Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer e-ISSN: 2548-964X
Vol. 5, No. 1, Januari 2021, hlm. 115-123 http://j-ptiik.ub.ac.id

Implementasi *Load Balancing* Web Server Pada POX *Controller* Berbasis Penggunaan Memori Dengan Agen Psutil Pada *Software Defined Network*

Cindy Zefira Afiani[1], Heru Nurwarsito[2]

Program Studi Teknik Informatika, Fakultas Ilmu Komputer, Universitas Brawijaya
Email: [1]cindyzefira@gmail.com, [2]heru@ub.ac.id

Abstrak

Peningkatan *request* untuk memenuhi kebutuhan pengguna dalam mengakses internet terjadi seiring berkembangnya teknologi. Jumlah *request* yang meningkat juga diiringi oleh kenaikan jumlah trafik yang sangat tinggi yang dapat menyebabkan *overload*. Metode *load balancing* web server yang diimplementasikan pada arsitektur jaringan *software defined network* dapat menjadi solusi untuk meningkatkan kinerja dari server. Beberapa penelitian sebelumnya menerapkan *load balancing* web server dengan menggunakan algoritma *round robin*, *response time* dan algoritma berbasis *resource* server seperti CPU *usage* yang memiliki kinerja yang berbeda-beda. *Resource* server seperti memori dapat menjadi parameter dalam algoritma *load balancing.* Server dengan penggunaaan memori yang terkecil akan melayani *request* yang diminta oleh *client*. Algoritma berbasis penggunaan memori menggunakan agen psutil untuk mengetahui informasi memori pada server dan diterapkan dalam POX *controller*. Hasil dari penelitian ini adalah *request* dari *client* dikirimkan ke server dengan penggunaan memori terkecil. Kemudian dilakukan perbandingan dengan algoritma CPU *usage* dengan parameter *throughput*, *response time*, dan *memory utilization.* Pada pengujian dengan *request* koneksi terbesar sejumlah 1800 *request*, algoritma berbasis penggunaan memori memiliki nilai *throughput* yang lebih tinggi yaitu 29,44 KB/s dibandingan algoritma CPU *usage* yaitu 27,78 KB/s. Sedangkan algoritma berbasis penggunaan memori memiliki nilai *response time* yang lebih rendah yaitu 5,68 ms dibandingan algoritma CPU *usage* yaitu 5,88 ms. Pada server dengan kapasitas memori tertinggi, algoritma berbasis penggunaan memori memiliki nilai *memory utilization* yang lebih rendah yaitu 12,94% dibandingkan algoritma CPU *usage* yaitu 13,20%.

Kata kunci: *load balancing*, POX *controlller*, *agen psutil*, *software defined network*, *memory usage*

Abstract

Increased requests to fulfill the needs of the user in accessing the internet occur as technology develops. The increasing number of requests is also accompanied by a very high increase in traffic which can cause overload. Web server load balancing method which is implemented in the network architecture known as software defined network can be a solution to improve the performance of the server. Several previous studies had implemented web server load balancing using algorithms like round robin, response time, and server resource based algorithms such as CPU usage based algorithm which had different performances. Resources of a server such as memory can serve as a parameter in the load balancing algorithm. The server with the smallest memory usage will service requests which are requested by the client. Memory usage based algorithm uses psutil agent to find out memory information on the server and is implemented in POX controller. The result of this study is sending requests by clients to the server with the smallest memory usage. The memory usage based algorithm is compared with the CPU usage based algorithm in terms of parameters such as throughput, response time, and memory utilization. In testing with 1800 requests as the largest connection requests, the memory usage based algorithm has a higher throughput value of 29.44 KB/s compared to the CPU usage based algorithm, which is 27.78 KB/s. Meanwhile, the memory usage based algorithm has a lower response time value of 5.68 ms compared to the CPU usage based algorithm, which is 5.88 ms. At the server with the highest memory capacity, the memory usage based algorithm has a lower memory utilization value of 12.94% compared to the CPU usage based algorithm, which is 13.20%.

Keywords: *load balancing*, POX *controller*, *psutil*, *software defined network*, *memory usage*

1. PENDAHULUAN

Terdapat beberapa penelitian dengan menggunakan metode *load balancing* pada *software defined network* dengan hasil kinerja yang berbeda. Penelitian pertama dilakukan oleh Hong Zhong (2016) dengan menerapkan skema algoritma *load balancing* berdasarkan *response time* yang dimiliki oleh server dan menggunakan Floodlight *controller*. Sehingga dalam penelitian ini akan menerapkan *load balancing* menggunakan algoritma berbasis penggunaan memori dengan agen psutil untuk mengetahui informasi *resource* dalam server sehingga paket data akan dikirimkan ke *server* dengan penggunaan memori terkecil. Penelitian ini akan menggunakan POX *controller* serta Open vSwitch sebagai switch.

Penelitian kedua dilakukan oleh Dr. Mustafa (2017) menerapkan *load balancing* dengan menggunakan algoritma *round robin*, algoritma *least connection*, dan algoritma *least loaded efficiency* dengan menggunakan OPNET sebagai simulator. Hasil dari penelitian ini adalah algoritma *least loaded efficiency* memiliki kinerja CPU *utilization* lebih tinggi dibandingkan algoritma algoritma *round robin*, dan algoritma *least connection*. Pada penelitian ini, dalam metode *load balancing* diterapkan algoritma berbasis penggunaan memori dengan agen psutil yang akan mendistribusikan trafik ke server dengan penggunaan memori terkecil.

