Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer e-ISSN: 2548-964X
Vol. 6, No. 7, Juli 2022, hlm. 3560-3569 http://j-ptiik.ub.ac.id

Implementasi *Load Balancing* menggunakan Metode Regresi Linier pada *Software Defined Network*

Ahmad Ali Hamdan[1], Primantara Hari Trisnawan[2], Fariz Andri Bakhtiar[3]

Program Studi Teknik Informatika, Fakultas Ilmu Komputer, Universitas Brawijaya
Email: [1]hamdan@student.ub.ac.id, [2]prima@ub.ac.id, [3]fariz@ub.ac.id

Abstrak

Penggunaan internet yang semakin meluas akan menyebabkan tingginya trafik jaringan. Beberapa server perlu ditambahkan untuk menghindari *overload*. *Load balancing* juga diperlukan untuk mendistribusikan beban secara merata. *Load balancing* banyak diimplementasikan pada *Software Defined Network* (SDN) karena bersifat *programmable* dan fleksibel untuk jaringan yang rumit. Oleh sebab itu, penelitian ini mengusulkan pengimplementasian *load balancing* pada SDN menggunakan regresi linier. Isi *dataset* yang digunakan untuk momodelkan fungsi regresi linier sebanyak tiga puluh sampel. *Dataset* tersebut berisi informasi *central processing unit* (CPU), *random access memory* (RAM) dan jumlah *request*. Perhitungan regresi linier mengacu pada CPU dan RAM server. Hasil perhitungan tersebut merupakan nilai prediksi jumlah *request* setiap server. Server dengan jumlah *request* terkecil dipilih oleh *controller* untuk melayani *request*. Penelitian ini membandingkan antara regresi linier dan *round robin.* Pada pengujian distribusi trafik menggunakan regresi linier, server dengan spesifikasi tinggi mendapatkan distribusi lebih banyak dibandingkan server dengan spesifikasi rendah. Pada pengujian dengan *rate* 300 *request* per detik, regresi linier menghasilkan *response time* sebesar 169,3 hingga 353,4 milidetik, sedangkan *round robin* sebesar 339,2 hingga 1232,6 milidetik. Regresi linier menghasilkan *throughput* sebesar 241,5 hingga 310,7 KB/s, sedangkan *round robin* sebesar 129,4 hingga 179,8 KB/s. Pada pengujian CPU *utilization*, regresi linier menghasilkan penggunaan CPU tertinggi 75 persen, sedangkan *round robin* tertinggi 98 persen.

Kata kunci: *Load Balancing, Software Defined Network,* Regresi Linier, Mininet, POX *Controller*

Abstract

A growing number of internet users leads to a high amount of network traffic. Servers need to be added to avoid overload. Load balancing is also needed to distribute the load equally. Load balancing is widely implemented in SDN because it's programmable and flexible in a complex network. Therefore, this research proposes the implementation of load balancing in SDN using linear regression. The dataset for modeling linear regression function contains thirty samples. It contains the information of central processing unit (CPU), random access memory (RAM) and number of requests. The linear regression calculation is according to the server's CPU and RAM. The result is a prediction of the number of requests from each server. A server with the lowest request is chosen by the controller to handle the request. This research compares between linear regression and round-robin. In traffic distribution testing using linear regression, server with higher specifications gets more distribution than a server with lower specifications. In the testing of 300 requests per second, linear regression's response time is between 169.3 to 353.4 milliseconds, and round-robin is between 339.2 to 1232.6 milliseconds. Linear regression's throughput is between 241.5 to 310.7 KB/s, and round-robin is between 129.4 to 179.8 KB/s. In CPU utilization testing, the highest CPU usage in linear regression is 75 percent, and round-robin is 98 percent.

Keywords: *Load Balancing, Software Defined Network, Linear Regression, Mininet, POX Controller*

1. PENDAHULUAN

Dalam dekade terakhir ini, kebutuhan jaringan telah berubah dengan cepat seiring dengan meluasnya penggunaan internet. Pada bulan Januari tahun 2022, terdapat 204,7 juta

pengguna internet di Indonesia (Kemp, 2022). Penggunaan internet yang semakin meluas akan menyebabkan tingginya trafik jaringan. Jika sumber daya server tidak sepadan dengan banyaknya permintaan yang diminta oleh *client*, maka akan muncul sebuah permasalahan. Permasalahan yang muncul akibat server kehabisan *resource* akibat tingginya trafik jaringan adalah *overload*. *Overload* merupakan suatu kondisi di mana server gagal melayani permintaan *client* disebabkan server kehabisan sumber daya (Schroeder & Harchol-Balter, 2006). Solusi untuk mengatasi *overload* pada suatu server adalah dengan menambah beberapa server. Selain itu juga diperlukan sebuah metode untuk membagi beban pada server-server yang tersedia.

Load balancing adalah teknik pembagian beban untuk memastikan beban didistribusikan secara merata ke semua server (Ibrahim, dkk. 2021). *Load balancing* berfungsi mendistribusikan beban kerja ke beberapa komputer, kluster-kluster komputer, *central processing units*, *drive disk*, atau sumber daya lainnya untuk mencapai pemanfaatan sumber daya yang optimal, memaksimalkan *throughput*, meminimalisasi *response time* dan menghindari *overload* atau kelebihan beban (Sidhu & Kinger, 2013). Dengan begini, *load balancing* dapat digunakan untuk mengatasi permasalahan pembagian beban pada beberapa server secara merata.

