ระบบเช็คพอยต์แบบประสานสำหรับกลุ่มของเครื่องคอมพิวเตอร์เสมือนโดยใช้การประสานแบบบาร์ริเออร์และเวลาเสมือน

โดย

นายชยาวัฒน์ เพชรวิเศษ

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร
วิทยาศาสตรมหาบัณฑิต (วิทยาการคอมพิวเตอร์)
สาขาวิชาวิทยาการคอมพิวเตอร์
คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยธรรมศาสตร์
ปีการศึกษา 2560

ระบบเช็คพอยต์แบบประสานสำหรับกลุ่มของเครื่องคอมพิวเตอร์เสมือนโดยใช้การประสานแบบบาร์ริเออร์และเวลาเสมือน

โดย

นายชยาวัฒน์ เพชรวิเศษ

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร
วิทยาศาสตรมหาบัณฑิต (วิทยาการคอมพิวเตอร์)
สาขาวิชาวิทยาการคอมพิวเตอร์
คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยธรรมศาสตร์
ปีการศึกษา 2560

A COORDINATED CHECKPOINTING FOR CLUSTER OF VIRTUAL MACHINES USING BARRIER SYNCHRONIZATION AND TIME VIRTUALIZATION

BY

MR CHAYAWAT PECHWISES

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF MASTER OF SCIENCE (COMPUTER SCIENCE)

DEPARTMENT OF COMPUTER SCIENCE

FACULTY OF SCIENCE AND TECHNOLOGY

THAMMASAT UNIVERSITY

ACADEMIC YEAR 2017

มหาวิทยาลัยธรรมศาสตร์
คณะวิทยาศาสตร์และเทคโนโลยี

วิทยานิพนธ์

ของ

นายชยาวัฒน์ เพชรวิเศษ

เรื่อง

ระบบเช็คพอยต์แบบประสานสำหรับกลุ่มของเครื่องคอมพิวเตอร์เสมือนโดยใช้การประสานแบบบาร์ริเอ ร์และเวลาเสมือน

ได้รับการตรวจสอบและอนุมัติ ให้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร
วิทยาศาสตรมหาบัณฑิต (วิทยาการคอมพิวเตอร์)

เมื่อ วันที่ 16 กรกฎาคม พ.ศ. 2561

ประธานกรรมการสอบวิทยานิพนธ์ _____
(อาจารย์ ดร. ประภาพร รัตนธำรง)

กรรมการและอาจารย์ที่ปรึกษาวิทยานิพนธ์ _____
(ผู้ช่วยศาสตราจารย์ ดร. กษิดิศ ชาญเชี่ยว)

กรรมการสอบวิทยานิพนธ์ _____
(ผู้ช่วยศาสตราจารย์ ดร. เด่นดวง ประดับสุวรรณ)

กรรมการสอบวิทยานิพนธ์ _____
(ดร. ศรเทพ วรรณรัตน์)

คณบดี _____
(รองศาสตราจารย์ ดร. สมชาย ชคตระการ)

หัวข้อวิทยานิพนธ์	ระบบเช็คพอยต์แบบประสานสำหรับกลุ่มของเครื่องคอมพิวเตอร์เสมือนโดยใช้การประสานแบบบาร์ริเออร์และเวลาเสมือน
ชื่อผู้เขียน	นายชยาวัฒน์ เพชรวิเศษ
ชื่อปริญญา	วิทยาศาสตรมหาบัณฑิต (วิทยาการคอมพิวเตอร์)
สาขาวิชา/คณะ/มหาวิทยาลัย	สาขาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยธรรมศาสตร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์	ผู้ช่วยศาสตราจารย์ ดร. กษิดิศ ชาญเชี่ยว
ปีการศึกษา	2560

บทคัดย่อ

เนื่องจากความผิดพลาดสามารถเกิดขึ้นได้ทุกเมื่อ ดังนั้น Virtual Cluster หรือ Cluster ของ Virtual Machines (VMs) ที่ต้องใช้เวลาในการประมวลเป็นเวลานาน จึงจำเป็นต้องมีระบบที่มีความสามารถแบบ Fault Tolerance เพื่อให้ทนทานต่อความเสียหายต่างๆ ซึ่งวิธีการเช็คพอยต์แบบประสาน (Coordinated Checkpointing) เป็นวิธีการหนึ่งที่ให้บริการ Fault Tolerance สำหรับ Virtual Cluster ได้ แต่ในการพัฒนาวิธีการนั้นจำเป็นต้องคำนึงถึงข้อมูลที่อยู่ในระหว่างการสื่อสารบนเครือข่าย เนื่องจากถ้าหากทำการเก็บสถานะของ VM ไม่สอดคล้องกันจะทำให้เกิดสถานการณ์ Inconsistent Global State และ Domino Effect ขึ้น ทำให้ไม่สามารถนำ Checkpoint กลับมา Restart ได้อย่างถูกต้อง นอกจากนั้นยังมีปัญหา TCP Backoff ที่เกิดจากความล่าช้าในการกู้คืนการสื่อสารในรูปแบบ TCP Protocol ซึ่งปัญหานี้ส่งผลกระทบโดยตรงต่อประสิทธิภาพในการทำงานของ Virtual Cluster ดังนั้นในวิทยานิพนธ์นี้ได้นำเสนอ ระบบเช็คพอยต์แบบประสานสำหรับกลุ่มของเครื่องคอมพิวเตอร์เสมือนโดยใช้การประสานแบบแบริเออร์และเวลาเสมือน โดยการนำแนวคิด Barrier Synchronization มาใช้เพื่อเป็นการบังคับให้การ Checkpoint เกิด Consistent Global State และยังสามารถแก้ไขปัญหา TCP Backoff ด้วยการประสานงานให้ VM ทำงานได้พร้อมกันมากยิ่งขึ้น นอกจากนั้นระบบได้ใช้หลักการ Time Virtualization โดยการกำหนดเวลาเสมือน (Virtual Time) ของ VMs เพื่อลดความแตกต่างของเวลาของ VM หลังจากที่มีการทำ Checkpoint ส่งผลให้การนำ Checkpoint กลับมา Restart ทำให้ VMs ภายใน Virtual Cluster มีเวลาที่แตกต่างกันน้อยลง ผู้วิจัยคาดว่าระบบนี้สามารถส่งผลให้การทำ Checkpoint/Restart มีคุณสมบัติโปร่งใส

เนื่องจากผู้ใช้ไม่จำเป็นต้องเข้าไปแก้ไข Host OS, Guest OS และส่งผลกระทบต่อ Application น้อยที่สุด ทำให้ผู้ใช้งานสามารถนำระบบไปใช้งานกับ Application ของ Virtual Cluster ได้สะดวกมากยิ่งขึ้น และระบบยังสามารถลดปัญหาความล่าช้าของปัญหา TCP Backoff ทำให้การทำ Checkpoint/Restart ทำงานได้อย่างมีประสิทธิภาพ

คำสำคัญ: คอมพิวเตอร์เสมือน, เช็คพอยต์แบบประสาน, การประมวลผลแบบกลุ่ม, ระบบจำลองเสมือนจริง, การประมวลผลแบบกลุ่มเมฆ

Thesis Title	A Coordinated Checkpointing for Cluster of Virtual Machines using Barrier Synchronization and Time Virtualization
Author	Mr Chayawat Pechwises
Degree	Master of Science (Computer Science)
Department/Faculty/University	Department of Computer Science Faculty of Science and Technology Thammasat University
Thesis Advisor	Assistant Professor Kasidit Chanchio, Ph.D.
Academic Year	2017

ABSTRACT

A cluster of virtual machines is a common platform for running MPI applications in cloud computing environments. However, most traditional methods to provide fault tolerance to these applications are not fully transparent and require specific, checkpointing-enabled MPI software. This thesis presents A Coordinated Checkpointing for Cluster of Virtual Machines using Barrier Synchronization and Time Virtualization, namely the Virtual Cluster Checkpoint-Restart (VCCR), to perform checkpoint and restart operations at hypervisor-level. VCCR is highly transparent to MPI applications and guest OS. In VCCR, a software framework consisting of a controller and agent processes is created to perform checkpoint and restart operations for the entire cluster. The checkpoint and restart protocols of VCCR are designed based on the principles of barrier synchronization and virtual time to maintain global consistency and efficiency. We have developed a prototype of VCCR on top the QEMU-KVM software and conducted experiments using NAS Parallel Benchmark. Experimental

results confirm that VCCR can correctly and efficiently checkpoint and restart a cluster of virtual machines.

Keywords: Cluster Computing, Virtualization, Checkpointing

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงได้เป็นอย่างดี เนื่องจากได้รับความอนุเคราะห์จาก ผู้ช่วยศาสตราจารย์ ดร. กษิดิศ ชาญเชี่ยว อาจารย์ที่ปรึกษาของวิทยานิพนธ์ ที่ให้ความรู้ คำแนะนำ และตรวจสอบแก้ไขข้อบกพร่องต่างๆ เพื่อให้วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงไปได้ด้วยดี และขอขอบคุณ ดร. ประภาพร รัตรธำรง ประธานกรรมการสอบวิทยานิพนธ์ รวมถึง ผู้ช่วยศาสตราจารย์ ดร. เด่นดวง ประดับสุวรรณ กรรมการสอบวิทยานิพนธ์ และ ดร. ศรเทพ วรรณรัตน์ ผู้ทรงคุณวุฒิที่กรุณาสละเวลาและให้ข้อเสนอแนะเพื่อปรับปรุงแก้ไขวิทยานิพนธ์ให้มีความสมบูรณ์ยิ่งขึ้น ผู้วิจัยขอกราบขอบพระคุณเป็นอย่างสูงไว้ ณ โอกาสนี้ด้วย

ขอกราบขอบพระคุณ ผู้ช่วยศาสตราจารย์ ดร. กษิดิศ ชาญเชี่ยว อาจารย์ผู้จุดประกายและให้ความรู้ คำแนะนำ และคอยช่วยเหลือเพื่อให้วิทยานิพนธ์ฉบับนี้สำเร็จไปได้ด้วยดี และขอกราบขอบพระคุณ คณาจารย์ทุกท่านที่ได้ให้ความรู้ตลอดเวลาที่ศึกษา ขอขอบพระคุณเจ้าหน้าที่ประจำภาควิชาวิทยาการคอมพิวเตอร์ทุกท่านที่คอยช่วยเหลือและสนับสนุนในการทำวิทยานิพนธ์

ขอกราบขอบพระคุณพ่อและแม่ที่อนุเคราะห์และเป็นกำลังใจตลอดระยะเวลาการศึกษา และทำวิทยานิพนธ์ และขอขอบคุณเพื่อนที่ศึกษาจากมหาวิทยาลัยธรรมศาสตร์ทุกท่านที่ช่วยเหลือในการเรียนรู้ตลอดการศึกษา

นายชยาวัฒน์ เพชรวิเศษ

สารบัญ

สารบัญตาราง

สารบัญภาพ

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ใน Data Center ใหญ่ๆที่ประกอบไปด้วย Cluster Computing เป็นจำนวนมากที่มีการประมวลผลอยู่ตลอดเวลามักจะมีเหตุขัดข้องทางด้าน Hardware เกิดขึ้นได้เสมอ ยกตัวอย่างเช่นบนระบบ Cluster ของบริษัท Google ที่ประกอบไปด้วย Server จำนวน 1800 เครื่อง ซึ่งอาจมีเครื่อง Server จำนวน 1000 เครื่องที่มีข้อผิดพลาดเกิดขึ้นได้ในช่วงหนึ่งปีแรกที่มีการใช้งาน [1] จึงจำเป็นต้องมีมาตรการการป้องกันการทำงานผิดพลาดของโปรแกรมและการสูญหายของข้อมูล ด้วยการสำรองสถานะของระบบในแบบต่างๆเพื่อป้องกันไม่ให้ระบบล้มเหลวในขณะที่ให้บริการอยู่โดยที่ผู้ใช้งานได้รับผลกระทบน้อยที่สุดเท่าที่จะเป็นไปได้

Cluster Computing คือ การประมวลผลโดยการนำคอมพิวเตอร์หลายๆเครื่องมาเชื่อมต่อกันผ่านระบบเครือข่าย เพื่อนำมาทำงานในจุดประสงค์เดียวกัน เหมาะสำหรับการนำมาใช้กับงานที่ต้องใช้ปริมาณในการประมวลผลสูง หรือมีปริมาณของข้อมูลเป็นจำนวนมาก อย่างเช่น การ Render ไฟล์ในงาน Multimedia ต่างๆ และการจำลองในการทดลองทางวิทยาศาสตร์ เช่น การจำลองอะตอม หรือ การจำลองสภาพอากาศ เป็นต้น

Virtual Cluster คือ การนำเครื่อง Virtual Machine บนเครื่องคอมพิวเตอร์จริง (Host) มาเชื่อมต่อบนระบบเครือข่ายเสมือน (Virtual Network) เพื่อนำมาใช้ประมวลผลในแบบ Cluster Computing เนื่องจาก Virtual Machine มีคุณสมบัติที่สามารถบริหารทรัพยากรได้ง่ายและสามารถปรับขนาดของ Virtual Machine เพื่อนำไปใช้ให้เหมาะสมกับงานที่ต้องการได้ ดังนั้นข้อดีของ Virtual Cluster คือ มีความยืดหยุ่นสูง (Scalable) สามารถเพิ่มหรือลดปริมาณของคอมพิวเตอร์เสมือนได้ง่ายและสามารถเปลี่ยนแปลงหรือเคลื่อนย้ายการปฏิบัติงานจากเครื่องคอมพิวเตอร์เครื่องหนึ่งไปยังเครื่องอื่นได้ นอกจากนั้นเนื่องจาก Virtual Cluster เป็นซอฟต์แวร์จึงทำให้ผู้ใช้ประหยัดค่าใช้จ่ายในการติดตั้งและดูแลรักษา Hardware ได้อีกด้วย

เนื่องจากความผิดพลาดสามารถเกิดขึ้นได้ทุกเมื่อ ดังนั้นระบบคอมพิวเตอร์จึงจำเป็นต้องมีการป้องกันเหตุการณ์เหล่านั้นเพื่อให้สามารถทำงานต่อไปได้ จึงได้เกิดแนวคิด Fault Tolerance เพื่อทำให้ระบบสามารถทนทานต่อความเสียหายต่างๆที่เกิดขึ้นในระบบ โดยระบบควรจะกู้คืนการประมวลผลหรือทำงานต่อไปได้เมื่อมีความผิดพลาดเกิดขึ้น สำหรับ Virtual Cluster ที่ต้องใช้เวลาในการประมวลเป็นเวลานาน หากมีความผิดพลาดเกิดขึ้นระหว่างการทำงาน ยกตัวอย่าง

เช่น ความขัดข้องทางด้าน Hardware หรือเกิดปัญหาของการประมวลผลผิดพลาดของโปรแกรมที่ส่งผลให้โปรแกรมจบการประมวลผลแบบมีข้อผิดพลาด ถ้าไม่มีระบบ Fault Tolerance มารองรับความผิดพลาดเหล่านี้ผู้ใช้ต้องเสียเวลาในการเริ่มต้นการประมวลผลจากจุดเริ่มต้นใหม่ทั้งหมด ด้วยเหตุนี้การทำ Fault Tolerance สำหรับ Virtual Cluster เพื่อให้ระบบสามารถทนทานต่อความเสียหายและสามารถกู้คืนการทำงานได้จึงมีความจำเป็นอย่างยิ่ง

การให้บริการ Fault Tolerance ที่เป็นที่นิยมมากที่สุดวิธีหนึ่ง คือ วิธีการทำ Checkpointing ซึ่งโดยหลักการ คือ การเก็บสถานะ (Snapshot) ของการประมวลผลของเครื่องคอมพิวเตอร์เสมือนลงสู่ Disk เป็นระยะๆ สำหรับในกรณีที่เกิดความผิดพลาดขึ้น ซึ่งผู้ใช้สามารถนำสถานะของการประมวลผลของเครื่องคอมพิวเตอร์เสมือนในจุดที่ใกล้เคียงกับช่วงเวลาในการประมวลผลที่เกิดความผิดพลาดมากที่สุดไปเริ่มต้นการทำงานใหม่ (Restart) บนเครื่องคอมพิวเตอร์จริง (Host) เครื่องอื่นได้

สำหรับการทำ Checkpoint/Restart สำหรับ Virtual Cluster คือ การเก็บสถานะของเครื่องเสมือนทั้งหมดบน Virtual Cluster และข้อมูลที่อยู่บนเครือข่ายเสมือน ลงบน Disk เป็นระยะๆ ซึ่งในระหว่างที่ทำการ Checkpoint บน Virtual Cluster อาจมีการติดต่อสื่อสารกันระหว่าง VM เกิดขึ้นได้ ดังนั้นจึงต้องคำนึงถึงข้อมูล (Messages) ที่ล่องลอยอยู่บนเครือข่ายเสมือน (Virtual Network) ในช่วงเวลาดังกล่าวด้วย จากงานวิจัยเกี่ยวกับ Coordinated Checkpointing ที่ผ่านมาส่วนใหญ่พบว่า ผู้ใช้จำเป็นต้องเข้าไปแก้ไขระบบปฏิบัติการของเครื่องจริง (Host OS) และระบบปฏิบัติการของเครื่องคอมพิวเตอร์เสมือน (Guest OS) เพื่อทำให้สามารถประสานงานในการทำ Checkpoint ได้สะดวกมากยิ่งขึ้น ซึ่งทางผู้วิจัยมองว่าการเข้าไปแก้ไขในส่วนต่างๆของระบบเป็นการเพิ่มภาระให้ Programmer และผู้ใช้มากเกินไป ทำให้ระบบใช้งานยาก จึงมีแนวคิดในการประสานงานในการทำ Coordinated Checkpointing บน Virtual Cluster แบบโปร่งใส (Transparency) ซึ่งพัฒนาอยู่ในระดับซอฟต์แวร์ที่ควบคุมการประมวลผลของ VM หรือ Hypervisor จึงไม่จำเป็นต้องเข้าไปแก้ไขหรือดัดแปลง Host OS และ Guest OS

อย่างไรก็ตาม ปัญหาที่มักจะเกิดขึ้นในการทำ Checkpoint/Restart บน Virtual Cluster ในแบบโปร่งใส แบ่งปัญหาออกเป็น 3 กลุ่ม ได้แก่ ปัญหา Inconsistent Global State, ปัญหาการประสานงานในการทำ Checkpoint/Restart และ ปัญหา TCP Backoff

1.1.1 ปัญหา Inconsistent Global State และ Domino Effect

ปัญหาที่สำคัญของการ Checkpoint บน Virtual Cluster ประการหนึ่งคือ ปัญหาการจัดการ Messages ที่กำลังติดต่อสื่อสารกันระหว่าง VMs บน Virtual Cluster ในขณะที่ทำการ Checkpoint ของ VMs เรียกอีกอย่างว่าการทำ Snapshot ของ VM เนื่องจากในขณะที่มีการทำ Checkpoint อาจมี Messages ที่กำลังล่อยลอยอยู่ในระบบเครือข่าย ระบบ Checkpoint จะต้องใช้ Protocol จัดการจัดเก็บ Message เหล่านี้อย่างถูกต้องและสอดคล้องกับการเก็บสถานะของ VM ใน Virtual Cluster หากการเก็บ Messages และการเก็บสถานะของ VM ไม่สอดคล้องกันจะทำให้เกิดสถานการณ์ Inconsistent Global State และ Domino Effect ขึ้นได้ ผู้วิจัยจะบรรยายรายละเอียดของสถานการณ์เหล่านี้ในบทที่ 2

1.1.2 ปัญหาการประสานงานในการทำ Checkpoint/Restart แบบโปร่งใส บน Virtual Cluster

เนื่องจากระบบ Virtual Cluster ประกอบไปด้วย VM จำนวนมาก จึงต้องมีการออกแบบและพัฒนากระบวนการในการประสานงานแบบ Coordinate Checkpointing Protocol ในการประสานงาน VM ภายใน Virtual Cluster การพัฒนา Protocol นี้มีความท้าทายเนื่องจาก ประการที่หนึ่ง ต้องมีการออกแบบตัวประสานงานเพื่อติดต่อสื่อสารกับ VM ภายใน Virtual Cluster ประการที่สองต้องคำนึงถึงการจัดการข้อมูลที่อยู่ในระหว่างการสื่อสารบนเครือข่าย และประการที่สาม ต้องทำการประสานงานในการทำ Checkpoint โดยไม่ให้เกิดปัญหา Inconsistent Global State

1.1.3 ปัญหา TCP Backoff ในการทำ Checkpoint/Restart แบบโปร่งใส บน Virtual Cluster

TCP Backoff คือ ปัญหาความล่าช้าของการสื่อสารบน TCP Connection หลังจากที่การทำ Checkpoint ของ Virtual Cluster เสร็จ สถานการณ์ TCP Backoff เกิดจากความเหลื่อมล้ำของเวลาในการ Resume ของแต่ละ VMs ที่ไม่เท่ากัน ทำให้ VMs ผู้ส่งและ VM ผู้รับสื่อสารกันด้วย TCP Protocol ตื่นขึ้นมาในเวลาที่แตกต่างกันส่งผลให้ข้อมูลสูญหาย และเมื่อผู้ส่งไม่ได้รับ ACK กลับมาภายในเวลาที่กำหนดจะทำให้เกิด TCP Retransmission ขึ้นมาในที่สุด ดังนั้นถ้าหาก VMs ถูกทำการ Resume ในเวลาที่แตกต่างกันมากขึ้นเพียงใด

จะทำให้ VMs เสียเวลาไปกับ TCP Retransmission มากยิ่งขึ้น ซึ่งปัญหานี้เป็นปัญหาใหญ่ที่ส่งผลกับประสิทธิภาพการทำงานของระบบ Virtual Cluster

การประสานงานในการทำ Checkpoint บน Virtual Cluster ส่งผลให้เกิดปัญหา TCP Backoff เป็นอย่างมาก เนื่องจากการประสานงานดังกล่าวประกอบไปด้วยการทำ Snapshot ของ VM ทุกเครื่อง ซึ่งมักจะเกิดในเวลาที่แตกต่างกัน เนื่องจาก (มีช่องว่างของเวลาที่มีความเหลื่อมล้ำกันระหว่าง VMs อยู่เสมอ โดยเฉพาะอย่างยิ่งหากมีปริมาณของ VMs ภายใน Virtual Cluster เพิ่มมากขึ้น) สาเหตุของความเหลื่อมล้ำของเวลาในการทำ Snapshot ของ VM ได้แก่ ประการแรกคือ เวลาของ VM ทุกเครื่องมักจะไม่ใช่เวลาเดียวกันเสมอไปเนื่องจาก Clock Skew (Timing Skew) ประการที่สอง เนื่องจากผู้ประสานงานต้องใช้เวลาในการสื่อสารกับ VM ด้วยตัวเองทั้งหมด ดังนั้นคำสั่งในการทำ Snapshot จากโปรแกรมผู้ประสานงานอาจส่งไปถึง VM แต่ละเครื่องในเวลาที่แตกต่างกัน และประการที่สาม (ภายใต้ข้อกำหนดว่า เวลาของ VM จะหยุดเดินในขณะที่ VM นั้นทำ Snapshot) การที่ VM แต่ละเครื่องมีขนาดที่แตกต่างกันทำให้ระบบใช้เวลาในการทำ Snapshot ของ VM แตกต่างกัน ส่งผลให้เมื่อ VM ทำการ Resume จากการทำ Snapshot ทำให้เวลาของ VM แต่ละเครื่องอาจมีความแตกต่างกันมากขึ้น

ทางผู้วิจัยจึงนำเสนอ Checkpoint/Restart Protocol ในการทำ Coordinated Checkpointing บน Virtual Cluster ที่ได้นำแนวคิดการประสานงานแบบ Barrier Synchronization มาประยุกต์ใช้ในขั้นตอนการทำ Checkpoint/Restart โดยประสานงานให้ VMs ทุกเครื่องต้องหยุดรอชั่วขณะหนึ่ง เพื่อรอให้ VMs ทุกเครื่องมีสถานะพร้อมทำงานพร้อมกันเพื่อเป็นการบังคับให้ช่วงเวลาดังกล่าวเกิดเหตุการณ์ Consistent Global State ทำให้สามารถรับประกันได้ว่าจะไม่เกิดเหตุการณ์ Inconsistent Global State ณ จุดใดๆ บน Timeline และนอกจากนั้นผู้วิจัยได้นำแนวคิดของการจำลองเวลาของ Virtual Cluster เรียกว่า Virtual Time ซึ่งเป็นเวลาของตัวเองโดยจะแยกออกจากเวลาของ Host OS อย่างสิ้นเชิง เพื่อให้การควบคุมเวลาของส่วนประกอบต่างๆภายในระบบสัมพันธ์กัน ทำให้การ Restart จากจุด Checkpoint ที่ถูกหยุดด้วย Virtual Time นั้น สามารถนำสถานะของ Network ในแต่ละ VM ที่มีอยู่กลับมาใช้พร้อมกับ Snapshot ของ VM ได้ทันที โดยให้มีผลต่อความแตกต่างของเวลาที่เป็นสาเหตุทำให้เกิดปัญหา TCP Backoff น้อยที่สุดเท่าที่จะเป็นไปได้

จากแนวคิดดังกล่าว ผู้วิจัยเสนอว่าจะออกแบบและพัฒนาระบบ Coordinated Checkpointing Framework ซึ่งประกอบด้วย ผู้ประสานงาน (Coordinator) และผู้ดำเนินงาน (Agents) เป็นส่วนประกอบหลักที่นำมาช่วยในการประสานงานในการทำ Checkpoint/Restart บน Virtual Cluster และมีการนำ OpenVSwitch ซึ่งเป็น Software Defined Network เพื่อช่วยในการควบคุมข้อมูลในระหว่างการสื่อสารบนเครือข่าย และสร้าง Virtual Network เพื่อใช้เชื่อมต่อ VM ภายใน Virtual Cluster บน Host ต่างเครื่องกัน ผู้วิจัยจะพัฒนา Protocol เพื่อควบคุมการทำงานของ Coordinate Checkpointing Framework ได้แก่ Checkpoint Protocol และ Restart Protocol นอกจากนี้ ผู้วิจัยจะทดสอบ Coordinate Checkpoint Framework กับโปรแกรม Benchmarks ที่มีการสื่อสารระหว่าง VM ใน Virtual Cluster ด้วย TCP Protocol และ NAS Parallel Benchmark