Penelitian ketiga dilakukan oleh Riski Julianto (2017) dengan menerapkan *load balancing* berbasis CPU *usage* dan diterapkan pada server heterogen yaitu kapasitas server yang berbeda. Hasil dari penelitian ini berupa *request* dikirimkan ke server yang memiliki nilai *core* yang besar dibandingan server yang memiliki nilai *core* yang lebih kecil. Berdasarkan penelitian tersebut, penelitian ini menggunakan metode *load balancing* web server berbasis penggunaan *resource* pada server yaitu memori. Distribusi *load* akan diarahkan ke server berdasarkan server mana yang memiliki penggunaan memori terkecil.

Penelitian keempat dilakukan oleh Muharrom Abdillah (2019) dengan menganalisis perbandingan kinerja menggunakan beberapa metode *load balancing* yang berbeda yaitu *load balancing* berbasis CPU, *load balancing* berbasis *response time* dan *load balancing* menggunakan algoritma *round robin* sebagai pembanding. Hasil dari penelitian ini berupa *load balancing* menggunakan algoritma *round robin* memiliki nilai *throughput* tertinggi, *load balancing* berbasis CPU memiliki pembagian beban lebih stabil dalam CPU *usage*, dan *load balancing* berbasis *response time* lebih unggul dalam pengujian *response time*. Sehingga pada penelitian ini ingin mengetahui hasil dan kinerja dari implementasi *load balancing* web server dengan parameter penggunaan memori.

Terakhir, penelitian kelima dilakukan oleh Muhammad Sholeh (2019) dengan menerapkan algoritma *least connection* dengan agen psutils untuk mengetahui nilai jumlah beban koneksi yang dimiliki oleh server. Hasil yang diperoleh penelitian tersebut adalah algoritma *least connection* dengan agen psutils berhasil mendistribusikan trafik berdasarkan jumlah koneksi yang dimiliki oleh server tersebut. Berdasarkan penelitian tesebut, maka penelitian ini ingin menggunakan algoritma berbasis penggunaan memori dengan agen psutil untuk mengetahui informasi *resource* pada setiap server dan mendistribusikan trafik menuju server dengan penggunaan memori terkecil.

2. LANDASAN KEPUSTAKAAN

2.1 *Software Defined Network*

Software defined network merupakan paradigma arsitektur baru dalam bidang jaringan komputer dengan karakteristik dinamis, *manageable*, *cost-effective*, dan *adaptable*, yang sangat ideal untuk kebutuhan aplikasi saat ini yang bersifat dinamis dan menyediakan *bandwidth* tinggi (Ummah & Abdillah, 2016). Pada saat ini, *software defined network* atau disebut dengan SDN digunakan sebagai solusi untuk mengubah keterbatasan infrastruktur jaringan. *Software defined network* memiliki mekanisme untuk memisahkan *data plane* dan *control plane* sehingga *network administrator* dapat memodifikasi protokol sepanjang perangkat jaringan. SDN juga menyediakan *network programmability* yang membantu untuk mengatasi sifat dinamis dari jaringan pada saat ini (Dumka, 2018).

2.2 OpenFlow

Komunikasi antara *control plane* dan *data plane* dilakukan dengan menggunakan protokol *interface* pemrograman aplikasi (API) *southbound interface* seperti protokol OpenFlow (Khondoker , 2018). OpenFlow merupakan protokol komunikasi *interface* standar pertama yang didefinisikan diantara *control* dan *forwarding layer* dalam arsitektur *software defined network* (McKeown, et al., 2008). Protokol OpenFlow mendefinisikan pesan spesifik dan format pesan ditukarkan antara *controller* dari *control plane* dan perangkat dari

data plane (Göransson & Black, 2014). OpenFlow menyediakan akses langsung dan manipulasi dari *forwarding plane* pada perangkat jaringan seperti *switch* dan *router*, baik *physical* dan *virtual* (*hypervisor based*).

2.3 OpenFlow Switch

Arsitektur *load balancing* terdiri dari jaringan *switch* OpenFlow dengan POX *controller* dan beberapa server yang terhubung ke port *switch* OpenFlow (Kaur, et al., 2015). Kecerdasan jaringan terpusat secara logis pada *controller* pada SDN yang memegang penuh gambaran kondisi dalam jaringan. Hasilnya, jaringan muncul dalam aplikasi dan *policy engine*s sebagai *logical* switch (Manggala, et al., 2015).

2.4 Open vSwitch

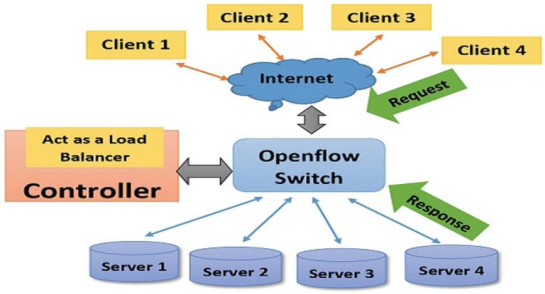
Open vSwitch merupakan perangkat lunak *multilayer switch* yang dilisensikan oleh *open source* Apache 2 (A Linux Foundation Collaborative Project, 2020). Open vSwitch merupakan perangkat yang sangat cocok sebagai virtual *switch* pada lingkungan *virtual machine*. Open vSwitch didesain untuk mendukung distribusi melalui beberapa *physical* server. Selain itu, Open vSwitch juga mendukung berbagai teknologi perangkat virtualisasi yang berbasis Linux seperti Xen/ XenServer, KVM, dan VirtualBox.

