Analisa Kinerja Apache dan Nginx dalam Arsitektur Microservice Menggunakan Siege

Howard Christopher Yoel Unsong[1], Justinus Andjarwirawan[2]
Program Studi Informatika Fakultas Teknologi Industri Universitas Kristen Petra
Jl. Siwalankerto 121 – 131 Surabaya 60236
Telp. (031) – 2983455, Fax. (031) – 8417658
yoelhoward@gmail.com[1], justin@petra.ac.id[2]

ABSTRAK

Dari permasalahan yang ada yaitu bagaimana cara memastikan bahwa permintaan API melalui jaringan internal tidak akan memperlambat keseluruhan waktu respon dan bagaimana hasil menggunakan *load balancer* tanpa memikirkan *scalability*.

Skripsi ini bertujuan untuk menganalisa kinerja web service pada arsitektur *microservice* antara Apache dan Nginx dengan menggunakan siege pada IP versi 4. Metode yang dipakai adalah kualitatif yang membedakan dengan penelitian lainnya adalah penelitian ini menggunakan arsitektur *microservice* dan menggunakan siege sebagai alat untuk menjalankan skenario-skenario yang diuji.

Hasil dari percobaan ini adalah web server Apache lebih unggul pada *response time* dan *throughput* tetapi untuk transaksi yang didapat web server Nginx lebih unggul. Kesimpulan sesuai dengan penekanan dalam rumusan masalah jadi kinerja *web server* Apache lebih baik.

Kata Kunci: nginx, apache, *throughput, response time*, siege.

ABSTRACT

From the existing problem, there is how to make sure that an API with an internal network will not slow down the response time overall and how to use load balancer results without scalability.
This thesis aims to solve this problem by analyzing the performance of the web service on the microservice architecture between apache and nginx by using siege on IP version 4. The method used is qualitative, which distinguishes it from other studies, this study uses a microservice architecture and uses siege as a tool to run scenarios that are tested.
The result is that the apache web server works better than the nginx web server in response time and throughput, but the nginx web server hits transactions more than apache . The conclusion is in accordance with the emphasis in the problem formulation, so that the performance of the apache web server is better.

Keywords: *nginx, apache, throughput, response time, siege.*

1. PENDAHULUAN

Arsitektur Mikroservis adalah salah satu cara yang membuat satu aplikasi besar dipecah-pecah menjadi beberapa bagian layanan yang lebih kecil dan terhubung satu sama lainnya dengan menggunakan API [16]. Arsitektur Mikroservis bergantung dengan HTTP untuk mengkoneksikan atau menyambungkan setiap *service*nya, maka web server merupakan *core* yang penting dalam melakukan proses. Dengan dimulainya trend yang menggunakan *Microservice* maka akan dilakukan pengujian performa dari web server. Pengujian kali ini akan menggunakan Apache dengan Nginx sebagai objek.

Apache adalah sebuah *web server* yang dibuat tahun 1995 lebih tua dibanding dengan Nginx, Apache ditujukan untuk menyediakan web server yang aman, stabil dan fleksibel. Sebagai contoh, Apache mengelola modul *multi-process* untuk mempertahankan koneksi yang sama namun dengan cara yang berbeda. Hal tersebut sangat menguntungkan karena administrator dapat mengaplikasikan kriteria untuk tiap koneksi [2], tetapi server ini membutuhkan lebih banyak *resources* karena kerjanya cukup berat. Sedangkan teknologi Nginx merupakan pengembangan dari Apache sehingga sekarang ini berkembang secara pesat sejak dipublikasi pada tahun 2004, sehingga sudah mulai banyak dipakai pada umum, Nginx dikenal dengan arsitektur *asynchronous* dan *events-driven architecture*. sudah pasti Nginx lebih dikenal karena server tidak terlalu membutuhkan banyak *resource* sehingga Nginx lebih ringan saat digunakan dibandingkan dengan Apache serta Nginx juga lebih efisien ketika menampilkan konten statis.

