การเพิ่มประสิทธิภาพการอพยพแบบคงสถานะการทำงานของคอมพิวเตอร์เสมือนโดยใช้เครื่องแม่ข่ายด้วยการบีบอัดข้อมูลสองชั้น

โดย

นางสาววิตรี พันวินิต

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร
วิทยาศาสตรมหาบัณฑิต (วิทยาการคอมพิวเตอร์)
ภาควิชาวิทยาการคอมพิวเตอร์
คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยธรรมศาสตร์
ปีการศึกษา 2558

การเพิ่มประสิทธิภาพการอพยพแบบคงสถานะการทำงานของคอมพิวเตอร์เสมือนโดยใช้เครื่องแม่ข่ายด้วยการบีบอัดข้อมูลสองชั้น

โดย

นางสาวสาวิตรี พันวินิต

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร
วิทยาศาสตรมหาบัณฑิต (วิทยาการคอมพิวเตอร์)
ภาควิชาวิทยาการคอมพิวเตอร์
คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยธรรมศาสตร์
ปีการศึกษา 2558

EFFICIENT LIVE MIGRATION OF VIRTUAL MACHINES USING TWO-LEVEL DATA COMPRESSION

BY

MISS SAVITRI PHANVINIT

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 2015

มหาวิทยาลัยธรรมศาสตร์
คณะวิทยาศาสตร์และเทคโนโลยี

วิทยานิพนธ์

ของ

นางสาวสาวิตรี พันวินิต

เรื่อง

การเพิ่มประสิทธิภาพการอพยพแบบคงสถานะการทำงานของคอมพิวเตอร์เสมือน
โดยใช้เครื่องแม่ข่ายด้วยการบีบอัดข้อมูลสองชั้น

ได้รับการตรวจสอบและอนุมัติ ให้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร
วิทยาศาสตรมหาบัณฑิต (วิทยาการคอมพิวเตอร์)

เมื่อ วันที่ 7 กรกฎาคม พ.ศ. 2559

ประธานกรรมการสอบวิทยานิพนธ์ _____
(ผู้ช่วยศาสตราจารย์ ดร. เสาวลักษณ์ วรรธนาภา)

กรรมการและอาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก _____
(ผู้ช่วยศาสตราจารย์ ดร. กษิดิศ ชาญเชี่ยว)

กรรมการสอบวิทยานิพนธ์ _____
(ดร. เด่นดวง ประดับสุวรรณ)

กรรมการสอบวิทยานิพนธ์ _____
(ดร. ศรเทพ วรรณรัตน์)

คณบดี _____
(รองศาสตราจารย์ ปกรณ์ เสริมสุข)

หัวข้อวิทยานิพนธ์	การเพิ่มประสิทธิภาพการอพยพแบบคงสถานะการทำงานของคอมพิวเตอร์เสมือนโดยใช้เครื่องแม่ข่ายด้วยการบีบอัดข้อมูลสองชั้น
ชื่อผู้เขียน	นางสาวสาวิตรี พันวินิต
ชื่อปริญญา	วิทยาศาสตรมหาบัณฑิต
สาขาวิชา/คณะ/มหาวิทยาลัย	สาขาวิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยธรรมศาสตร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์	ผู้ช่วยศาสตราจารย์ ดร. กษิดิศ ชาญเชี่ยว
ปีการศึกษา	2558

บทคัดย่อ

เทคโนโลยีระบบคอมพิวเตอร์เสมือนทำให้สามารถใช้คอมพิวเตอร์ได้อย่างคุ้มค่าและเกิดประโยชน์สูงสุด เมื่อเกิดปัญหากับเครื่องคอมพิวเตอร์จริงที่คอมพิวเตอร์เสมือนปฏิบัติการอยู่ ผู้ดูแลระบบสามารถใช้เทคนิคการอพยพแบบคงสถานะการทำงานในการอพยพคอมพิวเตอร์เสมือนโดยไม่จำเป็นต้องหยุดการทำงานของคอมพิวเตอร์เสมือน ทำให้ระบบคอมพิวเตอร์เสมือนสามารถทำงานได้อย่างต่อเนื่องและไม่ส่งผลกระทบต่อผู้ใช้งาน อย่างไรก็ตาม การอพยพคอมพิวเตอร์เสมือนขนาดใหญ่ที่มีการประมวลผลหรือใช้หน่วยความจำปริมาณมาก ในปัจจุบันยังไม่สามารถทำได้อย่างมีประสิทธิภาพ และหากทำการอพยพคอมพิวเตอร์เสมือนผ่านเครือข่ายที่มีแบนด์วิดท์ต่ำ ยิ่งทำให้ประสิทธิภาพการอพยพของคอมพิวเตอร์เสมือนแย่ลง ดังนั้นจึงได้มีการพัฒนาการอพยพแบบคงสถานะการทำงานแบบเทรด ซึ่งพัฒนาขึ้นบนซอฟต์แวร์เควีเอ็ม โดยสร้างเทรดเพื่อถ่ายโอนสถานะหรือหน่วยความจำของคอมพิวเตอร์เสมือนแบบขนาน หลังจากนั้นได้เพิ่มเครื่องแม่ข่าย ในการบีบอัดข้อมูล ในงานวิจัยนี้ผู้วิจัยได้พัฒนาต่อยอดโดย นำเสนอ Framework เพื่อบีบอัดหน่วยความจำในระดับเพจ ทำให้หน่วยความจำที่ถูกส่งผ่านไปบนเครือข่ายถูกบีบอัด 2 ชั้น ส่งผลให้ข้อมูลที่ส่งผ่านเครือข่ายลดลง และส่งผลกระทบต่อการทำงานของคอมพิวเตอร์เสมือนน้อยกว่าวิธีการอพยพที่มีอยู่ ผลการวิจัยพบว่า วิธีการบีบอัด 2 ชั้นสามารถช่วยให้การอพยพคอมพิวเตอร์เสมือนที่ใช้หน่วยประมวลผลและหน่วยความจำมากได้อย่างมีประสิทธิภาพเพิ่มขึ้น และมีประสิทธิภาพยิ่งขึ้นเมื่ออพยพผ่านเครือข่ายที่มีแบนด์วิดท์ 100 Mbps

คำสำคัญ: คอมพิวเตอร์เสมือน, การอพยพแบบคงสถานะการทำงาน, การบีบอัดข้อมูล

Thesis Title	Efficient Live Migration of Virtual Machines Using Two-Level Data Compression
Author	Miss Savitri Phanvinit
Degree	Master of Science
Major Field/Faculty/University	Computer Science Faculty of Science and Technology Thammasat University
Thesis Advisor	Asst. Prof. Dr. Kasidit Chanchio
Academic Years	2015

ABSTRACT

Virtualization technology has increased computer usability, allowing administrators to perform maintenance and allocate resources efficiently. When computers have issues or needs system scale adjustment, administrators may use live migration to migrate the running virtual machine to the physical host with minimal downtime. In doing so, the computer retains high availability and end-user experience remains unaffected. However, migrating CPU and/or a memory intensive virtual machine may cause service interruption or prolonged migration time or even failure. These are more problematic for migration over a low-bandwidth network. Thread-based Live Migration is implemented on Open Source KVM software. Additional threads are added to transfer the virtual machine state over parallel TCP connection and external data compression server. To further improve performance of Thread-based Live Migration, a two-level data compression framework is proposed in this thesis to reduce the amount of data transferred over the network and decrease downtime and total migration time. Results show that two-level data compression can improve the performances, especially when migrating virtual machines over a 100 Mbps network.

Keywords: Virtual machine, Live migration, Data compression

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงได้เป็นอย่างดี เนื่องจากได้รับความอนุเคราะห์จากผู้ช่วยศาสตราจารย์ ดร. กษิดิศ ชาญเชี่ยว ซึ่งเป็นอาจารย์ที่ปรึกษาและควบคุมวิทยานิพนธ์ เป็นผู้จุดประกายหัวข้อวิทยานิพนธ์ ให้คำปรึกษาแนะแนว ช่วยตรวจทานและแก้ไขความถูกต้องของเนื้อหาตลอดจนให้กำลังใจแก่ผู้วิจัยเสมอมา ผู้วิจัยซาบซึ้งในความกรุณา และขอกราบขอบพระคุณเป็นอย่างสูงไว้ ณ โอกาสนี้ด้วย

ขอขอบคุณ ผู้ช่วยศาสตราจารย์ ดร. เสาวลักษณ์ วรรธนาภา ประธานกรรมการสอบวิทยานิพนธ์ ดร. เด่นดวง ประดับสุวรรณ และ ดร. ศรเทพ วรรณรัตน์ กรรมการสอบวิทยานิพนธ์ ที่ให้ข้อเสนอแนะในการปรับปรุงแก้ไขวิทยานิพนธ์ให้มีความสมบูรณ์ยิ่งขึ้น

ขอขอบคุณอาจารย์ภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยธรรมศาสตร์ ที่ถ่ายทอดวิชาความรู้ ตลอดจนเจ้าหน้าที่ภาคช่วยอำนวยความสะดวกในทุกด้านแก่ผู้วิจัยตลอดระยะเวลาที่ศึกษาอยู่ โดยเฉพาะอย่างยิ่ง คุณพิมพ์ภาวรรณ ฉิมชาญเวช

ขอขอบคุณป๊าม้า อาเจ๊ก อาโกว อาซิ่ม กล้วย เหงี่ยนและเพื่อนร่วมรุ่นทุกท่านที่คอยเอาใจใส่ ห่วงใย ช่วยเหลือเกื้อกูลและเป็นกำลังใจให้กันมาโดยตลอด

นางสาวสาวิตรี พันวินิต

สารบัญ

สารบัญตาราง

สารบัญภาพ

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

เทคโนโลยีระบบคอมพิวเตอร์เสมือน (Virtualization Technology) ทำให้เครื่องคอมพิวเตอร์เครื่องหนึ่งมีระบบปฏิบัติการได้หลายระบบและหลายแพลตฟอร์ม ด้วยการใช้ซอฟต์แวร์ในการจำลองคอมพิวเตอร์เสมือนหลายๆ เครื่องบนคอมพิวเตอร์จริงเพียงเครื่องเดียว โดยแต่ละคอมพิวเตอร์เสมือนนั้น มีหน่วยความจำ ฮาร์ดดิสก์ และอุปกรณ์เน็ตเวิร์กที่เป็นอิสระต่อกัน ซึ่งเทคโนโลยีนี้ช่วยให้เกิดการจัดสรรทรัพยากรอย่างมีประสิทธิภาพและหลากหลาย ลดต้นทุนในบริหารจัดการ เพิ่มความยืดหยุ่นในการใช้ทรัพยากรของเครื่อง และเอื้ออำนวยต่อการปรับเปลี่ยนขนาดของระบบ

การอพยพแบบคงสถานะการทำงาน (Live Migration) เป็นฟีเจอร์ของคอมพิวเตอร์เสมือน (Virtual Machine) ที่ทำให้การอพยพคอมพิวเตอร์เสมือนต้นทางที่กำลังทำงานอยู่ไปทำงานบนคอมพิวเตอร์เสมือนปลายทางได้โดยไม่จำเป็นต้องหยุดการทำงานของคอมพิวเตอร์เสมือนหรือ ใช้เวลาในการหยุดน้อยที่สุด ในปัจจุบัน ได้มีการนำการอพยพแบบคงสถานะการทำงานมาใช้อย่างแพร่หลายในการบริหารจัดการทรัพยากรของระบบ ตัวอย่างในการนำการอพยพแบบคงสถานะการทำงานไปใช้งาน เช่น เมื่อมีการใช้ทรัพยากรของเครื่องมากเกินกว่าที่เครื่องจะรองรับได้ เมื่อต้องการปิดปรับปรุงระบบ เมื่อต้องการเพิ่มประสิทธิภาพของโปรแกรมประยุกต์ก็สามารถทำการอพยพแบบคงสถานะการทำงานไปยังเครื่องที่มีสมรรถนะสูงขึ้น หรือ ต้องการจัดสรรทรัพยากรระบบให้มีประสิทธิภาพมากขึ้น โดยอพยพคอมพิวเตอร์เสมือนที่กระจายกันอยู่ให้มารวมกันในคอมพิวเตอร์จริงเครื่องเดียวกัน หรือแม้แต่เมื่อเกิดปัญหากับอุปกรณ์ฮาร์ดแวร์ของเครื่องคอมพิวเตอร์จริง ก็สามารถอพยพคอมพิวเตอร์เสมือนไปยังเครื่องจริงเครื่องอื่นได้ แม้เมื่อเกิดภัยพิบัติต่างๆ ต้องการอพยพคอมพิวเตอร์เสมือนไปในที่ห่างไกล อีกซีกโลกหนึ่งก็ทำได้เช่นกันด้วยเทคโนโลยีการอพยพแบบคงสถานะการทำงาน

การอพยพแบบคงสถานะการทำงานใช้เทคนิคการถ่ายโอนข้อมูลหน่วยความจำของคอมพิวเตอร์เสมือน ในขณะที่หน่วยความจำของคอมพิวเตอร์เสมือนยังคงถูกใช้งานอยู่ เนื่องจากความก้าวหน้าทางเทคโนโลยี เครื่องคอมพิวเตอร์มีสมรถนะสูง ซอฟต์แวร์มีความหลากหลายและบางซอฟต์แวร์มีการประมวลผลแบบใช้หน่วยความจำในปริมาณมาก (Memory Intensive) ซึ่งมากและเร็วกว่าที่การอพยพแบบคงสถานะการทำงานในปัจจุบัน จะสามารถถ่ายโอนหน่วยความจำของ

คอมพิวเตอร์เสมือนผ่านเครือข่ายได้อย่างมีประสิทธิภาพ ในบางกรณีจำเป็นจะต้องหยุดการทำงานของคอมพิวเตอร์เสมือนเป็นเวลานาน บางกรณีใช้เวลารวมสำหรับอพยพคอมพิวเตอร์เสมือนยาวนานมาก

ด้วยเหตุนี้ จึงมีงานวิจัยเกิดขึ้นมากมายเพื่อเพิ่มประสิทธิภาพของการอพยพแบบคงสถานะการทำงาน ด้วยการลดเวลารวมสำหรับการอพยพคอมพิวเตอร์เสมือน (Total Migration Time) และเวลาหยุดการทำงานของคอมพิวเตอร์เสมือน (Downtime) หนึ่งในงานวิจัยนั้นคือ Time-bound Thread-based Live Migration (TLM) (Kasidit Chanchio & Phithak Thaenkaew, 2014) ซึ่งเป็นเทคนิคการอพยพแบบคงสถานะการทำงานที่ทำการปรับปรุง KVM (Kernel-based Virtual Machine) ด้วยการสร้างเทรด (Thread) เพิ่มขึ้นมาช่วยในการถ่ายโอนหน่วยความจำ และทำงานไปพร้อมกับเทรดหลักของคอมพิวเตอร์เสมือน สามารถอพยพคอมพิวเตอร์เสมือนที่มีหน่วยประมวลผลความเร็วสูงและใช้หน่วยความจำมากได้สำเร็จในขอบเขตของเวลา ส่งผลให้เวลารวมสำหรับอพยพคอมพิวเตอร์เสมือนลดลง เมื่อเทียบกับการอพยพแบบคงสถานะการทำงานของ KVM เดิม อย่างไรก็ตาม TLM ใช้วิธี CPU Over-committed เพื่อลดการเกิดเพจที่เปลี่ยนแปลง (dirty page) ทำให้เวลาหยุดทำงานของคอมพิวเตอร์เสมือนน้อยลง ซึ่งวิธีนี้ส่งผลต่อการทำงานของคอมพิวเตอร์เสมือน จึงมีงานวิจัยที่ ได้ปรับปรุง TLM ด้วยการเพิ่มเครื่องแม่ข่ายที่ทำหน้าที่บีบอัดข้อมูล (Z-Server) ทำให้ข้อมูลที่ถูกถ่ายโอนมีขนาดเล็กลง ชื่อ TLM-Z: Thread-Based Live Migration of Virtual Machines Using Data Compression Servers (พิทักษ์ แท่นแก้ว, 2557) ซึ่งสามารถลดเวลารวมสำหรับการอพยพและเวลาหยุดการทำงานของคอมพิวเตอร์เสมือนลงสำหรับโปรแกรมจำนวนหนึ่ง และลดได้มากขึ้นเมื่อเครือข่ายมีแบนด์วิดท์ลดลง

