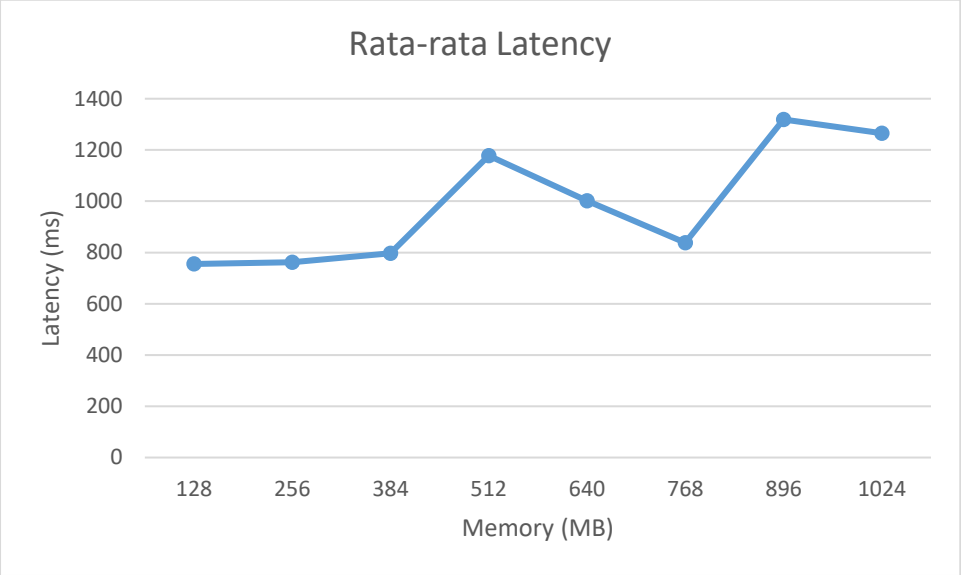
Tabel 4.2 Hasil Pengujian Permintaan HTTP tanpa menggunakan *Load Balancer* dengan Jumlah Memori yang Berbeda.

Memori (MB)	Sukses	Tidak Sukses	Prosentase Sukses (%)	Rata-rata Latency (ms)
128	173705	6528	96,38	755
256	170144	4705	97,31	762
384	171949	6435	96,39	797
512	178254	5374	97,07	1178
640	163320	6813	96,00	1002
768	163126	6281	96,29	838
896	162390	4415	97,35	1319
1024	159105	5761	96,51	1265

Dari pengujian yang dilakukan didapatkan hasil seperti pada tabel 4.2. Dari hasil tersebut jika diubah menjadi grafik maka akan didapatkan grafik seperti pada gambar 4.3 dan 4.4.



Gambar 4.3 Grafik Prosentase dari Jawaban Server ke Klien tanpa menggunakan load balancer dengan memori yang berbeda



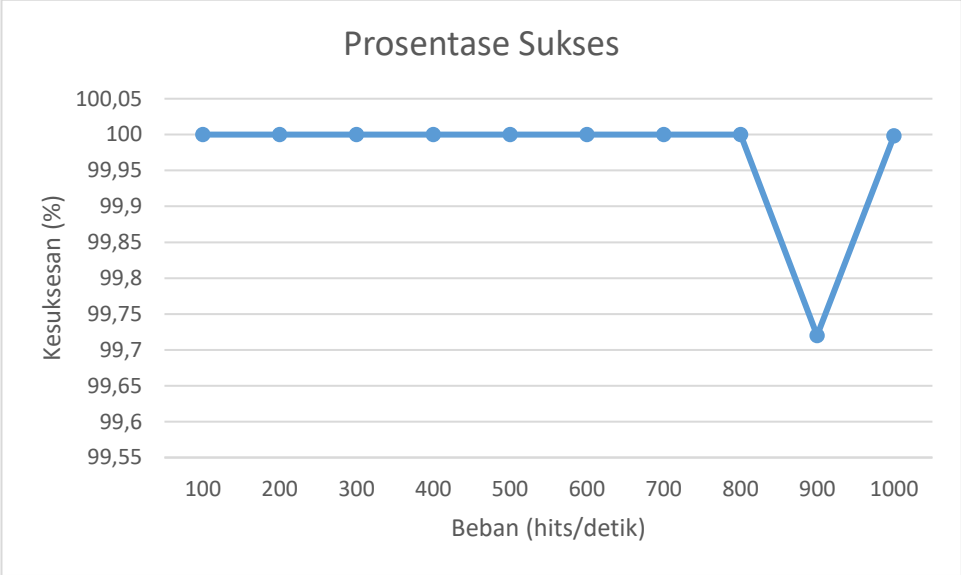
Gambar 4.4 Grafik Latency dari Jawaban Server ke Klien tanpa Load Balancer dengan Memori yang berbeda.

4.1.3 Pengujian dengan Load Balancer

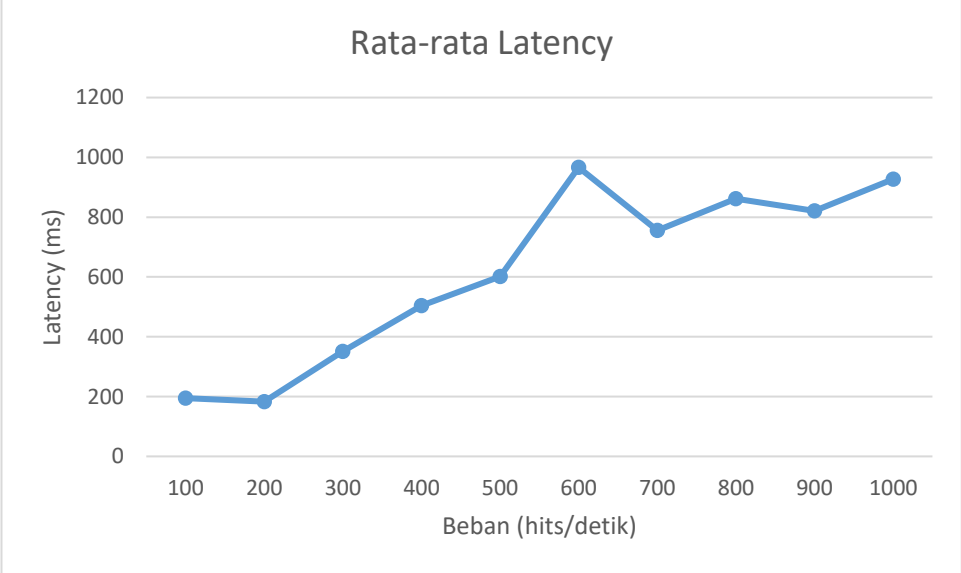
Pada pengujian kali ini, yaitu hampir sama dengan pengujian 4.1 tetapi kali ini menggunakan *load balancer* dengan *upstream* server sebanyak dua buah. Pengujian juga dilakukan dengan bermacam macam jumlah permintaan setiap detiknya. Durasi pengujian ini juga dilakukan sebesar 60 detik.

Tabel 4.3 Hasil Pengujian Permintaan HTTP menggunakan *Load Balancer* dengan Jumlah Permintaan yang Berbeda.

Hits	Sukses	Tidak Sukses	Prosentase Sukses (%)	Rata-rata Latency (ms)
100	29001	0	100	195
200	53832	0	100	183
300	54015	0	100	351
400	49944	0	100	504
500	48994	0	100	601
600	34265	0	100	966
700	51435	0	100	756
800	51694	0	100	862
900	56629	159	99,720	821
1000	57815	1	99,998	927



Gambar 4.5 Grafik Prosentase dari Jawaban Server ke Klien menggunakan load balancer dengan jumlah permintaan yang berbeda



Gambar 4.6 Grafik Latency dari Jawaban Server ke Klien dengan Load Balancer dengan Jumlah Permintaan yang berbeda.

