Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer e-ISSN: 2548-964X
Vol. 2, No. 11, November 2018, hlm. 4663-4670 http://j-ptiik.ub.ac.id

Implementasi *Load Balancing* untuk Kontroler *Software Defined Network*

Adi Iman Utama[1], Widhi Yahya[2], Dany Primanita Kartikasari[3]

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

Abstrak

Perkembangan kebutuhan jaringan tradisional yang semakin kompleks akan mempengaruhi tingginya biaya investasi dan *maintenance* perangkat keras sehingga dibutuhkan solusi untuk perkembangan tersebut dengan menggunakan *Software Defined Network (SDN)*. SDN merupakan sebuah konsep pemisahan antara *control plane* dan *data plane* pada perangkat jaringan seperti *router* dan *switch*, sedangkan pada jaringan tradisional *control plane* dan *data plane* ini digabung menjadi satu. Dengan menggunakan *single* kontroler pada konsep SDN didapatkan nilai *latency* dengan rata-rata 1000 hingga 1900 respon/detik dan *throughput* dengan rata-rata 50 hingga 300 flow/detik, nilai tersebut mengalami penurunan performa ketika jumlah *switch* yang terhubung dalam lingkungan jaringan bertambah. Dalam penelitian ini menerapkan *load balancing* untuk melakukan pembagian beban pada beberapa kontroler yang terhubung agar performa dari tiap kontroler dapat berjalan dengan maksimal. Kontroler yang digunakan dalam penelitian ini adalah POX sedangkan untuk mengimplementasikan *load balancing* kontroler akan digunakan HAproxy pada komputer server. Hasil pengujian menunjukkan bahwa seiring bertambahnya jumlah kontroler dengan diberikan *load balancing* akan memberikan hasil *latency* dan *throughput* yang semakin baik dilihat dari penggunaan 3 kontroler didapatkan nilai *latency* dengan rata-rata 2000 hingga 3300 respon/detik dan *throughput* dengan rata-rata 70 hingga 4300 flow/detik, tetapi seiring bertambahnya jumlah kontroler maka beban dari server *load balancing* akan semakin meningkat dilihat dari CPU dan Memori *Usage*.

Kata kunci: *Software Defined Network (SDN), Load balancing, HAproxy, kontroler*

Abstract

The development of increasingly complex needs of traditional networks will affect the high cost of investment and hardware maintenance so that required solutions to these developments by using Software Defined Network (SDN). SDN is a concept of separation between control plane and data plane on network devices such as routers and switches, whereas in traditional networks the control plane and data plane are combined into one. Using a single controller on the SDN concept obtained a latency value with an average of 1000 to 1900 responses / second and throughput with an average of 50 to 300 flow / second, the value decreased performance when the number of switches connected in the network environment increased. In this research apply load balancing to do the load sharing on some controller that connected so that performance of each controller can run with maximal. The controller used in this research is POX while to implement load balancing controller will be used HAproxy on server computer. The test results show that as the increasing number of controllers with load balancing will provide better latency and throughput results from the use of 3 controllers obtained the latency value with an average of 2000 to 3300 responses / second and throughput with an average of 70 to 4300 flow / second, but as the number of controllers increases load of server load balancing will increase seen from CPU and Memory Usage.

Keywords: *Software Defined Network (SDN), Load balancing, HAproxy, controller*

1. PENDAHULUAN

Penggunaan perangkat jaringan dalam beberapa dekade ini seperti *router* dan *switch* telah digunakan dalam banyak lingkungan dan dapat menjalankan fungsi jaringan tradisional yaitu *filtering* dan *forwarding* paket menuju tujuan akhir. Walaupun perangkat jaringan ini sudah mengesankan, banyaknya perkembangan

baik dalam segi ukuran dan kompleksitas membuat jaringan tradisional ini berkurang performanya dalam mengelola lalu lintas data (Goransson, P. dan Black, C., 2014). Alasan untuk fakta diatas mencakup biaya yang semakin meningkat untuk memiliki dan mengoperasikan peralatan jaringan, kebutuhan untuk mempercepat inovasi jaringan dan peningkatan permintaan pusat data modern (Goransson, P. dan Black, C., 2014). Solusi untuk perkembangan dari kebutuhan jaringan yang semakin kompleks yaitu dengan penekanan biaya untuk investasi perangkat keras jaringan. Inovasi yang paling tepat digunakan adalah *Software Defined Network* (SDN).

