Kumpulan jurnaL Ilmu Komputer (KLIK)
Volume 06, No.02 Juni 2019
ISSN: 2406-7857

PENGARUH IMPLEMENTASI LOAD BALANCING DAN TUNING WEB SERVER PADA RESPONSE TIME RASPBERRY PI

Dodon Turianto Nugrahadi[1], Rudy Herteno[2], Muhammad Anshari[3]
[1,2,3] Prodi Ilmu Komputer FMIPA ULM
Jl. A. Yani Km 36 Banjarbaru, Kalimantan Selatan
Email: Dodonturianto@ulm.ac.id

Abstract

The rapid development of technology, the increase in web-based systems and development of microcontroller device, have an impact on the ability of web servers to respond in serving client requests. This study aims to analyze load balancing methods algoritma round robin and tuning that significant influence for the response time and the number of clients who are able to be handled in serving clients on the web server with microcontroller device. From this study with Stresstool testing the response time was 2064, 2331,4 and 1869,2ms for not using load balancing and 2270, 2306,2 and 2202ms with load balancing from 700 requests served by web servers. It can be concluded that web server response times that use load balancing are smaller than web servers without load balancing. Furthermore, using tuning with the response time obtained at 3103.4ms from 1100 requests. So, with tuning can reduce response time and increase the number of requests. With level significant level calculatio, have it khown that tuning configuration give significant effect for the response time and the number of clients in microcontroller.

Keywords: *Web server, Raspberry, Load balancing, Response time, Stresstool.*

Abstrak

Perkembangan implementasi teknologi yang pesat, seiring dengan perkembangannya sistem berbasis web dan perangkat mikrokontroler, berdampak pada kemampuan web server dalam memberikan tanggap untuk melayani permintaan klien. Penelitian ini bertujuan untuk menganalisis metode load balance algoritma round robin dan tuning yang berpengaruh terhadap waktu tanggap dan banyaknya jumlah klien yang mampu ditangani dalam melayani klien pada web server dengan mikrokontroler. Dari penelitian ini dengan pengujian Stresstool didapatkan waktu tanggap sebesar 2331,4, 2064, 1869,2*ms untuk tanpa load balancing dan* 2270, 2306,2 *dan* 2202*ms dengan load balancing dari 600 permintaan yang dilayani web server. Dapat disimpulkan bahwa waktu tanggap web server yang menggunakan load balancing lebih kecil dibandingkan web server yang tidak menggunakan load balancing. Selanjutnya menggunakan tuning dengan waktu tanggap sebesar 3103,4ms dari 1100 permintaan. Jadi, tuning dapat mempersingkat waktu tanggap dan meningkatkan jumlah permintaan yang dilayani web server. Selanjutnya dengan penghitungan tingkat pengaruh, bahwa diketahui konfigurasi load balancing algoritma round robin dan tuning memberikan pengaruh secara signifikan terhadap waktu tanggap dan jumlah permintaan pada mikrokontroler.*

***Kata kunci** : Web server, Raspberry, Load balancing, Waktu tanggap, Stresstool, Jmeter, Klien*

1. PENDAHULUAN

Perkembangan teknologi yang pesat, seiring dengan meningkatnya perkembangan sistem berbasis *web,* menurut *netcraft.com* pada bulan maret 2016, tercatat ada sebanyak 171 juta situs yang aktif, penggunaan *web server* berbasis Nginx tercatat mengalami kenaikan sebanyak 669.000 situs dan mengalami penambahan sebanyak 15.000 komputer yang menjalankan *web server* ini*, web server* berbasis *apache* juga mengalami kenaikan sebanyak 447 komputer.

Penggunaan aplikasi *web server* cenderung dijalankan pada *hardware* yang berspesifikasi tinggi, kecenderungan ini dikarenakan adanya ekspektasi permintaan yang tinggi terhadap sebuah *web server* sehingga memerlukan *hardware* yang besar agar melayani permintaan dengan baik. Pada kenyataan dilapangan penggunaan aplikasi *web server* tidak akan menangani segitu banyak permintaan, sehingga tidak bakal memanfaatkan sepenuhnya potensi *hardware* yang dimilikinya. *Hardware* dengan spesifikasi yang tinggi biasa ditempatkan pada ruang *server* khusus.

