Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer e-ISSN: 2548-964X
Vol. 3, No. 1, Januari 2019, hlm. 109-118 http://j-ptiik.ub.ac.id

Implementasi *Load Balancing Web* Server Dengan Algoritme *Weighted Least Connection* Pada *Software Defined Network*

Hafizhul Karim[1], Rakhmadhany Primananda[2], Widhi Yahya[3]

Program Studi Teknik Informatika, Fakultas Ilmu Komputer, Universitas Brawijaya
Email: [1]hafis.karim5@gmail.com, [2]rakhmadhany@ub.ac.id, [3]widhi.yahya@ub.ac.id

Abstrak

Permasalahan pada *web* server yaitu banyaknya *request* yang dilakukan oleh *user* sehingga beban kerja pada server semakin meningkat, dapat memungkinkan server mengalami *down*. *Load balancing* merupakan teknik untuk mendistribusikan *traffic* yang masuk di antara server yang telah tersedia sehingga *request* dapat ditangani dan memberikan respon yang lebih cepat. Pada penelitian ini menggunakan algoritme *weighted least connection*. Algoritme *weighted least connection* membagi beban server berdasarkan nilai *weight* (bobot). Arsitektur *Software Defined Network* (SDN) bersifat *programmable* dan memberikan fasilitas kepada *Network Administrator* untuk merancang dan mengimplementasikan strategi *load balancing* tanpa keterbatasan vendor. Untuk implementasi pada penelitian ini menggunakan mininet dan controller POX. Parameter yang digunakan yaitu *connection rate, response time, throughput*, dan CPU *usage*. *Tool* yang digunakan untuk pengujian yaitu httperf. Hasil dari pengujian *connection rate, response time* dan *throughput* dengan memberikan *request* dari *client* sebanyak 3000 dengan *rate* 75, 15, dan 300 yaitu nilai *connection rate* tertinggi sebesar 296,28 Conn/s, nilai *response time* terkecil yaitu 2,34 ms dan nilai *throughput* tertinggi sebesar 805,66 KB/s. Dan yang terakhir hasil dari pengujian CPU *usage*, server 1 memiliki nilai 21,6%, server 2 memiliki nilai 24,6%, dan server 3 memiliki nilai 26,3%.

Kata kunci: *Software Defined Network (SDN), Load Balancing, Weighted Least Connection, Bobot*

Abstract

The problem of the web server is many of requests used by the user so that the workload on the server increases quickly resulting in the server down. Load balancing is a technique to distribute incoming traffic between available servers so that requests can be handled and respond faster. This research uses weighted least connection algorithm. The weighted least connection algorithm divides the server load based on the weight value. Software Defined Network Architecture (SDN) is programmable and provides facilities to the Network Administrator to design and implement load balancing strategies without vendor limitations. For implementation use mininet and controller used with POX. Parameters used for testing are connection rate, throughput, and CPU usage. The tool used for testing is httperf. The result of connection rate and throughput testing by giving client request 3000 with the rate of 75, 150, and 300 with the highest connection rate is 296,28 Conn/s, the lowest response time is 2,34 ms and the highest throughput value is 805,66 KB/s. And the last result of testing CPU usage, server 1 has a value of 21.6%, server 2 has a value of 24.6%, and server 3 has a value of 26.3%.

Keywords: *Software Defined Network (SDN), Load Balancing, Weighted Least Connection, Weight*

1. PENDAHULUAN

Permasalahan pada *web* server ketika request yang dilakukan oleh user sangat banyak sehingga beban kerja server semakin meningkat, dapat memungkinkan server mengalami down dalam waktu yang singkat. Hal ini terjadi karena *request* yang ditangani oleh satu *web* server. Oleh karena itu dibutuhkan *web* server dengan pendekatan *multiple* server. Fungsi dari pendekatan *multiple* server yaitu sistem *web* server direplika untuk meningkatkan waktu respon (Singh & Kumar, 2011). Pada *multiple* server dibutuhkan mekanisme pembagian beban supaya kinerja *web* server tetap stabil. Mekanisme pembagian beban menggunakan

teknik *load balancing*.

Load balancing merupakan sebuah teknik untuk mendistribusikan *traffic* yang masuk di antara *server* yang telah tersedia sehingga request/permintaan dapat ditangani dan memberikan respon yang lebih cepat (Rahman, et al., 2014). Dalam teknik *load balancing* terdapat beberapa algoritme antara lain, *round robin, weighted round robin, least connection, weighted least connection*, dan lainnya. Algoritme *weighted least connection* hampir sama dengan *least connection*, hanya saja algoritme ini menentukan bobot kinerja dari setiap server (Red Hat, 2014). Setiap server akan diberikan bobot, semakin tinggi nilai bobot yang diberikan pada server maka server akan menerima *request* lebih banyak dari server yang memiliki nilai bobot rendah. Algoritme *least connection* bekerja melakukan pembagian *request* berdasarkan banyaknya koneksi yang sedang dilayani oleh server, lalu server yang melayani koneksi dengan jumlah paling sedikit akan diberikan beban untuk *request* berikutnya (Supramana & Prismana, 2016). Pada pengujian beban statis, algoritme *weighted least connection* lebih baik dalam mendistribusikan permintaan HTTP, balasan HTTP, dan *throughput*, namun waktu tanggap atau *response time* lebih lambat jika dibandingkan dengan algoritme *least connection* (Angsar, 2014). Hal ini terjadi karena algoritme *weighted least connection* dapat menampung jumlah koneksi yang berbeda, sedangkan algoritme *least connection* dapat menampung jumlah koneksi yang sama (Rahmana, 2017).

