Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK) DOI: 10.25126/jtiik.202184294
Vol. 8, No. 3, Juni 2021, hlm. 525-532 p-ISSN: 2355-7699
Akreditasi KEMENRISTEKDIKTI, No. 36/E/KPT/2019 e-ISSN: 2528-6579

PENINGKATAN UTILISASI JARINGAN *DISTRIBUTED STORAGE SYSTEM* MENGGUNAKAN KOMBINASI *SERVER* DAN *LINK LOAD BALANCING*

Hawwin Purnama Akbar[*1], Achmad Basuki[2], Eko Setiawan[3]

[1,2,3] Fakultas Ilmu Komputer, Universitas Brawijaya
Email: [1]haw2win@student.ub.ac.id, [2]abazh@ub.ac.id, [3]ekosetiawan@ub.ac.id
[*]Penulis Korespondensi

(Naskah masuk: 04 November 2021, diterima untuk diterbitkan: 09 Juni 2021)

Abstrak

Distributed Storage System (DSS) memiliki sejumlah perangkat server penyimpanan yang terhubung dengan banyak perangkat *switch* untuk meningkatkan utilisasi jaringan. DSS harus memperhatikan keseimbangan beban pada sisi server penyimpanan dan *traffic* data pada semua jalur yang terhubung. Jika beban pada sisi server penyimpanan dan *traffic* data tidak seimbang, maka dapat menyebabkan *bottleneck network* yang menurunkan utilisasi jaringan. Kombinasi *server* dan *link load balancing* adalah solusi yang tepat untuk menyeimbangkan beban pada sisi server penyimpanan dan *traffic* data. Penelitian ini mengusulkan metode kombinasi algoritme *least connection* sebagai metode *server-load balancing* dan algoritme *global first fit* sebagai metode *link load balancing*. Algoritme *global first fit* merupakan salah satu dari algoritme *load balancing hedera* yang bertujuan untuk menyeimbangkan *traffic* data berukuran besar (10% dari *bandwidth*), sehingga terhindar dari permasalahan *bottleneck network*. Algoritme *least connection* merupakan salah satu algoritme *server load balancing* yang menggunakan jumlah total koneksi dari server untuk menentukan prioritas server. Hasil evaluasi kombinasi metode tersebut didapatkan peningkatan pada rata-rata *throughput* sebesar 77,9% dibanding hasil metode *Equal Cost Multi Path* (ECMP) dan *Round robin* (RR). Peningkatan pada rata-rata penggunaan *bandwidth* sebesar 65,2% dibanding hasil metode ECMP dan RR. Hasil Penggunaan *CPU* dan *memory* pada *server* di metode kombinasi ini juga terjadi penurunan beban *CPU* sebesar 34,29% dan penurunan beban penggunaan *memory* sebesar 9,8% dibanding metode ECMP dan RR. Dari hasil evaluasi, penerapan metode kombinasi metode *server* dan *link load balancing* berhasil meningkatkan utilisasi jaringan dan juga mengurangi beban server.
Kata kunci: *DSS, Load balancing, bottleneck network*

IMPROVEMENT OF DISTRIBUTED STORAGE SYSTEM NETWORK UTILIZATION USING COMBINATION OF SERVER AND LINK LOAD BALANCING

Abstract

Distributed Storage System (DSS) has a number of storage server devices that are connected to multiple switch devices to increase network utilization. DSS must pay attention to the balance of the load on the storage server side and data traffic on all connected lines. If the load on the storage server side and data traffic is not balanced, it can cause a network bottleneck that reduces network utilization. The combination of server and link-load balancing is the right solution to balance the load on the server side of storage and data traffic. This study proposes a combination of the least connection algorithm as a server-load balancing method and the global first fit algorithm as a link-load balancing method. The global first fit algorithm is one of Hedera's load balancing algorithms which aims to balance large data traffic (10% of bandwidth), so as to avoid network bottleneck problems. Least connection algorithm is one of the server balancing algorithms that uses the total number of connections from the server to determine server priority. The results of the evaluation of the combination of these methods showed an increase in the average throughput of 77.9% compared to the results of the Equal Cost Multi Path (ECMP) and Round robin (RR) methods. The increase in the average bandwidth usage is 65.2% compared to the results of the ECMP and RR methods. The results of CPU and memory usage on the server in this combination method also decreased CPU load by 34.29% and a decrease in memory usage load by 9.8% compared to the ECMP and RR methods. From the evaluation results, the application of a combination of the server method and the link load balancing method has succeeded in increasing network utilization and also reducing server load.