Salah satu alternatif untuk menanggulangi kebutuhan akan spesifikasi *hardware* yang tinggi adalah dengan memanfaatkan mikrokontroler raspberry pi*.* Raspberry Pi merupakan komputer mini yang memiliki ukuran kecil yaitu sebesar ukuran kartu *ATM* tetapi mampu menjalankan tugas yang sama dengan komputer *PC.* Karena raspberry pi merupakan sebuah komputer mini maka dimungkinkan menjalankan aplikasi *web server.*

Seiring dengan meningkatnya permintaan terhadap akses sebuah *web server*, Pada penelitian Didik Aribowo (2013) menyebutkan, *single server* yang selalu menerima permintaan dari banyak pengguna, secara perlahan akan terjadi *overload* dan *crash* sehingga berdampak pada permintaan yang tidak dapat dilayani oleh *single server* [2].

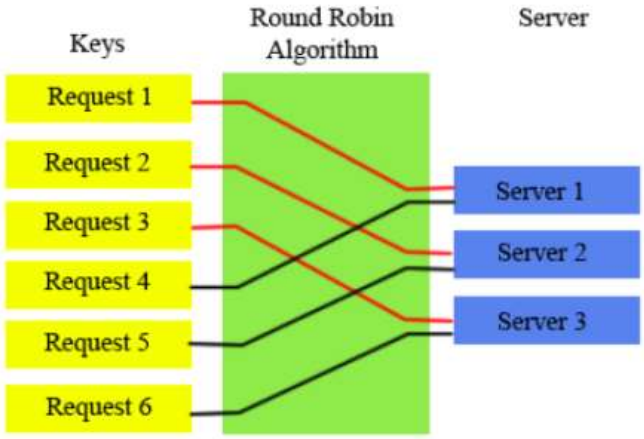
Salah satu solusinya, menurut Taufik Febrianto (2014), teknik *load balancing* merupakan salah satu teknik yang digunakan untuk meningkatkan kinerja dan tingkat ketersediaan *server*, yaitu dengan membagi *request* yang datang ke beberapa *server* sekaligus, sehingga beban yang ditanggung oleh masing-masing *server* lebih ringan [3]. Selain itu menurut handoko (2017), rata-rata waktu respon minimal pada arsitektur yang tidak menggunakan *load balancing* lebih besar dibandingkan dengan arsitektur yang menggunakan *load balancing* [4]. Teknologi *Load Balancing* memiliki beberapa algoritma dalam mendistribusikan beban salah satunya adalah algoritma *round robin.* Algoritma *Round robin* melakukan pembagian beban secara bergiliran dan berurutan dari satu *server* ke *server* yang lain sehingga membentuk putaran.

Pada pendapat lain, menurut Xue Liu dkk (2003) *Tuning* merupakan salah satu cara untuk meningkatkan performansi sebuah *web server,* dengan cara mengoptimalkan pengaturan parameter konfigurasi dengan benar dapat meningkatkan kinerja, terutama pada beban kerja [5].

Implementasi *load Balancing* dan *tuning* mempengaruhi performansi jumlah waktu tanggap dan jumlah permintaan yang dapat dilayani sebuah web server, karena *load balancing* membagi setiap permintaan dan meneruskan ke *web server,* serta *tuning* dapat meningkatkan performansi *web server.*

2. METODE PENELITIAN

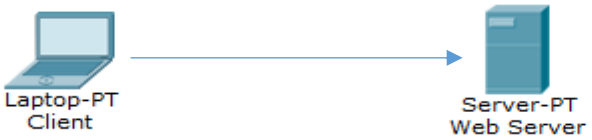
Algoritma *Round robin* merupakan algoritma yang digunakan untuk metode *load balancing* dengan membagi beban ke *server – server* lain sehingga membentuk putaran. Algoritma *round robin* digunakan pada *server* untuk menyeimbangkan beban pada beberapa *server*. Penjadwalan algoritma *round robin* memberikan tugas-tugas ke semua bagian dalam bentuk matematika. *Round robin* memilih *server* dengan i = (i + 1) mod n setiap kali, dimana n adalah jumlah *server* [1].



Gambar 1. Algroritma *round robin.*

2.1 Perancangan Arsitektur

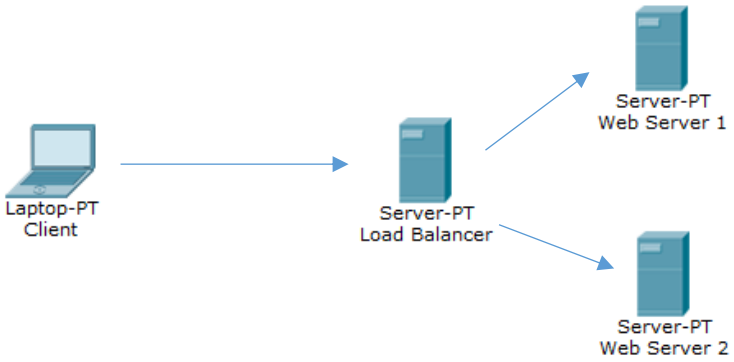
Perancangan arsitektur tanpa menggunakan *load balancing* hanya mengunakan satu buah raspberry pi sebagai *web server*.