2.5 POX *Controller*

POX merupakan salah satu *controller* OpenFlow berbasis python dan secara luas digunakan oleh komunitas penelitian karena kemudahan pemrogramannya (Ali, et al., 2018). POX *controller* memiliki cara yang efisien untuk mengimplementasikan protokol OpenFlow. Untuk menghubungkan antara OpenFlow *controller* dan OpenFlow *logical switch* dapat menggunakan OpenFlow *channel* sebagai *interface*. *Controller* melakukan konfigurasi dan mengatur *switch*, menerima informasi dari *switch*, dan mengirimkan informasi paket ke *switch* melalui *interface* tersebut (Open Networking Foundation, 2014).

2.6 *Load Balancing*

Load balancing merupakan mekanisme untuk mendistribusikan beban trafik melalui dua atau lebih jalur koneksi secara seimbang, agar trafik dapat berjalan optimal, memaksimalkan nilai *throughput*, memperkecil nilai *response time* serta menghindari *overload* pada salah satu jalur koneksi (Sirajuddin, et al., 2012). *Load balancing* web server merupakan salah satu cara yang dapat digunakan untuk meningkatkan kinerja serta tingkat ketersediaan web server, yaitu dengan membagi *request* yang datang melalui beberapa web server sekaligus, sehingga beban yang ditanggung oleh setiap server lebih ringan (Lukitasari & Oklilas, 2010).



Gambar 1. *Load Balancing* pada *Software Defined Network*

Aplikasi *load balancer* terdiri dari sebuah algoritma yang digunakan dalam menentukan keputusan untuk memilih sebuah server dari server-server dan mendistribusikan beban secara bersamaan sesuai dengan *request* oleh *client* (Joshi & Gupta, 2019). *Load balancing* pada arsitektur jaringan *software defined network* diterapkan pada *controller*. Paket data yang diminta oleh *client* akan diterima oleh *switch*. Selanjutnya *switch* akan memeriksa informasi pada *flow table* untuk mengetahui apakah informasi *flow* tersedia atau tidak. Jika *flow* tersedia, maka *switch* hanya akan melakukan *update* pada *flow table* dan meneruskan paket tersebut ke server yang akan merespon paket tersebut. Sedangkan ketika *flow* tidak tersedia, maka *switch* akan mengirim pesan ke *controller*. Selanjutnya, *controller* yang menentukan keputusan yang akan diterapkan pada paket data tersebut berdasarkan algoritma yang diterapkan. Algoritma dalam *controller* akan menentukan server mana yang akan melayani permintaan dari *client*. Kemudian hasil keputusan oleh *controller* dikirimkan kembali ke *switch* sehingga *switch* akan melakukan *update* pada *flow table* dengan menyimpan informasi terbaru dan memproses paket dengan mengirimkannya ke server yang telah ditentukan.

2.7 Flask

Flask merupakan sebuah *web framework* yang menggunakan bahasa Python. Flask merupakan salah satu jenis *microframework* yang biasanya merupakan *framework* dengan karakteristik sedikit atau tanpa ketergantungan pada *library* eksternal (Pallets, 2020). Tidak seperti web *framework* yang lainnya, Flask tidak ditentukan oleh sebuah organisasi tertentu untuk proyek yang besar. Cara membangun aplikasi diserahkan secara penuh ke *developer*. Dengan menggunakan Flask dan bahasa *python*,

developer dapat membuat sebuah web yang terstruktur dan dapat mengatur *behaviour* suatu web dengan lebih mudah.

2.8 Agen Psutil

Psutil atau *python system and process utilities* merupakan *library* lintas-platform yang digunakan untuk mengambil informasi tentang proses yang berjalan dan pemanfaatan sistem (CPU, memori, disk, jaringan, sensor). Dengan menggunakan modul psutil pada python, didapatkan *cross-platform interface* yang sederhana antara python dan sistem untuk mengakses informasi sistem yang bervariasi. Library ini berguna terutama untuk memantau sistem, profil, serta membatasi sumber daya proses dan manajemen proses yang sedang berjalan (Rodola, 2020).

2.9 *Httperf*

Httperf merupakan *tool* untuk menguji kinerja dari web server. *Httperf* menyediakan fasilitas yang fleksibel dalam menghasilkan berbagai *load* HTTP dan untuk mengukur kinerja dari server (Mosberger & Jin, 1998). *Httperf* dapat mengadakan trafik yang dapat menghasilkan *request* untuk mengukur kinerja *load balancing* web server pada *software defined network*.

2.10 Algoritma Berbasis Penggunaan Memori

Algoritma yang diterapkan dalam metode *load balancing* web server ini menggunakan memori sebagai parameter *resource* pada server. Memori sebagai salah satu parameter *resource* pada server akan digunakan sebagai parameter untuk mengetahui kondisi dari *server*. Dengan mengetahui nilai penggunaan *resource* pada server yaitu memori, *request* akan dikirimkan server yang memiliki nilai penggunaan memori terkecil. Algoritma berbasis penggunaan memori yang diterapkan pada *load balancing* web server bertujuan untuk mengurangi resiko kegagalan server dalam melayani *request* dari *client*. Jumlah *client* dan lama waktu dari *client* mengakses sebuah *server* dapat mempengaruhi tingkat penggunaan CPU dan memori pada *server* (Gandhi, et al., 2002).