Keywords: *DSS, Load balancing, bottleneck network*

1. PENDAHULUAN

Survey kondisi *cloud server* tahun 2020 dari flexera menyatakan 93% dari perusahaan *enterprise* besar dengan jumlah pegawai diatas 1000 sudah mengadopsi sistem *cloud computing* (Flexera, 2020). Jaringan *data center* dalam skala besar dibutuhkan untuk mendukung perkembangan layanan *cloud* tersebut. Arsitektur *data center* memiliki subsistem yaitu jaringan *Distributed Storage System* (DSS). DSS dapat menyediakan tempat penyimpanan secara terdistribusi menjadi beberapa tiruan atau *copy* dan ditempatkan pada node yang berbeda dalam jaringan (Liu dkk., 2015). Jaringan dengan skala besar dengan topologi berjenis *multi-layer* dibutuhkan untuk peningkatan utilisasi jaringan DSS dengan efektif. Pada jaringan *multi-layer* juga terdapat sebuah masalah untuk mengatur dan menjadwalkan lalu lintas data pada jalur yang terdapat dalam jaringan (Guillen dkk., 2018b). Salah satu solusi untuk untuk mengatur dan menjadwalkan lalu lintas data adalah dengan mendistribusikan lalu lintas data secara seimbang di semua jalur yang ada.

Agar lalu lintas data dapat didistribusikan secara seimbang, terdapat mekanisme *load balancing.* Mekanisme *load balancing* bertujuan untuk membagi secara rata *flow* dari berbagai jalur yang berbeda (Zhang dkk., 2018). Tujuan umum dari mekanisme *load balancing* yaitu, 1) meningkatkan kinerja, 2) *robustness,* 3) skalabilitas, dan 4) *energy-efficiency*. Peningkatan kinerja jaringan dapat dilakukan dengan mekanisme *load balancing* yang dapat dibagi menjadi dua berdasarkan targetnya yaitu *link load balancing* untuk mengatur lalu lintas data pada *traffic data* dan *server load balancing* untuk mengatur lalu lintas data pada server penyimpanan (Zhang dkk., 2018). Untuk mengaplikasikan mekanisme *load balancing* dibutuhkan pendekatan *Software Defined Network* (SDN) dengan *openflow switch* agar dapat memprogram lalu lintas data dalam jaringan. SDN adalah arsitektur jaringan yang memisahkan fungsi *control plane* dan *data plane* dari *switch*, sehingga dapat memprogram fungsi *control plane* dari semua perangkat dengan satu *controller* saja (Xia dkk., 2018) (Kreutz dkk., 2015).

Terdapat penelitian yang membahas tentang penggunaan *link load balancing* dengan menggunakan sebuah metode bernama Hedera (Al-Fares dkk., 2010). dalam metode hedera terdapat algoritme *Global First Fit* yaitu algoritme pasif dan terpusat yang bertujuan untuk mengatur dan menyeimbangkan *flow* yang berukuran besar (>10% dari ukuran *bandwidth*) sehingga dapat menghindari terjadinya *bottleneck network. Bottleneck network* yaitu suatu fenomena saat kinerja atau kapasitas sistem dibatasi oleh keterbatasan suatu komponen. Kinerja Algoritme *Global First Fit* telah dibandingkan dengan Algoritme *Equal Cost Multi Path* (ECMP) yang merupakan algoritme dasar dari *link load balancing*, hasil dari perbandingan menunjukkan algoritme *Global First Fit* dapat membagi rata beban ke semua jalur yang tersedia dan mengungguli kinerja dari algoritme ECMP, tetapi algoritme ini masih belum membagi rata beban pada sisi *server* atau *storage* jika digunakan pada jaringan DSS.

Penelitian terkait algoritme *server load balancing* adalah algoritme *least loaded server* (Chen dkk., 2014). Algoritme ini mengatur beban server dengan parameter beban yang dimiliki server atau dengan metode *least loaded*, semakin sedikit beban pada server, maka semakin besar prioritas server untuk dipilih. Untuk mengetahui beban yang dimiliki server, algoritme ini menggunakan informasi yang diperoleh dari protokol *Simple Network Management Protocol* (SNMP). Penelitian Chen hanya menggunakan satu jalur yang menghubungkan *host* dengan berbagai *server,* sehingga tidak memperhitungkan jaringan dengan jalur lebih dari satu seperti pada jaringan DSS. Terdapat algoritme *least connection* yang juga merupakan algoritme *server load balancing.* Algoritme *least connection* menggunakan parameter total koneksi yang ada pada setiap server untuk menentukan prioritas server. Terdapat penelitian yang telah membandingkan algoritme least connection dan least loaded oleh Mustafa (Mustafa, 2017) yang menyatakan algoritme *least connection* dapat mendistribusikan lalu lintas pada *server* dengan lebih baik dibanding algoritme *least loaded server* dan *round-robin*.

