# Perbandingan Algoritma *Weighted Least Connection* dan *Weighted Round Robin* pada Load Balancing Berbasis *Docker Swarm*

Salsabilah Aulia Rahman[1], T. Yudi Hadiwandra[2]
[1, 2] Universitas Riau, Kampus Bina Widya KM. 12,5, Simpang Baru, Kec. Tampan, Kota Pekanbaru, Riau 28293, Indonesia
*Email: salsabilah.aulia0117@student.unri.ac.id[1], tydihw@lecturer.unri.ac.id[2]*

**Abstrack** - Technological advancements have altered Indonesians' television viewing habits from analogue to digital television and the popularity of video-on-demand services. However, the enhancing use of video-on-demand services poses a challenge to sustaining servers' performance in the face of high demand. The solution is implementing efficient load balancing and scheduling algorithms, such as Docker Swarm, as a container virtualisation technology. This study examined three video resolution categories, 240p, 720p, and 1080p, and requests ranging from 1 to 10.000. The study's findings demonstrated that the higher the video resolution, the larger the video file size, which reduced throughput and enhanced response time, request failure, and CPU utilization. Based on the specified parameters, the analysis revealed that the weighted least connection algorithm is preferable to the weighted round-robin algorithm. Implementing a docker swarm positively affected server performance maintenance compared to a single server. This study presented further insight concerning implementing technology and algorithms to enhance video-on-demand servers' performance and ensure users' optimal viewing experience.

**Keywords** - *load balancing, docker swarm, weighted least connection, weighted round robin, video on demand*

***Intisari*** - Perkembangan teknologi telah mengubah kebiasaan menonton televisi di Indonesia dengan peralihan dari televisi analog ke digital dan popularitas layanan *video on demand*. Namun, peningkatan penggunaan video on demand membawa tantangan dalam menjaga performa server yang harus melayani permintaan tinggi. Solusinya adalah menerapkan teknik load balancing dan algoritma penjadwalan yang efisien, seperti menggunakan *docker swarm* sebagai teknologi virtualisasi container. Penelitian ini menguji 3 kategori resolusi video 240p, 720p, dan 1080p serta variasi permintaan *request* dari 1 hingga 10.000. Hasil penelitian menunjukkan bahwa semakin besar resolusi video, ukuran file video juga semakin besar, yang mengakibatkan penurunan *throughput* dan peningkatan *response time*, *request loss*, serta *CPU Utilization*. Hasil analisis menunjukkan bahwa algoritma *weighted least connection* lebih unggul daripada *weighted round robin* berdasarkan parameter yang ditentukan. Implementasi *docker swarm* berdampak positif dalam menjaga performa server dibandingkan dengan menggunakan single server. Penelitian ini memberikan wawasan mengenai penerapan teknologi dan algoritma dalam meningkatkan performa server *video on demand* serta memastikan pengalaman menonton yang optimal bagi pengguna.

**Kata Kunci -** *load balancing, docker swarm, weighted least connection, weighted round robin, video on demand*

## I. PENDAHULUAN

Salah satu contoh kemajuan teknologi di Indonesia yang terkait dengan menonton televisi adalah kemunculan berita pada awal tahun 2022 tentang rencana pemerintah untuk menghapuskan televisi analog secara bertahap. Namun, situasi tersebut tidak menimbulkan masalah serius karena pemerintah telah menyediakan opsi alternatif seperti layanan *video on demand* sebagai solusi pengganti [1]. Data yang diperoleh dari PT The Nielsen Company

Indonesia menunjukkan bahwa peralihan dari televisi analog ke digital telah meningkatkan jumlah penonton layanan *video on demand* atau *streaming* sebesar 36,8%, sedangkan penonton televisi broadcast hanya sebesar 24,2% [2].

Permintaan yang tinggi terhadap layanan *video on demand* menyebabkan penurunan performa server yang disebabkan oleh resource yang tidak memadai, seperti kebutuhan bandwidth yang besar, routing yang efisien, dan metode pengiriman konten yang tepat. Hal ini memunculkan masalah pada penyedia layanan web video on demand, dimana saat mengalami penurunan performa terdapat keluhan dari pelanggan hingga pelanggan tersebut pindah berlangganan ke penyedia layanan web lainnya [3]. Hal ini menyebabkan kerugian yang semakin parah jika penyedia layanan web tidak melakukan pengembangan pada servernya. Situasi ini menyebabkan perlunya menerapkan teknik yang tepat untuk memuat layanan server *video on demand* dengan memiliki tingkat ketersediaan tinggi dan mampu menangani beban traffic yang tinggi melalui proses pemerataan kompleks.

Penerapan *load balancing* pada web server membantu membagi beban trafik secara merata. Namun, ketika server dihadapkan pada beban trafik yang semakin besar, diperlukan algoritma penjadwalan yang dapat mengatasi beban tersebut dan mengurangi beban kerja pada server tersebut [4]. Penulis akan menggunakan dua algoritma penjadwalan dari *load balancing* dalam server *video on demand*, yaitu algoritma *weighted least connection* dan *weighted round robin*. Algoritma sebelumnya tidak memenuhi kebutuhan resource dari server *video on demand* karena tidak memperhitungkan bobot beban server dan tidak memanfaatkan server yang memiliki kapasitas lebih besar. Untuk mengatasi penurunan performa, penulis menerapkan teknik yang dapat membagikan lalu lintas secara merata untuk mengoptimalkan kinerja server. Implementasi kedua algoritma ini menggunakan teknologi virtualisasi berbasis container berupa *docker swarm* [5]. Hasil penelitian menunjukkan bahwa penggunaan *docker swarm* telah terbukti mampu meningkatkan kinerja dengan memanfaatkan sumber daya secara lebih efisien [6]. *Docker swarm* dapat diimplementasikan di google cloud platform yang menawarkan solusi skalabilitas dalam layanan cloud computing[7].

Berdasarkan permasalahan yang telah dijelaskan diatas, maka penulis meneliti layanan web *video on demand* menggunakan *docker swarm* dan membandingkan pada dua algoritma penjadwalan untuk melihat kinerja dalam meningkatkan performa web serta memiliki tingkat ketersediaan tinggi yang mampu menangani beban traffic yang tinggi melalui proses pemerataan kompleks.

## II. SIGNIFIKANSI STUDI

### *A. Tinjauan Pustaka*

Penelitian pertama telah dilakukan oleh [8] dengan judul *Analisis Performansi Proses Scaling pada Kubernetes dan Docker swarm Menggunakan Metode Horizontal Scaler*. Penelitian ini menunjukkan bahwa *docker swarm* memiliki performa lebih baik dalam uji load testing, sementara *kubernetes* unggul dalam scaling up dan scaling down. Kedua *container orchestration* ini dapat mengatasi perubahan intensitas beban kerja dari waktu ke waktu yang dapat diterapkan dalam mengatasi penurunan performa server, namun penelitian ini belum mencakup pengujian algoritma penjadwalan.