Load balancing banyak diimplementasikan pada arsitektur *Software Defined Network* (SDN). Hal ini dikarenakan SDN bersifat *programmable* dan juga fleksibel untuk jaringan yang rumit (Hamdan, dkk, 2021). Secara garis besar, SDN memisahkan *control plane* dari *data forwarding plane* (McKeown, dkk. 2008). Sebagai contohnya *data forwarding* menggunakan switch OpenFlow yang dapat diprogram melalui OpenFlow *controller*. Switch-switch ini menggunakan sebuah protokol untuk berkomunikasi dengan *controller*. *Controller* SDN bertanggung jawab atas perangkat *data plane* yaitu *router* dan *switch* (Feamster, dkk. 2013). *Load balancing* yang diimplementasikan pada SDN dapat memberikan beberapa keuntungan. Salah satu keuntungan penggunaan *load balancing* pada SDN adalah tidak diperlukannya *dedicated hardware* yang otomatis dapat menekan biaya. Dengan kelebihan yang dimiliki oleh arsitektur SDN, *load balancing* cocok diterapkan pada SDN yang memiliki kompleksitas tinggi dan konfigurasi yang rumit.

Heterogenitas server dapat menjadi permasalahan pada sebuah layanan web server berskala besar. Suatu server dapat ditambahkan atau dihapus sewaktu-waktu pada layanan web server. Server-server yang ada juga mungkin saja memiliki spesifikasi yang berbeda. Hal ini menjadi tantangan untuk menentukan metode *load balancing* yang dapat menyesuaikan kondisi dari semua server yang tersedia. Penggunaan metode *load balancing* yang bersifat dinamis tentunya diperlukan agar konfigurasi tidak dilakukan berulang setiap terjadi perubahan. Metode *load balancing* yang diperlukan adalah metode yang dapat beradaptasi sesuai kondisi dari server dan dapat mendistribusikan beban secara merata sesuai ketersediaan *resource* dari server yang ada.

Berdasarkan permasalahan yang telah dijelaskan, penelitian ini mengusulkan implementasi *load balancing* pada *Software Defined Network* dengan menggunakan perhitungan regresi linier. Regresi linier memiliki kegunaan untuk menunjukkan hubungan antara variabel respon dan variabel prediktor (Montgomery, dkk. 2021). Dengan begitu regresi linier dapat memprediksikan sesuatu berdasarkan variabel yang disebut prediktor. Prediksi yang dilakukan juga berpacu pada *dataset*. *Dataset* merupakan sekumpulan informasi di masa lalu, dalam kasus penelitian ini adalah informasi server di masa lalu. Dengan menggunakan regresi linier, sistem *load balancing* diharapkan mampu membagi beban berdasarkan ketersediaan *resource* dari server meskipun semua servernya memiliki spesifikasi yang berbeda-beda. Dalam penelitian ini, CPU dan RAM merupakan variabel prediktor. Sedangkan jumlah *request* akan menjadi variabel respon. VanVoorhis & Morgan (2020) mengatakan bahwa penggunaan sepuluh sampel per jumlah prediktor sudah dapat dikatakan baik. Berdasarkan penelitian tersebut, akan dibuat *dataset* berjumlah tiga puluh buah yang berfungsi untuk memodelkan fungsi regresi liniernya. *Dataset* tersebut berisi variabel respon dan prediktor yang merupakan informasi dari setiap server. Informasi dari server akan dikirimkan terus menerus ke *controller*. Lima data terakhir pada *dataset* akan diperbarui setiap lima detik. Informasi CPU dan RAM dari server akan dijadikan masukan sebagai perhitungan regresi linier. Hasil perhitungan tersebut

merupakan nilai prediksi jumlah *request* pada setiap server. Server dengan jumlah *request* terkecil akan dipilih oleh *controller* untuk melayani *request*.

2. KAJIAN PUSTAKA

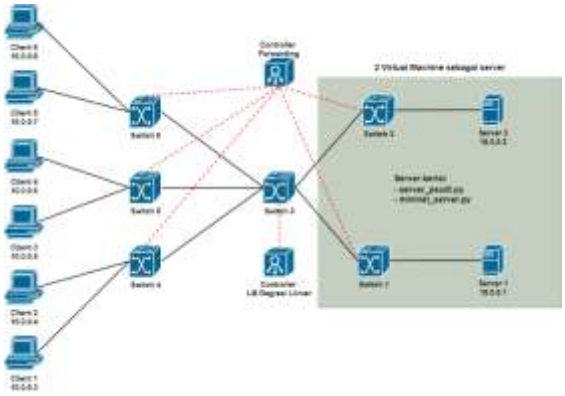
Utomo, dkk (2020) melakukan penelitian dengan mengimplementasikan *load balancing* menggunakan metode klasifikasi *Naive Bayes*. Penelitian ini juga membandingkan antara algoritme *Naive Bayes* dan algoritme *Fuzzy*. Implementasi sistem dilakukan pada emulator mininet dengan satu buah *controller* dan *switch*, serta terdapat tiga server dan satu *client*. Penelitian ini melakukan dua skenario pengujian yakni pengujian teks *high* dan teks *low*. Parameter pengujian meliputi *throughput*, *response time*, *memory usage* dan CPU *usage*. Hasil yang didapat adalah algoritme *Naive Bayes* memiliki kinerja lebih baik dibandingkan dengan algoritme *Fuzzy*.