Load balancing dengan arsitektur *software defined network* (SDN) bersifat *programmable* dan memberikan fasilitas kepada *Network Administrator* untuk merancang dan mengimplementasikan strategi algoritme *load balancing* tanpa keterbatasan vendor (Sabiya & Singh, 2016). *Software defined network* (SDN) merupakan sebuah arsitektur yang memisahkan *control plane* dan *data plane* pada perangkat jaringan seperti *switch* dan *router*, sedangkan pada jaringan tradisional *control plane* dan *data plane* ini digabung menjadi satu pada perangkat jaringan. *Control plane* berfungsi untuk mengatur logika jaringan dan *data plane* berfungsi untuk mengatur bagaimana paket akan diteruskan pada hop berikutnya (Kreutz, et al., 2014). Selain itu penerapan SDN merupakan sebuah solusi dalam perkembangan dari kebutuhan jaringan yang semakin kompleks yaitu dalam segi penekanan biaya untuk investasi perangkat keras jaringan. (Paul Göransson, 2014). Oleh karena itu arsitektur dengan pendekatan SDN sangat diperlukan.

Pada penelitian sebelumnya terkait tentang *load balancing* dengan pendekatan *software defined network* (SDN) sudah pernah diterapkan salah satunya penelitian yang dilakukan oleh Sabiya dan Japinder Singh yang berjudul "*Weighted Round-Robin Load Balancing Using Software Defined Network*". Penelitian tersebut menggunakan server heterogen. Pada masing-masing server diberikan nilai weight. Server 1 diberikan nilai weight 1, server 2 diberikan nilai weight 3, dan server 3 diberikan nilai weight 6. Hasil yang didapatkan yaitu algoritme *weighted round robin* bekerja dengan cukup baik dalam pengalokasian beban atau *request client* yang hampir merata pada setiap server (Sabiya & Singh, 2016). Berdasarkan dari penelitian sebelumnya yang menjadi pembeda dalam penelitian ini yaitu algoritme *load balancing* yang digunakan *weighted least connection*.

Oleh karena itu penelitian ini menerapkan algoritme *weighted least connection* karena pada setiap server memiliki nilai bobot yang berbeda dan menggunakan server heterogen. Penentuan nilai bobot (*weight*) ditentukan oleh *administrator*. *Administrator* dapat mengkonfigurasi server mana yang dapat menampung koneksi yang banyak dan sedikit. Nilai bobot secara *default* bernilai 1, tidak boleh bernilai 0. Pada algoritme *weighted least connection* nilai bobot disesuaikan dengan spesifikasi server, misalkan terdapat 2 server yang memiliki spesifikasi berbeda. Server 1 memiliki spesifikasi lebih unggul dibanding server 2, maka nilai bobot pada server 1 lebih besar dari server 2. Arsitektur jaringan yang digunakan yaitu *software defined network* karena memberikan manajemen jaringan yang dinamis dan kemudahan dalam melakukan konfigurasi jaringan yang dapat dilakukan terpusat pada *controller*. Dengan melakukan penelitian ini, diharapkan menjadikan kinerja *web* server yang lebih baik.

2. LANDASAN TEORI

2.1. *Load Balancing*

Load balancing merupakan sebuah teknik untuk mendistribusikan *traffic* yang masuk di antara *server* yang telah tersedia sehingga request/permintaan dapat ditangani dan memberikan respon yang lebih cepat (Rahman,

et al., 2014). Tugas dari *load balancing* untuk mendistribusikan beban kerja ke beberapa sumber daya komputasi dan juga untuk mengoptimalkan penggunaan *resource*, memaksimalkan *throughput*, dan menurunkan waktu respons agar tidak membebani sumber daya tunggal (Golinelli, 2015). *Load balancing* juga dapat diimplementasikan pada perangkat lunak maupun perangkat keras tergantung pada jenis beban yang didistribusikan. Perangkat lunak *load balancing* lebih hemat biaya daripada perangkat keras *load balancing* karena tidak memerlukan perangkat keras khusus.

2.2. *Weighted Least Connection*

Algoritme *weighted least connection* hampir sama dengan *least connection*, hanya saja algoritme ini menentukan bobot kinerja dari setiap *server* (Red Hat, 2014). Setiap *server* akan diberikan bobot, semakin tinggi nilai bobot yang diberikan pada *server* maka *server* akan menerima *request* lebih banyak dari *server* yang memiliki nilai bobot rendah.

Mekanisme pemilihan *server* berdasarkan algoritme *weighted least connection* dapat dilihat *pseudocode* berikut.

```
for (m = 0; m < n; m++) {
  if (W(Sm) > 0) {
    for (i = m+1; i < n; i++) {
      if (C(Sm)*W(Si) >
C(Si)*W(Sm))
        m = i;
        }
        return Sm;
       }
      }
```

Gambar 1. *Psuedocode* Algoritme *Weighted Least Connection*

Pada Gambar 1 diatas menjelaskan cara kerja *load balancing* dengan algoritme *weighted least connection* yaitu pertama melakukan pengecekan seluruh *server* yang sedang aktif. Lalu mengecek bobot *server* apakah bobot *server* tersebut lebih dari 0 jika ya maka akan dilanjutkan ke proses berikutnya. Setelah itu mengecek *server* saat ini dan *server* berikutnya untuk membandingkan nilai koneksi dan nilai bobot.