Konsep *Software Defined Network* ini melakukan pemisahan antara *control plane* dan *data plane* pada perangkat jaringan seperti *router* dan *switch*. *Control plane* adalah bagian yang berfungsi untuk mengatur logika jaringan seperti algoritma *routing* dan tabel *routing*, sedangkan *data plane* adalah bagian yang berfungsi untuk mengatur bagaimana paket akan diteruskan menuju *next hop* berikutnya setelah paket tiba pada *forwarding engine* (Chao, H.J., dan Bin, L., 2007) sedangkan pada jaringan umum biasanya *control plane* dan *data plane* ini tergabung dalam satu perangkat. Pada SDN kebutuhan performa yang baik untuk kontroler sangat diperlukan, melihat kompleksitas jaringan yang semakin meningkat mengakibatkan beban kontroler pun semakin bertambah. Kompleksitas jaringan terjadi ketika jumlah switch dalam jaringan semakin meningkat sehingga dalam kondisi tersebut penggunakan satu kontroler bukanlah pilihan yang tepat karena akan menghasilkan nilai *latency* yang tinggi untuk beberapa *flows*, hal tersebut disebabkan karena beban kerja yang tinggi pada satu kontroler yang tersedia (Vinayagamurthy, D., Balasundaram, J., 2012).

Dilihat dari permasalahan tersebut, maka penambahan jumlah kontroler dengan melakukan mekanisme *load balancing* dapat diterapkan untuk membagi rata beban kerja yang akan ditangani oleh masing-masing kontroler. Penerapan *load balancing* sangat membantu dalam permasalahan ini, dimana tujuan *load balancing* adalah memungkinkan pembagian beban menjadi seimbang (Guo, Z., Su, M., dan Duan, Z., 2014). Dengan menggunakan *load balancing* maka kinerja dari masing-masing kontroler akan menjadi seimbang dan *load balancing* ini dapat digunakan sebagai *backup* (*high availability*) jika ada salah satu kontroler yang mengalami kerusakan maka kontroler lainnya masih bisa melayani *traffic* jaringan yang sedang berjalan. *Load balancing* juga sudah banyak dikembangkan pada SDN, contohnya pada *web server* yang tujuannya untuk meringankan beban masing-masing *web server* dari *request client* tetapi masih menggunakan *single* kontroler.

Parameter keberhasilan dari *load balancing* kontroler pada SDN dapat dilihat berdasarkan *latency* dan *throughput* yang diberikan dari masing-masing kontroler. Dikatakan berhasil apabila nilai *latency* dan *throughput* dari penggunaan *load balancing* pada SDN ini lebih baik daripada *single* kontroler, namun parameter ini juga harus dilihat dari hasil CPU dan Memori *usage* pada server *load balancing* agar dapat diketahui juga seberapa besar beban yang diterima pada sisi server *load balancing*. Maka dari itu pada penelitian ini, performa dari server *load balancing* diukur berdasarkan parameter CPU dan Memori *usage*. Sedangkan performa dari tiap kontroler dapat diukur dengan menggunakanan parameter *flow setup rate* dan *flow setup delay*. *Flow setup rate* adalah banyak flow yang mampu direspon oleh kontroler, sedangkan *flow setup delay* adalah waktu yang dibutuhkan sebuah kontroler dalam memberikan respon (Vengainathan, B ., Anton, B., dan Vishwas, M., 2014).

Berdasarkan paparan di atas judul yang diambil dalam penelitian ini adalah "Implementasi *Load Balancing* Untuk Kontroler *Software Defined Network*". Kontroler yang digunakan pada penelitian ini adalah POX. POX adalah *platform open-source* yang menggunakan bahasa pemrograman *Python* serta memungkinkan proses perancangan dan pembangunan jaringan lebih cepat dan lebih umum dibandingkan pendahulunya NOX (Al-Shabibi, A., dan McCauley, M., 2015). Fokus dari penelitian ini adalah untuk mengetahui efek penerapan *load balancing* kontroler dalam menangani sebuah *traffic* pada jaringan dengan melakukan pengujian menggunakan parameter uji *flow setup rate* dan *flow setup delay* pada kontroler.