Aza & Urrea (2019) melakukan penelitian dengan mengimplementasikan *load balancing* menggunakan algoritme *round robin* pada jaringan *wireless* kampus menggunakan SDN ketika melakukan video *streaming*. Penelitian ini didasari oleh sulitnya pengelolaan jaringan yang sulit dan distribusi yang tidak merata antar *Acess Point* (AP). *Controller* yang digunakan pada penelitian ini adalah *POX Controller*. Penelitian ini melakukan dua skenario percobaan antara *load balancer* dan jaringan tradisional. Hasilnya, *throughput* yang dihasilkan skenario jaringan yang menggunakan *load balancer* lebih besar dibandingkan dengan skenario jaringan tradisional.

Deepa & Cheelu (2017) melakukan perbandingan algoritme-algoritme *load balancing* di *cloud computing*. Diantaranya adalah *round robin*, *min-min*, *min-max*, *ant colony optimization* dan *honey bee foraging*. Tentunya setiap algoritme-algoritme tersebut memiliki keunggulan dan kekurangan masing-masing. *Round robin* memiliki keunggulan kinerja yang baik pada siklus eksekusi CPU yang singkat, tetapi membutuhkan waktu yang lama ketika menangani jumlah *task* yang banyak. *Min-min* memberikan hasil yang baik untuk *task* sedikit, tetapi rawan akan *starvation* atau kondisi dimana tidak pernah mendapat *resource* setelah terjadi *deadlock*. *Ant colony optimization* memiliki keunggulan dalam segi waktu komputasi, tetapi membutuhkan waktu lama dalam pencarian dan terlalu kompleks. *Honey bee foraging* memiliki keunggulan dapat mengurangi *response time* dan meningkatkan *throughput*, tetapi memakan waktu lama ketika memuat *task* dengan prioritas rendah.

3. PERANCANGAN SISTEM

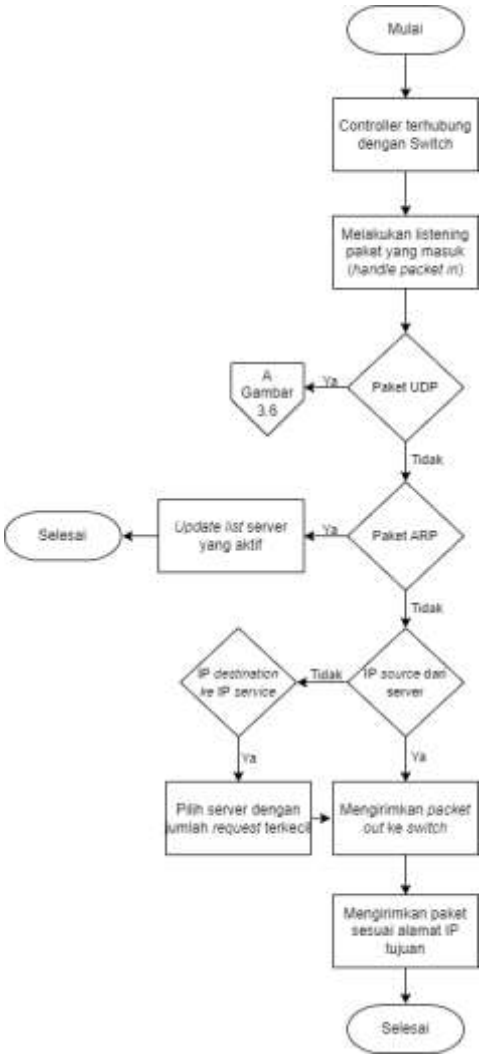
3.1. Perancangan Topologi SDN



Gambar 1. Topologi SDN

Gambar 1 merupakan topologi untuk arsitektur SDN yang akan diimplementasikan pada simulator jaringan Mininet. Topologi ini terdiri dari satu buah *controller* yang berfungsi sebagai *load balancer* dengan menggunakan perhitungan regresi linier, satu buah *controller* sebagai *forwarding*, enam buah *switch*, enam *client* dan dua server. Untuk keperluan pengujian, HTTPerf dapat dijalankan pada salah satu *client*. Untuk menunjukkan bahwa servernya beragam, kedua server dibuat dengan spesifikasi yang berbeda. Selain itu, terdapat dua *controller* yaitu *controller forwarding* dan *load balancing*. *Controller forwarding* terhubung dengan *switch* 1, 2, 4, 5 dan 6 sedangkan *controller load balancing* terhubung dengan *switch* 3. Tiap-tiap server berisi mininet_server.py yang merupakan kode program untuk membangun jaringan mininet, serta server_psutil.py yang merupakan *library* psutil untuk mendapatkan informasi penggunaan CPU dan RAM. Kemudian informasi tersebut akan dikirimkan ke *controller* untuk dihitung menggunakan regresi linier. Server dengan prediksi jumlah *request* terkecil akan dipilih untuk menangani *request* dari *client* berdasarkan hasil perhitungan regresi linier.