2.3. *Software Defined Network*

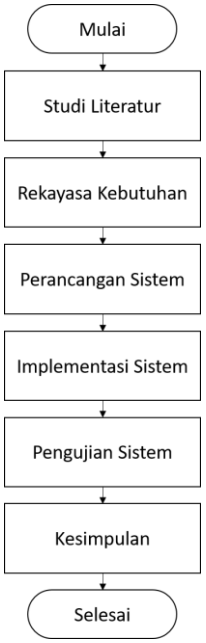
Software Defined Network (SDN) merupakan sebuah arsitektur yang memisahkan *network control/control plane* dan *forwarding function/data plane* yang memungkinkan *network control* menjadi program yang dapat diprogram secara langsung dan infrastruktur yang mendasarinya akan diabstraksikan untuk aplikasi dan layanan jaringan (Kreutz, et al., 2014). Manfaat dari SDN adalah arsitektur baru yang dinamis, mudah dikelola, hemat biaya, dan mudah beradaptasi sehingga ideal untuk *bandwidth* yang besar dan dinamis dari aplikasi saat ini.

2.4. *OpenFlow*

Openflow merupakan komunikasi antara *control plane* dan *data plane* pada arsitektur *software defined network* (Lee, 2014). *Openflow* adalah protokol standar *Open Network Foundation* (ONF) untuk mengontrol tabel *forwarding switch* atau *router* jarak jauh. Kontroler *Openflow* bagian dari kontroler SDN yang memberikan instruksi melalui channel ke *switch Openflow*. Sebuah *switch Openflow* mengelola beberapa *flow table* untuk menangani dan meneruskan paket ke tujuan.

3. METODOLOGI PENELITIAN

Pada bab ini menjelaskan studi literatur, rekayasa kebutuhan, perancangan sistem, implementasi sistem, pengujian system, dan kesimpulan untuk penelitian yang berjudul Implementasi *Load Balancing* Di *Web Server* Dengan Algoritme *Weighted Least Connection* Pada *Software Defined Network.* Alur penelitian dapat dilihat pada Gambar 2



Gambar 2. Diagram Alir Metode Penelitian

3.1. Studi Literatur

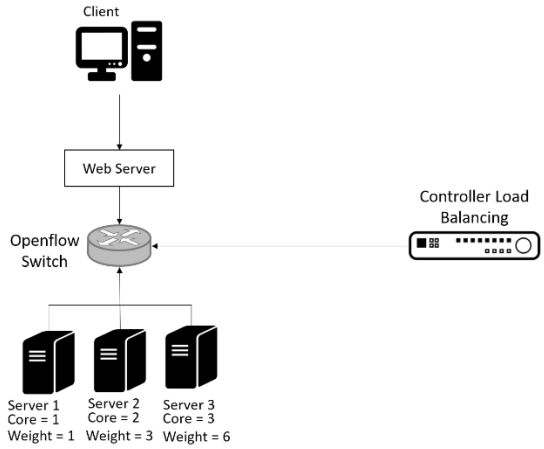
Pada sub bab ini menjelaskan studi literatur yang dilakukan berdasarkan pemahaman dasar-dasar teori yang terkait dengan beberapa konsep seperti *load balancing* dan *software defined network.* Literatur yang digunakan dalam penelitian ini didapatkan dari berbagai sumber seperti buku, paper (jurnal ilmiah), dan dokumen dari internet yang memiliki relevansi dengan penelitian ini.

3.2. Analisis Kebutuhan

Pada sub bab ini menjelaskan analisis kebutuhan untuk menentukan kebutuhan apa saja yang diperlukan. Kebutuhan sistem dapat mempermudah proses perancangan dan implementasi. Kebutuhan pada penelitian ini terdiri dari kebutuhan fungsional dan kebutuhan non-fungsional.

3.3. Perancangan Sistem

Perancangan sistem dilakukan setelah seluruh kebutuhan sistem dilakukan agar pembangunan sistem pada penelitian ini dapat terarah dan terstruktur. Perancangan sistem dapat dilihat pada Gambar 3.2.



Gambar 3. Perancangan Sistem

Pada Gambar 3 menjelaskan perancangan sistem yang digunakan. Sistem ini nantinya akan dipecah menjadi beberapa subsistem yaitu subsistem *controller*, subsistem *switch*, subsistem *web server*, dan subsistem *client*.

3.4. Implementasi Sistem

Implementasi sistem dilakukan setelah perancangan sistem dan perancangan algoritme selesai. Implementasi dilakukan sesuai dengan perancangan sistem. Pertama melakukan instalasi mininet pada sistem operasi linux, lalu melakukan instalasi *controller* POX, setelah itu membangun arsitektur *load balancing web server* pada *software defined network* di mininet *server*, kemudian membuat algoritme *weighted least connection* sebagai mekanisme *load balancing*, selanjutnya melakukan akses ke halaman *web server*, dan yang terakhir implementasi pengujian dengan parameter *throughput, connection rate*, dan *CPU usage* untuk mengetahui performa dan *kinerja web server*.

3.5. Pengujian dan Analisis Sistem

Pengujian dan analisis sistem dilakukan setelah mengetahui kinerja sistem apakah kebutuhan yang sudah di analisis sesuai atau tidak. Setelah itu melakukan analisis sekaligus melihat hasil dari *connection rate, response time, throughput*, dan *CPU usage*.