2. KAJIAN PUSTAKA

Kajian pustaka membahas penelitian yang telah ada dan yang diusulkan. Pada penelitian ini kajian pustaka diambil dari beberapa penelitian yang relevan yang pernah dilakukan. Dasar atau acuan yang berupa teori-teori atau temuan-

temuan melalui hasil berbagai penelitian sebelumnya merupakan hal yang sangat perlu dan dapat dijadikan sebagai data pendukung. Salah satu data pendukung yang menurut peneliti perlu untuk dijadikan bagian tersendiri adalah penelitian terdahulu yang relevan dengan permasalah yang sedang dibahas dalam penelitian ini.

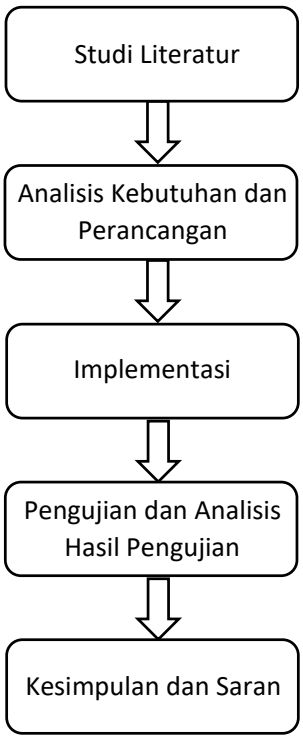
Penelitian pertama adalah penelitian yang dilakukan oleh Murdha dan Sawang (2015), fokus dari penelitian tersebut adalah bagaimana perbandingan performa yang dihasilkan dari penggunaan *single* server kontroler oleh dua kontroler berbeda yaitu POX dan Floodlight dimana pada dasarnya menggunakan bahasa pemrograman yang berbeda. POX dibangun dengan bahasa pemrograman Python dan berdasarkan arsitektur kontroler sebelumnya yaitu NOX, sedangkan Floodlight dibangun dengan bahasa pemrograman Java dan berdasarkan arsitektur kontroler Beacon.

Penelitian kedua adalah penelitian yang dilakukan oleh Mahdiyanah dan Naela (2016), fokus dari penelitian tersebut adalah bagaimana *load balancing* dengan menggunakan algoritma yang bersifat statis masih dapat berjalan dengan baik. Tujuan dari penelitian ini adalah agar *web server* dapat membagi bebannya secara optimal pada *Software Defined Network*. Hasil dari penelitian ini adalah *load balancing* dapat berjalan dengan baik dan membagi beban secara bergantian.

Penelitian ketiga adalah penelitian yang dilakukan oleh Jehn Ruey (2015) fokus dari penelitian tersebut adalah bagaimana penggunaan *load balancing* dengan *proposed algorithm* dibandingkan dengan *round-robin*, *randomized load balancing* dan LABERIO dibawah *Abilene network* dilihat dari *latency* dan *throughput*. Hasil yang didapatkan yaitu penggunaan *proposed algorithm* lebih unggul dibanding algoritma lainnya. Hal ini dapat dilihat dari path atau alur yang dipilih kontroler dan nilai *throughput* yang stabil dibandingkan algoritma lain.

3. METODOLOGI

Pada bab ini menjelaskan langkah-langkah yang akan dilakukan dalam penelitian Implementasi *Load Balancing* untuk Kontroler *Software Defined Network* yang terdiri dari beberapa tahapan yaitu studi literatur, analisis kebutuhan dan perancangan, implementasi, pengujian dan analisis hasil pengujian, hingga kesimpulan dan saran. Alur penelitian dapat dilihat pada Gambar 1



Gambar 1. Diagram Alir Tahapan Penelitian

Pada penelitian yang dilakukan, pertama akan dilakukan Studi literatur yang bertujuan untuk memperlajari serta memahami konsep-konsep sistem agar ketika dilakukan perancangan tidak terlalu mengalami kendala. Selanjutnya yaitu analisis kebutuhan dan perancangan dengan tujuan agar mengetahui hal-hal yang diperlukan dalam pengembangan sistem untuk menghindari penggunaan sumberdaya yang tidak perlu dan juga agar penelitian yang akan dilakukan dapat berjalan dengan baik dan sistematis.

