jurnalteknik.unisla.ac.id/index.php/informatika ISSN: 2503-071X
JOUTICA Volume 7 No.2 Tahun 2022 E-ISSN: 2621-511X

PENENTUAN JUMLAH SERVER MENGGUNAKAN *SERVER RESPONSE TIME* UNTUK PENERAPAN *LOAD BALANCER* PADA APLIKASI PENDAFTARAN SISWA BARU

STUDI KASUS: DINAS PENDIDIKAN KABUPATEN XYZ

Muhammad Khoirul Hasin[1], Afif Zuhri[2]
Teknik Otomasi, Teknik Kelistrikan Kapal, Politeknik Perkapalan Negeri Surabaya
Jl. Teknik Kimia, Kampus ITS Sukolilo
E-mail: khoirulhasin@ppns.ac.id

ABSTRAKS
Salah satu kabupaten di Jawa Timur (Jatim) akan menyelenggarakan Penerimaan Peserta Didik Baru (PPDB) secara daring, perkiraan ada sekitar 17 ribu pendaftar yang akan melanjutkan jenjang pendidikan dari Sekolah Dasar (SD/MI) ke Sekolah Menengah Pertama (SMP) Negeri. Semua pihak berharap agar sistem tersebut berjalan dengan lancar. Oleh karena itu, penelitian ini mensimulasikan kebutuhan server terhadap jumlah permintaan pengguna secara simultan pada sistem dalam kondisi paling maksimal dengan menggunakan loading.io yang nantinya diterapkan untuk infrastruktur load balancer. Dengan menggunakan loading.io akan ditemukan SRT (server response time), dimana diperoleh nilai 464 ms, dari nilai tersebut untuk mencapai Time to First Byte (TTFB) cukup, yaitu antara 200 ms – 1 s maka membutuhkan 7 server untuk API web service. Untuk pengujian, penulis menggunakan tiga sample server API web service yang diterapkan dengan Nginx load balancer. Dari hasil pengujian diperoleh selama sepuluh kali percobaan ketiga server tersebut sudah berhasil terakses dan ketika salah satu server dimatikan aplikasi PPDB masih berjalan dengan lancar dengan sepuluh kali percobaan.

Kata Kunci: PPDB, load balancer, server time response, time to first byte

ABSTRACT
One of the regencies in East Java (East Java) will hold an online New Student Admission (PPDB), it is estimated that there are around 17 thousand registrants who will continue their education from Elementary School (SD/MI) to State Junior High School (SMP). Everyone hopes that the system will run smoothly. Therefore, this study simulates server requirements for the number of simultaneous user requests on the system in the maximum condition using loading.io which will later be applied to the load balancer infrastructure. By using loading.io you will find SRT (server response time), where the value is 464 ms, from that value to reach Time to First Byte (TTFB) is sufficient, which is between 200 ms - 1 s, it requires 7 servers for API web service. For testing, the author uses three sample server API web services that are implemented with the Nginx load balancer. From the test results, it was obtained that during ten trials the three servers had been successfully accessed and when one server was turned off the PPDB application was still running smoothly with ten attempts.

Keywords: PPDB, load balancer, server time response, time to first byte

1. PENDAHULUAN

1.1 Latar Belakang

Salah satu kabupaten di Jawa Timur (Jatim) akan menyelenggarakan Penerimaan Peserta Didik Baru (PPDB) secara daring, perkiraan ada sekitar 17 ribu pendaftar yang akan melanjutkan jenjang pendidikan dari Sekolah Dasar (SD/MI) ke Sekolah Menengah Pertama (SMP) Negeri. Semua pihak berharap agar sistem tersebut berjalan dengan lancar.