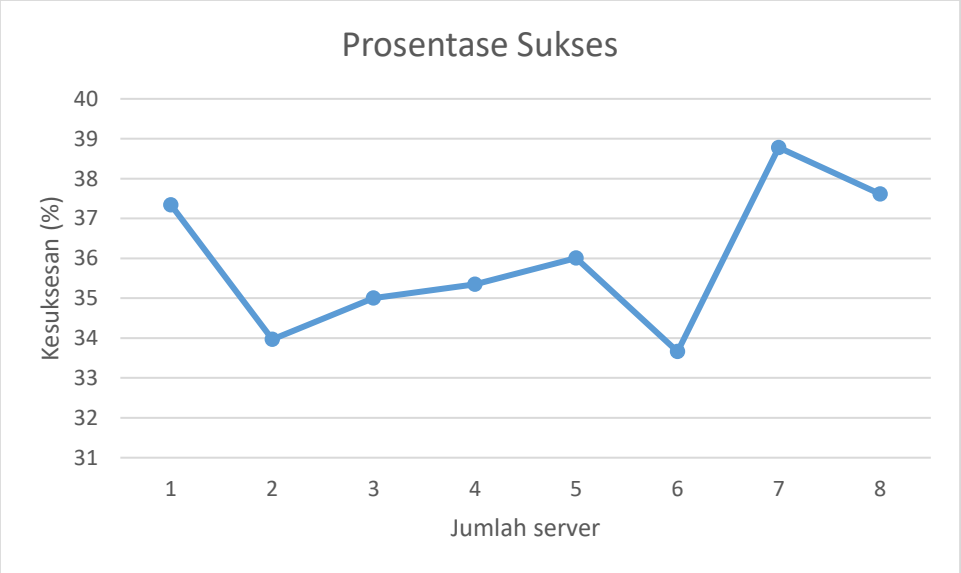
4.1.4 Pengujian Load Balancer dengan Jumlah Server yang Berbeda

Pada pengujian kali ini, dilakukan juga permintaan HTTP ke server web, Tetapi kali ini server web menggunakan *load balancer* dengan jumlah *upstream*

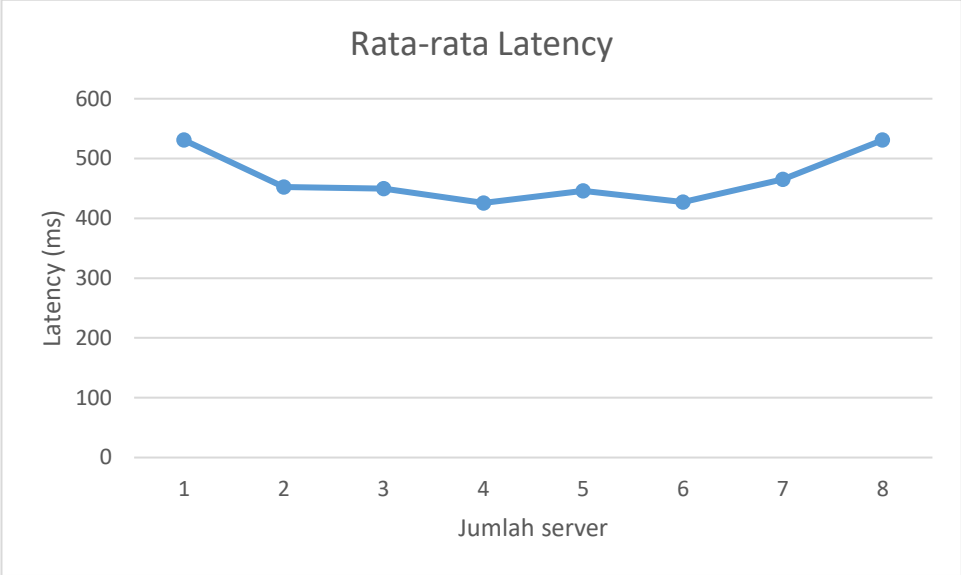
server yang berbeda beda. Jumlah permintaan diberikan sebasar 5000 kali per detik dengan durasi pengujian sebesar 60 detik.

Tabel 4.4 Hasil Pengujian Permintaan HTTP menggunakan *Load Balancer* dengan Jumlah *upstream* server yang Berbeda.

Server	Sukses	Tidak Sukses	Prosentase Sukses (%)	Rata-rata Latency (ms)
1	77689	130364	37,341	531
2	82342	160098	33,964	453
3	89533	166227	35,007	450
4	85389	156174	35,349	426
5	80297	142723	36,004	446
6	83848	165253	33,660	427
7	87405	137987	38,779	465
8	85566	141938	37,611	531



Gambar 4.7 Grafik Prosentase dari Jawaban Server ke Klien dengan menggunakan load balancer dengan upstream server yang berbeda



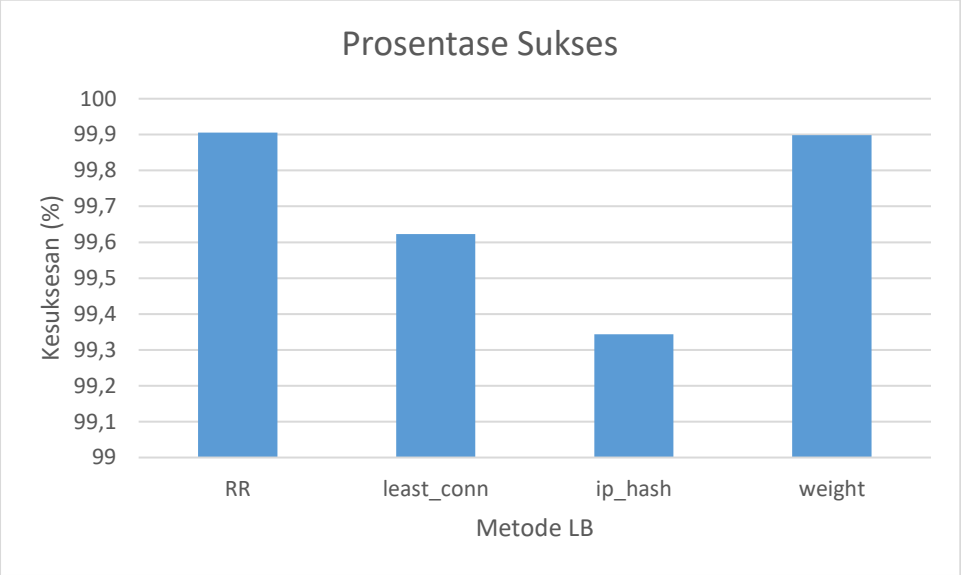
Gambar 4.8 Grafik Latency dari Jawaban Server ke Klien menggunakan Load Balancer dengan Jumlah Upstream server yang berbeda.

4.1.5 Pengujian Beberapa Metode Load Balancing

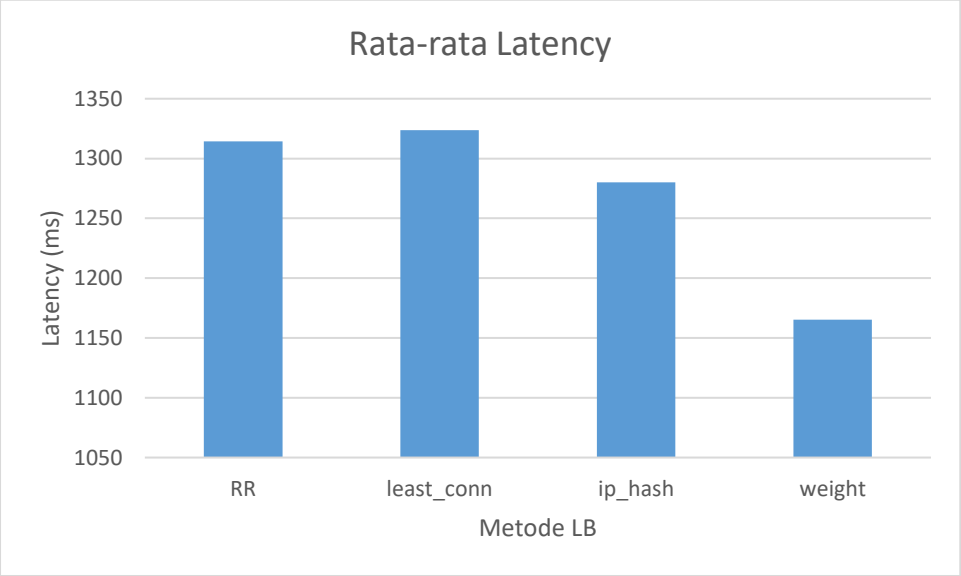
Pada pengujian ini di berikan permintaan http ke server yang menggunakan metode load balancing yang berbeda beda. Permintaan HTTP diberikan sebesar 1000 kali per detik dengan durasi sebesar 300 detik. Server upstream yang digunakan sebanyak empat buah dengan spesifikasinya yang sama.