Penelitian kedua dilakukan oleh [4] dengan judul *Implementasi Load Balancing dengan Lightweight Virtualization Menggunakan Docker untuk Layanan Video on Demand*. Penelitian ini menerapkan *load balancing* pada container docker dengan *kubernetes* dan menunjukkan kinerja yang lebih baik dibandingkan dengan server tunggal. *Load balancing* membagi beban kerja dan trafik ketiga server, meningkatkan efisiensi dan ketahanan sistem. Penelitian hanya berfokus pada *kubernetes*, algoritma *round robin*, dan *least connection*, sehingga belum diterapkan dengan algoritma lain yang memiliki kinerja berbeda.

Penelitian ketiga dilakukan oleh [9] dengan judul *Implementasi Virtualisasi Server Berbasis Docker Container.* Penelitian ini menunjukkan bahwa *docker* dapat memanfaatkan sumber daya hardware dengan baik dan efisien, ditunjukkan dengan penggunaan CPU di bawah 15% dan memori di bawah 7 MiB. Oleh karena itu, dapat diterapkan dalam server *video on demand* dalam meningkatkan ketersediaan server.

Penelitian keempat dilakukan oleh [10] dengan judul *Implementasi Load Balancing Dengan Algoritma Penjadwalan Weighted Round Robin Dalam Mengatasi Beban Web Server*. Penelitian ini menunjukkan bahwa kinerja load balancing pada cluster dapat diukur dari *response time* dan *throughput*. *Weighted round robin* memiliki dampak signifikan untuk web server, namun penelitian ini belum menggunakan *docker swarm* dan hanya fokus pada satu algoritma, sehingga belum mengatasi masalah pada *video on demand.*

Penelitian kelima dilakukan oleh [5] dengan judul *Implementasi* Load Balancing *dengan Algoritma Least Weight Connection Menggunakan Zevenet*. Penelitian ini menunjukkan bahwa *load balancing* penting untuk membagi beban kinerja server. Metode pembagian beban dengan memberikan bobot performa pada setiap server terbukti efektif. Namun, penelitian ini hanya berfokus pada zevenet untuk *load balancing,* sehingga belum menguji *docker swarm* dengan algoritma *weighted least connection.*

Penelitian keenam dilakukan oleh [11] dengan judul *Pemanfaatan Google Cloud Dan Teknik Load Balancing Untuk Optimalisasi Performa Akses Halaman Web.* Penelitian ini menggunakan metode apache benchmark menunjukkan bahwa load balancing CPU dengan Google Cloud Platform berhasil mengatasi server secara efektif. Namun, penelitian ini belum menerapkan dan menguji efektivitas algoritma penjadwalan dalam konteks *video on demand.*

Penelitian ketujuh dilakukan oleh [12] dengan judul *High Availability Server Using Raspberry Pi 4 Cluster and Docker swarm*. Penelitian ini menunjukkan bahwa Raspberry Pi Cluster tipe 4B dengan *docker swarm* dapat membangun sistem server berkinerja tinggi. Penelitian ini berfokus pada penggunaan Raspberry Pi Cluster tipe 4B sebagai solusi infrastruktur web server yang ekonomis dan efisien dalam penggunaan energi, namun belum melibatkan penelitian menggunakan *cloud computing* sebagai alternatif.

## *B. Landasan Teori*

### *1. Video on Demand*

Video on Demand (VOD) adalah perangkat lunak yang memungkinkan pengguna menonton video secara online dari berbagai lokasi. Layanan ini termasuk dalam kategori *streaming on-demand*, di mana video dijalankan langsung dari server penyimpanan media dan dapat diakses melalui perangkat client pengguna. Pengguna dapat menikmati berbagai jenis video, seperti film, video edukasi, dan animasi, baik dengan cara diunduh terlebih dahulu atau dengan melakukan streaming langsung [4]. Layanan VOD dapat menggunakan berbagai proses, seperti streaming, progressive downloading, atau metode download biasa.

### *2. HLS Video Streaming*

HLS (HTTP Live Streaming) adalah protokol untuk mengirimkan video dan audio secara real-time melalui jaringan HTTP. Video dipecah menjadi segmen kecil yang dapat diunduh dan diputar secara independen. Pengaturan pemutaran video ditentukan oleh file playlist M3U8, yang memandu pemutar video dalam memilih segmen yang akan diunduh dan diputar [13]. HLS adalah standar layanan streaming video yang populer dan diakui sebagai pilihan utama untuk menyajikan pengalaman streaming yang optimal bagi pengguna.

### *3. Resolusi Video*

Terdapat beberapa tingkatan resolusi video yang umum digunakan, seperti resolusi 240p, 720p, dan 1080p. Pemilihan resolusi video yang tepat perlu mempertimbangkan faktor-faktor

seperti jenis konten yang akan ditampilkan, perangkat yang digunakan oleh pengguna, dan kondisi jaringan. Resolusi yang lebih tinggi cenderung menghasilkan tampilan yang lebih baik, namun juga memerlukan lebih banyak bandwidth dan sumber daya untuk melakukan streaming atau pemutaran video. Oleh karena itu, perlu mempertimbangkan keseimbangan antara kualitas visual dan efisiensi sumber daya dalam memilih resolusi video[14] .

*4. Load Balancing*

Load balancing mengurangi risiko overload pada server dengan membagi beban trafik dari client ke beberapa jalur. Tujuannya adalah agar permintaan dari client tidak menumpuk pada satu server, dan server dapat beroperasi optimal. Teknik *load balancing* mengalokasikan lalu lintas masuk ke berbagai server, memastikan penanganan permintaan klien yang cepat dan efisien. Dengan demikian, *load balancing* meningkatkan kinerja dan kehandalan server saat menghadapi beban trafik tinggi [4].

*5. Virtualization Container*

Virtualisasi berbasis container atau virtualization container adalah salah satu teknik virtualisasi server yang berbeda dengan virtual mesin tradisional. Dalam virtualisasi berbasis container, seperti yang digunakan oleh *Docker*, tidak memerlukan pengaturan virtual mesin atau hypervisor, karena *Docker* telah menyediakan layanan yang memisahkan sistem operasi secara mandiri.*Docker* merupakan contoh dari teknologi virtualisasi berbasis container yang telah populer dan banyak digunakan dalam dunia pengembangan perangkat lunak [6].

*6. Docker swarm*

*Docker swarm* adalah kontroler container untuk mengatur dan mendistribusikan container dalam skala besar [8]. Terdiri dari node Manager dan node Worker, *Docker swarm* mengoptimalkan distribusi dan penjadwalan container secara efisien. Cocok untuk mengelola server *video on demand* dengan kebutuhan resource yang memadai [6]. Platform ini menggunakan routing mesh, memungkinkan setiap node dalam *swarm* mengakses semua port layanan yang berjalan di dalamnya [15].

