การประเมินสมรรถนะการเชื่อมต่อเครือข่ายคอนเทนเนอร์คูเบอร์เนตส์
Performance evaluation of K8s container networking

พรรักษา กมลเวชช์ (Pornraksa Kamolvage)[1] และชัยพร เขมะภาตะพันธ์ (Chaiyaporn Khemapatapan)[2]
สาขาวิชาวิศวกรรมศาสตร์คอมพิวเตอร์และโทรคมนาคม
วิทยาลัยนวัตกรรมด้านเทคโนโลยีและวิศวกรรม มหาวิทยาลัยธุรกิจบัณฑิตย์
[1]npicmy@outlook.com, [2]chaiyaporn@dpu.ac.th

บทคัดย่อ

คูเบอร์เนตส์หรือ K8s เป็นระบบจัดการประมวลผลแบบเสมือนอีกแบบที่พัฒนาโดยกูเกิ้ลกำลังได้รับความนิยมเป็นอย่างมากสำหรับการประมวลผลคลาวด์ โดยมีพื้นฐานมาจากการใช้งานและจัดการคอนเทนเนอร์ของดอคเกอร์บนโหนดประมวลผลหลายโหนดที่เชื่อมต่อเป็นคลัสเตอร์ อย่างไรก็ตามการเชื่อมต่อเครือข่ายบนคลัสเตอร์ของคอนเทนเนอร์ทำได้หลายวิธี ดังนั้นงานวิจัยนี้จึงทำการเปรียบเทียบประสิทธิภาพทางด้านเวลาในการตอบสนองที่ใช้ในการตอบกลับและอัตราการส่งผ่านข้อมูลของการให้บริการข้อมูลด้วยเว็บเซิร์ฟเวอร์ของการเชื่อมต่อเครือข่ายคอนเทนเนอร์ 3 แบบได้แก่ Flannel, Calico และ Canal จากผลการทดลองพบว่า Calico มีประสิทธิภาพดีที่สุดจากทั้ง 3 วิธี โดยเฉพาะอย่างยิ่งกรณีที่มีปริมาณทราฟฟิกเป็นจำนวนมาก สำหรับวิธี Flannel และ Canal ให้ประสิทธิภาพใกล้เคียงกัน เนื่องจากทั้ง 2 วิธีอยู่บนพื้นฐานของ VxLAN นั่นเอง อย่างไรก็ตาม Flannel มีการนำมาใช้งานที่ง่ายที่สุด

คำสำคัญ: คอนเทนเนอร์ การเชื่อมต่อเครือข่าย คูเบอร์เนตส์ ฟาแนล คาลิโค แคแนล

Abstract

Kubernetes, K8s, is a new virtualized computing orchestration developed by Google and being popular cloud computing. K8s is designed for deployment and management Docker containers on clustered computer nodes. However, there are many container networking schemes for the clustering. Thus, in this paper, performance comparison based on throughput and response time from Web server which connected via 3 various container networking: Flannel, Calico and Canal will be evaluated. From the experimental results, Calico achieved best performance especially for heavy traffic requirement. Flannel and Canal stand the similar performance because both methods used VxLAN technology. However, Flannel is most simply for implemented.

Keyword: Docker Container, Container Networking, Kubernetes, Flannel, Calico, Canal

1. บทนำ

ทุกวันนี้เทคโนโลยีการประมวลผลเสมือนโดยเฉพาะด็อกเกอร์คอนเทนเนอร์ (Docker Container) มีบทบาทในระบบประมวลผลบนอินเตอร์เน็ตหรือคลาวด์ (Cloud) มากเนื่องมาจากการใช้ทรัพยากรที่น้อยกว่า และสามารถย้ายการติดตั้งได้ง่ายและทำงานได้เร็ว การทำงานร่วมกันเป็นคลัสเตอร์ของคอนเทนเนอร์กลายเป็นปัจจัยสำคัญในการสร้างระบบให้บริการบนระบบคลาวด์ ซึ่งสามารถจัดการได้โดยใช้งาน Orchestrator เช่น Docker Swarm หรือ Kubernetes [1] ซึ่งเป็นระบบควบคุมกำกับการทำงานในการประมวลผลขนาดเล็กที่เป็นระบบเสมือนได้ อย่างไรก็ตาม Kubernetes หรือ K8s มีแนวโน้มการใช้งานอย่างกว้างขวางกว่า ในการเชื่อมโยงระหว่างโหนดคอนเทนเนอร์ของ Kubernetes สามารถเลือกใช้งานได้จากหลายวิธี [2] ซึ่งสิ่งนี้ส่งผลต่อความเร็วและเวลาในการส่งข้อมูลระหว่างคอนเทนเนอร์และจากคอนเทนเนอร์มายังผู้ใช้งาน ซึ่งโดยปกติวิธีการแบบ Flannel จะเป็นวิธีที่ถูกแนะนำให้ใช้งานโดยปริยายจาก Google ดังนั้นในงานวิจัยชิ้นนี้ได้ทำการเปรียบเทียบประสิทธิภาพทางด้านเวลาในการ