3.6. Kesimpulan

Kesimpulan dilakukan setelah semua tahapan telah selesai dilakukan, yaitu dari tahap analisis kebutuhan hingga tahap pengujian dan analisis sistem. Kesimpulan diambil berupa jawaban atas rumusan masalah yang telah ditentukan sebelumnya. Saran untuk

memberikan pertimbangan dengan mengembangkan penelitian yang lebih lanjut.

4. REKAYASA KEBUTUHAN

4.1 Deskripsi Umum Sistem

Sistem *load balancing* menggunakan algoritme *weighted least connection* merupakan sistem penyeimbang beban trafik untuk layanan *web* dengan jumlah koneksi dan nilai bobot (*weight*) pada tiap *server* sebagai faktor pembagian beban tersebut. Sistem ini menggunakan *flow* sebagai penentu jumlah koneksi pada *server*. Untuk pembagian beban pada *server*, informasi dari *flow* tersebut akan dikelola menggunakan algoritme *weighted least connection* yang sudah diterapkan pada *controller* tersebut.

4.2 Analisis Kebutuhan

Analisis kebutuhan sistem terbagi menjadi dua yaitu kebutuhan fungsional dan kebutuhan non-fungsional. Kebutuhan fungsional sistem yaitu *controller* dapat menjalankan modul *load balancing* menggunakan POX, *switch* dapat menerima paket dari *client*, *web server* dapat memberikan respon terhadap *request* dari *client*, *client* dapat mengirimkan *request* terhadap *web server*. Kebutuhan non-fungsional terdiri dari kebutuhan perangkat lunak dan kebutuhan perangkat keras. Kebutuhan perangkat lunak terdiri dari:

- Sistem operasi Linux Ubuntu 14.04
- Mininet versi 2.2.2
- Httperf
- Openflow versi 1.3

Kebutuhan perangkat keras yaitu 1 buah laptop dengan spesifikasi:

- *Processor* intel core i5-3230M CPU @2.60 GHz
- RAM 4 GB
- *Harddisk* kapasitas 1000 GB

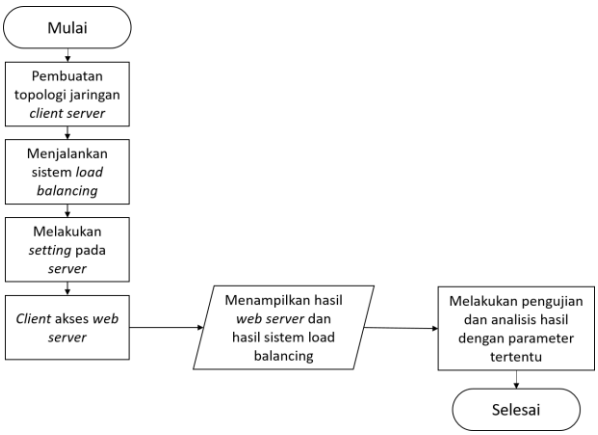
5. PERANCANGAN DAN IMPLEMENTASI

Pada bab ini menjelaskan tentang tahap-tahap perancangan yang dilakukan dalam pengerjaan penelitian ini. Bagian ini juga menjelaskan lebih mendalam pada konsep perancangan *load balancing*, alur komunikasi, pengecekan paket, alur sistem, dan diagram blok perancangan sistem.

5.1. Perancangan

5.1.1. Perancangan Arsitektur Software Defined Network

Pada bagian sub bab ini akan dijelaskan perancangan sistem dimulai dengan tahapan yang ada pada bab sebelumnya hingga pengujian yang dilakukan terhadap sistem.



Gambar 4. Diagram Alir Perancangan Arsitektur *Software Defined Network*

Pada Gambar 4 menjelaskan perancangan sistem yang dilakukan. Untuk memulai perancangan implementasi sistem *load balancing* dimulai dengan membuat topologi jaringan *client server* pada mininet, menjalankan sistem *load balancing*, melakukan *setting* pada *server*, *client* akses *web server*, menampilkan hasil sistem *load balancing*, dan yang terakhir melakukan pengujian dan analisis dengan parameter tertentu.

5.1.2. Perancangan Algoritme

Algoritme *weighted least connection* dipasang pada modul *load balancing* yang nantinya akan dijalankan menggunakan POX *Controller*. *Pseudocode* perancangan algoritme *weighted least connection* dapat dilihat pada Tabel 1.

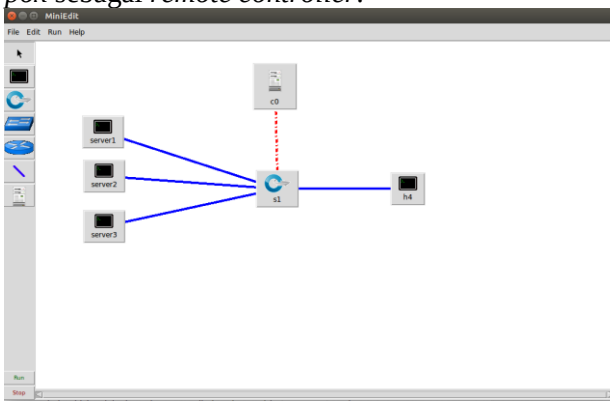
Tabel 1. *Pseudocode* Perancangan Algoritme *Weighted Least Connection*

```
1  menerima informasi koneksi
2  dari flow
3  simpan semua server yang
4  aktif ke variabel serverSet
5  inisialisasi variabel
6  selected_server adalah server
7  pertama
8  for (semua server pada server
9  set)
10  if (server pertama bisa
11 melayani koneksi)
12
```

```
13   server selanjutnya = index
14  server pertama + 1
15   for (server selanjutnya
16  sampai seluruh serverSet)
17   if (koneksi dari server
18  sekarang * bobot server
19  berikutnya > koneksi server
20  berikutnya * bobot server
21  sekarang)
22   index terbesar = index
23  server selanjutnya
24   selected_server = server
25  dengan index terbesar
```