Tabel 4.5 Hasil Pengujian Permintaan HTTP menggunakan *Load Balancer* dengan metode *load balancing* Berbeda.

Metode LB	Sukses	Tidak Sukses	Prosentase Sukses (%)	Rata-rata Latency (ms)
RR	176456	167	99,905	1314
least_conn	169948	643	99,623	1324
ip_hash	190651	1260	99,343	1280
weight	213173	217	99,898	1165



Gambar 4.9 Grafik Prosentase Sukses dari Jawaban Server ke Klien menggunakan Load Balancer dengan Metode Load Balancing yang berbeda.



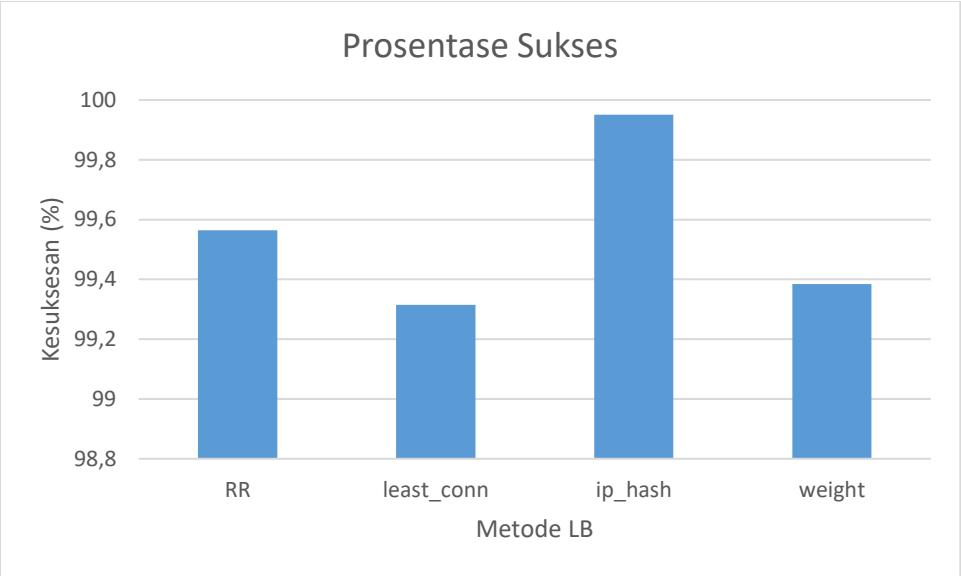
Gambar 4.10 Grafik Latency dari Jawaban Server ke Klien menggunakan Load Balancer dengan Metode Load Balancing yang berbeda.

Dilihat dari latency yang didapatkan hasil terbaik terjadi pada saat menggunakan metode Weight, sedangkan hasil terburuk terjadi saat menggunakan metode least_conn.

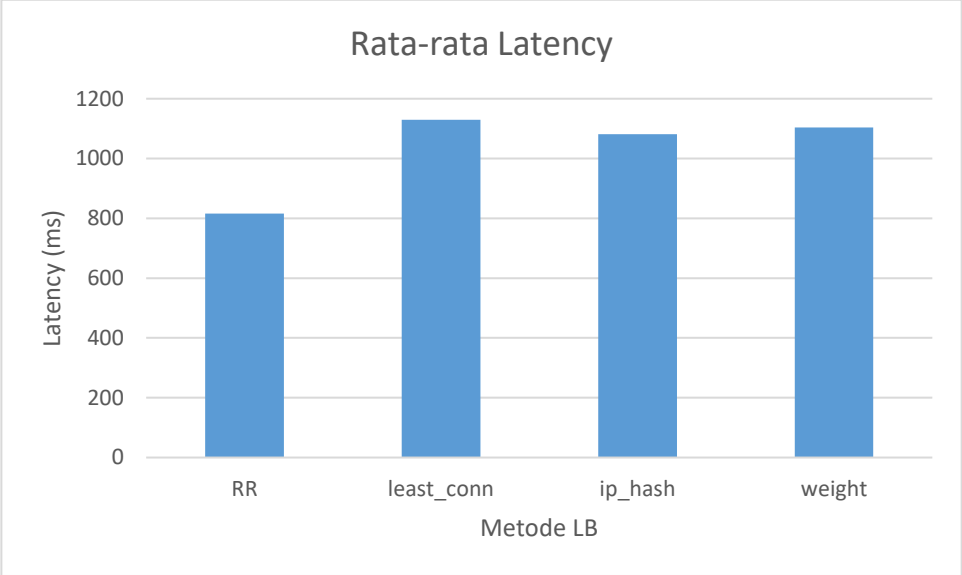
4.1.6 Pengujian Load Balancer dengan Upstream Server yang Berbeda Spesifikasi

Tabel 4.6 Hasil Pengujian Permintaan HTTP metode *load balancing* dengan Server Upstream yang Berbeda.

Metode LB	Sukses	Tidak Sukses	Prosentase Sukses (%)	Rata-rata Latency (ms)
RR	305452	1337	99,564	816
least_conn	231971	1600	99,315	1129
ip_hash	227337	112	99,951	1082
weight	219830	1362	99,384	1104



Gambar 4.11 Grafik Kesuksesan dari Jawaban Server ke Klien dengan Metode Load Balancing yang berbeda dan sepesifikasi server upstream yang berbeda.

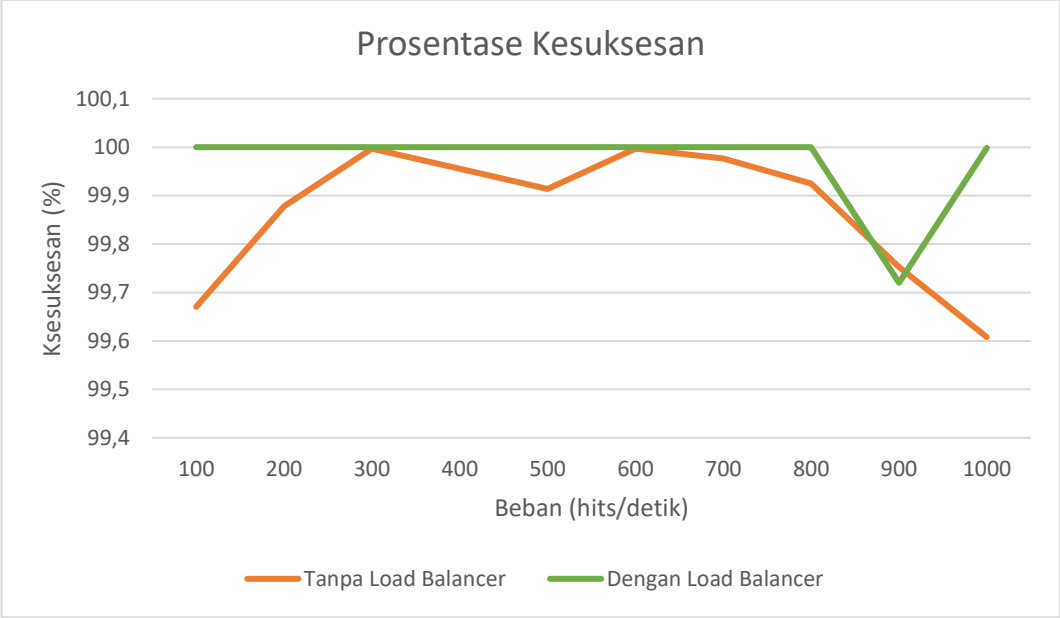


Gambar 4.12 Grafik Latency dari Jawaban Server ke klien dengan Metode Load Balancing yang Berbeda dan Spesifikasi Server Upstream yang berbeda.