1.2 วัตถุประสงค์

1.2.1 เพื่อศึกษาการทำงานของ Software Defined Network และ Virtual Network ภายใน Virtual Cluster

1.2.2 เพื่อพัฒนา Coordinated Checkpointing Framework และ Components ต่างๆ เช่น Coordinator และ Agents

1.2.3 เพื่อศึกษาและพัฒนา Checkpoint/Restart Protocol ในการทำ Coordinated Checkpointing แบบโปร่งใส บน Virtual Cluster

1.2.4 ทดสอบความถูกต้อง (Validation) ของการ Checkpoint/Restart บน Virtual Cluster

1.2.5 เพื่อวัดประสิทธิภาพของการทำงานของการทำ Coordinated Checkpointing Framework ที่พัฒนาขึ้นโดยใช้ Benchmark โปรแกรมที่มีการสื่อสารระหว่าง VM ใน Virtual Cluster ด้วย TCP Protocol

1.2.6 เพื่อวัดประสิทธิภาพของการทำงานของ Coordinated Checkpointing Framework ที่พัฒนาขึ้นโดยใช้ NAS Parallel Benchmark

1.3. สมมติฐานของงานวิจัย

1.3.1 ข้อกำหนดเบื้องต้น (Assumptions)

(1) กำหนดให้เวลาของแต่ละ VM มีเวลาเป็นของตัวเอง เรียกว่า “Virtual Time” และต้องแตกต่างจากเวลาของ Host

(2) กำหนดให้เมื่อมีการทำ Snapshot แล้ว Virtual Time จะต้องหยุดเดิน

(3) เมื่อ VM หยุดทำงาน แล้ว Virtual Time จะต้องหยุดเดิน

(4) เมื่อ VM ถูกทำการ Resume หรือ Restart การทำงาน แล้ว Virtual Time จะเดินต่อ

1.3.2 สมมติฐาน (Hypothesis)

(1) การทำ Barrier Synchronization ใน Virtual Cluster สามารถป้องกันการเกิดปัญหา Inconsistent Global State ได้

(2) Checkpoint Protocol สามารถจัดการการทำ Coordinated Checkpoint บนระบบ Virtual Cluster ได้อย่างถูกต้อง

(3) Restart Protocol สามารถจัดการการทำ Restart ของ Virtual Cluster จาก Checkpoint ได้อย่างถูกต้อง

(4) Checkpoint Protocol สามารถลดการเสียเวลาของ TCP Protocol อันเนื่องมาจาก TCP Backoff ได้อย่างมีประสิทธิภาพ

(5) Restart Protocol สามารถลดการเสียเวลาในการทำงานของ TCP Protocol อันเนื่องมาจาก TCP Backoff ได้อย่างมีประสิทธิภาพ

1.4 ขอบเขตของงานวิจัย

1.4.1 ระบบยังไม่สามารถดักจับ TCP Messages ที่ล่องลอยอยู่บน Virtual Network ในระหว่างการทำ Checkpoint ได้

1.4.2 ระบบยังไม่สามารถดักจับ UDP Messages ที่ล่องลอยอยู่บน Virtual Network ในระหว่างการทำ Checkpoint ได้

1.4.3 ระบบยังไม่สามารถทำ Live Checkpointing ได้

1.5 ข้อจำกัดของงานวิจัย

สำหรับสภาพแวดล้อมของงานวิจัยจะทำการทดลองโดยใช้เครื่อง Server จริงจำนวน 2 เครื่อง โดยประกอบไปด้วย Virtual Cluster ที่รันอยู่ในเวลาของตัวเองที่เป็นโลกส่วนตัวโดยแยกเป็นอิสระออกจากเวลาจริง (Actual Time) ที่เรียกว่า Virtual Time

1.5.1 ข้อจำกัดทางด้าน Hardware

(1) Host Server จำนวน 2 เครื่อง

(2) Network Switch จำนวน 1 เครื่อง ความเร็ว 10Gbps และ 100Mbps

1.5.2 ข้อจำกัดทางด้าน Hardware

(1) QEMU KVM: ระบบจะ Implement เพื่อใช้งานร่วมกับ QEMU-KVM Hypervisor เท่านั้น

(2) Host/Guest OS: ระบบจะใช้ Ubuntu OS version 14.04.5 TLS

(3) Software Defined Network: ระบบจะใช้ OpenVSwitch ในการทำ Virtual Network ภายในระบบ

(4) Python 3.5: ระบบ Framework จะพัฒนาบน Python 3.5 บน Ubuntu OS สามารถสนับสนุนเฉพาะ API ที่สามารถรองรับกับ Version 3.5 เท่านั้น

1.6 ประโยชน์ที่คาดว่าจะได้รับ

1.6.1 ระบบสามารถป้องกันการเกิด Inconsistent Global State และเหตุการณ์ The Domino Effect ในการทำ Checkpoint/Restart แบบโปร่งใส บน Virtual Cluster ได้

1.6.2 ระบบสามารถลดระยะเวลาการเหลื่อมล้ำของ VM ในระหว่างการทำ Checkpoint/Restart แบบโปร่งใส บน Virtual Cluster ได้มากยิ่งขึ้น

1.6.3 ระบบสามารถลดระยะเวลาการเกิด TCP Backoff จากการทำ Checkpoint/Restart แบบโปร่งใส บน Virtual Cluster ได้มากยิ่งขึ้น

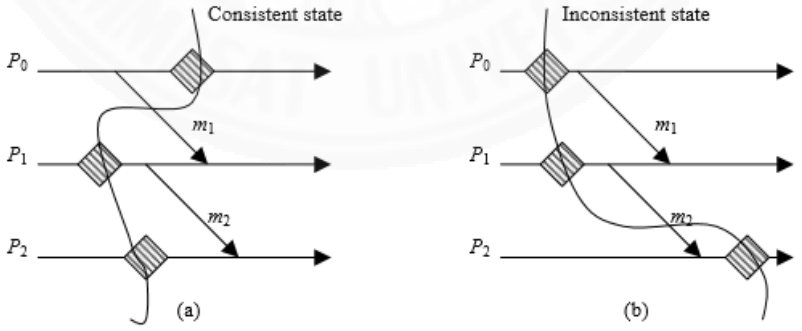
1.6.4 ระบบสามารถกลับมาทำงานต่อได้อย่างถูกต้อง หลังจากมีการทำ Checkpoint/Restart แบบโปร่งใส บน Virtual Cluster

บทที่ 2
วรรณกรรมและงานวิจัยที่เกี่ยวข้อง

2.1 ความรู้พื้นฐานและทฤษฎีที่เกี่ยวข้อง

2.1.1 Global State

Global State [2] คือ state ของ processes ที่ทำงานแบบขนานและมีการติดต่อสื่อสารกันในช่วงเวลาหนึ่ง เมื่อมีการ checkpoint เกิดขึ้นในระหว่างที่ process มีการติดต่อสื่อสารกัน อาจทำให้เกิด global state ที่ไม่สามารถใช้งานได้ ที่เรียกว่า Inconsistent Global State ซึ่งหมายถึงเกิดเหตุการณ์ที่ไม่สอดคล้องกับความเป็นจริง เช่น ระบบได้ทำการ checkpoint ที่ P1 และ P2 ซึ่งในระหว่างนั้น P1 ได้ทำการส่ง message ไปยัง P2 โดยที่ P1 ได้ถูก checkpoint เสร็จสิ้นก่อนที่จะทำการส่ง message บนจุด C(1,0) และในขณะเดียวกัน P2 ได้รับ message ก่อนที่จะถูก checkpoint ที่จุด C(2,0) หลังจากนั้น P2 เกิดข้อผิดพลาดทำให้ไม่สามารถทำงานต่อได้ จึงจำเป็นต้อง Roll-back กลับไปยัง C(1,0) และ C(2,0) เพื่อย้อนกลับไปทำงานต่อจากเดิม ซึ่งในกรณีดังกล่าว จะทำให้เกิด Inconsistent Global State เนื่องจาก P2 นั้นได้รับ message ที่ P1 ยังไม่ได้ส่งออกไปซึ่งไม่สอดคล้องกับความเป็นจริง ทำให้ไม่สามารถใช้งาน Global state นี้ได้



ภาพที่ 1.1 (a) Consistent Global State Cut และ (b) Inconsistent Global State Cut

2.1.2 Stable Storage

โดยปกติแล้วเมื่อระบบทำการ checkpoint จะต้องนำไปเก็บไว้ที่ปลอดภัยและสามารถคงทนต่อความผิดพลาดได้ ในความเป็นจริงแล้ว Stable Storage เป็นเพียงนามธรรมที่ใช้เรียกสถานที่เก็บ checkpoint file หรือข้อมูลสำคัญที่เกี่ยวข้องต่อการกู้คืนระบบ เช่น event log files หรือ messages ที่ถูกส่งระหว่าง processes เป็นต้น ซึ่งข้อมูลเหล่านี้ถูกนำไปเก็บบน disk storage อีกทอดหนึ่ง โดย Stable Storage นั้นมีเทคนิคการนำไปใช้อยู่หลายรูปแบบ เช่น ในระบบที่ใช้กับ Single failure จะให้เก็บข้อมูลลงบน Volatile memory (Borg et al. 1989; Johnson และ Zwaenepoel 1987) [3] ส่วนระบบที่ต้องใช้กับ transient failures จะเก็บไว้บน local disk ของ host สำหรับระบบที่เป็น non-transient failures จะต้องเก็บข้อมูลไว้ที่ใดๆ โดยต้องไม่อยู่บน host ที่รันงานอยู่

2.1.3 Garbage Collection

เนื่องจากทุกครั้งที่เราทำการ checkpoint เราไม่สามารถรู้ได้ว่าทุกไฟล์นั้นสามารถนำไปใช้กู้คืนระบบได้โดยที่ไม่เกิด Inconsistent Global State ซึ่งจะกลายเป็น Checkpoint ที่ไม่เป็นประโยชน์และทำให้เกิด Overhead และสิ้นเปลืองทรัพยากรอย่างมาก ดั้งนั้นจึงมีการคิดค้น Algorithm เกี่ยวกับการลดปริมาณ checkpoint ที่ไม่เป็นประโยชน์เหล่านี้ โดยการค้นหา Consistent set ของ checkpoint ที่ใช้งานล่าสุด ที่เรียกว่า Recovery Line (Randell 1975) [3] และทำการลบข้อมูลที่เกิดก่อนเหตุการณ์นี้ทั้งหมด ทำให้ลดปริมาณ checkpoint ที่ไม่เป็นประโยชน์ออกไปได้

2.1.4 Checkpoint-Based Rollback Recovery

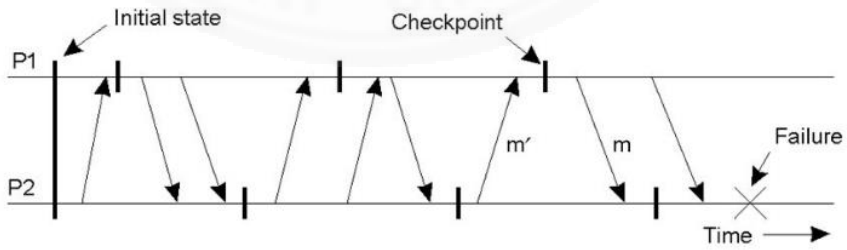
จากการศึกษาของ (Elnozahy, Alvisi, Wang, & Johnson, 2002, p.7) [3] ได้ศึกษาเทคนิคการค้นหา Consistent Global State ด้วยการกู้คืนระบบจากการ checkpoint ซึ่งได้จำแนกเทคนิคออกเป็น 3 ประเภท ได้แก่ Uncoordinated Checkpoint, Coordinated Checkpoint และ Communication-induced Checkpoint

2.1.4.1 Uncoordinated Checkpoint

เป็นเทคนิคการ checkpoint โดยที่แต่ละ process แยกกันทำ checkpoint อย่างเป็นอิสระ โดยไม่คำนึงถึงการสื่อสารระหว่าง process แต่เทคนิคดังกล่าวอาจจะทำให้เกิดการ checkpoint ที่ไม่เป็นประโยชน์จำนวนมากและมีโอกาสเกิดปัญหา Inconsistent Global State สูง อาจเป็นสาเหตุทำให้เกิด The Domino Effect ตามมาได้

The Domino Effect

สาเหตุที่ทำให้เกิด The Domino Effect (Randell 1975) [3] เนื่องจากระบบไม่สามารถทำการ Rollback กลับไปยัง Global State ที่ประกอบไปด้วย Consistent Global State ทั้งหมดได้ จนสุดท้ายต้องกลับไปเริ่มต้นระบบที่ Initial State ใหม่ทั้งหมด จากภาพที่ 2.1 แสดงให้เห็นถึงเหตุการณ์ที่ทำให้เกิด Domino Effect กล่าวคือ P1 และ P2 ได้มีการติดต่อสื่อสารกันและทำการ checkpoint เป็นระยะๆ สุดท้าย P2 เกิดปัญหาทำให้ต้อง rollback กลับไปยัง checkpoint ล่าสุด C(1,3) และ C(2,3) แต่เมื่อกลับไปดูที่ checkpoint ล่าสุดของ P2 (C(2,3)) ปรากฏว่าเกิด Inconsistent Global State เนื่องจาก P2 ได้รับ m ก่อนที่ P1 จะทำการส่ง ทำให้ P2 ต้องย้อนกลับไปยัง C(2,2) เพื่อไม่ให้เกิด Inconsistent Global State ในขณะเดียวกัน ในจุด C(1,3) ได้เกิด Inconsistent Global State เช่นเดียวกัน เนื่องจาก P1 ได้รับ m’ ก่อนที่ P2 จะทำการส่ง ดังนั้นจึงต้องย้อนกลับไปใช้จุด checkpoint ก่อนหน้า จากภาพจะเห็นได้ว่า เมื่อมีการย้อน checkpoint กลับไปทุกครั้งจะเกิด Inconsistent Global State เกิดขึ้นทั้งหมด สุดท้ายจึงต้องกลับไปเริ่มต้นใหม่ที่จุด Initial State



ภาพที่ 2.1 เหตุการณ์ Domino Effect

2.1.4.2 Coordinated Checkpoint

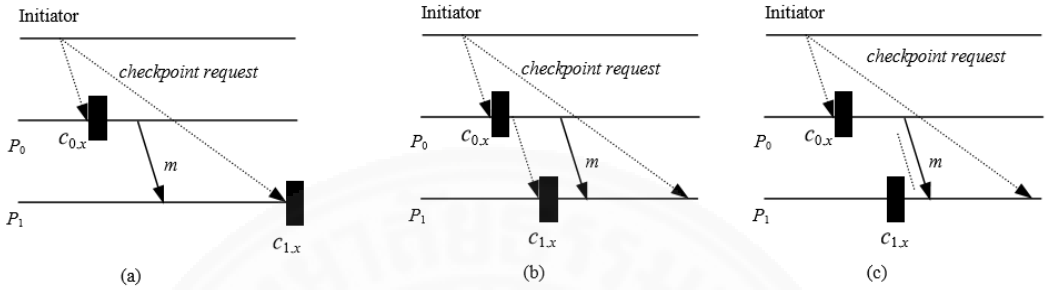
จากการศึกษาของ Chandy และ Lamport (1985) [3] ทำให้วิธีการนี้สามารถป้องกันการเกิด Domino Effect ได้โดยใช้ Process ที่ทำหน้าที่คอยประสานงานเพื่อตรวจสอบ Consistent Global State ในการทำ Checkpoint โดยเฉพาะ ในเทคนิคนี้แต่ละ Process จะทำการกู้คืนจาก checkpoint ล่าสุดของตัวมันเองเท่านั้น ซึ่ง Permanent checkpoint ของแต่ละ Process จะมีเพียงแค่อันเดียวเท่านั้นและจะถูกเก็บไว้บน Stable Storage อย่างปลอดภัย แต่วิธีนี้จะมีข้อเสียคือเกิด Overhead สูงและทำให้เกิดความล่าช้าในการ Checkpoint มากขึ้น จึงจำเป็นต้องมีการทำ Garbage Collection เพื่อลดข้อมูลที่ไม่ใช้งานออกไปด้วย

ด้วยวิธีที่ตรงไปตรงมาของ Tamir และ Sequin 1984 [3] ได้นำเสนอให้มีการปิดกั้นการสื่อสารระหว่างการทำ Checkpoint ซึ่งเป็นเทคนิคการ checkpoint โดยมี Coordinator เป็นผู้ประสานงานในการทำ Checkpoint โดยจะทำหน้าที่กระจาย Request message ไปยัง Process ทั้งหมดเพื่อขอทำ Checkpoint หลังจากนั้น Process จะหยุดการทำงานและ flush ช่องการสื่อสารทั้งหมดแล้วส่ง Acknowledgement กลับไป เมื่อ Coordinator ได้รับ ACK ครบทั้งหมดแล้วก็จะทำการส่ง Message ไปบอก Process ทั้งหมดว่าการ Checkpoint นั้นเสร็จสมบูรณ์แล้ว ซึ่ง Checkpoint ดังกล่าวจะเป็นเพียง Checkpoint ชั่วคราวเท่านั้น หลังจากนั้นแต่ละ Process จะทำการลบ checkpoint เดิมที่เก็บไว้ แล้วนำ Checkpoint ชั่วคราวไปเก็บไว้แทน สุดท้าย Process ทั้งหมดจะเริ่มต้นการทำงานอย่างอิสระและทำการสื่อสารกันตามปกติ ซึ่งการทำงานลักษณะนี้จะทำให้เกิด Overhead ค่อนข้างสูง ดังนั้นจึงมีอีกวิธีการหนึ่งที่ปรับปรุงในการลด Overhead เหล่านั้น คือ Non-blocking Checkpoint Coordination (Elnozahy et al. 1992) [3]

2.1.4.3 Non-blocking Checkpoint Coordination

จากปัญหาดังกล่าวจึงได้มีการพัฒนาวิธีแบบการไม่ปิดกั้นการสื่อสารขึ้นมา แต่การเปิดช่องการสื่อสารในขณะทำการ checkpoint อาจจะทำให้เกิด Checkpoint Inconsistency ขึ้นมาได้ ดังภาพที่ 2.2 (a) จึงมีการนำแนวคิด Distributed snapshot (Chandy และ Lamport 1985) [3] นำมาใช้ ดังภาพที่ 2.2 (b) คือ Initiator ทำการ Broadcast Checkpoint Request ที่เรียกว่า Markers ไปยังทุกๆ Process หลังจากนั้นเมื่อมี Process ใดได้รับ Marker อันแรกแล้ว ก็จะทำการแทรก Marker ที่ได้รับแล้ว Broadcast ไปยัง Process อื่นๆก่อนที่จะทำการส่ง Message m จาก Application แต่ Protocol นี้จะสามารถทำงานได้ถูกต้องก็ต่อเมื่อช่องทางการสื่อสารเป็น FIFO เท่านั้น ซึ่งถ้าหากช่องทางการสื่อสารเป็นแบบ non-FIFO จะไม่สามารถทำวิธีดังกล่าวได้ จึงมี

การศึกษาเพิ่มเติมโดยใช้วิธี Piggybacked (Lai และ Yang 1987) [3] เพื่อให้สามารถนำวิธีดังกล่าวไปใช้กับช่องทางการสื่อสารที่เป็น non-FIFO ได้ ดังภาพ (c) โดยนำ Marker ที่ P0 ได้รับจาก Initiator ส่งไปพร้อมกับ Message m เมื่อ P1 ได้รับ Marker แล้วจึงทำการ Checkpoint ก่อนที่จะรับ m เพื่อป้องกันไม่ให้เกิด Checkpoint Inconsistency

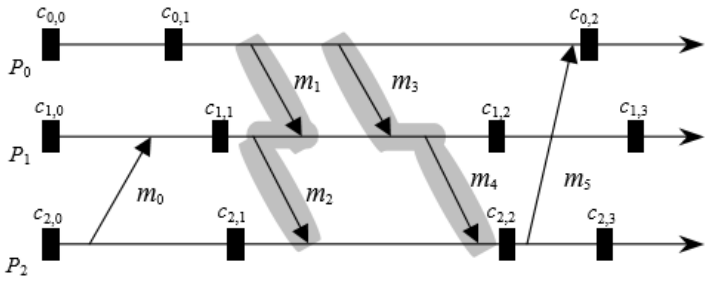


ภาพที่ 2.2 การทำงานของ Coordinate Checkpointing

2.1.4.4 Communication-induced Checkpoint

เพื่อหลีกเลี่ยง Domino Effect โดยไม่จำเป็นต้องให้ Process ติดต่อสื่อสารกัน จึงมีการพัฒนา Communication-induced Checkpoint protocol (CIC) ขึ้นมา ซึ่งเทคนิคนี้จะประกอบไปด้วย Checkpoint อยู่สองประเภทคือ Local checkpoint และ Forced checkpoint โดยที่ Local checkpoint จะเป็นการทำ Checkpoint อย่างอิสระในแต่ละ Process ส่วน Forced checkpoint จะถูกทำเพื่อป้องกันการเกิด Useless checkpoint โดยรับประกันว่าจะต้องเป็นไปตามกลไกลของ Recovery line เนื่องจาก Useless checkpoint จะไม่มีทางเป็นส่วนหนึ่งของ Consistent Global State ที่เป็นสาเหตุทำให้เกิด Overhead และสิ้นเปลืองทรัพยากร

ในการทำงานของ Protocol นี้จะไม่ใช้การติดต่อสื่อสารระหว่าง Process หรือใช้ Message พิเศษแบบที่มีการทำ Piggy-backed บน Coordinate Checkpointing สำหรับการตัดสินใจทำ Force checkpoint ซึ่งจะขึ้นอยู่กับข้อมูลของการสื่อสารกันระหว่าง Process ของ Receiver โดยจะทำการคำนวณหารูปแบบที่ทำให้เกิด Useless checkpoint และตัดสินใจทำ Force checkpoint เพื่อทำลายรูปแบบดังกล่าว ซึ่งรูปแบบการตัดสินใจนี้ถูกพัฒนาโดยมีพื้นฐานมาจาก Z-Path และ Z-Circle



ภาพที่ 2.3 Z-Part และ Z-Circle

Z-Path (Zigzag path) คือ เส้นทางของ Messages ที่เชื่อมต่อกันระหว่าง Checkpoint สองจุด จากภาพ 2.3 จะเห็นได้ว่า มีการส่ง Message [m1, m2] และ [m3, m4] จาก C(0,1) ไปยัง C(2,2) ทำให้กลายเป็น Z-Path ระหว่าง C(0,1) และ C(2,2)

Z-Circle คือ Z-Part ที่มีจุดเริ่มต้นและจุดสิ้นสุดที่ Checkpoint บน Process เดียวกัน ซึ่ง Netzer และ Xu [3] ทำการพิสูจน์แล้วว่า Checkpoint ใดๆนั้นจะเป็น Useless Checkpoint ก็ต่อเมื่อ Checkpoint นั้นเป็นส่วนหนึ่งของ Z-Circle ดังนั้นหากต้องการรับประกันว่าจะไม่ให้เกิด Useless Checkpoint คือ จะต้องไม่มี Z-part ใดกลายเป็น Z-Circle จากภาพที่ 2.3 จะเห็นได้ว่า C(2,2) วิ่งอยู่บน Z-Part [m5, m3, m4] ซึ่งเป็นส่วนหนึ่งของ Z-Circle ของ C(0,2) ทำให้ C(2,2) กลายเป็น Useless Checkpoint

2.2 โปรแกรมซอฟต์แวร์และเทคโนโลยี (Software and Technology)

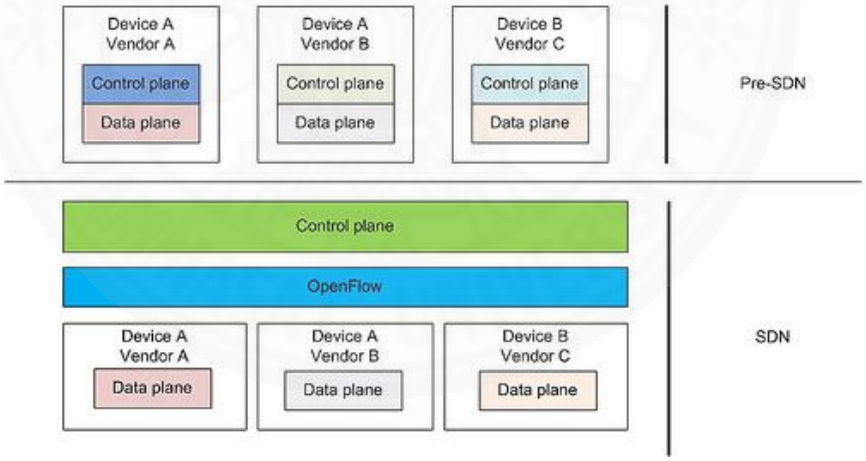
2.2.1 QEMU-KVM Hypervisor

KVM Hypervisor (Kernel-base Virtual Machine) [12] เป็น ระบบบริการจัดการทางด้าน Virtual Machine ที่พัฒนาต่อยอดมาจาก KVM ซึ่ง Software ดังกล่าวมีชื่อว่า QEMU ซึ่งพัฒนาโดย RedHat สำหรับ KVM มีคุณสมบัติ Full Virtualization ที่สามารถทำงานร่วมกับสถาปัตยกรรม x86 ที่รองรับการทำ Virtualization ในระดับ Hardware อย่าง Intel VT หรือ AMD-V ได้อย่างดี ทำให้ Virtual Machine Monitor (VMM) สามารถควบคุมจัดการทรัพยากรบน Host OS ได้ดีมากยิ่งขึ้น และ KVM ยังสามารถทำงานร่วมกับหลายๆ VMs บน Host เครื่องเดียวโดยไม่จำเป็นต้องทำการแก้ไขไฟล์ OS Image อีกด้วย

2.2.2 Software Defined Network (SDN)

ทางผู้วิจัยได้มีความสนใจในการนำเทคโนโลยี Software Defined Network เข้ามาช่วยในการควบคุมการไหลของข้อมูลและการจัดเก็บข้อมูล Frame ที่ลอยอยู่บน Network ไปเก็บไว้บน Stable Storage ได้สะดวกมากขึ้น