5.2 Implementasi

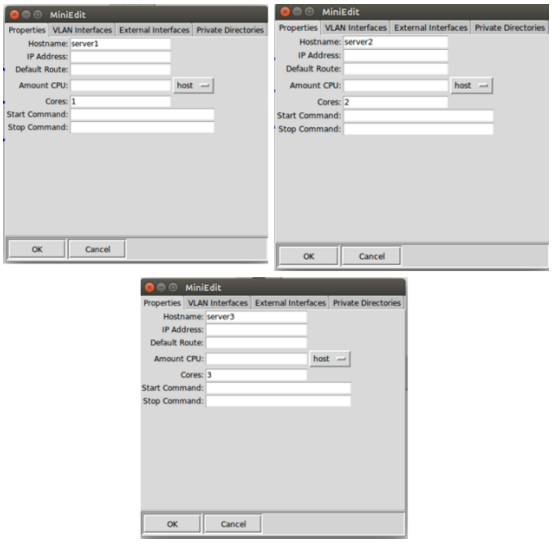
Pada tahap implementasi ini, dimulai dari membuat topologi menggunakan mininet dengan *single switch*, empat *host*, dan *controller pox* sebagai *remote controller*.



Gambar 5. Pembuatan Topologi

Pada Gambar 5 merupakan hasil dari pembuatan topologi. Dari gambar diatas dapat diketahui bahwa menambahkan *host* sebanyak empat (*server*1, *server*2, *server*3, h4), *single switch* (s1), dan *controller* (c0). Untuk menghubungkan masing-masing host dilakukan *adding links* dari (*server*1,s1) sampai (h4,s1) yang berarti *server*1 sampai h4 terhubung dengan satu *switch*. Setelah selesai dalam pembuatan topologi, selanjutnya *controller* dan *switch* akan menjalankan sistem.

Setelah itu melakukan konfigurasi *server* heterogen menggunakan mininet dengan mengisi *core*/jumlah prosesor pada tiap *server*.



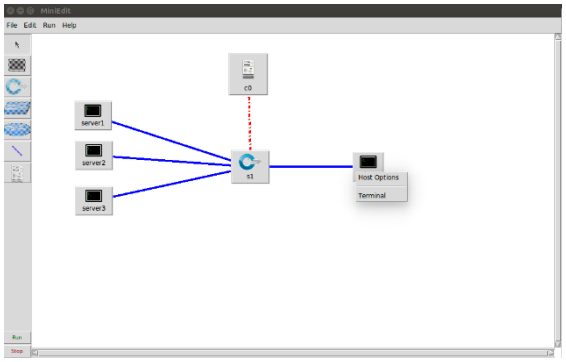
Gambar 6. Pengesetan *Core*/Jumlah Prosesor Pada *Server*

Pada Gambar 6 menunjukan pengesetan *core*/jumlah prosesor pada setiap *server*. Pada *server* 1 di set dengan jumlah prosesor 1, *server* 2 di set dengan jumlah prosesor 2, dan *server* 3 di set dengan jumlah prosesor 3.

Lalu menjalankan *server* seperti pada Gambar 7 dengan menjalankan *command* python -m SimpleHTTPServer 80 yang berarti server menggunakan protokol HTTP dan *port* 80.

Gambar 7. Setting Server

Langkah berikutnya melakukan konfigurasi *setting client* seperti pada Gambar 8 yang menjelaskan untuk membuka terminal pada setiap *host*, klik kanan pada *host* yang dipilih lalu klik terminal. Pada *host* ini nantinya digunakan *client* untuk melakukan *request*.



Gambar 8. Setting Client

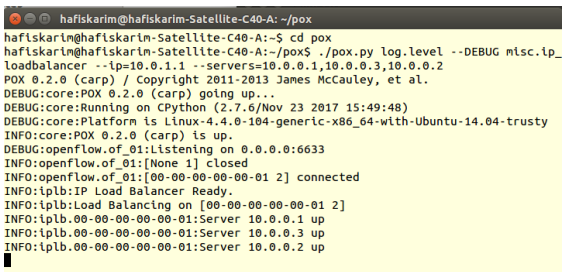
5.2.1 Menjalankan *script* ip_loadbalancer

Untuk menjalan sistem *load balancing* dengan algortime *weighted least connection* pada *software defined network* menggunakan POX *controller* yaitu menjalankan *script* sebagai berikut:

Tabel 2. *Script* ip_loadbalancer

```
$ ./pox.py log.level --DEBUG
misc.ip_loadbalancer --
ip=10.0.1.1 --
servers=10.0.0.1,10.0.0.3,10.0.0.
2
```

Pada Tabel 2 menjelaskan untuk melakukan *import module load balancing* dengan direktori pada pox. Kemudian dilanjutkan dengan menjalankan program *load balancing*. Setelah itu mengatur alamat *web server* dengan ip 10.0.1.1 dan *server* pada *host* dengan ip 10.0.0.1 sampai 10.0.0.3. Berikut tampilan setelah menjalankan *script* tersebut.