Gambar 1. Arsitektur *web server* tanpa menggunakan *load balancing*

Pada gambar 1, arsitektur dibuat dengan menggunakan satu unit raspberry pi yang berfungsi sebagai *web server* dan satu unit *laptop* yang berfungsi sebagai klienyang saling terhubung mengunakan media kabel UTP dengan konektor RJ 45. Arsitektur ini diimplementasikan pada skenario 1, 2 dan 3.

Perancangan arsitektur menggunakan load balancing mengunakan tiga buah raspberry pi, *satu* raspberry pi berfungsi sebagai *balancer* dan dua raspberry pi berfungsi sebagai *web server*.



Gambar 2. Arsitektur *web server* menggunakan *load balancing* dengan algoritma *Round robin*

Pada gambar 2, arsitektur dibuat dengan 1 unit raspberry pi sebagai *load balancing* dan dua unit raspberry pi sebagai *web server* dan satu unit *laptop* sebagai klien yang saling terhubung menggunakan kabel UTP, konektor RJ45 dan *Switch.* Penggunaan kabel UTP, konektor RJ45 dan *switch* berpengaruh pada proses komunikasi data, akan tetapi tidak diperhitungkan dalam penelitian ini. Arsitektur ini diimplementasikan pada skenario 4, 5 dan 6.

Pada arsitektur *web server* yang tidak menggunakan load balancing, raspberry pi menggunakan sistem operasi *Raspbian Stretch Lite* dan menggunakan *web server* Nginx. Untuk *laptop* klien menggunakan sistem operasi windows 7 dari Microsoft. Setelah setiap *raspberry pi* selesai di instalasi, maka tahap selanjutnya adalah melakukan konfigurasi *IP*. Untuk *Web server* yang digunakan adalah 192.168.100.51 dengan *netmask* 255.255.255.0 dan *gateway* 192.168.100.1. Dan untuk *laptop (PC)* klien *IP* yang digunakan adalah 192.168.100.25 dengan netmask 255.255.255.0 dan gateway 192.168.100.1.

Pada arsitektur *web server* yang menggunakan *load balancing* dengan algoritma *round robin*. W*eb server* yang diimplementasikan sebagai *load balancer* tersebut menggunakan *Raspbian Stretch Lite* dan *Nginx*. Dua *raspberry pi* berfungsi sebagai *web server* dan satu *raspberry pi* berfungsi sebagai *load balancer*, setelah diinstalasi, tahap selanjutnya adalah konfigurasi *IP*. Untuk raspberry pi pertama diberikan IP 192.168.100.51 *netmask* 255.255.255.0 dan *gateway* 192.168.100.1. Raspberry pi kedua diberikan *IP* 192.168.100.52 *netmask* 255.255.255.0 dan *gateway* 192.168.100.1. Untuk raspberry pi ketiga diberikan *IP* 192.168.100.53 netmask 255.255.255.0 dan *gateway* 192.168.100.1 sebagai *load balancer*.

2.2 Skenario

Pada tuning dilakukan berbagai skenario, tujuan melakukan skenario adalah melakukan pengujian arsitektur tanpa *load balancing* dan arsitektur menggunakan *load balancing*. Dalam Penelitian ini dilakukan sebanyak 6 skenario. Dalam skenario dilakukan perubahan konfigurasi *tuning* yang telah disesuaikan.