Untuk mengetahui kinerja dari Apache dengan Nginx digunakan *tools* yang paling sering digunakan yaitu Siege dan Httperf. Siege adalah *open source stress* atau *regression test* dan *benchmark utility* jadi bisa ditujukan pada satu URL dengan pengguna yang dapat ditentukan jumlah pengguna simulasinya atau dapat membaca banyak URL ke memori dan ditujukan secara bersamaan. Program akan melaporkan jumlah hit yang direkam, *byte* yang ditransfer, waktu respon, konkurensi dan status pengembalian. Untuk kelebihannya fitur dapat dikonfigurasi dengan opsi berbagai baris perintah yang juga menyertakan nilai *default* untuk meminimalkan kerumitan pemanggilan program, sedangkan untuk kekurangannya tidak bisa digunakan pada yang berbasis *operating system* microsoft karena siege mengandalkan fitur POSIX.1b, tetap bisa diterapkan pada *operating system* windows [11]. Sedangkan untuk Httperf adalah program yang berfungsi sebagai pengukur kinerja atau performa dari Web Server, yang diciptakan oleh David Mossberger dari HP Labs. Untuk kelebihan dari Httperf yaitu memiliki fitur yang bisa dirubah-rubah ketika mengeksekusi pekerjaan sesuai dengan variabel yang diberikan, untuk kekurangan tidak bisa multitasking karena penggunaan CPU, semua memori dialokasikan untuk menjalankan httperf [11].

Dengan dilihat dari permasalahan yang ada bahwa bagaimana cara memastikan bahwa permintaan API melalui jaringan internal tidak akan memperlambat keseluruhan waktu respons [13] dan bagaimana hasil menggunakan *load balancer* tanpa memikirkan scalability. Untuk membuktikan masalah tersebut maka pengujian kali ini akan diujikan pada dua web server terkenal sebagai pembanding yaitu Apache dan Nginx untuk dilihat waktu responnya beserta dengan performanya, dengan

mempertimbangkan kekurangan dan kelebihan dari Httperf dan Siege sebagai *tools* maka dipilihlah Siege sebagai tools untuk melakukan pengujian ini. Pengujian ini akan dilakukan pada web yang berbasiskan mikroservis yang dibuat khusus untuk melakukan berbagai skenario yang akan diujikan. Jika diambil dari web studi kasus maka akan kesusahan untuk mengetahui apakah web studi kasus tersebut menggunakan arsitektur mikroservis atau monolitik.

2. DASAR TEORI

2.1 Docker

Docker adalah platform terbuka untuk mengembangkan, mengirim, dan menjalankan aplikasi. Docker memungkinkan untuk memisahkan aplikasi dari struktur awal sehingga dapat mengirimkan perintah dengan cepat. Dengan Docker bisa melakukan untuk mengolah struktur dengan cara yang sama seperti mengelola struktur pada aplikasi semula. Dengan memanfaatkan metode yang ada untuk pengiriman, pengujian, dan penerapan kode dengan cepat admin dapat secara signifikan mengurangi penundaan antara penulisan kode dan menjalankannya dalam perintah yang dilakukan [8].

2.2 Kontainer

Kontainer adalah sebuah *Operating System* Virtual yang berlaku sebagai pembungkus yang membungkus aplikasi bersama dengan *dependency* dan *environment*-nya. Setiap kontainer ini memiliki proses yang tertutup dari kontainer lain maupun dengan host OS. Prinsip kontainer ini diibaratkan kapal kargo, yang mana kapal kargo itu adalah sebuah aplikasi yang kompleks dan kontainernya adalah *service-service* kecil yang banyak dimuat dan saling terhubung dengan menggunakan Rest API untuk menjalankan aplikasinya [10].

2.3 Apache

Apache adalah sebuah *software server* yang dirancang untuk dijalankan sebagai proses *daemon* mandiri, apache digunakan untuk membuat koneksi antara server dengan *web browser* dengan protokol *HyperText Transfer* (HTTP). Apache mempunyai fitur yang cukup untuk masa sekarang yaitu bisa mengkonfigurasikan *error message* serta autentikasi yang berdasarkan basis data, saat digunakan Apache akan membuat kumpulan proses untuk menangani permintaan [3].

Arsitektur dari Apache adalah klien-server yang berbasiskan *thread* yang memiliki modul, modul ini berguna untuk mengaktifkan atau menonaktifkan fitur tambahan yang dilakukan oleh administrator, modul tersebut antara lain keamanan, URL *rewriting* dan otentikasi password dan yang lainnya [6].