Gambar 9. Menjalankan *Load Balancing*

Pada Gambar 9 merupakan hasil dari *command* untuk *menjalankan load balancing*. Dari gambar diatas dapat diketahui ketika *controller* pox sudah aktif maka *controller* akan melakukan *listening* pada *port* 6633, maka sistem *load balancing* siap untuk dijalankan. Lalu *server* dengan ip 10.0.0.1 sampai 10.0.0.3 akan *up* atau aktif.

Selanjutnya *client* melakukan *request* dengan melakukan *command* #curl pada alamat ip 10.0.1.1. lalu akan muncul halaman *doctype* html pada *web server* yang nantinya data tersebut digunakan untuk melakukan *request* ke *server*.

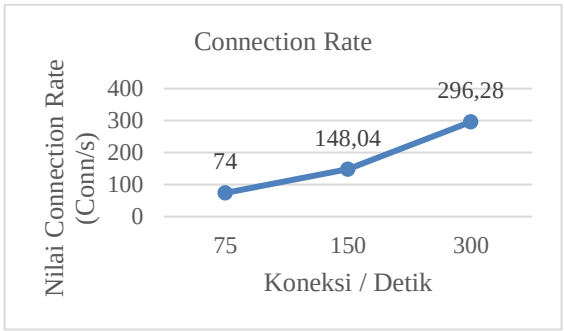
Gambar 10. *Client* Akses *Web Server*

Pada Gambar 10 merupakan hasil dari *client* melalukan *request* ke *web server*. Dari gambar diatas dapat diketahui bahwa *request client* mengunakan *command #curl* telah dikirim oleh *server* yang berupa *doctype html.*

6. PENGUJIAN DAN HASIL

6.1. Pengujian *Connection Rate*

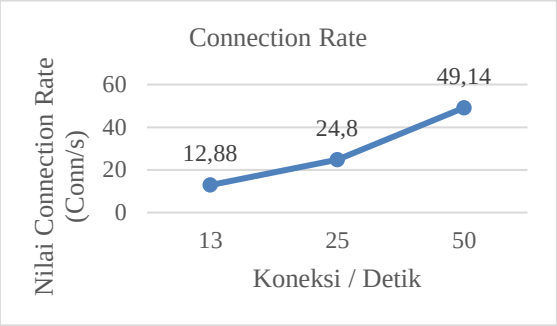
Pada pengujian skenario ini terbagi menjadi 2 yaitu client mengakses halaman html dengan memberikan *request* sebanyak 3000 dengan *rate* 75, 150, 300 dan client mengakses gambar ukuran data sebesar 2,2 MB dengan memberikan *request* dari *client* sebanyak 500 dengan *rate* 13, 25, dan 50. Pengujian ini dilakukan sebanyak 5 kali yang nantinya hasil pengujian akan diambil dari nilai rata-rata.



Gambar 11. Perbandingan Nilai *Connection Rate* dengan Jumlah Koneksi 3000 dan *Rate* 75, 50, dan 100 Koneksi/Detik

Pada Gambar 11 menunjukkan nilai *connection rate* pada pengujian yang dilakukan. Ketika pengujian dilakukan dengan *rate* 75

koneksi/detik memperoleh nilai *connection rate* 74 Conn/s, pada pengujian 150 koneksi/detik memperoleh nilai *connection rate* 148,04 Conn/s dan pengujian 300 koneksi/detik memperoleh nilai *connection rate* 296,28 Conn/s.

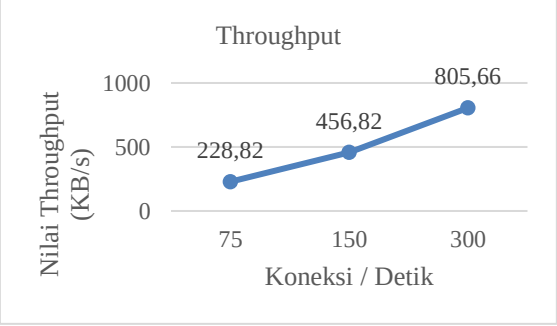


Gambar 12. Perbandingan Nilai *Connection Rate* dengan Jumlah Koneksi 500 dan *Rate* 13, 25, dan 50 Koneksi/Detik

Pada Gambar 12 menunjukkan nilai *connection rate* pada pengujian yang dilakukan. Ketika pengujian dilakukan dengan *rate* 13 koneksi/detik memperoleh nilai *connection rate* 12,88 Conn/s, pada pengujian 25 koneksi/detik memperoleh nilai *connection rate* 24,8 Conn/s dan pengujian 50 koneksi/detik memperoleh nilai *connection rate* 49,14 Conn/s.

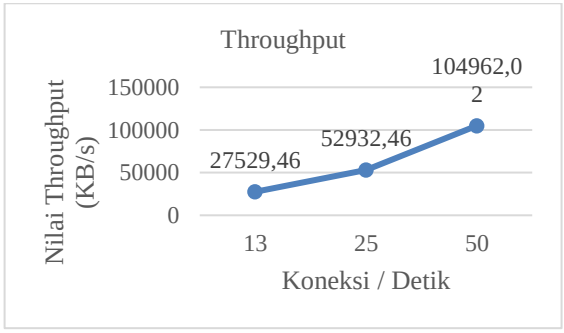
6.2. Pengujian *Throughput*

Pada pengujian skenario ini terbagi menjadi 2 yaitu client mengakses halaman html dengan memberikan *request* sebanyak 3000 dengan *rate* 75, 150, 300 dan client mengakses gambar ukuran data sebesar 2,2 MB dengan memberikan *request* dari *client* sebanyak 500 dengan *rate* 13, 25, dan 50. Pengujian ini dilakukan sebanyak 5 kali yang nantinya hasil pengujian akan diambil dari nilai rata-rata.