4.1.7 Perbandingan Web Server tanpa Load Balancer dan dengan Load Balancer

Tabel 4.7 Perbandingan Kesuksesan Jawaban webserver tanpa load Balancer dan Menggunakan Load Balancer

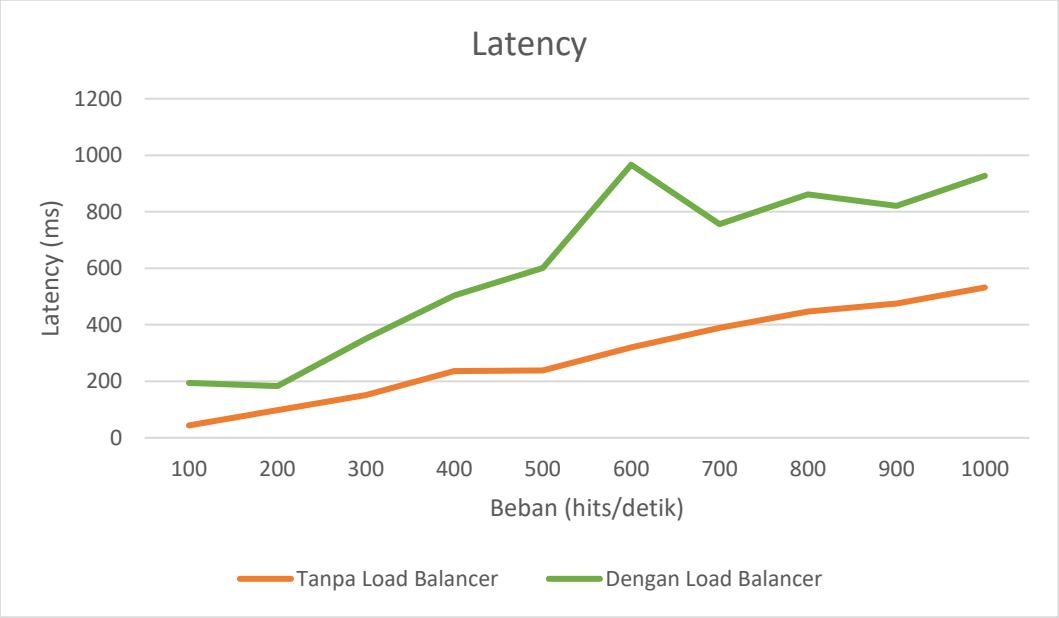
Hits (Permintaan/dtk)	Prosentase Sukses tanpa Load Balancer	Prosentase Sukses dengan Load Balancer
100	99,670	100
200	99,878	100
300	99,997	100
400	99,955	100
500	99,914	100
600	99,997	100
700	99,976	100
800	99,925	100
900	99,753	99,720
1000	99,608	99,998



Gambar 4.13 Grafik Perbandingan Kesuksesan Jawaban webserver tanpa load Balancer dan Menggunakan Load Balancer

Tabel 4.8 Perbandingan Latency Jawaban webserver tanpa load Balancer dan Menggunakan Load Balancer

Hits (Permintaan/dtk)	Latency tanpa Load Balancer (ms)	Latency dengan Load Balancer (ms)
100	44	195
200	98	183
300	152	351
400	236	504
500	239	601
600	320	966
700	389	756
800	446	862
900	476	821
1000	532	927



Gambar 4.14 Grafik Perbandingan Latency Jawaban webserver tanpa load Balancer dan Menggunakan Load Balancer.

4.2 Analisis Data Hasil Pengujian

Pada pengujian Permintaan HTTP tanpa Load Balancer didapatkan tabel 4.1. Dilihat dari hasil tersebut diketahui bahwa hasil terbaik didapatkan pada saat permintaan dilakukan sebesar 300 per detik dan hasil terburuk didapatkan saat permintaan sebesar 1000 per detik. Dilihat dari *latency* yang didapatkan hasil terbaik didapatkan saat permintaan dilakukan dengan 100 per detik dan terburuk saat permintaan 1000 per detik. Dari grafik 4.2 didapatkan tren yang semakin naik jika permintaan per detiknya semakin besar. Hal tersebut terjadi karena semakin banyak permintaan maka semakin besar pula proses yang dibutuhkan server untuk menjawab permintaan tersebut.

Pada pengujian Permintaan HTTP tanpa Load Balancer dengan memori yang berbeda, jika dilihat dari gambar 4.3 dapat diketahui kesuksesan server untuk menjawab permintaan nilainya naik turun meskipun memory pada server ditambahkan. Dari nilai *latency* yang didapatkan juga terjadi naik turun meskipun memori juga ditambahkan.

Dari hasil pengujian permintaan HTTP dengan menggunakan Load balancer dapat diketahui saat permintaan per detik kurang dari 800 maka server dapt menjawab semua permintaan dengan sukses. Tetapi saat permintaan menjadi 900

per detik atau yang lebih besar, terjadi sebagaian permintaan tidak dapat dijawab oleh server. Dilihat dari latency-nya juga didapatkan tren yang semakin besar jumlah permintaan perdetiknya maka latency juga semakin besar.

Dilihat dari hasil latency yang didapatkan pada pengujian permintaan http menguunkan load balancer dengan jumlah server yang berbeda didapatkan latency yang hampir sama meskipun menggunakan jumlah upstream server yang berbeda.

Pada pengujian beberapa metode load balancer didapatkan prosentase kesuksesan dari server maka didapatkan hasil terbaik terjadi saat menggunakan metode load balancing berupa Round-Robin(RR) sedangkan hasil terburuk terjadi saat menggunakan Metode ip_hash. Hal tersebut terjadi karena Metode Round Robin membagi rata semua request yang didapatkan ke beberapa server uptream. Sedangkan pada metode ip_hash semua request tertuju pada satu server upstream, sesuai dengan alamat IP klien.

Sedangkan pada pengujian beberapa metode load balancer yang menggunakan server upstream yang berbeda didapatkan metode yang terbaik adalah IP hash. Metode tersebut mempunyai prosentase kesuksesan yang lebih besar daripada yang lain. Prosentase tersebut mempunyai nilai yang lebih tinggi karena pada metode IP hash load balancer akan merekam ip client dan meneruskan ke salah satu server upstream sesuai dengan nilai hash-nya. Dan kebetulan server upstream tersebut mempunyai sepesifikasi server yang lebih tinggi daripada server upstream lainnya.

Pada perbandingan web server tanpa menggunakan load balancer dan dengan menggunakan load balancer dapat diketahui prosentase kesuksesan dengan menggunakan load balancer lebih baik di setiap permintaan hit perdetik kecuali pada saat 900 hit per detik. Untuk latency disetiap permintaan hit per detik selalu lebih baik tanpa menggunakan load balancer. Hal itu disebabkan karena web server dengan menggunakan Load Balancer harus melewati Load Balancer terlbeih dahulu sebelum diterima oleh upstream server yang memberikan jawaban.

Halaman ini sengaja dikosongkan

BAB 5
KESIMPULAN DAN SARAN

Pada bab ini akan diuraikan beberapa kesimpulan dan saran yang dapat diambil dari pembahasan sebelumnya dan saran mengenai masalah yang bisa dibahas sebagai kelanjutan dari penelitian ini.