ในงานวิจัยนี้มีวัตถุประสงค์ที่จะเพิ่มประสิทธิภาพการอพยพแบบคงสถานะการทำงานแบบเทรดด้วยการบีบอัด โดยผู้วิจัยเห็นว่าการถ่ายโอนข้อมูลเฉพาะที่มีการเปลี่ยนแปลงน่าจะทำให้ข้อมูลที่ถูกถ่ายโอนผ่านเครือข่ายมีปริมาณน้อยลง และการบีบอัดข้อมูลของ Z-Server มีประสิทธิภาพเพิ่มขึ้น รวมทั้งการเพิ่มการบีบอัดอีกชั้น จะทำให้ปริมาณหน่วยความจำที่ถูกถ่ายโอนผ่านเครือข่ายน้อยลง ซึ่งจะส่งผลให้เวลารวมสำหรับการอพยพ (Total Migration Time) และเวลาหยุดการทำงานของคอมพิวเตอร์เสมือน (Downtime) ลดลง ดังนั้น งานวิจัยนี้จึงได้การนำเสนอ Framework สำหรับการบีบอัดแบบใหม่ที่เพิ่ม Memory Server เข้ามาระหว่างคอมพิวเตอร์เสมือนและเครื่องแม่ข่าย Z-Server เพื่อจัดการบีบอัดหน่วยความจำ 2 วิธี ได้แก่

(1) ด้วยการเข้ารหัสแบบเดลตา (Delta Compression) โดยใช้วิธีการ Exclusive OR (XOR) ในงานวิจัยนี้เราอ้างอิงถึงการอพยพแบบคงสถานะการทำงานด้วยเฟรมเวิร์คบีบอัดข้อมูลสองชั้นแบบนี้ว่า TLM-XZ

(2) ด้วยการเพิ่มการบีบอัดข้อมูลด้วย Blosc (A Blocking, shuffling and lossless compression library) (Alted, 2016) ในงานวิจัยนี้เราอ้างอิงถึงการอพยพแบบคงสถานะการทำงานโดยใช้การบีบอัดข้อมูลสองชั้นแบบนี้ว่า TLM-BZ

ผู้วิจัยได้สร้างระบบต้นแบบของ Framework และ Memory Server ข้างต้นและทำการทดลองโดยอพยพเครื่องคอมพิวเตอร์เสมือนที่รันโปรแกรมแบบ OpenMP จาก NAS Parallel Benchmark (NPB) (Dunbar, n.d.) ผ่านระบบเครือข่ายแบบ 1 Gbps และ 100 Mbps ตามลำดับ โปรแกรมที่ใช้ทดสอบได้แก่ (1) โปรแกรม SP Class C (SP.C) (2) โปรแกรม BT Class C (BT.C) (3) CG Class C (CG.C) (4) IS Class C (IS.C) (5) LU Class C (LU.C) (6) MG Class C (MG.C) และ (7) UA Class C (UA.C) ซึ่งเป็นโปรแกรมประมวลผลทางวิทยาศาสตร์ที่ใช้หน่วยความจำและหน่วยประมวลผลมาก

ผลการวิจัยพบว่า

(1) การอพยพแบบคงสถานะการทำงานแบบ TLM-XZ มีประสิทธิภาพใกล้เคียงกับการอพยพแบบคงสถานะการทำงานแบบ TLM-Z สำหรับโปรแกรม BT.C, CG.C, LU.C, MG.C และ SP.C

(2) การอพยพแบบคงสถานะการทำงานแบบ TLM-XZ มีประสิทธิภาพด้อยกว่าการอพยพแบบคงสถานะการทำงานแบบ TLM-Z สำหรับโปรแกรม IS.C 26% และ UA.C 8%

(3) การอพยพแบบคงสถานะการทำงานแบบ TLM-XZ มีประสิทธิภาพดีกว่าการอพยพแบบคงสถานะการทำงานแบบ TLM-Z บนเครือข่ายแบนด์วิดท์ 100 Mbps

(4) การอพยพแบบคงสถานะการทำงานแบบ TLM-Z และแบบ TLM-XZ มีประสิทธิภาพไม่ต่างกันมากนัก แสดงให้เห็นว่า การเข้ารหัสข้อมูลแบบเดลตาด้วยวิธี XOR ไม่ได้ช่วยให้ประสิทธิภาพการบีบอัดข้อมูลของ LZ4 เพิ่มขึ้น อีกทั้งยังทำให้ประสิทธิภาพการบีบอัดข้อมูลของ LZ4 ลดลงสำหรับโปรแกรม IS.C และ UA.C

(5) การอพยพแบบคงสถานะการทำงานแบบ TLM-BZ ผ่านเครือข่าย 1 Gbps ด้วยวิธีบีบอัด Snappy มีประสิทธิภาพดีกว่า TLM สำหรับโปรแกรม BT.C 15%, CG.C 68%, MG.C 14% และ SP.C 16% และดีกว่า TLM-Z สำหรับโปรแกรม BT.C, CG.C, MG.C และ SP.C ดีขึ้น 7%, 19%, 13% และ 8% ตามลำดับ และการอพยพแบบคงสถานะการทำงานแบบ TLM-BZ ผ่านเครือข่ายแบนด์วิดท์ 100 Mbps ด้วยวิธีบีบอัด LZ4 มีประสิทธิภาพดีกว่า TLM และ TLM-Z สำหรับทุกโปรแกรมยกเว้น UA.C

(6) ผลการวิจัยยังพบว่า ลักษณะของข้อมูลและขนาดของเครือข่ายแบนด์วิดท์เป็นปัจจัยที่ส่งผลต่อประสิทธิภาพในการบีบอัดข้อมูลอย่างมาก เห็นได้ชัดจาก การอพยพแบบคงสถานะการ

ทำงานแบบ TLM-BZ สำหรับโปรแกรม IS.C เมื่อเครือข่ายมีแบนด์วิดท์ลดลงจาก 1 Gbps เหลือ 100 Mbps ประสิทธิภาพของการบีบอัดดีขึ้น ส่งผลให้ TLM-BZ ดีกว่า TLM และ TLM-Z แต่การบีบอัดกลับไม่แสดงผลใดๆ ต่อโปรแกรม UA.C อีกทั้งประสิทธิภาพการบีบอัดของแต่ละโปรแกรมก็ต่างกัน

1.2 วัตถุประสงค์ของการวิจัย

1.2.1 เพื่อพัฒนาประสิทธิภาพของการอพยพแบบคงสถานะการทำงาน แบบ TLM และ TLM-Z

1.2.2 เพื่อวิเคราะห์หาค่าของพารามิเตอร์ที่ดีที่สุดที่สามารถเพิ่มประสิทธิภาพการอพยพแบบคงสถานะการทำงานแบบ TLM และ TLM-Z บนระบบเครือข่าย 1 Gbps และ 100 Mbps

1.2.3 เพื่อเปรียบเทียบประสิทธิภาพการอพยพแบบคงสถานะการทำงานแบบ TLM, TLM-Z, TLM-XZ และ TLM-BZ

1.2.4 เพื่อศึกษาและพัฒนาการอพยพแบบคงสถานะการทำงานด้วยการบีบอัดข้อมูล

1.2.5 เพื่อวิเคราะห์หาเทคนิคการบีบอัดข้อมูลที่เหมาะสมกับลักษณะของหน่วยความจำของโปรแกรม NPB

1.2.6 เพื่อหาค่าพารามิเตอร์และวิธีการบีบอัดที่ดีที่สุดของ Blosc ที่เหมาะสมกับลักษณะของหน่วยความจำของโปรแกรม NPB

1.3 สมมุติฐานในการวิจัย

1.3.1 การเข้ารหัสแบบเดลตาด้วยวิธี XOR ช่วยเพิ่มประสิทธิภาพการบีบอัดข้อมูลของ Z-Server

1.3.2 การเข้ารหัสแบบเดลตาด้วยวิธี XOR ทำให้เวลาหยุดการทำงานและเวลารวมสำหรับการอพยพของ TLM-XZ น้อยกว่า TLM และ TLM-Z

1.3.3 การเข้ารหัสแบบเดลตาด้วยวิธี XOR ทำให้ประสิทธิภาพของการอพยพแบบคงสถานะการทำงานบนระบบเครือข่าย 100 Mbps ดีกว่า 1 Gbps

1.3.4 การเพิ่มการบีบอัดข้อมูลด้วย Blosc ทำให้เวลาหยุดการทำงานและเวลารวมสำหรับการอพยพของ TLM-BZ น้อยกว่า TLM และ TLM-Z

1.3.5 การเพิ่มการบีบอัดข้อมูลด้วย Blosc ทำให้ประสิทธิภาพของการอพยพแบบคงสถานะการทำงานบนระบบเครือข่าย 100 Mbps ดีกว่า 1 Gbps

1.4 ขอบเขตของการวิจัย

1.4.1 ศึกษาข้อมูลหน่วยความจำที่ถูกถ่ายโอนระหว่างการอพยพแบบคงสถานะการทำงานแบบเทรดเพื่อพัฒนาและหาแนวทางในการพัฒนาระบบ TLM และ TLM-Z ให้มีประสิทธิภาพมากขึ้น

14.2 ศึกษาเพื่อปรับปรุงและพัฒนาเทคนิคในการเพิ่มประสิทธิภาพด้วยการบีบอัดข้อมูล

1.4.3 ศึกษาเพื่อปรับปรุงและพัฒนากลไกการทำงานของ Memory Server

1.4.4 ศึกษาและเปรียบเทียบเพื่อหาความสัมพันธ์ของ Downtime และ Migration Time ระหว่าง TLM, TLM-Z และ TLM-XZ

1.4.5 ศึกษาและเปรียบเทียบเพื่อหาความสัมพันธ์ของ Downtime และ Migration Time ระหว่าง TLM, TLM-Z และ TLM-BZ

1.5 ข้อจำกัดของการวิจัย

งานวิจัยนี้ใช้ซอฟต์แวร์ของ KVM ติดตั้งบนระบบปฏิบัติการลินุกซ์ Ubuntu 12.04 LTS มี Memory Server ทำงานเสมือนเป็น Super Cache และ ใช้ netcat ในการทำหน้าที่เป็น Server และ Client ให้กับ Z-Server ซึ่งใช้โปรแกรม LZ4 ในการบีบอัดหน่วยความจำ วัดประสิทธิภาพการอพยพแบบคงสถานะการทำงานโดยใช้ชุดโปรแกรมทาง Scientific ด้วยเบนมาร์ก NPB 3.3.1 จำนวน 7 โปรแกรม ทำการทดลองบนเครือข่ายที่มีความเร็ว 1 Gbps และ 100 Mbps โดยใช้ tool ในการกำหนดแบนด์วิดท์ทาง Lan และถือว่าดิสก์อิมเมจของคอมพิวเตอร์เสมือนต้นทางและปลายทางเป็นชุดเดียวกันโดยการแชร์ผ่านระบบแชร์ไฟล์ผ่านเครือข่าย (Network File System)

1.6 ประโยชน์ที่คาดว่าจะได้รับ

1.6.1 สามารถเพิ่มประสิทธิภาพของการอพยพแบบคงสถานะการทำงานของคอมพิวเตอร์เสมือนที่ใช้หน่วยความจำและหน่วยประมวลผลกลางมาก

16.2 สามารถลดเวลาในการหยุดการทำงานและเวลารวมสำหรับการอพยพของการอพยพแบบคงสถานะการทำงานแบบ TLM-Z บนเครือข่ายที่มีแบนด์วิดท์ 1 Gbps และ 100 Mbps

1.6.3 พบแนวทางในการเพิ่มประสิทธิภาพที่เหมาะสมกับลักษณะหน่วยความจำของแต่โปรแกรม

1.6.4 สามารถนำ Memory Server และ Z-Server ไปประยุกต์ใช้การถ่ายโอนหน่วยความจำผ่านเครือข่ายอื่นๆ ได้

1.7 คำนิยามศัพท์

เทคโนโลยีระบบคอมพิวเตอร์เสมือน (Virtualization Technology) คือ เทคโนโลยีสำหรับการจำลองสภาพแวดล้อมให้เสมือนมีคอมพิวเตอร์หลายเครื่องทำงานอยู่ภายในคอมพิวเตอร์เครื่องจริง โดยอาศัยการทำงานของซอฟต์แวร์ด้านเวอร์ชวลไลเซชั่นเป็นตัวจัดการในเรื่องต่างๆ ไม่ว่าจะเป็นเรื่องของการใช้ฮาร์ดแวร์ ระบบปฏิบัติการ ระบบไฟล์ ระบบเครือข่าย และไฟล์วอลล์ ให้กับระบบเสมือนแต่ละตัว ทำให้การเชื่อมต่อระบบจากภายนอกไม่สามารถแยกได้ว่ากำลังติดต่อกับระบบเสมือนหรือระบบจริง (Icesolution, n.d.)