Pada **skenario 1** merupakan pengujian dengan arsitektur tanpa *load balancing* dan tidak dilakukan perubahan pada parameter *worker_processor*, *worker_connection*, *keepalive request dan timeout* serta *compression*. **Skenario 2** dilakukan pada arsitektur tanpa *load balancing*, dilakukan perubahan pada

parameter *worker_processor* menjadi 4, *worker_connection* menjadi 1024*, keepalive request* dan *timeout* menjadi 320 dan 300s serta *compression* diaktifkan. **Skenario 3** dilakukan pada aristektur tanpa *load balancing,*dilakukan perubahan pada parameter *worker_processor* menjadi 2, *worker_connection* menjadi 768, *keepalive request*dan *timeout* menjadi 320 dan 300s. **Skenario 4** dilakukan pada arsitektur menggunakan *load balancing* dengan tidak ada perubahan pada parameter *worker_processor*, *worker_connection*, *keepalive request dan timeout* serta *compression*. **Skenario 5** dilakukan pada arsitektur menggunakan *load balancing*, dilakukan perubahan pada parameter *worker_processor* menjadi 4, *worker_connection* menjadi 1024*, keepalive request* dan *timeout* menjadi 320 dan 300s serta *compression* diaktifkan. **Skenario 6** dilakukan pada aristektur menggunakan *load balancing,*dilakukan perubahan pada parameter *worker_processor* menjadi 2, *worker_connection* menjadi 768, *keepalive request*dan *timeout* menjadi 320 dan 300s.

Konfirgurasi ini berdasarkan buku pedoman dari *web server* nginx beberapa konfigurasi tuning yang berpengaruh terhadap peningkatan performa *web server* nginx. Berdasarkan beberapa percobaan awal dari penelitian ini, maka pada penelitian ini menggunakan parameter *worker_processor, worker_connection, keepalive request, timeout* dan *compression* sebagai parameter tuning. Parameter tersebut menggunakan nilai parameter maksimal dan parameter setengah dari maksimal. Untu memudahkan dalam membaca perubahan konfigurasi pada setiap skenario, perubahan konfigurasi disajikan dalam tabel, Berikut tabel perubahan konfigurasi yang dilakukan dalam penelitian ini :

Tabel 1. Konfigurasi skenario

	Tanpa Load Balancing			Menggunakan Load Balancing		
	Skenario 1	Skenario 2	Skenario 3	Skenario 4	Skenario 5	Skenario 6
Worker Processor	Auto	4	2	Auto	4	2
Worker Connection	768	1024	768	768	1024	768
Keepalive Request	-	320	320	-	320	320
Keepalive Timeout	65	300	300	65	300	300
Compression	Off	On	On	Off	On	On

2.3 Pengumpulan Data

Pengumpulan data dilakukan dengan cara melakukan *http request* pada setiap skenario arsitektur *web server* dengan mengunakan *software Jmeter*. Perintah yang digunakan untuk melakukan *http request* yaitu dengan cara memasukkan *Thread Group , number of thread, ramp-up period,* dan *loop count* selanjutnya mengisi *Jmeter element* yaitu *config element* sebagai *http request* dan mengisi alamat *web server,* yang terakhir adalah menambahkan *listener* yaitu *summary report.*

Proses *toolsjmeter* dengan mengirimkan permintaan kepada *web server* dimulai dari 100 dan meningkat dengan kelipatan 100 sampai ditemukan nilai permintaan yang menimbulkan kesalahan (*error-rate*> 0). Nilai permintaan *error* inilah yang diambil sebagai nilai permintaan maksimum.

3. HASIL DAN PEMBAHASAN

3.1 Hasil

Tabel 2. Hasil pengumpulan data arsitektur *web server* tanpa menggunakan *load balancing* algoritma *round robin* tanpa tuning skenario 1

Permintaan	*Average*	*Error*
100	21,4	0%
200	72	0%
300	634	0%
400	1184,6	0%
500	1745,6	0%
600	2331,4	0%
700	2788,6	0%
800	3121	0%
900	2954,4	11.47%

Pada tabel 2, didapatkan hasil pengumpulan data pada arsitektur *web server* tanpa menggunakan *load balancing* dan tanpa tuning dengan data yang bervariasi pada setiap klien. Pada percobaan ini didapatkan maksimum permintaan yang dapat dilayani *web server* adalah sebesar 800 permintaan, dikarenakan nilai error > 0% pada permintaan 900 yaitu sebesar 11,47%.

Tabel 3. Hasil pengumpulan data arsitektur *web server* tanpa menggunakan *load balancing* algoritma *round robin* dengan tuning skenario 2

Permintaan	*Average*	*Error*
100	18,4	0%
200	110	0%
300	623,2	0%

Permintaan	*Average*	*Error*
400	1106,8	0%
500	1592,2	0%
600	2064	0%
700	2414,2	0%
800	2119,2	11,524 %

Pada tabel 3, didapatkan hasil pengumpulan data pada arsitektur *web server* tanpa *load balancing* dan tuning skenario 2, dengan variasi data pada setiap klien. Pada percobaan ini didapatkan maksimum permintaan yang dapat dilayani *web server* adalah sebesar 700 permintaan, dikarenakan nilai error > 0% pada permintaan 800 yaitu sebesar 11,524%.