Oleh karenanya, yang harus menjadi perhatian adalah kehandalan dari sebuah sistem aplikasi pendaftaran untuk bisa menangani ratusan permintaan proses sehingga tidak akan terjadi kegagalan sistem yang disebabkan melonjaknya jumlah pengguna. Salah satunya solusinya adalah menambahkan *resource* (*processor* dan memori) server namun cara tersebut sangatlah mahal karena semakin tinggi spesifikasi server maka harganya jauh semakin tinggi jika dibandingkan menyewa beberapa server yang jumlahnya sama dengan satu server. Selain itu, penggunaan satu server memiliki resiko tinggi terjadi kegagalan karena tidak ada server lain yang bisa menggantikannya jika terjadi kegagalan. Oleh karenanya, solusinya adalah melakukan distribusi beban ke beberapa server kecil.

Selain itu, cara ini juga bisa menghindari kegagalan sistem, jika terjadi kegagalan dari satu server maka server lain bisa menggantikannya. Salah satu teknologi yang bisa menangani hal tersebut adalah *load balancing*. Selain beberapa keunggulan yang sudah disebutkan diatas, *load balancing* juga memberikan fleksibilitas menyesuaikan jumlah server sesuai dengan beban sistem (HTTP Load Balancing, 2022). Penelitian ini mensimulasikan

ISSN: 2503-071X
E-ISSN: 2621-511X

kebutuhan server terhadap jumlah permintaan pengguna secara simultan pada sistem dalam kondisi paling maksimal dengan menggunakan *loading.io*.

1.2 Referensi

1.2.1 Web server

Web server merupakan perangkat lunak yang dipasang di server yang menerima permintaan HTTP/HTTPS dari klien (contoh: *browser*). Browser membuat permintaan melalaui *web page* menggunakan HTTP/HTTPS kemudian web server meresponnya dan mengirimnya ke klien, jika berhasil, respon tersebut berupa konten (gambar, video, text, html, atau bentuk lainnya), sebaliknya jika gagal, respon tersebut berupa pesan kesalahan (Ramadani, 2018).

Ada beberapa jenis web server yang sudah beredar, diantaranya: Apache, Nginx, IIS, dan Lighttpd. Pada penelitian ini, digunakan Nginx yang dikenal memiliki keunggulan dalam menangani berbagai macam permintaan pengguna dibandingkan dengan web server lainnya, seperti permintaan dengan tingkat kepadatan lalu lintas yang sangat padat. Nginx memang lebih unggul dari segi kualitas, kecepatan, dan dalam hal performanya.

1.2.2 Nginx

Nginx adalah salah satu jenis web server dengan performa yang andal. Nginx ini cukup popular, menurut W3Tect, ada sekitar 34% website menggunakan Nginx sebagai web servernya. Web server yang dikembangkan oleh Igor Sysoev ini, software engineer Rusia, menawarkan penggunaan dalam konkurensi tinggi yang dapat menangani ribuan koneksi secara bersamaan dengan penggunaan memori yang rendah.

Sampai saat ini Nginx adalah satu-satunya web server gratis yang menerapkan proses asynchronous, menjalankan beberapa proses dalam waktu yang bersamaan tanpa harus menunggu proses lain selesai (HTTP Load Balancing, 2022). Salah satu fitur yang ada di dalam Nginx adalah *load balancing*, yang mana fitur tersebut akan diterapkan dalam penelitian ini.

1.2.3 *Load Balancing*

Load balancing adalah proses distribusi beban traffic ke beberapa server, yang bertujuan untuk memastikan tidak ada server yang terganggu karena banyaknya beban permintaan. Akibat dari server kelebihan beban adalah server tersebut akan merespon sangat lambat bahkan tidak terhubung sama sekali (HTTP Load Balancing, 2022). Sedangkan, pelaku/alat load balancing disebut *load balancer*.

Secara garis besar, cara kerja *load balancer* sebagai berikut: klien/*browser* malakukan permintaan ke aplikasi, kemudian *load balancer* menerima dan merespon dengan cara mendistribusikan beban tersebut ke beberapa server, jika salah satu dari server mati/bermasalah, load balancer akan mengalihkan beban tersebut ke server lain.