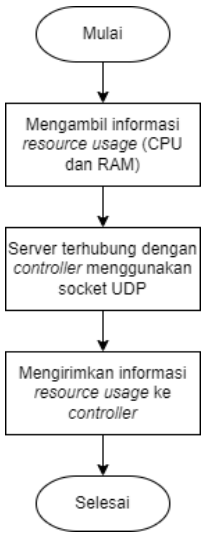
3.2. Perancangan *Load Balancing*



Gambar 2. *Flowchart Load Balancing*

Gambar 2 menjelaskan alur proses *load balancing* bekerja. Proses *load balancing* diawali dengan *controller* yang terhubung dengan *switch*. *Controller* akan melakukan *listening* untuk menunggu paket yang masuk. Ketika ada paket masuk, *controller* akan memeriksa apakah paket tersebut merupakan paket UDP. Jika ya, maka dapat dipastikan bahwa paket tersebut merupakan informasi *resource* yang dikirimkan ke *controller* menggunakan protokol UDP. Proses yang dilakukan ketika paket UDP masuk akan dijelaskan pada sub bab perancangan *dataset*. Jika bukan UDP, *controller* akan memeriksa apakah paket tersebut adalah paket ARP. Jika paket tersebut merupakan paket ARP, *controller* akan melakukan *update* daftar server yang aktif. Jika alamat IP bersumber dari server, dapat dipastikan paket tersebut merupakan paket dari server untuk dikirimkan ke *client*. *Controller* akan mengirimkan paket ke switch sesuai alamat IP dari *client*. Jika alamat IP tujuan merupakan alamat IP *service*, dapat dipastikan paket tersebut berasal dari *client*, yang kemudian akan dipilih server dengan jumlah *request* terkecil. *Controller* akan mengirimkan paket ke switch dengan tujuan alamat IP server dengan jumlah *request* terkecil.

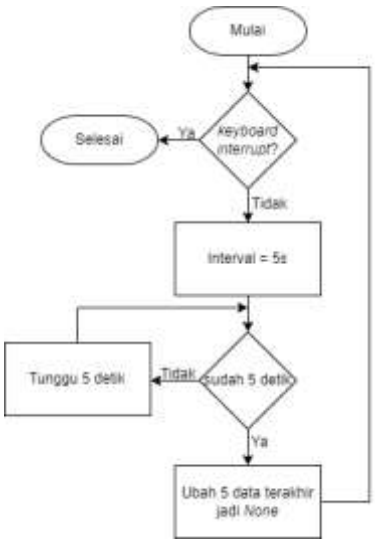
3.3. Perancangan Psutil



Gambar 3. *Flowchart* Psutil

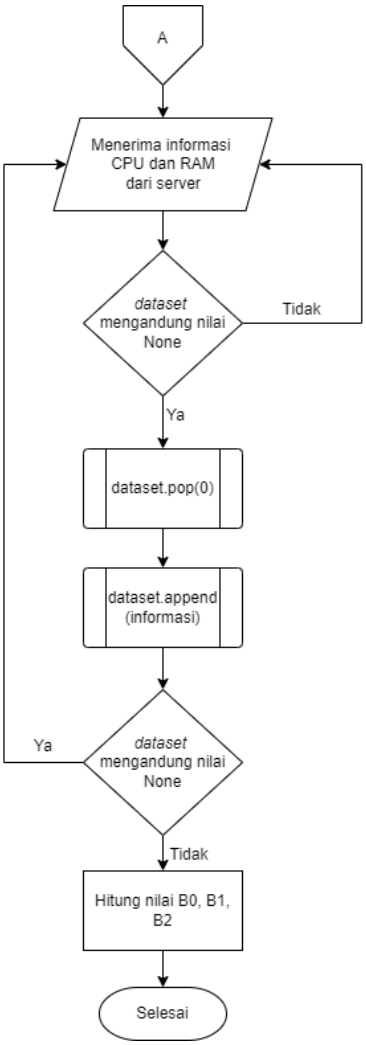
Gambar 3 menjelaskan alur kerja Psutil. Proses diawali dengan server membaca informasi *resource* CPU dan RAM. Setelah informasi CPU dan RAM didapatkan, Informasi tersebut akan dikirimkan ke *controller* menggunakan *socket* UDP.

3.4. Perancangan *Dataset*



Gambar 4. *Flowchart* Hapus *Dataset* Setiap 5 Detik

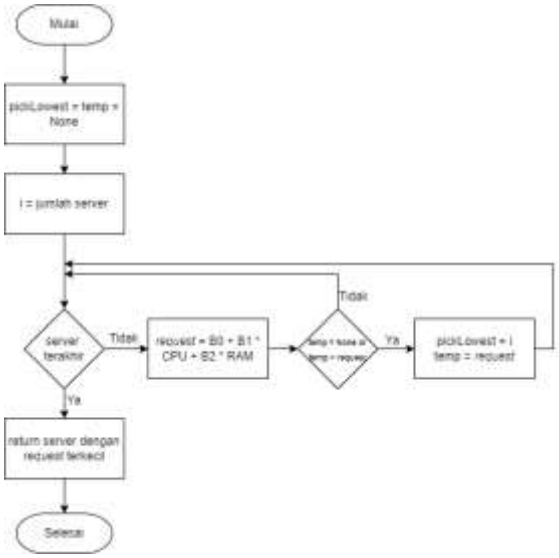
Gambar 4 merupakan alur proses penghapusan lima *dataset* terakhir setiap lima detik. Proses ini akan jalan terus menerus. Proses ini juga akan memicu proses pembaruan *dataset* pada pada Gambar 5.



