การเก็บสถานะการประมวลผลของเครื่องคอมพิวเตอร์เสมือนไลฟ์แบบเทรด โดยใช้ระบบฐานข้อมูล NoSQL

Thread-based Live Checkpointing of Virtual Machines using NoSQL database

รุจรดา เย็นเยือก[1] และ กษิดิศ ชาญเชี่ยว[2]
ภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยธรรมศาสตร์
99 ม. 18 ต. คลองหนึ่ง อ. คลองหลวง จ.ปทุมธานี 12121
E-mail: [1]n.rutrada@hotmail.com , [2]kasiditchanchio@gmail.com

บทคัดย่อ

บทความนี้นำเสนอวิธีการใหม่เพื่อจัดเก็บสถานะการประมวลผลของเครื่องคอมพิวเตอร์เสมือนลงสู่หน่วยเก็บข้อมูลสำหรับการประมวลผลแบบทนทานต่อความผิดพร่องเรียกว่า การทำเช็คพอยต์ติ้งไลฟ์แบบเทรด (Thread-base Live Checkpointing หรือ TLC) ซึ่งจะสร้างเทรดใหม่ขึ้นมาเพื่อทำการเช็คพอยต์สถานะของเครื่องคอมพิวเตอร์เสมือนในขณะเดียวกันกับที่เครื่องคอมพิวเตอร์เสมือนนั้นทำงาน โดยที่ข้อมูลสถานะส่วนหนึ่งจะถูกเก็บบนระบบฐานข้อมูลแบบ NoSQL ผู้วิจัยได้สร้างซอฟต์แวร์ต้นแบบของวิธีการ TLC บนระบบ Kernel-based Virtual Machine และทดลองใช้ TLC เก็บสถานะของเครื่องคอมพิวเตอร์เสมือนแบบ SMP ขณะประมวลผลโปรแกรม NPB Benchmark ที่ใช้ซีพียูและหน่วยความจำมากสามโปรแกรม ผู้วิจัยพบว่าวิธีการแบบ TLC มีประสิทธิภาพการทำงานมากกว่าวิธีการเช็คพอยต์แบบเดิมของระบบ KVM เนื่องจากใช้ระยะเวลาน้อยกว่า 0.20 ถึง 0.31 เท่าของเวลาที่วิธีการเช็คพอยต์แบบเดิมใช้

คำสำคัญ: การเช็คพอยต์, คอมพิวเตอร์เสมือน

Abstract

This paper presents a novel Thread-base Live Checkpointing (TLC) mechanism of Virtual Machines that collects the state of a virtual machine **live** while the virtual machine is running and stores partial state information on a NoSQL database to increase checkpointing performance. We have extended the Kernel-based Virtual Machine software to host the TLC mechanism and conducted experiments on SMP virtual machines running three CPU and memory-intensive NPB benchmarks. The results show TLC performed several times better than the traditional virtual machine checkpointing method.

Keywords: Checkpointing, Virtual Machines

1. คำนำ

ในปัจจุบันคอมพิวเตอร์เสมือน (Virtual Machine) ถูกนำไปใช้สำหรับการประมวลผลสมรรถนะสูงเพิ่มมากขึ้นเนื่องจากเทคโนโลยีการทำคอมพิวเตอร์เสมือนได้รับการพัฒนาทั้งด้านซอฟต์แวร์และฮารดแวร์ให้มีประสิทธิภาพใกล้เคียงกับคอมพิวเตอร์จริง เป็นผลให้การประมวลผลสมรรถนะสูงได้รับประโยชน์จากความสามารถและคุณลักษณะของคอมพิวเตอร์เสมือนไปด้วย

ความสามารถประการหนึ่งของการทำคอมพิวเตอร์เสมือน (Virtualization) ที่เป็นประโยชน์สำหรับการประมวลผลสมรรถนะสูงคือความสามารถในการทำเช็คพอยต์ติ้ง (Checkpointing) ซึ่งเป็นการจัดเก็บสถานะ (State) ของเครื่องคอมพิวเตอร์เสมือนทั้งเครื่องลงบนหน่วยเก็บข้อมูลถาวร (Persistent Storage) เพื่อให้ผู้ใช้สามารถเรียกคืน (Recovery) การประมวลผลของคอมพิวเตอร์เสมือนต่อจากตำแหน่งที่ทำการเช็คพอยต์บนเครื่องคอมพิวเตอร์จริง (Host Computer) เครื่องใหม่ได้หลังจากที่เครื่องคอมพิวเตอร์จริง เดิมของเครื่องคอมพิวเตอร์เสมือนล่ม