Penelitian terkait lainnya yaitu kombinasi mekanisme *link load balancing* dan *server load balancing* (Guillen dkk., 2018a). Penelitian tersebut mengimplementasikan kombinasi algoritme *Max Disjoint Path* untuk *link load balancing* dan *Least Loaded Server* untuk *server load balancing* pada arsitektur jaringan DSS. Guillen menggunakan jaringan *data center* dengan model *fat tree network* (FTN) sebagai studi kasus dari DSS. Penelitian yang dilakukan berhasil mengimplementasikan kombinasi mekanisme *load balancing* dengan hasil yang melebihi hingga 150% nilai rata-rata *throughput* dibanding mekanisme *load balancing* tunggal. Kekurangan dari penelitian ini yaitu eksperimen yang dilakukan hanya menggunakan *flow* dengan ukuran kecil dengan rata-rata penggunaan *bandwidth* sebesar 6.5% dari kapasitas *bandwidth* yang disediakan. Meskipun *flow* dikirim dengan waktu yang panjang, tidak akan memenuhi kapasitas jalur yang disediakan dan tidak memperhatikan terjadinya *collision* atau tabrakan data yang dapat menyebabkan *bottleneck network*.

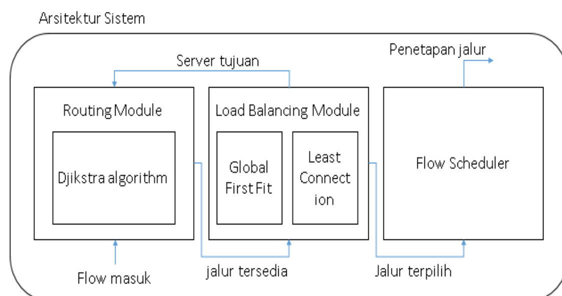
Berdasarkan penjelasan sebelumnya, maka ditemukan permasalahan baru yaitu dapat terjadinya *collision* yang dapat menyebabkan *bottleneck network* pada mekanisme kombinasi *load balancing*, sehingga dapat dikembangkan lagi untuk meningkatkan utilisasi jaringan DSS. Oleh karena itu penelitian ini mengusulkan kombinasi *server* dan

link load balancing berbasis SDN untuk meningkatkan utilisasi jaringan DSS secara optimal. *Link load balancing* menggunakan algoritme *Global first fit* yang dikombinasikan dengan *server load balancing* menggunakan algoritme *least connection*. Hasil penelitian dianalisis berdasarkan *throughput, bandwidth, CPU usage*, dan *memory usage*.

2. METODE PENELITIAN

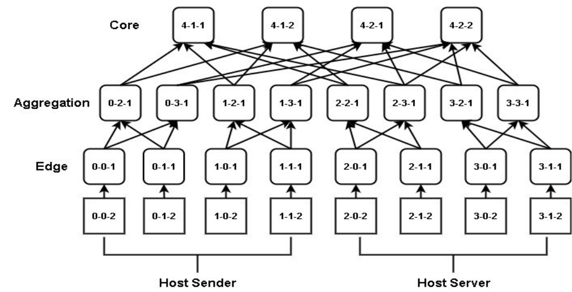
2.1. Arsitektur Sistem

Metode kombinasi *load balancing link* dan server diajukan untuk menangani permasalahan pada DSS. Pada arsitektur sistem terdapat tiga komponen utama yang mengendalikan perilaku *switch* yang terhubung dalam jaringan seperti pada Gambar 1. Komponen pertama yaitu *load balancing module* yang terdiri dari algoritme *global first fit* dan algoritme *least connection*, komponen kedua yaitu *routing module* yaitu modul untuk menentukan jalur terpendek antara *host* dan *server* yang dituju dari semua jalur yang tersedia. *Routing module* ini menggunakan algoritme *Dijkstra* untuk menemukan *path* terdekat dan juga berfungsi mengoleksi informasi dari jalur yang dibutuhkan oleh algoritme *load balancing*. Komponen ketiga yaitu *flow scheduler* yang berfungsi untuk menempatkan jalur yang telah ditentukan oleh algoritme *load balancing* ke jalur dalam jaringan agar bisa dilalui *flow* yang akan dikirim. Agar dapat menguji *flow* dengan ukuran besar, akan menggunakan kapasitas jalur pada topologi sebesar 1 GbE pada semua jalur yang menghubungkan *switch* dan *host*.



Gambar 1. Arsitektur Sistem

Untuk topologi jaringan, akan digunakan topologi dari DCN berjenis FTN dengan jumlah *host* = 8 seperti pada Gambar 2. terdapat 4 node yang berperan sebagai *host sender*, dan 4 node berperan sebagai *host server storage* dari jaringan DSS. Terdapat 20 buah *switch* yang terdiri dari 4 *switch* pada *core tier*, 8 buah *switch* pada *aggregation tier*, dan 8 buah *switch* pada *edge tier*. Semua *switch* akan terhubung kepada *controller* yang berfungsi sebagai *control plane* dari semua perangkat *switch* yang terhubung. Topologi akan diimplementasikan menggunakan *emulator* mininet pada sistem operasi LINUX 16.04 dengan ram sebesar 4Gb dan menggunakan *pox controller* sebagai SDN *controller*.