Setelah analisis kebutuhan dan perancangan dilakukan, selanjutnya adalah tahap implementasi. Implementasi dengan mengacu pada perancangan yang dibuat sebelumnya. Selanjutnya adalah tahap pengujian dan analisis hasil pengujian yang bertujuan untuk mengetahui performa yang dihasilkan dari *load balancing* kontroler setelah diterapkan pada sebuah lingkungan jaringan dengan parameter yang sudah ditentukan.

Setelah semua tahapan selesai dilakukan maka dilakukan pengambilan kesimpulan dan saran, kesimpulan akan diambil berdasarkan hasil pengujian dan analisis terhadap sistem yang dibangun.

4. PEMBAHASAN

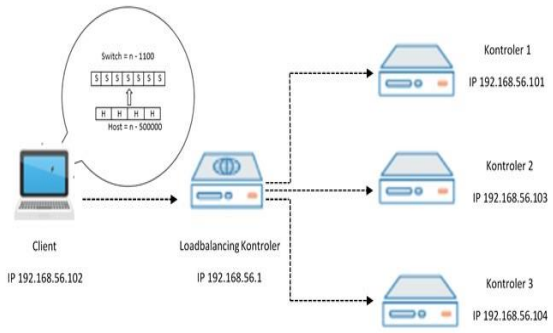
Pada pembahasan ini akan menjelaskan bagaimana proses perancangan dibuat dan bagaimana perancangan yang dibuat tersebut diimplementasikan pada sebuah lingkungan jaringan untuk dilakukan pengujian *load balancing* kontroler pada penelitian ini.

4.1. Perancangan Arsitektur Jaringan

Perancangan arsitektur jaringan dibutuhkan terkait dengan komunikasi antara mesin yang digunakan. Arsitektur jaringan yang digunakan adalah model komunikasi client-server, oleh karena itu dalam perancangan ini akan dijelaskan gambaran mesin yang digunakan pada client-server.

Untuk setiap kontroler akan dibuat dalam *virtual machine* sehingga dibutuhkan 3 VM yang masing-masing berisi satu kontroler POX. *Load balancing server* dibuat dalam host PC sedangkan untuk *client* yang berisi simulator Cbench akan dibuat pada VM yang berbeda dengan kontroler.

Didalam sistem nantinya gambaran terkait dengan peranan dari masing-masing perangkat yang terhubung didalam sistem dapat dilihat pada Gambar 2 berikut:



Gambar 2. Rancangan peran tiap mesin dalam sistem

Pada client, terdapat simulator cbench yang berisi virtualisasi host dan *switch* yang jumlahnya dapat dimanipulasi sesuai dengan kebutuhan pengujian. Client akan terhubung ke *load balancing server* yang menjadi jembatan untuk menuju ke ketiga kontroler POX yang tersedia. Pembagian beban kerja akan disesuaikan dengan algoritma Round robin yang digunakan dalam penelitian ini.

4.2. Implementasi Arsitektur Jaringan

Arsitektur jaringan yang digunakan adalah model komunikasi client-server. Banyak server yang diperlukan dalam pengujian ini sejumlah 3 server yang dibangun pada *Virtual Machine* yang masing-masing berisi kontroler POX yang telah selesai dikonfigurasikan. Ketiga server ini nantinya akan terhubung dengan HAproxy sebagai *loadbalancing* server yang dikonfigurasikan pada host PC, oleh karena itu konfigurasi pengalamatan IP diperlukan untuk memberikan identitas tiap server agar dapat diakses oleh satu sama lain.

Setelah semua alamat IP mesin telah dirubah dan tersimpan, maka langkah selanjutnya adalah melakukan pengujian dan menganalisa hasil dari pengujian tersebut.