แต่การทำเช็คพอยต์ติ้งสำหรับคอมพิวเตอร์เสมือนที่เป็นแบบ SMP ที่มีหน่วยความจำหลักขนาดใหญ่ และประมวลผลโปรแกรมสมรรถนะสูงนั้นทำได้ยากเนื่องจากวิธีการที่มีอยู่ต้องหยุดพักคอมพิวเตอร์เสมือนเป็นเวลานานเพื่อเก็บสถานะบนหน่วยเก็บข้อมูลถาวรโดยเฉพาะอย่างยิ่งเมื่อหน่วยเก็บข้อมูลถาวรนั้นอยู่บนเครื่องคอมพิวเตอร์จริงต่างเครื่องจากที่คอมพิวเตอร์เสมือนปฏิบัติการอยู่

งานวิจัยนี้นำเสนอการทำเช็คพอยต์ติ้งไลฟ์แบบเทรด หรือ Thread-based Live Checkpointing (TLC) ซึ่งจะสร้างเทรดใหม่ขึ้นมาเพื่อทำการเช็คพอยต์สถานะของเครื่องคอมพิวเตอร์เสมือน **ไลฟ์** คือทำไปพร้อมๆกันกับการประมวลผลของเครื่องคอมพิวเตอร์เสมือน โดยที่ข้อมูล

สถานะส่วนหนึ่งจะถูกเก็บบนระบบฐานข้อมูลแบบ NoSQL [1] เพื่อเพิ่มประสิทธิภาพของการเช็คพอยต์ติ้ง ผู้วิจัยได้สร้างระบบต้นแบบและทดสอบกับโปรแกรม NPB benchmarks [2] สามโปรแกรมและพบว่าวิธีการ TLC ใช้เวลาเพียงแค่ 0.08 ถึง 0.14 เท่าของเวลาที่วิธีการเดิมใช้ในการหยุดการทำงานโดยรวมของเครื่องคอมพิวเตอร์เสมือน (นอกจากนั้น TLC จะพยายามกระจายการหยุดนี้ออกไปเป็นเวลาย่อยๆ เพื่อให้ผู้ใช้รู้สึกถึงการขัดจังหวะการทำงานให้น้อยที่สุด) และพบว่า TLC ใช้เวลาในการทำเช็คพอยต์ติ้งทั้งหมดเพียง 0.20 ถึง 0.31 ของเวลาที่ใช้โดยวิธีการเดิม

2. ทบทวนวรรณกรรม

งานวิจัยทางด้านเช็คพอยต์ติ้งที่ผ่านมาส่วนใหญ่เป็นการเก็บสถานะของโพรเซส ยกตัวอย่างเช่นระบบ Condor [3] ระบบ libckpt [4] และระบบ BLCR [5] เป็นต้น ในระบบเหล่านี้ผู้ใช้จำเป็นต้องติดตั้งซอฟต์แวร์เพิ่มเติมเช่น library หรือ OS kernel module บนระบบปฏิบัติการของเครื่องคอมพิวเตอร์จริง อันเป็นภาระของผู้ใช้ที่จะต้องจัดหาและติดตั้งแอพพลิเคชั่นและซอฟแวร์สำหรับเช็คพอยต์และ OS เวอร์ชั่นที่ทำงานร่วมกันได้ ทำให้เกิดความพึ่งพากัน (Dependency) สูง เป็นผลให้การเปลี่ยนแปลงปรับปรุงส่วนใดส่วนหนึ่งทำได้ยาก

ในทางกลับกันการทำเช็คพอยต์ติ้งในระดับของคอมพิวเตอร์เสมือนนั้นมีความโปร่งใส (Transparency) กับผู้ใช้ แอพพลิเคชั่น และ Guest OS ที่ปฏิบัติการบนระบบคอมพิวเตอร์เสมือนสูงและไม่เป็นภาระของผู้ใช้อีกต่อไป เพราะการทำเช็คพอยต์ติ้งเกิดขึ้นในไฮเปอร์ไวเซอร์ (Hypervisor) ซึ่งเป็นซอฟต์แวร์จัดการการประมวลผลของระบบคอมพิวเตอร์เสมือนที่อยู่ในระดับต่ำกว่าเกส โอเอส (Guest OS)

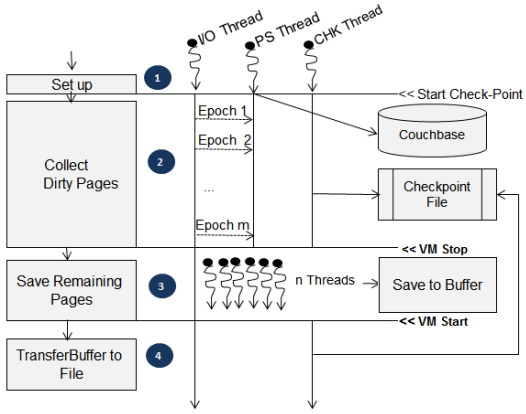
อย่างไรก็ตามการทำเช็คเช็คพอยต์ติ้งของคอมพิวเตอร์เสมือนที่มีอยู่ในปัจจุบันมีข้อเสียคือไฮเปอร์ไวเซอร์ต้องหยุดพักการประมวลผลของคอมพิวเตอร์เสมือนเพื่อจัดเก็บสถานะของอุปกรณ์ (Device) ทุกอย่างรวมทั้งหน่วยความจำหลัก (ซึ่งมักจะมีขนาดใหญ่) ของระบบคอมพิวเตอร์เสมือนลงสู่แผ่นดิสก์ การหยุดนี้มักจะใช้เวลานานเมื่อขนาดของหน่วยความจำมีขนาดใหญ่และการเขียนข้อมูลลงแผ่นดิสก์ทำผ่านระบบเครื่อข่ายเช่น NFS