SDN เป็นแนวคิดหนึ่งที่เข้ามาแก้ไขปัญหาดูแลจัดการระบบเครือข่ายที่ยุ่งยาก ซึ่งกำลังได้รับความนิยมอย่างมากในปัจจุบัน จากปัญหาการเข้าถึง Network และการควบคุม Flow Control ทำให้เกิดแนวคิดการทำ Software Network ที่สามารถ กำหนด หรือ โปรแกรม ขึ้นมาตามที่ผู้ใช้ต้องการ จึงเป็นที่มาของคำว่า Software-defined Network ซึ่งแนวคิดนี้ได้ทำให้เกิดการแบ่งระดับชั้นข้อมูลที่เรียกว่า Control Plane ซึ่งเป็นระดับชั้นที่ควบคุมเส้นทางการไหลของข้อมูล และ Data Plane เป็นระดับชั้นของข้อมูล ในแง่ของ SDN จะควบคุมการไหลของข้อมูลโดยทำให้เป็น Centralize Control คือ รวบรวมการควบคุมเครือข่ายให้อยู่ในจุดเดียว เพื่อควบคุมข้อมูลหลายๆ data plane โดยที่ Control-plan จะมุ่งเน้นไปที่การเข้าไปควบคุม State บนอุปกรณ์ที่ใช้ในการควบคุม data-plane โดยตรง เช่น Route หรือ Switch เป็นต้น โดยผ่าน API ที่สามารถโปรแกรมได้ที่เรียกว่า OpenFlow



ภาพที่ 2.4 แสดงความแตกต่างระหว่าง Pre-SDN และ SDN

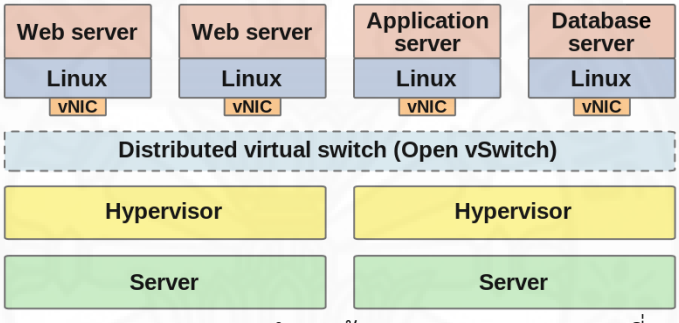
2.2.1.1 OpenFlow

OpenFlow เป็น Protocol ที่ทำงานร่วมกับ Virtual Switch ซึ่งทำหน้าที่ควบคุมเส้นทางการไหลของข้อมูลที่ผ่าน Virtual Switch นั้น โดยประกอบไปด้วย ตารางที่เก็บ Rules หรือเงื่อนไขในการไหลของ Packets ถ้าหากมี Packets ที่ไหลเข้ามาตรงตามเงื่อนไขในตารางดังกล่าว Packets นั้นจะถูกนำไปพิจารณาตามเงื่อนไขของ OpenFlow โดยจะมี Action ต่างๆ เช่น

Dropping, Forwarding หรือ Flooding packets เป็นต้น ขึ้นอยู่กับการตั้งเงื่อนไขของผู้ใช้ ซึ่งผู้ใช้เองสามารถนำ OpenFlow ไปใช้งานให้เป็นได้ทั้ง Router, Switch หรือแม้กระทั่ง Firewall หรือ NAT ก็สามารถทำได้

2.2.1.2 OpenVSwitch

OpenVSwitch (OVS) เป็น Open-Source Software ที่ช่วยในการทำ Virtual multi-layer switch ได้สะดวกมากยิ่งขึ้น และยังรองรับ OpenFlow จากหลายๆค่ายได้ เช่น NetFlow, sFlow, SPAN, RSPAN, CLI, LACP และ 802.1ag และ OVS นั้นมีจุดเด่นตรงที่สามารถทำงานข้าม Platform ได้โดยไม่จำเป็นต้องแก้ไขหรือปรับแต่ง software ใดๆ และสามารถทำงานแบบกระจายร่วมกับ server จริงจำนวนหลายเครื่องพร้อมกันได้



ภาพที่ 2.5 OpenVSwitch สามารถทำงานข้าม platform บน host ที่แตกต่างกัน [3]

นอกจากนั้นยังสามารถทำงานร่วมกับ Hypervisor หลายตัวเช่น Xen, Linux KVM, Proxmox VE และ VirtualBox รวมไปถึงได้ถูกนำเอาไปใช้งานร่วมกับระบบ Cloud Computing platform ซึ่งเป็น Infrastructure as a Service อย่าง OpenStack ด้วย จึงทำให้ OVS รองรับให้ทำงานร่วมกับ Virtual Machine ได้อย่างมีประสิทธิภาพ

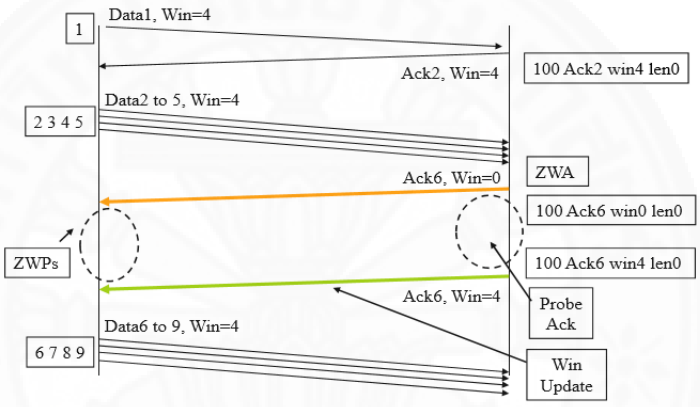
2.3 งานวิจัยที่เกี่ยวข้อง (Related Works)

2.3.1 งานวิจัยที่เกี่ยวข้องกับการจัดการข้อมูล TCP

2.3.1.1 Persistent TCP: Freeze Machanism [5]

จากปัญหาที่กล่าวไว้ตอนต้น ผู้วิจัยจึงสนใจวิธีการ Coordinate checkpointing และมองเห็นวิธีที่สามารถช่วยในการจัดการ Message ที่ถูกส่งระหว่าง Process หากเรามอง Message ให้อยู่ในระดับของ Network โดยที่จำเป็นต้องทำให้การเชื่อมต่อที่มีความน่าเชื่อถือ

(Reliability Connection) จึงจำเป็นต้องใช้ช่องการสื่อสารที่เป็นแบบ FIFO ซึ่งในระดับของ Network จะมี TCP/IP Protocol ให้บริการอยู่ และในส่วนของ Coordinate Checkpoint Protocol แบบ Blocking นั้นจำเป็นต้องทำการหยุดการสื่อสารระหว่าง Process ก่อนทำการ Checkpoint ซึ่งในระดับของ Network นั้นก็มีงานวิจัยของ C. So-In, R. Jain และ G. Dommety (2009) [6] ได้ศึกษาการทำ TCP Freeze technique เพื่อทำการหยุดการเชื่อมต่อของ TCP connection ชั่วคราวแล้วทำงานต่อได้อย่างถูกต้องด้วยช่องทางสื่อสารเดิมโดยที่ไม่ต้องปิดการเชื่อมต่อหรือสร้างการเชื่อมต่อใหม่ เทคนิคดังกล่าวจะใช้การส่ง



ภาพที่ 2.6 TCP Freeze Operation

ในปกติแล้วเมื่อมีการเชื่อมต่อ TCP Connection แล้ว Sender จะทำการส่งข้อมูลไปด้วยขนาดที่จำกัดตามขนาดของ Window Size ของ Receiver หากฝั่ง Receiver สามารถรับข้อมูลได้เพิ่มมากขึ้นก็จะทำการส่ง Acknowledgement พร้อมกับเพิ่มขนาดของ Window Size ให้มีขนาดมากขึ้น และในทำนองเดียวกัน หาก Receiver ไม่สามารถรับข้อมูลได้เนื่องจากปัญหาจากทางฝั่ง Receiver เช่น Buffer ของ Receiver เต็ม ดังนั้น Receiver ก็จะทำการส่ง ACK พร้อมกับขนาดของ Window Size ที่เท่ากับ 0 หรือเรียกว่า Zero Window Advertisement (ZWA) ในเหตุการณ์ดังกล่าวจะไม่มีการปิดการเชื่อมต่อเนื่องจากฝั่ง Receiver เองยังคอยส่ง ZWA บอกทาง Sender อยู่ตลอดว่ายังไม่สามารถรับข้อมูลได้ ซึ่ง Packets ดังกล่าวจะถูกเรียกว่า Zero Window Probe (ZWP) สุดท้ายแล้วหาก Receiver พร้อมที่จะรับข้อมูลแล้วก็เพียงส่ง ACK พร้อมกับ Window Size ที่มีค่าไม่เป็น 0 ก็จะสามารถส่งข้อมูลต่อจากเดิมได้ทันที ดังภาพที่ 3.1

แต่ถ้าหากมองปัญหาในโลกความจริงแล้ว สาเหตุที่ทำให้เกิดปัญหาการเชื่อมต่อถูกตัดขาดอาจเกิดจากการเชื่อมต่อไม่เสถียรทำให้ไม่สามารถรับส่งข้อมูลได้อย่างครบถ้วนและเกิด Timeout ไปในที่สุด ดังนั้นจึงนำไปสู่การทำ PETS: Persistent TCP using Simple Freeze (C. So-In, R. Jain และ G. Dommety, 2009) [6] โดยมีการคิดค้น Protocol เพื่อป้องกันการถูกตัดการเชื่อมต่อที่เกิดจากปัญหาดังกล่าว ในอัลกอริทึมนี้จะมุ่งเน้นไปที่การค้นหา “The Correct Acknowledgement Number” ซึ่งก็คือการค้นหา ACK ปัจจุบันที่กำลังทำการสื่อสารกันภายใน “Link” ซึ่งคำว่า “Link” ในที่นี้หมายถึง เส้นทางการสื่อสารใดๆที่เชื่อมต่อกันระหว่าง PETS Modules ในการค้นหา Packets ดังกล่าวมีเทคนิคหลักๆอยู่ 3 แบบ คือ Stateful Approach, Iterative Approach และ Best-guess Approach

Stateful Approach

ในเทคนิคนี้จะทำการเก็บ States ทั้งหมดลงบน PETS modules ในขณะที่มีการเชื่อมต่อตามปกติ ซึ่งในวิธีนี้จะสามารถหา The Real Acknowledgement Number ได้อย่างแน่นอน แต่ก็มีข้อเสียคือจะต้องมีการเก็บ States เป็นจำนวนมากทำให้เกิดความล่าช้าในการสื่อสาร

Iterative Approach

ในกรณีที่ Link ถูกตัดการเชื่อมต่อ เทคนิคนี้จะทำการค้นหา The Real Acknowledgement Number โดยดูจาก ACK Packets ที่มีการซ้ำกัน 3 Packets ข้อดีคือ มีการเก็บ States ที่น้อยกว่าแบบวิธี Stateful แต่ในบางกรณีอาจจะใช้เวลาในการค้นหานานมากจนเกิด Timer expired และทำให้ Application ทำการปิดการเชื่อมต่อลง

Best-guess Approach

PETS Modules ทำหน้าที่จะคอยสังเกตุการณ์ Retransmission Packets ของแต่ละการเชื่อมต่อ หากมี Retransmission ลำดับที่สองเกิดขึ้นใน Link ระบบจะถือว่าการเชื่อมต่อของ Link นั้นถูกตัดขาด หลังจากนั้น PETS Module จะทำการส่ง ZWA กลับไปยังอีกฝั่งทันที เมื่อการเชื่อมต่อถูกตัดขาดจริง และ ค่า Retransmission Timeout (RTO) หมดอายุลง TCP Congestion Window จะทำการลด Window Size ลง หมายความว่าฝั่งผู้ส่งจะสามารถส่งหมายเลข ACK ที่สัมพันธ์กับ Un-Acknowledgement Packet แรกได้เท่านั้น ทำให้ระบบสามารถตรวจสอบได้ว่าหมายเลข ACK ดังกล่าวเป็นหมายเลขที่ใช้ใน ZWA จริง

จากกวิธีการทั้งหมดที่กล่าวมา PETS ประกอบไปด้วยสองส่วนหลัก คือ Link State Detection และ TCP Persistence (ZWA) โดยที่ Link State Detection จะทำหน้าที่คอยเฝ้าดูการไหลของ TCP Connections และคอยเตือนหากมี Link ใดถูกตัดการเชื่อมต่อ ส่วน TCP Persistence (ZWA) จะทำหน้าที่คอยดูแลไม่ให้การเชื่อมต่อถูกตัดขาด รวมไปถึงการค้นหาหมายเลข ACK เมื่อต้องการกู้คืน Link จากการถูกตัดการเชื่อมต่อชั่วคราว

2.3.2 งานวิจัยที่เกี่ยวข้องกับ Coordinated Checkpointing

2.3.2.1 VNsnap: Taking snapshots of virtual networked infrastructures in the cloud

เป็นงานวิจัยที่ทำ Coordinate Checkpointing โดยใช้ Virtual Switch คอยควบคุมการประสานงานในการ Checkpoint บน Virtual Cluster ทุกอย่าง โดยนำ Mattern Algorithm มาประยุกต์ใช้เพื่อแยกแยะระหว่าง Messages ในช่วงเวลาก่อนและหลังทำการ Checkpoint ซึ่งจะมีเทคนิคของ Messages Coloring ทำให้สามารถแก้ไขปัญหา Inconsistent Global State ได้ อีกทั้งยังมี VIOLIN Switch เป็น Virtual Switch ที่ช่วยในการทำการ Checkpoint รวมถึงการเก็บ Messages ที่ล่องลอยอยู่บนเครือข่ายและสามารถฉีด Messages ดังกล่าวกลับเข้าไปยังเครือข่ายเมื่อทำการ Resume/Restart ให้กลับมาใช้งานได้อีกครั้ง

2.3.2.2 Transparent checkpoints of closed distributed systems in Emulab

เป็นงานวิจัยที่ทำ Coordinate Checkpointing ที่นำไป Implement บน XEN Hypervisor โดยการปรับปรุงแก้ไขให้สามารถประสานเวลาในการ Snapshot ของ VMs ได้ ด้วยการทำ Time Virtualization ซึ่งเป็นการจำลองเวลาของ VM ให้เป็นของตัวเองเพื่อแยกจาก Host OS โดยสิ้นเชิง และมีการทำ Synchronization โดยการปรับเวลาของ VMs ใน Virtual Cluster ให้ตรงกัน และสามารถกำหนดช่วงเวลาในการทำ Checkpoint ของ Virtual Cluster ได้ อีกทั้งยังได้พัฒนา Temporal Firewall เพื่อให้ระบบหยุดรอจนกว่า VMs ทุกเครื่องจะพร้อมทำการ Resume ทำให้ลดระยะเวลาการเหลื่อมล้ำของเวลาในการ Snapshot ของแต่ละ VMs บน Virtual Cluster ได้

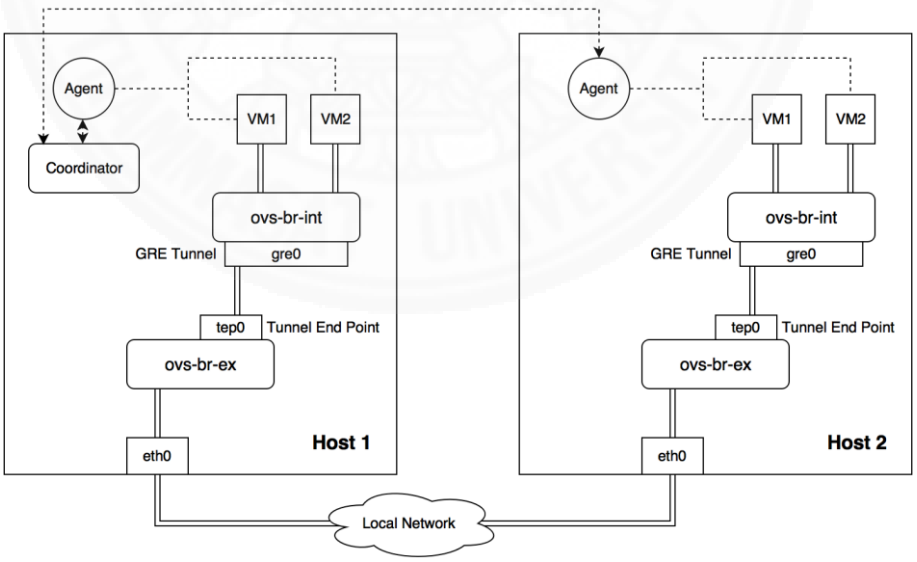
บทที่ 3
วิธีการวิจัย

3.1 โครงสร้างและภาพรวมของระบบ (Architecture Overview)

ในงานวิจัยนี้ ผู้วิจัยได้ออกแบบและพัฒนาระบบ Coordinated Checkpointing แบบโปร่งใสขึ้น โดยสภาพแวดล้อมโดยรวมของระบบจะประกอบไปด้วย 2 ระดับ ซึ่งมีรายละเอียดดังต่อไปนี้

(1) ระดับโครงสร้าง (Structure) จะประกอบไปด้วย Virtual Machine และ Virtual Network จะถูกติดตั้งอยู่บน Host ทุกเครื่อง เพื่อสร้างเป็น Virtual Cluster

(2) ระดับซอฟต์แวร์เฟรมเวิร์ค (Framework) คือระบบ Coordinated Checkpointing Framework ติดตั้งอยู่บน Host ทุกเครื่อง มีองค์ประกอบหลัก คือ ผู้ประสานงาน และ ผู้ดำเนินงาน ซึ่งจะคอยทำหน้าที่ในการประสานงานในการทำ Checkpoint/Restart



ภาพที่ 3.1 Architecture Overview

3.1.1 ระดับโครงสร้าง (Virtual Cluster และ Virtual Network)

การทำ Virtual Network เป็นวิธีที่ถูกนำไปใช้งานกันทั่วไปในระบบ Distributed Computing หรือแม้กระทั่งในระบบ Infrastructure as a Service ที่ให้บริการด้าน Cloud Computing เช่น OpenStack ก็มีการทำ Virtual Network โดยใช้ GRE Tunnel ในการเชื่อมต่อเครือข่ายภายในระหว่าง Node ต่างๆที่อยู่บน Host

จากภาพที่ 3.1 สำหรับ Virtual Cluster ในระบบนี้ประกอบไปด้วย Virtual Machine (VM) ที่เชื่อมต่อกันหลายเครื่อง ซึ่ง VMs ดังกล่าวอาจทำงานอยู่บน Host เครื่องเดียวกันหรือต่างเครื่องกันก็ได้ สำหรับการเชื่อมต่อ VMs ใน Virtual Cluster ผู้วิจัยได้นำ Software Defined Network เข้ามาช่วยในการทำ Virtual Network ซึ่งในระบบนี้จะนำซอฟต์แวร์ OpenVSwitch มาใช้เป็น Virtual Switch ภายใน Virtual Cluster ทั้งหมด โดยที่ Virtual Network จะประกอบไปด้วย Virtual Switch และ Virtual Machine สำหรับ VMs ที่อยู่ภายใน Host เดียวกันสามารถสื่อสารกันผ่าน Virtual Switch ภายใน Host แต่ในกรณีที่ VMs อยู่ต่าง Host กัน VMs จะสื่อสารผ่าน GRE Tunneling จากภาพที่ 3.1 จะเห็นได้ว่า Virtual Switch มีอยู่สองแบบ คือ External Bridge (ovs-br-ex) และ Internal Bridge (ovs-br-int)

External Bridge จะทำหน้าที่เป็น Bridge เพื่อใช้เชื่อมต่อระหว่าง Host โดยมี Interface ขาออกเชื่อมต่อกับ NIC จริง ในขณะที่ Internal Bridge ใช้เพื่อเชื่อมต่อกับ VMs ภายใน Host เดียวกัน จากภาพ 3.1 ovs-br-int ได้เชื่อมต่อกับ ovs-br-ex เพื่อใช้เป็น GRE End-Point ในการสร้าง GRE Tunnel สำหรับการสื่อสารระหว่าง Hosts สำหรับกรณีของการสื่อสารภายใน Host จะมี Internal Bridge เชื่อมต่อกับ Interface ชนิด TAP เพื่อนำไปเชื่อมต่อกับ VMs ซึ่ง TAP Interface จะถูกสร้างขึ้นมาอัตโนมัติเมื่อมีการสร้าง VMs ขึ้นมาบน Host

3.1.2 ระดับซอฟต์แวร์เฟรมเวิร์ค (Coordinated Checkpointing Framework for Virtual Cluster)

จากปัญหาที่กล่าวไว้ตอนต้น ทางผู้วิจัยจึงได้ออกแบบ Coordinated Checkpointing Framework สำหรับ Virtual Cluster ซึ่งเป็นระบบประสานงานของการ Checkpoint/Restart บน Virtual Cluster มีจุดประสงค์เพื่อลดความแตกต่างของเวลาในการทำ Checkpoint/Restart ของแต่ละ VMs ให้ใกล้เคียงกันมากที่สุด ซึ่งในหลักการเบื้องต้นของ Framework นี้คือ การลดระยะเวลาในการส่งคำสั่งจากผู้ประสานงานไปยัง Hypervisor ของแต่ละ VMs ให้รวดเร็วที่สุด และพยายามประสานงานให้ VMs ทุกตัวทำการ Restart หรือ Restart VMs ให้

เสร็จโดยใช้เวลาใกล้เคียงกันมากที่สุด สำหรับ Framework ดังกล่าวนี้ประกอบไปด้วย 2 องค์ประกอบหลัก คือ Coordinator เป็นผู้ประสานงาน และ Agents เป็นผู้ดำเนินงาน

Coordinator

Coordinator คือ ผู้ประสานงานที่เชื่อมต่อกับ Agents ทุกตัวที่อยู่ใน Framework โดยจะทำงานอยู่บน Host ใดก็ได้ที่เชื่อมต่อกับ Virtual Cluster โดยหน้าที่หลักของ Coordinator คือการกระจายคำสั่งให้กับ Agents และทำหน้าที่ตรวจสอบความพร้อม (Two-Phase Commit) ให้กับ Agents ว่า VMs ทั้งหมดใน Virtual Cluster นั้นพร้อมในการประมวลผลคำสั่งแล้วหรือไม่

นอกจากนั้น Coordinator ยังทำหน้าที่ Monitor คอยตรวจสอบสถานะต่างๆของระบบ เช่น สถานะของ Agents และ การตั้งค่าเกี่ยวกับการ Checkpoint/Restart เป็นต้น อีกทั้งยังคอยรับคำสั่งโดยตรงจาก Administrators อีกด้วย

Agents

Agents คือ ผู้ดำเนินงาน ซึ่งเป็นตัวแทนของ VMs ในการประสานงานในแต่ละ Host โดยที่ Agents จะทำงานอยู่เบื้องหลัง (Background Process หรือ Daemon Process) เพื่อคอยรับคำสั่งจาก Coordinator อยู่ตลอดเวลา และเป็นผู้ควบคุมการทำงานของ Virtual Switch เพื่อจัดการการสื่อสารข้อมูลในระหว่างการทำ Checkpoint และ Restart

หน้าที่หลักของ Agents คือ การรอรับคำสั่งจาก Coordinator และแปลคำสั่งจาก Coordinator เพื่อส่งคำสั่งไปยัง Hypervisor ของ VMs ทุกตัวที่รับผิดชอบ และต้องคอยทำ Two-Phase Commit เพื่อประสานงานกับ Coordinator ทุกครั้งที่ได้รับคำสั่ง โดยที่ Coordinator จะเป็นผู้ส่ง Request Message ของคำสั่งใดๆไปยัง Agents ทุกตัวเพื่อเตรียมความพร้อม หลังจากนั้น Agent แต่ละตัวจะทำการตรวจสอบความพร้อมของการประมวลผลคำสั่งของ VMs ที่รับผิดชอบ เมื่อ VMs ทุกเครื่องพร้อมประมวลผลคำสั่ง Agent จะส่ง Ready Message กลับไปและคอยรับ Commit Message เพื่อยืนยันความพร้อมของ VMs ทุกเครื่องในระบบ หลังจากนั้น Agents จึงจะแปลคำสั่งที่ถูก Request Message แล้วส่งให้ Hypervisor ของ VMs แต่ละเครื่องประมวลผลคำสั่งดังกล่าวต่อไป

สำหรับงานในอนาคตผู้วิจัยคาดว่าจะออกแบบ Agents ที่เชื่อมต่อกับ Hypervisor ที่มีความสามารถในการทำ Live Checkpointing แบบ Time bound, Pre-copy Live

Checkpointing (TPLCR) [10] ซึ่งเป็น QEMU-KVM ที่ทำการปรับปรุงให้สามารถทำ checkpoint ได้รวดเร็วยิ่งขึ้น โดยใช้เทคนิคของการ Live Checkpointing เข้ามาเสริม ซึ่งเป็นแนวทางที่สามารถส่งผลให้ลดระยะเวลา Downtime ในการทำ Checkpoint ของ VMs ในระบบ Virtual Cluster ลงได้

Stable Storage

ในส่วนของ Checkpoint Files ที่ถูกสร้างขึ้นจาก Framework นั้น ประกอบไปด้วย Snapshot File คือ State ของ VM ณ ขณะใดขณะหนึ่งที่ถูกบันทึกเอาไว้ ประกอบไปด้วย CPU State คือ สถานะการประมวลผลของ CPU ในขณะนั้น รวมไปถึงข้อมูลภายใน Register และ Cache ต่างๆ และ Memory Dump คือ Memory Pages ที่ถูกเรียกขึ้นมาใช้งานในขณะนั้น ซึ่งจะสัมพันธ์กับ Disk Image ของ VM นั้นๆด้วย ดังนั้นก่อนนำไปใช้จะต้องมั่นใจว่า Disk Image ที่ใช้นั้นถูกต้องและสัมพันธ์กับ Memory Dump ที่นำไปใช้ เพื่อไม่ให้เกิดความผิดพลาดในกรณีที่มีการทำ Page Fault หลังจากการ Restart VMs ซึ่งอาจทำให้ VM หยุดการทำงานลงได้

