Implementasi dan Analisis Computer Clustering System dengan Menggunakan Virtualisasi Docker

Tanjung P Kusuma[1], Dr. Ir. Rendy Munadi, M.T.[2], Danu Dwi Sanjoyo., S.T., M.T.[3]
[1,2,3]Teknik Telekomunikasi, Fakultas Teknik Elektro – Univesitas Telkom
Jln. Telekomunikasi No.1 Terusan Buah Batu Bandung 40257 Indonesia
happyshittake@gmail.com[1], rendymunadi@telkomuniversity.ac.id[2], danudwj@telkomuniversity.ac.id[3]

ABSTRAK
Sistem komputasi terdistribusi menjadi suatu kebutuhan dalam implementasi aplikasi web saat ini. Efisiensi penggunaan sumber daya perangkat keras dan tingginya pengguna aplikasi web menjadi pendorong implementasi sistem ini di arsitektur-arsitektur aplikasi web sekarang, namun kompleksnya penggunaan, dan perancangan sistem ini menjadi penghalang implementasinya.
Teknologi virtualisasi *container* menjadi solusi untuk menjalankan sistem terdistribusi yang mudah dijalankan, dikonfigurasi dan mempunyai skalabilitas tinggi. Ditambah banyaknya alat-alat menjalankan *container* pada sistem terdistribusi, menjawab masalah kompleksitas implementasi sistem terdistribusi pada aplikasi web.
Pada tulisan ini diimplementasikan sistem terdistribusi yang dapat dengan cepat menjadwalkan aplikasi baru dalam suatu *cluster*, semua oprasi dilakukan dengan *zero downtime*, serta *fault tolerance*.

Kata Kunci : *distributed computing, container virtualization, clustering, docker, docker swarm.*

ABSTRACT
Distributed computing system becomes a necessity in the implementation of web applications today. The efficiency of the use of hardware resources and the high number of web application users are driving the implementation of this system in the current web application architectures, but the complexity of use, and the design of these systems impedes implementation.
Container virtualization technology becomes the solution to run a distributed system that is easy to run, configurable and has high scalability. Plus the number of tools running container on distributed systems, addressing the complexity of implementing distributed systems in web applications.
In this paper, it has been implemented a distributed system that can quickly schedule new applications in a cluster, all operations performed with zero downtime, as well as fault tolerance.

Keyword: *distributed computing, container virtualization, clustering, docker, docker swarm*.

I. PENDAHULUAN

1.1 Latar Belakang

Pada era *cloud computing* ini dibutuhkan system komputasi teridistribusi yang dapat mengabstraksi kemampuan perangkat keras dalam menjalankan proses komputasi. Efisiensi penggunaan sumber daya perangkat keras, replikasi proses untuk *high availability*, serta permintaan untuk sistem yang lebih toleran terhadap kesalahan sistem mendorong perkembangan sistem ini.

Teknologi virtualisasi juga semakin berkembang mengikuti kebutuhan abstraksi proses komputasi. Teknik virtualisasi tradisional dirasa terlalu memakan banyak sumber daya perangkat keras untuk menjalankan proses komputasi, sehingga teknik virtualisasi berbasi *container* yang lebih ringan menjadi pilihan yang menarik. Virtualisasi *container* juga menjadi pilihan untuk menjalankan sistem terdistribusi, perusahaan internet besar seperti Google menggunakan virtualisai *container* untuk menjalankan sistem terdistribusinya.

Pada tulisan ini, penulis mengimpementasikan sistem terdistribusi berbasis virtualisasi *container* dan menganalisa bagaimana sistem tersebut mempengaruhi efisiensi dalam menjalankan proses komputasi.

1.2 Tujuan dan Manfaat

a. Melakukan implementasi layanan *clustering server* dengan menggunakan Docker.
b. Melakukan analisa dan memberikan kesimpulan dari performance sistem *clustering server* yang meliputi parameter (throughput, latency, dan CPU Utilization) menggunakan Docker.