*7. Weighted Least Connection*

Algoritma Weighted Least Connection (WLC) merupakan pengembangan dari algoritma Least Connection. Algoritma ini membagi beban trafik sesuai dengan kapasitas pemrosesan server yang telah ditentukan sebelumnya. Penggunaan WLC mencegah overload pada server saat permintaan tinggi, sehingga meningkatkan performa keseluruhan server secara optimal. Bobot kapasitas pemrosesan diatur dalam pengaturan virtual server berdasarkan jadwal koneksi jaringan [5].

*8. Weighted Round Robin*

Algoritma *weighted round robin* adalah pengembangan dari algoritma *round robin*. Pada *weighted round robin*, server diberikan bobot beban berdasarkan kemampuannya. Server dengan bobot lebih tinggi akan menerima lebih banyak koneksi dari client. Hal ini memungkinkan distribusi trafik yang lebih adil dan efisien, meningkatkan responsivitas sistem secara optimal[16].

*9. Parameter Pengujian*

Parameter pengujian yang digunakan dalam penelitian ini meliputi *throughput*, *response time*, *request loss*, dan *CPU Utilization*.

a. *Throughput* adalah mengukur jumlah data atau permintaan yang berhasil diproses oleh server dalam satuan waktu tertentu. Parameter ini memberikan informasi penting tentang

efektivitas transmisi data dalam lingkungan penelitian yang bersangkutan [14]. T*hroughput* ini dapat diukur dalam bentuk request per detik (RPS) atau permintaan per detik.

b. *Response time* merupakan waktu yang diperlukan untuk menyelesaikan satu permintaan (*request*) dan mengirimkannya kembali ke client. Parameter ini mengukur seberapa cepat server memberikan respons terhadap permintaan yang masuk. Satuan response time diukur dalam "milliseconds" (ms) atau milidetik.
c. *Request loss* mengacu pada persentase paket data yang gagal proses transmisi. Parameter ini mengukur jumlah permintaan yang tidak berhasil diproses atau direspon oleh server karena terlalu banyak beban atau alasan lainnya.
d. *CPU Utilization* adalah parameter ini mengukur sejauh mana CPU pada server digunakan untuk memproses permintaan. Satuannya adalah "persentase" (%), yang menunjukkan seberapa besar CPU digunakan dalam bentuk persentase dari kapasitas total CPU yang ada. CPU utilization yang tinggi dapat menandakan bahwa server sedang mengalami beban yang berat.

Pengujian dilakukan menggunakan aplikasi Jmeter, yang berfungsi untuk mengamati dan menganalisis hasil dari masing-masing parameter pengujian. Hasil-hasil dari parameter-perameter performa ini akan memberikan gambaran tentang seberapa baik performa dari aplikasi atau server yang sedang diuji.

## *C. Metode Penelitian*

Penyusunan skripsi ini ada beberapa metodologi yaitu studi literatur, identifikasi masalah, identifikasi kebutuhan, perancangan sistem, implementasi sistem, pengujian sistem dan analisis hasil pengujian.

### *1. Studi Literatur*

Metode penelitian ini melibatkan pengumpulan sumber data dari literatur seperti jurnal, prosiding, artikel ilmiah, dan bacaan terkait dengan judul penelitian. Sumber data diperoleh dari penelitian sebelumnya dari e-library universitas lain, internet, Google Scholar, dan IEEE. Penulis melakukan pengembangan dari literatur yang ada untuk mengidentifikasi masalah penelitian yang dapat diinvestigasi guna membantu menyelesaikan penelitian ini.

### *2. Identifikasi Masalah*

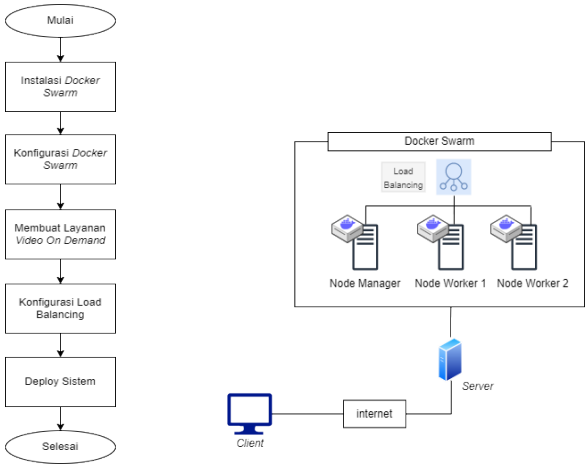
Masalah yang diteliti dalam penelitian ini adalah penurunan performa web server Video on Demand akibat tingginya permintaan dengan menggunakan docker container dan algoritma *weighted least connection* serta *weighted round robin* untuk load balancing berbasis *docker swarm*.

### *3. Identifikasi Kebutuhan*

Penelitian ini mengidentifikasi kebutuhan spesifikasi perangkat untuk membuat sistem yang akan diteliti, termasuk perangkat keras seperti Google Cloud Server dan laptop untuk analisis performa Docker swarm. Perangkat lunak yang digunakan termasuk *docker swarm, Nginx,* dan *Apache JMeter.*

### *4. Perancangan Sistem*

Perancangan sistem ini menjelaskan tahapan implementasi *load balancing* dengan algoritma *weighted least connection* dan *weighted round robin* pada *video on demand* berbasis docker swarm menggunakan google cloud platform. Dilakukan juga perancangan flowchart sistem *load balancing* menggunakan *docker swarm* dan arsitektur *docker swarm* untuk sistem cloud, seperti di gambar 1 ini:




(a) (b)

Gambar 1 (a) Flowchart Perancangan Sistem (b) Arsitektur *Docker swarm*

*5. Implementasi*

Pada tahap implementasi, sistem dibangun berdasarkan rancangan yang telah dibuat. Fokusnya adalah meneliti implementasi load balancing menggunakan google cloud platform dengan algoritma *weighted least connection* dan *weighted round robin* pada layanan Server *Video on Demand* berbasis virtualisasi container *docker swarm*.

*6. Pengujian Sistem*

Pada tahap pengujian, penelitian ini menggunakan Apache JMeter sebagai alat untuk menguji load balancing berdasarkan parameter yang telah ditentukan. Pengujian akan melibatkan skenario pengujian tertentu yang akan dijalankan dalam uji coba load balancing, sebagai tabel I berikut:

TABEL I
SKENARIO PENGUJIAN

| Skenario | Algoritma Load Balancing | Jenis Video | Besar *Request* | Keterangan |
|---|---|---|---|---|
| Skenario 1 | Tanpa Load Balancing (single server) | 240p, 720p dan 1080p | 10<br>100<br>1000<br>10000 | Parameter Pengujian:<br>1. *Throughput*<br>2. *Response time*<br>3. *Request loss*<br>4. *CPU Utilization*<br>*Request* yang akan dilakukan adalah melakukan akses ke web server menggunakan tool apache Jmeter. Pengujian ini langsung ke video yang ada dan dilihat hasil dari media segmentnya. |
| Skenario 2 | *Weighted Least Connection* | 240p, 720p dan 1080p | 10<br>100<br>1000<br>10000 | |
| Skenario 3 | *Weighted Round Robin* | 240p, 720p dan 1080p | 10<br>100<br>1000<br>10000 | |

*7. Analisis Hasil Pengujian*

Pada bagian ini akan menampilkan hasil analisis sesuai dengan metode pengujian yang digunakan. Hal ini dilakukan dengan membandingkan hasil-hasil pengujian tersebut dan menentukan perbandingan yang terbaik. Hasil analisis pengujian ini akan dianalisa dengan melakukan perbandingan dengan setiap algoritma *load balancing* beserta jumlah *request*.