ตอบสนองที่ใช้ในการตอบกลับ (Response Time) ไปยังผู้ใช้บริการของ 3 วิธีด้วยกัน คือ Flannel, Calico และ Canal

2. ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 Kubernetes

Kubernetes เป็นเครื่องมือช่วยบริหารจัดการคอนเทนเนอร์ที่ทำงานอยู่ในระบบสภาพแวดล้อมแบบคลัสเตอร์ (Cluster) เป็นเครื่องมือโอเพ่นซอร์สที่บริษัท Google พัฒนาขึ้น ถูกนำมาใช้ในขั้นตอนการนำคอนเทนเนอร์ไปใช้งาน (Deployment) สำหรับ Kubernetes จะจัดการระบบเครือข่ายและจัดภาระการทำงานให้กับคอนเทนเนอร์ [3] โดยที่ผู้ใช้งานจะสั่งการทำงานผ่านทางโปรแกรม kubectl และเซอร์วิสต่างๆ จะทำงานอยู่บนเครื่องมาส์เตอร์ (Master Server) การทำงานของ Kubernetes มีส่วนประกอบดังนี้

2.1.1 Master Server

ส่วนควบคุมการทำงานหลักของ Kubernetes ผู้ใช้งานจะสั่งงานต่างๆ ผ่านทางเครื่องมาสเตอร์ไปยังเครื่องมินเนี่ยน (Minion Server) โดยเครื่องมาสเตอร์จะต้องมีโปรแกรมเพื่อทำงานกับ Kubernetes ดังนี้

2.1.1.1 Etcd

เป็นการให้บริการเก็บข้อมูลแบบ Key-Value ซึ่งแต่ละเครื่องเซิร์ฟเวอร์จะใช้ Etcd เป็นตัวกลางในการติดต่อสื่อสารภายในคลัสเตอร์ เช่น สภาพแวดล้อมต่างๆ ของเครื่องเซิร์ฟเวอร์ภายในคลัสเตอร์ โดยการทำงานของ Etcd จะมีการเก็บข้อมูลในรูปแบบของ HTTP/JSON เมื่อต้องการเรียกดูข้อมูลสามารถเรียกข้อมูลของค่าที่ต้องการ โดย Etcd จะทำการตอบกลับค่ามาให้

2.1.1.2 API Service Server

ส่วนนี้เป็นส่วนสำคัญของเครื่องมาสเตอร์ และเป็นส่วนที่จัดการงานต่างๆ ของเครื่องที่อยู่ภายในคลัสเตอร์ โดยจะทำการอ่านค่าสถานะต่างๆ จาก Etcd

การเรียกใช้งาน API Service Server จะสามารถเรียกใช้งานผ่าน RESTful ซึ่งเป็นข้อได้เปรียบที่ทำให้ผู้พัฒนาสามารถพัฒนาเครื่องมือขึ้นมาติดต่อ API Service Server ได้เอง

2.1.1.3 Controller Manager Server

ส่วนที่ใช้ในการจัดการงานเกี่ยวกับการสร้างคอนเทนเนอร์ โดยจะทำการอ่านค่าสถานะต่างๆ ของคอนเทนเนอร์ผ่านทาง Etcd และเป็นส่วนที่ใช้ในการจัดการลดหรือการขยายการให้บริการด้วยเช่นกัน

2.1.1.4 Scheduler Server

ส่วนในการจัดการบริหารงานในแต่ละเครื่องเซิร์ฟเวอร์ภายในคลัสเตอร์ โดยจะทำการตรวจสอบไม่ให้มีการใช้งานทรัยากรเกินกว่าทรัพยากรที่มีว่างอยู่ และเมื่อมีการสร้างคอนเทนเนอร์ขึ้นใหม่ ส่วนนี้จะเป็นส่วนที่จัดการหาเครื่องมินเนี่ยนเพื่อรองรับคอนทนเนอร์ใหม่ที่เกิดขึ้น