Algoritma berbasis penggunaan memori menggunakan agen psutil untuk mengetahui informasi *resource* pada setiap server. Agen psutil akan mengirimkan informasi tersebut ke *controller*. Kemudian, c*ontroller* akan membuat keputusan dalam mengirimkan *request* dengan melihat perbandingan kondisi penggunaan memori pada setiap server terlebih dahulu. Setelah didapatkan kondisi penggunaan memori, maka *server* yang memiliki nilai penggunaan memori yang terkecil akan menerima *request* dari *client.*

3. METODOLOGI PENELITIAN

3.1 REKAYASA KEBUTUHAN

3.1.1 Kebutuhan Fungsional

Kebutuhan fungsional dalam sistem ini meliputi:

1. Sistem *load balancing* dapat mengirimkan request ke web server dengan penggunaan memori terkecil.
2. Agen psutil pada web server dapat melihat informasi pada *resource* setiap web server.
3. *Client* dapat mengirimkan *request* yang akan direspon oleh web server yang telah ditentukan oleh metode *load balancing*.

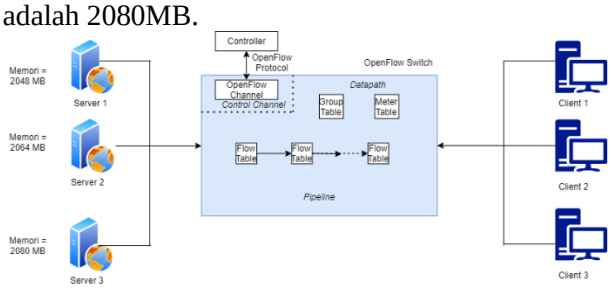
3.1.1 Kebutuhan Non-Fungsional

Kebutuhan non-fungsional terbagi atas kebutuhan perangkat keras dan perangkat lunak yang dibutuhkan sitem. Kebutuhan perangkat keras meliputi *processor* AMD Ryzen 5 2500U, 2.00 GHz, RAM sebesar 8 GB, dan storage 1 GB. Sedangkan kebutuhan perangkat lunak diantaranya yaitu Ubuntu 20.04 LTS, VirtualBox, OpenFlow, Open vSwitch, POX *controller,* flask, agen psutil, dan *httperf.*

3.2 PERANCANGAN

3.2.1 Perancangan Arsitektur Jaringan *Software Defined Network*

Perancangan topologi yang akan diterapkan pada sistem terdiri dari satu *controller*, satu *switch*, tiga web server, dan tiga *client*. Pada arsitektur jaringan ini menggunakan POX sebagai *controller*, Open vSwitch sebagai virtual *switch*, tiga server dengan spesifikasi *resource* memori yang berbeda, dan tiga *client*. Nilai *resource* memori pada server 1 adalah 2048MB, pada server 2 adalah 2064MB, dan pada server 3 adalah 2080MB.



Gambar 2. Perancangan Topologi Sistem

3.2.2 Perancangan *Controller*

Controller yang digunakan untuk *load balancing* web server dalam membangun arsitektur jaringan *software defined network* adalah POX *controller*. Pada penelitian ini, algoritma berbasis penggunaan memori pada metode *load balancing* web server akan

diterapkan pada *controller*. *Controller* akan dijalankan pada sistem operasi Ubuntu Desktop 20.04 LTS. *Controller* yang akan terkoneksi dengan Open vSwitch sebagai virtual *switch* memiliki IP *address* yaitu 192.168.56.100. Untuk menjalankan metode *load balancing* web server diperlukan sebuah IP virtual yaitu IP *service* dengan alamat yaitu 192.168.1.101. Penggunaan IP *service* bertujuan agar IP *address* asli dari *controller* tidak diketahui oleh siapapun selain *controller*.

3.2.3 Perancangan *Switch*

Switch yang digunakan adalah Open vSwitch yang mendukung protokol *software defined network* yaitu OpenFlow. Open vSwitch berfungsi untuk menerima paket yang datang dalam jaringan dan menangani paket tersebut berdasarkan informasi dalam *flow table* yang sudah ditentukan oleh *controller*.

Tabel 1. Spesifikasi Switch

Bridge	Interface	Port	Tujuan
br0	br0	Port br0	*Switch*
br0	-	-	*Controller*
br0	vnet1	Port vnet1	Server 1
br0	vnet2	Port vnet2	Server 2
br0	vnet3	Port vnet3	Server 3
br0	vnet4	Port vnet4	*Client* 1
br0	vnet5	Port vnet5	*Client* 2
br0	vnet6	Port vnet6	*Client* 3

3.2.4 Perancangan Server

Server berfungsi untuk menerima *request* dari *client*. Pada penelitian ini, *web server* yang digunakan adalah Flask.