5. PENGUJIAN DAN ANALISA

Terdapat 5 model pengujian yang akan dilakukan pada *loadbalancing* kontroler POX yaitu pengujian *flow setup delay* dan pengujian *flow setup rate*, dimana secara umum urutan pengujian yang akan dilakukan seperti pada Tabel 1 berikut:

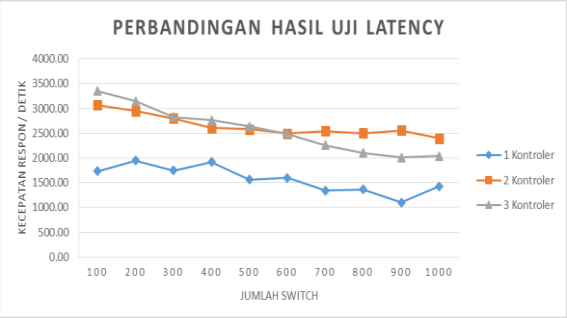
Tabel 1. Urutan Pengujian

No	Jumlah Switch dan Host	Pengujian
1	Switch Variasi dan Host Tetap	Uji Latency 1 Kontroler POX
		Uji Latency 2 kontroler POX
		Uji Latency 3 kontroler POX
2	Switch Tetap dan Host Variasi	Uji Latency 1 Kontroler POX
		Uji Latency 2 Kontroler POX
		Uji Latency 3 Kontroler POX
3	Switch Tetap dan Host Variasi	Uji Throughput 1 Kontroler POX
		Uji Throughput 2 Kontroler POX
		Uji Throughput 3 Kontroler POX
4	Switch Variasi dan Host Tetap	Uji Throughput 1 Kontroler POX
		Uji Throughput 2 Kontroler POX
		Uji Throughput 3 Kontroler POX
5	Switch Tetap dan	Uji CPU dan Memori

Host Variasi	usage

5.1. Pengujian Latency dengan variasi Switch

Hasil uji *latency* dari setiap variasi jumlah *loadbalancing* kontroler POX dan variasi *switch* memberikan nilai yang berbeda-beda antara satu sama lain. Perbandingan hasil uji dapat dilihat pada diagram grafik pada Gambar 3 berikut.

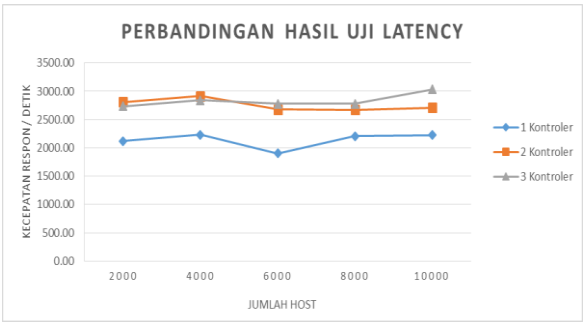


Gambar 3. Grafik analisis perbandingan hasil uji *latency*

Dilihat dari perbandingan yang disajikan pada Gambar 3 diatas maka untuk 1 kontroler cenderung memiliki nilai respon yang tidak terlalu tinggi dibandingkan dengan 2 dan 3 kontroler POX. Nilai respon paling tinggi ada pada 3 kontroler POX dengan kisaran respon antara 1.700 hingga 3.600, tetapi ada penurunan tingkat respon diatas 90 *switch*. Sama seperti 2 kontroler POX, ketika sudah melewati 90 *switch* respon yang diberikan cenderung menurun namun untuk jumlah 600 *switch* hingga 1000 *switch*, respon yang diberikan 2 kontroler POX ini jauh lebih stabil dan tidak menurun banyak dibandingkan dengan 3 kontroler POX dalam rentang jumlah *switch* yang sama. Penambahan jumlah *switch* sangat berpengaruh dengan performa kontroler, karena cenderung membuat performa kontroler menjadi menurun jika dilakukan penambahan jumlah *switch*.

5.2. Pengujian Latency dengan variasi Host

Hasil uji *latency* dari setiap variasi jumlah *loadbalancing* kontroler POX dan variasi host memberikan nilai yang berbeda-beda antara satu sama lain. Perbandingan hasil uji dapat dilihat pada diagram grafik pada Gambar 4 berikut.