1.3 Rumusan Masalah

a. Rumusan masalah yang diangkat dalam proposal Tugas Akhir ini adalah sebagai berikut:

b. Bagaimana membangun *computer cluster* dengan menggunakan Docker.
c. Bagaimana performance Quality of Service (QoS) dari jaringan Sistem *computer cluster* dengan menggunakam Docker

1.4. Batasan Masalah

1. Perancangan *computer cluster* dengan menggunakan Docker Swarm.
2. Menggunakan 1 buah server Master dan 3 buat server Worker
3. Tidak membahas keamanan pada sistem ini
4. Menggunakan NFS

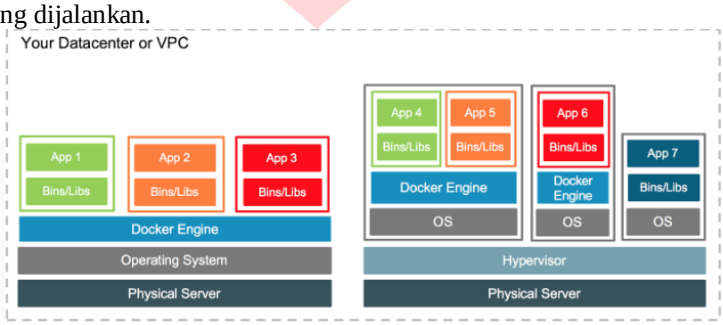
1.5. Metode Penelitian

Melakukan studi literatur dengan cara mencari, mengumpulkan dan membaca berbagai referensi dari buku, artikel, jurnal dan referensi dari internet mengenai sistem komputasi terdistribusi, teknologi virtualisasi container, dan linux sistem *administration.* Hasil studi literatur yang didapatkan akan dijadikan sebagai bahan untuk dasar teori dalam pembuatan tugas akhir ini.

II. DASAR TEORI

2.1 Sistem Komputasi Terdistribusi

Container adalah virtualisasi pada level sistem operasi dimana tiap proses atau aplikasi yang dijalankan tiap *container* memiliki kernel yang sama. Hal ini menjadi keuntungan sendiri dibandingkan virtualisasi pada level mesin (*virtual machine*), dimana *virtual machine* membutuhkan kernel sistem operasi yang berbeda-beda tiap aplikasi yang dijalankan.



Gambar 2 perbandingan *container* (kiri) dan *virtual machine* (kanan)

Teknologi *container* sendiri dipopulerkan oleh Docker, sebuah *startup* dari Silicon Valley, yang membuat standard untuk membuat *container image*, menjalankan *container*, dan mempublikasi *container image* yang telah dibuat. Dengan virtualisasi *container*, kita dapat menjaankan aplikasi di sistem operasi linux dan windows dengan konsisten terlepas bagaimanapun *environment* berjalannya *container*. Hal ini sangat membantu untuk menjaga konsistensi aplikasi dalam sistem terdistribusi.

2.2.1. Container Image

Container image adalah sekumpulan instruksi bagaimana aplikasi didalam *container* dibuat dan dijalankan.

2.2.2. Docker compose

Docker compose adalah alat untuk mengkonfigurasi bagaimana docker *image* dijalankan, dengan alat ini kita bisa mengkonfigurasi port apa yang akan kita buka, *environment* aplikasi, dll

```
web:
  build: .
  ports:
   - "5000:5000"
  volumes:
   - .:/code
  links:
   - redis
redis:
  image: redis
```

Gambar 4 contoh docker compose

2.2.3. Jaringan antar *container*

Terkadang *container-container* yang sudah berjalan perlu untuk saling berkomunikasi agar dapat menjalankan layanan dengan baik. Docker sendiri mempunyai fitur untuk membuat jaringan ini. Mode jaringan yang dapat dilakukan dalam Docker sendiri adalah :

- Bridge *Network*

Mode default pada Docker, *container* yang terhubung pada jaringan mode ini dapat saling berkomunikasi dengan menggunakan ip *address* dan *port* masing-masing *container*. *Container-containter* yang terhubung dengan mode ini, harus berada pada satu *host.*

- Overlay *Network*

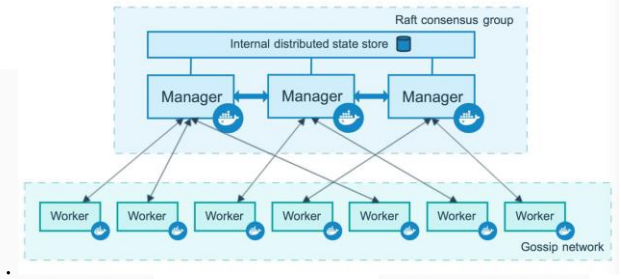
Mode ini hanya tersedia pada Docker mode swarm, mode ini merupakan mode jaringan yang cerdas dikarenakan *container* yang terhubung dapat saling berkomunikasi dengan *hostname* secara dinamis.