ไฮเปอร์ไวเซอร์ส่วนใหญ่ไม่ว่าจะเป็น vmware [6] หรือ KVM [7] หรือ Xen [8] มีความสามารถในการทำเช็คพอยต์ติ้ง (บางไฮเปอร์ไวเซอร์เรียกว่าการสร้าง snapshot) อยู่แล้ว แต่ทุกไฮเปอร์ไวเซอร์ต้องหยุดพักการประมวลผลดังที่กล่าวมาแล้วข้างต้น งานวิจัยของ วศินีย์ ศิริปุณย์และกษิดิศ ชาญเชี่ยว [9] เป็นงานวิจัยแรกที่นำเสนอวิธีการเช็คพอยต์ติ้งแบบ ไลฟ์ ซึ่งอนุญาตให้การทำเช็คพอยติ้งเกิดขึ้นพร้อมกับการประมวลผล

ในงานวิจัยนี้ผู้วิจัยได้ปรับปรุงระบบดังกล่าวให้สามารถรองรับการทำงานของคอมพิวเตอร์เสมือนแบบ SMP และส่งข้อมูลสถานะส่วนหนึ่งไปเก็บที่ระบบฐานข้อมูลแบบ NoSQL เพื่อเพิ่มประสิทธิภาพในการเก็บสถานะให้มากขึ้น

3. วิธีการ TLC

วิธีการเช็คพอยติ้งแบบ TLC มีหลักการคือการโอนภาระงานในการเช็คพอยต์ไปไว้ที่เทรดใหม่เพื่อพยายามให้การเช็คพอยต์ส่งผลกระทบกับการประมวลของเครื่องคอมพิวเตอร์เสมือนให้น้อยและใช้ประโยชน์ของซีพียู Multi-core ในขณะเดียวกัน ในงานวิจัยนี้ผู้วิจัยได้เพิ่มขยายซอฟต์แวร์ KVM (qemu-kvm version 1.0) ซึ่งเป็นไฮเปอร์ไวเซอร์แบบโอเพ่นซอรส์ให้ทำเช็คพอยติ้งแบบ TLC ได้



รูปที่ 1 โครงสร้างและขั้นตอนการทำงานของ TLC

จากรูปที่ 1 ตามปกติแล้วเมื่อ KVM ประมวลผลคอมพิวเตอร์เสมือนแบบ SMP KVM จะสร้างเทรดสำหรับจัดการ I/O และดูแลระบบของคอมพิวเตอร์เสมือนเรียกว่า IO Thread (ดังภาพ) และสร้างเทรดเพื่อรองรับการประมวลผลหนึ่งเทรดต่อ SMP คอร์หนึ่งคอร์ TLC แบ่งการทำงานออกเป็น 4 ขั้นตอน ได้แก่ **ขั้นตอนที่หนึ่ง** เมื่อเริ่มทำการเช็คพ้อยน์ TLC จะสร้างเทรดขึ้นใหม่สองเทรด ชื่อว่า CHK Thread และ PS Thread ดังภาพ และประสานการทำงานของเทรดทั้งสองกับ IO Thread

ขั้นตอนที่สอง CHK Thread ทำหน้าที่สแกนและเก็บข้อมูลหน่วยความจำของคอมพิวเตอร์เสมือนตั้งแต่เพจ (page) 0 ไปจนถึงเพจสุดท้ายลงบนไฟล์เก็บสถานะเรียกว่า Checkpoint File (กำหนดให้เรียกว่าการทำงานของ CHK Thread ว่า ขั้นตอน 2.1) ในขณะเดียวกันที่ CHK Thread ทำงาน TLC อนุญาตให้ IO Thread รองรับการประมวลผลของคอมพิวเตอร์เสมือนตามปกติ แต่ TLC จะขัดจังหวะการทำงานของ IO Thread เป็นระยะๆทุกๆ 3 วินาที กำหนดให้เรียกช่วงเวลาการประมวลผลในอดีตของคอมพิวเตอร์เสมือนก่อนที่จะเกิดการขัดจังหวะแต่ละครั้งว่าหนึ่ง Epoch ในขณะที่มีการขัดจังหวะ TLC จะเรียกใช้ KVM system call เพื่อสอบถามฮาร์ดแวร์ว่ามีหน่วยความจำหลักของคอมพิวเตอร์เสมือนเพจใดบ้างที่ได้รับการเปลี่ยนแปลงในช่วง Epoch ที่ผ่านมาและเก็บค่า Dirty Bits ของเพจเหล่านั้นก่อนที่จะปล่อยให้ IO Thread กลับไปประมวลผล Epoch ถัดไป ในขณะเดียวกัน PS Thread จะทำงานพร้อมๆกันไปและทำหน้าที่เดียวคือส่งข้อมูลของเพจที่ Dirty ไปเก็บที่ฐานข้อมูล NoSQL (กำหนดให้เรียกว่าการทำงานของ IO Thread และ PS Thread ในนี้ว่าขั้นตอน 2.2) เนื่องจากข้อมูลเพจที่ถูกเก็บในขั้นตอน 2.2 นั้นใหม่กว่าข้อมูลที่ถูกเก็บในขั้นตอน 2.1 ดังนั้น TLC จะลดจำนวนเพจที่ต้องเก็บโดยให้ขั้นตอน 2.1 ละเว้นไม่เก็บข้อมูลของเพจที่ Dirty ในขั้นตอน 2.2