Tabel 4. Hasil pengumpulan data arsitektur *web server* tanpa menggunakan *load balancing* algoritma *round robin* dengan tuning skenario 3

Permintaan	*Average*	*Error*
100	21,4	0%
200	61,8	0%
300	436,2	0%
400	850,8	0%
500	1352	0%
600	1869,2	0%
700	2255,6	0%
800	2529,6	0%
900	2725,6	0%
1000	2967,2	0%
1100	3103,4	0%
1200	2905,2	10,5%

Pada tabel 4, didapatkan hasil pengumpulan data pada arsitektur *web server* tanpa menggunakan *load balancing* dan tuning skenario 3, dengan data yang bervariasi pada setiap klien. Pada percobaan ini didapatkan maksimum permintaan yang dapat dilayani *web server* adalah sebesar 1100 permintaan, dikarenakan nilai error > 0% pada permintaan 1200 yaitu sebesar 10,5%.

Tabel 5. Hasil pengumpulan data arsitektur *web server* menggunakan *load balancing* algoritma *round robin* tanpa tuning skenario 4

Permintaan	*Average*	*Error*
100	30,4	0%
200	270,2	0%
300	709	0%
400	1238	0%
500	1764,6	0%
600	2270	0%
700	2544,4	0%
800	2959	4,15%

Pada tabel 5, didapatkan hasil pengumpulan data pada arsitektur *web server*dengan menggunakan *load balancing* dan tuning skenario 4, dengan variasi data pada setiap klien. Pada percobaan ini didapatkan maksimum permintaan yang dapat dilayani *web server* adalah sebesar 700 permintaan, dikarenakan nilai error > 0% pada permintaan 800 yaitu sebesar 4,15%.

Tabel 6. Hasil pengumpulan data arsitektur *web server* menggunakan *load balancing* algoritma *round robin* dengan tuning skenario 5

Permintaan	*Average*	*Error*
100	30,6	0%
200	229,2	0%
300	750,2	0%
400	1159,6	0%
500	1424,2	0%
600	2306,2	0%
700	2376,2	7,9%

Pada tabel 6, didapatkan hasil pengumpulan data pada arsitektur *web server* dengan menggunakan *load balancing* dan tuning skenario 5, dengan variasi data pada setiap klien. Pada percobaan ini didapatkan maksimum permintaan yang dapat dilayani *web server* adalah sebesar 600 permintaan, dikarenakan nilai error > 0% pada permintaan 700 yaitu sebesar 7,9%.

Tabel 7. Hasil pengumpulan data arsitektur *web server* menggunakan *load balancing* algoritma *round robin* dengan tuning skenario 6

Permintaan	*Average*	*Error*
100	50,4	0%
200	237,2	0%
300	668,4	0%
400	1244,8	0%
500	1556,4	0%
600	2202	0%
700	1994,2	11,3%

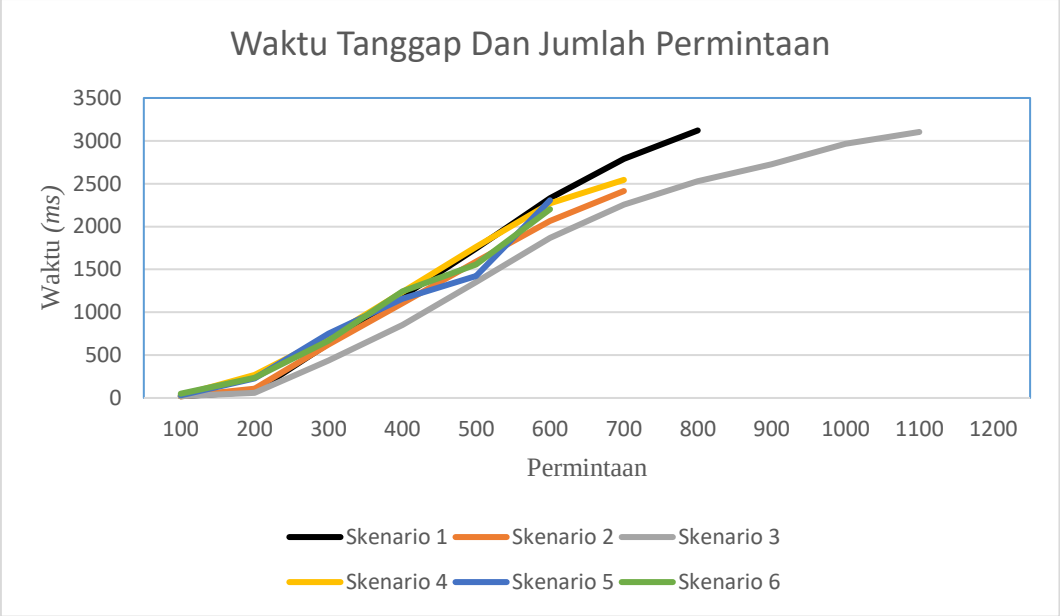
Pada tabel 7, didapatkan hasil pengumpulan data pada arsitektur *web server* dengan menggunakan *load balancing* dan tuning skenario 6, dengan variasi data pada setiap klien. Pada percobaan ini didapatkan maksimum permintaan yang dapat dilayani *web server* adalah sebesar 600 permintaan, dikarenakan nilai error > 0% pada permintaan 700 yaitu sebesar 11,3%.