2.3. Container orchestration tool

Container orchestration tool adalah alat untuk mengatur jalannya *container* pada sistem terdistribusi, alat ini mengatur replikasi *container,* membagi-bagi jalannya container di berbagai komputer, *self healing process*, dll. Penulis akan berfokus pada alat yang disediakan oleh docker yaitu Docker Swarm.

2.3.1. Docker Swarm [3]

Docker swarm adalah *engine* untuk menjalankan Docker secara terdisbrusi, dengan *engine* ini beberapa *node* Docker bisa digabungkan sehingga membentuk *cluster* docker. Semua perintah dan fitur pada Docker tetap tersedia dalam mode ini, ditambah beberapa perintah untuk manajemen *cluster* Docker



Gambar 5 ilustrasi arsitektur Docker swarm

Beberapa konsep kunci yang ada pada Docker swarm adalah sebagai berikut:

- Manager node

Manager node adalah node yang bertugas untuk mengatur status *cluster*. *Node* bertugas membagikan tugas ke worker node, membuat replika proses, memperbaiki kesalahan pada *worker node*, dll. Apabila manager *node down* atau *crash*, semua service yang berjalan normal akan tetap berjalan seperti biasa, namun kita tidak bisa menjadwalkan *service* baru dan fitur *self healing* process tidak dapat dilakukan. Agar funsi manajerial *cluster* tetap berjalan, maka *node manager* dapat dibuat *cluster* lebih dari satu node, *cluster manager node* menggunakan algoritma Raft Consesus untuk memilih *leader* utama *manager node*:

- 3 *manager node* mentoleransi 1 hilangnya *manager node*
- 5 *manager node* mentoleransi 2 hilangnya *manager node*
- N *manager node* mentoleransi (N-1)/2 hilangnya *manager node*

Menambah lebih banyak *manager node* tidak berarti menambah skalabilitas atau performa *cluster*, secara umum, semakin banyak *manager node* proses manajerial *cluster* akan menjadi lebih lambat.

- Worker node

Worker node adalah *node* yang tugasnya hanya menjalankan tugas yang diberikan oleh *manajer node*. *Worker node* tidak dapat melakukan fungsi manajerial *cluster* ataupun tahu mengenai kondisi *cluster.*

- Load balancer

Docker swarm menggunakan *ingress load balancing* untuk mem*publish service* ke jaringan luar. *Ingress load balancer* ini sendiri merupakan *proxy server* yang tugasnya membuat *routing* agar *container* yang berjalan dapat diakses dari luar

- *Service* dan *task*

Service adalah sekumpulan *task* yang dijalankan pada *worker node*. *Service* merupakan struktur utama dalam *cluster* swarm dan pusat interaksi pengguna dengan *cluster*.
Task merupakan *container* yang dijalankan pada *container*.

2.4. Network File System

*Network File System (*nfs) adalah sistem yang berjalan pada sistem operasi linux memungkinkan *sebuah host* untuk mengakses *folder* atau *file* yang berada pada *host* lain, seakan-akan *file* atau *folder* tersebut *filesystem* lokal. NFS menggunakan tcp untuk berkomunikasi, walaupun udp dapat digunakan untuk sebagai protokol pengiriman, namun sangat tidak direkomendasikan untuk penggunaan skala besar. [4]

2.5. Haproxy *Load Balancer*

Haproxy adalah *proxy server* yang memiliki fitur untuk mendistribusikan trafik ke *server-server* yang ada dibelakangnya. [5] Haproxy merupakan *proxy server* yang sudah sangat teruji, dengan berbagai *benchmark* yang memngunggulkan *software* ini. Haproxy sendiri hanya dapat men-*loadbalance* protokol tcp atau htttp. Beberapa mode algoritma *loadbalancing* sudah termasuk dalam set algoritma yang bisa langsung digunakan pada Haproxy:

- Roundrobin

Merupakan mode default yang digunakan Haproxy, setiap *request* yang masuk akan diteruskan ke *backend server* secara bergilir. Algoritma pemilihan *server* ini berjalan dengan dinamis, dengan seiringnya waktu Haproxy akan mengatur bobot pemilihan sesuai dengan performa *server*.