คอมพิวเตอร์เสมือน (Virtual Machine) คือ การจำลององค์ประกอบต่างๆ เช่น หน่วยความจำ ฮาร์ดดิสก์ และอุปกรณ์เน็ตเวิร์กขึ้นมาเหมือนคอมพิวเตอร์เครื่องหนึ่งและมีระบบปฏิบัติการและทำงานอยู่บนเครื่องคอมพิวเตอร์จริง

อพยพแบบคงสถานะการทำงาน (Live Migration) คือ การอพยพคอมพิวเตอร์เสมือนต้นทางซึ่งกำลังทำงานอยู่ไปยังคอมพิวเตอร์เสมือนปลายทาง โดยที่ไม่จำเป็นต้องหยุดการทำงานหรือใช้เวลาในการหยุดน้อยที่สุด ในงานวิจัยนี้อพยพคอมพิวเตอร์เสมือนด้วยการถ่ายโอนข้อมูลหน่วยความจำเพจและสถานะของคอมพิวเตอร์เสมือน และแชร์ที่เก็บข้อมูลเสมือน (Virtual Hard Disk) ผ่านระบบแชร์ไฟล์ผ่านเครือข่าย (Network File System)

แบนด์วิดท์ (Bandwidth) คือ ช่องสัญญาณที่ใช้ในการส่งผ่านข้อมูล โดยในงานวิจัยนี้จะทำการอพยพแบบคงสถานะการทำงานผ่านเครือข่ายที่มีแบนด์วิดท์ 1 Gbit/s และ 100 Mbit/s

ไฮเปอร์ไวเซอร์ (Hypervisor) คือ ซอฟต์แวร์ในการจำลองระบบคอมพิวเตอร์เสมือน ซึ่งมีหน้าที่จัดการกับระบบปฏิบัติการของคอมพิวเตอร์เสมือน รวมทั้งควบคุมการเข้าถึงและจัดการทรัพยากรฮาร์ดแวร์

KVM (Kernel-based Virtual Machine) คือ ไฮเปอร์ไวเซอร์ที่ทำงานบนระบบปฏิบัติการลินุกซ์และหน่วยประมวลผลที่มีสถาปัตยกรรมฮาร์ดแวร์ X86 ที่ได้รับการเพิ่มความสามารถในการสนับสนุนการจำลองระบบคอมพิวเตอร์เสมือน

TLM (Time-bound thread-based Live Migration) คือ การอพยพแบบคงสถานะการทำงานของคอมพิวเตอร์เสมือนแบบเทรดที่พัฒนาขึ้นบน KVM

Z-Server คือ เครื่องแม่ข่ายในการบีบอัดข้อมูล ประกอบด้วย Compression Server และ Decompression Server ซึ่งอาศัย netcat ในการเชื่อมต่อเครือข่ายแบบ TCP และใช้ LZ4 ในการเข้ารหัสและถอดรหัสข้อมูลหน่วยความจำ

TLM-Z คือ การอพยพแบบคงสถานะการทำงานของคอมพิวเตอร์เสมือนแบบเทรดโดยมีการเพิ่มเครื่องแม่ข่าย Z-Server ในการบีบอัดด้วยวิธี LZ4

TLM-XZ คือ การอพยพแบบคงสถานะการทำงานของคอมพิวเตอร์เสมือนแบบเทรดโดยมีการเข้ารหัสแบบเดลตาด้วยวิธี XOR ก่อนทำการบีบอัดด้วยเครื่องแม่ข่าย Z-Server

Blosc (A Blocking, shuffling and lossless compression library) เป็นไลบรารีการบีบอัดข้อมูลที่อาศัยเทรดและใช้เทคนิคการแบ่งข้อมูลออกเป็นบล็อก (blocking technique) ด้วยวิธี Divide and Conquer เพื่อให้ข้อมูลมีขนาดเล็กเพียงพอที่จะใส่ในแคชได้ และอาศัยคำสั่งการเรียงลำดับข้อมูล (shuffling) ซึ่งเป็นหนึ่งในชุดคำสั่งของหน่วยประมวลผลกลาง เพื่อทำให้การบีบอัดข้อมูลมีอัตราการบีบอัดที่ดีขึ้นและเร็วขึ้น โดยใช้วิธีบีบอัดข้อมูลแบบไม่มีการสูญหาย (lossless data compression) ที่มีความเร็วในการบีบอัดสูงได้แก่ bloscLZ, LZ4, LZ4hc, Snappy หรือ Zlib

TLM-BZ คือ การอพยพแบบคงสถานะการทำงานของคอมพิวเตอร์เสมือนแบบเทรดโดยมีการเพิ่ม Blosc ในการบีบอัดข้อมูลระดับหน่วยความจำเพจโดยใช้เทรดและเทคนิคการเรียงลำดับข้อมูลเพื่อช่วยให้การบีบอัดมีประสิทธิภาพและเร็วขึ้น โดยมีวิธีบีบอัดให้เลือกคือ bloscLZ, LZ4, LZ4hc, Snappy และ Zlib ก่อนทำการบีบอัดด้วยเครื่องแม่ข่าย Z-Server

เวลารวมสำหรับการอพยพ (Total Migration Time) คือ เวลาทั้งหมดที่ใช้ในการอพยพคอมพิวเตอร์เสมือนตั้งแต่การใช้คำสั่งให้มีการอพยพจนกระทั่งคอมพิวเตอร์เสมือนปลายทางเริ่มการทำงาน

เวลาหยุดการทำงาน (Downtime) คือ เวลาที่คอมพิวเตอร์เสมือนหยุดทำงานเพื่อถ่ายโอนหน่วยความจำที่เหลืออยู่ไปยังคอมพิวเตอร์เสมือนปลายทาง

CPU over-committed คือการ map ซีพียูคอร์ของคอมพิวเตอร์เสมือนไปที่ซีพียูของคอมพิวเตอร์จริง โดย map ซีพียูคอร์ของคอมพิวเตอร์จริง 1 คอร์กับ 4 หรือ 8 ซีพียูคอร์ของคอมพิวเตอร์เสมือน ซึ่งโดยปกติ ถ้าคอมพิวเตอร์เสมือนมี 8 คอร์ hypervisor จะ map 8 คอร์ของคอมพิวเตอร์เสมือนกับ 8 ซีพียูคอร์ของคอมพิวเตอร์จริง การทำ cpu over-committed จะทำให้คอมพิวเตอร์เสมือนทำงานช้าลงและผลิต dirty pages น้อยลง

บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

ในงานวิจัยเกี่ยวกับการอพยพแบบคงสถานะการทำงานของคอมพิวเตอร์เสมือนนี้ จำเป็นต้องมีความรู้พื้นฐานเกี่ยวกับเทคนิคการบีบอัดข้อมูล เทคโนโลยีการจำลองระบบคอมพิวเตอร์เสมือน และจำเป็นต้องอาศัยแนวคิดจากงานวิจัยต่างๆ ที่เกี่ยวข้อง

2.1 ทฤษฎีที่เกี่ยวข้อง

2.1.1 Compression (การบีบอัดข้อมูล)

การเข้ารหัสข้อมูลแบบเดลตา (Delta Compression) เป็นกระบวนการแปลงข้อมูลเพื่อให้ได้ข้อมูลเฉพาะส่วนที่แตกต่าง โดยอาศัยความแตกต่างระหว่างข้อมูลในอดีตและปัจจุบัน การเข้ารหัสด้วยเดลตาช่วยลดความซ้ำซ้อนของข้อมูลลง ดังนั้น การเก็บข้อมูลเฉพาะส่วนที่แตกต่างกันนั้นจะทำให้สามารถจัดเก็บข้อมูลได้อย่างมีประสิทธิภาพมากขึ้น

การบีบอัดข้อมูล (Data Compression) เป็นกระบวนการเข้ารหัสข้อมูลเพื่อให้ใช้จำนวนบิตในการเก็บข้อมูลน้อยลงกว่าเดิม (ชนิดา เรืองศิริวัฒนกุล, n.d.) การบีบอัดข้อมูลช่วยลดปริมาณการใช้ทรัพยากร เช่น ใช้แบนด์วิดท์ของระบบเครือข่ายน้อยลง เมื่อข้อมูลถูกบีบอัด (compress) แล้วก็ต้องมีการคลาย (decompress) หรือ ถอดรหัสเพื่อให้ได้ข้อมูลเดิมกลับมา

การบีบอัดข้อมูล มี 2 แบบ คือ

(1) การบีบอัดแบบสูญเสียข้อมูลไปบางส่วน (Lossy Compression) เป็นการบีบอัดโดยตัดข้อมูลต้นฉบับบางส่วนออกเพื่อลดขนาดไฟล์ โดยข้อมูลที่ซ้ำซ้อนจะถูกตัดทิ้งอย่างถาวร ทำให้ผลลัพธ์ที่ได้จากการบีบอัดไม่สมบูรณ์เหมือนกับข้อมูลต้นฉบับและคุณภาพของข้อมูลลดลงด้วย

(2) การบีบอัดแบบไม่สูญเสีย (Lossless Compression) เป็นการบีบอัดข้อมูลที่ไม่ทำให้ข้อมูลเกิดการสูญหายระหว่างการบีบอัด ผลลัพธ์ที่ได้จึงมีความสมบูรณ์เหมือนต้นฉบับ

ในงานวิจัยนี้ใช้วิธีบีบอัดข้อมูลแบบไม่สูญเสียเท่านั้น ซึ่งมีแนวคิดพื้นฐานจากวิธีบีบอัดดังต่อไปนี้

รันเลงธ์เอนโคดดิง (Run-length encoding - RLE) ใช้หลักการในการนับจำนวนข้อมูลที่อยู่ติดกันและซ้ำกัน แทนด้วยจำนวนที่นับได้และตามด้วยข้อมูลตัวที่มีการซ้ำกัน ข้อมูลที่มีค่าซ้ำกันนั้นจึงถูกจัดเก็บเสมือนเป็นรหัส ทำให้ขนาดของข้อมูลมีขนาดเล็กลง ตัวอย่างเช่น 00000000111111 เมื่อใช้ RLE แล้วจะได้ข้อมูลออกมาเป็น #8 0 #6 1

การเข้ารหัสฮัฟฟ์แมน (Huffman Coding) (ลัญฉกร วุฒิสิทธิกุลกิจ, 2549) เป็นวิธีบีบอัดที่อาศัยข้อมูลทางสถิติ (Probability) ในการเกิดอักขระหรือสัญลักษณ์แต่ละตัวเป็นพื้นฐานสำคัญ การเข้ารหัสฮัฟฟ์แมนมีขั้นตอนการทำงานดังนี้

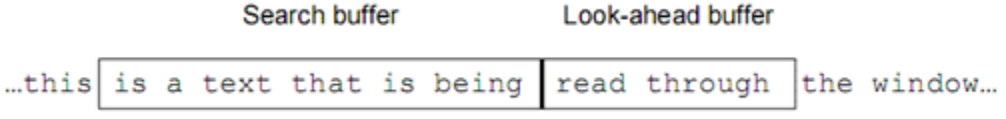
ขั้นตอนที่ 1 ให้เรียงลำดับของสัญลักษณ์ทั้งหมดในแนวตั้ง โดยให้สัญลักษณ์ที่มีความน่าจะเป็นสูงสุดอยู่บนสุด

ขั้นตอนที่ 2 พิจารณาเฉพาะสัญลักษณ์สองตัวล่างสุด กำหนดบิต 0 ให้กับสัญลักษณ์ตัวบน และกำหนดบิต 1 ให้กับสัญลักษณ์ตัวล่าง จากนั้นให้ผนวกสัญลักษณ์ทั้งสองตัวและแทนด้วยสัญลักษณ์ใหม่ที่มีค่าความน่าจะเป็นในการเกิดเท่ากับผลรวมของความน่าจะเป็นทั้งสองสัญลักษณ์บวกกัน

ขั้นตอนที่ 3 หลักจากผ่านขั้นตอนที่ 2 จะมีจำนวนสัญลักษณ์รวมลดลงหนึ่งตัวเสมอ จากนั้นให้กลับไปทำกระบวนการในขั้นตอนที่ 1 ใหม่ และให้กระทำเช่นนี้เรื่อยไปจนกว่าจะเหลือสัญลักษณ์เพียง 2 ตัวสุดท้าย ถึงจะสิ้นสุดกระบวนการเข้ารหัส

Lempel-ZW Abraham Lempel, Jacob Ziv, and Terry Welch เป็นหลักการคิดที่อาศัย Hash Table ในการเก็บรูปแบบของข้อมูลที่เคยส่งไปแล้ว เมื่อรับข้อมูลเข้ามาใหม่ จะทำการตรวจเช็คกับ Hash Table และทำการเก็บรูปแบบที่ยังไม่มีไว้ในตาราง หากว่ามีรูปแบบนั้นอยู่ในตารางแล้ว จะทำการส่งข้อมูลที่เป็นรหัสออกไปแทน ซึ่งเป็นหลักการคิดที่ถูกนำไปต่อยอดเทคนิคการบีบอัดแบบอื่น หลักการทำงานของเทคนิคนี้ใช้การอ้างถึงรูปแบบซึ่งไม่ใช่ได้มาจากไฟล์ที่ถูกบีบอัดเท่านั้น ยังคงอ้างอิงถึงไฟล์ต้นฉบับเช่นกัน ทำให้ถึงจะมีความเหมือนกันแค่นิดเดียวระหว่างไฟล์ใหม่กับไฟล์เก่า แต่จะยังคงได้อัตราการบีบอัดข้อมูลที่ดี

LZ77 เป็นวิธีบีบอัดที่มีการเก็บข้อมูลในอดีตที่ผ่านมาไม่นานเพื่อนำไปใช้เป็นข้อมูลอ้างอิงสำหรับการเปรียบเทียบข้อมูลที่กำลังเข้ารหัส คำรหัสที่ได้จากการเข้ารหัสเป็นตัวชี้ตำแหน่งของข้อมูลในอดีตที่อยู่ก่อนหน้า การเข้ารหัส LZ77 ใช้ Sliding windows (ภาพที่ 2.1) ซึ่งเป็นหน้าต่างเสมือนที่เลื่อนไปตามแถวของข้อมูล ที่เวลาใดเวลาหนึ่งจะเห็นเฉพาะข้อมูลที่อยู่ตรงหน้าต่าง ซึ่งประกอบด้วย บัฟเฟอร์ค้นหา (search buffer) และบัฟเฟอร์มองไปข้างหน้า (look ahead buffer)



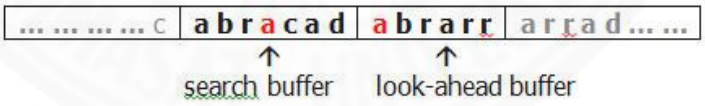
ภาพที่ 2.1 ตัวอย่างหน้าตาของ sliding window
(การบีบอัดข้อมูลตามขั้นตอนวิธีของ Jacob Ziv และ Abraham Lempel (LZ77), n.d.)

บัฟเฟอร์ค้นหา ทำหน้าที่เป็น dictionary ส่วนบัฟเฟอร์มองไปข้างหน้า ทำหน้าที่เป็น keyword สำหรับค้นใน dictionary บัฟเฟอร์ทั้งสองเป็นส่วนหนึ่งของ sliding window จึงเลื่อนไปพร้อมกัน และขนาดของบัฟเฟอร์ทั้งสองนี้เป็นพารามิเตอร์ที่สำคัญในการประยุกต์ใช้งาน โดยมีสมมุติฐานว่า รูปแบบของข้อมูลที่ซ้ำกัน (pattern) เกิดขึ้นในพิสัยของบัฟเฟอร์ทั้งสองนี้ ผลของการเข้ารหัสจะแสดงในรูปแบบ <offset, length, character> โดย Offset คือ ระยะห่างนับจากตำแหน่งที่พบสัญลักษณ์ที่ตรงกันครั้งแรก Length คือ จำนวนของสัญลักษณ์ที่ต่อเนื่องกันในบัฟเฟอร์ที่เหมือนกัน และ Character คือ สัญลักษณ์ที่อยู่ในบัฟเฟอร์มองไปข้างหน้าซึ่งอยู่ ณ ตำแหน่งถัดมาจากกลุ่มสัญลักษณ์ที่ตรงกัน

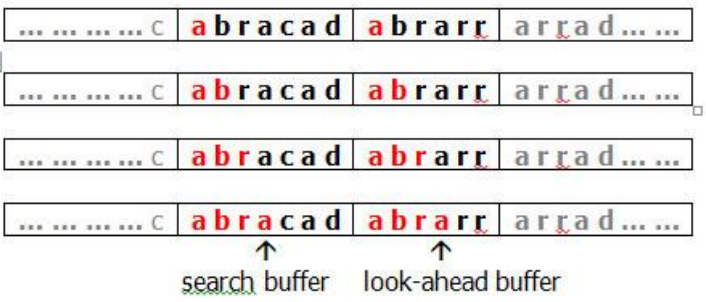
ตัวอย่างหลักการของ LZ77 (การบีบอัดข้อมูลตามขั้นตอนวิธีของ Jacob Ziv และ Abraham Lempel (LZ77), n.d.)