Gambar 2. Topologi Sistem

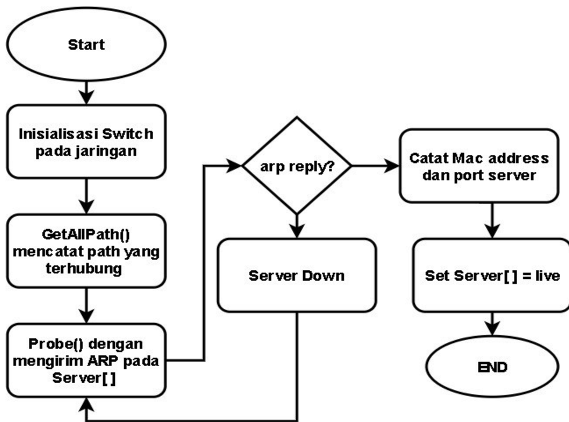
2.2. Mekanisme kombinasi server dan *link load balancing*

Untuk mekanisme kombinasi *server* dan *link load balancing*, akan digunakan fungsi dari *Least connection* dan fungsi dari *global first fit*. Diagram alir sistem akan dibagi menjadi dua, pertama adalah diagram yang menunjukkan mekanisme fungsi *probe()* seperti pada Gambar 3. Pertama, *controller* akan menginisialisasi *switch* saat terdapat koneksi dan mengubah status *switch* menjadi aktif atau *"up"*. Lalu setelah jumlah total *switch* sama dengan total *switch* yang berstatus aktif, akan dijalankan fungsi *GetAllPath()* yang berfungsi mencatat jalur yang terhubung pada setiap *switch* serta *cost* dari setiap jalur, informasi ini akan digunakan untuk menentukan jalur yang terpilih dari algoritme *routing*. Setelah itu terdapat fungsi *probe()* untuk mengecek kondisi *server* menggunakan *arp request* dan mencatat *server* yang aktif. Fungsi *probe()* ini dilakukan berulang kali sampai sistem dimatikan. Jika *server* membalas *arp request*, maka *server* tersebut akan dicatat sebagai *server* aktif dan diberi waktu *timeout* untuk selanjutnya akan diproses pada algoritme *least connection*.

Selanjutnya terdapat mekanisme fungsi utama dari *load balancing* yang dapat dicermati pada Gambar 4. Fungsi utama pada bagian ini adalah *handlepacketin()* yang akan berjalan saat terdapat paket yang masuk dari *host* menuju *switch*. Dalam sistem ini digunakan jenis *routing protocol reactive* yaitu proses pencarian jalur *routing* yang dilakukan jika ada permintaan terlebih dahulu (*on-demand*). Pada *OVS switch* paket yang diterima dapat diperoleh alamat sumber dan alamat tujuan.

Saat paket sampai pada *switch*, maka *controller* akan mencatat alamat *mac* dan *port* tujuan dari node pengirim paket tersebut. Jika alamat *mac* tujuan merupakan alamat *virtual IP* yang telah disiapkan, maka akan diproses oleh algoritme *least connection* untuk dipilihkan satu *server* dengan koneksi terkecil dari kumpulan *server* yang aktif. Alamat dari *server* yang telah terpilih akan diproses oleh algoritme *global first fit* untuk menentukan jalur yang akan dituju. Jika alamat sumber dari paket yang diterima

merupakan alamat dari *server* yang aktif maka paket akan langsung diproses oleh algoritme *global first fit* untuk ditentukan jalur menuju tujuan. Dari kedua percabangan tersebut akan diproses oleh fungsi *installFlow()* dan *SendFlow()* yang berfungsi memasang *flow* yang ada pada *flow table* dan mengirim data pada *flow* yang sudah terpasang. Jika alamat tujuan atau sumber dari paket yang masuk tidak termasuk pada *server* yang ada, maka akan diteruskan ke semua node untuk mencari node yang mengetahui alamat paket tersebut pada fungsi *flood()*.