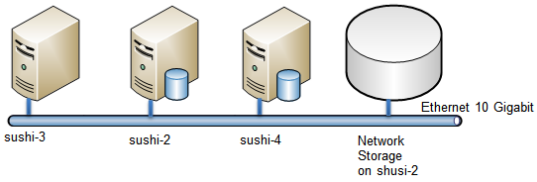
สำหรับฐานข้อมูล NoSQL ที่ใช้ในงานวิจัยนี้นั้นเป็น Couchbase Server [1] ซึ่งเก็บข้อมูลที่ใช้งานไว้ในหน่วยความจำของเครื่องคอมพิวเตอร์และในแผ่นดิสก์ NoSQL มีความเหมาะสมสำหรับการเก็บข้อมูลเพจเพราะ ข้อมูลเพจสามารถอ้างอิงได้ด้วยแอดเดรสหน่วยความจำ และการเขียนข้อมูลลง NoSQL ทำได้เร็วเพราะเก็บข้อมูลในหน่วยความจำ

ขั้นตอนที่สาม หลังจากที่ขั้นตอน 2.1 สแกนและเก็บข้อมูลในเพจสุดท้ายแล้ว TLC จะจบการทำงานของ PS Thread และหยุดพักการทำงานของเทรดทุกๆเทรดที่ประมวลผลคอมพิวเตอร์เสมือนเพื่อให้ IO Thread เก็บข้อมูลของ Dirty เพจที่เหลืออยู่ เพื่อให้การหยุดนี้สั้นที่สุดที่จะเป็นไปได้ TLC จะสร้างเทรดขึ้นมาจำนวนหนึ่ง (n Threads ในรูปที่ 1 โดยที่ในงานวิจัยนี้ n คือจำนวนซีพียูคอร์ทั้งหมดที่เครื่องคอมพิวเตอร์จริงมี) ให้ช่วยกันทำสำเนาของ Dirty เพจเหล่านั้นไปยังพื้นที่อีกส่วนหนึ่งในหน่วยความจำของเครื่องคอมพิวเตอร์จริงเรียกว่า บัฟเฟอร์

ขั้นตอนที่สี่ TLC จะสั่งให้ทุกๆเทรดของคอมพิวเตอร์เสมือนประมวลผลต่อและให้ CHK Thread เขียนข้อมูลในบัฟเฟอร์ลงเก็บในCheckpoint File ไปในขณะเดียวกัน และจบการทำเช็คพอยต์หลังจากนั้น

4. ผลการทดลองเบื้องต้น

ผู้วิจัยได้ทำการทดลองเบื้องต้นเพื่อทดสอบประสิทธิภาพของระบบ TLC โดยเตรียมการทดลองดังรูปที่ 2



รูปที่ 2 ระบบคอมพิวเตอร์ที่ใช้ในการทดลอง

จากรูปผู้วิจัยติดตั้งคอมพิวเตอร์จริงจำนวน 3 เครื่อง ได้แก่ sushi2 , sushi3 , sushi4 ใช้ระบบปฏิบัติการ Ubuntu Server เวอร์ชั่น 11.10 เครื่องมี CPU แบบ AMD 12 Core 2.1 GHz มีหน่วยความจำขนาด 48 GB มี HDD แบบ SISC 15K rpm ขนาด 900 GB ทั้ง 3 เครื่องเชื่อมต่อกันด้วยเนตเวิร์คความเร็วสูงขนาด 10 GB โดยที่เครื่อง sushi3 นั้นติดตั้ง KVM ที่ทำ TLC และรันระบบคอมพิวเตอร์เสมือนและแอพพลิเคชั่นที่ใช้ในการทดสอบ

Images ของคอมพิวเตอร์เสมือนทั้งหมดที่ใช้ในการทดสอบรวมทั้ง Checkpoint File จะอยู่บนเครื่อง sushi2 และเข้าถึงได้ผ่าน NFS ส่วนฐานข้อมูล NoSQL นั้นผู้วิจัยติดตั้ง Couchbase-Server เวอร์ชั่น 1.8 ที่เครื่องโฮสต์ sushi-2 และ sushi-4 และกำหนดให้แต่ละเครื่องรองรับ ข้อมูลในหน่วยความจำได้ 16 GB