2.1.2 Minion Server หรือ Worker Server

เครื่องมินเนี่ยนจะทำหน้าที่รองรับงานจากเครื่องมาสเตอร์ โดยเชื่อมต่อกับเครื่องมาสเตอร์ผ่านทางเครือข่าย ซึ่งเครื่องมินเนี่ยนจะมีโปรแกรมด็อกเกอร์ทำงานอยู่ เมื่อมีการสร้างงานขึ้นมาใหม่จากเครื่องมาสเตอร์ เครื่องมินเนี่ยนจะดำเนินการสร้างด็อกเกอร์คอนเทนเนอร์ขึ้นตามที่ได้รับงานมา ซึ่งเครื่องมินเนี่ยนจะมีส่วนของการทำงานดังต่อไปนี้

2.1.2.1 Kubelet Service

ส่วนที่ติดต่อกับเครื่องมาสเตอร์เพื่อรับข้อมูลหรือรับคำสั่งต่างๆ เช่น การรับคำสั่งสร้างคอนเทนเนอร์

2.1.2.2 Proxy Service

ส่วนที่ทำหน้าที่ส่งการร้องขอการให้บริการจากด็อกเกอร์โฮสต์ไปยังด็อกเกอร์คอนเทนเนอร์ และทำการส่งผลลัพท์กับคืน ซึ่งส่วนของ Proxy Service นี้จะสามารถจัดการการให้บริการในรูปแบบการกระจายงานได้ (Load Balancing)

2.1.3 หน่วยการทำงานของ Kubernetes

การทำงานของคอนเทนเนอร์ที่ทำงานอยู่บน Kubernetes ประกอบไปด้วยหน่วยการทำงานดังนี้

2.1.3.1 Pods

ชื่อเรียกของกลุ่มคอนเทนเนอร์ ในหนึ่ง Pods สามารถมีคอนเทนเนอร์ได้ตั้งแต่ 1 ตัวขึ้นไป แต่โดยทั่วไปนิยมให้มีแค่หนึ่งตัวในหนึ่ง Pods เพื่อใช้ในการรองรับการให้บริการแอพพลิเคชั่นที่ติดตั้งในคอนเทนเนอร์ และสำหรับแอพพลิเคชั่นสามารถมีจำนวนมากกว่าหนึ่ง Pods โดยขึ้นอยู่กับความสามารถในการรองรับการให้บริการของแอพพลิเคชั่นนั้น

2.1.3.2 Services

ส่วนของการเชื่อมต่อกับคอนเทนเนอร์ โดยส่วนนี้จะทำการส่งการร้องขอใช้บริการต่างๆ ไปยังคอนเทนเนอร์ที่อยู่ใน Pods หรือเรียกอีกอย่างได้ว่า Services คือส่วนของอินเตอร์เฟสที่ใช้เชื่อมต่อระหว่างผู้ใช้บริการกับคอนเทนเนอร์ที่ให้บริการ

2.1.3.3 Replication Controllers

ส่วนจัดการเพิ่มหรือลด Pods เป็นในลักษณะของการขยายขนาดหรือลดขนาดการให้บริการ ซึ่งการขยายหรือลดจะเป็นในลักษณะรูปแบบแนวนอน (Horizontal) และหน้าที่อีกส่วนหนึ่งคือการจัดการในเรื่องรื่นของแอพพลิเคชัน (Version) ที่ทำงานในคอนเทนเนอร์ เช่น เดิมมี Pods ที่ให้บริการแอพพลิเคชั่นรุ่นที่หนึ่ง ต่อมาผู้ให้บริการต้องการนำแอพพลิเคชั่นรุ่นที่สองขึ้นมาทำงานแทน และเมื่อแอพพลิเคชั่นรุ่นที่สองสามารถทำงานได้แล้ว Replication Controllers จะทำการปิดแอพพลิเคชั่นรุ่นหนึ่งลงโดยอัตโนมัติ

2.2 Container Networking

Container Networking [4][5][6] ถูกนำมาใช้ในการแก้ไขปัญหาการเชื่อมโยงระหว่างคอนเทนเนอร์ ซึ่งมีอยู่หลายวิธีการสำหรับในงานวิจัยนี้นำเสนอวิธีการ 3 วิธี ดังนี้