5.1 Kesimpulan

Kesimpulan yang didapatkan dari pengujian yang sudah dilakukan adalah sebagai berikut:

1. Semakin besar permintaan HTTP ke server maka semakin besar pula Latency dari jawaban ke klien seperti yang ditunjukkan pada pengujian HTTP tanpa Load Balancer. Hal tersebut dikarenakan semakin banyak antrian di server web untuk diproses.
2. Metode penyeimbang terbaik pada server upstream yang mempunyai spesifikasi yang sama adalah dengan menggunakan metode Round-Robin. Hal tersebut dapat dilihat pada pengujian beberapa metode penyeimbang beban dengan prosentase jawaban tertinggi dari metode yang lain yaitu sebesar 99,905 persen. Hal tersebut terjadi karena semua server mempunyai kemampuan dan konten yang sama, sehingga apabila diberikan beban yang merata, maka semua server akan mempunyai beban yang sama. Tetapi latency terbaik didapatkan dengan metode weight.
3. Metode penyeimbang terbaik pada server upstream yang mempunyai satu buah server memori server yang berbeda adalah ip hash. Hal tersebut dapat dilihat pada pengujian bahwa didapatkan prosentase jawaban tertinggi dari pada metode yang lain yaitu sebesar 99,95 persen. Hal tersebut terjadi karena Load Balancer menghitung nilai hash dari klien dan didapatkan nilai yang mengarahkan permintaan ke salah satu server, dan secara kebetulan server yang ditunjuk mempunyai server berspesifikasi lebih tinggi dari pada server yang lain.

5.2 Saran

Berdasarkan hasil perancangan sistem dan pengujian yang telah dilakukan, dapat diberikan beberapa saran untuk pengembangan *HTTP load balancer* adalah sebagai berikut:

1. Menggunakan aplikasi load balancer yang lain dan membandingkan hasilnya. Seperti Load balancer berupa perangkat keras atau berupa perangkat lunak yang lain.
2. Digunakan beberapa aplikasi web yang bermacam macam yang membutuhkan komputasi yang tinggi maupun yang rendah.

DAFTAR PUSTAKA

[1] X. Chi, B. Liu, Q. Niu dan Q. Wu, Web Load Balance and Cache Optimization Design based Nginx Under Highconcurrency Environment, 2012.

[2] Q. Zhang dan A. Riska, Workload-Aware Load Balancing for Clustered Web Servers, 2005.

[3] H. Bryhni, E. Klovning dan O. Kure, A Comparison of Load Balancing Techniques for Scalable Web Server, 2000.

[4] M.-T. Thai, Y.-D. Lin dan Y.-C. Lai, "A joint network and server load balancing algorithm for chaining virtualized network functions," 2016.

[5] M.-T. Thai, Y.-D. Lin, P.-C. Lin dan Y.-C. Lai, "Hash-based load balanced traffic steering on softswitches for chaining virtualized network functions," 2017.

[6] J. Lenhardt, K. Chen dan W. Schiffmann, "Energy-Efficient Web Server Load Balancing," *IEEE Systems Journal,* 2017.

[7] S. Di dan W. Zheng, "Request dispatching algorithms for web server clusters based on load balancing," *Tsinghua Science and Technology,* 1999.

[8] G. Pacifici, M. Spreitzer, A. Tantawi dan A. Youssef, "Performance management for cluster-based web services," *IEEE Journal on Selected Areas in Communications,* 2005 .

[9] "The Apache HTTP Server Project," [Online]. Available: https://httpd.apache.org/.

[10] "The Reliable, High Performance TCP/HTTP Load Balancer," [Online]. Available: http://www.haproxy.org/.

[11] "NGINX | High Performance Load Balancer, Web Server, and Reverse Proxy," Igor Sysoev, [Online]. Available: https://www.nginx.com/.

[12] "Apache JMeter," [Online]. Available: https://jmeter.apache.org/.

[13] M. Anicas, "An Introduction to HAProxy and Load Balancing Concepts," 2014. [Online]. Available: https://www.digitalocean.com/community/tutorials/an-introduction-to-haproxy-and-load-balancing-concepts.

[14] T. Mauro, “Choosing an NGINX Plus Load-Balancing Technique,” 2015. [Online]. Available: https://www.nginx.com/blog/choosing-nginx-plus-load-balancing-techniques/.

[15] “Cooler Master N400,” [Online]. Available: http://global.pchome.com.tw/prod/DRAE0O-A77722399. [Diakses 2018].

[16] “HP SERVER HP PROLIANT DL 380P GEN 8,” 2018. [Online]. Available: http://jnp.ae/product/hp-server-hp-proliant-dl-380p-gen-8-32-cores-total/.

[17] “PowerEdge M830 Blade Server,” [Online]. Available: http://www.dell.com/id/business/p/poweredge-m830/pd. [Diakses 2018].

[18] “Cisco Catalyst 3560G-24TS Switch,” 2018. [Online]. Available: https://www.cisco.com/c/en/us/support/switches/catalyst-3560g-24ts-switch/model.html.

[19] W. Tarreau, “HAProxy Configuration Manual,” 2018. [Online]. Available: http://cbonte.github.io/haproxy-dconv/1.8/configuration.html#balance.

LAMPIRAN

Konfigurasi Load Balancer

IP Hash:

```
upstream antates_http {
        ip_hash;
        server 10.199.14.191:80;
        server 10.199.14.192:80;
        server 10.199.14.193:80;
        server 10.199.14.194:80;
}

upstream antates_https {
        ip_hash;
        server 10.199.14.191:443;
        server 10.199.14.192:443;
        server 10.199.14.193:443;
        server 10.199.14.194:443;
}

server {
        listen 80 default_server;
        server_name antates.its.ac.id;

           index index.php index.html index.htm;
           location / {
             proxy_pass http://antates_http;
             proxy_set_header Host $host;
             proxy_set_header X-Real-IP $remote_addr;
             proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
             proxy_set_header X-Forwarded-Proto $scheme;
           }
           access_log /var/log/nginx/antates.its.ac.id.access.log;
           error_log /var/log/nginx/antates.its.ac.id.error.log;
}

server {
        listen 443 default_server;
        server_name antates.its.ac.id www.antates.its.ac.id;

        ssl on;
        ssl_certificate         /etc/ssl/its/bundle_star_its_ac_id.crt;
        ssl_certificate_key     /etc/ssl/its/star_its_ac_id.key;
        ssl_protocols           TLSv1 TLSv1.1 TLSv1.2;
        ssl_ciphers             "EECDH+AESGCM:EDH+AESGCM:ECDHE-RSA-
AES128-GCM-SHA256:AES256+EECDH:DHE-RSA-AES128-GCM-
SHA256:AES256+EDH:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-
```

```
AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-
AES128-SHA256:ECDHE-RSA-AES256-SHA:ECDHE-RSA-AES128-
SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES128-SHA256:DHE-RSA-
AES256-SHA:DHE-RSA-AES128-SHA:ECDHE-RSA-DES-CBC3-
SHA:EDH-RSA-DES-CBC3-SHA:AES256-GCM-SHA384:AES128-GCM-
SHA256:AES256-SHA256:AES128-SHA256:AES256-SHA:AES128-
SHA:DES-CBC3-
SHA:HIGH:!aNULL:!eNULL:!EXPORT:!DES:!MD5:!PSK:!RC4";
        ssl_prefer_server_ciphers on;
        keepalive_timeout       60;
        ssl_session_cache       shared:SSL:10m;
        ssl_session_timeout     10m;
#         ssl_dhparam             /etc/ssl/certs/dhparam.pem;

        location / {
                proxy_pass https://antates_https;
                proxy_set_header Host $host;
                proxy_set_header X-Real-IP $remote_addr;
                proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
                proxy_set_header X-Forwarded-Proto $scheme;
        }

        access_log /var/log/nginx/antates.its.ac.id.access.log;
        error_log /var/log/nginx/antates.its.ac.id.error.log;

           location /nginx_status {
                        # Turn on nginx stats
                        stub_status on;

                        # I do not need logs for stats
                        access_log   off;

                        # Security: Only allow access from 192.168.1.100 IP #
                        allow 202.46.129.4;
                        allow 202.46.129.16;
                        allow 103.94.188.4;

                        # Send rest of the world to /dev/null #
                        deny all;
           }
}
```