- Static-rr

Mode ini sama dengan roundrobin namun perbedaannya mode ini berjalan secara statis dan mode ini menggunakan CPU yang lebih kecil.

- Leastconn

Server dengan koneksi trafik yang paling akan dipilih. Mode ini sangat cocok untuk jenis-jenis aplikasi yang membutuhkan sesi koneksi yang panjang, seperti LDAP, SQL, TSE, dll, namun mode ini kuran cocok untuk sesi yang pendek seperti HTTP. Mode ini berjalan dengan dinamis, yang berarti bobot pemilihan server dikonfigurasikan seiring berjalannya Haproxy.

2.6. Video On Demand over HTTP Pseudo Streaming

Video On Demand (vod) merupakan layanan yang menawarkan konten video kepada pengguna. Dibandingkan dengan layanan *live streaming*, vod tidak memberikan konten secara *realtime*, namun konten yang udah ada dapat di-*download* oleh pengguna. HTTP *pseudo streaming* sendiri merupakan teknik untuk memberikan fitur *seek video* pada pengguna, sehingga pengguna dapat menjalankan video dari tengah atau darimana saja pengguna menginginkan.

2.7. *Quality of Service*

2.7.1. Throughput

Throughput dalam jaringan telekomunikasi merupakan rata-rata pengiriman sukses daam suatu pengiriman (satuan bps). Sedangkan, sistem througput atau jumlah throughput merupakan jumlah rata-rata paket data yang sukses dikirimkan oleh semua terminal pada sebuah jaringan

Pada umumnya throughput maksimum sering dikenal sebagai througput. Throughput maksimum dari sebuah titik atau jaringan komunkasi menandakan kapasitas dari jaringannya.

2.7.2. Delay (latency)

Delay atau yang dikenal dengan istilah latency adalah waktu yang dibutuhkan paketuntuk menempuh jarak dari asal ke tujuan. Dalam implementasi, nilai delay yang dimaksud adalah lamanya waktu yang ditempuh paket dari application source ke application tujuan. Besarnya delay dapat disebabkan oleh banyak hal. Faktor yang mempengaruhi salah satu diantaranya adalah penentuan prioritas paket dari algoritma scheduling yang digunakan.

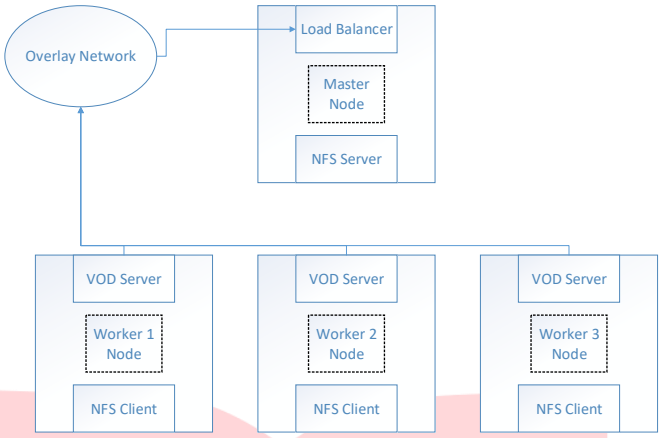
III. PERANCANAN SISTEM

3.1 Arsitektur Sistem

Arsitektur yang akan diimplementasikan tugas akhir ini adalah sebagai berikut:

- manager node
- 3 worker node

- *load balancer* external



Gambar 7- Ilustrasi arsitektur implementasi

3.2 Perancangan Perangkat Keras Perangkat keras yang akan digunakan pada tugas akhir ini adalah 5 komputer dengan spesifikasi 1 GB ram dan 1 core cpu, dengan menggunakan sistem operasi Ubuntu 16.04.