Gambar 5. *Flowchart* Psutil

Gambar 5 merupakan proses lanjutan dari perancangan *load balancing* ketika paket ini merupakan UDP. Alur pengisian *dataset* diawali dengan *controller* menerima informasi *resource* dari server. Setelah itu akan dicek apakah *dataset* memiliki nilai *None*. Apabila terdapat nilai *None*, program akan menghapus nilai *None* tersebut dan kemudian akan memasukkan informasi baru. *Controller* akan mengeluarkan nilai *None* di indeks pertama dan memasukkan informasi *resource* server dari indeks terakhir. Kemudian akan dicek kembali apakah *dataset* masih memiliki nilai *None* atau tidak. Proses ini akan dilakukan berulang hingga *dataset* terisi penuh atau tidak memiliki nilai *None*. Perhitungan nilai B0, B1, B2 dapat dilakukan ketika *dataset* sudah terisi penuh. Proses ini akan membuat *dataset* bersifat dinamis dan isinya selalu berubah-ubah mengikuti kondisi server.

3.5. Perancangan Metode Regresi Linier



Gambar 6. *Flowchart* Pemilihan Server

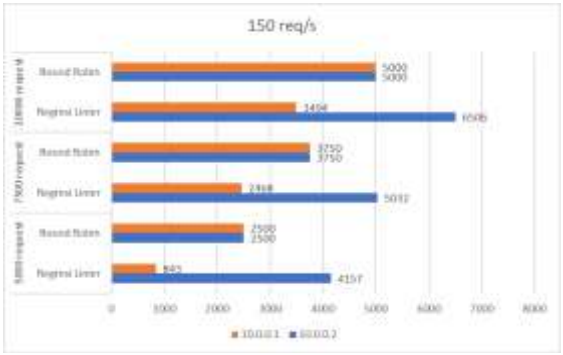
Gambar 6 merupakan alur pemilihan server dengan *request* terkecil menggunakan perhitungan regresi linier. Proses diawali dengan mendeklarasikan variabel *pickLowest* dan *temp* yang nantinya berguna sebagai perbandingan. Kemudian terdapat proses perulangan sebanyak server yang aktif. Setelah itu, akan dilakukan proses perhitungan menggunakan fungsi regresi linier. B0, B1 dan B2 merupakan nilai yang didapat dari perhitungan *dataset* sebelumnya. Sedangkan CPU dan RAM merupakan informasi *resource* masing-masing server. Ketika proses perulangan baru dilakukan pertama kali, maka server tersebut akan dianggap server terkecil dengan memasukkannya ke variabel *pickLowest*. Proses perulangan ke dua dan seterusnya, akan dilakukan perbandingan dengan server sebelumnya. Ketika semua server sudah dihitung, *controller* akan mendapatkan server dengan jumlah *request* terkecil.

4. PENGUJIAN

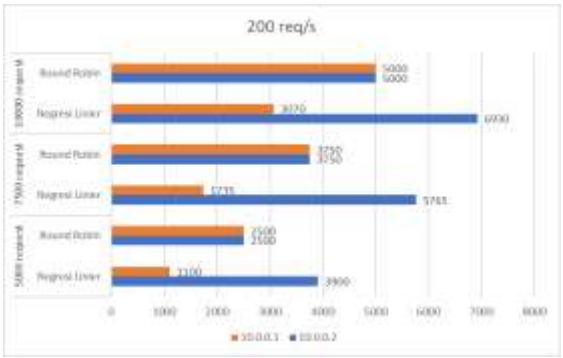
Pengujian dilakukan dengan skenario *rate* 100, 150, 200, 250 dan 300 *request* per detik. Masing-masing skenario dilakukan tiga kali percobaan sejumlah 5000, 7500 dan 10000 *request.*

4.1. Pengujian Distribusi Trafik

Gambar 7. Distribusi Trafik Dengan *Rate* 100

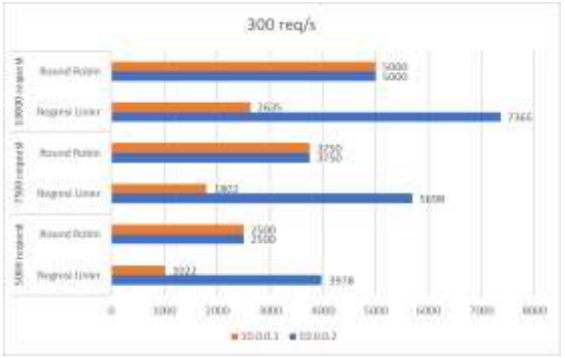


Gambar 8. Distribusi Trafik Dengan *Rate* 150



Gambar 9. Distribusi Trafik Dengan *Rate* 200

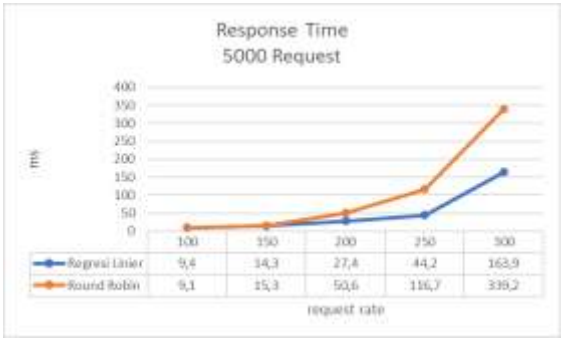
Gambar 10. Distribusi Trafik Dengan *Rate* 250