ครั้งแรกพบสายอักขระที่ตรงกันที่ offset = 2, length = 1

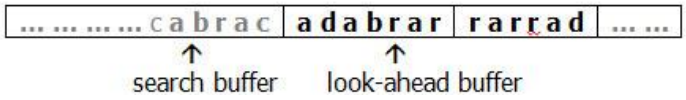
ครั้งที่สอง พบสายอักขระที่ตรงกันที่ offset = 4, length = 1, ความยาวเท่าครั้งแรก



ครั้งที่สาม พบสายอักขระที่ตรงกันที่ offset = 7, length = 4, ความยาวสูงกว่าจึงเปลี่ยนแทนคู่ลำดับ (offset, length) ที่เก็บไว้เดิมด้วยค่าใหม่



ตัวเข้ารหัสส่งข้อมูลออกเป็น (7, 4, r) จากนั้นจึงเลื่อน sliding window ไปทางขวา length + 1 = 5



2.1.1.1 การบีบอัดข้อมูลของ LZ4

LZ4 (Collet, 2015) เป็นอัลกอริทึมที่อยู่ในตระกูล LZ77 ใช้ในการบีบอัดข้อมูลซึ่งมุ่งเน้นที่ความเร็วของการบีบอัดข้อมูลและถอดรหัสข้อมูล ค้นหาค่าที่ซ้ำซ้อนกัน โดยอาศัย Hashtable และไม่ค้นหาความเป็นไปได้ทั้งหมดของค่าที่ซ้ำซ้อนกัน และทำการบีบอัดกับข้อมูลในระดับไบต์

LZ4 มีความเร็วในการบีบอัดข้อมูลอยู่ที่ 400 MB/s ต่อคอร์ เหมาะกับการใช้งานของซีพียูที่มีหลายคอร์ และสามารถแปลงข้อมูลกลับคืนมาด้วยความเร็วหลาย GB/s ต่อคอร์ ซึ่งโดยปกติแล้วความเร็วในการแปลงข้อมูลกลับนั้นสามารถเร็วได้ถึงความเร็วของแรมซึ่งถูกจำกัดบนระบบการทำงานหลายคอร์

ตารางที่ 2.1 ผลการวัดประสิทธิภาพของ LZ4 เมื่อเทียบกับเครื่องมือในการบีบอัดอื่นๆ (“Lz4 - Extremely Fast Compression,” n.d.)

Compressor	Ratio	Compression	Decompression
memcpy	1.000	4200 MB/s	4200 MB/s
LZ4 fast 17 (r129)	1.607	**690 MB/s**	**2220 MB/s**
LZ4 default (r129)	**2.101**	**385 MB/s**	**1850 MB/s**
LZO 2.06	2.108	350 MB/s	510 MB/s
QuickLZ 1.5.1.b6	2.238	320 MB/s	380 MB/s
Snappy 1.1.0	2.091	250 MB/s	960 MB/s
LZF v3.6	2.073	175 MB/s	500 MB/s
zlib 1.2.8 -1	2.730	59 MB/s	250 MB/s
LZ4 HC (r129)	**2.720**	22 MB/s	**1830 MB/s**
zlib 1.2.8 -6	3.099	18 MB/s	270 MB/s

LZ4 Sequence

Token : ==> 4-high-bits : literal length / 4-low-bits : match length

Token	*Literal length+ (optional)*	Literals	Offset	*Match length+ (optional)*
1-byte	0-n bytes	0-L bytes	2-bytes (little endian)	0-n bytes

ภาพที่ 2.2 ส่วนประกอบของบล็อกการบีบอัดข้อมูล (Compression block) ของ LZ4 (“RealTime Data Compression - LZ4 explained,” n.d.)

บล็อกที่ถูกบีบอัด (Compression block) ของ LZ4 นั้นประกอบด้วย sequence หลายๆ sequence เรียงต่อกัน ในแต่ละ sequence ดังแสดงในภาพที่ 2.2 นั้น คือชุดของ literals ซึ่งเป็นส่วนที่ไม่ถูกบีบอัดหรือข้อมูลที่ไม่ซ้ำ ตามด้วยส่วนที่ใช้ในการก็อปปี้ match (ส่วนที่ระบุถึงการซ้ำของข้อมูล)

ในแต่ละ sequence จะเริ่มต้นด้วย token ซึ่งมีขนาด 1 ไบต์ ซึ่งใน 1 ไบต์นี้จะแยกออกมาเป็น 2 ฟิลด์ ซึ่งแต่ละฟิลด์มีขนาด 4 บิต และมีค่าอยู่ระหว่าง 0-15

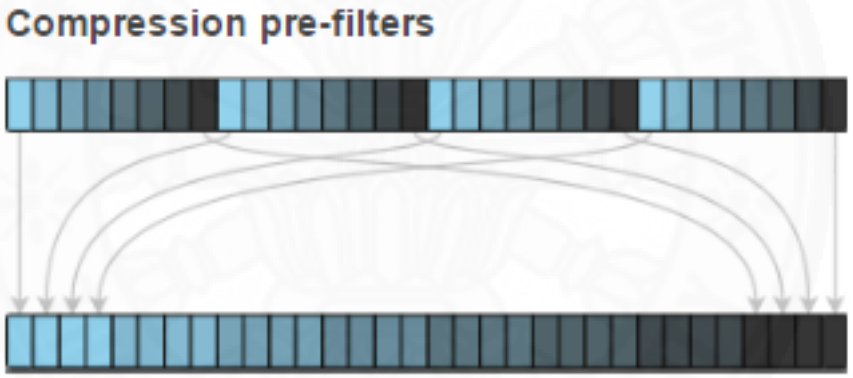
ค่าใน token นั้นจะใช้สำหรับระบุความยาวและส่วนประกอบต่างๆ ใน sequence นั้น โดย 4 บิตแรก (high-bits) นั้นจะบอกถึงความยาวของ literal และ 4 บิตหลัง (low-bits) นั้นจะบอกถึงความยาวของ match ซึ่งความยาวของ literal และ match นั้น ถ้าหากว่าเกินกว่า 15 จะมีส่วน literal length เพิ่มขึ้นมา เพื่อเก็บค่าความยาวของ literal โดยแต่ละไบต์จะเก็บค่าได้ 255 เมื่อ literal length มีความยาวมากขึ้น ก็สามารถเพิ่มขนาดของ literal length ได้ โดยไม่มีจำกัดขนาดไว้ (no size limit) จาก literals คือ operation ที่ใช้ในการก็อปปี้ match ซึ่งเริ่มจาก offset ซึ่งมี ขนาด 2 ไบต์ แบบ little endian (ไบต์แรกคือ low ไบต์ที่ 2 คือ high) ระบุถึงตำแหน่งของ match ที่ถูกก็อปปี้มาจาก literals ซึ่ง 1 หมายถึงตำแหน่งปัจจุบัน ลบ 1 byte และค่าสูงสุดของ offset คือ 65535

ในการคำนวณความยาวของ match จะใช้ฟิลด์ที่สองของ token ซึ่งเป็น 4 บิตหลัง (low-bits) โดยความยาวที่น้อยที่สุดของ match เท่ากับ 4 ไบต์ เรียกว่า minmatch ดังนั้น ค่า 0 จึงหมายถึง 4 ไบต์ และ 15 หมายถึง 19+ ไบต์ ซึ่งสามารถขยายให้มากขึ้นได้เช่นเดียวกันกับ literals

literal decoder สามารถสร้างข้อมูลต้นฉบับด้วยข้อมูลจาก offset และ ความยาวของ match

2.1.1.2 Blosc (A Blocking, shuffling and lossless compression library)

Blosc เป็นเครื่องมือสำหรับบีบอัดข้อมูลที่มีประสิทธิภาพสูงสำหรับข้อมูลแบบไบนารี ใช้ blocking technique ด้วย Divide and conquer เพื่อทำให้บล็อกของข้อมูลสามารถใส่ในแคชที่ L1 หรือ L2 ได้ รวมทั้งมีการใช้ shuffle filter (ภาพที่ 2.3) จากชุดคำสั่ง SSE2 (Streaming SIMD Extensions 2) ซึ่งเป็นชุดคำสั่ง SIMD (Single Instruction Multiple Data) บนหน่วยประมวลผลกลางที่มีสถาปัตยกรรมฮาร์ดแวร์แบบ X86 อยู่ในหน่วยประมวลผลสมัยใหม่ของ Intel และ AMD ในการเรียงลำดับข้อมูลไบต์หรือบิตใน Data Stream ก่อนการบีบอัด ทำให้สามารถบีบอัดข้อมูลได้เร็วและมีอัตราการบีบอัดที่ดีขึ้น ดังนั้น Blosc จึงเป็นเครื่องมือบีบอัดที่มีประสิทธิภาพสูง สามารถถ่ายโอนข้อมูลไปยังแคชได้เร็วกว่าวิธีดั้งเดิมผ่านการเรียกใช้งานของระบบปฏิบัติการโดยการใช้ฟังก์ชัน memcpy()



ภาพที่ 2.3 การใช้ฟิลเตอร์ในการเรียงลำดับข้อมูลก่อนการบีบอัด (Alted, 2016).

Blosc เป็นวิธีบีบอัดที่มุ่งเน้นความเร็วในการบีบอัด โดยค้นหาการเกิดซ้ำของข้อมูลให้เร็วที่สุดเท่าที่จะเป็นได้ โดยอาศัยเทรดเข้ามาช่วย และมีภาระงานเพียงเล็กน้อยสำหรับข้อมูลที่ไม่สามารถบีบอัดได้ ซึ่งมีภาระงานสูงสุดที่เพิ่มขึ้นเพียง (16 + 4 * จำนวนเทรด) ไบต์

Blosc มาพร้อมกับเครื่องมือบีบอัดข้อมูล ดังต่อไปนี้ BloscLZ (ใช้หลักการบีบอัด FastLZ), LZ4, LZ4HC, Snappy, และ Zlib

fastLZ (Hidayat, n.d.) เป็นวิธีการบีบอัดข้อมูลซึ่งเน้นความเร็ว เหมาะกับการบีบอัดและถอดรหัสข้อมูลแบบเรียลไทม์ มีภาระงาน (overhead) ต่ำมาก และเป็น thread-safe ซึ่งปลอดภัยและสามารถนำมาใช้กับประมวลผลแบบใช้เทรด

Snappy (Google, n.d.) คิดค้นขึ้นโดย Google โดยอาศัยแนวคิดของ LZ77 เป็นวิธีการบีบอัดข้อมูลที่เน้นความความเร็วในการบีบอัดและได้อัตราการบีบอัดที่สมเหตุสมผล

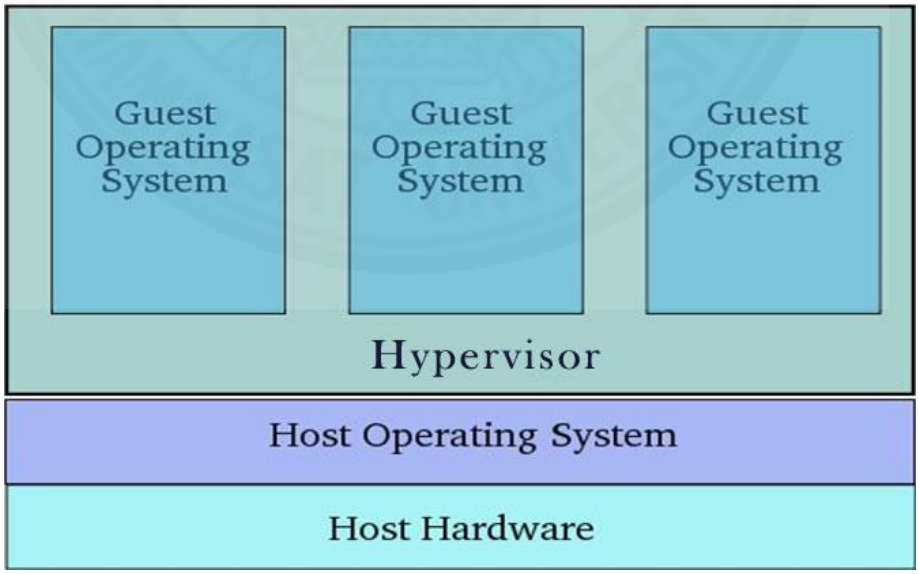
เช่น เมื่อเปรียบเทียบขนาดไฟล์ที่ได้จากการบีบอัดเทียบกับ Zlib จะได้ขนาดไฟล์ที่ใหญ่กว่า ประมาณ 20% ถึง 100% สามารถบีบอัดได้ที่ความเร็ว 250 MB/Sec ซึ่ง Snappy ถูกใช้งานมากมายในผลิตภัณฑ์ของ Google

LZ4HC (LZ4 Hash Chain หรือ LZ4 High Compression) เป็นวิธีการบีบอัดเดียวกันกับ LZ4 โดยค้นหารูปแบบซ้ำหลายรอบเพื่อให้ได้อัตราการบีบอัดที่สูง ส่งผลให้ใช้เวลาในการบีบอัดมากขึ้น

Zlib ใช้หลักการ Deflate ซึ่งเกิดจากการนำข้อดีของอัลกอริทึม Huffman และ LZ77 มารวมกัน เป็นการบีบอัดข้อมูลที่เน้นอัตราการบีบอัด และสามารถใช้ข้อมูลทางสถิติหรือความน่าจะเป็นช่วยให้สามารถบีบอัดได้อย่างมีประสิทธิภาพมากขึ้น

2.1.2 เทคโนโลยีระบบคอมพิวเตอร์เสมือน (Virtualization Technology)

เทคโนโลยีระบบคอมพิวเตอร์เสมือนนั้นเป็นเทคโนโลยีที่ใช้ซอฟต์แวร์มาทำการสร้างสภาพแวดล้อมจำลองของคอมพิวเตอร์เสมือน (Virtual Machine) ขึ้นมา ทำให้คอมพิวเตอร์จริงหนึ่งเครื่องนั้นสามารถมีคอมพิวเตอร์เสมือนหลายเครื่องทำงานโดยการแชร์ทรัพยากรของคอมพิวเตอร์จริง โดยในแต่ละคอมพิวเตอร์เสมือนนั้นมีทรัพยากรหน่วยความจำ, ฮาร์ดดิสก์, อุปกรณ์เน็ตเวิร์ก ระบบปฏิบัติการและซอฟต์แวร์เป็นของตนเองโดยอิสระต่อกัน อีกทั้งคอมพิวเตอร์เสมือนเหล่านั้นยังสามารถดำเนินการบนระบบปฏิบัติการที่แตกต่างกันได้

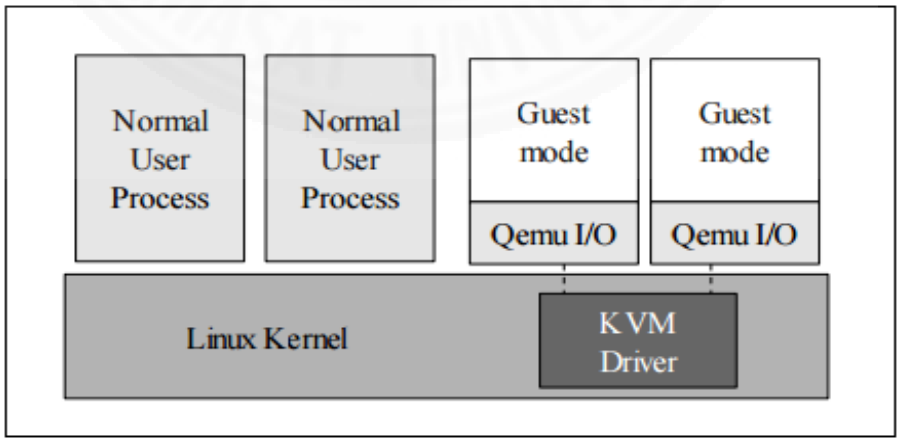


ภาพที่ 2.4 ภาพจำลองของเทคโนโลยีคอมพิวเตอร์เสมือน
(An Overview of Virtualization Techniques, n.d.)