3.3 Perancangan Perangkat Lunak

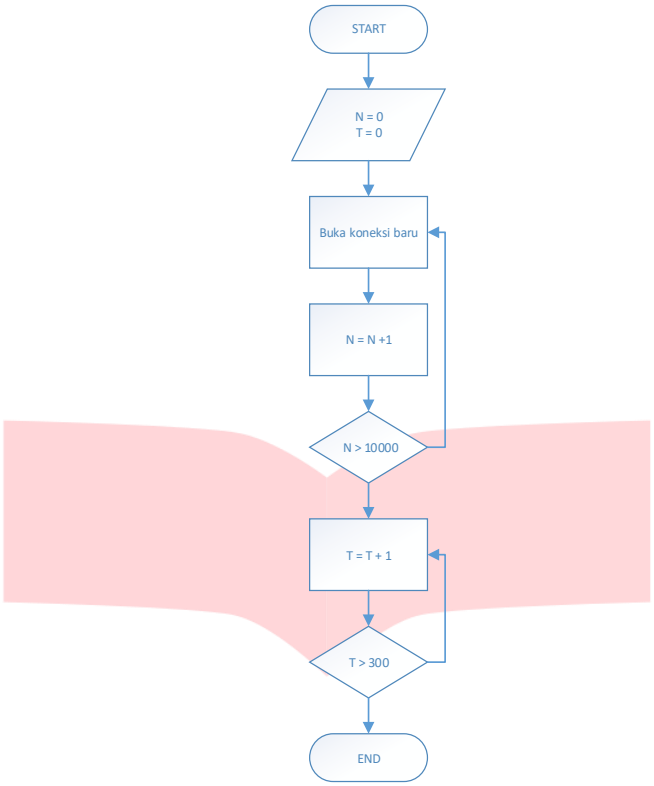
Perangkat lunak yang digunakan antara lain:

- Docker sebagai virtualisasi *container* dan clustering *container*
- HAproxy sebagai eksternal *loadbalancer*
- Sysstat sebagai alat *monitoring server*

3.4. Skenario Pengujian

Pada penilitian kali ini akan dilakukan pengujian dengan menjalankan dua layanan yaitu *video streaming*. Untuk perbandingan dengan metode virtualisasi lainnya, data pengujian akan dibandingkan dengan penilitian yang lain [6]. Penelitian tersebut menggunakan virtualisasi hypervisor dengan platform Open Nebula. Adapun parameter yang akan diambil adalah *performance Qos* (*througput*, *delay*) dan *resource usage* (CPU *usage*). Pengujian dilakukan sebanyak 3 kali kemudian diambil rata-rata keseluruhannya.

3.5. Diagram alir pengujian



IV. PENGUJIAN DAN ANALISIS

4.1. Gambaran Analisis

Pada bab ini merupakan hasil dari pengujian yang dilakukan terhadap dua layanan yang dijalankan pada *cluster*, yaitu layanan *video on demand streaming*. Parameter-parameter yang diuji adalah *latency*, CPU *utilization*, dan *throughput*. Pengujian ini dilakukan pada *server* yang memiliki spesifikasi 2 *core* CPU dan 4 GB RAM, yang pada tulisan ini disebut *server tester*. Pengujian ini dibantu oleh perangkat lunak *monitoring resource server linux* SYSTAT, yang berfungsi sebagai perangkat untuk mengambil data kondisi tiap *node cluster*.

4.2. Pengujian layanan *vod streaming*

Pengujian layanan ini dilakukan dengan cara *server tester* meminta file video yang disediakan *cluster*. Koneksi yang dibuka *server tester* sebanyak 10000 koneksi, pengujian ini dilakukan sebanyak 3 kali.

4.2.1. Layanan *vod* berjalan pada satu *node worker*

Gambar 9 - visualisasi berjalannya proses pada pengujian vod pada 1 *node worker*

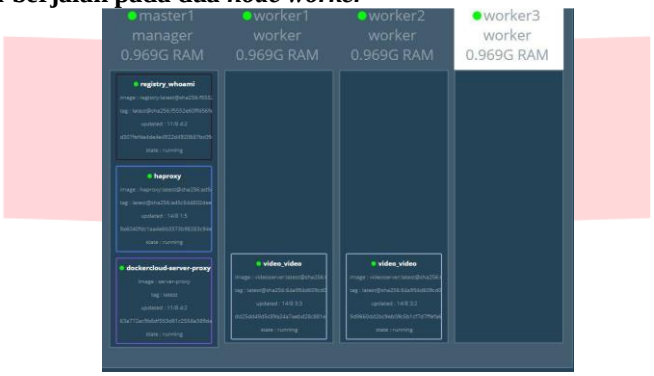
Pada skenario ini layanan *vod* hanya berjalan pada satu *node worker*, sehingga tidak terjadi proses distribusi *load*, sehingga performa layanan sendiri terbatas hanya satu *server*.

