Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer e-ISSN: 2548-964X
Vol. 3, No. 1, Januari 2019, hlm. 395-403 http://j-ptiik.ub.ac.id

Analisis Perbandingan Kinerja Metode *Load Balancing* Berbasis *CPU* Dengan Berbasis *Response Time* Pada *Software Defined Network*

Muharrom Abdillah[1], Widhy Yahya[2], Achmad Basuki[3]

Program Studi Teknik Informatika, Fakultas Ilmu Komputer, Universitas Brawijaya
Email: [1]muharromab@gmail.com, [2]widhy.yahya@ub.ac.id, [3]abazh@ub.ac.id

Abstrak

Server merupakan bagian penting dalam sebuah layanan didalam jaringan komputer. Peran *server* dapat menentukan kualitas baik buruknya dari layanan tersebut. Semakin banyak *request*, akan membuat *server* terbebani dan membuat kinerja dari *server* menjadi buruk. Solusinya adalah dengan menggunakan *multiple server* yang menerapkan metode *load balancing*, agar kinerja *server* menjadi lebih stabil. Metode *load balancing* diimplementasikan pada *software defined network.* Arsitektur ini menawarkan *scalability* dan *programmability* untuk penggunaan jaringan yang semakin kompleks seperti *load balancing web server*. Beberapa penelitian tentang *load balancing* pada *software defined network* dengan berbagai metode yang diterapkan akan menghasilkan kinerja yang berbeda-beda, diantaranya yaitu metode *round robin* , metode berbasis *CPU usage* dan metode berbasis *response time.* Penelitian ini memberikan analisis terhadap kinerja dari metode *round robin*, metode berbasis *CPU dan* metode berbasis *response time.* Pengujian dilakukan dengan tiga kategori *rate* yaitu *low* untuk *rate* 40, *medium* untuk *rate* 80 dan *high* untuk *rate 160.* Pengujian dari masing-masing *rate* dilakukan dalam waktu 10 detik. Parameter pengujian yang digunakan adalah *throughput* dan *CPU usage.* Hasil pengujian *throughput* menunjukkan bahwa metode *round robin* unggul pada pengujian kategori *low* dengan nilai rata-rata 32,64 KB/s , sedangkan pada pengujian kategori *medium* dan *high* metode berbasis *response time* lebih unggul dengan nilai rata-rata 65,02 KB/s dan 115,50 KB/s. Sementara pada pengujian *CPU usage*, metode berbasis *CPU* memiliki standar deviasi yang terendah dari semua kategori rate dengan nilai rata-rata 3,95%, 4,69% dan 5,90%.

Kata kunci: *load balancing, round robin, CPU, response time, software defined network*

Abstract

Server is part of important thing in a service inside computer network. the purpose of server can determine how fine the quality of the services. More request will make the server have more loads and decreasing performance of server. The solution is to use multiple servers that implement load balancing method, so that servers performance becomes more stable. Load balancing method implemented in software defined network. This architecture offers scalability and programmability for more complex network like load balancing for web server. There are any research about load balancing in SDN with several method implemented, they are round robin method, CPU usage based method and response time based method. This research give analysis to performance of round robin method, CPU usage based method and response time based method. The trial are doing to three categories of rates, they are low for rate 40, medium for rate 80 and high for rate 160. The trial are doing in 10 seconds for each rate. The parameters used in this trial are throughput and CPU usage. The throughput test resulting that round robin is excellent in low category with average value 32,64 KB/s, meanwhile in medium and high category response time method is more excellent with average values 65,02 KB/s and 115,50 KB/s. Whereas, in CPU usage test, CPU usage method have the lowest standart deviation with average values 3,95%, 4,69%, and 5,90%.

Keywords: *load balancing, round robin, CPU, response time, software defined network*

1. PENDAHULUAN

*Serve*r merupakan bagian penting dalam sebuah layanan didalam jaringan komputer. Peran *server* dapat menentukan kualitas baik buruknya dari layanan tersebut. Jika *server* yang dimiliki hanya ada satu dan bekerja tunggal, maka memungkinkan terjadinya "*a single point of failure" (SPOF)*. *SPO*F adalah kondisi *server* yang gagal memberi respon dan mengakibatkan sistem tidak berfungsi atau mati (Moniruzzaman, 2015). Masalah ini terjadi karena terlalu banyak *request* yang harus di *handle* oleh satu *server*. Solusinya, dibuatlah *multiple server* untuk mengatasi masalah tersebut. Didalam *multiple server* dibutuhkan metode dalam pembagian beban agar kinerja *server* tetap stabil. Metode yang digunakan adalah metode load balancing.