Least Connection:

```
upstream antates_http {
        least_conn;
        server 10.199.14.191:80;
        server 10.199.14.192:80;
        server 10.199.14.193:80;
        server 10.199.14.194:80;
        server 10.199.14.195:80;
}

upstream antates_https {
        least_conn;
        server 10.199.14.191:443;
        server 10.199.14.192:443;
        server 10.199.14.193:443;
        server 10.199.14.194:443;
}

server {
        listen 80 default_server;
        server_name antates.its.ac.id;

           index index.php index.html index.htm;
           location / {
             proxy_pass http://antates_http;
             proxy_set_header Host $host;
             proxy_set_header X-Real-IP $remote_addr;
             proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
             proxy_set_header X-Forwarded-Proto $scheme;
           }
           access_log /var/log/nginx/antates.its.ac.id.access.log;
           error_log /var/log/nginx/antates.its.ac.id.error.log;
}

server {
        listen 443 default_server;
        server_name antates.its.ac.id www.antates.its.ac.id;

        ssl on;
        ssl_certificate         /etc/ssl/its/bundle_star_its_ac_id.crt;
        ssl_certificate_key     /etc/ssl/its/star_its_ac_id.key;
        ssl_protocols           TLSv1 TLSv1.1 TLSv1.2;
        ssl_ciphers             "EECDH+AESGCM:EDH+AESGCM:ECDHE-RSA-
AES128-GCM-SHA256:AES256+EECDH:DHE-RSA-AES128-GCM-
SHA256:AES256+EDH:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-
AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-
```

```
AES128-SHA256:ECDHE-RSA-AES256-SHA:ECDHE-RSA-AES128-
SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES128-SHA256:DHE-RSA-
AES256-SHA:DHE-RSA-AES128-SHA:ECDHE-RSA-DES-CBC3-
SHA:EDH-RSA-DES-CBC3-SHA:AES256-GCM-SHA384:AES128-GCM-
SHA256:AES256-SHA256:AES128-SHA256:AES256-SHA:AES128-
SHA:DES-CBC3-
SHA:HIGH:!aNULL:!eNULL:!EXPORT:!DES:!MD5:!PSK:!RC4";
        ssl_prefer_server_ciphers on;
        keepalive_timeout       60;
        ssl_session_cache       shared:SSL:10m;
        ssl_session_timeout     10m;
#         ssl_dhparam             /etc/ssl/certs/dhparam.pem;

        location / {
                proxy_pass https://antates_https;
                proxy_set_header Host $host;
                proxy_set_header X-Real-IP $remote_addr;
                proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
                proxy_set_header X-Forwarded-Proto $scheme;
        }

        access_log /var/log/nginx/antates.its.ac.id.access.log;
        error_log /var/log/nginx/antates.its.ac.id.error.log;

           location /nginx_status {
                        # Turn on nginx stats
                        stub_status on;

                        # I do not need logs for stats
                        access_log   off;

                        # Security: Only allow access from 192.168.1.100 IP #
                        allow 202.46.129.4;
                        allow 202.46.129.16;
                        allow 103.94.188.4;

                        # Send rest of the world to /dev/null #
                        deny all;
                }
}
```

Round Robin:

```
upstream antates_http {
        server 10.199.14.191:80;
        server 10.199.14.192:80;
        server 10.199.14.193:80;
        server 10.199.14.194:80;
}

upstream antates_https {
        server 10.199.14.191:443;
        server 10.199.14.192:443;
        server 10.199.14.193:443;
        server 10.199.14.194:443;
}

server {
        listen 80 default_server;
        server_name antates.its.ac.id;

           index index.php index.html index.htm;
           location / {
             proxy_pass http://antates_http;
             proxy_set_header Host $host;
             proxy_set_header X-Real-IP $remote_addr;
             proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
             proxy_set_header X-Forwarded-Proto $scheme;
           }
           access_log /var/log/nginx/antates.its.ac.id.access.log;
           error_log /var/log/nginx/antates.its.ac.id.error.log;
}

server {
        listen 443 default_server;
        server_name antates.its.ac.id www.antates.its.ac.id;

        ssl on;
        ssl_certificate         /etc/ssl/its/bundle_star_its_ac_id.crt;
        ssl_certificate_key     /etc/ssl/its/star_its_ac_id.key;
        ssl_protocols           TLSv1 TLSv1.1 TLSv1.2;
        ssl_ciphers             "EECDH+AESGCM:EDH+AESGCM:ECDHE-RSA-
AES128-GCM-SHA256:AES256+EECDH:DHE-RSA-AES128-GCM-
SHA256:AES256+EDH:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-
AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-
AES128-SHA256:ECDHE-RSA-AES256-SHA:ECDHE-RSA-AES128-
SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES128-SHA256:DHE-RSA-
AES256-SHA:DHE-RSA-AES128-SHA:ECDHE-RSA-DES-CBC3-
```

```
SHA:EDH-RSA-DES-CBC3-SHA:AES256-GCM-SHA384:AES128-GCM-
SHA256:AES256-SHA256:AES128-SHA256:AES256-SHA:AES128-
SHA:DES-CBC3-
SHA:HIGH:!aNULL:!eNULL:!EXPORT:!DES:!MD5:!PSK:!RC4";
        ssl_prefer_server_ciphers on;
        keepalive_timeout       60;
        ssl_session_cache       shared:SSL:10m;
        ssl_session_timeout     10m;
#         ssl_dhparam             /etc/ssl/certs/dhparam.pem;

        location / {
                proxy_pass https://antates_https;
                proxy_set_header Host $host;
                proxy_set_header X-Real-IP $remote_addr;
                proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
                proxy_set_header X-Forwarded-Proto $scheme;
        }

        access_log /var/log/nginx/antates.its.ac.id.access.log;
        error_log /var/log/nginx/antates.its.ac.id.error.log;

           location /nginx_status {
                        # Turn on nginx stats
                        stub_status on;

                        # I do not need logs for stats
                        access_log   off;

                        # Security: Only allow access from 192.168.1.100 IP #
                        allow 202.46.129.4;
                        allow 202.46.129.16;
                        allow 103.94.188.4;

                        # Send rest of the world to /dev/null #
                        deny all;
           }
}
```

Weight:

```
upstream antates_http {
        server 10.199.14.191:80 weight=2;
        server 10.199.14.192:80;
        server 10.199.14.193:80;
        server 10.199.14.194:80;
}

upstream antates_https {
        server 10.199.14.191:443 weight=2;
        server 10.199.14.192:443;
        server 10.199.14.193:443;
        server 10.199.14.194:443;
}

server {
        listen 80 default_server;
        server_name antates.its.ac.id;

           index index.php index.html index.htm;
           location / {
             proxy_pass http://antates_http;
             proxy_set_header Host $host;
             proxy_set_header X-Real-IP $remote_addr;
             proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
             proxy_set_header X-Forwarded-Proto $scheme;
           }
           access_log /var/log/nginx/antates.its.ac.id.access.log;
           error_log /var/log/nginx/antates.its.ac.id.error.log;
}

server {
        listen 443 default_server;
        server_name antates.its.ac.id www.antates.its.ac.id;

        ssl on;
        ssl_certificate         /etc/ssl/its/bundle_star_its_ac_id.crt;
        ssl_certificate_key     /etc/ssl/its/star_its_ac_id.key;
        ssl_protocols           TLSv1 TLSv1.1 TLSv1.2;
        ssl_ciphers             "EECDH+AESGCM:EDH+AESGCM:ECDHE-RSA-
AES128-GCM-SHA256:AES256+EECDH:DHE-RSA-AES128-GCM-
SHA256:AES256+EDH:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-
AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-
AES128-SHA256:ECDHE-RSA-AES256-SHA:ECDHE-RSA-AES128-
SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES128-SHA256:DHE-RSA-
AES256-SHA:DHE-RSA-AES128-SHA:ECDHE-RSA-DES-CBC3-
```

```
SHA:EDH-RSA-DES-CBC3-SHA:AES256-GCM-SHA384:AES128-GCM-
SHA256:AES256-SHA256:AES128-SHA256:AES256-SHA:AES128-
SHA:DES-CBC3-
SHA:HIGH:!aNULL:!eNULL:!EXPORT:!DES:!MD5:!PSK:!RC4";
        ssl_prefer_server_ciphers on;
        keepalive_timeout       60;
        ssl_session_cache       shared:SSL:10m;
        ssl_session_timeout     10m;
#         ssl_dhparam             /etc/ssl/certs/dhparam.pem;

        location / {
                proxy_pass https://antates_https;
                proxy_set_header Host $host;
                proxy_set_header X-Real-IP $remote_addr;
                proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
                proxy_set_header X-Forwarded-Proto $scheme;
        }

        access_log /var/log/nginx/antates.its.ac.id.access.log;
        error_log /var/log/nginx/antates.its.ac.id.error.log;

           location /nginx_status {
                        # Turn on nginx stats
                        stub_status on;

                        # I do not need logs for stats
                        access_log   off;

                        # Security: Only allow access from 192.168.1.100 IP #
                        allow 202.46.129.4;
                        allow 202.46.129.16;
                        allow 103.94.188.4;

                        # Send rest of the world to /dev/null #
                        deny all;
           }
}
```