Tabel 2. Spesifikasi Server

Spesifikasi	Server 1	Server 2	Server 3
Sistem Operasi	Ubuntu Server 16.04 LTS	Ubuntu Server 16.04 LTS	Ubuntu Server 16.04 LTS
Prosessor	1 core	1 core	1 core
Memori	2048 MB	2064 MB	2080 MB

3.2.5 Perancangan *Client*

Client dapat melakukan *request* pada server. Pada penelitian ini, *client* menggunakan *tool* yaitu *httperf*. Penggunaan *httperf* sendiri dilakukan pada saat melakukan pengujian berdasarkan skenario untuk mengetahui kinerja sistem berdasarkan nilai parameter pengujian yang didapat.

Tabel 3. Spesifikasi *Client*

Spesifikasi	*Client* 1	*Client* 2	*Client* 3
Sistem Operasi	Ubuntu Server 16.04 LTS	Ubuntu Server 16.04 LTS	Ubuntu Server 16.04 LTS
Prosessor	1 core	1 core	1 core
Memori	1024 MB	1024 MB	1024 MB

3.2.6 Alokasi IP *Address*

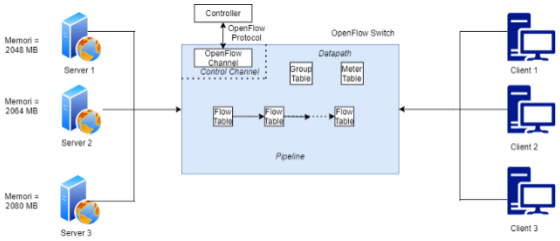
IP *address* yang digunakan merupakan *private network* dan seringkali digunakan sebagai IP *address* pada *consumer-grade router* yang umum untuk dihubungkan ke semua komputer dan perangkat yang dapat mengakses internet. IP *address* dengan cakupan 192.168.56.0/24 tergabung dalam class C dan memiliki jumlah maksimal 254 host sehingga cocok digunakan untuk membangun sistem yang sesuai dengan kebutuhan.

Tabel 4. Alokasi IP Address

Perangkat	IP Address
Server 1	192.168.56.10
Server 2	192.168.56.20
Server 3	192.168.56.30
Client 1	192.168.56.40
Client 2	192.168.56.50
Client 3	192.168.56.60

3.3 IMPLEMENTASI

Implementasi sistem *load balancing* web server pada arsitektur jaringan *software defined network* dilakukan dengan menggunakan POX sebagai *controller*, Open vSwitch sebagai virtual *switch*, tiga server yang diterapkan *web framework* flask sebagai web server, dan tiga *client*.



Gambar 3. Implementasi Sistem

Parameter memori sebagai parameter pada algoritma berbasis penggunaan memori, maka dalam sistem diterapkan dengan memberikan kapasitas memori yang berbeda pada setiap server. Untuk server 1, kapasitas memori sebesar 2048 MB. Pada server 2 kapasitas memori sebesar 2064 MB dan pada server 3 kapasitas memori adalah 2080 MB.

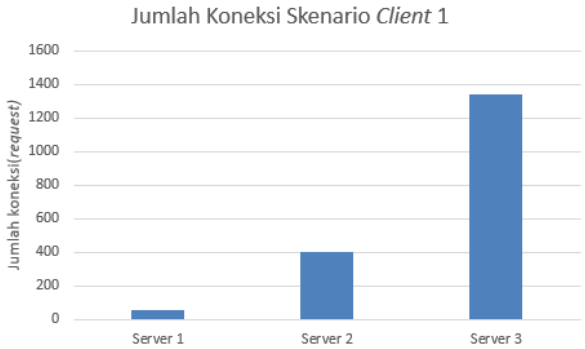
4. HASIL DAN PEMBAHASAN

4.1 Hasil Pengujian Fungsional

1. Hasil Pengujian Skenario *Client* 1

Tabel 5. Jumlah Koneksi pada Skenario *Client* 1

Nama	Jumlah Koneksi	Nilai *error*
Server-1	56	0%
Server-2	400	
Server-3	1344	
Total	1800	



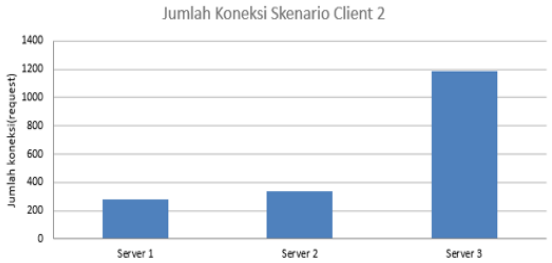
Gambar 4. Jumlah Koneksi pada Skenario *Client* 1

Nilai *error* bernilai 0% sehingga dapat disimpulkan *client* 1 berhasil mengirimkan semua *request* yang direspon oleh server yang melayaninya. Selain itu, jumlah *request* terbanyak dikirimkan ke server 3.

2. Hasil Pengujian Skenario *Client* 2

Tabel 6. Jumlah Koneksi pada Skenario Client 2

Nama	Jumlah Koneksi	Nilai *error*
Server-1	276	0%
Server-2	339	
Server-3	1185	
Total	1800	



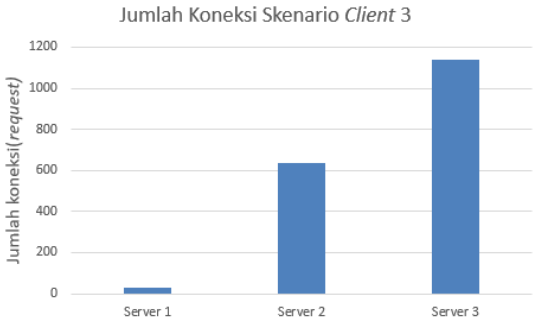
Gambar 5. Jumlah Koneksi pada Skenario *Client* 2

Nilai *error* bernilai 0% sama dengan hasil skenario sebelumnya sehingga dapat disimpulkan *client* 2 berhasil mengirimkan semua *request* yang direspon oleh server yang melayaninya. Server 3 juga menerima *request* dalam jumlah terbanyak.