ผู้วิจัยได้ติดตั้งเครื่องคอมพิวเตอร์เสมือนบนเครื่องโฮสต์คอมพิวเตอร์ sushi-3 โดยใช้ Guest OS เป็น Ubuntu เวอร์ชั่น 10.04 โดยกำหนดให้ใช้ SMP CPU 8 Core และมีหน่วยความจำ 16 GB เครื่องคอมพิวเตอร์เสมือนรัน Benchmark ที่ใช้เป็นโปรแกรมที่ใช้สำหรับวัดประสิทธิภาพการทำงานทางวิทยาศาสตร์โดยนำมาทำการทดสอบ TLC โปรแกรมวัดประสิทธิภาพนี้ได้มาจาก NAS Parallel benchmark ที่มีการใช้ CPU และ Memory จำนวนมาก ผู้วิจัยทำการคอมไพล์โดยใช้ Open MP สำหรับการทดลองครั้งนี้ได้เลือกมา 3 ชุดโปรแกรม คือ SP class D LU class D และ BT class D แต่ละชุดโปรแกรมจะใช้ CPU cores ทั้งหมดของเครื่องคอมพิวเตอร์เสมือนและใช้ Memory สูงสุด 12.1 GB, 9.6 GB และ 11.8 GB ตามลำดับ

ตารางที่ 1 : ผลการทดลองใน Stage 2 วัดผลด้วย NPB Benchmark

Benchmark	Stage 2.1			Stage 2.2
	Saved Pages	Skip Pages	Total Pages	PS: pages added to Couchbase
sp.D.x	4,194,475	4,021	4,198,496	181,925
lu.D.x	4,186,389	12,107	4,198,496	173,753
bt.D.x	4,178,348	20,148	4,198,496	201,001

ตารางที่ 2 : ผลการทดลองใน Stage 3 และ 4 ของ NPB Benchmark

Benchmark	Stage 3		Stage 4
	Pages Copied to Buffer	Pause Duration Time[sec]	Duration Time Save Buffer to File [sec]
sp.D.x	2,314,523	107	100.62
lu.D.x	1,663,001	61	68.97
bt.D.x	2,136,786	95	100.7

4.1 การวัดประสิทธิภาพของระบบ TLC

ผู้วิจัยได้การทดลองการสำหรับทำการเช็คพอยน์เพื่อเก็บสถานะการทำงานของคอมพิวเตอร์เสมือนแบบวิธี TLC ใช้เวอร์เชอร์แมชชีนที่เป็นเกส ขนาด 16 GB คิดเป็นจำนวน memory pages sizeได้ทั้งหมด 4,194,304 pages รวมกับ pages ที่ใช้สำหรับ ROM และ Device ต่างๆจะมีจำนวนรวมคือ 4,198,496 pages

จากตารางที่ 1 ที่ Stage 2.1 แสดงจำนวนของ Page ทั้งหมดที่ถูกเก็บโดยเมื่อทดสอบการทำงานโดยใช้ Benchmark ได้แก่ sp Class D ได้ค่าจำนวน Page ที่ถูกจัดเก็บลงสู่ Checkpoint File เป็น 4,194,475 pages และจำนวน page ที่ไม่ถูกจัดเก็บ เนื่องจากเป็น Pages ถูกจัดเก็บไปแล้ว 4,021 pages เมื่อทดสอบโดยใช้ lu Class D ได้ค่าจำนวน Page ที่ถูกจัดเก็บ 4,186,389 pages และจำนวน page ที่ไม่ถูกจัดเก็บ 12,107 pages และเมื่อวัดโดยใช้ bt Class D ได้ค่าจำนวน Page ที่ถูกจัดเก็บ 4,178,348 pages และจำนวน page ที่ไม่ถูกจัดเก็บ 20,148 pages

จากตารางที่ 1 ที่ Stage 2.2 แสดงจำนวนของ Page ทั้งหมดที่ถูกส่งไปจัดเก็บในระบบฐานข้อมูล NoSQL (Couchbase-Server) เมื่อทดสอบการทำงานโดยใช้ Benchmark ได้แก่ sp Class D ได้ค่าจำนวน Page ที่ถูกส่งไปจัดเก็บ 181,925 pages เมื่อทดสอบด้วย lu Class D ได้ค่าจำนวน Page ที่ถูกส่งไปจัดเก็บ 173,753 pages และเมื่อทดสอบโดยใช้ bt Class D ได้ค่าจำนวน Page ที่ถูกส่งไปจัดเก็บ 201,001 pages การใช้ NoSQL เพื่อช่วยเก็บข้อมูลของ TLC ช่วยทำให้ จำนวนเพจที่เหลืออยู่สำหรับจัดเก็บใน Stage 3 น้อยลง