ซิลเต็มส์เวอร์ชวลแมชชีน (System Virtual Machine) จัดสรรสภาพแวดล้อมที่สมบูรณ์ (Smith & Nair, 2005) ที่ทำให้ฮาร์ดแวร์แพลตฟอร์มของคอมพิวเตอร์เครื่องหนึ่ง เรียกว่า โฮส (Host) สามารถสนับสนุนคอมพิวเตอร์เสมือน เรียกว่า เกส (Guest) ให้ดำเนินการระบบปฏิบัติการ (OS) หลายระบบได้ในเวลาเดียวกัน และสามารถทำงานเป็นอิสระต่อกัน โดยอาศัยซอฟต์แวร์ในการจำลองระบบคอมพิวเตอร์ ซึ่งโดยทั่วไปจะเรียกว่า เวอร์ชวลแมชชีนมอนิเตอร์ (Virtual Machine Monitor) หรือ ไฮเปอร์ไวเซอร์ (Hypervisor) ทำหน้าที่ในจัดการกับระบบปฏิบัติการของเกส (Guest OS) รวมทั้งควบคุมการเข้าถึงและจัดการทรัพยากรฮาร์ดแวร์ทั้งหมด

ไฮเปอร์ไวเซอร์ถูกจำแนกเป็น 2 ประเภท (Popek & Goldberg, 1974) คือ ไฮเปอร์ไวเซอร์ที่สามารถดำเนินการได้โดยตรงบนฮาร์ดแวร์ของคอมพิวเตอร์โดยไม่จำเป็นต้องอาศัยระบบปฏิบัติการของโฮส ตัวอย่างไฮเปอร์ไวเซอร์ในกลุ่มนี้ได้แก่ Xen, Hyper-V และ VMware ส่วนอีกกลุ่มคือ ไฮเปอร์ไวเซอร์ที่ทำงานบนคอมพิวเตอร์ที่มีระบบปฏิบัติการหรือโฮสโอเอส (Host OS) ตัวอย่างเช่น VirtualBox และคีมู (QEMU)

อย่างไรก็ตาม การจำแนกไฮเปอร์ไวเซอร์ข้างต้นนั้น ยังคงไม่ชัดเจนนัก เนื่องจากเควีเอ็ม (KVM – Kernel-based Virtual Machine) เป็นระบบเคอเนลเบสเวอร์ชวลแมชชีนซึ่งเป็นซอฟต์แวร์ที่ถูกพัฒนามาจากคีมู และทำงานบนฮาร์ดแวร์ x86 ที่ได้รับการเพิ่มความสามารถในการสนับสนุนการทำเวอร์ชวลไลเซชันเข้าไปนั้น ได้ถูกนำไปรวมเป็นเคอเนลของลินุกซ์ ทำให้ระบบปฏิบัติการลินุกซ์ของโฮสทำงานเสมือนกับเป็นไฮเปอร์ไวเซอร์ ในขณะเดียวกัน KVM นั้นก็เป็นไฮเปอร์ไวเซอร์ที่ทำงานบนระบบปฏิบัติการของโฮส



ภาพที่ 2.5 สถาปัตยกรรมของเควีเอ็ม (Qumranet Inc., 2006)

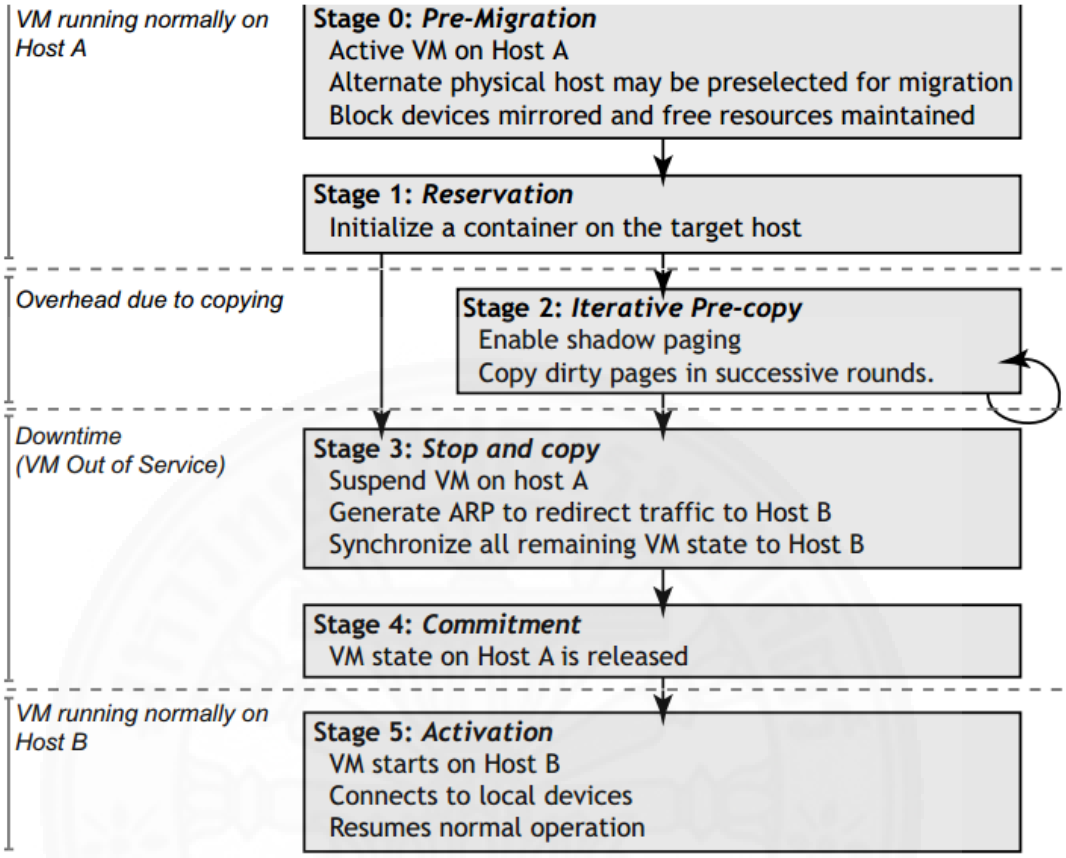
โดยปกติแล้วลินุกซ์จะทำงานในสองโหมด คือ โหมดเคอเนล (Kernel Mode) และโหมดผู้ใช้ (User Mode) (Qumranet Inc., 2006) เควีเอ็มเคอเนลโมดูลได้เพิ่มโหมดที่สาม โหมดเกส (Guest Mode ใน ภาพที่ 2.5) ซึ่ง ในโหมดเกสนี้ก็จะมีโหมดเคอเนลและโหมดผู้ใช้ของมันเองด้วยเช่นกัน ในโหมดเกสนี้ KVM จะสร้างหน่วยความจำที่แยกออกจากโปรเซสในระดับของผู้ใช้ขึ้นมาสำหรับสนับสนุนการทำงานของคอมพิวเตอร์เสมือน ซึ่งจะเป็นการทำงานในระดับการทำงานของโปรแกรมผู้ใช้ (user space program) คือเป็นโปรเซสหนึ่งของโฮสโอเอส และเนื่องจาก KVM นั้นเป็นไดร์เวอร์สำหรับการเพิ่มความสามารถให้กับซีพียูจริง (physical cpu) สำหรับสถาปัตยกรรมแบบ x86 ที่รองรับการทำระบบคอมพิวเตอร์เสมือน ทำให้คอมพิวเตอร์เสมือนทำงานได้อย่างมีประสิทธิภาพ

2.1.3 การอพยพแบบคงสถานะการทำงาน (Live Migration)

เทคโนโลยีระบบคอมพิวเตอร์เสมือนก่อให้เกิดการจัดสรรทรัพยากรอย่างคุ้มค่าและมีประสิทธิภาพ เพิ่มความยืดหยุ่นในการใช้ทรัพยากร หนึ่งในฟีเจอร์ที่ทำให้ระบบเสมือนจริงพร้อมใช้งานตลอดเวลา (High Availability) และ เพิ่มความทนทานต่อความบกพร่อง (Fault Resilience) ก็คือ การอพยพแบบคงสถานะการทำงาน (live migration) ซึ่งก็เป็นกระบวนการในการอพยพคอมพิวเตอร์เสมือนขณะที่กำลังทำงานอยู่บนเครื่องคอมพิวเตอร์จริงเครื่องหนึ่งไปทำงานต่อยังเครื่องคอมพิวเตอร์อีกเครื่องหนึ่งได้ โดยไม่ต้องหยุดการทำงานของคอมพิวเตอร์เสมือนหรือใช้เวลาในการหยุดน้อยที่สุด และยังสามารถทำการอพยพคอมพิวเตอร์เสมือนนี้จากซีกโลกหนึ่งไปยังอีกซีกโลกหนึ่งได้เช่นกัน

การอพยพแบบคงสถานะการทำงานนั้นมีอยู่ 2 แนวคิดหลัก คือ pre-copy และ post copy ซึ่ง pre-copy นั้นเป็นวิธีที่ได้รับความนิยมมากกว่า เนื่องจากประสบความสำเร็จในการอพยพ รวมทั้งมีประสิทธิภาพในการอพยพมากกว่า และส่งผลเสียน้อยกว่า

การอพยพแบบคงสถานะการทำงานแบบ pre-copy นั้นหลักๆ คือ stage 1 การเตรียมเครื่องคอมพิวเตอร์เสมือนต้นทางและปลายทาง stage 2 คือ การอพยพหน่วยความจำทั้งหมดไปก่อนแล้วจึงเข้าสู่ stage 3 คอมพิวเตอร์เสมือนต้นทางถูกสั่งให้หยุดทำงาน เพื่อถ่ายโอนข้อมูลหน่วยความจำที่เปลี่ยนแปลงที่เหลืออยู่จากการถ่ายโอนหน่วยความจำใน stage ที่ 2 และเมื่อถ่ายโอนหน่วยความจำทั้งหมดแล้ว จะเริ่มการทำงานที่คอมพิวเตอร์เสมือนปลายทาง แนวคิดดังกล่าวข้างต้นนั้น มีรากฐานมาจากแนวคิดของ Clark (Clark et al, 2005) ภาพที่ 2.6



ภาพที่ 2.6 แสดงการทำงานของการอพยพแบบคงสถานะการทำงานแบบ pre-copy (Clark et al, 2005)

การอพยพแบบคงสถานะการทำงานแบบ pre-copy ของ KVM มี 3 ขั้นตอน คือ

ขั้นตอนที่ 1 การเตรียมคอมพิวเตอร์เสมือนต้นทางและปลายทาง

ขั้นตอนที่ 2 คือ การอพยพหน่วยความจำ

ขั้นตอนที่ 3 KVM จะกำหนดเวลาหยุดคอมพิวเตอร์เสมือนไว้ก่อน และจะสั่งให้คอมพิวเตอร์เสมือนหยุดทำงานเมื่อข้อมูลหน่วยความจำที่เหลืออยู่นั้นสามารถถูกถ่ายโอนไปได้สำเร็จภายในเวลาหยุดคอมพิวเตอร์เสมือนที่ได้กำหนดไว้ เมื่อคอมพิวเตอร์เสมือนหยุดทำงาน คอมพิวเตอร์เสมือนต้นทางถ่ายโอนหน่วยความจำที่เหลืออยู่ทั้งหมดไปยังคอมพิวเตอร์เสมือนปลายทาง แล้วจึงเริ่มการทำงานที่คอมพิวเตอร์เสมือนปลายทาง

เมื่ออพยพคอมพิวเตอร์ที่มีการใช้หน่วยประมวลผลและหน่วยความจำมากทำให้ KVM ใช้เวลาในการอพยพยาวนาน เนื่องจากข้อมูลหน่วยความจำนั้นมีการเปลี่ยนแปลงมากและเร็ว ทำให้เงื่อนไขเวลาหยุดการทำงานที่กำหนดไว้นั้นยากที่จะเป็นไปได้ ดังตารางที่ 2.2 ซึ่งแสดงถึง

ประสิทธิภาพในการอพยพแบบคงสถานะการทำงานของคอมพิวเตอร์เสมือนที่รันโปรแกรมทางวิทยาศาสตร์ NPB 5 โปรแกรม คลาส C ซึ่งใช้หน่วยประมวลผลและหน่วยความจำมาก คือ IS.C, MG.C, LU.C, UA.C และ SP.C โดยกำหนดเวลาหยุดการทำงานไว้น้อยกว่า 2 วินาที

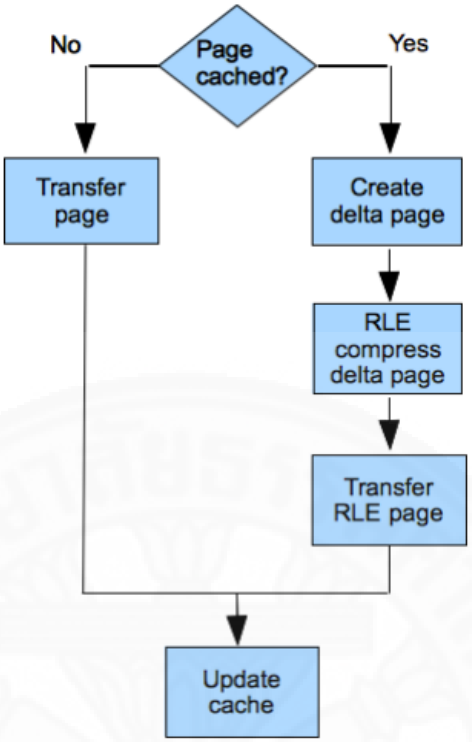
ตารางที่ 2.2 แสดงเวลารวมสำหรับการอพยพของ KVM ผ่านเครือข่ายแบนด์วิดท์ 1 Gbps (พิทักษ์ แท่นแก้ว, 2557)

BENCHMARK	MIGRATION TIME 1Gbps (ms)			
	KVM	KVM with Delta Compression		
		default cache size	1GB cache size	2GB cache size
is.C	93958	94425	71679	68941
mg.C	240983	242479	242856	243445
lu.C	551289	> 612454	550822	> 612099
ua.C	551489	> 612881	551145	> 612777
sp.C	612056	> 612442	> 612764	> 10 minutes