3. Hasil Pengujian Skenario *Client* 3

Tabel 7. Jumlah Koneksi pada Skenario *Client* 3

Nama	Jumlah Koneksi	Nilai *error*
Server-1	27	0%
Server-2	635	
Server-3	1138	
Total	1800	



Gambar 6. Jumlah Koneksi pada Skenario Client 3

Sama seperti pengujian skenario sebelumnya, server 3 menerima *request* terbanyak dibandingkan dengan server lainnya. Nilai error bernilai 0% sama dengan hasil scenario sebelumnya sehingga dapat disimpulkan *client* 3 berhasil mengirimkan semua *request* yang direspon oleh server yang melayaninya.

4. Hasil Pengujian Skenario *Service Availability*

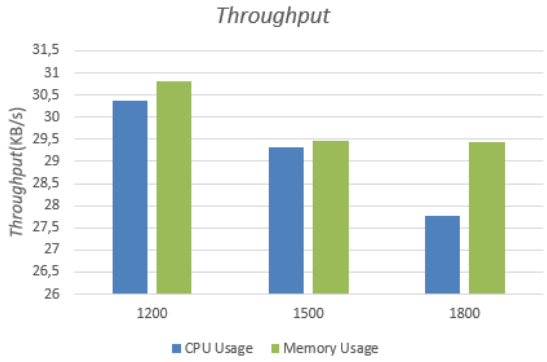
Tabel 8. Jumlah Koneksi pada Skenario *Service Availability*

Nama	Jumlah Koneksi	Nilai *error*
Server-2	256	0%
Server-3	1544	
Total	1800	

Nilai *error* bernilai 0% karena *client* 1 berhasil mengirimkan semua *request* yang direspon oleh server yang aktif. Jumlah *request* terbanyak dikirimkan ke server 3.

4.2 Hasil Pengujian Fungsional

4.2.1 Hasil Pengujian *Throughput*

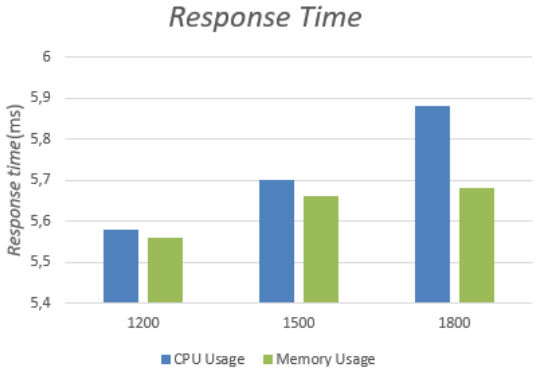


Gambar 7. Pengujian Throughput

Nilai *throughput* tertinggi pada algoritma CPU *usage* terdapat di pengujian dengan *request* koneksi terkecil yaitu 1200 *request* sebesar 30,36 KB/s. Sedangkan nilai *throughput* terendah terdapat pada pengujian dengan *request* koneksi terbesar yaitu 1800 *request* sebesar 27,78 KB/s. Konsep dari algoritma CPU *usage* adalah mengirimkan *request* ke server dengan penggunaan CPU terendah. Pada algoritma berbasis penggunaan memori, nilai *throughput*

tertinggi pada algoritma berbasis penggunaan memori terdapat di pengujian dengan *request* koneksi terkecil yaitu 1200 *request* sebesar 30,82 KB/s. Sedangkan nilai *throughput* terendah terdapat pada pengujian dengan *request* koneksi terbesar yaitu 1800 *request* sebesar 29,44 KB/s. Algoritma berbasis penggunaan memori mengirimkan *request* ke server dengan penggunaan memori terkecil. Algoritma berbasis penggunaan memori memiliki perfomansi kinerja *throughput* yang lebih baik daripada pada algoritma CPU *usage*.