จากตารางที่ 2 ที่ Stage 3 แสดงจำนวนของ Page ทั้งหมดที่ถูกส่งไปจัดเก็บลงบัฟเฟอร์ ซึ่งเป็นขั้นตอนที่ทำการหยุดคอมพิวเตอร์เสมือนเพื่อทำการจัดเก็บ Dirty Pages และทำการสร้างเทรดจำนวนเท่ากับ CPU ของโฮสต์คือ 12 เทรด มาช่วยการจัดเก็บข้อมูลไปที่บัฟเฟอร์เมมมอรีของโฮสต์ เมื่อทดสอบด้วย Benchmark ได้แก่ sp Class D ได้ค่าจำนวน Page ที่ถูกจัดเก็บ 2,314,523 pages ใช้เวลาในการหยุด 107 วินาที

เมื่อทดสอบด้วย lu Class D ได้ค่าจำนวน Page ที่ถูกจัดเก็บ 1,663,001 pages ใช้เวลาในการหยุด 61 วินาที และเมื่อทดสอบโดยใช้ bt Class D ได้ค่าจำนวน Page ที่ถูกจัดเก็บ 2,136,786 pages ใช้เวลาในการหยุด 95 วินาที ซึ่งเวลาเหล่านี้เป็นเวลาที่ TLC หยุดพักการคอมพิวเตอร์เสมือนอย่างต่อเนื่องนานที่สุด ซึ่งน้อยกว่าการเช็คพอยต์แบบเดิมมาก ดังที่จะได้กล่าวถึงต่อไป

จากรูปตารางที่ 2 Stage 4 แสดงจำนวนระยะเวลาในการจัดเก็บข้อมูลที่อยู่บนบัฟเฟอร์ลงสู่ไฟล์ เมื่อทดสอบด้วย Benchmark ได้แก่ sp Class D ใช้เวลา 100.62 วินาที เมื่อทดสอบด้วย lu Class D ใช้เวลา 100.62 วินาที 68.97 วินาที และเมื่อทดสอบโดยใช้ bt Class D ใช้เวลา 100.70 วินาที

ตารางที่ 3 : ระยะเวลาทั้งหมดที่ใช้การหยุดของ TLC เทียบกับการเช็คพอยต์แบบเดิมของ KVM

Benchmark	TLC Acc.Overheads[sec]	KVM[sec]	TLC/KVM
sp.D.x	149.10	1,393.34	0.11
lu.D.x	102.02	1,314.74	0.08
bt.D.x	154.18	1,123.09	0.14

4.2 เปรียบเทียบประสิทธิภาพของ TLC กับวิธีเดิม

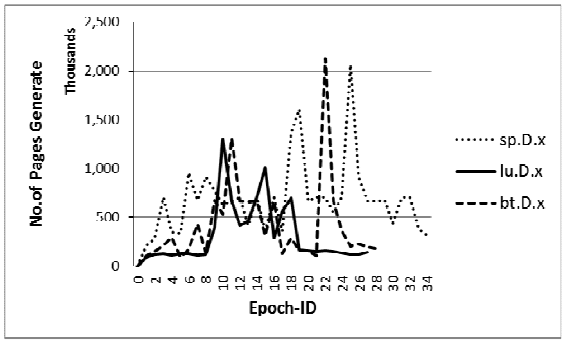
จากตารางที่ 3 ระยะเวลารวมที่ใช้ในการหยุดที่ stage 2 ด้วยวิธีแบบ TLC ซึ่งเกิดจากการกำหนดให้มีการขัดจังหวะการทำงานของ I/O ทุกๆ 3 วินาที เพื่อที่จะทำการตรวจสอบว่ามี Dirty Pages เกิดขึ้นใหม่ ณ เวลานั้น รวมกับเวลาในการหยุดของ Stage 3 เมื่อทดสอบด้วย Benchmark โดยใช้โปรแกรม sp Class D ใช้เวลาหยุดรวม 149.10 วินาที เมื่อทดสอบโดยใช้โปรแกรม lu Class D ใช้เวลาหยุดรวม 102.02 และเมื่อทดสอบโดยใช้โปรแกรม bt Class D ใช้เวลาหยุดรวม 154.18 วินาที ในขณะที่เมื่อทำการเปรียบเทียบกับการหยุดระบบเพื่อทำเช็คพอยต์แบบเดิมของ KVM จะใช้เวลาหยุดรวม 1,393.34 วินาที 1,314.74 วินาที และ 1,123.09 วินาที เมื่อทดสอบโดยใช้โปรแกรม sp Class D , lu Class D

และ bt Class D ตามลำดับ ดังนั้นการทำงานด้วยวิธี TLC ใช้เวลาน้อยกว่าเป็น 0.11 0.08 และ 0.14 ของวิธีเดิม เมื่อทดสอบ Benchmark ได้แก่ sp Class D , lu Class D และ bt Class D ตามลำดับ