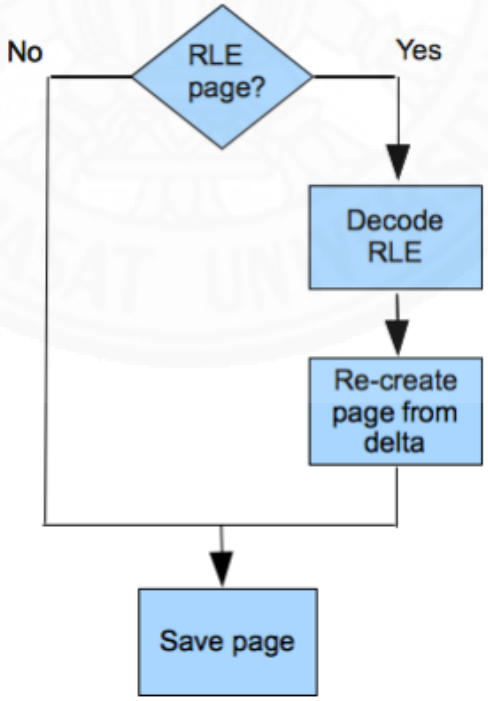
2.2 งานวิจัยที่เกี่ยวข้อง

อัลกอริทึมที่ใช้ในการอพยพแบบคงสถานะการทำงานที่มีอยู่นั้นไม่สามารถอพยพคอมพิวเตอร์เสมือนที่มีการใช้งานหน่วยประมวลผลและหน่วยความจำมาก หรืออพยพคอมพิวเตอร์เสมือนผ่านเครือข่ายที่มีแบนด์วิดท์ต่ำได้อย่างมีประสิทธิภาพ เนื่องจาก อัตราการเกิด dirty page สูงเกินกว่าจะสามารถอพยพแบบคงสถานะการทำงานได้สำเร็จโดยไม่กระทบการทำงานของคอมพิวเตอร์เสมือน ดังนั้น งานวิจัยเรื่อง Evaluation of Delta Compression Techniques of Efficient Live Migration of Large Virtual Machines (Svärd, Hudzia, Tordsson & Elmroth, 2011) จึงได้นำเสนอแนวคิดของการใช้ Delta Compression เพื่อเพิ่มอัตราการส่งข้อมูลและลด downtime เนื่องจากเทคนิคนี้จะทำให้ข้อมูลมีขนาดเล็กลง โดยงานวิจัยนี้ได้เลือกวิธีทำ delta compression ด้วย XOR binary Zero RLE (XBZRLE)

หลักการทำ delta compression ของเครื่องต้นทาง (ภาพที่ 2.7) คือ เมื่อเครื่องต้นทางรับเพจ จะทำการเช็คว่ามีเพจเวอร์ชันก่อนหน้าอยู่ในแคชหรือไหม ถ้าใช่ จะทำการสร้าง delta page ขึ้นมาจากการนำเพจนั้นไปทำ XOR กับเพจใหม่ หลังจากนั้นจะทำการบีบอัดข้อมูลด้วย binary Zero RLE และทำการตั้งค่า compression flag ที่หัวของเพจ แล้วจึงทำการอัพเดทแคชและส่งเพจที่ถูกบีบอัดไป



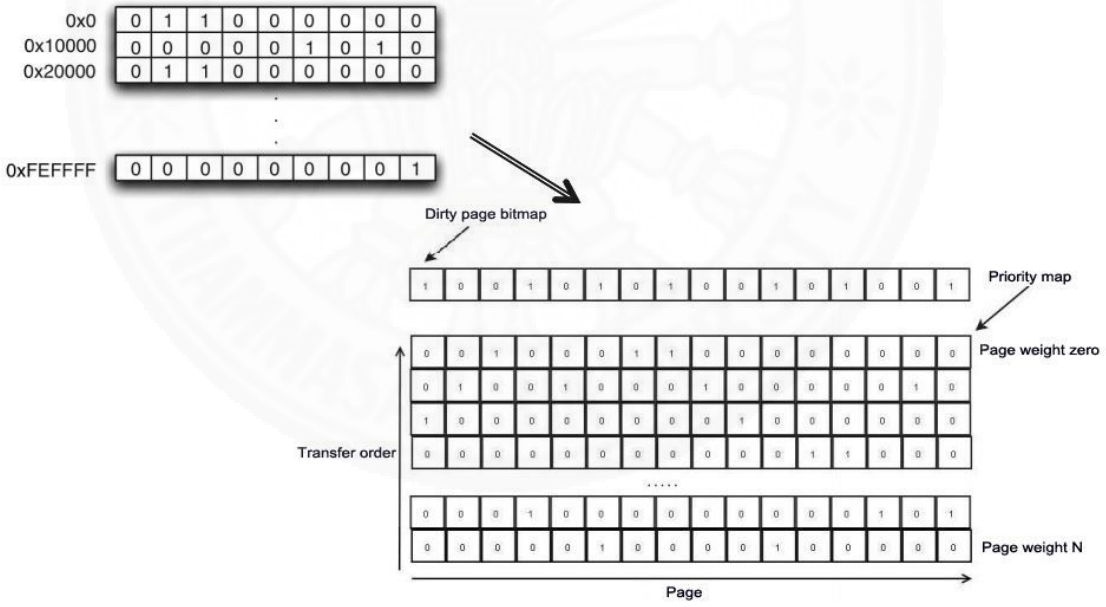
ภาพที่ 2.7 หลักการทำ delta compression ของคอมพิวเตอร์เสมือนต้นทาง



ภาพที่ 2.8 หลักการทำ delta compression ของคอมพิวเตอร์เสมือนปลายทาง

หลักการทำ delta compression ของคอมพิวเตอร์ปลายทาง (ภาพที่ 2.8) คือ คอมพิวเตอร์ปลายทางจะทำการเช็คว่า เพจนั้นได้มีการตั้งค่า compression flag ไว้ที่หัวของเพจหรือไม่ หากใช่ จะทำการแปลง delta page นั้นด้วยวิธี XOR กับเพจเวอร์ชันก่อนหน้านี้ เพื่อให้ได้เพจใหม่นั่นเอง

งานวิจัยชื่อ High performance live migration through dynamic page transfer reordering and compression (Svärd, Tordsson, Hudzia, & Elmroth, 2011) เล็งเห็นว่าการอพยพแบบคงสถานะการทำงานคอมพิวเตอร์เสมือนที่กำลังทำงานกับโปรแกรมที่มีการประมวลผลสูงและใช้หน่วยความจำในปริมาณมาก หรือ การอพยพคอมพิวเตอร์เสมือนผ่านเครือข่ายที่มีแบนด์วิดท์ต่ำ จะมีปริมาณข้อมูลที่ส่งผ่านจำนวนมากมาย เพจของหน่วยความจำที่ถูกเขียนซ้ำบ่อยถูกส่งซ้ำหลายครั้ง ซึ่งเป็นการสูญเสียเครือข่ายแบนด์วิดท์ที่มีค่าไป จึงได้นำเสนอการผสมผสานเทคนิคการจัดอันดับการส่งเพจหน่วยความจำเพื่อลดความเสี่ยงจากการส่งหน่วยความจำที่ถูกเขียนซ้ำบ่อยๆ ร่วมกับการบีบอัดข้อมูล



ภาพที่ 2.9 กลไกการทำงานของการจัดอันดับเพจหน่วยความจำของ KVM (Hudzia, 2011)

ภาพที่ 2.10 การจัดอันดับหน่วยความจำของ KVM (Hudzia, 2011)

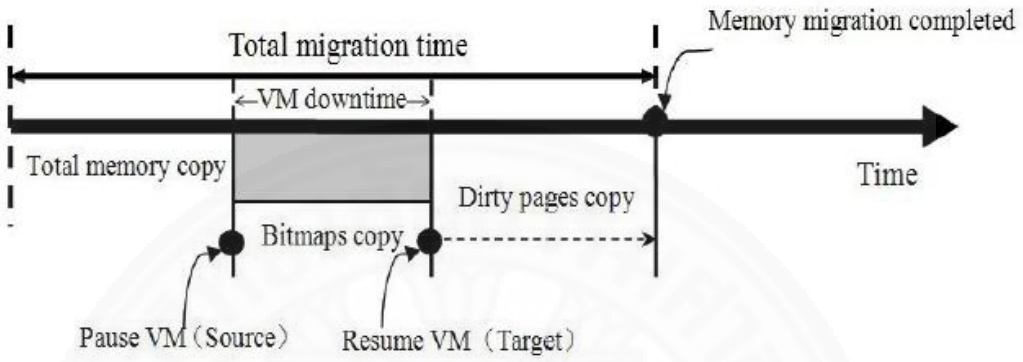
ภาพที่ 2.9 แสดงให้เห็นถึงการทำงานของกลไกการจัดอันดับเพจหน่วยความจำของ KVM ซึ่งจะใช้การคำนวณน้ำหนักให้กับเพจ ซึ่งเพจที่ถูกเขียนซ้ำหลายครั้งจะมีน้ำหนักมากขึ้นเรื่อยๆ และจะถูกจัดอันดับให้ถูกส่งไปทีหลัง ภาพที่ 2.10 ซึ่งแสดงให้เห็นถึงการจัดอันดับใหม่โดยที่เพจที่ถูกเขียนซ้ำบ่อยๆ จะถูกจัดให้ไปอยู่ด้านล่าง และเพจที่มีอัตราการเขียนซ้ำน้อยจะถูกเรียงไปไว้ข้างบน เทคนิคนี้จึงทำให้มีปริมาณหน่วยความจำที่ต้องส่งผ่านเครือข่ายลดลง อย่างไรก็ตาม เทคนิคนี้ยังไม่สามารถลดเวลาหยุดคอมพิวเตอร์เสมือนได้มากพอ จึงต้องนำเทคนิคการบีบอัดข้อมูลจากงานวิจัยก่อนหน้านี้ (Svärd, Hudzia, Tordsson & Elmroth, 2011) ซึ่งใช้ delta compression ด้วยการทำ XOR binary Zero RLE (XBZRLE) มาใช้ร่วมกัน ซึ่งทำให้เวลาในการอพยพคอมพิวเตอร์เสมือนผ่านเครือข่ายที่มีแบนด์วิดท์ 1000 Mbps ลดลงประมาณ 30% และสามารถช่วยลดปริมาณหน่วยความจำที่ถูกถ่ายโอนลงถึง 39%

อย่างไรก็ตาม KVM ใช้ delta compression โดยอาศัยแคชและพัฒนากลไกการทำ delta compression และ page reordering บนคอมพิวเตอร์เสมือนเลยซึ่งส่งผลกระทบต่อการทำงานของคอมพิวเตอร์เสมือน

งานวิจัยชื่อ HMDC: Live Virtual Machine Migration Based on Hybrid Memory Copy and Delta Compression (Hu, Zhao, Xu, Ding & Chu, 2013) นำเสนอการอพยพแบบคงสถานะการทำงานที่มีชื่อว่า HMDC โดยมีแนวคิดแบ่งเป็น 2 ส่วนคือ ส่วนแรกคือ Hybrid Memory Copy ซึ่งได้ผสมผสานวิธีอพยพแบบคงสถานะการทำงานแบบการถ่ายโอนหน่วยความจำจากคอมพิวเตอร์เสมือนต้นทาง (Memory Pushing Copy) ซึ่งเป็นวิธีการของ Pre-Copy และ แบบการดึงหน่วยความจำที่คอมพิวเตอร์เสมือนต้องการใช้จากเครื่องต้นทาง (Memory Pulling Copy) ซึ่งเป็นวิธีการของ Post-Copy เข้าด้วยกัน แต่ยังคงอยู่บนพื้นฐานของ Pre-Copy ส่วนที่สองคือ การใช้

วิธี delta compression โดยการใช้หลักการ XOR binary RLE (run-length encoding) ในช่วงที่ทำการก็อปปี้ dirty pages

โดยยังคงมีกลไกการทำงาน 3 ขั้นตอน (ภาพที่ 2.11) ดังนี้



ภาพที่ 2.11 กลไกการทำงานของ HMDC

ขั้นตอนที่ 1 (Stage 1) : ถ่ายโอนหน่วยความจำทั้งหมดจากคอมพิวเตอร์เสมือนต้นทางไปยังคอมพิวเตอร์เสมือนปลายทาง ในระหว่างที่ทำการถ่ายโอนหน่วยความจำ หากบางเพจมีการถูกแก้ไข HMDC จะทำการก็อปปี้ข้อมูลดั้งเดิมไปใส่ไว้ในแคชก่อน และจะทำเครื่องหมายของเพจที่ถูกแก้ไข (dirty pages) ทั้งหมดนี้ลงใน dirtypage_bitmap

ขั้นตอนที่ 2 (Stage 2): เป็นช่วงที่หยุดการทำงานของคอมพิวเตอร์เสมือนต้นทาง HMDC ทำการสร้าง cache_bitmap จาก dirty_bitmap และ ข้อมูลที่อยู่ใน Cache แล้วคอมพิวเตอร์เสมือนต้นทางคัดลอกทั้ง 2 บิตแมบ (dirtypage_bitmap && cache_bitmap) ไปยังคอมพิวเตอร์เสมือนปลายทาง

ขั้นตอนที่ 3 (Stage 3): คอมพิวเตอร์เสมือนปลายทางเริ่มต้นการทำงาน และทำการร้องขอ dirty pages จากคอมพิวเตอร์ต้นทางเป็นระยะๆ หากว่า เพจที่ทำการร้องขอนี้มีข้อมูลดั้งเดิมเก็บอยู่ในแคช HMDC จะใช้วิธีการ XOR ระหว่างเพจปัจจุบันกับเพจก่อนหน้า เพื่อให้ได้ delta page และจะทำการบีบอัดข้อมูลเพื่อให้ได้ delta compression page ด้วยการใช้หลักการ XOR binary RLE เมื่อได้ delta compression page แล้ว คอมพิวเตอร์เสมือนต้นทางจะส่งไปยังคอมพิวเตอร์เสมือนปลายทาง ในขณะที่คอมพิวเตอร์เสมือนปลายก็จะใช้ XOR ระหว่าง delta page กับ ข้อมูลเก่าที่อยู่ในแคช เพื่อสร้างเพจใหม่ แต่หากว่า เพจที่ร้องขอไปนั้นไม่มีอยู่ในแคช คอมพิวเตอร์เสมือนต้นทางก็จะทำการส่งเพจนั้นมาให้คอมพิวเตอร์เสมือนปลายทางเลย

HMDC นั้นมีข้อมีกำหนดเบื้องต้นคือ อัตราการส่งข้อมูลนั้นน้อยกว่าความเร็วของเครือข่าย และ ขนาดของหน่วยความจำชั่วคราว (Cache) จะต้องมีขนาดใหญ่เพียงพอ