2.4 Nginx

NGINX merupakan sebuah software server HTTP dan Proxy yang *opensource* yang dapat digunakan juga sebagai proxy IMAP/POP3, Nginx banyak dipakai karena stabil, konsumsi sumber daya rendah dan memiliki performa yang bagus. Nginx memiliki beberapa fitur unggulan yaitu *reverse proxy multiple protocols* seperti SCGI, UWSGI dan memcached sedangkan untuk video bisa mendukung seperti FLV, HLS dan mp4 dan http/2 gateway serta bisa menjadi load balancer. *Opensource code* nginx pertama ditulis oleh orang Rusia yang bernama Igor Sysoev pada tahun 2002 dan dirilis ke publik pada tahun 2004 untuk menjawab permasalahan C10K. [15] Nginx menggunakan arsitektur *event-driven* dan asinkron, dimana *thread* yang sama akan di proses dalam sebuah bagian yaitu *worker process*, dan *worker process* sendiri terdiri atas *worker connection*. Dan semuanya bertujuan untuk menangani *request* yang muncul dari *thread*. *Worker connection* akan mengirimkan permintaan kepada *worker process* dan kepada *master process*, *worker connection* bisa menampung sampai 1024 *request* yang sama. Sedangkan bagian yang untuk menghasilkan hasil akhir adalah *master process*, hasil yang keluar merupakan hasil dari permintaan atau *request* tersebut [7].

2.5 *Response Time*

Response time adalah waktu yang diperlukan untuk menjalankan satu program yang dibutuhkan. Hal yang mempengaruhi *response time* dalam kinerja optimalnya adalah kompleksitas dari suatu perintah yang diberikan jika perintah semakin besar maka *response time* juga semakin lambat, *user* yang menggunakan juga berpengaruh dalam *response timenya* baik dari segi skill dan jumlah *user* yang mengakses, dan kemampuan dari perangkatnya sendiri semakin bagus spesifikasi dari perangkat maka semakin bagus juga response time yang diperoleh. Pada situasi yang baik adalah jika semakin besar *throughputnya* dan semakin kecil *response timenya* untuk sebuah media penyimpanan data, maka semakin baguslah pula performanya [12].

2.6 *Throughput*

Throughput merupakan bandwidth yang sesungguhnya. Jika *bandwidth* adalah batas maksimal maka *throughput* adalah data yang sesungguhnya, contohnya jika memakai Internet dengan menggunakan *bandwidth* 10 Mbps dan kecepatan yang tertera adalah 5 Mbps pada saat *download* file kecepatan yang tertera 5 Mbps ini yang disebut dengan *throughput*. *Throughput* bisa diukur dengan kondisi jaringan dan waktu tertentu yang dipakai untuk *transfer file* dengan ukuran tertentu. Contoh jika *bandwidthnya* 32 kbps lalu ketika mengunduh file 265 kbps dari Internet, file seharusnya ter*download* di komputer hanya dalam waktu 8 detik saja dari perhitungan 256/32, lalu jika file yang terunduh dalam waktu 10 detik maka *bandwidth* yang *real*nya adalah 256 kb / 10 detik yaitu 25,6 kbps [4].

2.7 Siege

Siege adalah alat yang diperuntukan untuk sebagai pengukur dan penguji *web server* untuk mengetahui kestabilan, performa dan perubahan atau perbaikan kode atau arsitektur, siege dipergunakan sebagai alat untuk mensimulasikan *user* yang akan mengakses sebuah aplikasi web sehingga bisa digunakan untuk menyesuaikan parameter yang diujikan [1]

2.8 Arsitektur Mikroservis

Arsitektur Mikroservis adalah arsitektur yang rancang bangunnya terbagi dari suatu aplikasi kompleks menjadi aplikasi yang lebih sederhana atau lebih kecil dan yang saling terhubung dengan menggunakan API. Setiap mikroservis memiliki arsitektur sendiri yang terdiri dari logika dan berbagai bersama dengan *dependency* dan *environment*-nya. Pola dari arsitektur mikroservis berpengaruh dalam hubungan antara aplikasi dengan database, masing-masing *services* memiliki skema database sendiri sehingga ada keuntungan tersendiri dari layanan mikroservis. Selain itu, *service-service* tersebut bisa menggunakan jenis database dan bahasa pemrograman yang bisa disesuaikan dengan keperluannya.

2.9 *Load Balancer*

Load Balancer adalah sistem yang bertugas untuk membagi *load* ke *node-node* yang dianggap bisa menjalankan load tersebut.