ตารางที่ 4 : ระยะเวลาที่ระบบหยุดการทำงานใน Stage 3

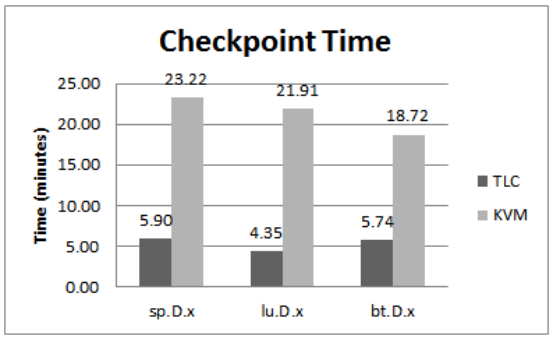
Benchmark	Max Pause Duration[sec]	KVM[sec]	TLC/KVM
sp.D.x	106.78	1,393.34	0.08
lu.D.x	60.69	1,314.74	0.05
bt.D.x	94.66	1,123.09	0.08

จากตารางที่ 4 ระยะเวลารวมที่ให้ในการหยุดที่ stage 3 ซึ่งเป็นขั้นตอนการเก็บสถานะครั้งสุดท้าย เมื่อใช้โปรแกรมแบบ TLC ทดสอบด้วย Benchmark ได้แก่ sp Class D ใช้เวลาหยุดรวม 106.78 วินาที เมื่อทดสอบด้วย lu Class D ใช้เวลาหยุดรวม 60.69 วินาที และเมื่อทดสอบโดยใช้ bt Class D ใช้เวลาหยุดรวม 94.66 วินาที ในขณะที่เมื่อทำการเปรียบเทียบกับการหยุดระบบเพื่อทำการเก็บสถานะการทำงานของโปรแกรมแบบ KVM ทดสอบด้วย Benchmark ได้แก่ sp Class D จะใช้เวลาหยุดรวม 1,393.34 วินาที ทดสอบด้วย lu Class D ใช้เวลาหยุดรวม 1,314.74 วินาที และทดสอบด้วย bt Class D ใช้เวลาหยุดรวม 1,123.09 วินาที ดังนั้นการทำงานด้วยวิธี TLC ใช้เวลาน้อยกว่าเป็น 0.08 0.05 และ 0.08 ของวิธีเดิม เมื่อทดสอบ Benchmark ได้แก่ sp Class D , lu Class D และ bt Class D ตามลำดับ



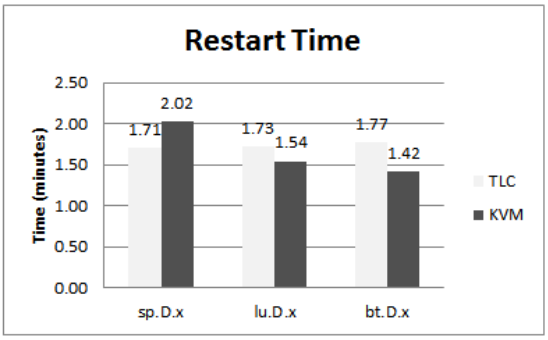
รูปที่ 3 : จำนวน Pages ที่ Dirty ในแต่ละ Epoch ID ในช่วง State 2

จากรูปที่ 3 จำนวน Dirty Pages ที่ถูกสร้างขึ้นมาใหม่ในแต่ละช่วงเวลา โดยที่ใน Stage ที่ 2 ของการทำงานแบบ TLC จะถูกกำหนดให้ทำการขัดจังหวะการทำงานของเวอร์ชวลแมชชีนทุกๆ 3 วินาที เพื่อทำการตรวจเช็คว่ามี Dirty Page ที่ถูกสร้างขึ้นมาใหม่อยู่จำนวนเท่าใด แล้วทำการนำไปจัดเก็บเข้าไว้ที่ NoSQL ดาต้าเบส เมื่อวัดผลด้วยโปรแกรม sp Class D จะพบว่ามีจำนวน Epoch_ID ทั้งหมด 35 (0 - 34) และมี Epoch_ID ที่ทำการสร้าง Dirty Pages ใหม่สูงสุดจำนวน 2,061,721 Pages ที่ Epoch-ID ที่ 25 เมื่อวัดผลด้วยโปรแกรม lu.D.x จะพบว่ามีจำนวน Epoch_ID ทั้งหมด 28 (0 - 27) และมี Epoch-ID ที่ทำการสร้าง Dirty Pages ใหม่สูงสุดจำนวน 1,301,169 Pages ที่ Epoch_ID ที่ 10 และเมื่อวัดผลด้วยโปรแกรม bt Class x จะมีจำนวน Epoch_ID ทั้งหมด 29 (0 - 28) และมี Epoch-ID ที่ทำการสร้าง Dirty Pages ใหม่สูงสุดจำนวน 2,132,551 Pages ที่ Epoch_ID ที่ 22