Load balancing adalah teknik untuk mendistribusikan beban pada dua atau lebih jalur koneksi secara seimbang agar dapat berjalan optimal, memaksimalkan *throughput*, memperkecil waktu tanggap dan menghindari *overload* pada salah satu jalur koneksi (Arianto & Sholeh, 2014). *Load balancing* akan membagikan beban yang datang ke beberapa *server*, gunanya untuk menghindari *overload* pada salah satu jalur koneksi. Umumnya metode *load balancing* sendiri sudah banyak diimplementasikan pada perangkat internet sekarang. Kekurangannya, pada perangkat yang sering digunakan sekarang *control plane* dan *data plane* tertanam pada perangkat jaringan yang sama, dalam artian *control plane* dan *data plane* tidak dipisah. Jadi apabila ingin bereksperimen dengan beberapa teknik seperti teknik *load balancing* baru, maka tidak akan dapat dilakukan. Karena pada perangkat internet sekarang, sistem operasi dan fitur-fiturnya sudah langsung *embedded* pada perangkat tersebut. Oleh karena itu kita memerlukan teknologi yang mampu menangani penggunaan jaringan yang sangat kompleks yaitu *Software defined network* (*SDN*).

SDN merupakan sebuah cara baru untuk mengelola, mengimplementasikan dan mendesain jaringan dimana *control plane* dan *data plane* dibuat terpisah. Arsitektur *SDN* dapat memudahkan dalam melakukan pemrograman secara langsung. Agar dapat berkomunikasi antara perangkat dan *controller* pada *SDN* menggunakan sebuah *protocol* yang disebut dengan *openflow*. *Openflow* adalah *protocol* untuk mengontrol *data plane* yang secara fisik sudah terpisah dari *control plane* menggunakan perangkat lunak yaitu *controller* pada sebuah *server* (Kartadie & Utami, 2014). Dengan adanya konsep *SDN* operator atau *network administrator* sangat dimudahkan dalam melakukan pengelolahan jaringan. Karena konsep utama pada arsitektur ini adalah sentralisasi kendali jaringan dengan semua pengaturan berada pada *control plane* (Ummah & Abdillah, 2016).

Penelitian tentang *load balancing* pada *SDN* sudah pernah dilakukan sebelumnya dengan berbagai metode dan menghasilkan kinerja yang berbeda-beda. Pertama "Implementasi *Load Balancing* di Web *Server* Menggunakan *Metode Berbasis Sumber Daya CPU* Pada *SDN"* (Julianto, 2017). Penelitian ini mencoba mengembangkan metode *load balancing* dengan mempertimbangkan kondisi dari *server* dengan cara mengambil sumber daya *CPU* yang paling ringan atau kecil. Pengujian pada penelitian ini dilakukan dengan menggunakan server *heterogen* atau dengan kapasitas *server* yang berbeda. Hasil yang didapatkan metode berbasis *CPU usage* dapat berjalan dengan baik dengan membagi beban paling banyak pada *server* yang memiliki jumlah *core* paling besar dibandingkan pada *server* yang memiliki jumlah *core* kecil. Penelitian yang ke dua adalah "An Efficient SDN Load balancing Scheme Based on *Server response time*" (Zhong & Fang, 2016). *SDN controller* akan mengambil sampel *server response time* dari *server* dengan mengirimkan pesan *paket_out* kepada *server*. Kemudian *server* membalas dengan mengirimkan *paket_in* dan dilakukan perhitungan. Hasilnya akan dibandingkan, kemudian ketika *client* melakukan *request* akan diarahkan pada *server* dengan *response time* paling kecil. Penelitian ini dilakukan pada *server homogen* atau dengan kapasitas server yang sama. Hasil yang diperoleh metode berbasis *response time* dapat berjalan dengan baik dengan mendistribusikan beban pada *server* yang memiliki *response time* tercepat. Pengujian pada penelitan ini dilakukan dengan menggunakan *server homogen.*

Dari penelitian yang sudah diakukan sebelumnya, maka peneliti berfokus pada analisis perbandingan kinerja yaitu "Analisis Perbandingan Kinerja Metode Berbasis *CPU* dengan berbasis *response time* untuk *Load*

balancing pada *Software Defined Network*". Penelitian ini ditujukan untuk mencari kinerja terbaik dengan menggunakan parameter yang sudah ditentukan dari metode berbasis *CPU* dan metode berbasis *response time*. Pada penelitian ini juga di lakukan analisis terhadap metode *round robin* yang digunakan sebagai pembanding terhadapa kedua metode tersebut. Alasan peneliti membandingkan metode berbasis *CPU* dan metode berbasis *response time* adalah karena pada metode pada metode berbasis *CPU* dan metode berbasis *response time* proses *load balancing* sama-sama dilakukan dengan melihat kondisi dari *server*. Selain itu, terdapat korelasi antara metode berbasis *CPU* dengan metode berbasis *response time*, dimana kondisi *CPU* dari *server* akan mempengaruhi waktu respon dari *server*. Sedangkan, alasan metode *round robin* digunakan sebagai pembanding adalah karena pada metode *round robin* proses *load balancing* dilakukan tanpa melihat kondisi dari *server.r*.