Pembagian modelnya didasarkan pada layer 3 dan 4 yaitu *layer transport* atau layer ke 7 sebagai *layer* aplikasi pada OSI Model. Dengan layer 3 dan 4, pembagian yang dilakukan adalah berdasarkan Port, IP dan paket TCP/UDP. Pada layer ke 3 dan ke 4 ini seringkali banyak dipergunakan sebagai arsitektur web server, karena mudah untuk di konfigurasi serta memenuhi syarat untuk mem*balancing* data serta dipakai untuk mem*balancing service* seperti database, DNS dan middleware [5].

3. PERANCANGAN

3.1 Alur Kerja

Pertama pengguna akan melakukan akses ke internet lalu *load balancer* mengarahkan kepada layanan login yang sebelumnya dibuat dengan docker untuk arsitektur mikroservisnya, pada layanan login dengan digunakan sistem token, setelah itu pengguna akan diarahkan oleh *load balancer* untuk masuk kepada layanan produk yang sebelumnya dibuat dengan docker untuk arsitektur mikroservisnya pada layanan produk ini menggunakan sistem GET, setelah itu pengguna akan diarahkan oleh *load balancer* untuk masuk kepada layanan keranjang yang sebelumnya dibuat dengan docker untuk arsitektur mikroservisnya pada layanan keranjang ini menggunakan sistem GET, setelah itu pengguna akan diarahkan oleh *load balancer* untuk masuk kepada layanan pembayaran yang sebelumnya dibuat dengan docker untuk arsitektur mikroservisnya pada layanan pembayaran ini menggunakan sistem GET. Pada sistem yang dibuat kebanyakan menggunakan GET karena Siege akan melakukan pengetesan terhadap GET *request*.

3.2 *User Interface*

Setelah mempelajari literatur-literatur tersebut, saatnya pengerjaan dimulai dengan membuat desain web untuk *user-interface*nya yang dibutuhkan sebagai *service* yang akan digunakan. *User-interface* yang akan dipakai adalah *user-interface* yang dianggap sering dikunjungi, yaitu halaman untuk Login, halaman untuk Produk, halaman untuk Keranjang, halaman untuk Pembayaran

3.3 *Database*

Setelah halaman-halaman *user-interface* sudah terbuat, berikutnya adalah pembuatan database, pada saat ini database yang dibutuhkan adalah sesuai dengan halaman-halaman *user-interface* yaitu database untuk pengguna, produk, keranjang dan pembayaran.

3.4 Kontainer

Setelah database dibuat, hal yang dikerjakan berikutnya adalah membuat kontainer-kontainer dengan menggunakan bantuan docker. Kontainer dibuat sebagai wadah dari masing-masing layanan yang berisi *user-interface*, database, API, *dependency* dan *environment* lainnya yang menunjang layanan.

3.5 Load Balancer

Setelah menyelesaikan pembuatan kontainer maka selanjutnya melakukan pengaturan pada *load balancer* yaitu mengatur API, Port yang dipakai, dan kontainer *image* docker untuk setiap layanan.

4. ANALISA

Setelah pembuatan *user interface*, database, kontainer dan *Load Balancer*. Saatnya masuk dalam percobaan, percobaan dilakukan menggunakan aplikasi Siege sebagai aplikasi *benchmark*. Tujuan dari uji coba ini adalah perbandingan hasil *throughput* dan *response time* antara kedua web server nginx dengan apache.

4.1 Implementasi Siege

Untuk melakukan percobaan dilakukan beberapa langkah sebagai berikut :

1. Memasukkan perintah pada siege dengan syntax siege -c –t –i Link
   A. -c digunakan untuk concurrent user (akan selalu bertambah sampai pada titik stabil).
   B. -t digunakan untuk menentukan waktu berapa lama saat pengujian dilakukan.
   C. -i digunakan untuk mensimulasikan penggunaan internet untuk transaksi
   D. Link digunakan untuk mengisi alamat *service* yang akan diuji.
2. Hasil, jika hasil sudah keluar maka akan melalui percabangan untuk pengecekan.
3. Percabangan, jika hasil tidak memenuhi kriteria maka kembali ke langkah pertama, jika berhasil maka selesai.
4. Kriteria dikatakan berhasil jika memenuhi :
   A. Web server tetap stabil memproses request.
   B. Hasil *throughput* tinggi.
   C. Hasil *response time* rendah.