สำหรับ Stable Storage ซึ่งเป็นพื้นที่สำหรับการเก็บข้อมูลสำรอง Checkpoint Files ของ Virtual Cluster โดยจะถูกเก็บไว้บน Host อื่นที่ไม่ได้ถูกรัน Virtual Cluster ซึ่งในขณะที่มีการทำ Checkpoint นั้น Framework จะสั่งให้ Agents คอยทยอยส่ง Checkpoint File ผ่านเครือข่ายไปยัง Host ปลายทางที่เตรียมไว้ ซึ่งมีโปรแกรมทำหน้าที่จัดเก็บ Checkpoint Files ลงบน Disk หรือเรียกว่า Storage Agent ในขณะเดียวกันหากมีการ Restart เกิดขึ้น เมื่อ Storage Agent ได้รับสัญญานดังกล่าวก็จะทำการอ่านข้อมูลจาก Disk และส่งกลับไปยัง Agents ทั้งหมดที่ดูแล Virtual Cluster เพื่อนำ Checkpoint File ไปทำการ Restart VMs ต่อไป

3.2 ขั้นตอนวิธีการแก้ไขปัญหา (Algorithm)

เนื่องจากทางผู้วิจัยต้องการพัฒนา Coordinate Checkpointing แบบโปร่งใส (Transparency) จึงมีแนวคิดที่สามารถเข้ามาช่วยในการแก้ปัญหาดังกล่าวได้อยู่ทั้งหมด 2 แนวคิด คือ

หลักการ Barrier Synchronization เพื่อแก้ไขปัญหา Inconsistent Global State

เพื่อแก้ไขปัญหา Global State ที่เกิดจากการสื่อสารของ VM ภายใน Virtual Cluster จึงต้องมีการแยกแยะ Messages ที่อยู่ในช่วงเวลาก่อนและหลังของการทำ Checkpoint ให้

แยกออกจากกันอย่างชัดเจนเพื่อแก้ไขปัญหา Inconsistent Global State ซึ่งจะต้องใช้เวลาพอสมควรในการประสานงานระหว่าง VMs เพื่อแยกแยะ Messages ดังกล่าวในการทำ Checkpoint/Restart ขึ้นอยู่กับความซับซ้อนของเครือข่ายภายใน Virtual Cluster โดยเฉพาะอย่างยิ่งหากบนเครือข่ายมีปริมาณของข้อมูลการสื่อสารเป็นจำนวนมากจะทำให้การค้นหา Consistent Global State Cut ที่เหมาะสมนั้นยากขึ้นเช่นกัน

ทางผู้วิจัยจึงได้มีแนวคิดการประสานงานที่เรียกว่า Barrier Synchronization คือ การ Block รอให้ VMs ทุกเครื่องทำงานพร้อมกัน ในช่วงที่ทำการ Checkpoint โดยการใช้ Two-Phase Commit Protocol เพื่อประสานงานระหว่าง VMs และ Coordinator เพื่อเป็นการแยกช่วงเวลาของก่อนและหลังการทำ Checkpoint ให้ชัดเจน ซึ่งจะเป็นการบังคับให้ช่วงเวลาดังกล่าวเกิดเหตุการณ์ Consistent Global State Cut จึงสามารถมั่นใจได้ว่า Checkpoint ดังกล่าวจะไม่มีเหตุการณ์ Inconsistent Global State และ Domino Effect เกิดขึ้น

นอกจากนั้นเนื่องจากการประสานงานแบบ Barrier Synchronization ยังทำให้ VMs ภายใน Virtual Cluster ทำงานได้พร้อมกันมากยิ่งขึ้น จึงทำให้ลดระยะเวลาที่เหลื่อมล้ำกันระหว่างการทำ Checkpoint ในแต่ละ VMs บน Virtual Cluster ได้มากยิ่งขึ้น ดังนั้นจึงส่งผลทำให้สามารถลดระยะเวลาการเกิดปัญหา TCP Backoff ลงได้ด้วย

ในงานวิจัยนี้ ผู้วิจัยจะนำเสนอบทพิสูจน์ทั้งทางทฤษฎีและปฏิบัติว่าการใช้ Barrier Synchronization สามารถป้องกันการเกิด Inconsistent Global State และเหตุการณ์ The Domino Effect ได้ และยังส่งผลทำให้ลดระยะเวลาที่เหลื่อมล้ำกันระหว่างการทำ Checkpoint/Restart ในแต่ละ VMs บน Virtual Cluster ได้มากยิ่งขึ้น

หลักการ Virtual Time เพื่อแก้ไขปัญหา TCP Backoff

แนวคิดของการจำลองเวลาของ Virtual Cluster คือ การกำหนดให้ VMs มีเวลาเป็นของตัวเอง โดยแยกออกจากเวลาของ Host OS ซึ่งในระบบนี้จะเรียกว่าเวลาดังกล่าวว่า Virtual Time ซึ่งเป็นแนวคิดที่สามารถทำให้เวลาของ Virtual Machine และรวมไปถึงเวลาของข้อมูลที่ไหลอยู่บนเครือข่ายบน Virtual Cluster มีเวลาเป็นของตัวเองซึ่งแยกเป็นอิสระจากโลกภายนอก ดังนั้นสิ่งที่เกิดขึ้นเกี่ยวกับเรื่องของเวลาต่างๆจะมีผลกระทบอยู่แค่ภายใน Virtual Cluster เท่านั้น สาเหตุที่ต้องใช้ Virtual Time คือ การทำให้การควบคุมเวลาของทั้ง Virtual Cluster ทำได้ง่ายมากยิ่งขึ้น

การหยุดเวลาเพื่อทำการ Checkpoint ซึ่งจะเกี่ยวข้องกับเวลาของข้อมูลที่ไหลอยู่บนเครือข่าย ณ เวลานั้นด้วย เมื่อเวลาถูกหยุดระบบจะทำการเก็บข้อมูลบนเครือข่ายไว้ และเมื่อนำ

ข้อมูลเหล่านั้นมาใช้งานต่อ เวลาของข้อมูลเหล่านั้นจะต้องสัมพันธ์กับเวลาของ VMs ด้วย เพื่อทำให้การทำงานของ Virtual Cluster ลื่นไหลโดยที่ยังไม่รู้ตัวว่าถูกหยุดการทำงาน

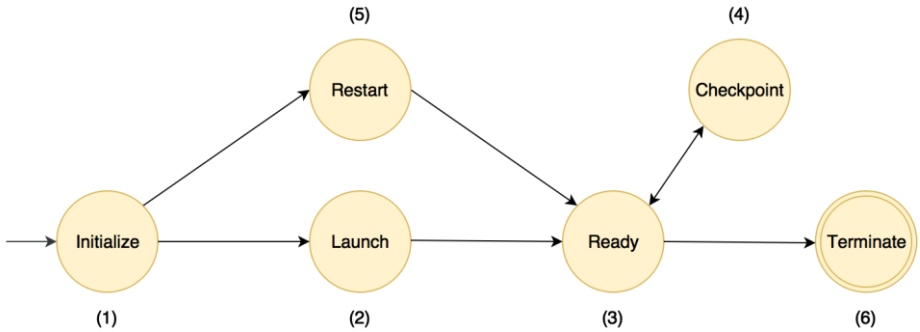
ในการทำ Checkpoint บน Virtual Cluster โดยทำการหยุดเวลาของระบบทั้งหมดก่อนทำการ Checkpoint ซึ่ง VMs ทั้งหมดและข้อมูล Checkpoint ที่ถูกเก็บไว้ทั้งหมดจะถูกหยุดในเวลาที่ใกล้เคียงกัน เมื่อมีการนำข้อมูลเหล่านั้นกลับมาใช้ด้วยการ Restart ในเวลาใดก็ตาม จะทำให้ Virtual Cluster สามารถทำงานต่อได้อย่างต่อเนื่องโดยที่ยังไม่รู้ตัวว่าถูกหยุดการทำงาน จากการทำงานในเวลาที่ใกล้เคียงกันมาก ทำให้เวลาที่เหลื่อมล้ำกันของ VMs ทั้งหมดและข้อมูลที่อยู่ในระหว่างการสื่อสารบนเครือข่ายมีเวลาที่แตกต่างกันน้อยลง จึงสามารถลดเวลาการเกิด TCP Backoff ได้มากยิ่งขึ้น

สำหรับ Virtual Time ของ Virtual Cluster จะถูกบริหารจัดการด้วยระบบ Framework ซึ่งคอยประสานเวลากันเพื่อให้เวลาของแต่ละ VMs ตรงกันให้มากที่สุด ส่วนการจำลองเวลาของ VM สามารถทำได้โดยใช้ QEMU-KVM Hypervisor เข้ามาช่วยจัดการได้ ซึ่งจะมี Option ในการเรียกใช้ VM ให้มีเป็นเวลาของตัวเองได้

จากแนวคิดดังกล่าว จึงนำไปสู่การพัฒนา Checkpoint/Restart Protocol ในการประสานงานในการทำ Coordinate Checkpointing ซึ่งจะอธิบายในหัวข้อถัดไป

3.2.1 State Diagram ของ Framework

ในการเริ่มต้นงานทำงานของ Framework ต้องทำการติดตั้ง Coordinator และ Agents ไว้บน Host ให้เรียบร้อย ในเบื้องต้นจะติดตั้ง Coordinator ไว้บนหนึ่งใน Host ใดๆในระบบ สำหรับ Agents จะถูกติดตั้งอยู่บน Host ทุกตัว ให้ตรวจสอบว่าได้มีการติดตั้งตามโครงสร้างของระบบ (เป็นไปตามหัวข้อที่ 3.1) ไว้เรียบร้อยแล้ว โดย Coordinator มีสถานะบ่งบอกลำดับขั้นตอนของการทำงานทั้งหมดใน Framework โดยมีทั้งหมด 6 สถานะ คือ Initialize, Launch, Ready, Checkpoint, Restart และ Terminated



ภาพที่ 3.2 The States of Framework

(1) Initialize State

ในการเริ่มต้นของระบบ Coordinator และ Agents จะถูกรันเป็น Background Process หรือ Daemon Process โดยที่ Coordinator จะทำตัวเป็นเสมือน Client เชื่อมต่อไปยัง Agents ที่เป็น Server รันอยู่บน Host ทุกเครื่องในระบบ เมื่อ Connection ระหว่างโปรแกรม Coordinator และ Agents ถูกสร้างขึ้นเรียบร้อยแล้ว Coordinator จะทำการตรวจสอบสถานะทั้งหมดของ Agents

หลังจากนั้น Framework จำทำการตรวจสอบว่าระบบถูกเริ่มต้นด้วยคำสั่ง Restart หรือไม่ หากเป็นการเริ่มต้นแบบปกติจะเตรียมตัวเข้าสู่สถานะพร้อมทำการสร้าง VMs หรือ Launch State แต่ถ้าหากระบบถูกเริ่มต้นใหม่ด้วยคำสั่ง Restart ระบบจะถูกปรับเปลี่ยนไปทำงานตาม Restart Protocol เพื่อเข้าสู่ขั้นตอนการกู้คืน Virtual Cluster จาก Checkpoint ที่ Admin กำหนดไว้

(2) Launch State

ในขั้นตอนแรกของการเตรียม VM จะต้องมีการกำหนดอย่างชัดเจนว่าจะนำ VM ไปรันไว้ที่ Host เครื่องใด โดยที่ Coordinator จะทำการส่งคำสั่ง “launch” ไปยัง Agent บน Host ที่ต้องการรัน VM ถัดจากที่ Agent ได้รับคำสั่งจะทำการกำหนด MAC Address เสมือนให้กับ VM โดยที่ MAC Address ดังกล่าวจะต้องไม่ซ้ำกับ VM เครื่องอื่นที่กำลังทำงานอยู่และ VMs ที่ถูกสร้างขึ้นมาภายหลัง นอกจากนั้นสามารถกำหนด Internal Network ให้แก่กลุ่มของ VMs ที่ทำการสร้างขึ้นมาได้ เมื่อเตรียม VMs เสร็จสิ้น ระบบจะพร้อมเข้าสู่สถานะพร้อมทำการ Checkpoint หรือ Ready State

(3) Ready State

สถานะพร้อมการทำ Checkpoint จาก Checkpoint File รวมถึง Terminate ด้วย ในเบื้องต้นจะมีการกำหนดรอบการทำ Checkpoint เป็นระยะตามที่ Admin กำหนดไว้ ในสถานะดังกล่าวระบบจะคอยจับตาดูความผิดพลาดที่เกิดขึ้นใน Virtual Cluster โดย Coordinator จะคอยทำการส่งสัญญานบอกให้ Agents คอยตรวจสอบว่า VMs ยังทำงานอยู่หรือไม่ หากมี VM เครื่องใดไม่สามารถส่งสัญญานตอบกลับมายัง Agent ระบบจะทำการหยุดการทำงานของ Virtual Cluster ทั้งหมดแล้วทำการแจ้งไปยัง Admin เพื่อทำการ Restart Virtual Cluster จาก Checkpoint File ทำการ Restart จากจุด Checkpoint ที่ต้องการด้วยการเริ่มต้นการทำงานใหม่แบบ Restart

(4) Checkpoint State

เมื่อระบบครบรอบเวลาทำการ Checkpoint หรือได้รับคำสั่งจากผู้ใช้ ระบบจะทำการ Checkpoint โดยทำงานตาม Checkpoint Protocol ที่กำหนดไว้ โดยจะประกอบไปด้วยขั้นตอนย่อยอยู่ 4 ขั้นตอน คือ Initialize, Stop, Save และ Resume โดยจะขอยกไปอธิบายรายละเอียดในหัวข้อ Checkpoint Protocol

(5) Restart State

เมื่อระบบตรวจพบเกิดเหตุขัดข้องขึ้นใน Virtual Cluster ระบบจะทำการ Restart จาก Checkpoint โดยทำงานตามกลไกล Restart Protocol ตามที่กำหนดไว้ โดยจะประกอบไปด้วยขั้นตอนย่อยอยู่ 3 ขั้นตอน คือ Initialize, Load และ Resume โดยจะขอยกไปอธิบายรายละเอียดในหัวข้อ Restart Protocol

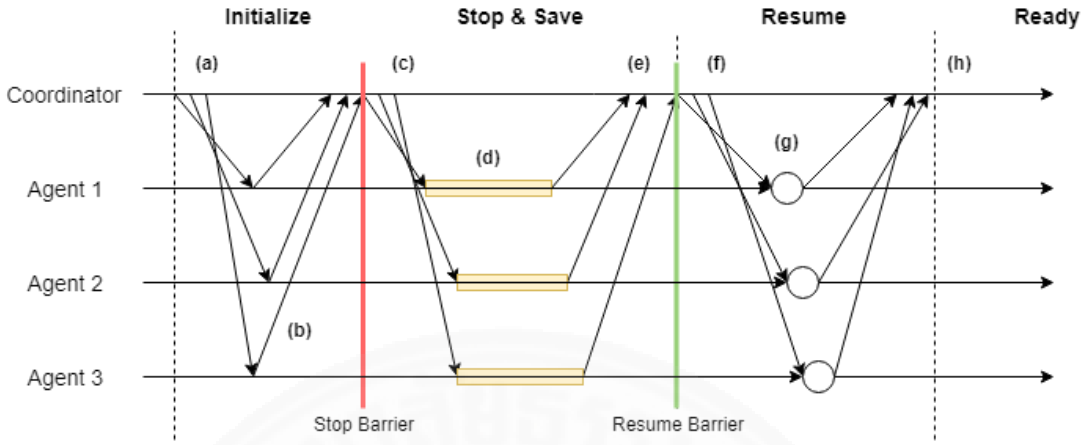
(6) Terminated State

ในขั้นตอนสุดท้าย เมื่อ VMs ทำงานเสร็จสิ้น ระบบจะเข้าสู่ Terminated State เพื่อสิ้นสุดการทำงาน หลังจากนั้นระบบจะสั่งให้ VMs หยุดการทำงานและคืนทรัพยากรให้แก่ระบบ

3.2.2 Checkpoint Protocol

เมื่อระบบถึงเวลาในการทำ Checkpoint ระบบจะเรียก Protocol นี้ขึ้นมาทำงาน ซึ่ง Protocol นี้จะช่วยจัดการการสร้าง Snapshot และการ Resume ของ VM ทุกเครื่องบน Virtual Cluster แบบ Coordinated Checkpointing ด้วยการทำ Barrier Synchronization ซึ่งจะมีอยู่ 2 ช่วงเวลาเท่านั้น คือ ช่วงเวลาก่อนทำการ Checkpoint และช่วงเวลาก่อนทำการ Resume เพื่อ Block รอให้ VMs ทำงานให้พร้อมกันมากที่สุด โดยในขั้นตอนแรกในการเตรียมการ Checkpoint ระบบจะทำ Barrier Synchronization เพื่อเตรียมการทำการสร้าง Snapshot ของ VMs ให้พร้อมกัน และในขณะเดียวกันระบบจะทำการดักจับข้อมูลบนเครือข่ายในขณะนั้นด้วย หลังจากที่ระบบทำการสร้าง Snapshot เรียบร้อยแล้ว ระบบจะทำ Barrier Synchronization อีกครั้งเพื่อให้ VMs ทุกเครื่องกลับมาทำงานต่อได้อย่างพร้อมเพรียงกัน

สำหรับขั้นตอนในการทำ Checkpoint Protocol จะประกอบได้ด้วย 4 ขั้นตอน คือ Initialize, Stop, Save และ Resume



ภาพที่ 3.3 Algorithm of Checkpoint Protocol

Initialize State

เมื่อ Coordinator ได้รับสัญญานให้ทำการ Checkpoint ก็จะส่งสัญญานไปยัง Agents เพื่อประสานงานในการเตรียมพร้อมการทำ Checkpoint โดยทำ Two-Phase Commit ซึ่งมีกระบวนการดังต่อไปนี้

(1) Coordinator ทำการส่ง Checkpoint Request Message ไปยัง Agents ทุกตัวที่อยู่ในระบบ ดังภาพที่ 3.3 (a)

(2) เมื่อ Agent(i) ทำการตรวจสอบความพร้อม VMs ทุกตัวที่รับผิดชอบให้เรียบร้อยและเตรียม Stable Storage ให้พร้อม หลังจากนั้น Agent(i) จะส่ง Acknowledgement (ACK) กลับไปยัง Coordinator เพื่อยืนยันความพร้อมในการทำ Checkpoint ดังภาพที่ 3.3 (b) โดยกำหนดให้ Agent(i) หมายถึง Agent โปรแกรมบน Host (ได้แก่ Agent1, Agent2 หรือ Agent3 ในภาพ 3.3)

(3) เมื่อ Coordinator ได้รับ ACK จาก Agents ครบทุกตัว จะเปลี่ยนทำงานเป็นสถานะ Stop State ถัดไป

หลังจากที่ Coordinator ได้รับ ACK จากการส่ง Checkpoint Request หมายความว่า Agents ทุกตัวภายใน Virtual Cluster ได้มีสถานะพร้อมในทำการ Checkpoint เรียบร้อยแล้ว ซึ่งกระบวนการทำ Checkpoint ซึ่งจะประกอบไปด้วย 3 ขั้นตอน ได้แก่ Stop, Save และ Resume

Stop&Save State

ในขั้นตอนดังกล่าว Virtual Cluster จะถูกสั่งให้หยุดการทำงานทั้งหมด โดยที่ Coordinator จะส่ง Commit Message จากการทำ Two-Phase Commit ในขั้นตอน Initialize ไปยัง Agents เพื่อสั่งให้หยุดการทำงานของ VMs ทุกตัวพร้อมกัน ตามกระบวนการต่อไปนี้

(1) Coordinator ทำการส่ง Commit Message กลับไปยัง Agent(i) เพื่อยืนยันว่า Agents ทุกตัวพร้อมทำการ Checkpoint แล้ว ดังภาพ 3.3 (c)

(2) Agent(i) ได้รับ Commit Message เป็นการเสร็จสิ้นการทำ Two-Phase Commit หลังจากนั้น Agent(i) จะทำการสั่ง Hypervisor ให้ทำการหยุดการทำงานของ VM ทั้งหมด (stop)

(3) Agent(i) ทำการสั่งให้ Hypervisor บน VM(k) ทำการสร้าง Snapshot (savevm) หลังจากที่ VM ถูกหยุดลงแล้วทันที ซึ่งระบบจะต้องหยุดรอให้ VMs ทั้งหมดสร้าง Snapshot ให้เสร็จก่อนจึงทำงานต่อไปได้ ดังภาพที่ 3.3 (d) โดยกำหนดให้ VM(k) เครื่อง VM บน Host ได้แก่ VM1 และ VM2

(4) หลังจากที่ VMs ทุกเครื่องสร้าง Snapshot เรียบร้อยแล้ว จะส่ง FIN-ACK Message กลับไปยัง Coordinator ดังภาพที่ 3.3 (e)

(5) เมื่อ Coordinator ได้รับ ACK จาก Agents ครบทุกตัวแล้ว ถือว่าเป็นการเสร็จสิ้นสถานะ Save State และเปลี่ยนสถานะไปยัง Resume State ต่อไป

หลังจากที่ Hypervisor บน VMs ทุกเครื่องได้ทำการ Snapshot เรียบร้อยแล้ว Agents ทุกตัวจะนำข้อมูลดังกล่าวส่งไปยัง Remote Server ที่เตรียมเอาไว้ โดยทำการติดต่อไปยัง Storage Agent ที่อยู่บน Remote Server เพื่อทำการส่ง file ซึ่ง Storage Agentดังกล่าวจะเตรียม Memory และ Disk ไว้เพื่อเก็บ Checkpoint file หมายเหตุ: ในระหว่างกระบวนการดังกล่าวระบบ Virtual Cluster สามารถเปลี่ยนสถานะการทำงานเข้าสู่สถานะ Resume State ได้ทันทีโดยไม่จำเป็นต้องรอให้ Agent ส่ง Checkpoint File ไปเก็บไว้ยัง Remote Backup ให้เสร็จเรียบร้อยก่อน ทำให้ช่วยลดระยะเวลา Downtime จากการรอ I/O ได้มากขึ้น

Resume State

หลังจากที่ทำการ Checkpoint เรียบร้อยแล้ว ขั้นตอนต่อไปคือการ Resume VMs ให้พร้อมกันมากที่สุดเพื่อลดความแตกต่างของเวลาในการ Resume ของแต่ละ VMs ที่เป็นสาเหตุทำ

ให้เกิด TCP Backoff ดังนั้นจึงต้องมีการประสานงานระหว่าง Coordinator และ Agents ด้วยการทำ Two-Phase Commit อีกครั้ง ซึ่งมีกระบวนการดังต่อไปนี้

(1) Coordinator ทำการส่ง Resume Request Message ไปยัง Agents ทุกตัวที่อยู่ในระบบ ดังภาพ 3.3 (f)

(2) เมื่อ Agent(i) ได้รับ Resume Request Message แล้ว จะทำการอนุญาตให้ Virtual Switch ทุกตัวใน Virtual Cluster ทำงานต่อได้

(3) หลังจากนั้น Agent(i) ตรวจสอบว่า Hypervisor ของ VM(k) ทุกเครื่องพร้อมทำการ Resume แล้ว หลังจากนั้น Agent(i) จะทำการส่ง จะส่ง Acknowledgement กลับไปยัง Coordinator เพื่อยืนยันความพร้อมในการทำ Resume VM ดังภาพ 3.3 (g)

(4) เมื่อ Coordinator ได้รับ ACK จาก Agents ครบทุกตัวเรียบร้อยแล้ว จะทำการส่ง Commit Message กลับไปยัง Agent(i) เพื่อยืนยันว่า Agents ทุกตัวพร้อมทำการ Resume และอนุญาตให้ Agent(i) ทำการ Resume VM ได้ทันทีเมื่อได้รับ Commit Message ดังภาพ 3.3 (h)

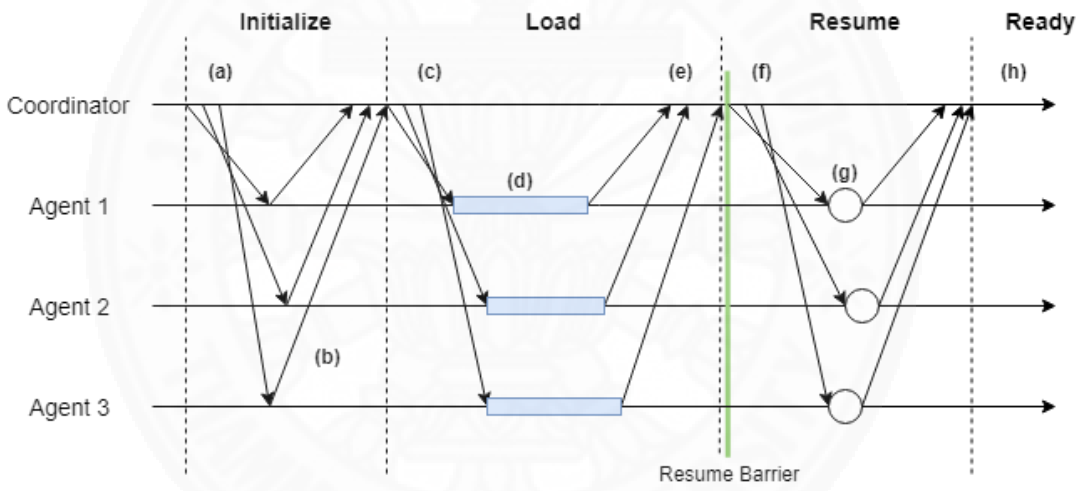
(5) หลังจากที่ Agent(i) ได้รับ Commit Message เป็นการเสร็จสิ้นการทำ Two-Phase Commit หลังจากนั้น Agent(i) จะสั่งให้ Hypervisor บน VM(k) เริ่มทำการ Resume VM (cont)

หลังจากที่ทำการ Resume เสร็จสิ้น ดังภาพ 3.3 (h) Virtual Cluster จะกลับมาทำงานตามปกติ แต่อาจเกิดความล่าช้าจากการเกิด TCP Backoff ขึ้นอยู่กับระยะเวลาของการ Resume ของแต่ละ VMs ว่ามีความแตกต่างกันมาน้อยเพียงใด หลังจากนั้นระบบจะกลับไปยังสถานะ Ready State อีกครั้ง