Tabel 8. Hasil Waktu Tanggap dan Jumlah Permintaan

Permintaan	Tanpa *Load Balancing*			Menggunakan *Load Balancing*		
	Skenario 1	Skenario 2	Skenario 3	Skenario 4	Skenario 5	Skenario 6
100	21,4	18,4	21,4	30,4	30,6	50,4
200	72	110	61,8	270,2	229,2	237,2
300	634	623,2	436,2	709	750,2	668,4
400	1184,6	1106,8	850,8	1238	1159,6	1244,8
500	1745,6	1592,2	1352	1764,6	1424,2	1556,4
600	2331,4	2064	1869,2	2270	2306,2	2202
700	2788,6	2414,2	2255,6	2544,4	-	-
800	3121,4	-	2529,6	-	-	-
900	-	-	2725,6	-	-	-
1000	-	-	2967,2	-	-	-

Permintaan	Tanpa *Load Balancing*			Menggunakan *Load Balancing*		
	Skenario 1	Skenario 2	Skenario 3	Skenario 4	Skenario 5	Skenario 6
1100	-	-	3103,4	-	-	-

Pada tabel 8, didapatkan hasil pengumpulan data pada arsitektur *web server* tanpa dan dengan menggunakan *load balancing* dengan algoritma *round robin* dengan variasi data pada setiap *klien*. Kemudian berdasarkan grafik untuk mempermudah melihat data yang didapatkan.



Gambar 3. Grafik hasil pengumpulan data waktu tanggap dan jumlah permintaan

3.2 Pembahasan

3.2.1. Waktu Tanggap

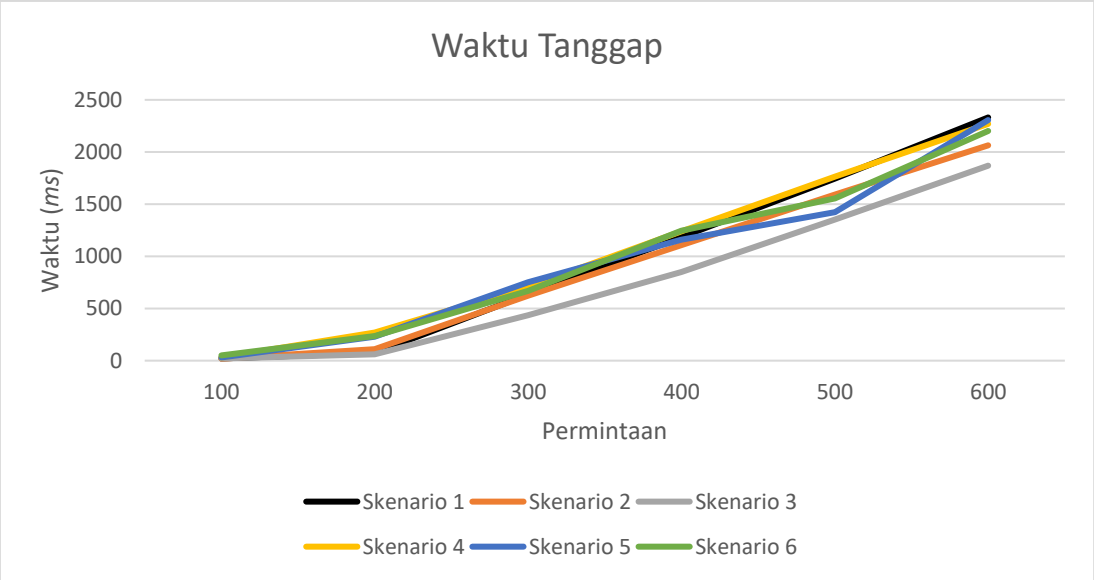
Analisis waktu tanggap bertujuan untuk mengetahui pengaruh penggunaan *load balancing* algoritma *round robin* dan *tuning*. Pertama arsitektur *web server* yang tidak menggunakan *load balancing* dibuat sesuai dengan gambar 1. Kemudian dibuat arsitektur yang ke dua sesuai dengan gambar 2 yaitu arsitektur *web server* yang menggunakan *load balancing* dengan algoritma *round robin*. Setelah kedua arsitektur dibuat, dilakukan pengambilan data dengan cara melakukan *load test* dengan tools penguji. Hasil yang didapat dari tahap pengambilan data dapat dilihat pada tabel 2.