4.2 Hasil

Metode pengujian yang dilakukan untuk uji coba sistem ini adalah dengan mencoba berbagai jumlah *concurrent user* dan waktu untuk di cobakan pada Siege *Benchmarking*. Metode pengujian ini mengutamakan pada hasil *response time* dan *throughput* yang dihasilkan. Setiap web server akan dilakukan 3 kali percobaan dengan *concurrent user* yang terdiri dari 250000, 500000 dan 750000, angka yang digunakan cukup besar karena untuk bisa menghasilkan perbedaan perbedaan hasil pengetesan yang cukup berbeda sedangkan untuk waktu percobaan akan terdiri dari 2 menit, 4 menit dan 6 menit yaitu durasi yang digunakan untuk melakukan percobaan sehingga menjadi 9 kombinasi antara *concurrent user* dengan waktu, angka-angka yang disebutkan pada *concurrent user* dan waktu adalah angka yang terpilih karena jika terlalu sedikit untuk *concurrent user* maka perbedaan yang terjadi akan sangat kecil sehingga dibutuhkan 250000, 500000 dan 750000 untuk dengan mudah dapat melihat perbedaan diantara ketiganya dengan waktu 2,4 dan 6 menit, selain itu harus menyesuaikan dengan kekuatan atau kesanggupan peralatan yang digunakan untuk melakukan percobaan ini.

Untuk hasil percobaan akan mengeluarkan berbagai hal, yaitu yang pertama adalah *Transaction* untuk mengetahui berapa kali transaksi terjadi yaitu angka yang menunjukkan berapa kali terjadinya transaksi lalu *Availability* untuk persentase dari ketersediaan transaksi dengan cara jumlah transaksi dibagi dengan waktu yang dibutuhkan saat proses transaksi *lalu Elapsed time* untuk mengetahui durasi waktu pengujian *Data transferred* untuk mengetahui berapa banyak data *payload* yang terkirim *Response time* untuk mengetahui jumlah waktu saat mengeksekusi dimulai dari transmisi lalu pemrosesan lalu *rendering* lalu *Transaction rate* untuk mengetahui jumlah persentase transaksi per detik berikutnya adalah *Throughput* untuk mengetahui jumlah dari kecepatan data yang dikirim lalu *Concurrency* untuk melihat jumlah rata-rata koneksi yang meningkat saat kinerja server menurun berikutnya *Successful transactions* untuk mengetahui jumlah angka transaksi yang berhasil lalu *Failed transaction* untuk mengetahui jumlah angka transaksi yang tidak berhasil lalu *Longest transaction* untuk mengetahui proses transaksi yang menghasilkan waktu terlama lama transaksi dalam satuan detik dan yang terakhir adalah *Shortest transaction* untuk mengetahui

proses transaksi yang menghasilkan waktu tercepat transaksi dalam satuan detik *Web Server* Apache

Percobaan untuk *web server* Apache ini melakukan 9 kombinasi percobaan dengan jumlah *concurrent user* yang terdiri dari 250000, 500000 dan 750000, sedangkan untuk waktu akan terdiri dari 2 menit, 4 menit dan 6 menit.

Jika dilihat berdasarkan waktu 2 menit, dari perhitungan rata-rata untuk *response time* dimenangkan *web server* Apache dan untuk *throughput* relatif karena bernilai sama jika di rata-rata, *throughput* rendah mungkin dikarenakan simulasi internet kurang baik, untuk *transaction* jika di rata-rata yang lebih bagus adalah milik *web server* Nginx, untuk lebih jelas dapat dilihat pada tabel 1 berikut.

Tabel 1. Hasil Pengujian dengan Waktu 2 Menit

Concurrent User	250000		500000		750000	
Time	2 minutes		2 minutes		2 minutes	
Web Server	Nginx	Apache	Nginx	Apache	Nginx	Apache
Response Time	2.25s	2.47s	2.26s	2.37s	2.25s	2.30s
Throughput	0.01 MB/s	0.01 MB/s	0.01 MB/s	0.01 MB/s	0.01 MB/s	0.01 MB/s
Transaction	53	48	53	51	53	52

Jika dilihat berdasarkan waktu 4 menit, dari perhitungan rata-rata untuk *response time* dimenangkan *web server* Apache dan untuk *throughput* relatif karena bernilai sama jika di rata-rata, *throughput* rendah mungkin dikarenakan simulasi internet kurang baik, untuk *transaction* jika di rata-rata yang lebih bagus adalah milik *web server* Nginx, untuk lebih jelas dapat dilihat pada tabel 2 berikut.