Load balancer bisa berupa *software* maupun *hardware*, yang mana pada penelitian ini menggunakan jenis software, yaitu *load balancer* yang ada di Nginx. Di dalam Nginx, ada beberapa metode load balancing, gratis dan berbayar, penelitian ini memilih yang gratis, yaitu *least connections*, yang mana metode ini memperbaiki dari metode defaultnya, yaitu *round robin*.

1.2.4. Metode *least connection*

Metode ini memperbaiki kekurangan metode *round robin* dalam membaca beban tiap server. Metode Least Connection menjaga distribusi traffic yang merata di semua server yang tersedia. Jika sebuah server memiliki beban koneksi yang besar, permintaan data akan didistribusikan ke server yang lebih luang. Hal ini dilakukan untuk menghindari overload pada server karena besarnya traffic yang diterima (Ibrahim Mahmood Ibrahim, Siddeeq Y. Ameen, Hajar Maseeh Yasin, dkk, 2021).

1.2.5 API *web service*

API singkatan dari Application Programming *Interface*, merupakan sebuah perangkat lunak yang memudahkan programmer dalam mengembangkan aplikasi, seorang programmer tidak perlu tahu detail dari suatu fungsi bekerja, dia hanya perlu menghubungkan API-nya saja. API juga memungkinkan komunikasi suatu aplikasi dengan aplikasi lainnya.

Sedangkan *web service* adalah layanan perangkat lunak yang mendukung interaksi dan komunikasi antar aplikasi di dalam suatu jaringan. Dalam melakukan komunikasi, umumnya *API web service* menggunakan JSON (MALIK, 2021) (Ramadani, 2018).

1.2.6 *Server response time*

Server response time (SRT) merupakan besaran waktu antara klien melakukan permintaan dan server merespon permintaan tersebut. SRT memiliki satuan yang dikenal dengan sebutan Time to First Byte (TTFB). TTFB ini mengukur lama waktu antara klien membuat permintaan dan pertama kali menerima *byte* data. Aturan yang biasanya digunakan dalam menentukan standar TTFB sebagai berikut: 1) sangat bagus: < 100 ms. 2) bagus: 100 – 200 ms. 3) cukup: *200* ms – 1 s. 4) jelek: > 1 s. Target penelitian ini TTFB nya antara bagus dan cukup (Kang, 2021).

2. PEMBAHASAN

2.1 Analisa Kebutuhan

Pembuatan aplikasi PPDB ini menggunakan bahasa pemrograman PHP dan Javascript, yang mana terbagi ke dalam dua bagian yaitu *frontend* dan *backend* (API *web service*). *Frontend* menggunakan *framework Javascript Angular 5*

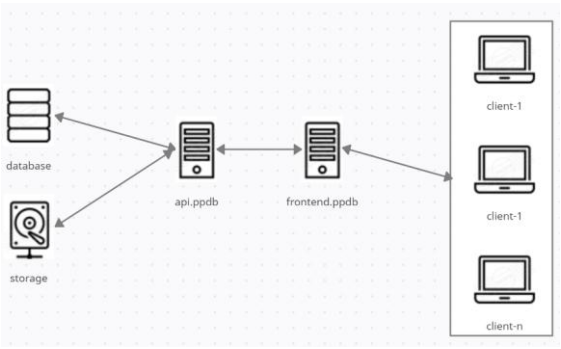
ISSN: 2503-071X
E-ISSN: 2621-511X

sedangkan *backend* menggunakan *framework PHP Codeigniter*. Database menggunakan MySQL 8. Penulis menggunakan Git untuk mempermudah melakukan *versioning*. Infrastrukturnya menggunakan layanan dari DigitalOcean, yaitu beberapa droplet (sebutan untuk VPS) dan *cluster database*, dengan detail sebagai berikut: 1 VPS untuk *frontend*, 1 MySQL database cluster, dan 1 VPS untuk API *web service*.