รูปที่ 4 เวลาที่ใช้ในการ Checkpoint ของ TLC กับของ KVM

จากรูปที่ 4 ระยะเวลาในการเช็คพอยต์ด้วยวิธีแบบ TLC ใช้ระยะเวลา น้อยกว่า 0.25 0.20 และ 0.31 เท่า เมื่อทดสอบกับ sp Class D , lu Class D และ bt Class D ตามลำดับ เพราะการเขียนข้อมูลลงสู่ดิสก์ของวิธีการเดิมต้องใช้เวลามาก เพราะต้องส่งข้อมูลผ่าน NFS และข้อมูลจะต้องถูกนำไปแทรกเก็บลงบนไฟล์ Image ของคอมพิวเตอร์เสมือน แต่วิธีการแบบ TLC ใช้การเขียนข้อมูลลงสู่ไฟล์แบบซีเควนเชียล และเก็บข้อมูลส่วนหนึ่งใน NoSQL จึงทำได้รวดเร็วกว่า



รูปที่ 5 Restart Time

จากรูปที่ 5 ผลการทดลองการเรียกคืนสถานะกลับมาของคอมพิวเตอร์เสมือนเวอร์ชั่น TLC กับแบบเดิมใช้ระยะเวลาในการเรียกคืนสถานะใกล้เคียงกัน คิดเป็น 0.84 1.12 และ 1.25 ของวิธีเดิม โดยที่วิธีการเรียกคืนสถานะแบบด้วยวิธีการแบบ TLC ใช้ระยะเวลา 1.71 1.73 และ 1.77 นาที ส่วนวิธีการเรียกคืนสถานะแบบเดิมใช้ระยะเวลา 2.02 1.54 และ 1.42 นาที เมื่อทดสอบกับ sp Class D , lu Class D และ bt Class D ตามลำดับ

5. สรุป

งานวิจัยนี้ทำการปรับปรุงประสิทธิภาพการทำงานของการเช็คพอยน์สำหรับคอมพิวเตอร์เสมือน จากเดิมหากต้องการเก็บสถานะการทำงานของคอมพิวเตอร์เสมือน จะต้องทำการหยุดคอมพิวเตอร์เสมือนซึ่งใช้เวลานาน

ผู้วิจัยได้ทำการปรับปรุงประสิทธิภาพโดยเสนอระบบ TLC ที่ใช้ เทรดมาช่วยสนับสนุนการเช็คพอยต์ และลดระยะเวลาในเก็บสถานะของคอมพิวเตอร์เสมือนระบบ TLC ใช้ระบบฐานข้อมูล NoSQL ที่เรียกว่า Couchbase มาประยุกต์ใช้ เพื่อช่วยจัดเก็บข้อมูลเพจของหน่วยความจำซึ่งทำให้เหลือจำนวนเพจที่ต้องหยุดคอมพิวเตอร์เสมือนเพื่อจัดเก็บน้อยลง ทำให้ TLC หยุดพักการทำงานของคอมพิวเตอร์เสมือนน้อยกว่าการเช็คพอยต์แบบเดิมมาก

และนอกจากนั้นเวลาที่ใช้ในการเช็คพอต์โดยรวม น้อยกว่าวิธีเดิมมากเช่นกัน ส่วนผลการทดลองเพื่อทำการเรียกคืนสถานะของคอมพิวเตอร์เสมือนแบบ TLC และแบบ KVM ให้ผลการทดลองใกล้เคียงกัน ซึ่งผู้วิจัยจะได้วิเคราะห์รายละเอียดต่อไป

งานวิจัยนี้ก็ยังต้องทำการศึกษาเพิ่มเติมเกี่ยวกับการจัดเก็บข้อมูลเพจด้วยเซิร์ฟเวอร์ที่ใช้เก็บข้อมูลเพจโดยเฉพาะ เพื่อเพิ่มประสิทธิภาพในการทำงานของ TLC

เอกสารอ้างอิง

[1] http://www.couchbase.com

[2] NPB, http://www.nas.nasa.gov/Resources/Software/npb_changes.html

[3] M. Litzkow and M. Solomon, "Supporting Checkpointing andProcess Migration Outside the UNIX Kernel", *Usenix Conf.* 1992.

[4] J. S. Plank, et al. “Libckpt: Transparent Checkpointing under Unix,” Proc. of the 1995 Winter USENIX Tech Conf. 1995.

[5] R. Gioiosa, et al. “Transparent, Incremental Checkpointing at Kernel Level: a Foundation for Fault Tolerance for Parallel Computers,” *Proc. of the 2005 ACM/IEEE conference on Supercomputing* , 2005.

[6] http://www.vmware.com

[7] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, “kvm: the linux virtual machine monitor,” *Proc. of Linux Symposium*, 2007.

[8] P. Barham, et al. “Xen and the Art of Virtualization,” *Proc. of the ACM SOSP*, 2003.

[9] Vasinee Siripoonya and Kasidit Chanchio, "Thread-based Live Checkpointing of Virtual Machines" The 10th IEEE International Symposium on Network Computing and Applications, August, 2011.