3.2.3 Restart Protocol

หากระบบตรวจพบความผิดพลาดที่เกิดขึ้นภายใน Virtual Cluster เนื่องจาก VM เครื่องใดเครื่องหนึ่งเกิดความผิดปกติหรือไม่สามารถทำงานต่อได้ หรือแม้กระทั่งปัญหาที่เกิดจากการทำงานของ VM ที่ทำให้เกิดผลลัพธ์ผิดพลาด ทำให้ผู้ใช้ต้องทำการ Restart จาก Checkpoint File ที่ได้ทำการสำรองเอาไว้ เพื่อเริ่มต้นการทำงานใหม่จากจุด Checkpoint ใดๆ ให้สามารถทำงานต่อจากจุดที่ใกล้เคียงกับจุดที่ผิดพลาดได้อย่างถูกต้อง

สำหรับ Restart Protocol จะช่วยทำให้ทำการ Resume ของ VM ทุกเครื่องบน Virtual Cluster ทำได้พร้อมเพรียงกันมากยิ่งขึ้น ในขั้นตอนในการกู้คืน State ของ Virtual Cluster จาก Checkpoint ใดๆที่ต้องการจะต้องทำการเตรียม Checkpoint File จาก Local Storage เข้าสู่ Memory เสียก่อน หลังจากนั้นระบบจะทำการกู้คืน State ให้แก่ VMs ทั้งหมด รวมถึงข้อมูลที่อยู่ในระหว่างการสื่อสารบนเครือข่ายในช่วงเวลาของ Checkpoint ดังกล่าว หลังจากนั้นจะทำการ Resume VM พร้อมกัน ซึ่งในการทำการ Resume ของ Virtual Cluster มีความจำเป็นต้องทำ Barrier Synchronization เช่นเดียวกับ Checkpoint Protocol โดยจะทำในช่วงที่ทำการ Resume VM เพื่อ Block รอให้ VMs ทำงานให้พร้อมกันมากที่สุด ในการทำ Restart Protocol จะประกอบได้ด้วย 3 ขั้นตอน คือ Initialize, Load และ Resume ดังแสดงในภาพ 3.4



ภาพที่ 3.4 Algorithm of Restart Protocol

Initialize State

เมื่อเกิดความผิดพลาดขึ้นบน Virtual Cluster และ Coordinator และ Agents ตรวจจับความผิดพลาดได้นั้น Coordinator สามารถแก้ไขปัญหาได้ 2 วิธี คือ วิธีการแรก Coordinator แจ้งความผิดพลาดไปยัง Admin ทราบเพื่อให้เข้ามาหยุดการทำงานของ Virtual Cluster และทำการ Restart ด้วยตนเอง วิธีการที่สอง Coordinator จะทำการส่งสัญญานไปบอก Agents ทุกตัวว่ามีข้อผิดพลาดขึ้นในระบบทำให้ต้อง Restart จากจุด C(i,n) ใดๆ โดยที่ C(i,n) หมายถึง Checkpoint File ของ Agent(i) ในลำดับครั้งที่ n

(1) Coordinator จะทำการส่ง Restart Request ไปยัง Agent(i) เพื่อตรวจสอบว่า Agent พร้อมที่จะทำการ Restart ดังภาพที่ 3.4 (a)

(2) เมื่อ Agent(i) ได้รับ Restart Request ก็จะทำการเตรียม VMs และ Checkpoint Files ไว้เพื่อทำการ Restart VM จาก Checkpoint และเมื่อเตรียมพร้อมทำการ Restart เรียบร้อยแล้ว Agent(i) จะส่ง ACK กลับไปยัง Coordinator เพื่อยืนยันความพร้อม ดังภาพที่ 3.4 (b)

(3) เมื่อ Coordinator ได้รับ ACK จาก Agent(i) ครบทุกตัวเรียบร้อยแล้ว ระบบจะเข้าสู่ Load State เพื่อทำการ Restart VM ต่อไป

Load State

ขั้นตอนการอ่านข้อมูลจาก Checkpoint File ขึ้นมาเพื่อเริ่มต้นการทำงานต่อจากจุด Checkpoint ที่กำหนดไว้

(1) ทันทีที่เข้าสู่ Load State ทาง Coordinator จะส่ง Load Request Message ไปยัง Agent(i) เพื่อทำการ Start VMs จาก Checkpoint C(i,n) ที่เตรียมไว้ ดังภาพที่ 3.4 (c)

(2) เมื่อ Agent(i) ได้รับ Load Request ก็จะทำการ Restart VMs จาก C(i,n) ทันที ดังภาพที่ 3.4 (d)

(3) หลังจากนั้น เมื่อ VMs ทุกเครื่อง Load เสร็จเรียบร้อย Agent(i) จะส่ง ACK กลับไปยัง Coordinator ว่าทำการ Load เสร็จสิ้น ดังภาพที่ 3.4 (e)

(4) Coordinator จะหยุดรอให้ Agent ทุกตัวทำงานเสร็จเพื่อเป็นการทำ Resume Barrier เมื่อ Coordinator ได้รับ Load FIN Message จาก Agents ครบแล้ว ก็จะเปลี่ยนสถานะเป็น Resume State เพื่อเริ่มการทำงานของ VMs ต่อไป

Resume State

เช่นเดียวกับ Resume State ใน Checkpoint Mode คือการ Resume VMs ให้พร้อมกันมากที่สุดเพื่อลดความแตกต่างของเวลาในการ Resume ของแต่ละ VMs โดยที่การประสานงานระหว่าง Coordinator และ Agents ซึ่งมีกระบวนการดังต่อไปนี้

(1) Coordinator ทำการส่ง Resume Request Message ไปยัง Agent(i) ทุกตัวที่อยู่ในระบบ (f)

(2) หลังจากนั้น Agent(i) ทำการออกคำสั่งให้ Hypervisor บน VM(k) ทำการคืนค่า State (loadvm) จาก Snapshot(k) ที่ได้จาก Checkpoint File C(i,n)

(3) เมื่อ Hypervisor บน VM(k) ทุกเครื่องได้ทำการกู้คืน State (loadvm) เสร็จเรียบร้อยแล้ว หลังจากนั้น Agent(i) จะทำการส่ง Acknowledgement กลับไปยัง Coordinator เพื่อยืนยันความพร้อมในการทำ Resume VM

(4) หลังจากที่ Coordinator ได้รับ ACK จาก Agent(i) ครบทุกตัวเรียบร้อยแล้ว จะทำการส่ง Commit Message กลับไปยัง Agent(i) เพื่อยืนยันว่า Agents ทุกตัวพร้อมทำการ Resume และอนุญาตให้ Agent(i) ทำการ Resume VM ได้ทันทีเมื่อได้รับ Commit Message

(5) Agents(i) ได้รับ Request Message หลังจากนั้น Agent(i) จะเริ่มสั่งให้ Hypervisor บน VM(k) ทำการ Resume VM (cont) จาก Snapshot(k)

(6) หลังจากที่ Agent(i) ทุกตัวทำงานเสร็จ ก็จะทำการส่ง FIN Message กลับไปยัง Coordinator เพื่อบอกว่าทำการ Restart เสร็จเรียบร้อยแล้ว

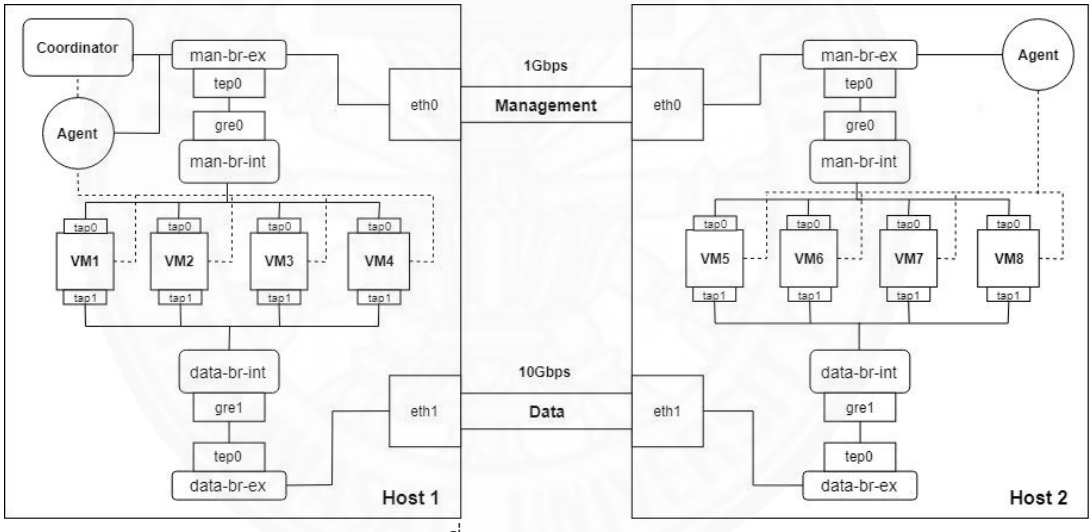
(7) เมื่อ Coordinator ได้รับ FIN Messages จาก Agents ครบทุกตัวแล้ว ถือว่าเป็นการสิ้นสุดการทำ Restart หลังจากนั้นระบบจะออกจาก Restart Mode

หลังจากที่ Virtual Cluster ถูก Restart ขึ้นมาทำงานต่อเสร็จสิ้น ระบบจะกลับไปยังสถานะ Ready State อีกครั้ง

บทที่ 4
ผลการวิจัยและอภิปรายผล

4.1 การออกแบบการทดลอง

ในงานวิจัยนี้ ทางผู้วิจัยได้ทำการทดลองระบบบนเครื่องคอมพิวเตอร์จริง (Host Computer) จำนวน 2 เครื่อง รุ่น Dell PowerEdge R720 server มีหน่วยประมวลผล Two hyper-threaded 10-cores Xeon E5 ความเร็ว 2.8GHz (CPU รวม 40 Cores) และ หน่วยความจำ 192 GB เก็บข้อมูลโดยใช้ NAS Storage ความจุขนาด 12TB เชื่อมต่ออยู่บนเครือข่าย 1Gbps และ 10Gbps



ภาพที่ 4.1 Virtual Cluster

4.1.1 Virtual Cluster

ผู้วิจัยได้ออกแบบ VC บนเครื่องคอมพิวเตอร์จริง 2 เครื่อง ดังที่กล่าวไว้ในบทที่ 3 โดย VC จะประกอบไปด้วย Virtual Network จำนวน 2 Network และ Virtual Machine จำนวน 8 เครื่อง ดังภาพที่ 4.1

4.1.1.1 Virtual Network

บนเครื่อง Host ทั้งสองจะถูกเชื่อมต่อกันด้วยเครือข่ายความเร็วสูง 1Gbps และ 10Gbps ทางผู้วิจัยได้ออกแบบ Virtual Network โดยใช้ Virtual Switch แบบ OpenVSwitch เพื่อเชื่อมต่อ VM ทุกเครื่องบน VC โดย Virtual Network ที่ผู้วิจัยสร้างขึ้นประกอบไปด้วย

Management Network

ทำหน้าที่เป็นเครือข่ายที่รองรับการส่งข้อมูลชนิดที่มีติดต่อสื่อสารกันเพื่อให้บริการ Service หรือสนับสนุนการประมวลผล Applications ได้แก่ SSH, NFS และการส่งคำสั่งของ Coordinator และ Agents ซึ่งเครือข่ายเสมือนนี้จะถูกติดตั้งบนเครือข่ายจริงความเร็ว 1Gbps (“Management” ในภาพที่ 4.1)

Data Network

เป็นเครือข่ายที่ใช้รับส่งข้อมูลในปริมาณมากและต้องการความเร็วในการรับส่งข้อมูลสูง โดยนำมาใช้ในการส่งข้อมูลของระบบ Message Passing Interface (MPI) ในการประมวลผล MPI Benchmark ของ VM บน VC ซึ่งถูกติดตั้งบนเครือข่ายความเร็ว 10Gbps (“Data” ในภาพที่ 4.1)

เครือข่าย Management และ Data จะมีโครงสร้างภายในที่เหมือนกัน โดยประกอบไปด้วย Virtual Switch อยู่ 2 switch ได้แก่

- External Bridge (man-br-ex สำหรับ Management Network และ data-br-ex สำหรับ Data Network ในภาพที่ 4.1) ทำหน้าที่เป็น GRE Tunneling และ Encapsulate GRE Packets ในการรับส่งข้อมูลระหว่าง Host
- Internal Bridge (man-br-int สำหรับ Management Network และ data-br-int สำหรับ Data Network ในภาพที่ 4.1) ทำหน้าที่เป็น GRE Endpoint เพื่อ De-Encapsulate GRE Tag จาก GRE Packets ที่ได้รับมาจาก External Bridge และส่งผ่านข้อมูลสู่ Tap Interface เพื่อส่งต่อให้กับ VM (ผ่าน Tap Interface จะถูกสร้างขึ้นเมื่อมีการสร้าง VM)

4.1.1.2 Virtual Machine

ในการวิจัยนี้ ทางผู้วิจัยได้ใช้ Hypervisor แบบ QEMU-KVM [12] เวอร์ชั่น 2.12.0 (Release เมื่อเดือนเมษายน 2561) ในการทดลองและตั้งค่าเครื่อง VM มีหน่วยประมวลผลเสมือน 2 vCPU และมีหน่วยความจำหลักขนาด 16GB และมี Network Interface Card (NIC) จำลอง อยู่ 2 NIC ซึ่ง NIC จำลองแต่ละอันจะใช้ TAP Interface (ดูภาพ 4.1 tap0 และ tap1) เชื่อมต่อ VM กับเครือข่าย Management และ Data Network ตามลำดับ (ดูภาพ 4.1 NIC1 และ NIC2)

สำหรับการเก็บข้อมูลของ VM ผู้วิจัยใช้ Disk Image แบบ QCOW2 (QEMU Copy On Write Version 2) ที่ประกอบไปด้วยไฟล์ (File) Disk Image 2 ชั้น ชั้นแรก คือ ไฟล์ Base Disk Image ติดตั้งระบบปฏิบัติการ Ubuntu Server OS เวอร์ชั่น 16.0.4 และ Software ที่ใช้สำหรับการทดลอง และชั้นถัดไปคือไฟล์ Overlay Disk Image เป็นการทำ External Copy on write layer ของไฟล์ Base Image ที่จะเก็บข้อมูลการเปลี่ยนแปลงของไฟล์ Base Image ในนั้น ซึ่งการเขียนข้อมูลใดๆลง Overlay Disk Image จะไม่ส่งผลกระทบต่อ Base Disk Image หาก VM เกิดการทำงานผิดพลาดหรือเกิดการเสียหายขึ้นบน Overlay Disk ผู้ใช้สามารถสร้าง Overlay Disk Image ขึ้นมาใหม่ได้

ในการทดลองผู้วิจัยได้สร้าง Base Image Overlay Disk Image ขึ้นมาทั้งหมดจำนวน 8 คู่ เพื่อใช้สำหรับ VM ทั้งหมด 8 เครื่อง บน VC ในแต่ละการทดลองผู้ใช้จะเก็บข้อมูลต่างๆจากการทดลองจำนวน 5 ครั้ง ในการทดลองแต่ละครั้ง Overlay Disk Image จะถูกสร้างใหม่ ในขณะที่ Base Image ยังคงเดิม

จากแนวคิด Virtual Time ของ VM ทางผู้วิจัยได้ตั้งค่าคุณลักษณะของการเปลี่ยนจังหวะเวลาของ Linux Guest OS ที่รัน QEMU-KVM ให้เป็นแบบ HPET (High Precision Event Timer) โดยไม่ใช้ KVM Clock หรือ Clock Synchronization กับเครื่องคอมพิวเตอร์จริง เพื่อให้ VM ทุกเครื่องบน VC มีเวลาเป็นของตัวเองโดยไม่ขึ้นอยู่กับ Host OS และได้มีการติดตั้ง NTP Server บน VM ทุกเครื่อง โดยมีเครื่องหนึ่งเป็น NTP Server และเครื่องที่เหลือเป็น NTP Client เพื่อประสานเวลาของ VM ทุกเครื่องให้มีเวลาที่ใกล้เคียงกันมากที่สุด ทำให้เกิดเป็นเวลาเสมือนของ VC และเป็นไปตามแนวคิด Virtual Time ในที่สุด

นอกจากนั้น ผู้วิจัยยังได้ตั้งค่าให้ QEMU-KVM เปิดใช้งานฟังก์ชัน HMP Monitor (Human Monitor Protocol) เพื่อให้ Agent สามารถส่งคำสั่งเข้าไปยัง Hypervisor เพื่อสั่งงาน VM ได้โดยตรงผ่านไฟล์ Unix Socket

จากภาพ 4.1 บนเครื่องคอมพิวเตอร์จริงแต่ละเครื่องจะมี Agent คอยบริหารจัดการ VM จำนวน 4 เครื่อง และ จะมี Coordinator ที่ติดตั้งอยู่บนเครื่องคอมพิวเตอร์จริงหนึ่งเครื่อง (Host1) เพื่อสั่งงาน Agent บนเครื่องคอมพิวเตอร์จริงทุกเครื่องผ่าน Management Network

4.2 การทดลอง TCP Backoff ด้วยโปรแกรม iPerf

โปรแกรม iPerf เป็น โปรแกรมเครื่องมือ Open Source ที่ใช้ในการวัดประสิทธิภาพของเครือข่ายแบบ TCP และ UDP ซึ่งสามารถวัดค่า Latency, Jitter และ การสูญหายของ Packets โดยสามารถกำหนดปริมาณของข้อมูล และระยะเวลาในการรับส่งข้อมูลได้

ผู้วิจัยต้องการค้นคว้าเกี่ยวกับการเกิดปัญหา TCP Backoff หลังจากการเริ่มต้นการทำงานใหม่จากการหยุดการทำงานของ VM จึงได้ออกแบบการทดลองโดยการใช้โปรแกรม iPerf ส่งข้อมูล TCP Packets แบบ Dual Test หรือ การสื่อสารแบบ 2 ทิศทาง บนเครือข่ายความเร็วสูง 10Gbps เพื่อวัดประสิทธิภาพของ Client ทั้งขาไปและขากลับ โดยทำการส่งข้อมูลแบบเต็ม Bandwidth เพื่อให้ตามสมมติฐาน กล่าวคือ ภายในเครือข่ายจะต้องมีการรับส่งข้อมูลในปริมาณมากอยู่ตลอดเวลา

ในการทดลอง ผู้วิจัยจะทำการรันโปรแกรม iPerf บน VM 2 เครื่อง โดยที่เครื่องหนึ่งทำหน้าที่เป็น Server และอีกเครื่องหนึ่งเป็น Client หลังจากนั้น ให้ทำการหยุดการทำงานของ VM พร้อมกัน หลังจากนั้นให้ VM ทั้งสองกลับมาเริ่มต้นการทำงานต่อจากจุดเดิมในเวลาที่แตกต่างกันโดยกำหนดเงื่อนไขดังต่อไปนี้

4.2.1 การหน่วงเวลาของ Client

หลังจากการหยุดการทำงานของ Server และ Client ผู้วิจัยได้เริ่มต้นการทำงานของ Server ต่อทันที แต่จะทำการหน่วงเวลาในการเริ่มต้นการทำงานต่อของ Client ชั่วขณะหนึ่งตามเวลาที่กำหนด โดยกำหนดให้มีการหน่วงเวลาเป็นเวลา 1, 2, 4, 8 และ 16 วินาที แล้วจึงอนุญาตให้ Client เริ่มต้นการทำงานต่อ

จากผลการทดลอง เมื่อมีการหน่วงเวลาของ Client เพื่อให้เริ่มต้นการหลังจากการทำงานของ Server ปรากฏว่า ทันทีที่ Server กลับมาทำงานต่อ โปรแกรม iPerf ยังคงหยุดการทำงานต่อและไม่สามารถทำงานต่อได้ จนกระทั่ง Client กลับมาทำงานต่อหลังจากการหน่วงเวลา

4.2.2 การหน่วงเวลาของ Server

หลังจากการหยุดการทำงานของ Server และ Client ผู้วิจัยได้เริ่มต้นการทำงานของ Client ต่อทันที แต่จะทำการหน่วงเวลาในการเริ่มต้นการทำงานต่อของ Server ชั่วขณะหนึ่งตามเวลาที่กำหนด โดยกำหนดให้มีการหน่วงเวลาเป็นเวลา 1, 2, 4, 8 และ 16 วินาที แล้วจึงอนุญาตให้ Server เริ่มต้นการทำงานต่อ

จากผลการทดลอง เมื่อมีการหน่วงเวลาของ Server เพื่อให้เริ่มต้นการทำงานช้ากว่า Client ปรากฏว่า หลังจากที่ Client กลับมาทำงานต่อแต่ Server ยังหยุดการทำงานอยู่ โปรแกรม iPerf ไม่สามารถส่งข้อมูลไปยัง Server ได้ หลังจากนั้นเมื่อให้ Server กลับมาทำงานต่อหลังจากการหน่วงเวลา โปรแกรม iPerf บนฝั่ง Client ไม่สามารถกลับมาทำงานต่อได้ทันทีแต่ต้องรอเป็นเวลา 1, 2, 4, 8 และ 16 วินาทีโดยประมาณ ตามระยะเวลาของการหน่วงเวลาในการทดลองก่อนที่นั้นโปรแกรม iPerf จะกลับมารับส่งข้อมูลต่อได้อย่างปกติ

จากการสังเกตผลการทดลองทั้งสองแบบพบว่า เมื่อให้ Client เริ่มต้นทำงานช้ากว่า Server ไม่พบว่าเกิดปัญหา TCP Backoff ในการทำงานของโปรแกรม iPerf เนื่องจาก ไม่มีการส่ง TCP Packets ใดส่งเข้ามายัง Server ทำให้ช่วงเวลาที่ Client หายไป

ในทางตรงกันข้าม เมื่อ Server เริ่มต้นทำงานช้ากว่า Client จะเกิดความล่าช้าในการรับส่งข้อมูลของโปรแกรม iPerf เนื่องจาก Client ทำการส่งข้อมูล TCP Packets ไปยัง Server แต่ไม่มีผู้รับ จึงเกิด Packet Loss แต่ Client ยังคงทยอยส่ง TCP Packet ต่อไปถึงแม้ว่าจะยังไม่ได้รับ ACK จากฝั่ง Server และในที่สุด เมื่อหมดเวลา Timeout ในการรอ Packet และ ยังไม่ได้รับ ACK ตอบกลับ Client จะหยุดรอเป็นระยะเวลาสองเท่าของ Timeout เดิมที่ Kernel กำหนดไว้ หลังจากนั้นจะทำการส่ง TCP Packet ดังกล่าวซ้ำ พร้อมกับลดค่า Window Size ลงและทำซ้ำต่อไปเรื่อยๆ ซึ่ง Timeout นั้นขึ้นอยู่กับค่า Round-Trip Time (RTT) ในการรับส่งข้อมูลของ TCP Packet ก่อนหน้า [11] หากข้อมูล Packet ที่รับส่งในขณะนั้นมีขนาดอยู่ในระดับของ Max Window size ทำให้ข้อมูลที่รับส่งมีขนาดใหญ่และใช้เต็ม Bandwidth จึงมีระยะเวลา Timeout มากขึ้นตามไปด้วย หลังจากที่ Server ถูกเรียกให้เริ่มต้นการทำงานต่อ Server จะส่ง ACK ต่อจาก Packet เดิมที่เคยได้รับ ก่อนที่จะถูกหยุดการทำงาน เมื่อฝั่ง Client ได้รับ ACK จึงทยอยส่ง Packet ดังกล่าว และค่อยๆเพิ่มค่า Window Size จนถึงระดับที่ส่งได้ก่อนหน้า

จากการวิเคราะห์ผลการทดลองพบว่า ระยะเวลา Timeout ที่ Client รอในการส่ง Packets นั้น เป็นระยะเวลาเท่ากับระยะเวลาของการหน่วงการเริ่มต้นการทำงานของ Server กล่าวคือ หากมีการหน่วงเวลาการเริ่มต้นทำงานของ Server เป็นเวลา 1, 2, 4, 8 และ 16 วินาที จะเกิด TCP Backoff เป็นเวลา 1, 2, 4, 8 และ 16 วินาที ตามลำดับ สรุปได้ว่าเมื่อฝั่ง Server หรือ ผู้รับ

ถูกเริ่มต้นการทำงานหลังจาก Client หรือ ผู้ส่ง จะทำให้เกิดปัญหา TCP Backoff เนื่องจาก Retransmission Protocol ซึ่งเป็นระยะเวลาเท่ากับระยะเวลาเริ่มต้นการทำงานที่แตกต่างกันระหว่าง Client และ Server

ข้อเท็จจริงที่ค้นพบจากการทดลองนี้ทำให้ผู้วิจัยออกแบบ Checkpoint และ Restart Protocol โดยใช้ Barrier Synchronization เพื่อทำให้เริ่มต้นการทำงานของ VM ที่เป็นทั้งผู้ส่งและผู้รับ เริ่มต้นการทำงานในเวลาที่ใกล้เคียงกันมากที่สุดเพื่อลดปัญหา TCP Backoff

4.3 NAS Parallel Benchmark

ผู้วิจัยได้ใช้ NAS Parallel Benchmark (NPB) เป็น Application ที่ทำงานอยู่บนระบบ MPICH NAS Parallel Benchmark เป็นโปรแกรมที่พัฒนาโดยองค์กร NASA Advanced Supercomputing (NAS) ซึ่งเป็นโปรแกรมมาตรฐานในการวัดประสิทธิภาพของ Parallel Supercomputers [13] ในขณะที่ MPICH [14] คือ Software Library Implementation ที่พัฒนาขึ้นโดย Argonne National Laboratory เพื่อใช้สำหรับสร้างโปรแกรมแบบ Parallel Processing ที่ใช้ API มาตรฐาน เรียกว่า Message Passing Interface (MPI) ซึ่งในวิทยานิพนธ์นี้ทางผู้วิจัยจะเรียก Application ที่ทำงานบน MPICH ว่า MPI Application

โปรแกรม NPB มีอยู่หลายชนิดเพื่อวัดรูปแบบของการประมวลผลที่แตกต่างกัน ทางผู้วิจัยได้เลือกมาทั้งหมด 4 โปรแกรม เพื่อนำมาใช้ในการทดสอบประสิทธิภาพของการทำ Checkpointing บน VC คือ Block Tri-diagonal solver (BT), Integer Sort (IS), Multi-Grid (MG) และ Fast Fourier Transform (FT) และได้เลือกขนาดของปัญหาของโปรแกรม BT ที่ใช้ในการทดลอง 2 ชนิด คือ Class C เป็นขนาดมาตรฐานของปัญหาในการทดสอบ Benchmark และ Class D ที่มีขนาดของปัญหาใหญ่กว่า Class C ถึง 16 เท่า