Gambar 11. Distribusi Trafik Dengan *Rate* 300

Gambar 7, 8, 9, 10, 11 merupakan hasil pengujian distribusi trafik. Secara jumlah distribusi, dapat dilihat bahwa regresi linier tidak mendistribusikan trafik secara merata dikarenakan regresi linier mempertimbangkan *resource* CPU dan RAM dari setiap server. Karena server 1 memiliki *resource* lebih kecil dibandingkan dengan server 2, maka pendistribusiannya pun akan condong ke server 2 yang memiliki *resource* lebih besar. Sedangkan pada algoritme *round robin* distribusinya merata ke semua server tanpa mempertimbangkan *resource* server.

4.2. Pengujian *Response Time*

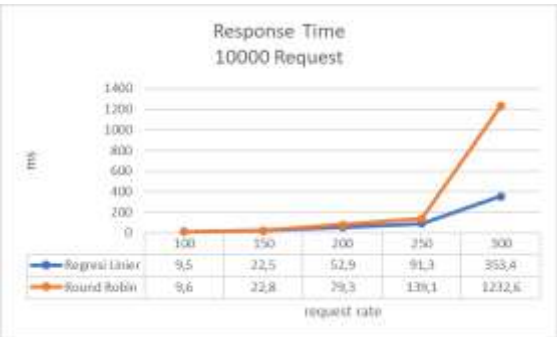


Gambar 12. *Response Time* Dengan 5000 *Request*

Gambar 12 merupakan grafik pengelompokkan percobaan 5000 *request* pada semua skenario. Berdasarkan grafik, dapat dilihat *response time* pada regresi linier dan *round robin* semakin meningkat seiring bertambahnya *rate* yang digunakan. Mulai terjadi kenaikan yang signifikan pada *rate* 200 hingga 300. Hal tersebut dikarenakan server mulai terbebani dengan penggunaan *rate* yang tinggi sehingga mempengaruhi kinerja server.

Gambar 13. *Response Time* Dengan 7500 *Request*

Gambar 13 merupakan grafik pengelompokkan percobaan 7500 *request* pada semua skenario. Pada percobaan sebanyak 7500 *request*, hasilnya pun menunjukkan bahwa semakin besar *rate* yang digunakan, maka semakin besar pula *response time*-nya. Hal ini dikarenakan beban yang semakin meningkat seiring penambahan *rate* yang digunakan.



Gambar 14. *Response Time* Dengan 10000 *Request*

Gambar 14 merupakan grafik pengelompokkan percobaan 10000 *request* pada semua skenario. Pola pada percobaan ini pun sama seperti sebelumnya, yaitu peningkatan *response time* seiring meningkatnya *rate* yang digunakan. Namun dapat dilihat pada *rate* 300, *response time* regresi linier dan *round robin* meningkat drastis. Hal ini dikarenakan percobaan dengan *rate* 300 sudah mencapai batas kemampuan server. Namun dapat dilihat bahwa regresi linier masih lebih baik dibandingkan dengan *round robin*. Hal tersebut dikarenakan regresi linier mampu membagi beban berdasarkan ketersediaan *resource* dari server. Sedangkan *round robin* hanya membagi trafik sama rata sehingga server yang memiliki spesifikasi rendah sangat terbebani.

4.3. Pengujian *Throughput*

Gambar 15. *Throughput* Dengan 5000 *Request*

Gambar 15 merupakan grafik pengelompokkan *throughput* dengan percobaan 5000 *request* pada semua skenario. Regresi linier dan *round robin* memiliki *throughput* yang relatif sama pada *rate 100* dan 150. Kemudian terdapat sedikit perbedaan pada *rate* 200, dan terlihat perbedaan yang signifikan pada *rate* 250 dan 300. *Throughput* regresi linier terus meningkat pada semua *rate*. Sedangkan *throughput round robin* menurun ketika *rate* 300. Hal ini dikarenakan regresi linier belum mengalami penumpukan *request* pada salah satu server sehingga *throughput*-nya lebih baik. Sedangkan *round robin* mengalami penuruan *throughput* disebabkan beban yang tinggi pada server dengan spesifikasi rendah sehingga *throughput*-nya menurun.

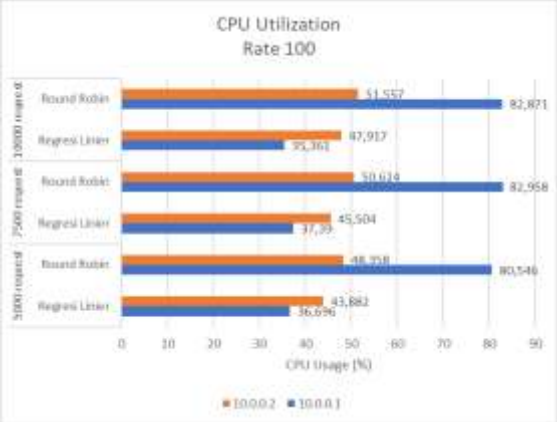
Gambar 16. *Throughput* Dengan 7500 *Request*

Gambar 16 merupakan grafik pengelompokkan *throughput* dengan percobaan 7500 *request* pada semua skenario. Pada percobaan ini, regresi linier dan *round robin* sama-sama mengalami penurunan *throughput* pada *rate* 300. Banyaknya *request* dan tingginya *rate* menjadi penyebab penurunan *throughput*. Namun kinerja regresi linier masih lebih baik dibandingkan dengan *round robin*.