Gambar 3. Diagram alir inisialisasi *switch*

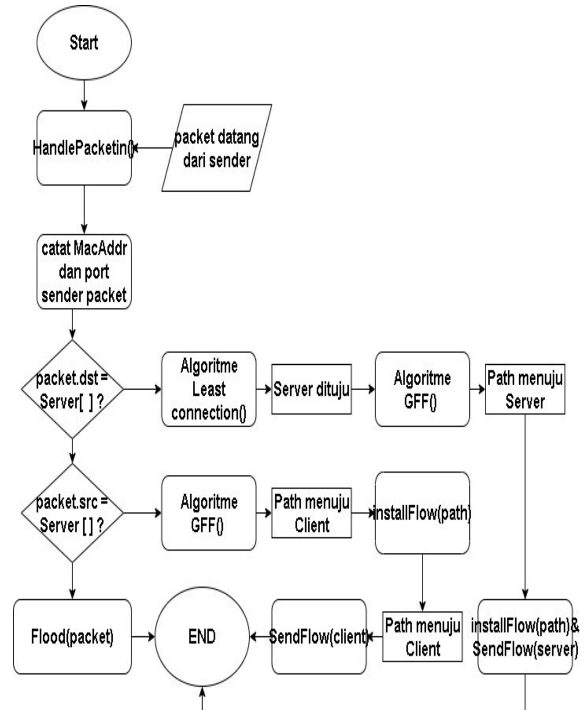
2.3. Skenario Pengujian

Metode evaluasi untuk penelitian ini berbentuk skenario pengujian yang akan dikirim *flow* dari empat *host* yang tersedia menuju virtual IP dimana keempat *server* terhubung. Pengiriman *flow* berjenis *tcp packet* menggunakan tools *iperf* dan dilakukan secara bergiliran dimulai dari *host* h1 sampai h4 lalu direkam hasilnya pada empat kali iterasi dengan waktu 20 - 50 detik untuk mendapatkan ukuran *flow* yang mencapai 10% dari ukuran *bandwidth*. Secara detail skenario pengiriman adalah sebagai berikut.

- *Request* berjenis paralel berjumlah 4 dikirim dari H1 menuju VIP selama 20 detik. (*flow* 1)
- 10 detik setelah *flow* 1 dimulai, *request* berjenis paralel berjumlah 2 dikirim dari H2 menuju VIP selama 30 detik. (*flow* 2)

Jeda 10 detik dari setiap flow yang dikirim untuk menguji sistem dalam melakukan pemilihan jalur, sedangkan panjang flow sebesar 20-50 detik untuk menguji kinerja algoritme least connection yang bekerja secara efisien jika total koneksi dari server berbeda. Dari skenario pengujian akan diuji parameter throughput, bandwidth, memory usage, dan cpu usage.

- 10 detik setelah *flow* 2 dimulai, *request* berjenis paralel berjumlah 2 dikirim dari H3 menuju VIP selama 40 detik. (*flow* 3)



Gambar 4. Diagram alir *Load balancing*

- 10 detik setelah *flow* 3 dimulai, *request* berjenis paralel berjumlah 4 dikirim dari H4 menuju VIP selama 50 detik. (*flow* 4)

Parameter *throughput* merupakan rasio yang menunjukkan seberapa banyak jumlah data yang terkirim dari *sender* ke *receiver* pada waktu tertentu. Parameter ini adalah salah satu parameter penentu tingginya kinerja dalam kinerja mekanisme *load balancing* (Wang dkk., 2014). Parameter *Bandwidth* adalah kapasitas suatu komponen jaringan seperti kabel ethernet yang dilewati oleh trafik paket data dengan jumlah tertentu. Dengan tingginya penggunaan *bandwidth* pada suatu jaringan menandakan bahwa *traffic* data sudah di distribusikan secara effisien (Kurose dan Ross, 2007). CPU *usage* adalah persentase penggunaan prosesor yang digunakan oleh aplikasi. CPU *usage* diukur berupa persentase dari 0 sampai 100 untuk setiap prosesor yang ada dalam sebuah perangkat. *Memory usage* adalah persentase penggunaan *memory* yang digunakan oleh aplikasi. *Memory usage* diukur berupa persentase dari 0 sampai 100 dari jumlah kapasitas *memory* yang digunakan.

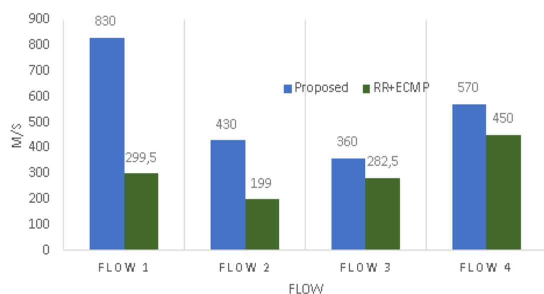
Hasil pengujian akan dibandingkan dengan algoritme *Round-Robin* (RR) dan ECMP. Algoritme RR adalah salah satu jenis algoritme tradisional yang sering dipakai pada metode server *load balancing* sebagai pembanding dikarenakan menggunakan skema yang sederhana. Cara kerja algoritme ini adalah *Flow* yang dikirim akan diteruskan ke *server* yang tersedia satu per satu secara urut (Mahmood dan Rashid, 2011). Algoritme ECMP merupakan algoritme dasar *load balancing* yang paling banyak

dipakai dalam jaringan *multipath* berbasis *multi layer* seperti DCN. Algoritme ECMP bekerja dengan membagi *flow* yang akan dikirim menjadi dua *flow* dengan beban yang sama ketika ada percabangan pada jaringan menggunakan penjadwalan *hash* atau *round robin* (Ye dkk., 2018).