- Block Tri-diagonal solver (BT) เป็นโปรแกรมแก้ไขปัญหา Synthetic System of Nonlinear PDEs โดยใช้อัลกอริทึม Block Tri-diagonal สาเหตุที่เลือกเพราะทางผู้วิจัยต้องการวัดประสิทธิภาพของการ Checkpointing ด้วยการเพิ่มประมาณของปัญหาระหว่าง Class C และ D
- Integer Sort (IS) เป็นโปรแกรมเรียงชุดตัวเลขโดยใช้อัลกอริทึมแบบ Bucket sort ใช้ทดสอบประสิทธิภาพของความเร็วในการคำนวณตัวเลขและความเร็วในการสื่อสารระหว่าง Node ซึ่งมีการใช้งาน Bandwidth บนเครือข่ายสูง ถูกนำมาใช้เพื่อวัด

ประสิทธิภาพในการแก้ไขปัญหา TCP Backoff ในกรณีที่มีปริมาณของข้อมูลบนเครือข่ายหนาแน่น

- Multi-Grid (MG) เป็นโปรแกรมแก้ไขปัญหา Three-dimensional Discrete Poisson Equation ที่ออกแบบมาให้มีการสื่อสารที่มีความซับซ้อนสูง โดยมีทั้งแบบระยะใกล้และระยะไกล ใช้ทดสอบประสิทธิภาพของการสื่อสารระหว่าง Node ถูกนำมาใช้เพื่อวัดประสิทธิภาพในการทำงานของ Barrier Synchronization และแก้ไขปัญหา TCP Backoff ในกรณีที่มีจำนวนของการสื่อสารเป็นจำนวนมากและมีความซับซ้อนสูง
- Fast Fourier Transform (FT) เป็นโปรแกรมแก้ไขปัญหา three-dimensional partial differential equation (PDE) โดยใช้อัลกอริทึม fast Fourier transform (FFT) โดยเป็นการแก้ไขปัญหาทดสอบความแม่นยำและเป็นการสื่อสารระยะไกล และยังมีการใช้ Memory ในปริมาณสูง จึงถูกนำมาใช้ในการวัดประสิทธิภาพในการ Checkpointing ของ VC ที่ต้องอาศัยความเร็วการอ่านและเขียนข้อมูลจาก Disk และวัดประสิทธิภาพในการทำงานของ Barrier Synchronization และแก้ไขปัญหา TCP Backoff ในกรณีที่มีปริมาณในการสื่อสารเป็นจำนวนมาก

ในการทดสอบ NPB ดังกล่าว ผู้วิจัยได้กำหนดให้ VC มี VM เป็นจำนวน 8 เครื่อง และได้ติดตั้ง MPICH และ NPB ไว้บน VM ทุกเครื่อง โดยกำหนดให้ VM จำนวน 1 เครื่องทำหน้าที่เป็น Head Node เพื่อคอยรับคำสั่งการประมวลผล Benchmark และกระจายงานไปยัง VM เครื่องอื่นทั้งหมด โดยผู้วิจัยได้ออกแบ่งการทดลองออกเป็น 3 ส่วน คือ โปรแกรม iPerf เป็น โปรแกรมเครื่องมือ Open Source ที่ใช้ในการวัดประสิทธิภาพ

4.3.1 การทดสอบการทำงานของ Benchmark โดยไม่มีการทำ Checkpoint ใดๆ บน VC

ผู้วิจัยได้ทำการทดลองเพื่อวัดค่าต่างๆของการทำงานของ NPB ในสภาพแวดล้อมของเครื่องคอมพิวเตอร์จริงที่ผู้วิจัยใช้ในการทำการทดลอง เพื่อเป็นอ้างอิงในการทำการ Checkpoint และ Restart ซึ่งในการทดลองจะจำแนกออกเป็น 2 ประเด็น คือ การวัดการใช้ทรัพยากรในการรัน NPB บน VC และ การวัดเวลาในการรัน NPB บน VC

NAS Parallel Benchmark	Memory Usage of VM (MB)	Peak Bandwidth (Gbps)	Execution Time (sec)
BT.C.16	333.488	0.184	129.042
BT.D.16	3,441.628	0.216	2019.745
IS.D.16	3,290.084	2.816	116.586
MG.D.16	3,654.488	0.264	167.128
FT.D.16	14,814.040	3.920	1035.050

ตาราง 1.1 ผลการทดลองการทำงานของ Benchmark โดยไม่ทำ Checkpointing และ การวัดการใช้ทรัพยากรในการประมวลผลของ NAS Parallel Benchmark บน VC

จากตาราง 1.1 คอลัมน์ที่ 1 แสดงถึง NPB ที่ใช้ในการทดลอง, Memory Usage of VM แสดงถึง จำนวนการใช้งาน Memory ทั้งหมดบน VM ที่ทำหน้าที่เป็น Head Node โดยใช้คำสั่ง “free” utility บนระบบบ Linux [15] มีหน่วยเป็น Megabyte, Peak Bandwidth แสดงถึงปริมาณ Bandwidth สูงสุดของการสื่อสารบน Data Network บนเครื่อง VM ที่ทำหน้าที่เป็น Head Node ที่ได้จากการรายงานผลข้อมูลด้วย “bmon” utility บนระบบ Linux [16] มีหน่วยเป็น Gbps และ Execution Time คือเวลาทั้งหมดที่ใช้ในการประมวลผลโปรแกรม MPI Application Benchmark โดยคำนวณจากการหาค่าเฉลี่ยของระยะเวลาในการประมวลผลโปรแกรม 5 ครั้ง

- Block Tri-diagonal solver (BT) - เนื่องจาก BT Class C และ BT Class D เป็น MPI Application Benchmark แบบเดียวกัน แต่ใช้ปริมาณของปัญหาที่แตกต่างกัน จากผลการทดลองจะสังเกตได้ว่า Class D ได้ใช้หน่วยความจำและปริมาณการสื่อสารเพิ่มมากขึ้นจาก Class C คือ 333MB และ 0.184Gbps เป็น 3.4GB และ 0.216Gbps ตามลำดับ ทำให้ระยะเวลาในการประมวลผล MPI Application ของ BT Class D ใช้เวลานานขึ้น จาก 129.042 วินาที เป็น 2019.745 วินาที
- Integer Sort (IS) - IS เป็น MPI Application Benchmark ที่ใช้ปริมาณในการสื่อสารระหว่าง Node สูง จากผลการทดลองโปรแกรม IS จะเห็นได้ว่ามีปริมาณการใช้งาน Bandwidth สูงถึง 2.816Gbps
- Multi-Grid (MG) - จากผลการทดลองโปรแกรม MG ซึ่งเป็น MPI Application ที่มีความซับซ้อนในการสื่อสารสูง ใช้หน่วยความจำในการประมวลผล 3.6GB

แต่สังเกตได้ว่ามีการใช้ปริมาณการสื่อสารไม่มาก โดยใช้ปริมาณการสื่อสารสูงสุดที่ 0.264Gbps และใช้เวลาในการประมวลผล 167.128 วินาที

- Fast Fourier Transform (FT) - จากผลการทดลองสังเกตได้ว่า VM ที่ใช้ประมวลผล FT มีการใช้ทรัพยากรในการประมวลผลอย่างมาก และ ใช้ทรัพยากรมากที่สุด โดยใช้หน่วยความจำ 14.8GB และส่งข้อมูลผ่านเครือข่ายสูงสุดที่ 3.92Gbps แต่ใช้เวลาในการประมวลผลเพียง 1035.05 วินาที

4.3.2 การวัด Overhead ในการ Checkpoint ด้วยทำการ Checkpoint

ในระหว่างการทำงานของ Benchmark ผู้วิจัยจะทำการ Checkpoint 1 ครั้ง ในจุดที่การทำงานของ MPI Application Benchmark ดำเนินการได้ 70% ของการทำงานทั้งหมด โดยอ้างอิงจากเวลาเฉลี่ยของการทำงานของ MPI Application Benchmark จากการทดลองที่แล้ว เนื่องจากจุด Checkpoint ดังกล่าว เป็นเวลาที่โปรแกรมมีการทำงานคงที่และใช้ทรัพยากรอย่างเต็มที่ ทำให้จุดเวลาดังกล่าวเหมาะสมในการนำมาใช้เป็นจุด Checkpoint ในการทดลอง

การวัด Overhead ของ Checkpoint Protocol จะแบ่งออกเป็น 2 ประเด็น คือ ระยะเวลาของแต่ละ Stage ในการทำ Checkpoint Protocol และ Overhead ของการประมวลผล MPI Application Benchmark

4.3.2.1 ระยะเวลาของ Stage ในการทำ Checkpoint Protocol

จากการทดลองการทำ Checkpointing จะทำการวัด Overhead ที่เกิดจากระยะเวลาการทำงานของ State ใน Checkpoint Protocol ประกอบไปด้วย Initialize, Stop&Save และ Resume ซึ่งระยะเวลาดังกล่าวคำนวณจากระยะเวลาเริ่มต้น State ไปจนถึงเปลี่ยน State โดยเฉลี่ยจากการทำการทดลอง 5 ครั้ง

NAS Parallel Benchmark	Checkpoint State		
	Initialize (ms)	Stop&Save (sec)	Resume (ms)
BT.C.16	0.8	55.082	0.2
BT.D.16	1.5	210.333	0.2
IS.D.16	1.1	170.978	0.2
MG.D.16	1.1	149.033	0.2
FT.D.16	1.2	686.398	0.3

ตาราง 1.2 ระยะเวลาในการทำงานของสถานะต่างๆของ Checkpoint Protocol

จากตาราง 1.2 แสดงผลการทดลองของการทำ Checkpointing 1 ครั้งบน VC ตาม NAS Parallel Benchmark ที่กำหนดไว้ ในคอลัมน์ NAS Parallel Benchmark แสดงถึงรายชื่อของโปรแกรม MPI Application Benchmark ที่ใช้ในการทดลอง สำหรับ Initialize, Stop&Save และ Resume แสดงถึง ระยะเวลาในการทำงานของ Initialize State มีหน่วยเป็น มิลลิวินาที, ระยะเวลาในการทำงานของ Stop&Save State มีหน่วยเป็น วินาที และ ระยะเวลาในการทำงานของ Resume State มีหน่วยเป็น มิลลิวินาที ตามลำดับ

จากการทดลองพบว่า ระยะเวลาในการทำงานของ Initialize State ของ BT Class C ใช้เวลาน้อยที่สุดอยู่ที่ 0.8ms ตามด้วย IS Class D และ MG Class D ใช้เวลาเท่ากัน คือ 1.1ms ถัดมาคือ FT Class D ใช้เวลา 1.2ms และสุดท้าย BT Class D ใช้เวลามากที่สุดที่ 1.5ms ตามลำดับ สำหรับระยะเวลาในการทำงานของ Stop&Save State พบว่า BT Class C ใช้เวลาน้อยที่สุด ตามด้วย MG Class D, IS Class D, BT Class D และ FT Class D ใช้เวลามากที่สุด ตามลำดับ และสุดท้ายระยะเวลาในการทำงานของ Resume State พบว่า BT Class C, BT Class D, IS Class D, MG Class D ใช้เวลาเท่ากันคือ 0.2ms ส่วน FT.D ใช้เวลามากกว่าเล็กน้อยที่ 0.3ms จะเห็นได้ว่า Checkpoint Protocol จะใช้เวลาส่วนใหญ่ไปกับ Stop&Save State เนื่องจาก ต้องใช้เวลาในการบันทึก State ลงบน Disk ซึ่งขึ้นอยู่กับขนาดของ Memory ที่ VM กำลังใช้งานอยู่ในขณะนั้น

4.3.2.2 Overhead ของการประมวลผล NAS Parallel Benchmark

ในการวัด Overhead ของการประมวลผล NAS Parallel Benchmark จะวัดจากระยะเวลาที่เพิ่มขึ้นของการประมวลผล MPI Application Benchmark จากไม่มีการทำ Checkpointing เปรียบเทียบกับการทำ Checkpointing

NAS Parallel Benchmark	Wall Clock Time (sec)		Checkpoint Time (sec)	Overhead (sec)
	Normal	Checkpoint		
BT.C.16	129.042	186.514	55.084	57.472
MG.D.16	167.128	322.241	149.034	155.113
IS.D.16	116.586	280.436	170.980	163.850
BT.D.16	2019.745	2340.636	210.334	320.891
FT.D.16	1035.050	1738.953	686.399	703.903

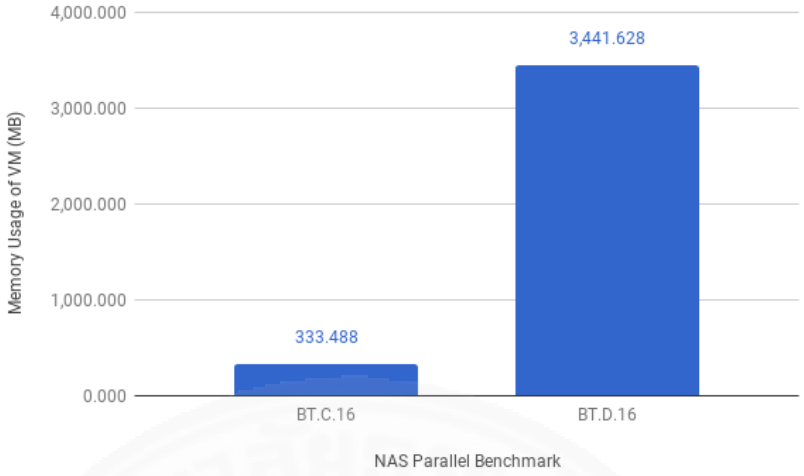
ตาราง 1.3 Overhead ในการทำ Checkpoint Protocol

จากตาราง 1.3 แสดง Overhead ในการทำ Checkpoint Protocol โดย Wall Clock Time คือ ระยะเวลาในการประมวลผล MPI Application Benchmark มีหน่วยเป็นวินาที ซึ่งประกอบไปด้วย 2 แบบ ได้แก่ แบบ Normal คือ การทำงานแบบปกติโดยไม่มีการ Checkpointing และแบบ Checkpoint คือ มีการทำ Checkpointing 1 ครั้ง และค่า Overhead คือ ค่า Overhead ของการประมวลผลที่เพิ่มขึ้นจากการทำ Checkpointing และสำหรับ Checkpoint Time คือ เวลาที่ใช้ในการ Checkpointing คำนวณจากเวลารวมทั้งหมดของทุก Stage ใน Checkpoint Protocol ซึ่งในค่าตัวเลขทั้งหมดนี้คำนวณมาจากค่าเฉลี่ยของการทดลอง 5 ครั้ง

จากผลการทดลองนี้ สามารถนำผลการทดลองมาวิเคราะห์ได้เป็น 3 ประเด็น คือ การเพิ่มขนาดของปัญหาในการประมวลผลโปรแกรม BT Class C และ Class D, ปัจจัยที่ส่งผลต่อ Overhead ในการทำ Checkpoint และ การพิสูจน์ความถูกต้องในการเก็บข้อมูลของ Checkpoint Protocol

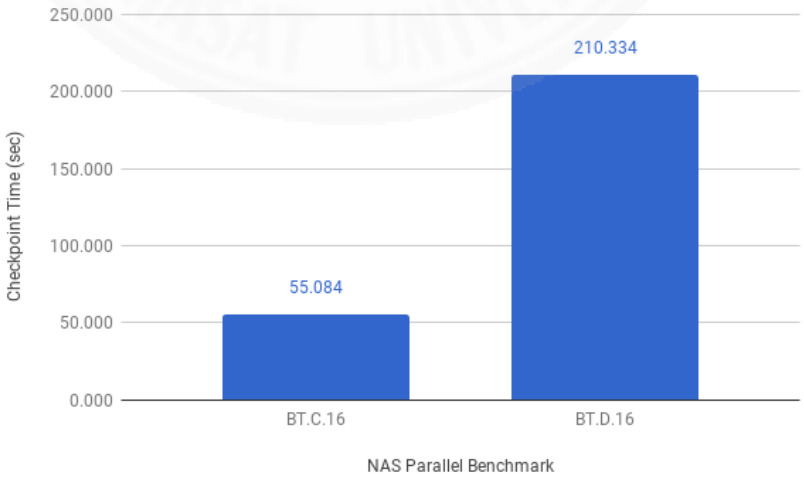
4.3.2.3 ผลกระทบจากการเพิ่มขนาดของปัญหาในการประมวลผลโปรแกรม BT Class C และ Class D

จากการทดลองเพิ่มขนาดของปัญหาในการใช้ด้วยการใช้โปรแกรม BT จากการสังเกตผลการทดลองด้วยการเพิ่มขนาดของปัญหาโดยใช้ Class C และ D ปรากฏว่า การเพิ่มขนาดของปัญหา ทำให้ VM ต้องใช้งาน Memory เพิ่มมากขึ้น ส่งผลกระทบต่อระยะเวลา Checkpoint Time ของ Checkpoint Protocol โดยตรง



ภาพที่ 4.2 เปรียบเทียบการใช้งาน Memory ของ VM ระหว่าง BT Class C และ D

ในการประมวลผลโปรแกรม BT Class C บน VM ใช้ Memory ขนาด 333MB (ดูภาพที่ 4.2) และเมื่อเพิ่มขนาดของปัญหาเป็น Class D ทำให้ขนาดของ Memory ที่ VM ต้องการใช้ในการประมวลผลเพิ่มขึ้นเป็น 3.4GB ซึ่งมากกว่าเดิมประมาณ 10 เท่า ส่งผลทำให้ Checkpoint Protocol ใช้เวลาเพิ่มมากขึ้นจาก 55.08 วินาที เป็น 210.33 วินาที (ดูภาพที่ 4.3) เนื่องจาก QEMU-KVM ต้องทำการ Snapshot เครื่อง VM ด้วยการ Dump ข้อมูลจาก CPU State, Memory และ Disk หากยิ่งมีการใช้ Memory มาก ต้องอาศัยเวลาในการทำ Snapshot ด้วยเก็บข้อมูลจาก Memory มากขึ้นตามไปด้วย



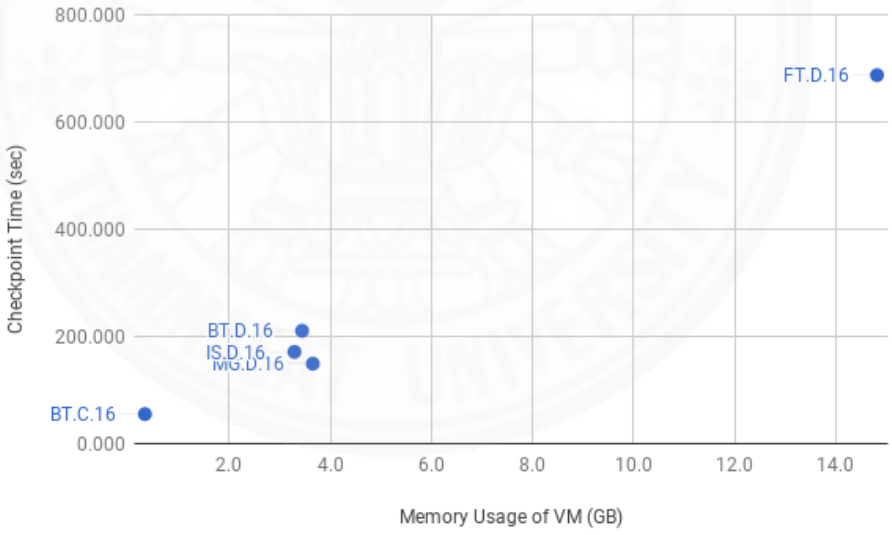
ภาพที่ 4.3 เปรียบเทียบเวลา Checkpoint Time ระหว่าง BT Class C และ D

จากข้อมูลสรุปได้ว่า การเพิ่มประมาณงานในการทำงานโดยใช้อัลกอริทึมแบบเดียวกัน ส่งผลให้เกิด Overhead ในการ Checkpoint เพิ่มขึ้นได้ เนื่องจากปริมาณการใช้งานของ Memory ของ VM เป็นปัจจัยที่ส่งผลโดยตรงกับค่า Overhead ในการ Checkpointing

4.3.2.4 ปัจจัยที่ส่งผลต่อ Overhead ในการทำ Checkpointing

จากการทดลองการเพิ่มขนาดของปัญหาในการประมวลผล MPI Application Benchmark ทำให้ทราบว่า Memory ในการใช้งานของ VM ส่งผลต่อระยะเวลาในการ Checkpointing ผู้วิจัยจึงได้วิเคราะห์หาปัจจัยที่ส่งผลต่อระยะเวลาในการ Checkpointing มากที่สุด และพบว่ามี 2 ปัจจัยที่ส่งผลต่อ Checkpoint Time ได้แก่ ปริมาณการใช้งาน Memory ของ VM และ ขนาดของ Snapshot ของ VM

(1) ปริมาณการใช้งาน Memory ของ VM

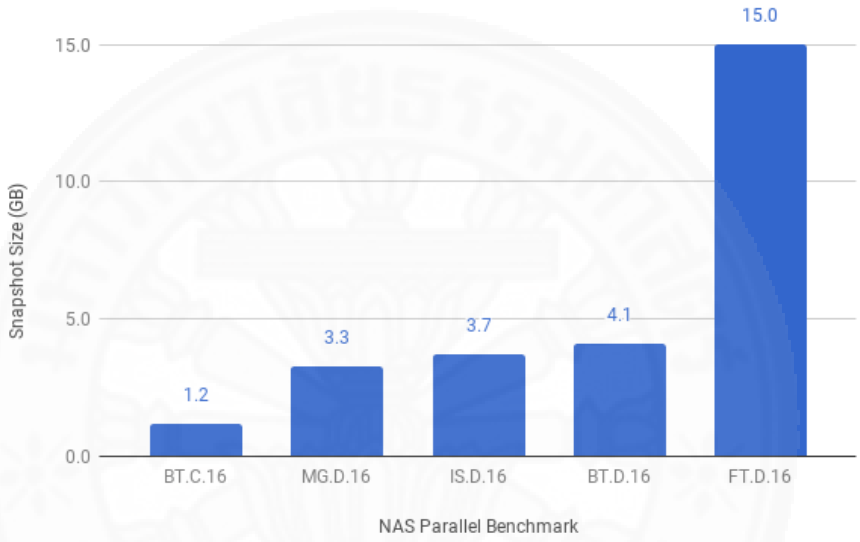


ภาพที่ 4.4 แสดงความสัมพันธ์ระหว่าง ปริมาณการใช้งาน Memory ของ VM และ Checkpoint Time

จากผลการทดลองพบว่า เวลา Checkpoint Time ของทุกโปรแกรม MPI Application Benchmark ไม่ได้สัมพันธ์กับ ขนาดของ Memory ที่ใช้ในการประมวลผล VM จากแผนภาพ 4.4 เห็นได้ว่า โปรแกรม MG Class D ใช้ Memory สูงกว่า โปรแกรม BT Class D และ IS Class D แต่ใช้เวลาในการทำ Checkpointing น้อยกว่า จากข้อมูลจึงสรุปได้ว่า ปริมาณการใช้

Memory ในการประมวลผลโปรแกรมเพิ่มมากขึ้น ไม่ได้ส่งผลให้ใช้เวลา Checkpoint Time เพิ่มมากขึ้นอย่างเห็นได้ชัด เนื่องจากข้อสังเกตดังกล่าว ผู้วิจัยจึงได้หันมาวิเคราะห์ปัจจัยขนาดของ Snapshot ของ VM ในอันดับถัดไป

(2) ขนาดของ Snapshot ของ VM ส่งผลต่อ Overhead ในการทำ Checkpointing

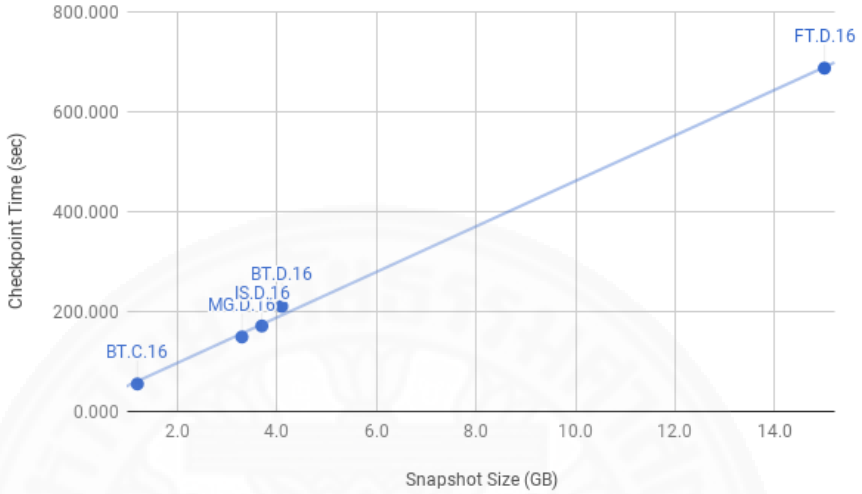


ภาพที่ 4.5 แสดงค่า Snapshot Size ของ MPI Application Benchmark

ปัจจัยสำคัญที่ส่งผลต่อขนาดของ Snapshot คือ ขนาดของ Memory ที่ VM กำลังใช้งานอยู่ แต่มีอีกปัจจัยหนึ่งที่ส่งผลต่อขนาดของ Snapshot คือ State ของ Disk Image ที่ VM กำลังประมวลผลอยู่ ซึ่งการเปลี่ยนแปลงข้อมูลใน Disk Image ของ VM มักเกิดจากพฤติกรรมการทำงานของ Process ที่ทำงานอยู่ภายใน VM และ ข้อมูลการสื่อสารบนเครือข่าย