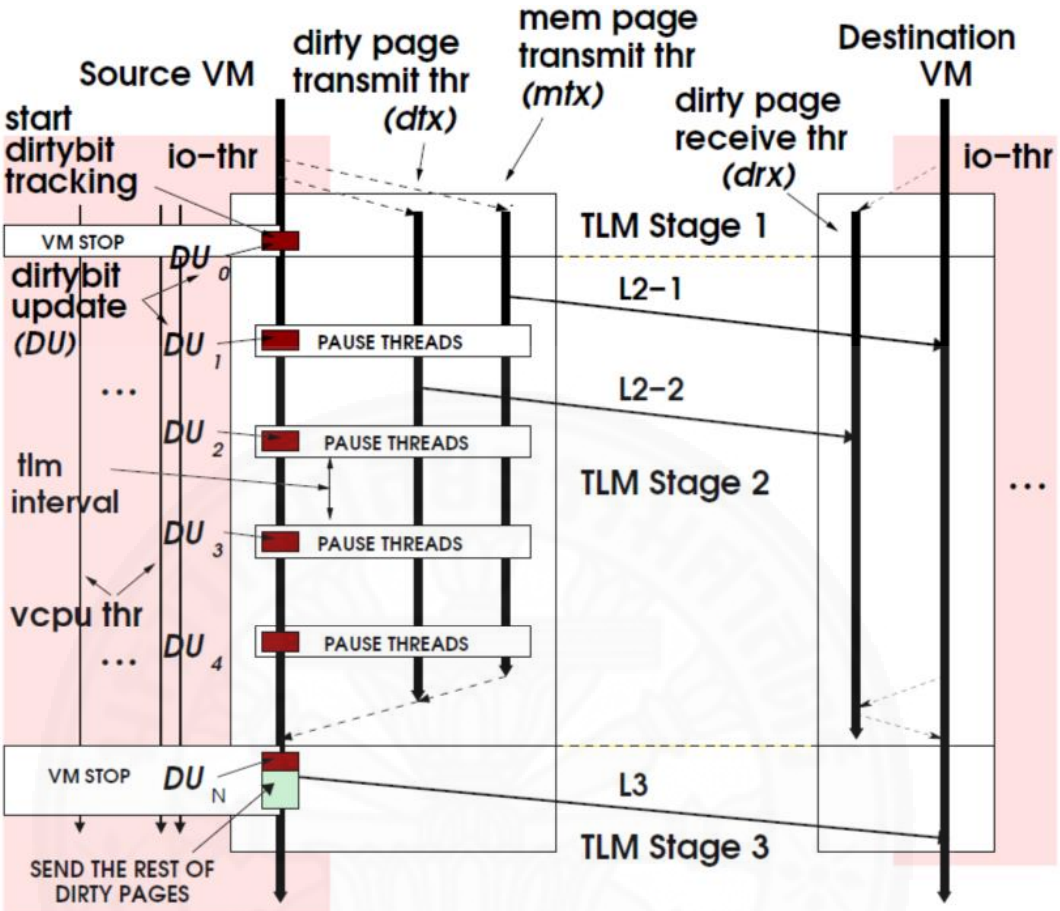
ผลการวิจัยสรุปว่า HMDC เพิ่มปริมาณการส่งเพจต่อวินาที (throughput) ลดจำนวนข้อมูลของการโอนย้ายลงด้วยการใช้เทคนิค delta compression และได้ก็อปปี้ dirty pages ได้อย่างรวดเร็ว จึงทำให้ HMDC ช่วยลดเวลารวมสำหรับการอพยพและเวลาหยุดคอมพิวเตอร์เสมือนเมื่อเทียบกับ การทำ Pre-copy และ XBRLE

Time-Bound, Thread-Based Live Migration of Virtual Machines (TLM) (Kasidit Chanchio & Phithak Thaenkaew, 2014) ปรับปรุงการอพยพแบบคงสถานะการทำงานของเควีเอ็มโดยการกำหนดให้มีขอบเขตในการอพยพ และสร้างเทรดเพิ่มขึ้นมาอีก 2 เทรดที่คอมพิวเตอร์เสมือนต้นทาง ได้แก่ mtx (memory page transmitter) และ เทรด dtx (dirty page transmitter) สร้างเทรดเพิ่มอีก 1 เทรดที่คอมพิวเตอร์เสมือนปลายทางคือ drx (dirty page receive) โดยเทรดที่สร้างเพิ่มขึ้นมาจะทำงานไปพร้อมกับเทรดหลักของคอมพิวเตอร์เสมือน

เทรด mtx ทำหน้าที่ถ่ายโอนหน่วยความจำที่เรียกว่า non-dirty page เมื่อเริ่มต้นการอพยพคอมพิวเตอร์เสมือน หน่วยความจำเพจจะเป็น non-dirty page ทั้งหมด ส่วนเทรด dtx ทำหน้าที่ถ่ายโอนหน่วยความจำที่เรียกว่า dirty page ในระหว่างที่ทำการอพยพคอมพิวเตอร์เสมือนหน่วยความจำจะมีการเปลี่ยนแปลงเรียกเพจนั้นว่า dirty page (เปรียบเสมือนว่า mtx ใช้แต่รถป้ายแดงเท่านั้น ส่วน dtx ใช้แต่รถมือสองเท่านั้น) และเทรดหลักของคอมพิวเตอร์เสมือนทำหน้าที่ควบคุมการอพยพ

กลไกการทำงานของ TLM แบ่งเป็น 3 ขั้นตอน (ภาพที่ 2.12) ดังนี้

ขั้นตอนที่ 1 เทรด io ของคอมพิวเตอร์เสมือนต้นทางสร้างเทรด mtx และ dtx หลังจากนั้น ทำการสร้างอาร์เรย์ของ dirty bit โดยสถานเริ่มต้นคือ clear และคอยสั่งให้โมดูล KVM kernel เริ่มกลไกการติดตามการเกิด dirty bit (dirty bit tracking mechanism) สร้างช่องทางการสื่อสารแบบขนาน 3 ช่องทาง คือ เทรด mtx สร้างการเชื่อมต่อ TCP กับเทรด io ที่คอมพิวเตอร์เสมือนปลายทาง (L2-1) ส่วนเทรด dtx สร้างการเชื่อมต่อ TCP กับเทรด drx ที่คอมพิวเตอร์เสมือนปลายทาง (L2-2) และเทรด io สร้างการเชื่อมต่อ TCP กับเทรด io ที่คอมพิวเตอร์เสมือนปลายทาง (L2-3)



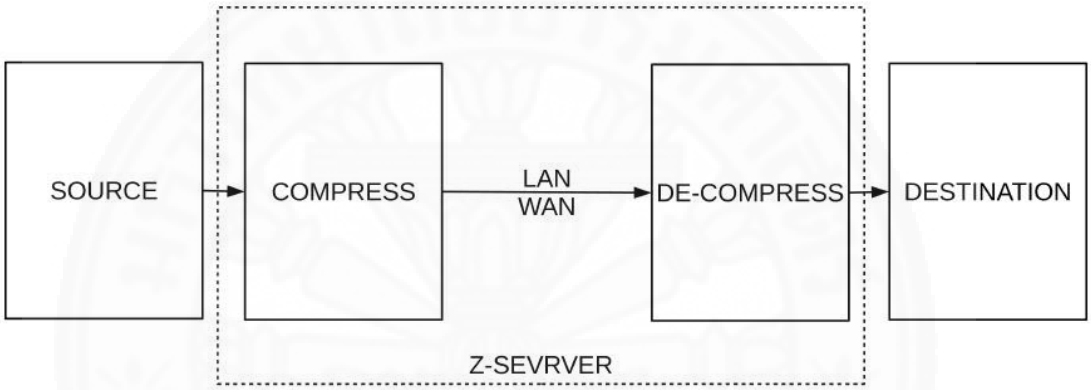
ภาพที่ 2.12 กลไกการทำงานของ TLM (Kasidit Chanchio & Phithak Thaenkaew, 2014)

ขั้นตอนที่ 2 เทรด mtx สแกนและถ่ายโอนหน่วยความจำ non-dirty page ของคอมพิวเตอร์เสมือนต้นทางตั้งแต่เพจที่ 0 ถึง เพจสุดท้าย ในขณะที่เทรด dtx อ่านค่าใน dirty bit array และถ่ายโอนหน่วยความจำที่เปลี่ยนแปลงไปยังคอมพิวเตอร์เสมือนปลายทาง

ขั้นตอนที่ 3 เมื่อ mtx ถ่ายโอนหน่วยความจำจนถึงเพจสุดท้ายคอมพิวเตอร์เสมือนต้นทางถูกสั่งให้หยุดทำงาน และเทรด io ถ่ายโอนหน่วยความจำที่เหลืออยู่ทั้งหมดไปยังคอมพิวเตอร์เสมือนปลายทาง และเริ่มการทำงานของคอมพิวเตอร์เสมือนปลายทาง

การกำหนดให้คอมพิวเตอร์เสมือนหยุดทำงานเมื่อเทรก mtx ถ่ายโอนหน่วยความจำถึงเพจสุดท้ายเช่นนี้ ทำให้ TLM สามารถอพยพแบบคงสถานะการทำงานได้สำเร็จภายในขอบเขตของเวลา อย่างไรก็ตาม TLM จำเป็นต้องใช้ฟีเจอร์ในการจัดสรรหน่วยประมวลผลกลาง (CPU over-committing) ของคอมพิวเตอร์เสมือนทำให้คอมพิวเตอร์เสมือนทำงานช้าลง เพื่อลดอัตราการเกิดการเปลี่ยนแปลงของหน่วยความจำ

งานวิจัยเรื่อง ทีแอลเอ็มแซด: ไลฟ์ไมเกรชันแบบเทรดบนคอมพิวเตอร์เสมือนโดยใช้เครื่องแม่ข่ายสำหรับบีบอัดข้อมูล (แท่นแก้ว, 2557) ได้ทำปรับปรุงระบบ TLM โดยการเพิ่มเครื่องแม่ข่ายในการบีบอัดข้อมูลที่มีชื่อว่า Z-Server (ภาพที่ 2.13) ซึ่งประกอบด้วย Compression Server และ Decompression Server ภายในเครื่องแม่ข่ายจะมีโปรแกรม netcat ทำหน้าที่ในการเชื่อมต่อเครือข่ายในลักษณะ client-server ได้ ทั้งแบบ TCP และ UDP โดยที่ไม่ต้องเขียนโปรแกรมแบบซ็อกเก็ต (Socket Programming) โดยเครื่องแม่ข่ายนี้ใช้ LZ4 ในการบีบอัดข้อมูลและถอดรหัสข้อมูลกลับมาเหมือนเดิม



ภาพที่ 2.13 ระบบ Z-Server (พิทักษ์ แท่นแก้ว, 2557)

บทที่ 3
วิธีการวิจัย

งานวิจัยนี้มีวัตถุประสงค์ที่จะเพิ่มประสิทธิภาพการอพยพแบบคงสถานะการทำงานของคอมพิวเตอร์เสมือนแบบเทรดด้วยการใช้เฟรมเวิร์คสำหรับการบีบอัดสองชั้นโดยการเพิ่มส่วนประกอบใหม่คือ ระบบบีบอัดหน่วยความจำเพจ หรือ Memory Server เข้ามาระหว่างคอมพิวเตอร์เสมือนและเครื่องแม่ข่ายสำหรับบีบอัด Z-Server ของระบบ TLM-Z

ในเฟรมเวิร์คสำหรับบีบอัดข้อมูลสองชั้นนี้ Memory Server ทำหน้าที่บีบอัดข้อมูลในระดับหน่วยความจำเพจ (Memory Page) ในขณะที่ Z-Server ทำหน้าที่บีบอัดข้อมูลแบบ Stream ในงานวิจัยนี้ ผู้วิจัยได้ออกแบบให้ Memory Server สามารถจัดการบีบอัดหน่วยความจำ 2 วิธี ได้แก่

(1) การเข้ารหัสข้อมูลแบบเดลตา โดยใช้วิธีการ XOR [TLM-XZ} และ

(2) การเพิ่มการบีบอัดข้อมูลด้วย Blosc ที่สามารถกำหนดให้ใช้วิธีการบีบอัดดังต่อไปนี้ BloscLZ, LZ4, LZ4hc, Snappy และ Zlib [TLM-BZ]

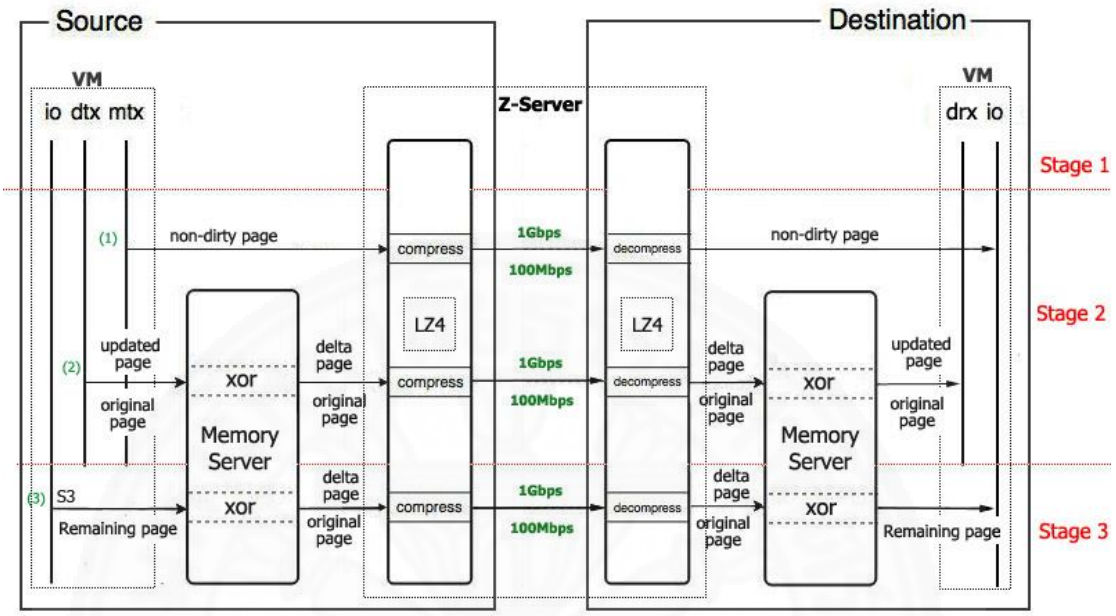
3.1 สถาปัตยกรรม

3.1.1 การอพยพแบบคงสถานะการทำงานแบบ TLM-XZ

การอพยพแบบคงสถานะการทำงานแบบ TLM-XZ ดังภาพที่ 3.1 ซึ่ง VM หมายถึงคอมพิวเตอร์เสมือนต้นทางและปลายทาง TLM-XZ ประกอบด้วยการทำงานของเทรด mtx และ dtx ซึ่งทำงานไปพร้อมกับการประมวลผลของคอมพิวเตอร์เสมือนต้นทางและเทรด drx ทำงานไปพร้อมกับการประมวลผลของคอมพิวเตอร์เสมือนปลายทาง โดยเทรด io ที่อยู่ในคอมพิวเตอร์เสมือนต้นทางและปลายทางทำหน้าที่ควบคุมการการอพยพแบบคงสถานะการทำงาน ถัดจากนั้นเป็น Memory Server ในฝั่งของคอมพิวเตอร์ต้นทาง (Source Memory Server) ซึ่งเป็นซอฟต์แวร์ที่จะรันอยู่บนคอมพิวเตอร์จริงเครื่องเดียวกันกับคอมพิวเตอร์เสมือน หรือ แยกเป็นเครื่องคอมพิวเตอร์ต่างหากก็ได้ ถัดจากนั้นไปก็คือ Z Server ซึ่งประกอบไปด้วย Compression Server บนฝั่งต้นทางที่จะทำการเข้ารหัสถอดรหัสโดยใช้เครื่องมือบีบอัด LZ4 และถ่ายโอนข้อมูลหน่วยความจำไปยังปลายทางผ่านระบบเครือข่าย

ในฝั่งปลายทางเริ่มต้นที่ Decompression Server ที่จะรับข้อมูลและถอดรหัสข้อมูลหน่วยความจำด้วย LZ4 และถ่ายโอนข้อมูลหน่วยความจำให้ Memory Server เพื่อถอดรหัส

คืนข้อมูลหน่วยความจำเพจแล้วถ่ายโอนข้อมูลหน่วยความจำนั้นต่อให้กับเทรด drx ที่ TLM-XZ สร้างขึ้นบนฝั่งผู้รับเพื่อนำข้อมูลหน่วยความจำเพจไปเก็บในหน่วยความจำของเครื่องคอมพิวเตอร์เสมือน