2.2.1 Flannel

วิธีการหนึ่งที่ออกแบบมาสำหรับ Kubernetes เป็นวิธีที่ไม่ซับซ้อนในการติดตั้ง โดยการติดตั้ง Flannel ในแต่ละโฮสต์จะมีเอเจนต์ที่ชื่อว่า Flanneld มีหน้าที่ในการดูแลจัดสรร subnet ในแต่ละเครื่อง การทำงานของ Flannel จะเรียกใช้ Kubernetes API หรือ Etcd เพื่อเก็บค่าการตั้งค่าของเน็ตเวิร์ค จัดสรร Subnet และข้อมูลอื่นๆ เช่น Public IP-ของโฮสต์ เป็นต้น และในการส่งต่อแพ็คเก็ตจะถูกส่งออกโดยใช้กระบวนการหลายอย่างรวมถึง VxLAN

Flannel ทำหน้าที่ในการจัดการเครือข่ายในระบบเลเยอร์ 3 ระหว่างหลายๆ โหนดในคลัสเตอร์ แต่ไม่สามารถจัดการคอนเทนเนอร์ที่เชื่อมต่อกับโฮสต์ได้ ทำได้เพียงเฉพาะวิธีการรับส่งข้อมูลระหว่างโฮสต์

สำหรับ Kubernetes มองว่าแต่ละ Pods มีความเป็นเอกลักษณ์ (Unique) และมีการทำ Routable IP ในคลัสเตอร์ ข้อดีของวิธีการนี้กำจัดความซับซ้อนในการ Port Mapping ที่มาจากการแชร์ IP จากโฮสต์เดียว

2.2.2 Calico

เป็นวิธีการที่เน้นทางด้านความปลอดภัยในการเชื่อมต่อสำหรับคอนเทนเนอร์และปริมาณงานที่เกิดขึ้นในเวอร์ชวลแมชีน

Calico สร้างและจัดการเครือข่ายในระดับเลเยอร์ 3 ซึ่งจะมีการทำ Routable IP เต็มรูปแบบ สำหรับการส่งข้อมูลจะไม่ต้องใช้ IP encapsulation หรือการแปลที่อยู่ของเครือข่าย ทำให้สามารถทำงานร่วมกันได้ง่ายขึ้น ในสภาพแวดล้อมที่เป็นในลักษณะซ้อนทับ (Overlay) Calico ใช้ IP-in-IP tunneling หรือสามารถทำงานกับเครือข่ายซ้อนทับอื่นๆ ได้ เช่น Flannel

Calico มีการประยุกต์ใช้กฎทางด้านความปลอดภัย (Network Policy) กับเวอร์ชวลอินเตอร์เฟสสำหรับควบคุมคอนเทนเนอร์และเวอร์ชวลแมชีน รวมทั้งมีการใช้งานกับโฮสต์อินเตอร์เฟสสำหรับเซิร์ฟเวอร์และเวอร์ชวลแมชีน

2.2.3 Canal

เป็นวิธีการที่รวมระหว่าง Calico กับ Flannel โดยในปัจจุบัน Canal เป็นเพียงรูปแบบการติดตั้งและการตั้งค่าเพื่อให้ทำงานร่วมกันบนเครือข่ายเดียวกันได้ ซึ่งในอนาคตอาจจะมีการปรับเปลี่ยนรูปแบบรวมกับ Calico และ Flannel เพื่อลดความซับซ้อนในการติดตั้งและการตั้งค่า

2.3 งานวิจัยที่เกี่ยวข้อง

Hao Zeng และคณะ [9] ได้ทำการวัดประสิทธิภาพของ Container Network ระหว่าง Flannel, Swarm Overlay และ Calico โดยมีการวัดค่าเวลาในการตอบกลับของการ Ping, TCP Throughput และ UDP Throughput พบว่า Calico มีประสิทธิภาพสูงสุด แต่การตั้งค่าในการใช้งานมีความซับซ้อนมากกว่าตัวอื่น โดยงานวิจัยนี้ไม่ได้พิจารณาทางด้านเวลาที่ใช้ในการตอบสนอง และ Container Network โดยวิธี Calico