2. LANDASAN KEPUSTAKAAN

2.1. *Software Defined Network*

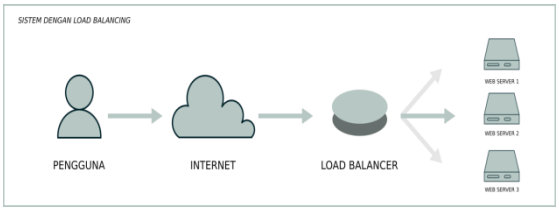
Software Defined Network adalah sebuah cara baru di dunia *networking*, merupakan sebuah pendekatan baru untuk membangun, mendesain serta me-*manage* jaringan komputer. Konsep dasar *SDN* sendiri adalah melakukan pemisahan antara *control plane* dan *forwarding plane*, dimana *controller* tersebut dapat kita program secara langsung (Kartadie & Utami, 2014). Dengan adanya pemisahan tersebut, maka diharapkan perangkat jaringan dapat di *manage* melalui *controller* saja, maka untuk mewujudkan tersebut dibutuhkan sebuah *API* untuk mengoneksikan seluruh perangkat jaringan kedalam sebuah *controller* yang dapat di program sesuai kebutuhan, dari sanalah paradigma *Software Defined Network* muncul, dimana jaringan dapat diatur atau di definisikan melalui sebuah *software*.

2.2. *Openflow*

Openflow merupakan protokol yang digunakan oleh *controller* agar dapat melakukan komunikasi dengan infrastruktur jaringan dibawahnya. *Openflow* menjelaskan bagaimana mekanisme komunikasi yang dilakukan antara *controller layer* dan *forwarding layer (*Open Networking Foundation, 2009). Selain itu *OpenFlow* juga memberikan akses langsung untuk mengatur dan mengubah *forwarding plane* pada *network device* baik *physical* maupun *virtual*. Tidak hanya *Openflow* satu-satunya protokol yang dapat digunakan untuk mengembangkan *SDN*. Masih banyak protokol lainnya baik yang *open source* maupun yang berbayar yang dapat digunakan untuk mengembangkan *Software Defined Network*.

2.3. *Load Balancing*

Load balancing bisa diartikan sebagai teknik atau proses pendistribusian lalu lintas jaringan dengan menggunakan perangkat-perangkat jaringan. Teknik ini akan



mendistribusikan beban *traffic* pada dua atau lebih jalur koneksi secara seimbang. *Load balancing* juga mendistribusikan beban kerja secara merata di dua atau lebih komputer, *link* jaringan, *CPU*, *hard drive* atau sumber daya lainnya, untuk mendapatkan pemanfaatan sumber daya yang optimal (Sirajuddin & Affandi, 2012)

Gambar 1. Sistem *Load Balancing*

Gambar 1 adalah gambar layanan web yang menggunakan mekanisme *load balancing*. Pada gambar tersebut terdapat 3 *web server* yang akan melayani *request* dari pengguna. *Load balancer* akan mendistribusikan sebuah *traffic* pada *web server* sesuai dengan metode yang digunakan. Dengan demikian metode *load balancing* dapat mencegah terjadinya *overload* pada sebuah *server*, dapat memperkecil waktu respon dan dapat memberikan *throughput* yang maksimal.

2.4. *Controller*

Controller merupakan bagian dari *SDN* yang memiliki peran sangat penting. Fungsi dari *controller* adalah mengelola kontrol aliran ke *switch / router* (melalui *API*) dan mengatur jalur mana yang akan dipilih. Pengaturan yang ada pada jaringan seperti *routing*, *load balancing* dan pengaturan lainnya ada pada *controller*, sehinnga *controller* juga bisa disebut sebagai "otak" dalam arsitektur jaringan tersebut (Lara & Kolasani, 2014).

2.5. Metode *Round Robin*

Metode *round robin* merupakan metode yang akan membagi beban secara merata pada *server*. Beban akan dibagikan dari satu *server* ke *server* lainnya secara bergiliran dan berurutan. Metode ini tidak memperdulikan kapasitas *server* ataupun beban *request*. Konsep dasar dari metode round robin adalah dengan menggunakan *time sharing*, pada intinya metode ini memproses antrian secara bergiliran. Jika terdapat 3 *server* (1,2,3), *request* pertama akan diarahan ke *server* 1, *request* ke dua akan diarahkan ke *server* 2, *request* ke 3 akan diarahkan ke server 3 dan *request* ke empat akan diarahkan kembali ke *server* 1, begitu seterusnya tanpa memperdulikan kapasitas pada *server* (Ellrod, Load Balancing – Round Robin, 2010).