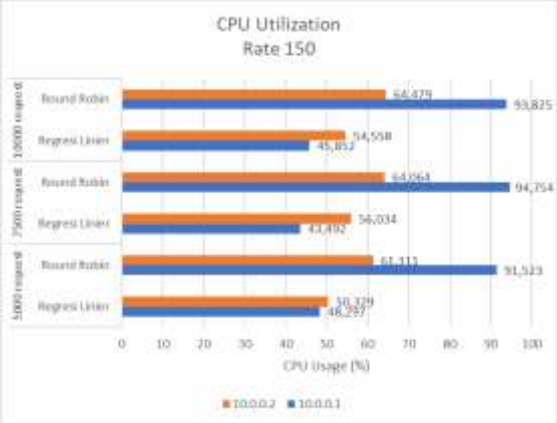
Gambar 17. *Throughput* Dengan 10000 *Request*

Gambar 17 merupakan grafik pengelompokkan *throughput* dengan percobaan 10000 *request* pada semua skenario. Sama seperti sebelumnya, percobaan dengan 10000 *request* mengalami penurunan *throughput* di kedua metode. Regresi linier masih memiliki kinerja *throughput* yang lebih baik dibandingkan dengan *round robin*. Hal ini dikarenakan regresi linier membagi beban berdasarkan ketersediaan *resource* sehingga kinerja yang dihasilkan lebih baik.

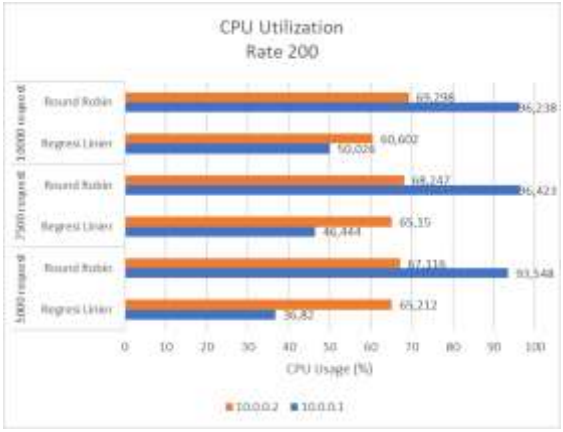
4.4. Pengujian CPU *Utilization*



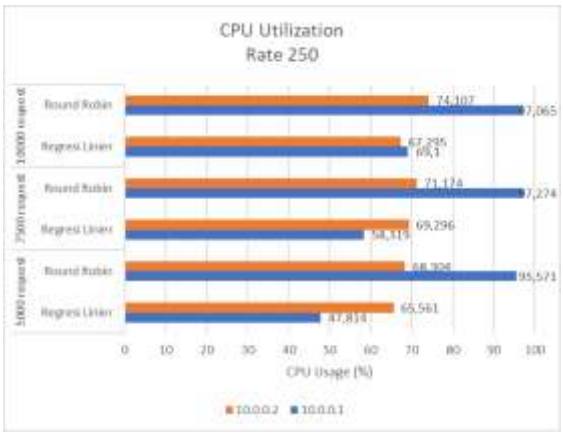
Gambar 18. CPU *Utilization* Dengan *Rate* 100



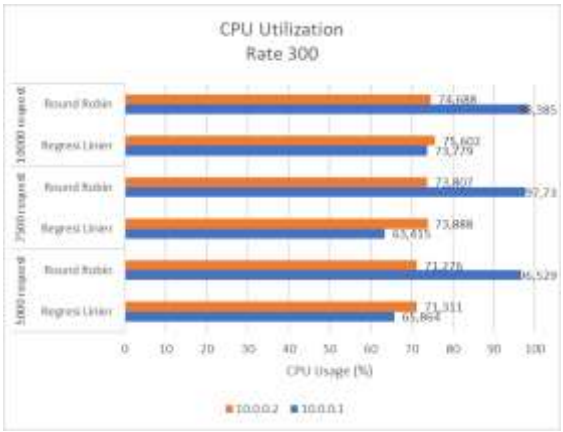
Gambar 19. CPU *Utilization* Dengan *Rate* 150



Gambar 20. CPU *Utilization* Dengan *Rate* 200



Gambar 21. CPU *Utilization* Dengan *Rate* 250



Gambar 22. CPU *Utilization* Dengan *Rate* 300

Gambar 18, 19, 20, 21, 22 merupakan hasil pengujian CPU *utilization*. Secara keseluruhan, penggunaan CPU regresi linier lebih kecil dibandingkan dengan *round robin*. Penggunaan CPU di server 2 pada *round robin* sangat tinggi dikarenakan *round robin* membagi distribusi sama rata tanpa memperhatikan *resource* server. Sedangkan regresi linier dapat membagi beban sesuai *resource* server agar server dengan *resource* kecil tidak terbebani.