Bin Xie และคณะ [10] ได้ทำการตรวจสอบประสิทธิภาพระหว่าง Host Mode กับ Overlay Network Mode บนแอพพลิเคชั่นที่เป็นรูปแบบ Streaming พบว่าในด้านการทำงาน Host Mode ให้ประสิทธิภาพดีว่าประมาณ 8% แต่ Overlay Network Mode ให้การทำงานที่มีเสถียรภาพมากกว่า

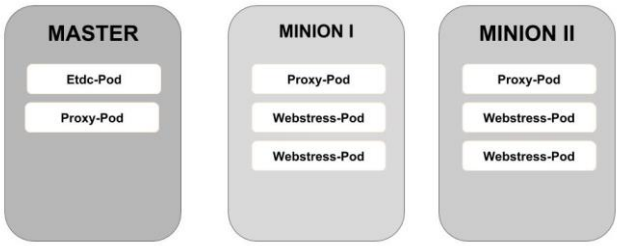
ด้านการถ่ายโอนข้อมูลระหว่างแพ็กเกจพบว่า Overlay Network Mode ให้ผลลัพท์ที่มากกว่า 50%

Jakob Struye และคณะ [11] ได้ทำการประเมินประสิทธิภาพเครือข่ายของคอนเทนเนอร์ระหว่างด็อกเกอร์ RKT และ LXC โดยทำการประเมินค่า Throughput และ Latency จากการกำหนดเน็ตเวิร์คเป็น Host, Bridge (NAT) และ Macvlan พบว่าในด้านของ Throughput ทางด้าน LXC. ให้ประสิทธิภาพดีที่สุด ทางด้านเครือข่าย Macvlan ให้ประสิทธิภาพดีที่สุด

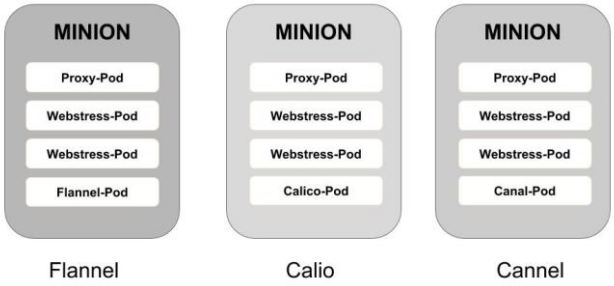
3. การดำเนินงานวิจัย

1. ดำเนินการออกแบบโครงสร้างของระบบที่ทำงานด้วย Kubernetes เพื่อให้บริการข้อมูล Web Server ด้วย Webstress บนเครื่องคอมพิวเตอร์จำนวน 3 เครื่อง โดยเป็นเครื่อง Master 1 เครื่อง และเครื่อง Minion 2 เครื่อง ซึ่งแต่ละโหนดกำหนดให้มี CPU 2 Core, Ram 2 GB และกำหนด Network Interface Card (NIC) ขนาด 100 Mbps

2. ตั้งค่า Container Network ให้กับระบบที่ได้ออกแบบไว้ โดยเมื่อทำการตั้งค่าแล้วเสร็จ จะเกิด Pods ของแต่ละ Container Network ที่แต่ละ Minion Server



ภาพที่ 1: โครงสร้างของระบบด็อกเกอร์คูเบอร์เนตส์ที่ออกแบบ



ภาพที่ 2: รูปแบบการใช้งาน Container Network

3. ติดตั้ง Webstress :ซึ่งเป็นเครื่องมือที่ใช้ในการทดสอบประสิทธิภาพการทำงานของเซิร์ฟเวอร์ที่เครื่อง Minion ทั้ง 2 เครื่อง เครื่องละ 2 Pod

4. ติดตั้ง Apache JMeter [12] ที่เครื่องโฮลต์ ซึ่งเป็นเครื่องมือที่ใช้ในการจำลองผู้ใช้บริการทำการร้องขอจากเครื่องโฮสต์ไปยังเครื่อง Master ผ่านทางโปรโตคอล http โดยจำลองให้มีการร้องขอข้อมูลเว็บจากระบบที่ทดสอบดังนี้