Perkiraan beban aplikasi ini adalah sekitar 17.225 pendaftar, ada 623 SD dengan jumlah panitia 1-3 orang, ada 42 SMP dengan jumlah panitia 3-5 orang, dan pengguna saat *peak time* sekitar 1000 orang. Aplikasi ini juga dilengkapi dengan fitur perangkingan secara realtime, cetak PDF, perhitungan jarak dan penampakan peta menggunakan API Google Maps, unduh laporan dalam bentuk excel dan unggah file. Target penelitian ini TTFB nya antara bagus dan cukup.

2.2 Perancangan

Gambar 1 menjelaskan, klien/*browser* mengakses web frontend di alamat frontend.ppdb, yang mana frontend tersebut terkoneksi server API web service. Server API *web service* ini nanti jumlahnya tergantung pada beban dan performa yang diinginkan. API terkoneksi dengan *server database* dan *storage*. Semua alamat yang tercantum menerapkan SSL (https) menggunakan Let's Encrypt. Semua layanan infrastruktur ini menggunakan DigitalOcean (Khan, 2021) (Khan, 2021).



Gambar 1. Perancangan infrastruktur layanana Digitals Ocean

2.3 Implementasi

Langkah pertama, melakukan pendaftaran ke DigitalOcean, mempersiapkan 4 *droplet*, 1 *space* (sebutan *storage server* untuk DigitalOcean), dan 1 *database*. 2 *droplet* yang menggunakan sistem operasi Ubuntu ini masing-masing digunakan untuk *frontend* dan API *web service*, yang masing-masing memiliki spesifikasi memory 1 GB, 1 CPU, 25 GB SSD, dan 1000 GB transfer. Semua *droplet* membutuhkan instalasi Nginx dan Let's Encrypt. Untuk *frontend*, perlu dipersiapkan NodeJs 8 untuk menginstal Angular 5. Untuk API *web service*, harus terinstall php7.4 yang dilengkapi dengan *extention* php7.4-fpm, php7.4-cli, php7.4-xml, php7.4-mysql, php7.4-mbstring, dan php7.4-gd yang berguna untuk mendukung fitur aplikasi PPDB.

Jangan lupa memberikan kepada API *web service* untuk bisa mengakses database dengan menambahkan *IP address* masing-masing API *web service* di dalam *Trusted Resources*. Berikan alamat domain pada kedua *droplet* sesuai dengan fungsinnya masing-masing.

Kemudian atur *API web service* tersebut menggunakan Nginx yang ada di dalam /etc/nginx/sites-available/default, dengan konten di Tabel 1. Tabel 1 menjelaskan bahwa SSL pada Nginx ini menggunakan Let's Encrypt dan ada *script* yang tidak dinampakkan yang berada pada snippets/server-param-php7.4.conf, yang mana script tersebut digunakan untuk mengatur aplikasi yang berbasiskan php7.4. Sedangkan untuk *frontend*-nya seperti yang ada pada Tabel 2.

Tabel 1. Pengaturan Nginx untuk server API web service

```
server {
    server_name api.ppdb;
    root     /var/www/api-n.ppdb;
    include snippets/server-param-php7.4.conf;

    listen 443 ssl;
    ssl_certificate /etc/letsencrypt/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;
}

server {
    if ($host = api.ppdb) {
        return 301 https://$host$request_uri;
    }
    server_name api.ppdb;
    listen 80;
                return 404;
}
```

Tabel 2. Pengaturan Nginx untuk server *frontend*

```
server{
    listen 80;
    server_name frontend.ppdb;
    root /var/www/frontend.ppdb;
    index index.html;

    listen 443 ssl;
    ssl_certificate /etc/letsencrypt/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    location / {
        try_files $uri$args $uri$args/ /index.html;
    }
}
```