## III. HASIL DAN PEMBAHASAN

### A. *Konfigurasi Sistem*

Dalam tahap ini dilakukan dengan mengkonfigurasi komponen dalam suatu sistem agar dapat berfungsi dengan baik. Seperti yang telah dijelaskan di identifikasi kebutuhan terdapat 3 *instance* untuk *docker swarm* (satu *node manager* dan dua *node worker*) dan 1 instance untuk *single server.* Berikut spesifikasi dari server- server tersebut, dapat dilihat pada tabel II:

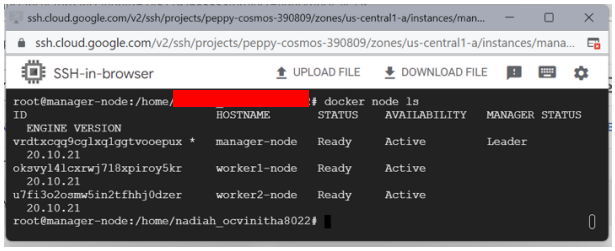
TABEL II
SPESIFIKASI SISTEM

| Perangkat | Spesifikasi | Keterangan |
|---|---|---|
| ***Node Manager*** | Tipe Mesin | n1-Standard-1 |
| | OS | Ubuntu 20.04 |
| | vCPU | 1 vCPU |
| | RAM | 3,75 GB |
| ***Node Worker 1*** | Tipe Mesin | n1-Standard-1 |
| | OS | Ubuntu 20.04 |
| | vCPU | 1 vCPU |
| | RAM | 3,75 GB |
| ***Node Worker 2*** | Tipe Mesin | n1-Standard-2 |
| | OS | Ubuntu 20.04 |
| | vCPU | 2 vCPU |
| | RAM | 7,5 GB |
| ***Single Server*** | Tipe Mesin | n1-Standar-1 |
| | OS | Ubuntu 20.04 |
| | vCPU | 1 vCPU |
| | RAM | 3,75 GB |

Berikut adalah konfigurasi yang dilakukan didalam setiap instance dalam menyiapkan layanan *video on demand* dengan *docker swarm.*

1. Memperbarui package list
   ```
   #sudo apt update
   ```
2. Memperbarui
   ```
   #sudo apt upgrade
   ```
3. Install docker.io
   ```
   #sudo apt install docker.io
   ```
4. Membuat host di node Manager
   ```
   #docker swarm init
   ```
   Perintah ini akan menghasilkan token untuk memasukkan node Worker ke *swarm.*

*Node manager* dan *node worker* dilakukan konfigurasi agar *docker swarm* dapat dijalankan di server, status dari *Swarm* yang berjalan dapat dilihat pada gambar 2 berikut:

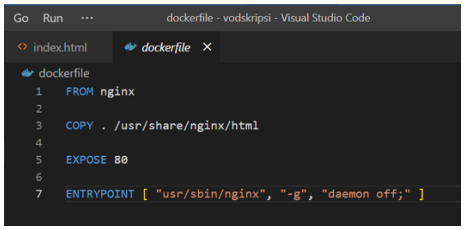



Gambar 2 Status *Swarm*

Setelah konfigurasi *docker swarm* dilakukan, dilanjutkan dengan memasukkan layanan *video on demand* dengan cara seperti berikut:

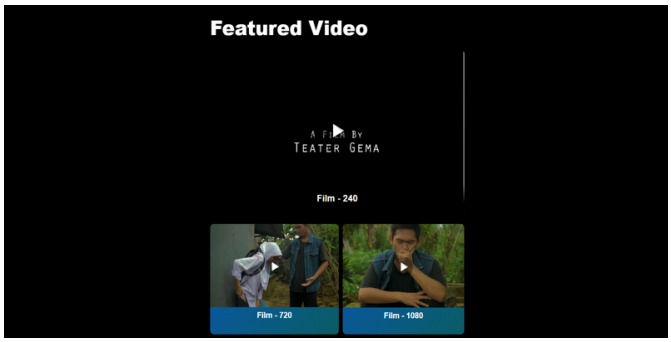
1. Mempersiapkan website *video on demand*
2. Mempersiapkan coding *dockerfile*

   Berikut coding *dockerfile* seperti di gambar 3:




Gambar 3 *Dockerfile*

3. Menjalankan dockerfile
   `#docker build -t namaimage .`
4. Mengunggah image ke repository docker
   `#docker push namaimage:v1`
5. Memasukkan image ke server
   `#docker pull namaimage:v1`
6. Menjalankan docker image
   `#docker create service –replicas –name namaservice namaimage`
   Tahapan ini menampilkan tampilan layanan video on demand di setiap server dengangambaran tampilan seperti yang terlihat pada gambar 4.




Gambar 4 Tampilan *Video on Demand*

Setelah layanan *video on demand* dapat berjalan pada sistem maka selanjutnya dilakukan konfigurasi *load balancing* pada *node manager* dengan memasukkan perintah di **nano /etc/nginx/sites-available/default.** Berikut konfigurasi algoritma *weighted least connection:*

```
upstream backend {
least_conn;
server 34.101.89.208 weight=2;
server 34.101.191.4 weight=3;
}
server {
listen 80;
location /{
proxy_pass http://backend;
} }
```

Berikut konfigurasi algoritma *weighted round robin*:

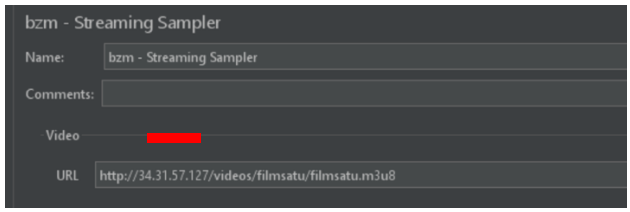
```
upstream backend {
server 34.101.89.208 weight=2;
server 34.101.191.4 weight=3;
}
server {
listen 80;
location /{
proxy_pass http://backend;
} }
```

*B. Analisis hasil Pengujian*

Analisis hasil pengujian menggunakan data media segment yang berisi segment-segment video dari sistem. Data diambil dari aplikasi Jmeter untuk melihat performa server atau aplikasi yang diuji, yang pengambilannya dilakukan seperti berikut:

1. Memasukkan URL yang akan diuji

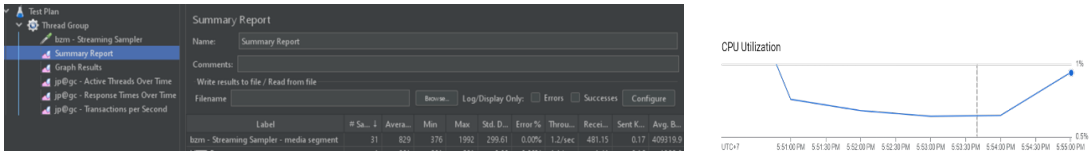
   URL yang diuji adalah URL video yang akan ditampilkan oleh server *video on demand,* seperti di gambar 5 berikut:




Gambar 5 Tampilan URL Pengujian

2. Memantau hasil pengujian

   Hasil pengujian ini dilihat pada Jmeter untuk parameter *throughput, response time* dan *request lost*, sedangkan *CPU Utilization* bisa dilihat di Google Cloud Platform. Hal ini dapat dilihat pada gambar 6 berikut:



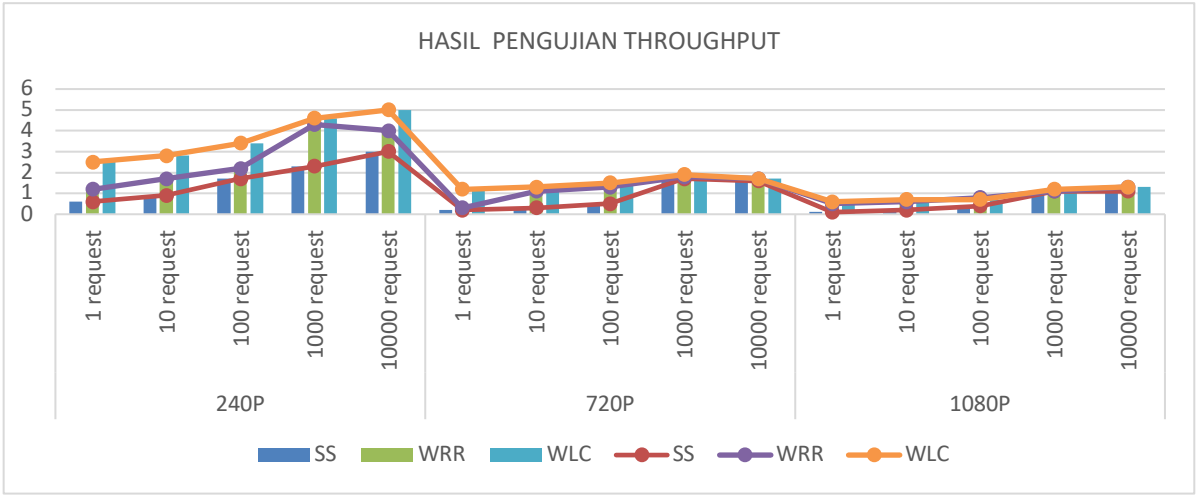

(a) (b)

Gambar 6 (a) Hasil Pengujian di Jmeter (b) Hasil Pengujian CPU Utilization

Oleh karena itu, berikut hasil pengujian *load balancing* berdasarkan parameter yang ditentukan:

*1. Throughput*

Terdapat hubungan terbalik antara penggunaan *throughput* dan resolusi file. Semakin kecil resolusi file, semakin besar *throughput* yang dihasilkan. Selain itu, pengujian dengan jumlah *request* yang lebih besar juga menghasilkan *throughput* yang lebih tinggi.




Gambar 7 Hasil Pengujian *Throughput* Keseluruhan

*1) Pengujian Throughput 240P*

Pada gambar 7 terlihat hasil pengujian menunjukkan bahwa algoritma *Weighted Least Connection* (WLC) memiliki performa terbaik dalam hal *throughput* pada kondisi 1, 10, 100, 1000, dan 10000 *request* pada resolusi video 240P dengan nilai *throughput* yang lebih tinggi daripada algoritma *Weighted Round Robin* (WRR). Pada kondisi 1 *request*, algoritma WLC menunjukkan performa yang baik dengan nilai *throughput* 2,5 permintaan per detik. Pada pengujian dengan jumlah *request* yang lebih tinggi, algoritma WLC tetap unggul dengan *throughput* mencapai nilai 5 permintaan per detik pada kondisi 10000 *request* daripada algoritma WRR yang mencapai nilai 4 permintaan per detik. Dengan demikian, algoritma WLC memiliki kemampuan yang lebih baik dalam menangani beban lalu lintas yang lebih besar dan menghasilkan *throughput* yang optimal daripada.

*2) Pengujian Throughput 720P*

Pada gambar 7 terlihat hasil pengujian menunjukkan bahwa algoritma *Weighted Least Connection* (WLC) memiliki performa terbaik dalam hal *throughput* pada kondisi 1, 10, 100, 1000, dan 10000 *request* pada resolusi video 720P dengan nilai *throughput* yang lebih tinggi daripada algoritma *Weighted Round Robin* (WRR). Pada kondisi 1 *request*, algoritma WLC menunjukkan performa yang baik dengan nilai *throughput* 1,2 permintaan per detik. Pada pengujian dengan jumlah *request* yang lebih tinggi, algoritma WLC tetap unggul dengan *throughput* mencapai nilai 1,7 permintaan per detik pada kondisi 10000 *request*. Dengan demikian, algoritma WLC memiliki kemampuan yang lebih baik dalam menangani beban lalu lintas yang lebih besar dan menghasilkan *throughput* yang optimal.

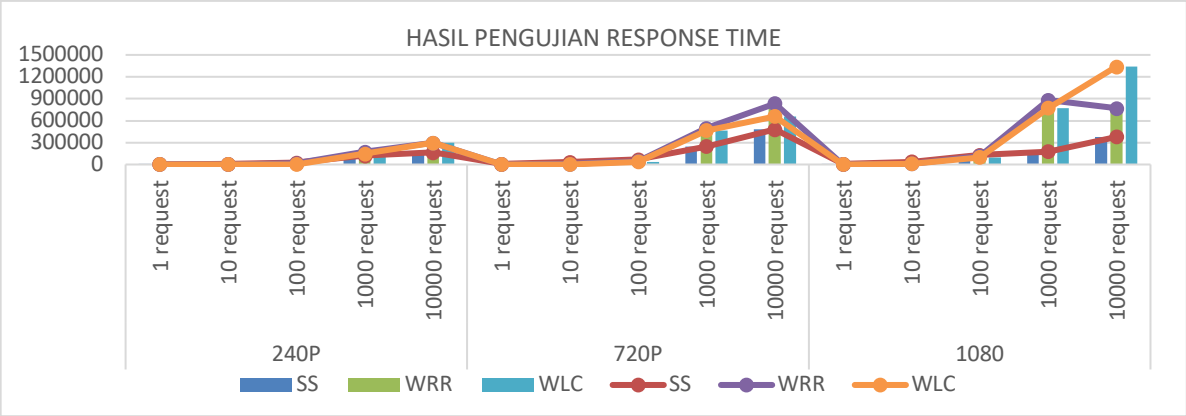
*3) Pengujian Throughput 1080P*

Pada gambar 7 terlihat hasil pengujian *throughput* di resolusi video 1080P, Algoritma *Weighted Least Connection* (WLC) menunjukkan *response time* yang lebih baik daripada algoritma *Weighted Round Robin* (WRR) dalam pengujian dengan 1, 10, 100, 1000, dan 10000 *request*. Namun dilihat dari *request* tertinggi yaitu 10000 *request* mendapatkan hasil yang sama yaitu 1,3 permintaan per detik. Hal ini dikarenakan algoritma WLC menghasilkan media segment lebih banyak dan membuat algoritma WLC mampu mempertahankan servernya jauh lebih baik. Dalam hal ini, algoritma WLC yang memiliki performa lebih baik untuk resolusi video 1080P.