4.1 วินาทีที่ 1: 100 requests

4.2 วินาทีที่ 2: 200 requests

4.3 วินาทีที่ 3: 300 requests

4.4 วินาทีที่ 4: 400 requests

4.5 วินาทีที่ 5: 500 requests

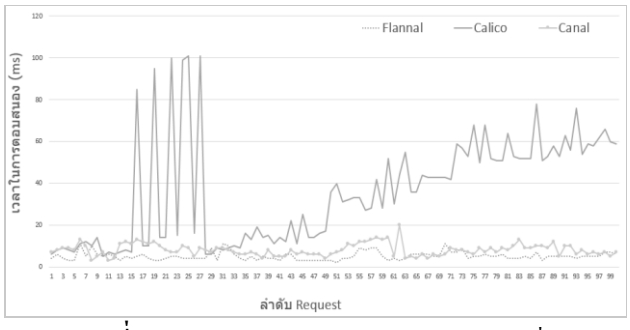
4.6 วินาทีที่ 6: 600 requests

โดยแต่ละ request ระบบจะตอบสนองด้วยการส่ง http response ขนาด 20 Kbytes จนครบทุก request

5. ประเมินประสิทธิภาพ โดยจะเป็นการประเมินประสิทธิภาพทางด้านเวลาในการตอบสนองที่ใช้ในการตอบกลับไปยังผู้ใช้บริการ และประสิทธิภาพทางด้านความเร็วในการรับและส่งข้อมูล (Upload/Download)

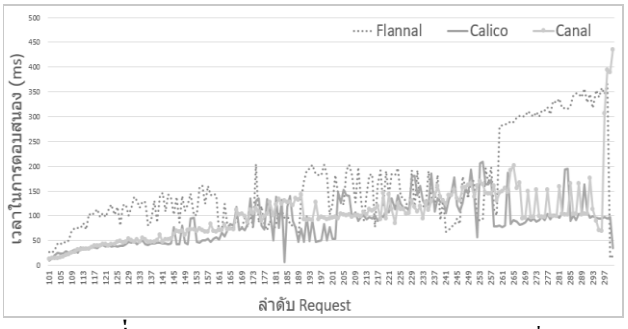
4. ผลการทดสอบและวัดประสิทธิภาพ

ผลการทดสอบประเมินประสิทธิภาพทางด้านเวลาในการตอบสนองและส่งกลับข้อมูล http response ของ Container Networking หรือรูปแบบการเชื่อมโยงเครือข่ายระหว่างคอนเทนเนอร์แต่ละประเภท ที่ได้จากการจำลองการทำงานและทดสอบในแต่ละช่วงเวลาแสดงดังรูปที่ 3 ถึง 6



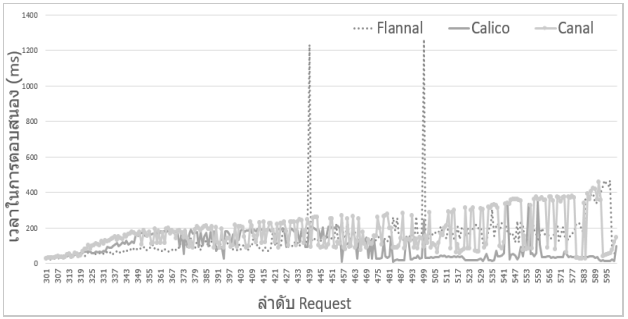
ภาพที่ 3: เวลาตอบสนอง กรณี 100 requests ในวินาทีที่ 1

จากภาพที่ 3 ผลการทดสอบเวลาที่ใช้ในการตอบสนองในกรณีมีการร้องขอจำนวน 100 requests จะพบว่าวิธีการของ Calico ให้ประสิทธิภาพทางด้านเวลาในการตอบสนองไม่ดี ใช้เวลาในการตอบสนองมากกว่าวิธี Flannel และ Canal

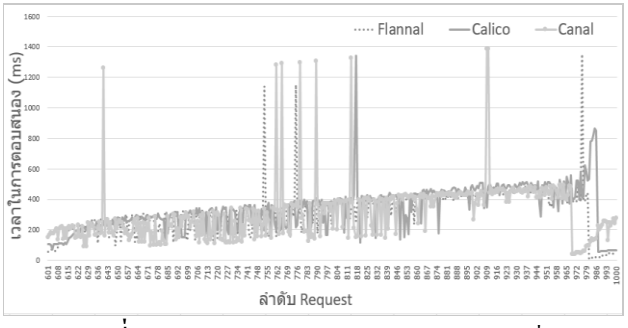


ภาพที่ 4: เวลาตอบสนอง กรณี 200 requests ในวินาทีที่ 2

จากภาพที่ 4 ผลการทดสอบเวลาที่ใช้ในการตอบสนองในกรณีมีการร้องขอจำนวน 200 requests ในวินาทีที่ 2 พบว่าวิธีการของ Calico ให้ประสิทธิภาพดีกว่าวิธีการของ Flannel และ Canal อย่างชัดเจน