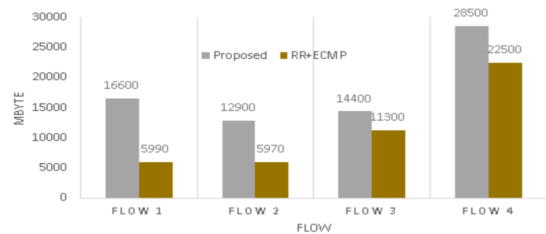
3. HASIL DAN PEMBAHASAN

3.1 *Throughput & Packet received*

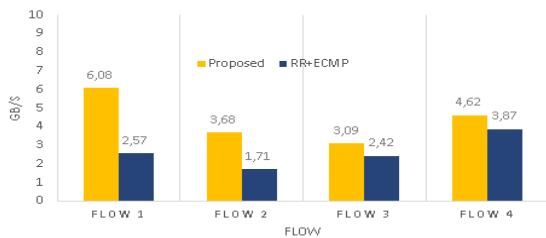
Hasil dari pengujian *throughput* didapatkan dari pengujian setiap *flow* yang ada seperti pada Gambar 5. Host sender diberikan label berdasarkan *mac* address yang dimiliki dan disederhanakan seperti pada Gambar 2. Keempat host yang bertindak menjadi sender adalah 0_0_2, 0_1_2, 1_0_2, dan 1_1_2. *Flow* 1 didapat dari pengiriman *host* 0_0_2, *flow* 2 didapat dari pengiriman *host* 0_1_2, dan seterusnya. Hasil yang diperoleh merupakan nilai rata-rata setiap *flow* dari waktu yang ditentukan. Nilai *throughput* dari metode yang diajukan mengungguli nilai dari metode *RR+ECMP*. Pada Gambar 6 adalah grafik dari hasil *packet received* dengan parameter M*byte*. Hasil evaluasi dari rata-rata nilai *throughput* metode kombinasi adalah 547,5 Mb/s dan rata-rata *byte received* sekitar 18,1Gb dibanding hasil dari rata-rata *throughput* metode ECMP dan RR adalah 307,75 Mb/s dan rata-rata *byte received* sekitar 11,4 Gb. Terdapat peningkatan nilai *throughput* sebesar 77,9% dibanding metode ECMP dan RR yang didapat dari perbedaan hasil rata-rata *throughput* pada metode kombinasi dan metode RR+ECMP. Hasil *throughput* dan *packet received* pada *flow* 1 dan *flow* 2 lebih tinggi dari pada nilai *host* lainnya dikarenakan *flow* pada *host* 0_0_2 dan 1_1_2 mengirimkan paralel *request* sejumlah 4, sedangkan *host* lainnya mengirim paralel *request* sejumlah 2. Hasil *throughput* dan *packet received* dari *flow* kedua dan ketiga juga turun dikarenakan jalur yang ada digunakan oleh dua atau lebih *flow* sehingga *bandwidth* yang digunakan juga akan ikut turun.



Gambar 5. Grafik hasil *Throughput*



Gambar 6. Grafik hasil *Packet received*



Gambar 7. Grafik hasil *Bandwidth*

3.2 *Bandwidth*

Pada Gambar 7 adalah grafik hasil penggunaan *bandwidth* dari semua *flow* yang diuji. Hasil diperoleh dari rata-rata penggunaan *bandwidth* setiap *flow* dari waktu yang ditetapkan di skenario pengujian. Penggunaan *bandwidth* dari semua *flow* berhasil mengisi lebih dari 1Gb yang merupakan 10% dari kapasitas *bandwidth* pada semua jalur, hal ini membuktikan bahwa penggunaan metode global first fit sukses menyeimbangkan *traffic* data yang berukuran lebih dari 10% kapasitas *bandwidth*. Hasil evaluasi dari rata-rata nilai *bandwidth* metode kombinasi adalah 4,36 Gb/s dibanding hasil dari rata-rata penggunaan *bandwidth* metode ECMP dan RR adalah 2,64 Gb/s. Dari hasil yang diperoleh terdapat peningkatan sebesar 65,2% dibanding hasil metode RR+ECMP yang didapat dari perbedaan hasil rata-rata penggunaan *bandwidth*. Data uji dari pengujian *throughput*, *bandwidth*, dan *packet received* dapat diamati dengan jelas pada Tabel 1.

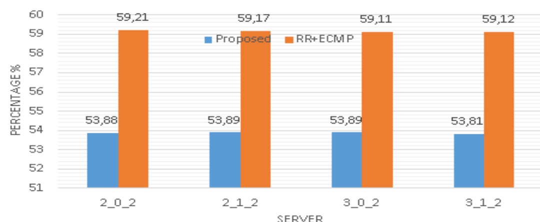
Tabel 1. Data hasil pengujian

Node	Detik (w)	paralel	jumlah packet	BW (Gb/s)	Tgpt (Mb/s)	
0_0_2	20	4	16600	6,08	830	proposed
0_1_2	30	2	12900	3,68	430	
1_0_2	40	2	14400	3,09	360	
1_1_2	50	4	28500	4,62	570	
0_0_2	20	4	5990	2,57	299,5	RR+ECMP
0_1_2	30	2	5970	1,71	199	
1_0_2	40	2	11300	2,42	282,5	
1_1_2	50	4	22500	3,87	450	