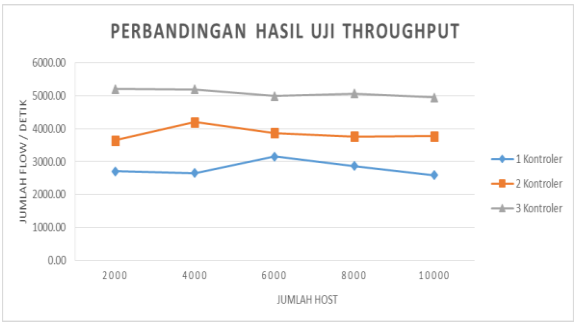


Gambar 4. Grafik analisis perbandingan hasil uji *latency*

Dilihat dari perbandingan yang disajikan pada Gambar 4 diatas maka untuk 1 kontroler cenderung memiliki nilai respon yang tidak terlalu tinggi dibandingkan dengan 2 dan 3 kontroler POX. Nilai respon paling tinggi ada pada 3 kontroler POX dengan kisaran respon rata-rata antara 2.200 hingga 3.000. Sedangkan untuk 2 kontroler POX walaupun respon rata-ratanya ada dibawah 3 kontroler namun respon yang diberikan 2 kontroler POX ini jauh lebih stabil dibandingkan dengan 3 kontroler POX dalam rentang jumlah host yang sama. Penambahan jumlah host tidak terlalu berpengaruh dengan performa kontroler, karena cenderung stabil walaupun terkadang nilainya naik turun dan tidak ada nilai penurunan yang konstan.

5.3. Pengujian Throughput dengan variasi Host

Hasil uji *throughput* dari setiap variasi jumlah *loadbalancing* kontroler POX dan variasi host memberikan nilai yang berbeda-beda antara satu sama lain. Perbandingan hasil uji dapat dilihat pada diagram grafik pada Gambar 5 berikut.



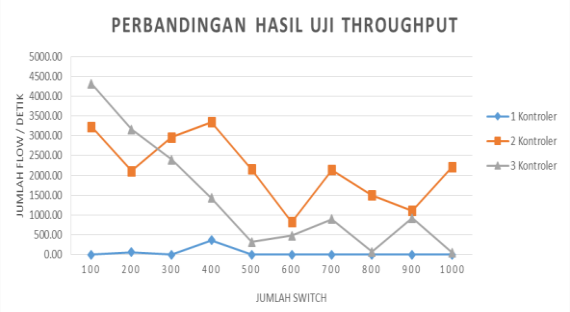
Gambar 5. Grafik analisis perbandingan hasil uji *throughput*

Dilihat dari perbandingan yang disajikan pada Gambar 5 diatas maka untuk 1 kontroler cenderung memiliki jumlah flow yang dapat

ditangani tidak terlalu tinggi dibandingkan dengan 2 dan 3 kontroler POX. Jumlah flow paling tinggi yang dapat ditangani ada pada 3 kontroler POX dengan kisaran jumlah flow antara 4.600 hingga 5.500 flow per detik. Dari grafik juga sudah dapat disimpulkan bahwa penambahan kontroler POX akan berpengaruh terhadap jumlah flow yang dapat dikelola tiap detiknya oleh kontroler, semakin banyak kontroler yang digunakan maka akan semakin banyak pula flow yang dapat ditangani. Penambahan jumlah host tidak terlalu berpengaruh dengan performa kontroler, karena cenderung stabil walaupun terkadang nilainya naik turun dan tidak ada nilai penurunan yang konstan.

5.4. Pengujian Throughput dengan variasi Switch

Hasil uji *throughput* dari setiap variasi jumlah *loadbalancing* kontroler POX dan variasi *switch* memberikan nilai yang berbeda-beda antara satu sama lain. Perbandingan hasil uji dapat dilihat pada diagram grafik pada Gambar 6 berikut.