จากแผนภาพที่ 4.5 จะเห็นได้ว่าขนาดของ Snapshot Size ของ IS.D และ BT.D มีขนาดมากกว่าของ MG.D อย่างเห็นได้ชัด ทุกครั้งที่ QEMU-KVM สั่งให้ VM ทำการ Snapshot สิ่งที่จะถูกเก็บลงบน Overlay Disk Image ที่ VM กำลังใช้งานอยู่ คือ ข้อมูล CPU State, Memory และ Disk State ซึ่งการเก็บข้อมูลของ Disk State ของ Disk Image ชนิด QCOW2 เรียกว่า การทำ Internal Snapshot ซึ่งเป็นการเก็บข้อมูลที่มีการเปลี่ยนแปลงจาก Original Overlay Disk ที่สร้างขึ้นมาเพื่อบันทึกลงบน Disk ซ้อนลงไปอีกระดับ ซึ่งจะถูกสำรองไว้ภายใน Disk

Image เดียวกัน ในลักษณะของการทำ Copy On Write จึงทำให้ Disk Image ที่ VM กำลังประมวลผลอยู่มีขนาดเพิ่มขึ้นตามไปด้วย



ภาพที่ 4.6 แสดงความสัมพันธ์ของ ขนาดของ Snapshot และ Checkpoint Time

เมื่อนำขนาดของ Snapshot ของแต่ละโปรแกรมมาวิเคราะห์ ผลปรากฏว่าขนาดของ Snapshot คลอดคล้องกับเวลาในการทำ Checkpointing จากแผนภาพที่ 4.6 โปรแกรม BT Class D, MG, IS BT Class D และ FT มีขนาดของ Snapshot เรียงลำดับจากน้อยไปมากตามลำดับ และยังใช้เวลาในการ Checkpoint เรียงลำดับจากน้อยไปมากตามลำดับเช่นเดียวกัน จากข้อมูลสรุปได้ว่า ขนาดของ Snapshot เป็นปัจจัยหลักที่ส่งผลต่อเวลาในการทำ Checkpointing โดยที่เวลาในการทำ Checkpointing จะแปรผันตรงตามขนาดของ Snapshot

4.3.2.5 ความถูกต้องในการเก็บข้อมูลของ Checkpoint Protocol บน VC

จากการทดลองทำ Checkpointing บน VC ด้วยการใช้ Checkpoint Protocol ที่ผู้วิจัยนำเสนอ บนระบบ A Coordinated Checkpointing Framework โดยการสั่งให้ทำ Checkpointing บน VC ในระหว่างการประมวลผล MPI Application Benchmark พบว่าสามารถทำงานได้อย่างถูกต้อง เนื่องจาก VC สามารถกลับมาทำงานต่อได้ และสามารถประมวลผลโปรแกรม MPI Application Benchmark ได้เสร็จสิ้นตามเวลาที่เหมาะสม

จึงสามารถพิสูจน์ได้ว่า Checkpoint Protocol สามารถเก็บข้อมูลของ Checkpoint VC ได้อย่างถูกต้อง เนื่องจาก สามารถทำงานได้เสร็จสิ้นตามเวลาที่เหมาสมเมื่อ

เปรียบเทียบกับระยะเวลาการทำงานแบบปกติ ของ MPI Application Benchmark และ สามารถพิสูจน์ได้ว่า Checkpoint Protocol สามารถจัดการการสื่อสารระหว่าง VM ภายใน VC ได้อย่างถูกต้อง เนื่องจาก VC สามารถกลับมาทำงานได้อย่างต่อเนื่องโดยไม่มีปัญหาในการสื่อสารข้อมูล หลังจากการทำ Checkpointing ทำให้พิสูจน์ได้ว่าแนวคิด Barrier Synchronization สามารถนำไปใช้บน Checkpoint Protocol ได้

4.3.3 การวัด Overhead ในการ Restart จากจุด Checkpoint

หลังจากที่ได้ทดลองทำการ Checkpoint การทำงานของ NAS Parallel Benchmark บน VC ในจุดที่การประมวลผลได้ประมาณ 70% ของการทำงานทั้งหมดในการทดลอง Checkpointing ในการทดลองนี้ ผู้วิจัยจะทำการเริ่มต้นการทำงานของ VC ด้วย Restart Protocol จากจุด Checkpoint ดังกล่าว เพื่อวัดค่า Overhead ของการทำงานของ Restart Protocol

4.3.3.1 ระยะเวลาของ Stage ในการทำ Restart Protocol

จากการทดลองการทำ Restart จะทำการวัด Overhead ที่เกิดจากระยะเวลาการทำงานของ State ใน Restart Protocol ประกอบไปด้วย Initialize, Load และ Resume ซึ่งระยะเวลาดังกล่าวคำนวณจากระยะเวลาเริ่มต้น State ไปจนถึงเปลี่ยน State โดยเฉลี่ยจากการทำการทดลอง 5 ครั้ง

NAS Parallel Benchmark	Restart State		
	Initialize (ms)	Load (sec)	Resume (ms)
BT.C.16	0.563	5.521	0.175
BT.D.16	0.557	15.439	0.226
IS.D.16	0.514	15.672	0.198
MG.D.16	0.525	13.055	0.198
FT.D.16	0.596	100.972	0.232

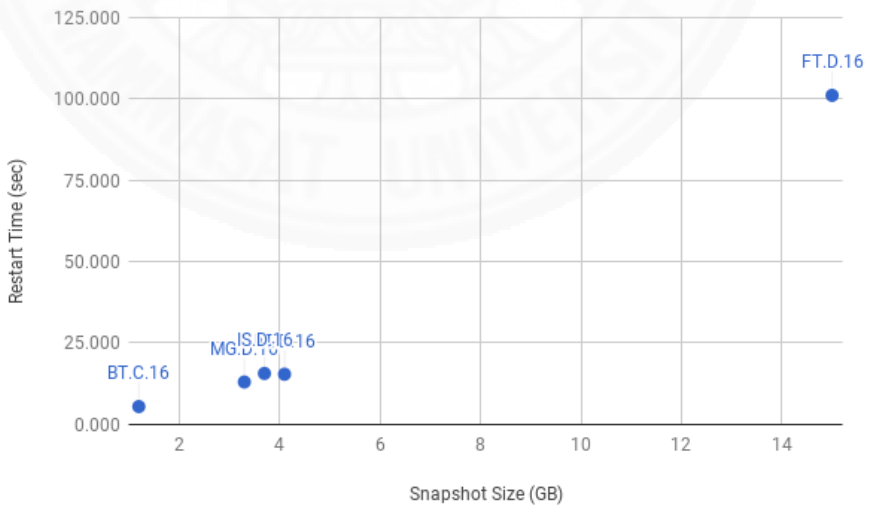
ตาราง 1.4 ระยะเวลาในการทำงานของแต่ละ State ของ Restart Protocol

จากตาราง 1.4 แสดงผลการทดลองของการทำ Restart จากจุด Checkpoint ที่กำลังประมวลผล MPI Application Benchmark ได้ประมาณ 70% ในคอลัมน์ NAS Parallel Benchmark แสดงถึงรายชื่อของโปรแกรม Benchmark ที่ใช้ในการทดลอง และสำหรับ

หลัก Initialize, Load และ Resume แสดงถึง ระยะเวลาในการทำงานของ Initialize State มีหน่วยเป็นมิลลิวินาที, ระยะเวลาในการทำงานของ Stop&Save State มีหน่วยเป็นวินาที และ ระยะเวลาในการทำงานของ Resume State มีหน่วยเป็นมิลลิวินาที ตามลำดับ

จากผลการทดลองสังเกตได้ว่าเวลาส่วนใหญ่ถูกใช้ไปกับ Load State เนื่องจาก QEMU-KVM ต้องทำการอ่านข้อมูล Checkpoint File ขึ้นมาจาก Disk เพื่อทำการ Load State การทำงานของ VM มาเริ่มต้นการทำงานต่อ ซึ่งระยะเวลาในการ Load ขึ้นอยู่กับปริมาณของ Memory ที่กำลังใช้งานบน VM ณ จุด Checkpoint และข้อมูล Internal Snapshot บน Disk ของ VM ซึ่งปริมาณของข้อมูลดังกล่าว คือปริมาณของ Snapshot

จากการวิเคราะห์ความสัมพันธ์ของขนาดของ Snapshot และเวลาที่ใช้ในการ Restart พบว่า ลักษณะของกราฟได้ผลลัพธ์คล้ายกับความสัมพันธ์ของขนาดของ Snapshot และเวลาที่ใช้ในการ Checkpoint จากแผนภาพ 4.7 โปรแกรม BT Class C ใช้เวลาน้อยที่สุด ถัดมาคือ MG, BT Class D, IS และ FT ซึ่งในกรณีของ BT Class D และ IS นั้นมีระยะเวลาในการ Restart แตกต่างกันน้อยมากอยู่ที่ 0.1 วินาที และมีขนาดของ Snapshot แตกต่างกันอยู่ 400MB เนื่องจากการอ่านข้อมูลบน Disk เร็วกว่าการเขียนข้อมูลบน Disk อย่างมาก หากมีปริมาณในการอ่านข้อมูลแตกต่างกันน้อยอย่างไม่มีนัยสำคัญ อาจทำให้ผลลัพธ์ของระยะเวลาในการอ่านมีค่าที่ใกล้เคียงกันได้ (ขึ้นอยู่กับอัตราการอ่านและเขียนข้อมูลของ Disk ด้วย)



ภาพที่ 4.7 แสดงความสัมพันธ์ของขนาดของ Snapshot และระยะเวลาในการ Restart

ในขณะเดียวกัน มีอีกปัจจัยหนึ่งที่ส่งผลต่อระยะเวลาของ Load State เล็กน้อย คือ การทำงานของ Agent ในขณะที่ส่งคำสั่งให้ Script ทำการสั่งให้ KVM อ่านข้อมูลจาก

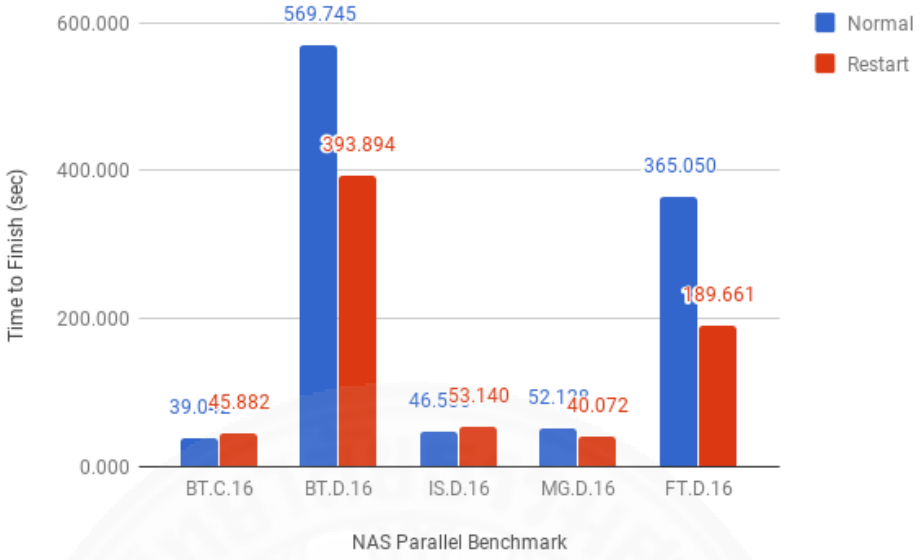
Snapshot มีความจำเป็นต้องกำหนดให้มีหน่วงเวลาเป็นเวลา 3 วินาที เพื่อรอให้ QEMU-KVM เรียก VM ขึ้นมาทำงาน ก่อนที่จะทำการ Load State ขึ้นมาจาก Snapshot ซึ่งช่วงเวลาดังกล่าวขึ้นอยู่กับขนาดของ Snapshot และ ความเร็วในการเรียก VM ขึ้นมาทำงานของเครื่องคอมพิวเตอร์จริง ทำให้อาจเกิดความล่าช้าใน Load State เพิ่มขึ้นจากหน่วงเวลาได้ (ซึ่งทางผู้วิจัยสามารถปรับปรุงแก้ไขประสิทธิภาพได้ในภายหลัง)

จากข้อมูลดังกล่าวสามารถสรุปได้ว่าขนาดของ Snapshot เป็นปัจจัยที่ส่งผลต่อระยะเวลาในการ Restart โดยที่ขนาดของ Snapshot เพิ่มขึ้นจะส่งผลให้ระยะเวลาในการ Restart เพิ่มขึ้น ในทางตรงกันข้าม หากขนาดของ Snapshot ลดน้อยลงจะส่งผลให้ระยะเวลาในการ Restart ลดน้อยลงเช่นเดียวกัน

4.3.3.2 ระยะเวลาในการประมวลผลของ MPI Application Benchmark

จากข้อดีของการทำ Checkpointing คือ สามารถนำ Checkpoint File ไปเริ่มต้นการทำงานใหม่บนเครื่องคอมพิวเตอร์จริงเครื่องอื่นได้ ซึ่งการเริ่มต้นโปรแกรมต่อจากจุด Checkpoint ที่ทำการบันทึกเอาไว้ ทำให้การทำงานของ MPI Application Benchmark อาจจะทำให้การทำงานเร็วขึ้นหรือช้าลง ขึ้นอยู่กับทรัพยากรและสภาพแวดล้อมของเครื่องคอมพิวเตอร์จริงที่นำ VC ไป Restart ทำให้ไม่สามารถนำ Wall Clock Time ของการประมวลผลของ MPI Application Benchmark จากการทำการ Restart มาเปรียบเทียบกับเวลา Wall Clock Time ในการทำงานแบบปกติได้

อย่างไรก็ตาม ผู้วิจัยสามารถนำระยะเวลาในการประมวลผลของโปรแกรม MPI Application Benchmark หลังจากทำการ Restart มาเปรียบเทียบกับระยะเวลาในการประมวลผลของโปรแกรม MPI Application Benchmark ในแบบที่ไม่มีการทำ Checkpointing เพื่อตรวจสอบว่าโปรแกรมสามารถทำงานได้เสร็จในเวลาที่เหมาะสมหรือไม่



ภาพที่ 4.8 เปรียบเทียบเวลาในการทำงานเสร็จจาก 70%-100% ของแบบปกติ และ แบบ Restart

จากแผนภาพ 4.8 แสดงถึงเวลาในการประมวลผลโปรแกรม MPI Application Benchmark ได้ประมาณ 70% จนทำงานเสร็จสิ้นของการทำงานแบบปกติ และหลังจากการทำ Restart บนเครื่องคอมพิวเตอร์จริงเครื่องเดียวกัน ซึ่งเห็นได้ว่าผลลัพธ์ออกมาได้น้อยกว่าหรือใกล้เคียงกัน จึงสามารถสรุปได้ว่าการนำ Checkpoint File มาเริ่มต้นการทำงานโดยใช้ Restart Protocol ในการทำ Restart บน VC สามารถเริ่มต้นการทำงานได้อย่างถูกต้อง และเนื่องจากการ Restart ทำให้ Application ทำงานต่อได้โดยใช้เวลาน้อยกว่าการเริ่มต้น Run Application ใหม่ทั้งหมด ผู้วิจัยจึงสรุปว่า ระบบ Checkpoint/Restart Protocol สามารถทำให้ Application ทำงานเสร็จในเวลาที่เหมาะสม

4.3.3.3 ความถูกต้องในการ Load ข้อมูลของ Restart VC

จากการทดลองการทำ Restart บน VC ด้วยการใช้ Restart Protocol ที่ผู้วิจัยนำเสนอ บนระบบ A Coordinated Checkpointing Framework โดยการนำ Checkpoint File ของ VC ในระหว่างการประมวลผล MPI Application Benchmark มาทำการ Restart เพื่อเริ่มต้นการทำงานต่อจากจุดที่ทำการ Checkpoint พบว่า VC สามารถกลับมาประมวลผล MPI Application Benchmark ต่อจากจุดเดิมได้ ทำให้พิสูจน์ได้ว่าการเก็บข้อมูลของ Checkpoint Protocol ทำงานได้อย่างถูกต้อง เนื่องจาก Checkpoint File สามารถนำกลับมา Restart ได้จริง

และสามารถพิสูจน์ได้ว่า Restart Protocol สามารถ Load ข้อมูลสถานะของ VC จาก Checkpoint File ได้อย่างถูกต้อง

ในขณะเดียวกัน สามารถสรุปได้ว่า การจัดการการสื่อสารระหว่างการ Restart บน VC ทำงานได้อย่างถูกต้อง เนื่องจาก VC สามารถกลับมาเริ่มต้นการทำงานต่อจากจุดเดิมได้อย่างต่อเนื่องโดยไม่มีปัญหาในการสื่อสารข้อมูลหลังจากการทำการ Restart บน VC และ สามารถประมวลผลโปรแกรม MPI Application Benchmark ได้เสร็จสิ้นตามเวลาที่เหมาสมเมื่อเปรียบเทียบกับระยะเวลาการทำงานแบบปกติ

นอกจากนั้นผู้วิจัยได้ทดลองนำ Checkpoint File จากการทำ Checkpointing บน VC นำกลับมาใช้หลังจากที่ทำการ Checkpointing เป็นระยะเวลา 1 วัน พบว่าสามารถนำไป Restart แล้วสามารถทำงานต่อได้และ ทำงานเสร็จในเวลาที่เหมาะสมได้อย่างถูกต้อง จึงสรุปได้ว่าแนวคิด Virtual Time สามารถนำมาใช้ในการ Checkpointing ได้

บทที่ 5
สรุปผลการวิจัยและข้อเสนอแนะ

5.1 สรุปผลการวิจัย

5.1.1 การลดปัญหา TCP Backoff

จากการทดลองพบว่าการนำแนวคิด Barrier Synchronization มาประยุกต์ใช้กับ Checkpoint/Restart Protocol สามารถลดระยะเวลาของความแตกต่างในการเริ่มต้นการทำงานของ VM ทุกเครื่องได้ ซึ่งในการทดลองโดยใช้แนวคิดดังกล่าวส่งผลทำให้ใช้ระยะเวลาในขั้นตอนการทำ Resume ได้น้อยกว่าหรือเท่ากับ 0.3 มิลลิวินาที หมายความว่า ระยะเวลาการเกิด TCP Backoff จะต้องน้อยกว่า 0.3 มิลลิวินาที ซึ่งเป็นระยะเวลาที่น้อยมาก จึงสามารถสรุปได้ว่า การใช้ Barrier Synchronization สามารถช่วยลดปัญหา TCP Backoff ลงได้อย่างมีนัยสำคัญ

5.1.2 ความถูกต้องของ Checkpoint และ Restart Protocol

จากการทดลองการวัด Overhead ในการ Checkpoint ด้วยทำการ Checkpointing และ การทดลองการวัด Overhead ในการ Restart จากจุด Checkpoint ซึ่งได้ทำการ Checkpoint และ Restart VC เป็นจำนวนมาก ผู้วิจัยพบว่า Checkpoint Protocol สามารถเก็บข้อมูลของ Checkpoint VC ได้อย่างถูกต้อง เนื่องจาก สามารถทำงานได้เสร็จสิ้นตามเวลาที่เหมาสมเมื่อเปรียบเทียบกับระยะเวลาการทำงานแบบปกติ ของ MPI Application Benchmark และสามารถพิสูจน์ได้ว่า Checkpoint Protocol สามารถจัดการการสื่อสารระหว่าง VM ภายใน VC ได้อย่างถูกต้อง เนื่องจาก VC สามารถกลับมาทำงานได้อย่างต่อเนื่องโดยไม่มีปัญหาในการสื่อสารข้อมูลหลังจากการทำ Checkpointing ทำให้พิสูจน์ได้ว่าแนวคิด Barrier Synchronization สามารถนำไปแก้ไขปัญหา Inconsistent Global State และ TCP Backoff และสามารถนำไปใช้ประยุกต์ใช้กับ Checkpoint Protocol ได้จริง

ในขณะเดียวกัน Restart Protocol บน VC สามารถทำงานได้อย่างถูกต้อง เนื่องจาก VC สามารถกลับมาเริ่มต้นการทำงานต่อจากจุดเดิมได้อย่างต่อเนื่องโดยไม่มีปัญหาในการสื่อสารข้อมูลหลังจากการทำการ Restart บน VC และ สามารถประมวลผลโปรแกรม MPI Application Benchmark ได้เสร็จสิ้นตามเวลาที่เหมาสมเมื่อเปรียบเทียบกับระยะเวลาการทำงานแบบปกติ และเนื่องจากการ Restart ทำให้ Application ทำงานต่อได้โดยใช้เวลาน้อยกว่าการเริ่มต้น Run Application ใหม่ทั้งหมด นอกจากนั้น Checkpoint File ยังสามารถสำรองข้อมูลของ VC เก็บ

ไว้ แล้วนำไป Restart ได้ทุกเวลา โดยที่สามารถทำงานต่อได้ และทำงานเสร็จในเวลาที่เหมาะสมได้อย่างถูกต้อง ตามแนวคิด Virtual Time

ผู้วิจัยจึงสรุปได้ว่า ระบบ Checkpoint/Restart Protocol สามารถทำงานได้อย่างถูกต้อง โดยใช้แนวคิด Barrier Synchronization เพื่อแก้ไขปัญหา Inconsistent Global State และ TCP Backoff และแนวคิด Virtual Time เพื่อนำการประมวลผลของ Application กลับมาทำงานต่อ ณ เวลาใดก็ได้ โดยที่ผู้ใช้ไม่จำเป็นต้องเข้าไปแก้ไข Application หรือ Guest OS ซึ่งเป็นไปตามสมมติฐานที่กล่าวไว้ในบทที่ 3

5.1.3 ผลกระทบจากการเพิ่มขนาดของปัญหาในการประมวลผล

จากการทดลองการเพิ่มขนาดของปัญหาในการใช้โปรแกรม BT โดยใช้ปัญหา Class C และ D ส่งผลให้เกิด Overhead ในการ Checkpoint เพิ่มขึ้น เนื่องจากปริมาณการใช้งานของ Memory ของ VM เป็นปัจจัยที่ส่งผลโดยตรงกับค่า Overhead ในการ Checkpointing กล่าวคือ หาก Application ที่กำลังประมวลผลใช้ Memory มาก Overhead ในการ Checkpointing จะมากขึ้น ในทางตรงกันข้ามหาก Application ที่กำลังประมวลผลใช้ Memory น้อย Overhead ในการ Checkpointing จะน้อยลง

5.1.4 ปัจจัยหลักที่ส่งผลต่อ Overhead ในการทำ Checkpointing

จากการประมวลผล MPI Application Benchmark ชนิดต่างๆ ในการทดลองการวัด Overhead ในการ Checkpoint ด้วยทำการ Checkpointing พบว่า ขนาดของ Snapshot เป็นปัจจัยหลักที่ส่งผลต่อเวลาในการทำ Checkpointing โดยที่เวลาในการทำ Checkpointing จะแปรผันตรงตามขนาดของ Snapshot เนื่องจาก ขนาดของ Snapshot ที่ได้จากการทำ Checkpointing ขึ้นอยู่กับพฤติกรรมการใช้ทรัพยากรในการประมวลผลของ MPI Application Benchmark ซึ่งปัจจัยสำคัญที่ส่งผลต่อขนาดของ Snapshot คือ ขนาดของ Memory ที่ VM กำลังใช้งานในขณะทำ Checkpointing และ State ของ Disk Image ที่ VM กำลังประมวลผลในขณะทำ Checkpointing

5.2 ข้อเสนอแนะ

จากการพัฒนาและทำการทดลองระบบ Coordinated Checkpointing Framework โดยใช้ Checkpoint และ Restart Protocol พบว่าสามารถนำไปใช้ได้จริง และสามารถนำไปประยุกต์ใช้กับระบบ Data Center ในปัจจุบันได้ ด้วยเหตุผลดังต่อไปนี้

(1) ระบบสามารถทำงานได้อย่างโปร่งใส โดยที่ผู้ใช้ไม่จำเป็นต้องเข้าไปแก้ไข Application หรือ Guest OS บน VM เพื่อใช้งาน Checkpoint/Restart Protocol และนอกจากนั้น การทำงานของ Application มีเวลาเป็นของตัวเองจากการประยุกต์ใช้แนวคิด Virtual Time ส่งผลทำให้เวลาของ Application ได้รับผลกระทบจากการทำ Checkpointing น้อยลง

(2) ถึงแม้ว่า Checkpoint Time ใช้เวลาส่วนใหญ่ไปกับขั้นตอน Stop&Save เนื่องจากความล่าช้าในการเขียนข้อมูลลง Disk แต่ Overhead เหล่านั้นสามารถทำให้ลดลงได้อย่างมีนัยสำคัญด้วยการเปลี่ยนวิธีการเก็บข้อมูลด้วย Hard disk เป็น non-volatile storages หรือ Solid State Drive (SSD) ซึ่ง Data Center ในปัจจุบันใช้ SSD กันอย่างแพร่หลาย

(3) แนวคิดในการทำ Barrier Synchronization และ Virtual Time สามารถลดระยะเวลาที่เกิดจากปัญหา TCP Backoff ได้จริง ซึ่งในการทดลองที่ผ่านมาแสดงให้เห็นว่าการทำการ Checkpointing และ Restart ในระหว่างการประมวลผล MPI Application Benchmark ที่มีการติดต่อสื่อสารโดยใช้ TCP Protocol แบบซับซ้อน สามารถประมวลผลต่อได้อย่างไหลลื่นและสามารถประมวลผลเสร็จได้อย่างถูกต้อง

(4) การนำระบบ Framework ไปประยุกต์ใช้กับเครื่องคอมพิวเตอร์จริง โดยผู้วิจัยต้องทำการศึกษาข้อมูลเพิ่มเติมเกี่ยวกับเครื่องมือและวิธีการที่จะนำมาต่อยอดในการประยุกต์ใช้กับเครื่องคอมพิวเตอร์จริง ซึ่งสามารถวิเคราะห์ได้ 2 ประเด็น คือ

   a. การเก็บสถานะของการทำงานของเครื่องคอมพิวเตอร์ ซึ่งโดยปกติแล้วการเก็บสถานะการทำงานของเครื่อง VM หรือ การทำ Snapshot จะถูกทำอยู่ในระดับ Virtualization หากจะเปลี่ยนไปเก็บสถานะของคอมพิวเตอร์จริงจำเป็นต้องเปลี่ยนวิธีการเก็บสถานะให้อยู่ในระดับ Process รวมไปถึงการเก็บสถานะของเครือข่ายบน Switch จริง และการบันทึกข้อมูลบน Disk ของ Process ด้วย

b. การนำแนวคิด Virtual Time มาประยุกต์ใช้กับเครื่องคอมพิวเตอร์จริง อาจจะสามารถทำได้โดยการเปลี่ยนเวลาของเครื่องจริงให้ตรงกับเวลาที่ของ Process ในขณะที่ทำการ Checkpoint แต่ต้องคำนึงถึงเวลาของข้อมูลการสื่อสารระหว่างเครื่องคอมพิวเตอร์ด้วย