Tabel 1 *Pseudocode* Metode *Round Robin*

```
1  Begin
2   initialize the lastServer value
3  to 0
4   Determine the totalServer
5   lastServer value is lastServer +
6  1 mod totalServer
7   Return lastServer
8  End
```

2.6. Metode berbasis *CPU Usage*

Metode berbasis *CPU usage* melakukan pembagian beban berdasarkan pemakaian *CPU* terkecil dari *server* (Julianto, 2017) . Ketika *user* melakukan *request*, sebelumnya *server* terlebih dahulu telah mengirimkan *resource* berupa *CPU usage* ke *controller*. Kemudian *controller* membandingkan nilai *resource* tersebut. Dari hasil perbandingan tersebut akan dimbil nilai *resource* yang terkecil. Kemudian ketika ada *request*, *controller* akan mengirimkan *paket_out* ke *switch* dengan perintah untuk mem-*forward* paket ke *server* yang memiliki pemakaian *CPU* terkecil.

Tabel 2 *Pseudocode* Metode Berbasis *CPU*

```
1  Begin
2   Resource CPU usage information
3  from web server
4   Store CPU usage information to
5  cpuUsage variable
6   Initilize minimumResource to web
7  server 1
8   If no resource in cpuUsage
9   Return minimumResource
10  Get minimum resource from
11 cpuUsage
12  Set minimium resource to
13 minimumResource
   Return minimumResource
   End
```

2.7. Metode berbasis *Response Time*

Metode berbasis *response time* melakukan pembagian beban berdasarkan waktu respon tercepat dari sebuah *server*. Beban berikutnya akan diberikan pada *server* dengan *waktu response tercepat*. Karena waktu respon *server* mencerminkan kemampuan beban pada *server* tersebut (Zhong & Fang, 2016). Untuk mendapatkan waktu respon dari setiap *server*, *controller* akan mengirimkan *packet_out* ke *switch*, kemudian *switch* akan meneruskan paket tersebut ke *server*. Kemudian *server* mengirimkan pesan balasan ke alamat sumber yaitu *controller*. Dari pesan balasan tersebut *controller* mendapatkan waktu respon dari *server*. Oleh karena itu metode berbsis *response time* ini sangat baik digunakan sebagai penyeimbang beban pada server.

Tabel 3 *Pseudocode* Metode Berbasis *Response Time*

```
1  Begin
2   The controller sends the ARP
3  request packet
4   The controller records the time
5  when the ARP packet is sent
6   The controller receives a packet
7  arp response
8   Controller calculates response
9  time and save to
10  server response's variable
11  Get minimum response time from
12 server response
13  Save minimum response time to
14 minimum responseTime
15  Return minimum responseTime
   End
```

2.8. *Web Server*

Web server merupakan perangkat lunak dalam *server* yang memberikan layanan berupa permintaan (*request*) kepada pengguna *web* dengan mengunakan protokol *HTTP* atau *HTTPS* yang biasa disebut dengan *browser*, dan akan mengirimkan kembali (*response*) hasil dari permintaan tersebut kedalam bentuk halaman *web*. Penggunaan paling umum dari sebuah *web server* adalah untuk menempatkan sebuah situs *web*, namun pada kenyataannya penggunaannya dapat diperluas sebagai tempat penyimpanan data ataupun untuk menjalankan sejumlah aplikasi bisnis (Kadir, 2003)

2.8. *Psutil*

Psutil merupakan modul *library python* yang menyediakan sebuah *interface* untuk

mengambil informasi pada semua proses yang berjalan dari suatu perangkat komputer seperti *CPU, memory, hardisk* dan sebagainya, dengan cara yang sederhana menggunakan *python* (Psutil Documentation, 2018).

2.9. *Httperf*

Httperf merupakan sebuah *tool* yang digunakan dalam melakukan pengujian untuk mendapatkan kinerja dari suatu *web server*. Kelebihan dari tool ini ringan dan stabil ketika dilakukan pengujian kinerja pada sebuah *web server* (Mosberger & Jin, 1998).

3. METODE PENELITIAN

3.1. Studi Literatur

Studi literatur bertujuan sebagai dasar-dasar teori yang digunakan untuk menunjang dalam penelitian ini. Adapun literatur yang digunakan dalam penelitian ini yaitu jurnal, artikel, buku, e-book, dan penelitian sebelumnya yang memiliki keterkaitan dengan penelitian ini.

3.2. Analisis Kebutuhan

Analisis Kebutuhan menjelaskan kebutuhan apa saja yang diperlukan dalam pengerjaan penelitian ini. Analisis kebutuhan diperlukan agar penelitian ini dapat berjalan dan mendapatkan hasil yang diinginkan.