Gambar 13. Perbandingan Nilai *Throughput* dengan Jumlah Koneksi 3000 dan *Rate* 75, 150, dan 300 Koneksi/Detik

Pada Gambar 13 menunjukkan nilai *throughput* pada pengujian yang dilakukan. Ketika pengujian dilakukan dengan *rate* 75 koneksi/detik memperoleh nilai *throughput* 228,82 KB/s, pada pengujian 150 koneksi/detik memperoleh nilai *throughput* 456,82 KB/s dan pengujian 300 koneksi/detik memperoleh nilai *throughput* 805,66 KB/s.

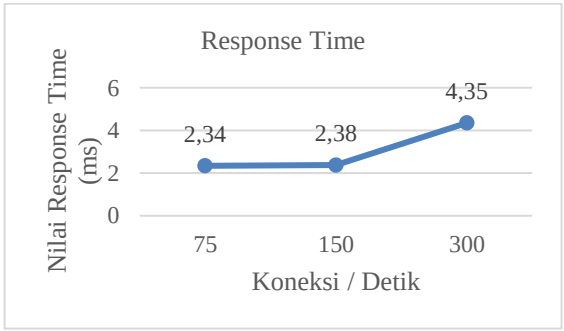


Gambar 14. Perbandingan Nilai *Throughput* dengan Jumlah Koneksi 500 dan *Rate* 13, 25, dan 50 Koneksi/Detik

Pada Gambar 14 menunjukkan nilai *throughput* pada pengujian yang dilakukan. Ketika pengujian dilakukan dengan *rate* 13 koneksi/detik memperoleh nilai *throughput* 27529,46 KB/s, pada pengujian 25 koneksi/detik memperoleh nilai *throughput* 52932,46 KB/s dan pengujian 50 koneksi/detik memperoleh nilai *throughput* 104962,02 KB/s.

6.3. Pengujian *Response Time*

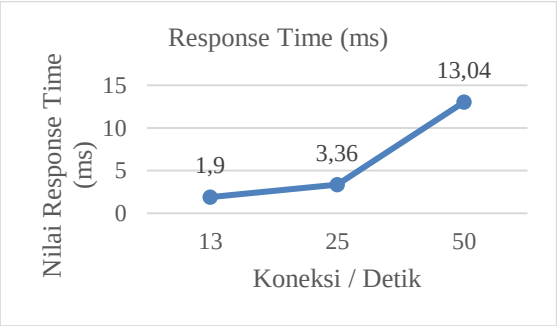
Pada pengujian skenario ini terbagi menjadi 2 yaitu client mengakses halaman html dengan memberikan *request* sebanyak 3000 dengan *rate* 75, 150, 300 dan client mengakses gambar ukuran data sebesar 2,2 MB dengan memberikan *request* dari *client* sebanyak 500 dengan *rate* 13, 25, dan 50. Pengujian ini dilakukan sebanyak 5 kali yang nantinya hasil pengujian akan diambil dari nilai rata-rata.



Gambar 15. Perbandingan Nilai *Response Time* dengan Jumlah Koneksi 3000 dan *Rate* 75, 150, dan 300 Koneksi/Detik

Pada Gambar 15 menunjukkan nilai *response time* pada pengujian yang dilakukan.

Ketika pengujian dilakukan dengan *rate* 75 koneksi/detik memperoleh nilai *response time* 2,34 ms, pada pengujian 150 koneksi/detik memperoleh nilai *response time* 2,38 ms dan pengujian 300 koneksi/detik memperoleh nilai *response time* 4,35 ms.

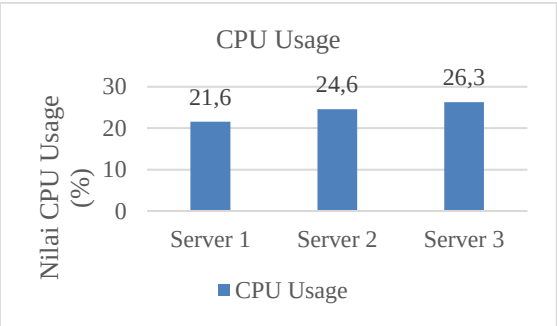


Gambar 16. Perbandingan Nilai *Response Time* dengan Jumlah Koneksi 500 dan *Rate* 13, 25, dan 50 Koneksi/Detik

Pada Gambar 16 menunjukkan nilai *response time* pada pengujian yang dilakukan. Ketika pengujian dilakukan dengan *rate* 13 koneksi/detik memperoleh nilai *response time* 1,9 ms, pada pengujian 25 koneksi/detik memperoleh nilai *response time* 3,36 ms dan pengujian 50 koneksi/detik memperoleh nilai *response time* 13,04 ms.

6.4. Pengujian *CPU Usage*

Pada pengujian skenario ini dilakukan untuk melihat penggunaan *CPU* dari setiap server. Pengujian ini dilakukan sebanyak 5 kali dengan *request* dari *client* sebanyak 3000 dan *rate* 300.