No	*Connected Request*	*Dropped Request*	*Avg Latency*	*Throughput*	*Cpu Utilization*
1	4701	5299	15.70s	25.014 MB/s	85%
2	4579	5421	14.43s	26.010 MB/s	94%
3	5130	4870	14.54s	27.807 MB/s	94.5%

Tabel 1 hasil pengujian layanan *vod streaming* pada satu worker

Dari pengujian ini terlihat rata-rata *latency* yang dialami oleh tiap *connected request* cukup besar, hal ini disebabkan layanan yang hanya berjalan pada satu *node* sehingga ketika *resource node* penuh, *request* yang belum mendapatkan balasan dari *server* terpaksa menunggu atau apabila waktu *timeout* sudah melebihi batas maka *request* terpaksa di-*drop.*

4.2.2. Layanan *vod* berjalan pada dua *node worker*



Gambar 10 - visualisasi berjalannya proses pada pengujian vod pada 2 *node worker*

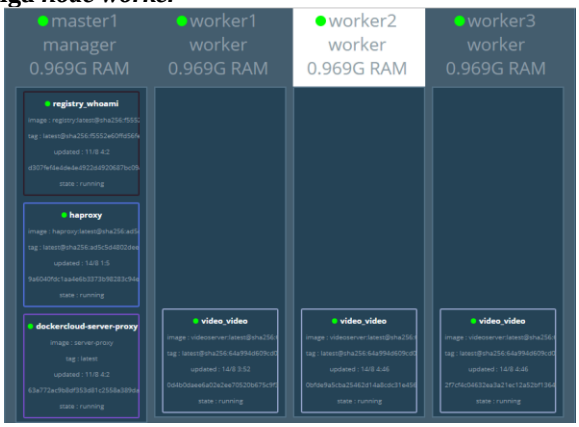
Pada skenario ini layanan *vod* berjalanan 2 *node worker*, sehingga terjadi proses distribusi proses yang menyebabkan layanan ini berjalan lebih efisien.

No	*Connected Request*	*Dropped Request*	*Avg latency*	*Throughput*	*CPU Utilization*	
					Worker 1	*Worker 2*
1	4096	5904	4.3s	23.23 MB/s	49%	67%
2	4147	5853	5.2s	26.61 MB/s	50%	63%
3	4059	5941	4.6s	27.10 MB/s	49%	65%

Tabel 2 hasil pengujian layanan *vod streaming* pada dua worker

Pada pengujian kali ini terlihat dengan menambah satu *node worker, latency* pada layanan menjadi turun sebesar 70%, dan CPU *utilization* juga turun 50%.

4.2.3. Layanan *vod* pada tiga *node worker*



Gambar 11 - visualisasi berjalannya proses pada pengujian vod pada 3 *node worker*

Pada pengujian ini layan *vod* berjalan pada semua *node worker* yang ada pada *cluster*. Pengujian ini menguji performa *cluster* dengan maksimal, dikarenakan penggunaan semua *node worker* ini.

No	Connected Request	Dropped Request	Avg Latency	Throughput	Cpu Utilization		
					Worker 1	Worker 2	Worker 3
1	3898	6102	2.8s	23.71 MB/s	30%	47%	41%
2	4113	5887	3s	23.55 MB/s	30%	48%	43%
3	3160	6840	2.1s	31.82 MB/s	31%	46%	42%

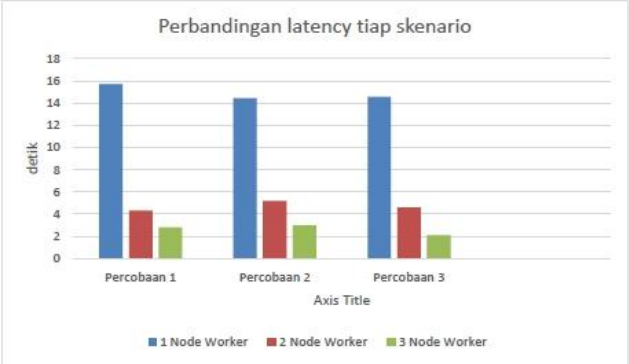
Tabel 3 hasil pengujian layanan *vod streaming* pada tiga worker

Hasil dari pengujian ini membuktikan semakin banyak *node worker* yang digunakan performa aplikasi semakin baik, hal ini dibuktikan *latency* yang semakin kecil sehingga waktu tunggu *client* yang semakin minimal. Penggunakan CPU juga semakin efisien dimana penggunakan rata-rata dibawah 50%.