Waktu tanggap adalah waktu tanggap yang diterima oleh klien ketika mengakses *sebuah web server.* Pada tabel 2, diambil data sebanyak 6 variasi jumlah permintaan dari hasil waktu tanggap yang dihasilkan dari *klien* yang mengakses *web server*.

Tabel 9. Hasil Waktu Tanggap

Permintaan	Tanpa *Load Balancing*			Menggunakan *Load Balancing*		
	Skenario 1	Skenario 2	Skenario 3	Skenario 4	Skenario 5	Skenario 6
100	21,4	18,4	21,4	30,4	30,6	50,4
200	72	110	61,8	270,2	229,2	237,2
300	634	623,2	436,2	709	750,2	668,4
400	1184,6	1106,8	850,8	1238	1159,6	1244,8
500	1745,6	1592,2	1352	1764,6	1424,2	1556,4
600	2331,4	2064	1869,2	2270	2306,2	2202

Untuk mempermudah pembahasan, maka data disajikan dalam bentuk grafik. Perbandingan waktu tanggap dapat terlihat pada gambar 4.



Gambar 4. Grafik perbandingan waktu tanggap

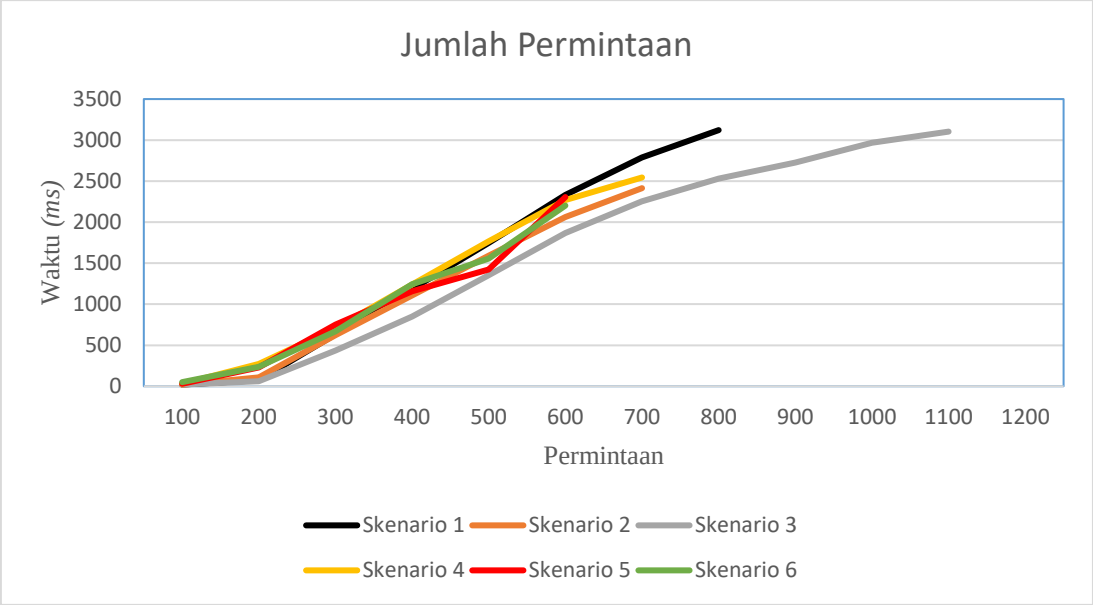
Pada gambar 4, terlihat bahwa pada tiap variasi jumlah permintaan, waktu tanggap pada arsitektur *web server* yang menggunakan *load balancing* algoritma *round robin* dan *tuning* lebih besar dibandingkan dengan arsitektur *web server* yang tanpa *load balancing* algoritma *round robin* dan arsitektur *web server* tanpa *load balancing* dengan *tuning.*