3.3 CPU dan *Memory Usage*

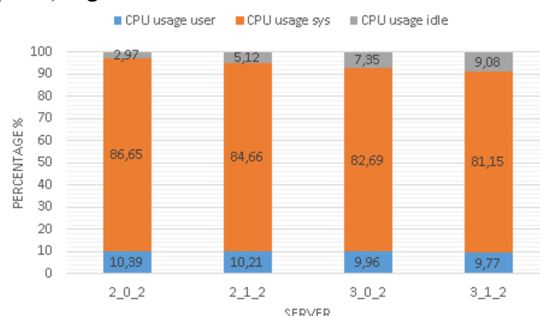
Parameter pengujian terakhir yaitu *memory usage* dan CPU *usage*. Hasil dari pengujian *memory usage* seperti pada Gambar 8. Satuan yang digunakan adalah persentase (%) dari pemakaian

memory swap+RAM dari total 100% kapasitas *memory* yang ada. Rata-rata penggunaan *memory* dari metode yang diajukan adalah 53,86% dan rata-rata penggunaan ECMP+RR adalah 59,15%. Dari hasil *memory usage* metode yang diajukan terdapat penurunan sebesar 9,81% dibanding hasil metode ECMP+RR. Kedua metode yang diuji juga menghasilkan penggunaan *memory* yang seimbang untuk setiap server. Hal ini membuktikan bahwa kedua metode dapat mengatur beban pada server dengan seimbang.



Gambar 8. Grafik hasil *memory usage*

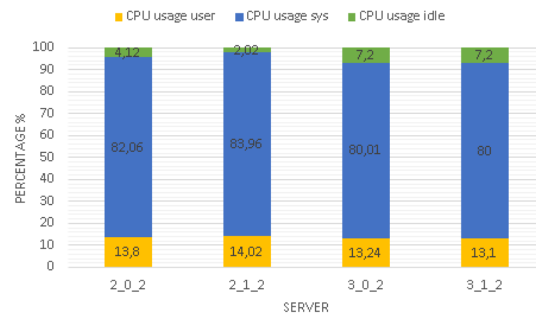
Untuk pengujian CPU *usage* digunakan persentase penggunaan *user, sys, dan idle*. *User* adalah penggunaan CPU pada aplikasi yang berjalan pada *user* tertentu, *sys* yaitu penggunaan CPU yang berjalan pada sistem kernel, dan *idle* yaitu CPU yang tidak digunakan. Total persentase dari ketiga parameter tersebut yaitu 100%. Hasil dari pengujian CPU *usage* pada metode yang diajukan dapat diamati pada Gambar 9. Persentase rata-rata dari *user* yaitu 10,08%, pada *sys* 83,78%, dan pada *idle* 6,13%. Sedangkan hasil dari pengujian cpu *usage* pada metode RR+ECMP dapat diamati pada Gambar 10, dengan hasil persentase dari parameter *user* sekitar 13,54%, pada *sys* sekitar 81,5%, dan pada *idle* yaitu 5,13%. Dari hasil rata-rata penggunaan CPU *usage* terjadi penurunan beban CPU sebesar 34,29% pada *user* CPU dibanding hasil metode ECMP+RR. Hasil yang didapat dari kedua metode dapat diketahui bahwa hampir semua kapasitas CPU (80%) digunakan.



Gambar 9. Grafik hasil proposed cpu *usage*

4. KESIMPULAN

Penelitian ini mengajukan metode kombinasi *link* dan server *load balancing* pada jaringan DSS guna meningkatkan utilisasi jaringan. Penerapan metode kombinasi yang diajukan berhasil meningkatan nilai *throughput* rata-rata 77% lebih tinggi dari hasil metode ECMP.



Gambar 10. Grafik hasil rr+ecmp cpu *usage*

Pengujian penggunaan *bandwidth* menunjukkan peningkatan dengan rata-rata 65% lebih tinggi dari hasil metode ECMP. Penggunaan *bandwidth* dari kedua metode berhasil mengisi lebih dari 1Gb yang merupakan 10% dari kapasitas *bandwidth* pada semua jalur, hal ini membuktikan bahwa penggunaan metode global first fit sukses mengatur *traffic* data yang berukuran lebih dari 10% kapasitas *bandwidth*. Hasil Penggunaan *cpu* dan *memory* pada *server* di metode kombinasi ini juga terjadi penurunan beban *cpu* sebesar 34,2% dan penurunan beban penggunaan *memory* sebesar 9,81% dibanding metode ECMP dan RR meskipun pembagian penggunaan *cpu* dan *memory* dari kedua metode seimbang. Dengan demikian algoritme *global first fit* dengan sukses mendeteksi adanya *bottleneck* pada jaringan dan menyeimbangkan *traffic* ke semua jalur yang ada. Dari hasil evaluasi, penerapan metode kombinasi metode *server* dan *link load balancing* berhasil meningkatkan utilisasi jaringan dan juga mengurangi beban server.