4.2.2 Hasil Pengujian *Response Time*

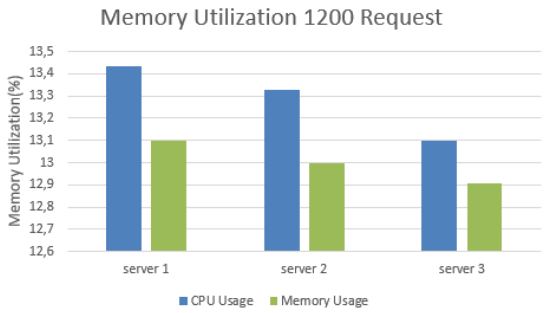


Gambar 8. Pengujian *Response Time*

Nilai *response time* terendah pada algoritma CPU *usage* terdapat di pengujian dengan *request* koneksi terkecil yaitu 1200 *request* sebesar 5,58 ms. Sedangkan nilai *response time* tertinggi terdapat pada pengujian dengan *request* koneksi terbesar yaitu 1800 *request* sebesar 5,88 ms. Algoritma CPU *usage* melakukan pengiriman *request* ke server dengan penggunaan CPU terendah sehingga jika terdapat salah satu server yang kelebihan beban maka respon akan lebih lambat. Pada algoritma berbasis penggunaan memori, nilai *response time* terendah pada algoritma berbasis penggunaan memori terdapat di pengujian dengan *request* koneksi terkecil yaitu 1200 *request* sebesar 5,56 ms. Sedangkan nilai *response time* tertinggi terdapat pada pengujian dengan *request* koneksi terbesar yaitu 1800 *request* sebesar 5,68 ms. Dengan menggunakan agen psutil untuk mengukur informasi kapasitas *resource* setiap server, algoritma berbasis memori dapat mengirimkan *request* ke server dengan kondisi penggunaan memori paling kecil. Algoritma berbasis penggunaan memori memiliki perfomansi kinerja *response time* yang lebih rendah daripada pada algoritma cpu *usage*.

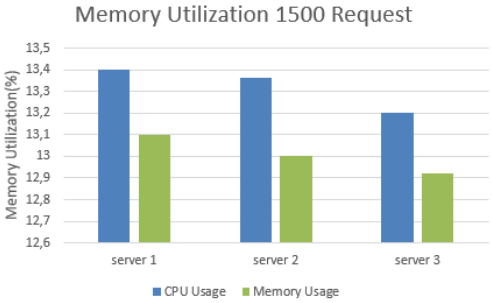
4.2.3 Hasil Pengujian *Memory Utilization*

1. Pengujian *Memory Utilization* dengan 1200 *Request*



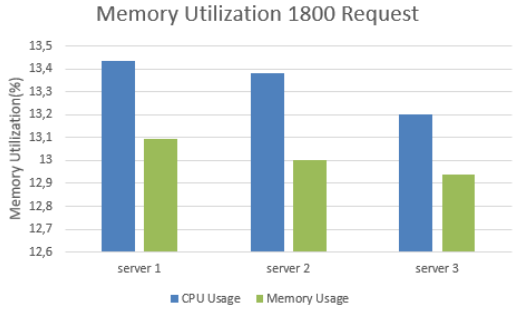
Gambar 9. Pengujian *Memory Utilization* dengan 1200 *Request*

2. Pengujian *Memory Utilization* dengan 1500 *Request*



Gambar 10. Pengujian *Memory Utilization* dengan 1500 *Request*

3. Pengujian *Memory Utilization* dengan 1800 *Request*



Gambar 11. Pengujian *Memory Utilization* dengan 1800 *Request*

Pada setiap pengujian menggunakan algoritma CPU *usage*, nilai *memory utilization* tertinggi terdapat pada server 1. Sedangkan nilai *memory utilization* terendah pada algoritma CPU *usage* terdapat pada server 3. Nilai *memory utilization* pada setiap server 3 berbeda pada setiap kategori *request* diantaranya pada 1200 *request* sebesar 13,10%, 1500 *request* sebesar 13,20%, 1800 *request* sebesar 13,201%. Pada algoritma CPU *usage*, *request* dikirimkan ke server dengan penggunaan CPU terendah.

Pada pengujian *memory utilization* pada algoritma berbasis penggunaan memori, nilai *memory utilization* tertinggi terdapat pada server 1. Sedangkan milai *memory utilization* terendah pada algoritma berbasis penggunaan memori terdapat pada server 3. Nilai *memory utilization*

pada setiap server 3 berbeda pada setiap kategori *request* diantaranya pada 1200 *request* sebesar 12,90%, 1500 *request* sebesar 12,91%, 1800 *request* sebesar 12,94%. Pada algoritma berbasis penggunaan memori, *request* dikirimkan ke server dengan penggunaan memori terkecil. Sehingga dapat disimpulkan algoritma berbasis penggunaan memori memiliki perfomansi kinerja *memory utilization* lebih baik daripada algoritma CPU *usage*.

5. KESIMPULAN

5.1 Kesimpulan

Implementasi *load balancing* web server pada POX *controller* berbasis penggunaan memori dengan agen psutil pada *software defined network* dapat berjalan dengan baik. Jumlah total *request* koneksi yang diminta oleh setiap *client* sama dengan total jumlah *request* koneksi yang berhasil direspon oleh server. Algoritma berbasis penggunaan memori berhasil mendistribusikan koneksi dari *client* ke *server* dengan penggunaan memori terkecil sehingga tidak terdapat nilai *error* dan dapat menghindari *overload*. Fungsi *service availability* juga berjalan dengan baik sehingga jika ada salah satu server yang mati, *request* koneksi dari *client* dapat langsung diarahkan ke server yang aktif.

Kinerja *load balancing* web server pada POX *controller* berbasis penggunaan memori dengan agen psutil pada *software defined network* berjalan dengan baik berdasarkan hasil kinerja server. Nilai *throughput* tertinggi terdapat pada request koneksi terkecil dengan rata-rata yaitu 30,82 KB/s karena kondisi server masih belum terlalu penuh. Kinerja nilai *response time* terendah terdapat pada *request* koneksi terkecil yaitu 5,56 ms karena semakin besar jumlah *request* maka nilai *response time* semakin tinggi. Nilai kinerja *memory utilization* terendah pada *request* koneksi terbesar terdapat pada server dengan kapasitas memori tertinggi *sebesar* 12,94%. Proses pengiriman *request* koneksi dari *client* berjalan optimal karena data dikirimkan ke server dengan penggunaan memori terkecil sesuai kebutuhan sistem.