Adapun analisis perhitungan Anova untuk waktu tanggap rata–rata menggunakan input range, diisi dengan tabel 9 dengan nilai α = 0,05 digunakan sebagai taraf signifikansi. Perhitungan Anova mengacu pada tinjauan pustaka, sehingga hasil perhitungan menggunakan Anova untuk hasil waktu tanggap. Diketahui nilai P-Value=0,000142, dimana skenario < 0,05 dengan demikian dapat dikatakan setiap

skenario implementasi *load balancing* algoritma *round robin* dan *tuning* berpengaruh terhadap waktu tanggap.

3.2.2. Permintaan

Jumlah Permintaan dikatakan maksimum jika terjadi nilai Error > 0. Pada tabel 2 dapat dilihat jumlah permintaan maksimum yang dapat dilayani *web server* sebesar 800 klien pada skenario 1, 700 klien pada skenario 2, 1100 pada skenario 3, 700 pada skenario 4, 600 pada skenario 5 dan 600 pada skenario 6. Pada saat pengujian terjadi kenaikan *cpu* yang mengakibatkan terjadinya *error.* Terlihat pada skenario 3 terdapat kenaikan sebesar 500 klien atau sebesar 83 %, dengan ini dapat dikatakan skenario atau *tuning* berpengaruh terhadap jumlah permintaan yang dapat dilayani *web server.* Untuk mempermudah pembahasan, maka data disajikan dalam bentuk grafik. Perbandingan jumlah klien dapat terlihat pada gambar 4.



Gambar 5. Grafik jumlah permintaan yang dapat dilayani *web server*

4. SIMPULAN

Berdasarkan hasil penelitian maka diperoleh kesimpulan dari hasil pembahasan sebagai berikut :

a. Waktu tanggap pada arsitektur tanpa *load balancing* sebesar 2331,4*ms* pada skenario 1, 2414,2 *ms* pada skenario 2, 2255,6 *ms* dan pada skenario 3. Sedangkan Waktu tanggap pada arsitektur menggunakan *load balancing* 2544,4 *ms* pada skenario 4, 2376,2 *ms* pada skenario 5 dan 2202 *ms* pada skenario 6
b. Jumlah permintaan yang dapat ditangani *web server* sebesar 800 pada skenario 1, 700 pada skenario 2, 1100 pada skenario 3, 700 pada skenario 4, 600 pada skenario 5 dan 600 pada skenario 6.
c. Implementasi *load balancing* algoritma *round robin* dan *tuning* berpengaruh pada waktu tanggap *web server* nginx dengan P-Value=0,000142, dan implementasi tuning pada w*eb server* dapat meningkatkan permintaan hingga 83 %.

DAFTAR PUSTAKA

[1] Andarrachmi Annisa. 2014. *"*Implementasi*Load Balancing* Dan*Virtual Machine* dengan Algoritma*Round Robin* Pada Sistem Informasi Penerimaan Pegawai Bppt*". Badan* Pengkajian dan Penerapan Teknologi *(BPPT).*

[2] Aribowo Didik dan Achmad Affandi*.* 2013*. "Optimalisasi Cluster Server Lms Dan Iptv Dengan Variasi Algoritma Penjadwalan". Telekomunikasi Multimedia Teknik Elektro ITS, Surabaya, Seminar Nasional Teknologi Informasi dan Multimedia 2013 STMIK AMIKOM Yogyakarta, 19 Januari.*

[3] Febrianto, Taufik. 2014. "Analisa *Load Balancing* Dengan Metode *LVS Direct Routing* Menggunakan Algoritma *Round Robin* dan *Least Connection*". Universitas Sebelas Maret: Surabaya.

[4] Handoko, Dodon T. N., Ichsan Ridani. 2017. "*Implementasi Dan Pengujian Performansi Load Balancing Dengan Algoritma Leastconn Pada Database Server*". Jurnal Elektronik Nasional Teknologi dan Ilmu Komputer (JENTIK)

[5] Liu Xue, dkk. 2003. "*Online Respone Time Optimization Of Apache Web Server*". Berlin: K. Jeffay, I. Stoica, and K. Wehrle (Eds.): IWQoS