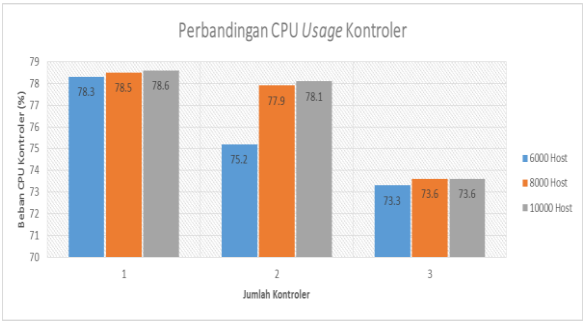


Gambar 6. Grafik analisis perbandingan hasil uji *throughput*

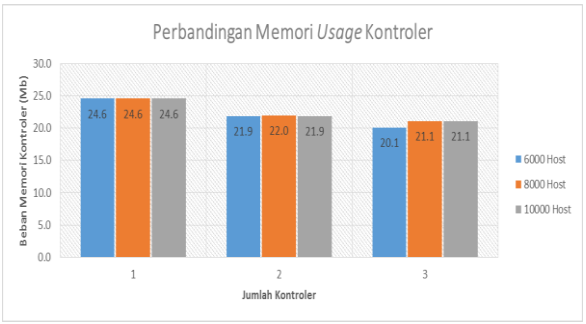
Dilihat dari perbandingan yang disajikan pada Gambar 6 diatas maka untuk 1 kontroler cenderung memiliki jumlah flow yang dapat ditangani tidak terlalu tinggi dibandingkan dengan 2 dan 3 kontroler POX. Jumlah flow paling tinggi yang dapat ditangani ada pada 3 kontroler POX dengan kisaran jumlah flow antara 29 hingga 4.000 flow per detik. Dari grafik juga dapat disimpulkan bahwa penambahan kontroler POX akan berpengaruh terhadap jumlah flow yang dapat dikelola tiap detiknya oleh kontroler, semakin banyak kontroler yang digunakan maka akan semakin banyak pula flow yang dapat ditangani. Penambahan jumlah *switch* sangat berpengaruh dengan performa kontroler, karena cenderung membuat performa kontroler menjadi menurun jika dilakukan penambahan jumlah *switch*.

5.5. Pengujian CPU dan Memori usage

Hasil Uji CPU dan Memori Usage dari setiap variasi penggunaan kontroler pada *loadbalancing* kontroler memberikan hasil yang berbeda-beda antara satu sama lain. Dari sini dapat dilihat perbandingan beban dari CPU dan Memori ketika sistem dijalankan. Perbandingan hasil uji CPU dan Memory kontroler *Usage* dapat dilihat pada diagram grafik berikut.



Gambar 7. Grafik perbandingan hasil uji CPU kontroler *usage*

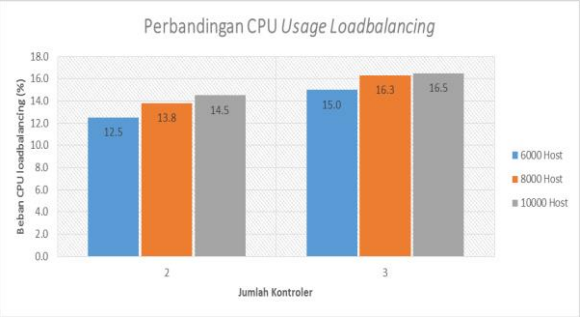


Gambar 8. Grafik perbandingan hasil uji Memory kontroler *usage*

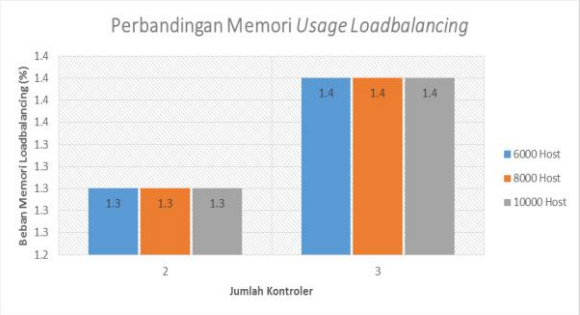
Dilihat pada Gambar 7 dan Gambar 8 maka 3 kontroler jauh lebih unggul dibandingkan penggunaan *single* kontroler dan 2 kontroler. Ini disebabkan karena *loadbalancing* akan membagi beban secara merata, sehingga jika semakin banyak server yang digunakan maka CPU dan memori *usage* dari setiap kontroler akan semakin ringan. Dari perbandingan CPU *usage* pada batas bawah dengan 6000 host, beban *single* kontroler mencapai 78.3% sedangkan pada 3 kontroler beban menurun menjadi 73.3%, begitu pula pada batas atas dengan 10000 host, beban *single* kontroler mencapai 78.6% sedangkan pada 3 kontroler beban menurun menjadi 73.6%. Selanjutnya dilihat dari perbandingan memori *usage* pada batas bawah dengan 6000 host,

beban dari *single* kontroler mencapai 24.6 Mb sedangkan pada 3 kontroler beban menurun menjadi 20.1 Mb, begitu pula pada atas atas dengan 10000 host, beban *single* kontroler mencapai 24.6 Mb sedangkan pada 3 kontroler beban menurun menjadi 21.1 Mb.