Kinerja antara *load balancing* web server pada POX *controller* berbasis penggunaan memori dengan agen psutil pada *software defined network* lebih baik dibandingkan dengan *load balancing* web server menggunakan algoritma CPU *usage*. *Request* pada algoritma berbasis penggunaan memori dikirimkan ke server dengan penggunaan memori terkecil sehingga proses *load balancing* web server berjalan lebih optimal. Pada proses *request* koneksi terbesar, algoritma berbasis penggunaan memori memiliki nilai *throughput* yang lebih tinggi yaitu 29,44 KB/s dibandingan algoritma CPU *usage* yaitu 27,78 KB/s. Sedangkan algoritma berbasis penggunaan memori memiliki nilai *response time* yang lebih rendah yaitu 5,68 ms dibandingan algoritma CPU *usage* yaitu 5,88 ms. Pada server dengan kapasitas memori tertinggi, algoritma berbasis penggunaan memori memiliki nilai *memory utilization* yang lebih rendah yaitu 12,94% dibandingkan algoritma CPU *usage* yaitu 13,20%.

5.2 Saran

1. Menerapkan perbandingan dengan algoritma *load balancing* web server yang lainnya sehingga dapat mengetahui hasil kinerja.
2. Melakukan pengujian dengan parameter kinerja yang lain dan lebih beragam.
3. Melakukan implementasi algoritma *load balancing* web server menggunakan jenis *controller* yang lain.
4. Mengimplementasikan sistem *load balancing* pada lingkungan nyata atau *real*.

6. DAFTAR PUSTAKA

A Linux Foundation Collaborative Project, 2020. *OVS Open vSwitch.* [Online] Available at: http://www.openvswitch.org/ [Accessed 08 Juli 2020].

Abdillah, M., Yahya, W. & Basuki, A., 2019. Analisis Perbandingan Kinerja Metode Load Balancing Berbasis CPU Dengan Berbasis Response Time Pada Software Defined Network. *JPTIIK*, Volume 3, pp. 395-403.

Ali, J., Lee, S. & Roh, B.-h., 2018. Performance Analysis of POX and Ryu with Different SDN Topologies. *ICISS.*

Dumka, A., 2018. *Innovations in Software-Defined Networking and Network Functions Virtualization.* Hershey PA: IGI Global.

Gandhi, N. et al., 2002. MIMO Control of an Apache Web Server: Modeling and Controller Design. *Proceedingsof the American Control Conference.*

Göransson, P. & Black, C., 2014. *Software Defned Networks A Comprehensive Approach.* Waltham: Elsevier Inc.

Joshi, N. & Gupta, D., 2019. A Comparative Study on Load Balancing Algorithms in Software Defined Networking. *ICST.*

Julianto, R., Yahya, W. & Akbar, S. R., 2017.

Implementasi Load Balancing Di Web Server Menggunakan Metode Berbasis Sumber Daya CPU Pada Software Defined Networking. *JPTIIK*, Volume 1, pp. 904-914.

Kaur, S., Singh, J., Kumar, K. & Ghumman, N. S., 2015. Round-Robin Based Load Balancing in Software Defined Networking. *IEEE*.

Khondoker , R., 2018. *SDN and NFV Security*. Switzerland: Springer International Publishing AG.

Lukitasari, D. & Oklilas, A. F., 2010. Analisis Perbandingan Load Balancing Web Server Tunggal Dengan Web server Cluster Menggunakan Linux Virtual Server. *Jurnal Generic*, 5(2).

Manggala, A. W., H. & Tanwidjaja, A., 2015. Performance Analysis of White Box Switch on Software Defined Networking Using Open vSwitch. *IEEE*.

McKeown, N. et al., 2008. OpenFlow: Enabling Innovation in Campus Networks. *ACM SIGCOMM Computer Communication Review*, 38(2).

Mosberger, D. & Jin, T., 1998. *httperf - A Tool for Measuring Web Server Performance.* Palo Alto CA: s.n.

Open Networking Foundation, 2014. *OpenFlow Switch Specification.* [Online] Available at: www.opennetworking.org [Accessed 29 Juni 2020].

P., 2020. *Flask Documentation (1.1.x).* [Online] Available at: https://flask.palletsprojects.com/ [Accessed 25 Mei 2020].

Rodola, G., 2020. *psutil documentation.* [Online] Available at: https://psutil.readthedocs.io/en/latest/# [Accessed 8 April 2020].

Sirajuddin, Affandi, A. & Setijadi, E., 2012. Rancang Bangun Server Learning Management System Menggunakan Load Balancer dan Reverse Proxy. *Jurnal Teknik ITS*, Volume 1.

Ummah, I. & Abdillah, D., 2016. Perancangan Simulasi Jaringan Virtual Berbasis Software-Define Networking. *Ind. Journal on Computing*, 1(1), pp. 95-106.

Wawge, P. & Tijare, P., 2014. Implementing Parameterized Dynamic Load Balancing Algorithm Using CPU and Memory. *International Journal of Innovative Research in Science, Engineering and Technology*, 3(5).

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