ISSN: 2503-071X
E-ISSN: 2621-511X

```
        server {
            if ($host = frontend.ppdb) {
                return 301 https://$host$request_uri;
            }
            server_name frontend.ppdb;
            listen 80;
            return 404;
}
```

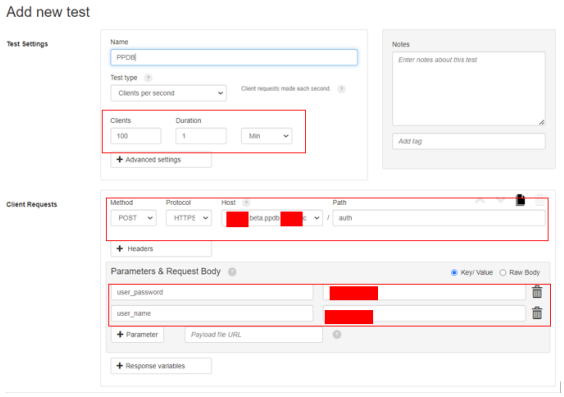
3. HASIL DAN PENGUJIAN

3.1 Pengujian *Server Response Time* (SRT)

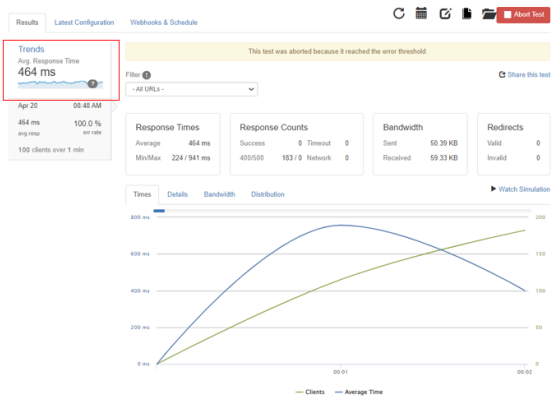
Berikutnya dilakukan uji coba perancangan Gambar 2 dengan menggunakan loader.id (Creating a Test, 2022), yang bertujuan untuk mengetahui *server response time* (SRT). Tentunya dengan spesifikasi server yang sudah disebutkan diatas.

Dari pengujian ini akan diketahui perkiraan jumlah server API *web service* yang akan dipasang di aplikasi PPDB dengan 1000 pengguna secara konkuren dalam setiap detiknnya. Pada Gambar 2 adalah tampilan pengaturan pengujian. Pada bagian *Test Settings* disebutkan ada 100 klien yang melakukan permintaan secara bersamaan dalam satu detik selama semenit. Pada bagian Client Request, penulis mengatur *method*-nya menggunakan POST, *protocol* menggunakan HTTPS, *host* yang diakses adalah alamat frontendnya dengan alamat frontend.ppdb pada *path* berisi *auth* atau halaman login. Untuk bisa login harus memasukkan *request body*, dimana *payload*-nya *user_name* dan *user*_password.

Hasil dari pengujian ini bisa dilihat pada Gambar 3, yaitu diperoleh bahwa rata-rata *response time*-nya adalah 464 ms sehingga bisa diperkirakan ketika 1000 pengguna dengan harapan TFBB- nya antara bagus (100 – 200 ms) dan cukup (200 mis – 1 s) maka bisa dihitung dengan melihat informasi Tabel 3 tersebut maka bisa diperoleh, untuk mencapai TFBB bagus maka diperlukan 31 VPS API *web service* sedangkan untuk TFBB cukup maka dibutuhkan sekitar 7-8 VPS API *web service* (Lihat Tabel 4). Dari sini bisa disimpulkan atas dasar anggaran bahwa keputusan yang paling tepat adalah memilih TFBB cukup dengan 7-8 VPS.