Dari pengujian 3 resolusi, algoritma *Weighted Least Connection* (WLC) unggul dalam parameter throughput dibandingkan *Weighted Round Robin* (WRR). Algoritma WLC memberikan bobot pada setiap server berdasarkan koneksi aktifnya, sehingga server dengan koneksi lebih sedikit menerima lebih banyak permintaan. Di sisi lain, algoritma WRR membagi beban trafik secara bergantian tanpa mempertimbangkan beban koneksi masing-masing server. Dengan cara kerja yang lebih cerdas, algoritma WLC menghindari situasi di mana satu server menjadi bottleneck karena beban koneksi yang berlebihan, sehingga throughput yang dihasilkan lebih optimal.

*2. Response time*

Terdapat hubungan positif antara resolusi file dan *response time*, semakin besar resolusi file, maka lama waktu yang dibutuhkan untuk mengirimkan respons dari server ke klien. Terlihat juga bahwa semakin banyak jumlah *request* yang diuji, semakin besar *response time.*




Gambar 8 Hasil Pengujian *Response Time* Keseluruhan

*1) Pengujian Response time 240P*

Hasil pengujian pada gambar 8 menunjukkan bahwa algoritma *Weighted Least Connection* (WLC) memiliki performa terbaik dalam hal *response time* pada kondisi 1, 10, 100, 1000, dan 10000 *request* pada resolusi video 240P dengan nilai *response time* yang lebih rendah daripada algoritma *Weighted Round Robin* (WRR). Pada kondisi 1 *request*, algoritma WLC lebih baik dengan *response time* 396 milidetik. Pada pengujian dengan jumlah 10000 *request*, algoritma WRR unggul sedikit dengan *response time* yang lebih tinggi mencapai selisih 2080 milidetik daripada algoritma WLC, namun hal ini dikarenakan jumlah media segment yang dihasilkan algoritma WLC lebih banyak. Dengan demikian, algoritma WLC memiliki kemampuan yang lebih baik dalam menangani beban lalu lintas yang besar dan menghasilkan *response time* yang optimal.

*2) Pengujian Response time 720P*

Dilihat pada gambar 8 algoritma *Weighted Least Connection* (WLC) menunjukkan *response time* yang lebih baik daripada algoritma *Weighted Round Robin* (WRR) dalam pengujian dengan 1, 10, 100, 1000, dan 10000 *request* dengan resolusi video 720P. Pada pengujian dengan 1 *request* hingga 10000 *request* ini algoritma WRR mencapai *response time* tertingginya yaitu berada di 832597 milidetik, sedangkan algoritma WLC berada di 658822 milidetik. Dalam semua pengujian, algoritma WLC menunjukkan performa yang lebih baik dalam hal *response time* dibandingkan algoritma WRR.

*3) Pengujian Response time 1080P*

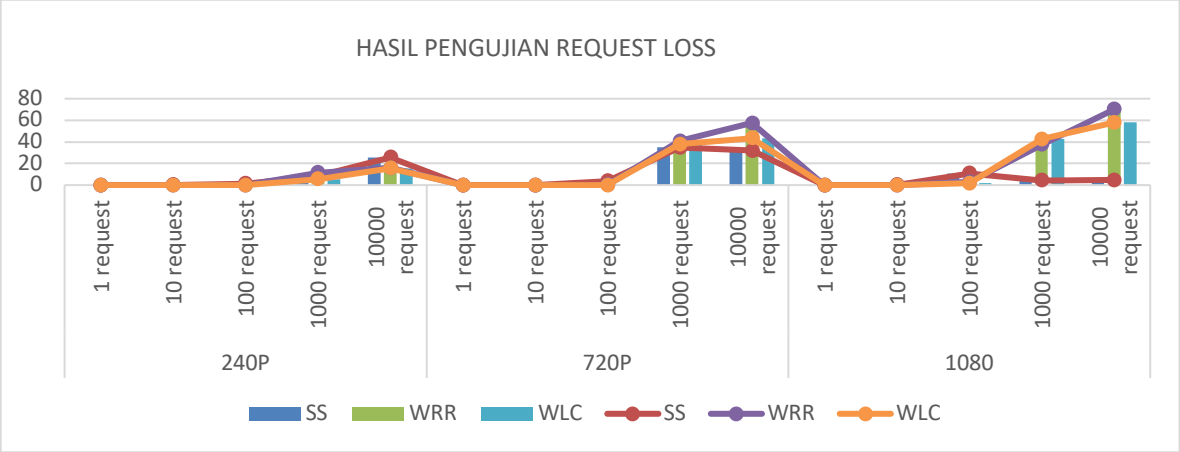
Hasil pengujian pada gambar 8 menunjukkan bahwa algoritma *Weighted Least Connection* (WLC) memiliki performa terbaik dalam hal *response time* pada kondisi 1, 10, 100, 1000, dan 10000 *request* pada resolusi video 1080P dengan nilai *response time* yang lebih rendah daripada algoritma *Weighted Round Robin* (WRR). Pada kondisi 1 *request*, algoritma WLC lebih baik dengan *response time* 1728 milidetik. Pada pengujian dengan jumlah 10000 *request*, algoritma WRR unggul sedikit dengan *response time* yang lebih rendah mencapai selisih 568742 milidetik daripada algoritma WLC, namun hal ini dikarenakan jumlah media segment yang dihasilkan algoritma WLC lebih banyak. Dengan demikian, algoritma WLC memiliki kemampuan yang lebih baik dalam menangani beban lalu lintas yang lebih besar dan menghasilkan *response time* yang optimal.

Dari pengujian tiga resolusi, algoritma *Weighted Least Connection* (WLC) menunjukkan performa lebih unggul dibandingkan *Weighted Round Robin* (WRR) pada parameter *response time*. Hal ini karena algoritma WLC memberikan prioritas pada server dengan koneksi lebih sedikit, mengurangi beban pada server yang mungkin telah jenuh, dan merespons permintaan

lebih cepat. Di sisi lain, algoritma WRR membagi beban secara bergantian tanpa mempertimbangkan beban koneksi, menyebabkan beberapa server mengalami peningkatan response time akibat terlalu banyak permintaan yang harus dihandle. Dengan cara kerjanya yang efisien, algoritma WLC mampu mengoptimalkan *response time* dan memberikan performa lebih baik dalam merespons permintaan pengguna pada server *video on demand.*

*3. Request Lost*

Terdapat hubungan positif antara resolusi file dan jumlah *request lost*. Artinya, semakin besar resolusi file, semakin tinggi kemungkinan terjadi error dalam permintaan. Selain itu, terlihat juga bahwa semakin banyak jumlah *request* yang diuji coba, semakin banyak pula *request lost* yang dihasilkan.