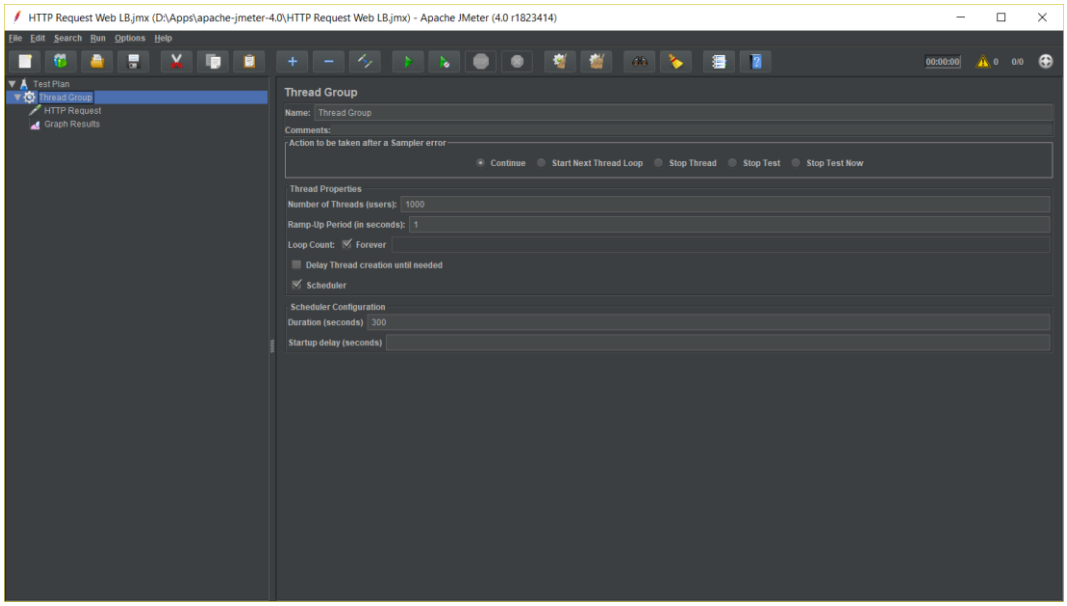
Konfigurasi JMeter pada klien sebagai penguji

```
<?xml version="1.0" encoding="UTF-8"?>
<jmeterTestPlan version="1.2" properties="4.0" jmeter="4.0 r1823414">
  <hashTree>
    <TestPlan guiclass="TestPlanGui" testclass="TestPlan" testname="Test
Plan" enabled="true">
      <stringProp name="TestPlan.comments"></stringProp>
      <boolProp name="TestPlan.functional_mode">false</boolProp>
      <boolProp name="TestPlan.tearDown_on_shutdown">true</boolProp>
      <boolProp name="TestPlan.serialize_threadgroups">false</boolProp>
      <elementProp name="TestPlan.user_defined_variables"
elementType="Arguments" guiclass="ArgumentsPanel" testclass="Arguments"
testname="User Defined Variables" enabled="true">
        <collectionProp name="Arguments.arguments"/>
      </elementProp>
      <stringProp name="TestPlan.user_define_classpath"></stringProp>
    </TestPlan>
    <hashTree>
      <ThreadGroup guiclass="ThreadGroupGui" testclass="ThreadGroup"
testname="Thread Group" enabled="true">
        <stringProp
name="ThreadGroup.on_sample_error">continue</stringProp>
        <elementProp name="ThreadGroup.main_controller"
elementType="LoopController" guiclass="LoopControlPanel"
testclass="LoopController" testname="Loop Controller" enabled="true">
          <boolProp name="LoopController.continue_forever">false</boolProp>
          <intProp name="LoopController.loops">-1</intProp>
        </elementProp>
        <stringProp name="ThreadGroup.num_threads">1000</stringProp>
        <stringProp name="ThreadGroup.ramp_time">1</stringProp>
        <boolProp name="ThreadGroup.scheduler">true</boolProp>
        <stringProp name="ThreadGroup.duration">300</stringProp>
        <stringProp name="ThreadGroup.delay"></stringProp>
      </ThreadGroup>
      <hashTree>
        <HTTPSamplerProxy guiclass="HttpTestSampleGui"
testclass="HTTPSamplerProxy" testname="HTTP Request" enabled="true">
          <elementProp name="HTTPsampler.Arguments"
elementType="Arguments" guiclass="HTTPArgumentsPanel"
testclass="Arguments" testname="User Defined Variables" enabled="true">
            <collectionProp name="Arguments.arguments"/>
          </elementProp>
          <stringProp name="HTTPSampler.domain">10.199.13.9</stringProp>
          <stringProp name="HTTPSampler.port">80</stringProp>
          <stringProp name="HTTPSampler.protocol">http</stringProp>
          <stringProp name="HTTPSampler.contentEncoding"></stringProp>
          <stringProp name="HTTPSampler.path"></stringProp>
```

```
        <stringProp name="HTTPSampler.method">GET</stringProp>
        <boolProp name="HTTPSampler.follow_redirects">true</boolProp>
        <boolProp name="HTTPSampler.auto_redirects">false</boolProp>
        <boolProp name="HTTPSampler.use_keepalive">true</boolProp>
        <boolProp
name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
        <stringProp name="HTTPSampler.embedded_url_re"></stringProp>
        <stringProp name="HTTPSampler.connect_timeout"></stringProp>
        <stringProp name="HTTPSampler.response_timeout"></stringProp>
      </HTTPSamplerProxy>
      <hashTree/>
      <ResultCollector guiclass="GraphVisualizer" testclass="ResultCollector"
testname="Graph Results" enabled="true">
        <boolProp name="ResultCollector.error_logging">false</boolProp>
        <objProp>
          <name>saveConfig</name>
          <value class="SampleSaveConfiguration">
            <time>true</time>
            <latency>true</latency>
            <timestamp>true</timestamp>
            <success>true</success>
            <label>true</label>
            <code>true</code>
            <message>true</message>
            <threadName>true</threadName>
            <dataType>true</dataType>
            <encoding>false</encoding>
            <assertions>true</assertions>
            <subresults>true</subresults>
            <responseData>false</responseData>
            <samplerData>false</samplerData>
            <xml>false</xml>
            <fieldNames>true</fieldNames>
            <responseHeaders>false</responseHeaders>
            <requestHeaders>false</requestHeaders>
            <responseDataOnError>false</responseDataOnError>

<saveAssertionResultsFailureMessage>true</saveAssertionResultsFailureMess
age>
            <assertionsResultsToSave>0</assertionsResultsToSave>
            <bytes>true</bytes>
            <sentBytes>true</sentBytes>
            <threadCounts>true</threadCounts>
            <idleTime>true</idleTime>
            <connectTime>true</connectTime>
          </value>
        </objProp>
```

```
        <stringProp name="filename">D:\Anta\Google
Drive\Kuliah\Thesis\Data\Web LB\LB_rr_5-1beda.csv</stringProp>
      </ResultCollector>
      <hashTree/>
    </hashTree>
  </hashTree>
 </hashTree>
</jmeterTestPlan>
```