3.2.1. Kebutuhan Perangkat Keras

Pada penelitian ini membutuhkan perangka keras dalam implementasinya. Perangkat keras yang digunakan dalam penelitian ini adalah

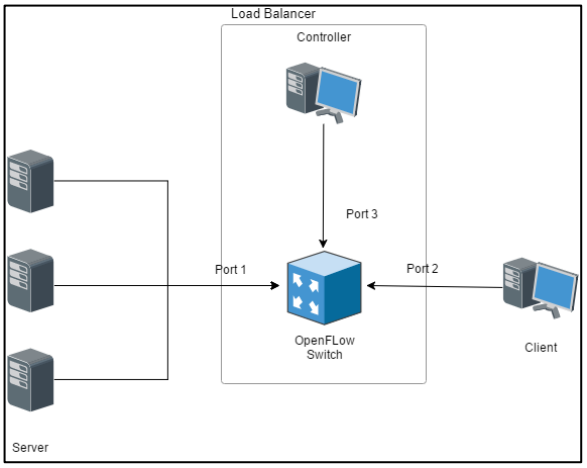
1. Komputer 1 sebagai *server* (virtual *server*)
2. Komputer 2 sebagai *controller*
3. Komputer 3 sebagai *switch*
4. Komputer 4 sebagai *client*

3.2.2. Kebutuhan Perangkat Lunak

Perangkat lunak mempunyai peran sebagai media interaksi yang menghubngkan perangkat keras dengan pengguna komputer. Adapun perangkat lunak yang digunakan dalam penelitian ini adalah.

1. Sistem operasi *ubuntu*
2. *Open vSwitch*
3. *Virtual Box*
4. *Text editor*
5. *Python*
6. *Browser*

3.3. Perancangan Sistem



Gambar 2. Diagram Perancangan Sistem

Gambar 2 merupakan gambaran sebuah sistem dimana di dalam sistem tersebut mengimplementasikan aritektur *SDN* dengan menggunakan tiga buah *server* yang telah terkonfigurasi, satu buah *SDN controller* dan satu buah *SDN switch* dan satu buah *client*. *Server* yang digunakan adalah tiga *server virtual* dengan jumlah *core* yang berbeda. Untuk *switch* yang digunakan adalah *switch vitual (OpenvSwitch)* yang di implementasikan pada sebuah *PC.* Untuk jenis *controller* sendiri yang digunakan adalah *POX controller*. Sedangkan untuk *client* menggunakan 1 *PC* yang menggunakan *tool Httperf* untuk melakukan *request.*

3.4. Implementasi Sistem

1. Konfigurasi perangkat *switch*

Switch yang digunakan adalah *Open vSwitch* yang di *install* pada sebuah komputer. Setelah *Open vSwitch* terinstal, dilakukan penambahan *bridge* yang bekerja sebagai *virtual switch* yang meneruskan paket sesuai dengan *flow-tabel* yang diberikan. Kemudian dilakukan pengaturan *IP address* 192.168.1.20. Selanjutnya dapat melakukan konfigurasi *Openflow* dan juga dapat menambahkan *IP* yang digunakan *controller* pada *switch.* Setelah proses konfigurasi *bridge* selesai selanjutnya dapat melakukan penambahan *port* yang akan digunakan oleh *switch* untuk meneruskan paket.

2. Konfigurasi perangkat *controller*

Konfigurasi yang dilakukan pada *controller* adalah melakukan pengesetan *IP* yaitu 192.168.1.10. *Controller* harus memiliki library *Python* dan *POX*, sehingga *controller* mampu

menjalankan proses *load balancing*
.

3. Konfigurasi perangkat *server*

Server pada penelitian ini menggunakan *server* virtual yaitu menggunakan *Virtual Box* sebagai pengganti *server real*. Dilakukan konfigurasi sebagai berikut.

- Jumlah core pada tiap *server* di atur berbeda. *Server* 1 dengan *core* 4, *server* 2 dengan *core* 2 dan *server* 3 dengan *core* 3.
- Melakukan set alamat IP dengan 3 *server* dengan cara yang sama dengan alamat *IP* :
  - 192.168.1.11 , alamat *IP Server* 1
  - 192.168.1.12 , alamat *IP Server* 2
  - 192.168.1.13 , alamat *IP Server* 3
- Implementasi metode *load balancing* dalam bentuk kode program pada *controller POX*

3.5. Pengujian dan Analisis

Skenario pengujian peratama kali dilakukan dengan mencari besar *rate* maksimal yang mampu ditangani sistem. Pada proses mencari maksimal *rate* dilakukan *request* menggunakan *httperf* dimula dengan *rate* dengan nilai kecil dibawah 100, hingga ditemukan maksimal *rate*. Nilai *rate* maksimal yang didapatkan yaitu 360 *req/s.* Selanjutnya nilai *rate* dikategorikan menjadi 3 kategori yaitu *low* , *medium* , dan *high*. Nilai *high* adalah 360 *req/s*, untuk nilai *medium* adalah setengah dari nilai *high* yaitu 180 *req/s*, dan untuk nilai *low* adalah setengah dari nilai *medium* yaitu 90 *req/s*. Pengujian dari masing-masing *rate* dilakukan dalam waktu 10 detik, kemudian didapatkan banyak koneksi untuk skenario ini yaitu :

- *Low* : 90 *req/s* (*rate*) dengan jumlah koneksi 900.
- *Medium* : 180 *req/s* (*rate*) dengan jumlah koneksi 1800.
- *High*: 360 *req/s* (*rate*) dengan jumlah konesi 3600.