Gambar 9 . Hasil Pengujian *Request Loss* Keseluruhan

*1) Pengujian Request loss 240P*

Terlihat dari gambar 9 algoritma *Weighted Least Connection* (WLC) memiliki performa yang lebih baik dalam mengurangi tingkat *request lost* dibandingkan dengan single server dan algoritma *Weighted Round Robin* (WRR). Dalam konteks ini, terlihat bahwa algoritma WRR memiliki tingkat *request lost* yang lebih tinggi dibandingkan dengan algoritma WLC yaitu di 10000 *request* mencapai 15,8% sedangkan algoritma WRR hanya di 15,7%. Hal ini menunjukkan bahwa penggunaan algoritma WLC dapat mengurangi jumlah *request lost*.

*2) Pengujian Request loss 720P*

Terlihat dari gambar 9 pengujian *request loss* di resolusi video 720P ini, dilihat dengan membandingkan antara dua algoritma, algoritma *Weighted Least Connection* (WLC) menunjukkan performa yang lebih baik dengan memiliki jumlah *request lost* yang lebih kecil dibandingkan algoritma lainnya. Dapat dilihat bahwa pada permintaan *request* tertinggi yaitu di 10000 *request*, algoritma *Weighted Round Robin* (WRR) mencapai *request loss* sebanyak 57,3% sedangkan algoritma WLC hanya menercapai 43,5%. Dalam konteks ini, dapat disimpulkan bahwa algoritma WLC lebih baik dalam mengatasi *request lost* dibandingkan dengan algoritma WRR di resolusi video 720P.

*3) Pengujian Request loss 1080P*

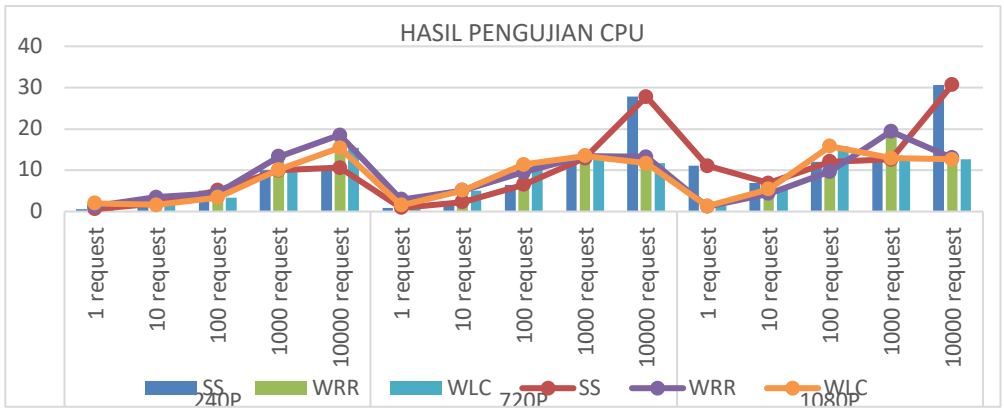
Terlihat dari gambar 9 pengujian *request loss* di resolusi video 1080P ini, dilihat dengan membandingkan antara dua algoritma, algoritma *Weighted Least Connection* (WLC) menunjukkan performa yang lebih baik dengan memiliki jumlah *request lost* yang lebih kecil dibandingkan algoritma lainnya. Dilihat bahwa *Weighted Round Robin* (WRR) menghasilkan *request lost* hingga 70%, hal ini menunjukkan bahwa algoritma WRR memiliki tingkat error

yang lebih tinggi dibandingkan dengan algoritma WLC. Dalam hal ini, algoritma WLC yang memiliki performa lebih baik untuk resolusi 1080P.

Dari pengujian tiga resolusi, algoritma *Weighted Least Connection* (WLC) unggul dibandingkan *Weighted Round Robin* (WRR) pada parameter *request loss.* Hal ini karena memberikan prioritas pada server dengan koneksi lebih sedikit, algoritma WLC menghindari server menjadi terlalu padat dan mengurangi tingkat *request loss*. Algoritma WLC memberikan performa lebih baik dalam menangani permintaan pengguna pada server *video on demand*.

*4. CPU Utilization*

Penggunaan CPU memiliki hubungan dengan resolusi file dan jumlah *request*. Resolusi file yang lebih besar akan menghasilkan penggunaan CPU yang lebih tinggi karena memerlukan lebih banyak sumber daya komputasi. Selain itu, semakin banyak jumlah *request* yang ditangani, semakin tinggi penggunaan CPU.




Gambar 10 Hasil Pengujian *CPU Utilization* Keseluruhan

*1) Pengujian CPU Utilization 240P*

Terlihat dari gambar 10 algoritma *Weighted Least Connection* (WLC) memiliki performa yang lebih baik dibandingkan algoritma *Weighted Round Robin* (WRR) pada resolusi video 240P karena menghasilkan tingkat penggunaan CPU yang lebih rendah. Dapat dilihat pada Gambar 10 dengan *request* tertinggi algoritma WRR mencapai nilai *CPU Utilization* di 18,5% sedangkan algoritma WLC hanya mencapai nilai 15,4% Oleh karena itu, algoritma WLC merupakan pilihan yang lebih baik untuk mengelola beban lalu lintas dan mengoptimalkan penggunaan sumber daya CPU.

*2) Pengujian CPU Utilization 720P*

Dari data yang didapatkan pada gambar 10 *Weighted Round Robin* (WRR) lebih unggul sedikit dari Algoritma *Weighted Least Connection* (WLC) pada resolusi video 720P karena menghasilkan tingkat penggunaan CPU yang lebih rendah. Hal ini dapat dilihat dari data tertinggi *CPU Utilization* didapatkan algoritma WLC mencapai nilai *CPU Utilization* di 13,5% sedangkan algoritma WRR hanya mencapai 13,4%. Hal ini menandakan algoritma WRR lebih baik jika di video dengan resolusi video 720P.

*3) Pengujian CPU Utilization 1080P*

Terlihat dari gambar 10 di resolusi video 1080P, algoritma *Weighted Least Connection* (WLC) memiliki performa yang lebih baik karena menghasilkan tingkat *CPU Utilization* yang lebih rendah. Hal ini dapat dilihat dari data tertinggi *CPU Utilization* didapatkan

algoritma WRR mencapai nilai *CPU Utilization* di 19,4% sedangkan algoritma WLC hanya mencapai 15,8%. Hal ini menandakan Algoritma WLC memberikan solusi yang lebih baik dalam mengoptimalkan penggunaan CPU dan memperbaiki kinerja sistem secara keseluruhan.