Tabel 2. Hasil Pengujian dengan Waktu 4 Menit

Concurrent User	250000		500000		750000	
Time	4 minutes		4 minutes		4 minutes	
Web Server	Nginx	Apache	Nginx	Apache	Nginx	Apache
Response Time	2.25s	2.43s	2.43s	2.36s	2.35s	2.34s
Throughput	0.01 MB/s	0.00 MB/s	0.01 MB/s	0.01 MB/s	0.00 MB/s	0.00 MB/s
Transaction	106	77	100	101	81	79

Jika dilihat berdasarkan waktu 6 menit, dari perhitungan rata-rata untuk *response time* dimenangkan *web server* Nginx dan untuk *throughput* relatif karena bernilai sama jika di rata-rata, *throughput* rendah mungkin dikarenakan simulasi internet kurang baik, untuk *transaction* jika di rata-rata yang lebih bagus adalah milik web server Apache, untuk lebih jelas dapat dilihat pada tabel 3 berikut.

Tabel 3. Hasil Pengujian dengan Waktu 6 Menit

Concurrent User	250000		500000		750000	
Time	6 minutes		6 minutes		6 minutes	
Web Server	Nginx	Apache	Nginx	Apache	Nginx	Apache
Response Time	2.33s	2.38s	2.83s	2.37s	3.22s	2.29s
Throughput	0.01 MB/s	0.01 MB/s	0.00 MB/s	0.01 MB/s	0.00 MB/s	0.01 MB/s
Transaction	155	151	127	151	113	157

Jika dilihat berdasarkan jumlah 250000 *concurrent user*, dari perhitungan rata-rata untuk *response time* dimenangkan web server Apache dan untuk *throughput* relatif karena bernilai sama jika di rata-rata, throughput rendah mungkin dikarenakan simulasi internet kurang baik, untuk *transaction* jika di rata-rata yang lebih bagus adalah milik *web server* Nginx, untuk lebih jelas dapat dilihat pada tabel 4 berikut.

Tabel 4. Hasil Pengujian dengan 250000 Concurrent User

Concurrent User	250000		250000		250000	
Time	2 minutes		4 minutes		6 minutes	
Web Server	Nginx	Apache	Nginx	Apache	Nginx	Apache
Response Time	2.25s	2.47s	2.25s	2.43s	2.33s	2.38s
Throughput	0.01 MB/s	0.01 MB/s	0.01 MB/s	0.00 MB/s	0.01 MB/s	0.01 MB/s
Transaction	53	48	106	77	155	151

Jika dilihat berdasarkan jumlah 500000 concurrent user, dari perhitungan rata-rata untuk *response time* dimenangkan *web server* Nginx dan untuk *throughput* relatif karena bernilai sama jika di rata-rata, *throughput* rendah mungkin dikarenakan simulasi internet kurang baik, untuk *transaction* jika di rata-rata yang lebih bagus adalah milik *web server* Apache, untuk lebih jelas dapat dilihat pada tabel 5 berikut.

Tabel 5. Hasil Pengujian dengan 500000 Concurrent User

Concurrent User	500000		500000		500000	
Time	2 minutes		4 minutes		6 minutes	
Web Server	Nginx	Apache	Nginx	Apache	Nginx	Apache
Response Time	2.26s	2.37s	2.43s	2.36s	2.83s	2.37s
Throughput	0.01 MB/s	0.01 MB/s	0.01 MB/s	0.01 MB/s	0.00 MB/s	0.01 MB/s
Transaction	53	51	100	101	127	151

Jika dilihat berdasarkan jumlah 750000 *concurrent user*, dari perhitungan rata-rata untuk *response time* dimenangkan *web server* Nginx dan untuk *throughput* relatif karena bernilai sama jika di rata-rata, *throughput* rendah mungkin dikarenakan simulasi internet kurang baik, untuk *transaction* jika di rata-rata yang lebih bagus adalah milik *web server* Apache, untuk lebih jelas dapat dilihat pada tabel 6 berikut.