Gambar 17. Perbandingan CPU *Usage*

Pada Gambar 17 menunjukkan bahwa penggunaan *CPU* pada server 1 nilai rata-rata yang didapat yaitu 21,6%, server 2 mendapatkan nilai rata-rata 24,6% dan server 3 mendapatkan nilai rata-rata 26,3%. Dari hasil pengujian diatas menunjukkan server 3 penggunaan *CPU*-nya lebih tinggi dibanding server 1 dan 2. Hal ini terjadi karena perbandingan jumlah koneksi pada setiap server dengan algoritme *weighted least connection* menampung jumlah koneksi yang berbeda.

7. KESIMPULAN

Penerapan *load balancing* dengan algoritme *weighted least connection* dapat membagi beban trafik berdasarkan bobot kinerja dari tiap *server*. Penentuan bobot kinerja ditentukan oleh administrator dengan melihat spesifikasi dari *server* yang digunakan. *Server* yang memiliki spesifikasi tinggi akan diberikan nilai bobot yang tinggi, sedangkan *server* yang memiliki spesifikasi yang rendah akan diberikan nilai bobot yang rendah. Nilai bobot/*weight* sangat berpengaruh dalam menampung jumlah koneksi oleh suatu *server*. *Server* yang memiliki spesfikasi yang lebih unggul dari *server* lainnya maka akan diberi nilai *weight* lebih besar agar dapat menampung koneksi yang lebih banyak.

Pada pengujian sistem yang dilakukan pada penelitian ini terdiri dari beberapa skenario diantaranya mengukur nilai *connection rate*, *response time*, *throughput* dengan memberikan request dari client sebanyak 3000 dengan rate 75, 150, dan 300. Pada skenario pengujian ketika *client* mengakses halaman html. Hasilnya untuk *connection rate* tertinggi yaitu pada pengujian dengan *rate* 300 sebesar 296,28 Conn/s, sedangkan pada pengujian dengan *rate* 75 memperoleh nilai *connection rate* terkecil yaitu 74 Conn/s. Untuk nilai *response time* terkecil yaitu pada pengujian dengan *rate* 75 yaitu 2,34 ms, sedangkan pada pengujian dengan *rate* 300 memperoleh nilai *response time* terbesar sebesar 4,35 ms. Untuk nilai *throughput* tertinggi yaitu pada pengujian dengan *rate* 300 sebesar 805,66 KB/s, sedangkan pada pengujian dengan *rate* 75 memperoleh nilai *throughput* terkecil yaitu 228,82 KB/s.

Lalu pada skenario pengujian ketika *client* mengakses gambar dengan ukuran data sebesar 2,2 MB dan memberikan *request* dari *client* sebanyak 500 dengan *rate* 13, 25, dan 50. Hasilnya untuk *connection rate* tertinggi yaitu pada pengujian dengan *rate* 50 sebesar 49,14 Conn/s, sedangkan pada pengujian dengan *rate* 13 memperoleh nilai *connection rate* terkecil yaitu 12,8 Conn/s. Untuk nilai *response time* terkecil yaitu pada pengujian dengan *rate* 13 yaitu 1,9 ms, sedangkan pada pengujian dengan *rate* 50 memperoleh nilai *response time* terbesar sebesar 13,04 ms. Untuk nilai *throughput* tertinggi yaitu pada pengujian dengan *rate* 50 sebesar 104962,02 KB/s, sedangkan pada

pengujian dengan *rate* 13 memperoleh nilai *throughput* terkecil yaitu 27529,46 KB/s. Pada penggunaan *CPU*, server 3 memiliki nilai penggunaan lebih tinggi sebesar 26,3% dibanding server lainnya. Hal ini terjadi karena server 3 dirancang untuk dapat menampung koneksi yang lebih banyak dibanding *server* 1 dan 2.

8. DAFTAR PUSTAKA

Angsar, N., 2014. *Pengujian Distribusi Beban Web dengan Algoritma Least Connection dan Weighted Least Connection*. Volume 3.

Golinelli, E. S., 2015. *A Software Defined Networking Evaluation Approach to Distrubting Load Using POX, Floodlight and BIG-IP 1600*. Oslo: University of Oslo.

Kreutz, D. et al., 2014. *Software Defined Networking: A Comprehensive Survey*. Procedings of the IEEE, Volume 103.

Laksana, E. K. D., 2016. *Analisis Openflow Load Balancing Web Server dengan Algoritma Least Connection.* Malang: Universitas Brawijaya.

Lee, K.-H., 2014. *Mobility Management Framework in Software Defined Networks.* International Journal of Software Engineering and Its Applications, Volume 8, pp. 1-10.

Rahmana, D., 2017. *Analisis Load Balancing Pada Web Server Menggunakan Algoritme Least Connection*. Malang: Universitas Brawijaya.

Rahman, M., Iqbal, S. & Gao, J., 2014. *Load Balancer as a Service in Cloud Computing*. 2014 IEEE 8th International Symposium on Service Oriented System Engineering.

Red Hat, I., 2014. *Red Hat Enterprise Linux 5*. [Online] Available at: https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/5/html/Virtual_Server_Administration/index.html [Diakses 12 September 2017].

Sabiya & Singh, J., 2016. *Weighted Round-Robin Load Balancing Using Software*. International Journal of Advanced Research in, 6(6), pp. 621-625.

Singh, H. & Kumar, D. S., 2011. *Dispatcher Based Dynamic Load Balancing on Web Server System*. International Journal of Grid and Distributed Computing, Volume 4, pp. 89-106.