Dari pengujian tiga resolusi, algoritma *Weighted Least Connection* (WLC) unggul dibandingkan *Weighted Round Robin* (WRR) pada parameter *CPU Utilization*. Algoritma WLC memberikan prioritas pada server dengan koneksi lebih sedikit, mengurangi beban pada server yang telah jenuh. Sebaliknya, pada algoritma WRR, beberapa server mungkin mengalami beban CPU lebih besar karena menerima permintaan lebih banyak, sedangkan server lainnya memiliki beban CPU yang lebih rendah karena menerima permintaan lebih sedikit. Dengan cara kerja yang lebih efisien, algoritma WLC memiliki *CPU utilization* yang lebih rendah dibandingkan WRR.

## IV. KESIMPULAN

Penelitian ini menunjukkan bahwa *docker swarm* dapat diimplementasikan dalam layanan web *video on demand*, mengoptimalkan penggunaan CPU pada server dengan membagikan kerja ke node-node lain, dikarenakan pada *single server* mencapai *CPU Utilization* di 70%. Hasil pengujian menunjukkan bahwa algoritma *Weighted Least Connection* (WLC) lebih unggul daripada *Weighted Round Robin* (WRR) dalam parameter *throughput, response time, request loss,* dan *CPU Utilization* pada resolusi video 240p, 720p, dan 1080p di semua kombinasi jumlah *request.* Namun, pada resolusi video 720p, *weighted round robin* memiliki sedikit performa lebih baik dalam CPU Utilization yaitu hanya selisih di 0,1%. Perbedaan ini disebabkan oleh cara kerja keduanya dalam mengatasi variasi beban trafik di antara server-server. Dengan memberikan prioritas kepada server dengan koneksi yang lebih sedikit, *Weighted Least Connection* (WLC) dapat menghindari situasi di mana satu server menjadi terlalu padat sementara server lainnya tidak optimal. Oleh karena itu, *Weighted Least Connection* (WLC) dalam load balancing pada server *video on demand* dapat meningkatkan performa server secara efisien dan menciptakan server yang memiliki ketersediaan tinggi. Dapat dilihat data secara keseluruhan bahwa semakin besar resolusi video tentu memiliki ukuran file video yang besar pula, oleh karena itu menghasilkan throughput semakin kecil dan response time, eror request beserta *CPU Utilization* semakin besar. Untuk pengembangan selanjutnya, disarankan untuk menguji berbagai algoritma penjadwalan dan parameter guna meningkatkan efisiensi pemanfaatan sumber daya hardware.

## REFERENSI

[1] Iskandar, "HEADLINE: ASO Alihkan Siaran TV Analog ke TV Digital, Apa Perbedaan, Keunggulan, dan Manfaatnya?," Nov. 03, 2022.

[2] T. N. Company, "Olahraga menyediakan lift untuk menyiarkan TV pada bulan September, tetapi semua tanda masih mengarah ke streaming," Oct. 2022.

[3] L. Kristianti, "Sempat 'down', layanan Netflix kini bisa digunakan kembali," Jul. 16, 2022. https://riau.antaranews.com/berita/292185/sempat-down-layanan-netflix-kini-bisa-digunakan-kembali (accessed Aug. 30, 2022).

[4] N. Indrawati, M. T. Rendy Munadi, and D. D. Sanjoyo, "Implementation of Load Balancer in Lightweight Virtualization Using Docker for Video On Demand Service," 2019.

[5] I. K. Dewi, R. Intari, and A. Marthotillah Rizqi, "Implementasi Load Balancing dengan Algoritma Least Weight Connection Menggunakan Zevenet," 2020. [Online]. Available: https://192.168.43.10:444.

[6] S. E. Prasetyo and Y. Salimin, "Analisis Perbandingan Performa Web Server Docker Swarm dengan Kubernetes Cluster," 2021. [Online]. Available: https://journal.uib.ac.id/index.php/combines

[7] N. M. Gupta, R. Singh, S. S. Das, S. K. Choudhary, and P. G. Student, "AWS vs AZURE vs GCP: Leaders Of The Cloud Race," *International Research Journal of Modernization in Engineering Technology and Science www.irjmets.com @International Research Journal of Modernization in Engineering*, vol. 04, no. 07, pp. 3243–3250, 2022, [Online]. Available: www.irjmets.com

[8] B. A. Firdaus, V. Suryani, and S. A. Karimah, "Analisis Performansi Proses Scaling pada Kubernetes dan Docker Swarm Menggunakan Metode Horizontal Scaler," in *e-Proceeding of Engineering*, 2020, pp. 7793–7808.

[9] S. Dwiyatno, E. Rakhmat, and O. Gustiawan, "Implementasi Virtualisasi Server Berbasis Docker Container," vol. 7, no. 2, 2020.

[10] A. Hanafiah, "Implementasi Load Balancing Dengan Algoritma Penjadwalan Weighted Round Robin Dalam Mengatasi Beban Webserver," *IT Journal Research and Development*, vol. 5, no. 2, pp. 226–233, Jan. 2021, doi: 10.25299/itjrd.2021.vol5(2).5795.

[11] ivan Firmansyah, "Pemanfaatan Google Cloud dan Teknik Load Balancing untuk Optimalisasi Performa Akses Halaman Web," 2021.

[12] T. Y. Hadiwandra and F. Candra, "High Availability Server Using Raspberry Pi 4 Cluster and Docker Swarm," *IT Journal Research and Development*, vol. 6, no. 1, pp. 43–51, Jul. 2021, doi: 10.25299/itjrd.2021.vol6(1).5806.

[13] D. Revy Pryana, M. T. Rendy Munadi, and P. Kisworo, "Implementasi dan Analisis Media Server Berbasis Docker Menggunakan Protokol HLS dan MPEG-Dash," in *e-Proceeding of Engineering*, 2020, pp. 3615–3622.

[14] P. D. Andini, Rachmawati, and I. Suandi, "Analisis Pengaruh Perubahan Encoder dan Resolusi Video Terhadap Kualitas Video Live Streaming pada Jaringan Wireless Politeknik Negeri Lhokseumawe," *Jurnal Tektro*, vol. 05, no. 02, pp. 137–143, Sep. 2021.

[15] C. Cérin, T. Menouer, W. Saad, and W. Ben Abdallah, "A New Docker Swarm Scheduling Strategy," in *Proceedings - 2017 IEEE 7th International Symposium on Cloud and Service Computing, SC2 2017*, Institute of Electrical and Electronics Engineers Inc., Mar. 2018, pp. 112–117. doi: 10.1109/SC2.2017.24.

[16] S. L. P. Panjaitan, "Analisis Perbandingan Performansi Load Balancing Menggunakan Algoritma Weighted Round Robin dengan Weighted Least Connection pada Software Defined Network," *Fakultas Informatika, Universitas Telkom*, 2019.

## UCAPAN TERIMA KASIH

Terimakasih kepada pihak-pihak yang telah mendukung terlaksananya penelitian ini, khususnya kepada Tim *Jurnal Informatika POLBENG* yang telah memfasilitasi penerbitan jurnal ini.