4.2.4. Perbandingan skenario pengujian

- Latency

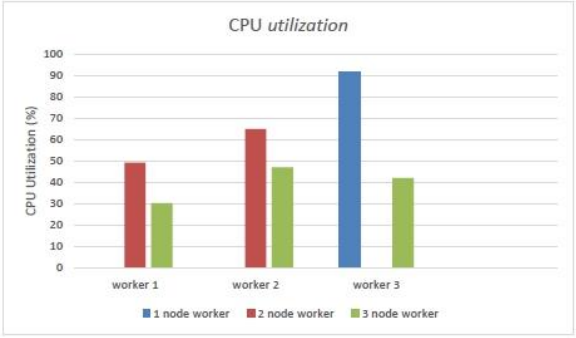
Performa aplikasi dapat diukur dari seberapa cepat *server* membalas *request client*. Semakin cepat *server* dapat merespon, waktu tunggu atau dalam istilah layanan *video streaming* (*buffering video)* semakin kecil.



Gambar 12 - perbandingan latency tiap scenario

- CPU *utilization*

Efisiensi penggunaan CPU juga menjadi hal penting dalam menjalankan aplikasi yang terdistribusi. Semakin kecil dan merata penggunaan CPU menunjukkan bahwa aplikasi tersebut berjalan dengan lebih efisien dan menggunakan resource *cluster* secara menyeluruh.



Gambar 13 - perbandingan penggunaan CPU tiap node worker tiap scenario

Terlihat dari penggunaan CPU terjadi pemerataan penggunaan tiap *node* nya.

V. PENUTUP

5.1 Kesimpulan

Dari hasil percobaan yang telah penulis lakukan dan analisa data yang ada dapat disimpulkan bahwa :

1. Dapat mengimplementasikan *computer cluster* dengan menggunakan Docker.

2. Berdasarkan hasil pengujian performansi Quality of Service (QoS) terlihat bahwa pengujian pada *computer cluster*, semakin banyak *node worker* yang digunakan untuk menjalankan layanan semakin baik

pula performa aplikasi. *Latency* pada layanan yang menggunakan 2 *node worker* terjadi penurunan sebesar 70% dibandingkan hanya menggunakan 1 *node worker*, sedanakan pada layanan yang menggunakan 3 *node worker* terjadi penurunan sebesar 80%. CPU *utilization* juga semakin efisien, pada layanan yang berjalan secara terdistribusi (lebih dari 1 *node*) penggunaan cpu tiap node berkisar 30 – 60%.

5.2 Saran

Saran yang bisa diberikan untuk penelitian selanjutnya dengan topik yang sama adalah :

1. Lakukan penelitian dengan menggunakan *software cluster orchestration* yang berbeda seprti KUBERNETES atau APACHE MESOS.

2. Lakukan penelitian pada jaringan overlay yang menghubungkan *container*

VI. DAFTAR PUSTAKA

[1] M. M. K. N. K. S. Ms.Kamble A.L, "Real Time And Distributed Computing Systems," *IOSR Journal of Computer Engineering.*

[2] Docker for the virtualization admin, 2016.

[3] V. Jurenka, "Virtualization using Docker," MASARYK UNIVERSITY, Brno, 2015.

[4]Red Hat, "Chapter9. Network FIle System," [Online]. Available: https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Storage_Administration_Guide/ch-nfs.html. [Accessed 15 8 2017].

[5] HAproxy, "HAProxy - the reliable performance TCP/HTTP load balancer," [Online]. Available: http://www.haproxy.org/#desc. [Accessed 15 8 2017].

[6] R. P. Danny, DESAIN DAN REALISASI LAYANAN PRIVATE CLOUD INFRASTRUKTUR AS A SERVICE (IAAS) MENGGUNAKAN PLATFORM OPEN NEBULA, bandung: Telkom University, 2016.