5.3 สิ่งที่คาดว่าจะทำในอนาคต

จากการพัฒนาระบบ A Coordinated Checkpointing Framework ผู้วิจัยได้มองเห็นจุดด้อยของระบบที่สามารถนำไปพัฒนาต่อยอดได้ โดยมีดังต่อไปนี้

5.3.1 Packet Loss Reduction

จากข้อดีของ Virtual Switch (OpenVSwitch) ซึ่งเป็น Software Defined Network ที่สามารถทำให้ควบคุมการสื่อสารของข้อมูลบนเครือข่ายได้ง่ายมากยิ่งขึ้น เช่น สามารถกำหนดจุดหมายของการส่งข้อมูลและดักจับข้อมูลในเครือข่ายได้สะดวก จึงมีแนวคิดที่นำมาช่วยในการจัดเก็บข้อมูลที่กำลังสื่อสารอยู่บนเครือข่ายในช่วงเวลาที่มีเหตุการณ์ Downtime เกิดขึ้น ดังนั้นทางผู้วิจัยจึงนำข้อดีดังกล่าวมาประยุกต์ใช้กับ Framework เพื่อช่วยลดอัตราการเกิด Packets Loss ที่เกิดจาก Downtime ในขณะที่ทำการ Checkpointing โดยใช้วิธีดักจับ Packets ที่กำลังสื่อสารอยู่บนเครือข่ายในช่วงของขั้นตอนการทำ Checkpointing และทำการฉีดกลับเข้าไปยังเครือข่ายในขั้นตอนการทำ Restart บน VC โดยจำแนกข้อมูลบนเครือข่ายตามการให้บริการในระดับของ Transport Layer ซึ่งได้แก่ User Datagram Protocol (UDP) และ Transmission Control Protocol (TCP)

สำหรับ UDP Packets นั้นสามารถบริหารจัดการได้ง่าย เนื่องจาก UDP Protocol อนุญาตให้ข้อมูลมีการสูญหายระหว่างการสื่อสารได้ ในทางกลับกันจึงไม่มีการตรวจสอบและติดตามระยะเวลาในการส่งข้อมูลตลอดเวลา ทำให้สามารถดักจับ UDP Packets และฉีดกลับเข้าไปในเครือข่ายเพื่อใช้งานได้ทันที ส่งผลทำให้ลดปริมาณของข้อมูลที่สูญหายระหว่างการทำ Checkpointing ได้อย่างมีประสิทธิภาพ

แต่สำหรับ TCP Packets ที่ให้บริการโดย TCP Protocol นั้นจะแก้ไขปัญหาโดยการลดอัตราการเกิด Packets Loss ได้ยาก เนื่องจาก TCP Packets มีการติดตามสถานะโดย TCP และมี timeout เป็นของตัวเอง ทำให้ช่วงเวลาในการ Inject TCP Packets กลับเข้าไปในเครือข่ายจะต้องอยู่ในช่วงเวลาที่ยังไม่เกิด TCP Retransmission ซึ่งมีระยะเวลาน้อยมาก หากในจังหวะที่ทำ

การ Inject TCP Packets เข้าไปในเครือข่ายไม่ได้อยู่ภายในช่วงเวลาดังกล่าว จะทำให้เกิดเหตุการณ์ที่ Messages และ ACK ของผู้ส่งและผู้รับไม่สัมพันธ์กันทำให้ Packets ที่ถูก Inject เข้านั้นไปถูก Drop และทำให้ไม่สามารถใช้ TCP Packets ดังกล่าวได้ จึงเป็นสาเหตุทำให้เกิด TCP Backoff ในที่สุด ดังนั้นหากต้องการใช้ประโยชน์จากการฉีด TCP Packets นั้นจะต้องทำให้สำเร็จในช่วงเวลาที่เหมาะสมเท่านั้นหรือที่เรียกกันว่า Window of Opportunity [8]

เนื่องจาก TCP สามารถทำ Retransmission ที่ช่วยแก้ปัญหาการสูญหายของข้อมูล จึงไม่จำเป็นต้องเข้าไปแก้ไขปัญหาดังกล่าว แต่สำหรับการลดการเกิด Packet Loss ของการส่งข้อมูลแบบ UDP เป็นงานที่ท้าทายและสามารถนำมาประยุกต์ใช้กับ Framework ได้ในอนาคต

5.3.2 Time-bound, Thread-Based Live Migration (TLM) Integration

Time-bound, Thread-Based Live Migration (TLM) [10] เป็น งานวิจัยที่ช่วยทำให้ QEMU-KVM สามารถทำ Live Migration ได้รวดเร็วมากยิ่งขึ้น ผู้วิจัยสังเกตเห็นว่า TLM สามารถนำมาใช้เป็นส่วนเสริมใน Framework ได้ โดยนำมาช่วยในการปรับปรุงในส่วนของขั้นตอน Save State ให้รวดเร็วยิ่งขึ้นด้วยการเปลี่ยนจากการสร้าง Snapshot ของ VM ทั้งหมดบน VC ให้เป็นการ Live Migration ของ VM ทั้งหมดไปยัง Memory Server ที่เตรียมไว้ เพื่อลดเวลา Downtime ที่เกิดขึ้นจากการรอ I/O ในการบันทึกข้อมูลลง Disk ทำให้ระยะเวลาในการ Checkpoint Time ลดลงอย่างมาก ผู้วิจัยคาดว่าการนำ TLM เข้ามาประยุกต์ใช้กับ Checkpoint Protocol สามารถช่วยลด Overhead ที่เกิดจาก Checkpoint Protocol ได้อย่างมีนัยสำคัญ

รายการอ้างอิง

1. Dean, J. (2010). Building Software Systems at Google and Lessons Learned. Retrieved from https://research.google.com/people/jeff/Stanford-DL-Nov-2010.pdf
2. Fischer, M., Griffeth, N., & Lynch, N. (1982). Global States of a Distributed System. IEEE Transactions on Software Engineering, SE-8(3), 198-202.
3. Elnozahy, E. N., Alvisi, L., Wang, Y., & Johnson, D. B. (2002). A survey of rollback-recovery protocols in message-passing systems. ACM Computing Surveys, 34(3), 375-408.
4. Jones, T. (2010, October 27). Virtual networking in Linux. Retrieved from https://www.ibm.com/developerworks/linux/library/l-virtual-networking/
5. Ong, H., Saragol, N., Chanchio, K., & Leangsuksun, C. (2009). VCCP: A transparent, coordinated checkpointing system for virtualization-based cluster computing. 2009 IEEE International Conference on Cluster Computing and Workshops.
6. So-In, C., Jain, R., & Dommety, G. (2009). PETS: Persistent TCP using simple freeze. 2009 First International Conference on Future Information Networks.
7. Feamster, N., Rexford, J., & Zegura, E. (2014). The road to SDN. ACM SIGCOMM Computer Communication Review, 44(2), 87-98.
8. Kangarlou, Ardalan, Patrick Eugster, and Dongyan Xu. "VNsnap: Taking snapshots of virtual networked infrastructures in the cloud." IEEE Transactions on Services Computing 5.4 (2012): 484-496.
9. Burtsev, A., Radhakrishnan, P., Hibler, M., & Lepreau, J. (2009). Transparent checkpoints of closed distributed systems in emulab. Proceedings of the Fourth ACM European Conference on Computer Systems - EuroSys 09.
10. Chanchio, K. (2015). TPLCR: Time-Bound, Pre-copy Live Checkpointing and Parallel Restart of Virtual Machines Using Distributed Memory Servers. 2015 Third International Symposium on Computing and Networking (CANDAR).

11. Seddigh, N., & Devetsikiotis, M. (2001). Studies of TCPs retransmission timeout mechanism. ICC 2001. IEEE International Conference on Communications. Conference Record (Cat. No.01CH37240).
12. QEMU. (2018, June 19). Retrieved from https://en.wikipedia.org/wiki/QEMU
13. NAS Parallel Benchmarks. (2018, June 19). Retrieved from https://en.wikipedia.org/wiki/NAS_Parallel_Benchmarks
14. MPICH. (2018, June 23). Retrieved from https://en.wikipedia.org/wiki/MPICH
15. The free Command. (2006, July 9). Retrieved from http://www.linfo.org/free.html
16. Graf, T. (2017, October 03). Bmon - Bandwidth Monitor. Retrieved from https://github.com/tgraf/bmon
17. Pechwises, C., & Chanchio, K. (2018). A Transparent Hypervisor-level Checkpoint-Restart Mechanism. The 15th International Joint Conference on Computer Science and Software Engineering (JCSSE2018).

ภาคผนวก

ภาคผนวก ก
การพัฒนาโปรแกรม A Coordinated Checkpoint Framework

สำหรับการพัฒนาเฟรมเวิร์ค ผู้วิจัยได้ใช้โปรแกรม Python 3.5 ในการพัฒนาขึ้น โดยมีองค์ประกอบหลักอยู่ทั้งหมด 4 ส่วน คือ Coordinator, Agent, State Machine และ Command Interface API

(1) **Coordinator** เป็นโปรแกรม TCP-Socket ฝั่ง Server โดยใช้วิธีแบบ Select บน Port 9500 ทำหน้าที่เป็น Server เชื่อมต่อกับ Agent คอยประสานงานในการทำ Checkpoint ของ VC

(2) **Agent** เป็นโปรแกรม TCP-Socket ฝั่ง Client บน Port 9500 ทำหน้าที่เป็น Client เชื่อมต่อกับ Coordinator และคอยรับคำสั่งจาก Coordinator และคอยสั่งงาน VM โดยใช้ Bash Shell Script ที่เตรียมไว้

(3) **State Machine** คือ โปรแกรมการจำลองสถานะของการทำงานของ Framework ซึ่งมีการเขียนกฎในการเปลี่ยนสถานะอย่างชัดเจนตาม State of Framework ในบทที่ 3 โดยการจำลองสถานะจะแบ่งเป็น 2 แบบ คือ

   a. แบบ Main State คือ สถานะหลักของ Framework ประกอบไปด้วย Initialize, Launch, Ready, Checkpoint, Restart, และ Terminate

   b. แบบ Sub State คือ สถานะย่อยของสถานะ Checkpoint และ Restart แยกกัน สำหรับสถานะ Checkpoint ประกอบไปด้วย Initialize, Stop&Save, และ Resume และสำหรับสถานะ Restart ประกอบไปด้วย Initialize, Load, และ Resume

(4) **Command Interface API** เป็นโปรแกรม Command-line User Interface เขียนด้วย Python 3.5 เพื่อให้ผู้ใช้สามารถสั่งงาน Coordinator จากเครื่องคอมพิวเตอร์ที่เชื่อมต่อกับ Management Network ของ VC บน Port 9505 โดยมีคำสั่งทั้งหมด 4 คำสั่ง ได้แก่

   a. คำสั่ง “launch” เป็นคำสั่งเรียก VM ขึ้นมาทำงาน โดยพิมพ์คำสั่ง “launch” และสามารถใส่พารามิเตอร์เพิ่มเติมได้ ได้แก่ พารามิเตอร์ “--host” คือ รายชื่อ IP Address ของเครื่อง Host บน Management Network และ พารามิเตอร์ “--n” คือ จำนวนเครื่อง VM ต่อเครื่อง Host

b. จำนวน 1 เครื่อง สำหรับค่า Default ของคำสั่งคือ IP Address ของเครื่อง Host ที่ใช้ทดสอบ และ ใช้เครื่อง VM เป็นจำนวน 4 เครื่อง ต่อเครื่อง Host 1 เครื่อง
c. คำสั่ง “checkpoint” เป็นคำสั่งเพื่อสั่งให้ Coordinator ทำการ Checkpointing VC โดยพิมพ์คำสั่ง “checkpoint” ตามด้วยพารามิเตอร์ “--name” คือ ชื่อของ Checkpoint File ที่ผู้ใช้กำหนด และสามารถเพิ่มพารามิเตอร์ “--delay” คือ ระยะเวลาหน่วงก่อนที่จะทำการ Checkpoint มีหน่วยเป็นวินาที มีค่า Default เป็น 0 วินาที
d. คำสั่ง “restart” เป็นคำสั่งเพื่อสั่งให้ Coordinator ทำการ Restart VC โดยใช้คำสั่ง “restart” ตามด้วยพารามิเตอร์ “--name” คือ ชื่อของ Checkpoint File ที่ผู้ใช้ต้องการนำมา Restart และสามารถเพิ่มพารามิเตอร์ “--delay” คือ ระยะเวลาหน่วงก่อนที่จะทำการ Restart มีหน่วยเป็นวินาที มีค่า Default เป็น 0 วินาที
e. คำสั่ง “terminate” เป็นคำสั่งหยุดการทำงานและปิดเครื่อง VM ทั้งหมดบน VC และทำการจบการทำงานของ Coordinator และ Agent โดยพิมพ์คำสั่ง “terminate”

สำหรับโปรแกรม Bash Shell Script ที่ผู้วิจัยได้เตรียมไว้เพื่อให้ Agent เรียกใช้งานเพื่อสั่งงาน QEMU-KVM Hypervisor โดยใช้ QEMU Monitor Protocol แบบ HMP ผ่าน Unix Socket โดยมีคำสั่งที่ใช้ ได้แก่

(1) คำสั่ง “qemu-system-x86_64” – เป็นคำสั่งที่ใช้เพื่อสั่งให้ Hypervisor รัน VM ขึ้นมา ความจริงแล้วคำสั่งนี้จะมีอยู่หลายแบบตามสถาปัตยกรรมของระบบ สำหรับคำสั่งนี้สามารถใช้กับเครื่องคอมพิวเตอร์สถาปัตยกรรม x86 และ x64 เท่านั้น โดยมีคำสั่งย่อยที่สำคัญ คือ
   a. “--kvmclock” และ “-rtc base=localtime,clock=vm” คือ ไม่ใช้ kvm clock ที่ Synchronize เวลากับเครื่อง Host โดยจะเปลี่ยนไปใช้ HPET ทำให้ VM มีเวลาเป็นของตัวเอง
   b. “-boot -vnc : [port-number]” – คือ เปิดฟังก์ชัน VNC Server เพื่อให้สามารถเข้าไปใช้ Console ของ VM ผ่านโปรแกรม VNC Client ผ่าน Port ที่กำหนด

c. “-monitor unix:[unix-socket-file],server,nowait” คือ เปิด QEMU Monitor แบบ HMP เพื่อให้ผู้ใช้สามารถส่งคำสั่ง QEMU ผ่านไฟล์ Unix Socket

d. “-netdev type=tap” คือ กำหนดให้ชนิดของ Network Interface เป็นชนิด TAP โดยที่จะสร้างขึ้นมาอัตโนมัติผ่าน Script ที่เตรียมไว้

e. “-device virtio-net-pci” กำหนด MAC Address ให้แก่ TAP Interface ที่กำหนดไว้

(2) คำสั่ง “loadvm” เป็นคำสั่งที่ใช้ load ข้อมูล Snapshot ของ VM

(3) คำสั่ง “savevm” เป็นคำสั่งที่ใช้ทำ Snapshot ของ VM

(4) คำสั่ง “stop” เป็นคำสั่งที่ใช้หยุดการทำงาน VM

(5) คำสั่ง “cont” เป็นคำสั่งที่สั่งให้ VM ทำงานต่อจากจากการหยุดการทำงาน

(6) คำสั่ง “quit” เป็นคำสั่งที่ใช้การปิดการทำงานของ QEMU-KVM เพื่อปิด VM

(7) คำสั่ง “info snapshots” เป็นคำสั่งที่ใช้ตรวจสอบรายละเอียดของ Snapshot ทั้งหมดที่อยู่ในไฟล์ Image ที่ใช้งานอยู่

สำหรับในการเตรียมการ VM ผู้วิจัยต้องสร้าง Base Image และ Install Ubuntu Server 16.04 และ NAS Parallel Benchmark เพื่อใช้ในการทำการทดลอง ในการสร้าง Base Image จะใช้คำสั่ง “qemu-img create -f qcow2 [image-name] [size-of-image]”

นอกจากนั้น ในการทดลองต้องแต่ละครั้งจะต้องมีการสร้าง Overlay Image จาก Base Image เพื่อง่ายต่อการทำการทดลองหลายครั้งและเพื่อป้องกันไม่ให้ Base Image ถูกแก้ไข โดยใช้คำสั่ง “qemu-img create -b [base-image] -f qcow2 [new-overlay-image]”

ภาคผนวก ข
การติดตั้ง Virtual Network

จากการออกแบบ Virtual Network ในบทที่ 3 ผู้วิจัยได้นำ OpenVSwitch ซึ่งเป็น Software Defined Network (SDN) มาใช้ในการสร้างเครือข่ายเสมือนขึ้นมา ในการทดลองนั้นผู้วิจัยใช้เครื่องคอมพิวเตอร์จริง (Host) 2 เครื่อง โดยจะมีการสร้าง GRE Tunneling เพื่อให้ VM ที่อยู่บนเครื่องคอมพิวเตอร์จริงทั้งสองสามารถเชื่อมต่อกันได้ ซึ่งในระบบจะมี Network อยู่ทั้งหมด 2 Network คือ Management Network และ Data Network โดยทั้งสอง Network จะมีขั้นตอนการสร้างที่เหมือนกัน ดังต่อไปนี้

(1) ทำการสร้าง Virtual Switch หรือ External Bridge (br-ex) เพื่อเชื่อมต่อกับ Port จริงของเครื่อง Host (กำหนดให้ eth0 คือ Port จริงของเครื่อง Host)

```
$ sudo ovs-vsctl add-br br-ex
$ sudo ovs-vsctl add-port br-ex eth0
```

(2) เมื่อทำการ Add Port แล้ว ให้เข้าไปแก้ไขไฟล์ /etc/network/interfaces เพื่อกำหนดค่า Network ให้ eth0 และ br-ex ให้เป็นค่าใหม่

```
$ sudo vi /etc/network/interfaces
...
auto eth0
        iface eth0 inet manual
auto br-ex
        iface br1 inet manual

$ sudo ifdown eth0
$ sudo ifup eth0
$ sudo ifup br-ex
```

(3) หลังจากตั้งค่า br-ex เสร็จสิ้น ให้เพิ่ม Interface ที่เป็น Tunnel Endpoint (TEP) เพื่อให้สามารถกำหนด IP Address เพื่อเป็นช่องทางสำหรับการสร้าง Tunnel โดยให้ตั้งชื่อว่า tep0 และกำหนดค่า IP Address ให้เป็นค่า IP Address ของ eth0 เดิมที่เคยตั้งค่าไว้ (ยกตัวอย่างด้วย IP Address 172.17.0.21)

```
$ sudo ovs-vsctl add-port br-ex tep0 – set interface
tep0 type=internal
$ sudo vi /etc/network/interfaces
...
auto tep0
        iface tep0 inet static
        address 172.17.0.21
        netmask 255.255.255.0
```

(4) ให้ทำขั้นตอนที่ (1), (2), และ (3) บนทั้งเครื่อง Host1 และ Host2 เมื่อเสร็จแล้วให้ลองใช้คำสั่ง ping โดยใช้ IP Address บน Network 172.17.0.0/24 ที่ทำการตั้งค่าไว้บน tep0 จะเห็นได้ว่า Host ทั้งสองสามารถ ping สำเร็จ

(5) หลังจากที่เตรียม Tunnel EndPoint เรียบร้อยแล้ว ต่อไปจะทำการสร้าง Virtual Switch อีกหนึ่งชั้น คือ Internal Switch หรือ “br-int” เพื่อเป็น Switch รองรับการเชื่อมต่อของ VM ภายในของแต่ละ Host โดยที่ VM ของแต่ละ Host จะสามารถคุยกันผ่าน GRE Tunnel ที่กำลังจะสร้างขึ้น โดยอันดับแรกให้สร้าง Virtual Switch “br-int” ขึ้นมาก่อน แล้วหลังจากนั้นให้สร้าง GRE Interface โดยแต่ละ Host จะต้องกำหนดให้ Remote IP ของปลายทางด้วย

Host1:

```
$ sudo ovs-vsctl add-br br-int
$ sudo ovs-vsctl add-port br-int gre0 – set interface gre0
type=gre options:remote_ip=172.17.0.22
```

Host2:

```
$ sudo ovs-vsctl add-br br-int
$ sudo ovs-vsctl add-port br-int gre0 – set interface gre0
type=gre options:remote_ip=172.17.0.21
```

(6) ติดตั้ง QEMU-KVM บน Host ทั้งสอง และสร้าง Shell Script ovs-ifup และ ovs-ifdown เพื่อให้ VM ที่สร้างขึ้นสามารถเชื่อมต่อกับ Virtual Switch “br-int” หลังจากนั้นให้ทำการเปลี่ยน Permission ของไฟล์ทั้งสองเพื่อให้สามารถ Execute ได้

ovs-ifup:

```
$ sudo vi /etc/openvswitch/ovs-ifup
#!/bin/sh
switch='br-int'
/sbin/ifconfig $1 0.0.0.0 up
ovs-vsctl add-port ${switch} $1
```

ovs-ifdown:

```
$ sudo vi /etc/openvswitch/ovs-ifdown
#!/bin/sh
switch='br-int'
/sbin/ifconfig $1 0.0.0.0 down
ovs-vsctl del-port ${switch} $1
```

(7) ในการรัน VM เพื่อเชื่อมต่อกับ Virtual Network จะต้องกำหนดให้ QEMU-KVM บน Host ทั้งสองไปเรียกใช้ Script จากข้อ (6) ในการเชื่อมต่อเข้ากับ “br-int” อัตโนมัติโดยการเพิ่มฟังก์ชัน “-netdev type=tap,script=${etcloc}/ovs-ifup,downscript=${etcloc}/ovs-ifdown,id=hostnet1” โดยที่ id จะต้องตรงกับ id ของฟังก์ชัน “-device” ด้วย

(8) จากนั้น เมื่อทำการรัน VM ให้เข้าไปตั้งค่า Network ใน /etc/network/interfaces ใน VM ทุกเครื่องเพื่อกำหนด Network ของ VM ทั้งหมดให้เชื่อมต่อภายใน Network เดียวกัน

ภาคผนวก ค
การใช้งาน NAS Parallel Benchmark

การตั้งค่าของ VM ในการทำการลดลอง NAS Parallel Benchmark (NPB) จะใช้โปรแกรม NPB3.3.1 (ล่าสุดเมื่อเดือน กรกฎาคม 2561) ซึ่งพัฒนาในรูปแบบโปรแกรม MPI ซึ่งจะถูกติดตั้งบน SHARE Directory โดยที่ VM ทั้งหมดสามารถเข้าถึงได้ ผู้วิจัยได้เลือก Compile โปรแกรม NPB เฉพาะที่เลือกใช้ทำการทดลอง คือ *BT.16.C, BT.16.D, IS.16.D, MG.16.D* และ *FT.16.D*

สำหรับการรัน NPB จะใช้ VM ทั้งหมด 8 เครื่อง ซึ่งจะทำงานอยู่บน Host1 และ Host 2 เครื่องละ 4 VM คือ VM1 ถึง VM4 จะทำงานอยู่บน Host1 ส่วน VM5 ถึง VM8 จะทำงานอยู่บน Host2 ซึ่ง VM2 ถึง VM7 จะถูก Clone มาจาก VM1 ในการประมวลผล MPI Application ของ NPB ประกอบไปด้วย เครื่อง Head Node จำนวน 1 เครื่อง คือ VM1 ทำหน้าที่กระจายงานให้กับ Node อื่นๆ และ Slave Node จำนวน 7 เครื่อง ทำหน้าที่เป็น Slave Node ในการรัน NPB โดยจะกำหนดให้มีการสื่อสารของ MPI Application บน Data Network เท่านั้น

ก่อนทำการประมวลผล NPB จะต้องตรวจสอบว่า VM ทุกเครื่องเชื่อมต่อกันทั้งหมด หากมี VM เครื่องใดไม่สามารถติดต่อกับเครื่อง VM Head Node ได้ โปรแกรมจะไม่สามารถประมวลผลได้ เนื่องจาก Head Node ไม่สามารถส่งข้อมูลไปยัง Node อื่นได้ สำหรับการเรียกใช้งาน NPB จะใช้คำสั่ง “mpiexec -n 16 -host vm1,vm2,vm3,vm4,vm5,vm6,vm7,vm8 [npb-name]”

ในการวัดเวลาของการทำการทดลอง NPB ผู้วิจัยได้เขียน Shell Script ไว้บน VM ที่เป็นเครื่อง Head Node เพื่อสั่งให้ประมวลผล NPB โดยก่อนทำการประมวลผล NPB ไฟล์ Shell Script จะสั่งให้ทำการบันทึกเวลาโดยการส่งคำสั่งผ่าน SSH บน Management Network ไปยัง Host 2 เพื่อไปบันทึกเวลาของการเริ่มต้นการประมวลผลของ NPB ลงบนไฟล์ Log เมื่อ NPB ประมวลผลเสร็จ Shell Script จะสั่งให้ส่ง SSH เข้าไปเขียนไฟล์ Log เดิมที่เขียนทิ้งไว้ก่อนหน้า เพื่อบันทึกเวลาของการประมวลผลเสร็จสิ้น

ประวัติผู้เขียน

ชื่อ	นายชยาวัฒน์ เพชรวิเศษ
วันเดือนปีเกิด	13 พฤศจิกายน 2533
วุฒิการศึกษา	ปีการศึกษา 2560: วิทยาศาสตรบัณฑิต (วิทยาการคอมพิวเตอร์) มหาวิทยาลัยธรรมศาสตร์
ตำแหน่ง	Freelance Programmer
ผลงานทางวิชาการ	

A Transparent Hypervisor-level Checkpoint-Restart Mechanism [17]

ประสบการณ์ทำงาน	2558 - 2561 Freelance Programmer 2558 - 2560 Teaching Assistant สาขาวิชาวิทยาการคอมพิวเตอร์ มหาวิทยาลัยธรรมศาสตร์