DAFTAR PUSTAKA

AL-FARES, M., RADHAKRISHNAN, S., RAGHAVAN, B., HUANG, N., DAN VAHDAT, A. (2010): Hedera, *Hedera: Dynamic Flow Scheduling for Data Center Network*,130–131. https://doi.org/10.1201/b16160-86

CHEN, W., LI, H., MA, Q., DAN SHANG, Z. (2014): Design and implementation of server cluster dynamic load balancing in virtualization environment based on OpenFlow, *ACM International Conference Proceeding Series*, 2014-May(May), 691–697. https://doi.org/10.1145/2619287.2619288

DIMAKIS, A. G., GODFREY, P. B., WU, Y., WAINWRIGHT, M. J., DAN RAMCHANDRAN, K. (2010): Network coding for distributed storage systems, *IEEE Transactions on Information Theory*, 56(9), 4539–4551. https://doi.org/10.1109/TIT.2010.2054295

FLEXERA (2020): 2020 Flexera State of the Cloud Report, 1–19.

GUILLEN, L., IZUMI, S., ABE, T., SUGANUMA, T., DAN MURAOKA, H. (2018a): SDN-based hybrid server and link load balancing in multipath distributed storage systems, *IEEE/IFIP Network Operations and Management Symposium: Cognitive Management in a Cyber World, NOMS 2018*, 1–6. https://doi.org/10.1109/NOMS.2018.8406286

GUILLEN, L., IZUMI, S., ABE, T., SUGANUMA, T., DAN MURAOKA, H. (2018b): SDN implementation of multipath discovery to improve network performance in distributed storage systems, *2017 13th International Conference on Network and Service Management, CNSM 2017*, 2018-January, 1–4. https://doi.org/10.23919/CNSM.2017.8256054

KANEKO, S., NAKAMURA, T., KAMEI, H., DAN MURAOKA, H. (2016): A guideline for data placement in heterogeneous distributed storage systems, *Proceedings - 2016 5th IIAI International Congress on Advanced Applied Informatics, IIAI-AAI 2016*, (disk M), 942–945.https://doi.org/10.1109/IIAI-AAI.2016.162

KREUTZ, D., RAMOS, F. M. V., VERISSIMO, P. E., ROTHENBERG, C. E., AZODOLMOLKY, S., DAN UHLIG, S. (2015): Software-defined networking: A comprehensive survey, *Proceedings of the IEEE*, 103(1), 14–76.
https://doi.org/10.1109/JPROC.2014.2371999

KUROSE, J., DAN ROSS, K. (2007): Chapter 1 : Introduction Chapter 1 Background Chapter 1 : roadmap “ Cool ” internet appliances What ’ s a protocol ?, 1–20.

LIU, Y., RAMESHAN, N., MONTE, E., VLASSOV, V., DAN NAVARRO, L. (2015): ProRenaTa: Proactive and reactive tuning to scale a distributed storage system, *Proceedings - 2015 IEEE/ACM 15th International Symposium on Cluster, Cloud, and Grid Computing, CCGrid 2015*, 453–464. https://doi.org/10.1109/CCGrid.2015.26

MAHMOOD, A., DAN RASHID, I. (2011): Comparison of load balancing algorithms for clustered web servers, *2011 International Conference on Information Technology and Multimedia: “Ubiquitous ICT for Sustainable and Green Living”, ICIM 2011*, (November). https://doi.org/10.1109/ICIMU.2011.6122721

MUSTAFA, M. E. (2017): Load Balancing Algorithms Round-Robin (Rr), Least-Connection , and Least Loaded Efficiency, *Computer Science and Telecommunications*, diperoleh melalui situs internet: http://gesj.internetacademy.org.ge/download.php?id=2886.pdf&t=1, 51(1), 25–29.

SUH, C., DAN RAMCHANDRAN, K. (2011): Exact-Repair MDS Code Construction Using, 57(3), 1425–1442.

WANG, T., SU, Z., XIA, Y., DAN HAMDI, M. (2014): Rethinking the data center networking: Architecture, network protocols, and resource sharing, *IEEE Access*, 2, 1481–1496. https://doi.org/10.1109/ACCESS.2014.2383439

XIA, W., WEN, Y., DAN FOH, C. H. (2018): A Survey on Software-Defined Networking, *Asian Pacific Journal of Reproduction*, **7**(2), 72–78. https://doi.org/10.4103/2305-0500.228016

YE, J. L., CHEN, C., DAN HUANG CHU, Y. (2018): A Weighted ECMP Load Balancing Scheme for Data Centers Using P4 Switches, *Proceedings of the 2018 IEEE 7th International Conference on Cloud Networking, CloudNet 2018*, 1–4. https://doi.org/10.1109/CloudNet.2018.8549549

ZHANG, J., YU, F. R., WANG, S., HUANG, T., LIU, Z., DAN LIU, Y. (2018): Load balancing in data center networks: A survey, *IEEE Communications Surveys and Tutorials*, 20(3), 2324–2325. https://doi.org/10.1109/COMST.2018.2816042

Halaman ini sengaja dikosongkan