Dalam pengujian ini juga ditunjukkan perbandingan dari CPU dan Memori usage dari server *loadbalancing*, perbandingan hasil uji dapat dilihat pada Gambar 10 dan Gambar 11 berikut.



Gambar 9. Grafik perbandingan hasil uji CPU *loadbalancing usage*



Gambar 10. Grafik perbandingan hasil uji Memory *loadbalancing usage*

Dilihat pada Gambar 9 dan Gambar 10 perbandingan CPU *usage* server *loadblancing*, semakin banyak jumlah *loadbalancing* kontroler yang digunakan maka CPU *usage* pada *loadbalancing* server cenderung meningkat dimana CPU *usage* pada batas bawah dengan 6000 host untuk 2 server adalah 12.5% sedangkan untuk 3 server beban meningkat menjadi 15.0%. Jika dilihat pada batas atas dengan 10000 host untuk 2 server adalah 14.5% sedangkan untuk 3 server beban meningkat menjadi 16.5%, hal ini disebabkan karena kontroler yang ditangani oleh server *loadbalancing* semakin banyak sehingga beban dari CPU akan meningkat seiring dengan bertambahnya server yang ditangani. Begitu juga dengan memori *usage* dimana dengan jumlah kontroler sebanyak 2, beban dari server *loadbalancing* berkisar pada 1.3Mb sedangkan dengan jumlah 3 kontroler yang ditangani beban meningkat menjadi 1.4Mb pada semua batas.

6. KESIMPULAN

Penerapan *load balancing* pada kontroler Software Defined Network digunakan untuk membagi beban antara kontroler yang menangani request dari client yang terhubung dalam jaringan. Pembagian beban berdampak pada performa kontroler POX dalam menangani request. Terbukti dari pengujian CPU dan Memori *Usage*, dimana seiring bertambahnya jumlah server kontroler maka kinerja tiap kontroler akan semakin ringan. Tetapi seiring bertambahnya kontroler maka beban dari *load balancing* server justru meningkat.

Berdasarkan serangkaian pengujian dan analisis yang telah dilakukan, *load balancing* pada kontroler POX akan memberikan performa terbaik dalam waktu respon ketika menggunakan 3 kontroler. Peningkatan respon ini diakibatkan dari penggunaan *loadbalancing* pada 2 dan 3 kontroler, sehingga performa yang didapatkan semakin meningkat seiring bertambahnya kontroler yang digunakan.

Berdasarkan serangkaian pengujian dan analisis yang telah dilakukan, *load balancing* pada kontroler POX akan memberikan performa terbaik dalam menangani flow ketika menggunakan 3 kontroler. Peningkatan jumlah flow yang dapat ditangani ini diakibatkan dari penggunaan *loadbalancing* pada 2 dan 3 kontroler, sehingga performa yang didapatkan semakin meningkat seiring bertambahnya kontroler yang digunakan.

DAFTAR PUSTAKA

Al-Shabibi, A., dan McCauley, M., 2015. POX Wiki. [online] Tersedia di : https://openflow.stanford.edu/display/ONL/POX+Wiki#POXWiki-Requirements [Diakses 5 Oktober 2017].

Chao, H.J., dan Bin, L., 2007. High Performance Switches And Routers. Willey Interscience : A John Willey & Sons Inc.

Goransson, P. dan Black, C., 2014. Software Defined Network : A Comprehensive Approach. Waltham, USA : Elsevier Inc.

Guo, Z., Su, M., dan Duan, Z., 2014. Improving The Performance of Load Balancing in Software Defined Network Through Load

Variance Based Synchronization. USA : Elsevier Inc.

Vengainathan, B., Anton, B., dan Vishwas, M., 2014. Benchmarking Methodology for OpenFlow SDN. [online] Tersedia di : <https://tools.ietf.org/html/draft-bhuvan-bmwg-of-controller-benchmarking-00> [Diakses 6 Maret 2017].

Vinayagamurthy, D., Balasundaram, J., 2012. Load Balancing between Controllers. University of Toronto, Canada : Department of Computer Science.