Parameter pengujian yang digunakan dalam penelitian ini adalah *throughput* dan *CPU usage*. Skenario pengujian *throughput* dan *CPU usage* dilakukan dengan kondisi setiap *server* menjalankan vidio yang sama untuk memberikan beban pada masing-masing *server*. *Client* melakukan *request* menggunakan *httperf* sesuai dengan kategori *rate*. Setelah dilakukan *request* dari masing-masing kategori *rate*, maka didapatkan nilai *throughput dan CPU usage*. Hasil pengujian *CPU usage* didapatkan dengan menjalankan *Agent Psutils* pada masing masing *server*.

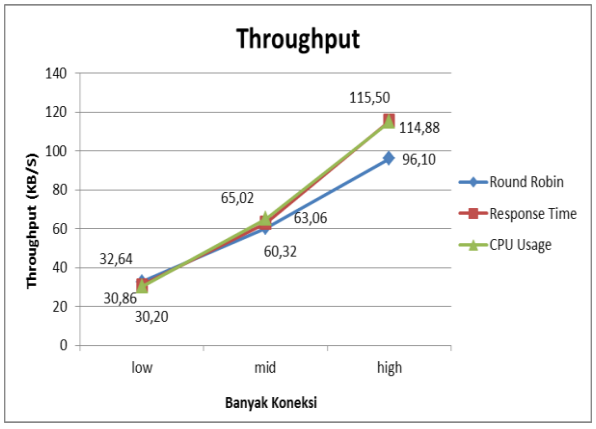
3.6. Kesimpulan

Pengambilan kesimpulan dilakukan setelah semua tahapan perancangan, implemetasi, dan pengujian sistem telah selesai dilakukan. Kesimpulan diambil dari hasil pengujian dan analisis terhadap penelitian yang telah dilakukan. Tahap terakhir penulisan adalah saran dan yang dimaksudkan untuk memberikan pertimbangan atas pengembangan penelitan yang lebih lanjut.

4. Hasil dan Pembahasan

4.1. Pengujian *Throughput*

Proses pengujian *throughput* dapat dilihat dari nilai perbandingan *throughput* antara metode *round robin*, metode berbasis *response time* dan metode berbasis *CPU usage*. Satuan yang digunakan untuk *throughput* yaitu *KB/s*. Nilai tersebut merupakan jumlah total transfer data yang sukses yang dapat dilakukan. Berikut hasil yang diperoleh :



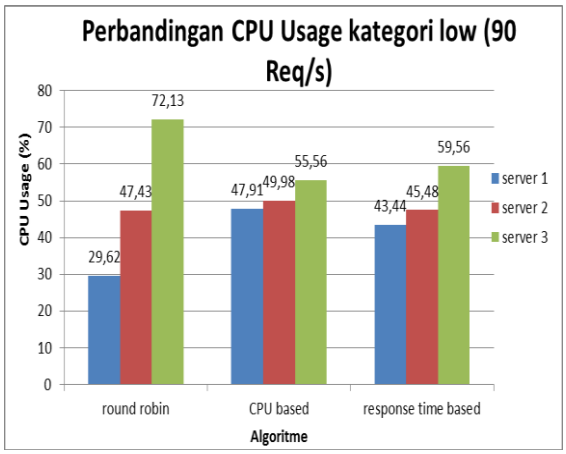
Gambar 3. Grafik Pengujian *Throughput*

Gambar 3 merupakan pengujian yang telah dilakukan. hasil yang didapatkan yaitu metode *round robin* memiliki nlai *throughput* unggul pada kategori *low*. Karena pada jumlah *request* yang rendah, kondisi *server* masih mampu melayani *request* dalam kondisi baik yang membuat metode *round robin* mempunyai *throughput* yang lebih unggul. Karena memang metode *round robin* membagikan beban pada tiga *server* secara bergantian. Tetapi semakin