Tabel 6. Hasil Pengujian dengan 750000 Concurrent User

Concurrent User	750000		750000		750000	
Time	2 minutes		4 minutes		6 minutes	
Web Server	Nginx	Apache	Nginx	Apache	Nginx	Apache
Response Time	2.25s	2.30s	2.35s	2.34s	3.22s	2.29s
Throughput	0.01 MB/s	0.01 MB/s	0.00 MB/s	0.00 MB/s	0.00 MB/s	0.01 MB/s
Transaction	53	52	81	79	113	157

5. PENUTUP

Berdasarkan hasil implementasi dan percobaan yang telah dilakukan, maka bisa disimpulkan bahwa berdasarkan hasil perhitungan rata-rata untuk *response time* dan *throughput* maka *web server* Apache lebih unggul dibandingkan dengan *web server* Nginx, tetapi berdasarkan untuk jumlah dari transaksi *web server* Nginx lebih banyak daripada *web server* Apache.

Untuk saran, berdasarkan hasil implementasi dan percobaan yang telah dilakukan, ada beberapa saran untuk kedepannya yang bisa dilakukan adalah sebagai berikut :

1. Algoritma yang digunakan pada *load balancer* bisa di coba dengan algoritma lainnya sebagai salah satu cara untuk menggali potensi yang menghasilkan hasil yang sama tetapi lebih akurat maupun menghasilkan hasil yang baru pada saat percobaan berikutnya.

2. Untuk jumlah *concurrent user* dan waktu bisa lebih diperbanyak sesuai dengan kemampuan alat yang digunakan untuk percobaan, sehingga bisa menghasilkan perbedaan yang signifikan antara percobaan dengan percobaan lainnya yang bertujuan lebih mudah untuk dianalisis berikutnya.

6. REFERENSI

[1] Antolovic, Zoran. 2017. Web App Performance Testing with Siege: Plan, Test, Learn. URl= https://www.sitepoint.com/web-app-performance-testing-siege-plan-test-learn/

[2] Apache HTTP Server Project. 2020. Apache Hypertext Transfer Protocol Server. URl= https://httpd.apache.org/docs/2.4/programs/httpd.html

[3] Apache. 2021. Apache HTTP Server Project. URl= https://httpd.apache.org/

[4] Asfihan, Akbar. 2019. Throughput Adalah : Cara Kerja dan Opsi-opsi Untuk Meningkatkan Throughput. URl= https://adalah.co.id/throughput/

[5] Boyke, Dian. 2016. Lebih Jauh Tentang Load Balancer (L3/4 dan L7). URl= https://indosystem.com/blog/lebih-jauh-tentang-load-balancer/

[6] C, Ariata. 2020. Apa itu apache? Pengertian serta Kelebihan dan Kekurangannya. URl= https://www.hostinger.co.id/tutorial/apa-itu-apache

[7] C, Ariata. 2019. Apa itu NGINX? Dan Bagaimana Cara Kerjanya?. URl= https://www.hostinger.co.id/tutorial/apa-itu-nginx

[8] Docker. 2020. Docker Overview. URl= https://docs.docker.com/get-started/overview/

[9] inixindo jogja. 2019. Container VS Virtual Machine. URl= https://inixindojogja.co.id/container-vs-virtual-machine/

[10] Irza, Intan Ferina. Zulhendra, Efrizon. 2017. *Analisis Perbandingan Kinerja Web Server Apache dan Nginx Menggunakan Httperf Pada Portal Berita (Studi Kasus beritalinux.com).* URl= http://ejournal.unp.ac.id/index.php/voteknika/article/view/8489

[11] Martha, Ragil., Yanuar Firdaus, Kusuma Ayu Laksitowening. 2010. *Analisa Perbandingan Response Time dan Throughput Pada XML dan DBMS Sebagai Media Penyimpanan Data.* URl= https://journal.uii.ac.id/Snati/article/viewFile/1931/1706

[12] Mishra, James. 2015. How do microservice system architectures avoid network bottlenecks?. URl= https://softwareengineering.stackexchange.com/questions/275734/how-do-microservice-system-architectures-avoid-network-bottlenecks

[13] NGINX. 2020. What is NGINX?. URl= https://www.nginx.com/resources/glossary/nginx/

[14] Pratama, Rizal Yogi.2018 Apa sih Microservice Itu?. URl= https://medium.com/codelabs-unikom/microservices-apaan-tuh-b9f5d56e8848