5. KESIMPULAN DAN SARAN

5.1. Kesimpulan

Berdasarkan perancangan, implementasi, pengujian dan pembahasan pada penelitian *load balancing* server menggunakan regresi linier, dapat disimpulkan beberapa poin, yaitu:

1. Implementasi *load balancing* pada SDN dapat dilakukan menggunakan POX *controller*. POX *controller* berperan sebagai pengatur terkait bagaimana beban didistribusikan. Metode pembagian beban dilakukan dengan menggunakan perhitungan regresi linier. Penelitian ini menggunakan informasi *resource* yang terdiri dari CPU dan RAM untuk melakukan *load balancing*. Untuk membaca informasi *resource* dari server, penelitian ini menggunakan Psutil. Informasi *resource* yang didapatkan menggunakan Psutil akan disimpan ke dalam *dataset. Dataset* berfungsi sebagai perhitungan fungsi model regresi linier. Selain itu, informasi *resource* juga digunakan untuk menentukan server mana yang memiliki *request* paling sedikit. Penentuan server dengan *request* paling sedikit dilakukan dengan perhitungan regresi linier. Dengan begitu, implementasi *load balancing* pada SDN menggunakan regresi linier berhasil dilakukan.
2. Hasil distribusi trafik *load balancing* menggunakan regresi linier condong ke server yang memiliki spesifikasi lebih besar. Regresi linier mampu menyeimbangkan beban berdasarkan ketersediaan *resource* dari masing-masing server. Meskipun server memiliki spesifikasi yang berbeda, beban trafik akan dibagi sesuai ketersediaan *resource* server untuk menghindari *overload* pada server dengan spesifikasi kecil.
3. Pada pengujian *response time* dengan *rate* rendah, regresi linier dan *round robin* memiliki kinerja yang relatif sama. Semakin tinggi *rate* yang digunakan, regresi linier mengungguli *round robin* pada kinerja *response time*. Pada pengujian *throughput*, regresi linier dan *round robin* memiliki pola yang sama seperti pengujian *response time*. Ketika *rate* yang digunakan kecil, regresi linier dan *round robin* memiliki kinerja yang relatif sama. Kinerja *throughput* pada regresi linier mengungguli *round robin* ketika *rate* yang digunakan semakin tinggi. Pada pengujian terakhir atau CPU *utilization*, penggunaan CPU pada regresi linier lebih baik dibandingkan *round robin*. Penggunaan CPU tertinggi pada regresi linier mencapai 75 persen. Sedangkan *round robin* mencapai 98 persen.

5.1. Saran

Saran-saran yang mungkin dapat berguna bagi pembaca dan penelitian selanjutnya yaitu:

1. Implementasi menggunakan server pada perangkat *real* agar tidak terjadi saling intervensi *resource* jika dijalankan pada perangkat yang sama atau virtual.
2. Menambahkan parameter informasi spesifikasi tiap server ke *dataset*, yang kemudian dilakukan normalisasi.
3. Meningkatkan akurasi perhitungan regresi linier dengan cara menambahkan banyaknya sampel pada *dataset*.
4. Melakukan optimasi program terkait kapan perhitungan regresi linier dilakukan. Sebagai contoh, perhitungan dilakukan setiap ada *request* dari *client* dan hanya dihitung jika terjadi perubahan informasi dari server. Optimasi program berguna untuk memangkas waktu yang dibutuhkan untuk menjalankan program dan juga mengurangi beban pada *controller*.
5. Skenario penambahan dan penghapusan server untuk menunjukkan perhitungan regresi linier tetap dinamis meskipun servernya berubah-ubah.

DAFTAR PUSTAKA

Aza, E. F. & Urrea, J. P., 2019. *Implementation of Round-Robin load balancing scheme in a wireless software defined network*. s.l., IEEE.

Deepa, T. & Cheelu, D., 2017. *A Comparative Study of Static and Dynamic Load Balancing Algorithms in Cloud Computing*. s.l., IEEE.

Feamster, N., Rexford, J. & Zegura, E., 2013. The Road to SDN: An intellectual history of programmable networks. *Queue*, 11(12), pp. 20-40.

Ibrahim, I. M. et al., 2021. Web Server

Performance Improvement Using Dynamic Load Balancing Techniques: A Review. *Asian Journal of Research in Computer Science*, 10(1), pp. 47-62.

Kemp, S., 2022. *Data Reportal.* [Online] Tersedia di: https://datareportal.com/reports/digital-2022-indonesia [Diakses 3 Maret 2022].

McKeown, N. et al., 2008. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Review*, 38(2), pp. 69-74.

Montgomery, D. C., Peck, E. A. & Vining, G. G., 2021. *Introduction to linear regression analysis*. s.l.:John Wiley & Sons.

Schroeder, B. & Harchol-Balter, M., 2006. Web Servers Under Overload: How Scheduling Can Help. *ACM Transactions on Internet Technology*, 6(1), pp. 20-52.

Sidhu, A. K. & Kinger, S., 2013. Analysis of Load Balancing Techniques in Cloud Computing. *International Journal of computers & technology*, 4(2), pp. 737-741.

Utomo, A. A., Trisnawan, P. H. & Primananda, R., 2020. Mekanisme Load Balancing Server Menggunakan Metode Naive Bayes. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*, Volume 4, pp. 1048-1055.