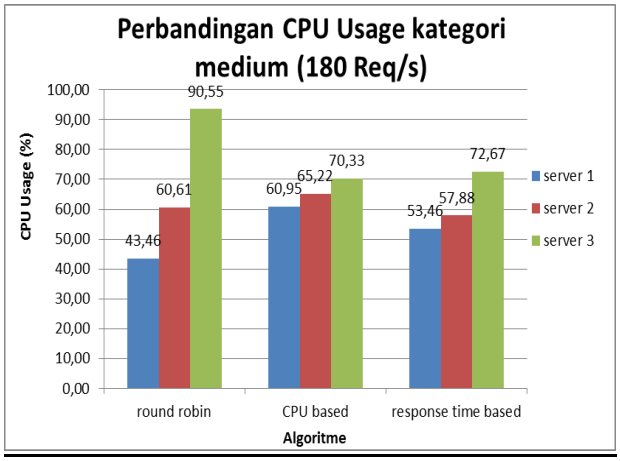
banyak *request*, *throughput* yang dihasilkan *round robin* akan lebih rendah dibandingkan metode berbasis *CPU* dan metode berbasis *response time*. bisa dilihat dari pengujian kategori *medium* dan *high* metode berbasis *response time* dan metode berbasis *CPU* lebih unggul. Hasil throughput yang didapatkan dari metode berbasis *response time* dan metode berbasis *CPU* tidak jauh beda, meskipun metode berbasis *response time* sedikit lebih unggul. karena memang metode berbasis *response time* merupakan representasi dari metode berbasis *CPU usage*, tetapi dilihat dari parameter yang berbeda. Jika pada metode berbasis *CPU usage* dilihat dari kondisi *CPU server* , tetapi pada metode berbasis *response time* dilihat dari parameter *response time* dari *server*. Karena kondisi *CPU* dari *server* juga akan mempengaruhi *response* time dari *server* tersebut.

4.2. Pengujian *CPU Usage*

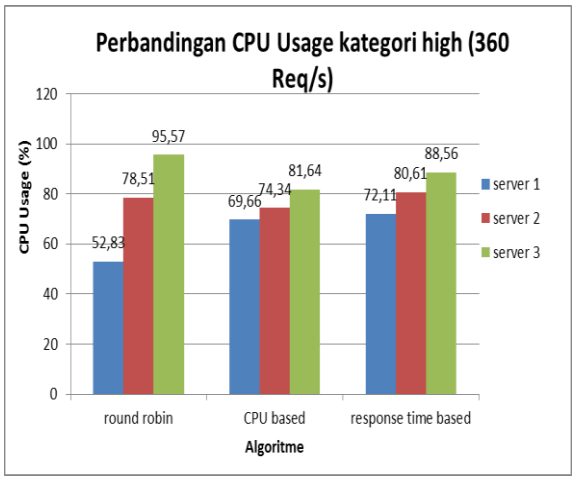
Proses pengujian *CPU usage* dapat dilihat dari nilai perbandingan *CPU usage* antara metode *round robin*, metode berbasis *response time* dan metode berbasis *CPU usage* . Satuan yang digunakan untuk *CPU usage* yaitu *%*. Berikut hasil yang diperoleh:



Gambar 4. Grafik Pengujian *CPU Usage* Kategori *Low*

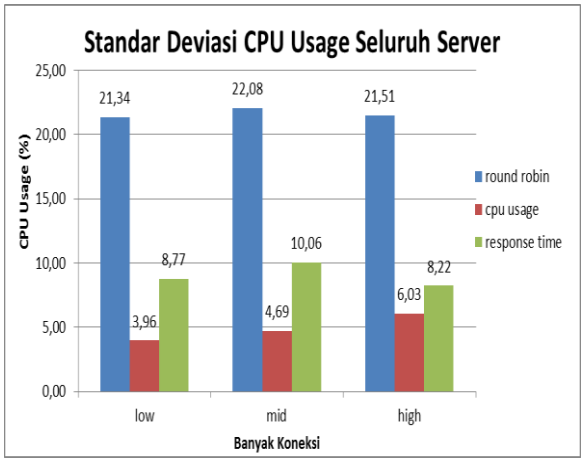


Gambar 5. Grafik Pengujian *CPU Usage* Kategori *Medium*



Gambar 6. Grafik Pengujian *CPU Usage* Kategori *High*

Gambar 4,5 dan 6 merupakan grafik hasil pengujian *CPU usage* setiap *server*. Bisa dilihat dari ketiga grafik tersebut bahwa metode *round robin* memiliki penggunaan *CPU* dengan rentan lumayan jauh dari ketiga *server*, karena metode *round robin* akan mendistribusikan *traffic* secara merata tanpa melihat kondisi dari server. Berbeda dengan metode berbasis *CPU usage* dan metode berbasis *response time* yang melihat kondisi dari *server* untuk pendistribusian *traffic*. Bisa dilihat dari semua kategori *rate*, untuk metode berbasis *CPU usage* dan metode berbasis *response time* selisih dari ketiga *server* tidak terlalu jauh seperti yang ada pada metode *round robin*. Dengan kata lain, *server* 1 dengan jumlah *core* 4 pada metode berbasis *CPU usage* dan pada metode berbasis *response time* mendapatkan pendistribusian *traffic* yang lebih banyak dibandingkan *sever* yang memiliki *jumlah core* 2 dan 1 yang memiliki *core* lebih rendah.