Gambar 2. Pengaturan pengujian SRT pada loader.io



Gambar 3. Hasil uji SRT

Tabel 3. Detail perkiraan beban server

pengguna	1000 orang
per server per 100 pengguna	464 ms
per server per 1000 pengguna	4640 ms
rata-rata bagus	150 ms
rata-rata cukup	600 ms

Tabel 4. Hasil perhitungan jumlah server dengan mempertimbangkan TFBB

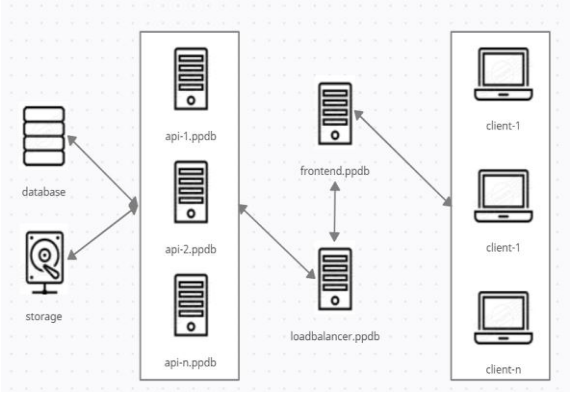
Jumlah VPS	
bagus (100 – 200 ms)	cukup (200 mis – 1 s)
30,93333333	7,733333333

3.2 Penerapan *Load Balancer*

Untuk rancangan *load balancer* bisa dilihat pada Gambar 4. *Frontend* tidak langsung berkomunikasi dengan API web service namun dihubungkan terlebih dahulu dengan *load balancer*, yang berfungsi untuk mendistribusikan beban pada setiap server API *web service*.

Untuk pengaturan *load balancer* Nginx lihat Tabel 5. Pada *attribute upstream* berikan API *web service* yang akan terkoneksi dengan *load balancer*, yang mana ketiganya menerapkan SSL, yaitu menggunakan port 443 dan untuk mengatur distribusi beban, load balancer ini menggunakan *least connection*, atau disebut dengan *least_conn* di Nginx. *Upstream* tersebut menggunakan nama *backend* sehingga pada attribute *proxy_pass* diberikan https://backend. Selain itu, atur alamat *load balancer* pada attribute *server_name*. Pada kasus ini menggunakan alamat *loadbalancer.ppdb.*

ISSN: 2503-071X
E-ISSN: 2621-511X



Gambar 4. Rancangan infrastruktur penerapan *load balancer*

Tabel 5. Pengaturan Nginx untuk *load balancer*

```
upstream backend {
    least_conn;
    server api-1.ppdb:443;
    server api-2.ppdb 443;
    server api-n.ppdb:443;
  }
  server {
    server_name loadbalancer.ppdb;
    location / {
      proxy_pass https://backend;
    }

    listen 443 ssl;
    ssl_certificate /etc/letsencrypt/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

}

  server {
    if ($host = loadbalancer.ppdb) {
        return 301 https://$host$request_uri;
    }
    listen 80;

   server_name loadbalancer.ppdb;
    return 404;
          }
```

3.3 Pengujian *Load Balancer*

Pengujian ini memastikan bahwa *load balancer* yang diterapkan sudah berjalan dengan semestinya. Penulis menggunakan 3 VPS untuk *API web service*-nya, yaitu api-1, api-2, dan api-3. Langkah yang dilakukan pada pengujian ini adalah dengan cara mengakses alamat https://lb.ppdb sebanyak sepuluh kali dan memastikan bahwa seluruh API *web service* sudah berkontribusi bergantian diakses oleh *load balancer*.

Lihat Tabel 6 untuk hasilnya, selama sepuluh kali percobaan dengan menggunakan metode *least_connection*, *load balancer* mengakses api-1 sebanyak 4 kali, api-2 sebanyak 4 kali dan api-3 sebanyak 2 kali. Hal ini menunjukkan bahwa saat penulis mengakses, server tidak dalam keadaan sibuk sehingga jarang sekali mengkses api-3 dan cukup mengakses api-1 dan api-2 saja.