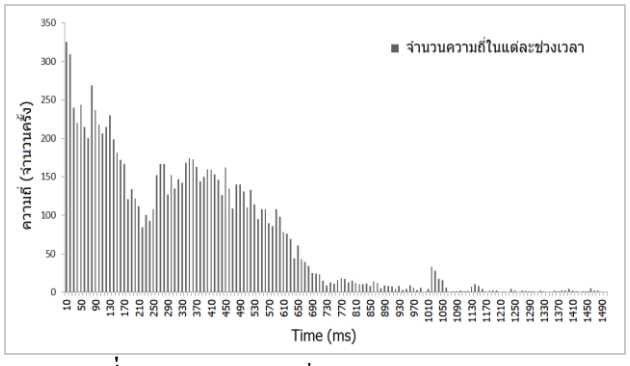
ภาพที่ 5: เวลาตอบสนอง กรณี 300 requests ในวินาทีที่ 3



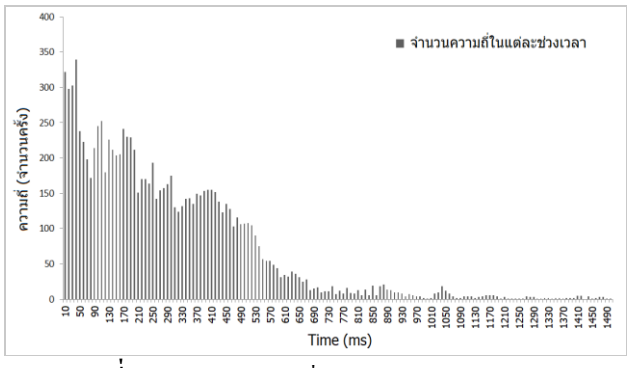
ภาพที่ 6: เวลาตอบสนอง กรณี 400 requests ในวินาทีที่ 4

จากภาพที่ 5 และ 6 ผลการทดสอบเวลาที่ใช้ในการตอบสนองในกรณีมีการร้องขอจำนวน 300 requests ในวินาทีที่ 4 และ 400 requests ในวินาทีที่ 4 ตามลำดับ ซึ่งมีทราฟิกจำนวนมากขึ้นมาก พบว่าวิธีการของ Calico ให้ประสิทธิภาพดีกว่าวิธีการของ Flannel และ Canal จึงกล่าวได้ว่าเมื่อมีการร้องขอเป็นจำนวนมากหรือ heavy traffic ตัว Calico มีแนวโน้มการตอบสนองได้เร็วกว่าวิธีการอื่น

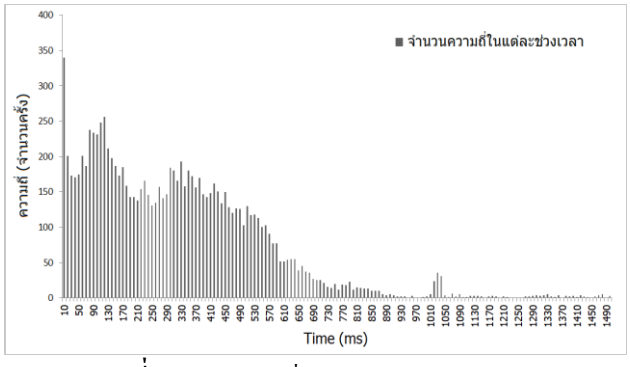
พิจารณาทางด้านการแจกแจงความถี่เวลาในการตอบสนองกับผู้ใช้บริการของการร้องขอข้อมูลทั้งหมดในแต่ละวิธี แสดงผลได้ดังภาพที่ 7 ถึง 8



ภาพที่ 7: การแจกแจงความถี่ในแต่ละช่วงเวลากรณี Flannel



ภาพที่ 8: การแจกแจงความถี่ในแต่ละช่วงเวลากรณี Calico



ภาพที่ 9: จำนวนความถี่ในแต่ละช่วงเวลากรณี Canal

จากภาพที่ 7, 8 และ 9 พบว่าวิธี Calico มีความถี่เวลาที่ใช้ในการตอบสนองที่ดีกว่าอีก 2 วิธี มีการกระจายตัวที่ค่อนข้างแคบกว่า แสดงให้เห็นถึงความมีประสิทธิภาพทางด้านเวลาที่ใช้ในการตอบสนองกับผู้ใช้งาน