Gambar 7. Grafik Pengujian Standar Deviasi *CPU Usage*

Gambar 7 menunjukkan standar deviasi dari seluruh *server*. Pengujian dengan metode berbasis *CPU usage* memiliki nilai yang lebih kecil dan stabil. Hasil yang sama juga didapatkan oleh metode berbasis *response time*, yang tidak berbeda jauh dengan apa yang dihasilkan oleh metode berbasis *CPU*. Karena memang kondisi *CPU* dari *server* akan mempengaruhi *response time* dari *server*. jika dibandingkan dengan metode *round robin* yang yang memiliki nilai lebih besar dan cenderung tidak stabil. Hal ini dikarenakan memang *round robin* yang tidak melihat kondisi *server* sama sekali dalam pendistribusian *traffic*.

5. KESIMPULAN

Dari data yang didapatkan, bahwa kinerja dari masing-masing metode memiliki hasil yang berbeda. Pada pengujian *throughput*, metode *round robin* lebih unggul dibandingkan metode berbasis *CPU usage* dan metode berbasis *response time* hanya pada kategori *rate low*, dengan nilai rata-rata 32,64 KB/s. Karena pada pengujian *rate low* kondisi *server* masih mampu melayani *request* dengan baik. Sedangkan pada pengujian kategori *rate medium* dan *high* metode berbasis *response time* lebih unggul dibandingkan metode *round robin* dan metode berbais *CPU usage* dengan nilai rata-rata 65,02 KB/s pada *rate medium* dan 115,50 KB/s pada *rate high*. Meskipun, hasil yang didapatkan tidak jauh berbeda dengan metode berbasis *CPU*. Karena memang kedua metode ini akan melakukan *load balancing* dengan melihat kondisi dari *server*. Sedangkan pada metode *round robin* pada jumlah *request* yang tinggi membuat *server* yang memiliki jumlah *core* rendah akan mengalami kelebihan beban yang membuat *throughput* yang didapatkan lebih rendah dibandingkan dengan metode berbasis *CPU* dan metode berbasis *response time*.

Pada pengujian *CPU usage*, metode berbasis *CPU* memiliki pembagian beban yang paling seimbang. Nilai rata-rata yang didapatkan dari standar deviasi adalah 3,95% pada *rate low*, 4,69% pada *rate medium* dan 5,90% pada rate *high*. *Serve*r tidak akan mengalami kondisi kelebihan beban, karena *traffic* jaringan akan diarahkan pada *server* yang memiliki pemakaian *CPU* terkecil. Hasil yang tidak jauh beda juga didapatkan oleh metode berbasis *response time.* Karena memang metode berbasis *response time* merupakan representasi dari metode berbasis *CPU.* Dimana kondisi *CPU server* akan mempengaruhi *response time* dari *server*. Sedangkan metode *round robin* menjadi metode yang tidak seimbang, karena *round robin* tidak memperdulikan kondisi dari *server.* Pembagian akan dilakukan secara bergiliran yang menyebabkan kondisi *server* yang memiliki *core* kecil akan mengalami kelebihan beban, sedangkan *server* yang memiliki *core* besar masih memiliki penggunaan *CPU* yang rendah.

5. DAFTAR PUSTAKA

Arianto, E., & Sholeh, M. (2014). Implementation of Load Balancing Two Line ISP Using Router OS. *Jurnal JARKOM*.

Ellrod, C. (2010). *Load Balancing – Least Connection*. Retrieved Oktober 18, 2016, from https://www.citrix.com/blogs/2010/09/02/load-balancing-least-connections/

Julianto, R. (2017). Implementasi Load Balancing di Web Server Menggunakan Metode Berbasis Sumber daya CPU Pada Software Defined Networking. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*.

Kadir, A. (2003). *Pengenalan Sistem Informasi.* Andi Yogyakarta.

Kartadie, R., & Utami, E. (2014). Prototipe Infrasturkture Software Defined Network Dengan Protokol Openflow Menggunakan Ubuntu Sebagai Kontroler. *JURNAL DASI*.

Lara, A., & Kolasani, A. (2014). Network Innovation using OpenFlow: A Survey. *CSE Journal Articles*.

Mosberger, D., & Jin, T. (1998). httperf A Tool for Measuring.

Open Networking Foundation. (2009). OpenFlow Switch Specification. Retrieved Oketober 19, 2016, from https://www.opennetworking.org/sdn-resources/sdn-definition

Sirajuddin, & Affandi, A. (2012). Rancang Bangun Server Learning Management System. *JURNAL TEKNIK ITS*.

Ummah, I., & Abdillah, D. (2016). Perancangan Simulasi Jaringan Virtual Berbasis Software-Define Networking. *Ind. Journal on Computing, I*(1).

Zhong, H., & Fang, Y. (2016). An Efficient SDN Load Balancing Scheme Based on Server Response Time. *Future Generation Computer Systems*.

Psutil Documentation. (2018, Maret 5). Retrieved from psutil documentation: http://psutil.readthedocs.io/en/latest/

Moniruzzaman, M. (2015). A High Availability Cluster Model Combined with Load Balancing and Shared Storage Technologi for Web Servers. *International Journal of Grid Distribution Computing*.