Selain percobaan ini untuk membuktikan server masih berjalan meskipun salah satu dari API *web service* terjadi kesalahan, penulis mematikan api-1. Hasilnya bisa dilihat pada Tabel 7. Selama percobaan sepuluh kali tidak terjadi kegagalan karena pekerjaan yang harusnya dikerjakan api-1 akan dialihkan ke api-2 dan api-3.

Tabel 6. Percobaan *load balancer*

Percobaan ke-	API web service
1	api-1
2	api-2
3	api-1
4	api-2
5	api-3
6	api-1
7	api-2
8	api-3
9	api-1
10	api-2

Tabel 7. Percobaan jika api-1 dimatikan

Percobaan ke-	API web service
1	api-2
2	api-3
3	api-3
4	api-2
5	api-3
6	api-2
7	api-3
8	api-3
9	api-2
10	api-2

4. KESIMPULAN DAN SARAN

4.1 Kesimpulan

Berdasarkan penelitian "Penentuan Jumlah Server Menggunakan Server Response Time untuk Penerapan Load Balancer pada Aplikasi Pendaftaran Siswa Baru" diperoleh kesimpulan sebagai berikut:

1. Untuk memperoleh *server response time* (SRT) bisa dilakukan menggunakan *tool*, salah satunya loader.id. Penelitian ini menghasilkan SRT sebesar 464 ms per 100 pengguna.
2. Dengan *server response time* bisa diperkirakan berapa jumlah server yang dibutuhkan untuk bisa diakses sekian pengguna secara konkuren, dengan harapan time to first byte (TTFB) bernilai cukup maka aplikasi PPDB ini memerlukan 7 VPS per 1000 pengguna.
3. Penelitian ini sudah berhasil menerapkan *load balancer* dengan metode *least connection*, dengan melakukan pengujian terhadap 3 API *web service* sebanyak 10 kali percobaan akses.

4.2 Saran

Berdasarkan penelitian yang dikembangkan ada beberapa hal yang bisa ditambahkan diantaranya:

1. Menggunakan beberapa *tool* untuk mengetahui SRT sebagai perbandingan
2. Benar-benar menerapkan VPS yang diperlukan berdasarkan jumlah pengguna danTTFB
3. Melakukan pengujian fitur yang seringkali diakses dan beresiko terjadi kegagalan

ISSN: 2503-071X
E-ISSN: 2621-511X

PUSTAKA

Creating a Test. (2022, April 25). Retrieved from Loader: https://support.loader.io/article/15-creating-a-test

HTTP Load Balancing. (2022, April 25). Retrieved from NGINX Docs: https://docs.nginx.com/nginx/admin-guide/load-balancer/http-load-balancer/

Ibrahim Mahmood Ibrahim, Siddeeq Y. Ameen, Hajar Maseeh Yasin, dkk. (2021). Web Server Performance Improvement Using Dynamic Load Balancing Techniques: A Review. *Asian Journal of Research in Computer Science*, 47-62.

Kang, S. K. (2021). Review on Traffic Engineering and Load Balancing Techniques in Software Defined Networking. *2021 Third International Conference on Inventive Research in Computing Applications (ICIRCA)* (p. 312). Springer.

Khan, S. H. (2021). Bringing Innovative Load Balancing to NGINX. *International Journal of Research in Engineering, Science and Management*, 121–126.

M. Kushwaha, B. L. Raina and S. Narayan Singh. (2021). Implementation Analysis of Load Balancing Procedures for Cloud Computing Domain. *2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)* (pp. 287-292). IEEE.

Malik, A. A. (2021). *Pembuatan Aplikasi Android Simulasi Load Balancing Menggunakan Algoritma Round Robin dan Least Connection Di Sekolah Vokasi Ipb*. Bogor: Institut Pertanian Bogor.

Ramadani, F. H. (2018). Web Services: A Comparison of Soap and Rest Services. *Modern Applied Science*, 175.