อัตราการส่งผ่านข้อมูลหรือ throughput จากเว็บเซิร์ฟเวอร์ไปยังเครื่องที่ทำการร้องขอ ซึ่งคำนวณได้จาก Apache JMeter สรุปได้ดังตารางที่ 1

ตารางที่ 1: เปรียบเทียบอัตราการส่งผ่านข้อมูล (Kbyte/s)

Flannel	Calico	Canal
6,797.40	6,820.88	6,693.53

จากตารางพบว่าอัตรการส่งผ่านข้อมูลสำเร็จนั้นวิธีการ Calico จะให้ผลลัพธ์ที่ดีกว่าวิธีการอื่น สอดคล้องกับระยะเวลาที่ใช้ในการตอบสนองดังที่ได้กล่าวไปแล้ว

5. สรุป

จากผลการทดลองจะเห็นได้ว่า Container Networking ของ Kubernetes แบบ Calico มีประสิทธิภาพทางด้านเวลาที่ใช้ในการตอบสนองต่อการร้องขอดีที่สุด เมื่อเทียบกับ Flannel และ Canal เนื่องจากลักษณะของ Calico เป็นการจำลองเน็ตเวิร์คในระดับเลเยอร์ 3 และไม่มีการทำ Encapsulation หรือ Tunnel ทำให้มี Overhead น้อยกว่าอีก 2 วิธี ส่งผลให้ใช้เวลาในการตอบสนองน้อยกว่า และเป็นวิธีที่เหมาะกับกรณีที่ผู้ใช้งานมีเป็นจำนวนมาก สำหรับวิธี Flannel และ Canal ให้ผลทางด้านประสิทธิภาพคล้ายกัน เนื่องจากทั้ง 2 วิธีอยู่บนพื้นฐานของ VxLAN ซึ่งต้องทำการ Encapsulate เฟรมข้อมูลทำให้มี overhead เกิดขึ้น

สำหรับงานวิจัยในอนาคตสามารถทำการศึกษาปัจจัยทางด้านอื่นๆ ไม่ว่าจะเป็นทางด้านความปลอดภัย ความยุ่งยากในการติดตั้ง การตั้งค่า ความยืดหยุ่นในการใช้งาน หรือทดสอบในสภาพแวดล้อมอื่นๆ เพื่อให้สามารถวิเคราะห์ได้ว่าวิธีการต่างๆ นั้นเหมาะสมหรือไม่

เอกสารอ้างอิง

[1] “Comparing Orchestration Engines in Rancher, A Closer look at Docker Swarm and Kubernetes”, Rancher Labs, July, 2017.

[2] Deploy on Kubernetes. Available online at https://docs.docker.com/docker-for-windows/kubernetes/

[3] Kubernetes. What is Kubernetes? Available online at http://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

[4] C. Boettiger, “An introduction to docker for reproducible research” *Acm Sigops Operating Systems Review*, Vol 49, Issue 1, pp. 71-79, 2015.

[5] D. Vohra, *Kubernetes microservices with docker*, Apres, Canada, 2016.

[6] E. Brewer, “Kubernetes and the path to cloud native,” Proceedings of the Sixth ACM Symposium on Cloud Computing, New York, USA, August 27-29, 2015.

[7] Flannel is a network fabric for containers. Available online at https://github.com/coreos/flannel

[8] Project Calico v3.0. Available online at https://www.projectcalico.org/wp-content/uploads/2018/01/ProjectCalico.v3.datasheet.pdf

[9] Hao Zeng, “Measurement and Evaluation for Docker Container Networking” *IEEE Trans. on Cyber-Enabled Distributed Computing (CyberC)*, Nanjing, China, 12-14 Oct, 2017, pp.105-108.

[10] Bin Xie, “Docker Based Overlay Network Performance Evaluation in Large Scale Streaming System” *IEEE Trans. on Advanced Information Management Communicates (IMCEC*), Xi’an, China, 3-5 Oct, 2016, pp.366-369.

[11] Jakob Struye, “Assessing the Value of Containers for NFVs” *International Conference on Network and Service Management* (CNSM), Tokyo, Japan, 26-30 Nov, 2017, pp.1-7.

[12] Apache JMeter. Available online at https://jmeter.apache.org/