Hasil generate file csv pada JMeter

```
timeStamp,elapsed,label,responseCode,responseMessage,threadName,dataType
,success,failureMessage,bytes,sentBytes,grpThreads,allThreads,Latency,IdleTi
me,Connect
1519692750138,47,HTTP Request,200,OK,Thread Group 1-
2,text,true,,11016,117,5,5,47,0,6
1519692750139,49,HTTP Request,200,OK,Thread Group 1-
1,text,true,,11016,117,5,5,49,0,43
1519692750170,20,HTTP Request,200,OK,Thread Group 1-
3,text,true,,11016,117,5,5,20,0,14
1519692750170,21,HTTP Request,200,OK,Thread Group 1-
4,text,true,,11016,117,5,5,21,0,15
1519692750186,6,HTTP Request,200,OK,Thread Group 1-
2,text,true,,11015,117,5,5,6,0,0
1519692750188,5,HTTP Request,200,OK,Thread Group 1-
1,text,true,,11015,117,5,5,5,0,0
1519692750185,8,HTTP Request,200,OK,Thread Group 1-
5,text,true,,11016,117,5,5,8,0,4
1519692750190,5,HTTP Request,200,OK,Thread Group 1-
3,text,true,,11015,117,5,5,4,0,0
```

```
1519692750191,5,HTTP Request,200,OK,Thread Group 1-
4,text,true,,11015,117,5,5,5,0,0
1519692750193,4,HTTP Request,200,OK,Thread Group 1-
1,text,true,,11015,117,5,5,4,0,0
1519692750192,6,HTTP Request,200,OK,Thread Group 1-
2,text,true,,11015,117,5,5,6,0,0
1519692750193,6,HTTP Request,200,OK,Thread Group 1-
5,text,true,,11015,117,5,5,6,0,0
1519692750195,5,HTTP Request,200,OK,Thread Group 1-
3,text,true,,11015,117,5,5,5,0,0
1519692750196,6,HTTP Request,200,OK,Thread Group 1-
4,text,true,,11015,117,7,7,5,0,0
1519692750198,5,HTTP Request,200,OK,Thread Group 1-
1,text,true,,11015,117,7,7,4,0,0
1519692750199,5,HTTP Request,200,OK,Thread Group 1-
5,text,true,,11015,117,7,7,4,0,0
1519692750198,6,HTTP Request,200,OK,Thread Group 1-
2,text,true,,11015,117,7,7,6,0,0
1519692750200,5,HTTP Request,200,OK,Thread Group 1-
3,text,true,,11015,117,7,7,5,0,0
1519692750202,4,HTTP Request,200,OK,Thread Group 1-
4,text,true,,11015,117,7,7,4,0,0
1519692750203,4,HTTP Request,200,OK,Thread Group 1-
1,text,true,,11015,117,7,7,4,0,0
1519692750204,4,HTTP Request,200,OK,Thread Group 1-
5,text,true,,11015,117,7,7,4,0,0
1519692750204,5,HTTP Request,200,OK,Thread Group 1-
2,text,true,,11015,117,7,7,4,0,0
1519692750205,6,HTTP Request,200,OK,Thread Group 1-
3,text,true,,11015,117,7,7,6,0,0
1519692750201,10,HTTP Request,200,OK,Thread Group 1-
6,text,true,,11016,117,7,7,10,0,4
1519692750201,10,HTTP Request,200,OK,Thread Group 1-
7,text,true,,11016,117,7,7,10,0,4
1519692750206,5,HTTP Request,200,OK,Thread Group 1-
4,text,true,,11015,117,7,7,5,0,0
1519692750208,6,HTTP Request,200,OK,Thread Group 1-
5,text,true,,11015,117,7,7,6,0,0
1519692750207,7,HTTP Request,200,OK,Thread Group 1-
1,text,true,,11015,117,7,7,7,0,0
1519692750209,5,HTTP Request,200,OK,Thread Group 1-
2,text,true,,11015,117,7,7,5,0,0
1519692750211,6,HTTP Request,200,OK,Thread Group 1-
6,text,true,,11015,117,8,8,6,0,0
1519692750211,6,HTTP Request,200,OK,Thread Group 1-
3,text,true,,11015,117,8,8,6,0,0
```

```
1519692750211,6,HTTP Request,200,OK,Thread Group 1-
7,text,true,,11015,117,8,8,6,0,0
1519692750211,6,HTTP Request,200,OK,Thread Group 1-
4,text,true,,11015,117,8,8,6,0,0
1519692750214,6,HTTP Request,200,OK,Thread Group 1-
1,text,true,,11015,117,8,8,6,0,0
1519692750214,6,HTTP Request,200,OK,Thread Group 1-
5,text,true,,11015,117,8,8,6,0,0
1519692750214,6,HTTP Request,200,OK,Thread Group 1-
2,text,true,,11015,117,8,8,5,0,0
1519692750217,6,HTTP Request,200,OK,Thread Group 1-
3,text,true,,11015,117,8,8,6,0,0
1519692750217,6,HTTP Request,200,OK,Thread Group 1-
7,text,true,,11015,117,8,8,6,0,0
1519692750217,6,HTTP Request,200,OK,Thread Group 1-
6,text,true,,11015,117,8,8,6,0,0
1519692750217,6,HTTP Request,200,OK,Thread Group 1-
4,text,true,,11015,117,8,8,6,0,0
1519692750217,9,HTTP Request,200,OK,Thread Group 1-
8,text,true,,11016,117,8,8,9,0,3
1519692750220,6,HTTP Request,200,OK,Thread Group 1-
1,text,true,,11015,117,8,8,6,0,0
1519692750220,6,HTTP Request,200,OK,Thread Group 1-
2,text,true,,11015,117,8,8,6,0,0
1519692750220,6,HTTP Request,200,OK,Thread Group 1-
5,text,true,,11015,117,8,8,6,0,0
1519692750223,6,HTTP Request,200,OK,Thread Group 1-
3,text,true,,11015,117,8,8,6,0,0
1519692750223,6,HTTP Request,200,OK,Thread Group 1-
7,text,true,,11015,117,8,8,6,0,0
1519692750223,6,HTTP Request,200,OK,Thread Group 1-
4,text,true,,11015,117,8,8,6,0,0
1519692750223,6,HTTP Request,200,OK,Thread Group 1-
6,text,true,,11015,117,8,8,6,0,0
1519692750226,7,HTTP Request,200,OK,Thread Group 1-
1,text,true,,11015,117,10,10,7,0,0
1519692750226,7,HTTP Request,200,OK,Thread Group 1-
8,text,true,,11015,117,10,10,7,0,0
1519692750226,7,HTTP Request,200,OK,Thread Group 1-
2,text,true,,11015,117,10,10,7,0,0
1519692750226,7,HTTP Request,200,OK,Thread Group 1-
5,text,true,,11015,117,10,10,7,0,0
1519692750229,7,HTTP Request,200,OK,Thread Group 1-
7,text,true,,11015,117,10,10,5,0,0
1519692750229,7,HTTP Request,200,OK,Thread Group 1-
4,text,true,,11015,117,10,10,7,0,0
```

```
1519692750229,7,HTTP Request,200,OK,Thread Group 1-
6,text,true,,11015,117,10,10,7,0,0
1519692750229,7,HTTP Request,200,OK,Thread Group 1-
3,text,true,,11015,117,10,10,7,0,0
1519692750233,7,HTTP Request,200,OK,Thread Group 1-
5,text,true,,11015,117,10,10,7,0,0
1519692750233,7,HTTP Request,200,OK,Thread Group 1-
2,text,true,,11015,117,10,10,7,0,0
.....
```

Gambar Server yang dipakai

Gambar storage yang dipakai