ภาพที่ 3.1 แสดงการทำงานของการอพยพแบบคงสถานะการทำงานแบบ TLM-XZ

กลไกการทำงานของ TLM-XZ มี 3 ขั้นตอน (ตามภาพที่ 3.1) ดังนี้

ขั้นตอนที่ 1 (stage 1) ขั้นตอนนี้เป็นการสร้างการเชื่อมต่อตั้งต้นสำหรับการอพยพแบบคงสถานะการทำงาน ที่คอมพิวเตอร์เสมือนต้นทาง ทำการสร้าง เทรด mtx และ dtx หลังจากนั้น เทรด io สร้างอาร์เรย์ของ dirty bit ถัดจากนั้น TLM-XZ จะสร้างช่องทางสื่อสารแบบขนานขึ้นสามช่องทางได้แก่

(1) เส้นทางเชื่อมต่อที่ 1 (Connection Path ที่ 1 ในภาพ) เทรด mtx สร้างการเชื่อมต่อแบบ TCP กับ Compression Server บนฝั่งผู้ส่งซึ่งจะสร้างช่องทางสื่อสาร TCP กับ Decompression Server บนเครื่องปลายทาง และ Decompression Server จะสร้างช่องทางสื่อสาร TCP กับเทรด io ของเครื่องคอมพิวเตอร์เสมือนปลายทาง

(2) เส้นทางเชื่อมต่อที่ 2 (Connection Path ที่ 2 ในภาพ) เทรด dtx สร้างการเชื่อมต่อกับ Memory Server บนฝั่งผู้ส่งซึ่งจะสร้างช่องทางสื่อสารกับ Compression Server บนฝั่งผู้ส่งและเชื่อมต่อกับ Decompression Server บนฝั่งผู้รับ และ Decompression Server จะเชื่อมต่อกับ Memory Server บนฝั่งผู้รับซึ่งจะส่งข้อมูลต่อให้กับเทรด drx ของเครื่องคอมพิวเตอร์เสมือนต่อไป

(3) เส้นทางเชื่อมต่อที่ 3 (Connection Path ที่ 3 ในภาพ) เทรด io ของเครื่องคอมพิวเตอร์เสมือนบนฝั่งผู้ส่งสร้างช่องทาง TCP เชื่อมต่อกับ Memory Server บนฝั่งผู้ส่งซึ่งจะสร้างช่องทางสื่อสารกับ Compression Server ซึ่งจะสร้างช่องทาง TCP ติดต่อกับ Decompression Server ที่จะส่งข้อมูลต่อให้ Memory Server บนฝั่งผู้รับและส่งต่อข้อมูลให้เทรด io ของคอมพิวเตอร์เสมือนปลายทางในที่สุด

ขั้นตอนที่ 2 (stage 2) เทรด mtx เริ่มถ่ายโอนหน่วยความจำ (non-dirty pages) ที่ 0 ถึง หน่วยความจำสุดท้ายไปยัง Compression Server เพื่อทำการบีบอัดด้วย LZ4 และถ่ายโอนต่อไปยัง Decompression Server ที่เครื่องปลายทาง ทำการถอดรหัสหน่วยความจำด้วย LZ4 และถ่ายโอนต่อไปยังคอมพิวเตอร์เสมือนปลายทาง โดยมีเทรด io ของเครื่องคอมพิวเตอร์เสมือนปลายทางมารับหน่วยความจำ

ในขณะที่ เทรด dtx ถ่ายโอนหน่วยความจำที่เปลี่ยนแปลง (dirty page) โดยในทุกๆ 1 วินาที (เป็นพารามิเตอร์ที่สามารถกำหนดได้ ในงานวิจัยนี้กำหนดไว้ที่ 1 วินาที) นั้น เทรด io จะอัพเดทอาร์เรย์ของ dirty bit และ เทรด dtx จะถ่ายโอนหน่วยความจำจากการอ่านอาร์เรย์ของ dirty bit ไปยัง Memory Server ซึ่งมีกลไกในการเข้ารหัสข้อมูลแบบเดลตาด้วยวิธี XOR โดย Memory Server จะตรวจเช็คหน่วยความจำที่ได้รับว่า เป็นหน่วยความจำเริ่มต้น $P_{orginal}$ หรือไม่ หากเป็นหน่วยความจำเริ่มต้น Memory Server จะทำการถ่ายโอนหน่วยความจำเริ่มต้น $P_{orginal}$ นั้นต่อไปให้กับ Compression Server ในทันที แต่หากว่าเป็นหน่วยความจำที่มีการเปลี่ยนแปลง $P_{updated}$ Memory Server จะเข้ารหัสหน่วยความจำด้วย XOR ได้หน่วยความจำเฉพาะส่วนต่าง P_{delta} ถ่ายโอนไปให้ Compression Server ทำการบีบอัดได้ $P_{delta\ compression}$ และถ่ายโอนต่อไปยัง De-compression Server ที่เครื่องปลายทางถอดรหัสหน่วยความจำ P_{delta} และถ่ายโอนต่อไปยัง Memory Server เข้ารหัสหน่วยความจำเพื่อให้ได้หน่วยความจำที่มีการเปลี่ยนแปลง $P_{updated}$ และจึงถ่ายโอนต่อไปยังเทรด drx ที่คอมพิวเตอร์เสมือนปลายทาง

ขั้นตอนที่ 3 (stage 3) เมื่อเทรด mtx ทำการสแกนและถ่ายโอนหน่วยความจำครบแล้ว คอมพิวเตอร์เสมือนต้นทางจะถูกสั่งให้หยุดการทำงาน และ เทรด io จะถ่ายโอนหน่วยความจำที่เหลืออยู่ทั้งหมด $P_{remaining}$ ไปให้ Memory Server ซึ่งมีกลไกการทำงานเหมือนดังขั้นตอนที่ 2 คือเข้ารหัสหน่วยความจำด้วย XOR ได้หน่วยความจำเฉพาะส่วนต่าง P_{delta} ถ่ายโอนไปให้ Compression Server ทำการบีบอัด $P_{delta\ compression}$ และถ่ายโอนต่อไปยัง De-compression Server ที่เครื่องปลายทางถอดรหัสหน่วยความจำ P_{delta} และถ่ายโอนต่อไปยัง Memory Server ทำการถอดรหัสหน่วยความจำด้วย XOR เพื่อให้ได้หน่วยความจำที่มีการเปลี่ยนแปลง $P_{updated}$ และถ่ายโอนต่อไปยังเทรด io ที่คอมพิวเตอร์เสมือนปลายทาง

**เนื่องจากว่า ในการเข้ารหัสเดลตาด้วยวิธี XOR แล้วพบว่า มีข้อมูลจำนวนหนึ่ง ซึ่งเมื่อทำการเปรียบเทียบระหว่างหน่วยความจำเริ่มต้นกับหน่วยความจำที่เปลี่ยนแปลง ปรากฏว่า หน่วยความจำทั้งสองเหมือนกันเลย ดังนั้น จึงมีกลไกในการสั่งให้ Memory Server ไม่ถ่ายโอนต่อหน่วยความจำนั้นไปยังคอมพิวเตอร์เสมือนปลายทาง

โดยสรุปแล้วการแปลงค่าของหน่วยความจำเพจบนเครื่องต้นทางเป็นไปตามสมการที่ (1) และ (2) และ การแปลงข้อมูลหน่วยความจำเพจบนเครื่องปลายทางเป็นไปตาม (3) และ (4)

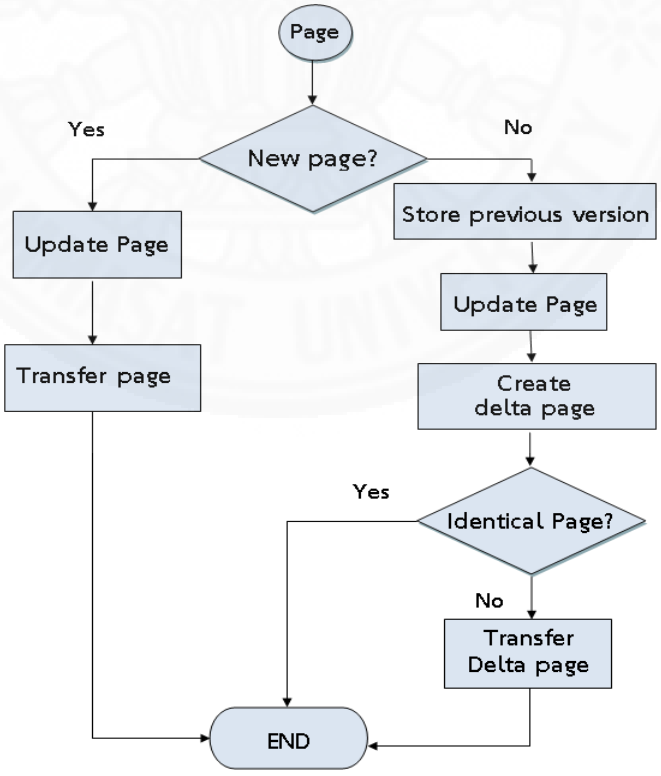
$$(P_{original} \oplus P_{updated}) = P_{delta} \quad (1)$$

$$Comp(P_{delta}) = P_{delta\ compression} \quad (2)$$

$$Decomp(P_{delta\ compression}) = P_{delta} \quad (3)$$

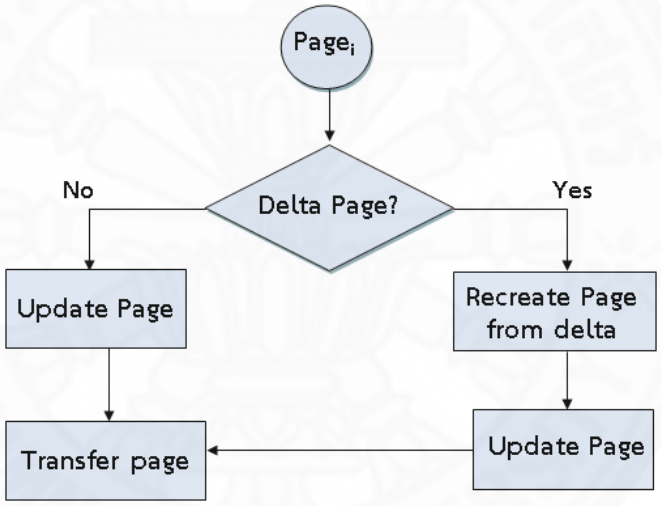
$$(P_{delta} \oplus P_{original}) = P_{updated} \quad (4)$$

3.1.1.1 กลไกการทำงานของ Memory Server



ภาพที่ 3.2 กลไกการทำงานของ memory server ที่เครื่องต้นทางสำหรับ TLM-XZ

การอพยพแบบคงสถานะการทำงาน TLM-Z แบบที่มีการเพิ่ม XOR มีกลไกการทำงานตามภาพที่ 3.2 คือ Memory Server ที่เครื่องต้นทางตรวจเช็คว่า หน่วยความจำเพจที่ถูกถ่ายโอนมานั้นเป็นหน่วยความจำเพจใหม่หรือไม่ หากเป็นหน่วยความจำใหม่ ก็จะเก็บสำเนาของหน่วยความจำนั้นไว้และถ่ายโอนหน่วยความจำนั้นให้กับ Compression Server หากเป็นหน่วยความจำเพจที่เคยถ่ายโอนมาแล้ว Memory Server จะทำการเปรียบเทียบความแตกต่างของหน่วยความจำเพจที่เพิ่งได้รับกับหน่วยความจำเพจที่ได้รับก่อนหน้าด้วยวิธี XOR ได้หน่วยความจำเพจเฉพาะที่แตกต่าง (delta page) และตรวจเช็คว่า delta page นี้เป็น identical page (หรือ delta page มีค่าเท่ากับ 0 แสดงว่า ข้อมูลหน่วยความจำที่ได้รับมาก่อนหน้ากับหน่วยความจำที่เพิ่งได้รับมานั้นเหมือนกัน) หากว่าเป็น identical page ก็ไม่จำเป็นต้องถ่ายโอนหน่วยความจำนั้นไปอีก หากไม่เป็น identical page จึงถ่ายโอน delta page ไปให้ Compression Server

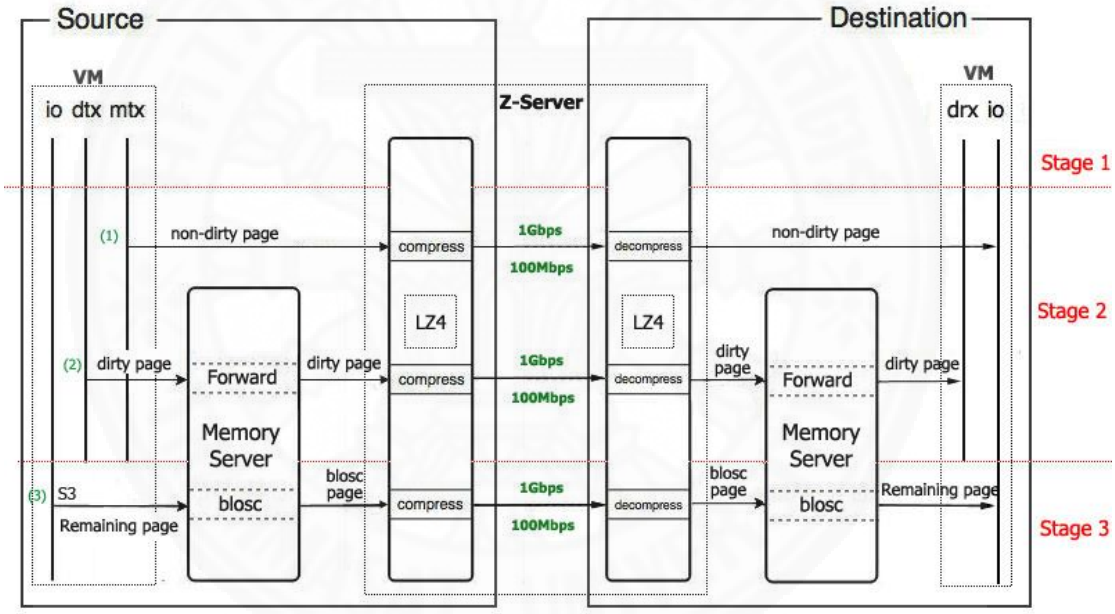


ภาพที่ 3.3 กลไกการทำงานของ memory server ที่เครื่องปลายทางสำหรับ TLM-XZ

ส่วน Memory Server ที่เครื่องปลายทางตามภาพที่ 3.3 มีกลไกดังนี้ Memory Server เช็คหน่วยความจำเพจที่ได้รับว่าเป็น delta page หรือไม่ หากไม่เป็น จะอัพเดทข้อมูลหน่วยความจำนั้นและถ่ายโอนหน่วยความจำนั้นไปยังคอมพิวเตอร์เสมือนปลายทาง แต่หากว่าเพจที่ได้รับมาเป็น delta page จำเป็นต้องถอดรหัสข้อมูลกลับด้วยวิธี XOR แล้วจึงอัพเดทหน่วยความจำเพจนั้น และถ่ายโอนหน่วยความจำเพจนั้นไปยังคอมพิวเตอร์เสมือนปลายทาง

3.1.2 การอพยพแบบคงสถานะการทำงานแบบ TLM-BZ

การอพยพแบบคงสถานะการทำงาน TLM-Z แบบที่มีการเพิ่ม Blosc นั้น Memory Server ที่เครื่องต้นทางจะทำหน้าที่ถ่ายโอนต่อหน่วยความจำไปให้กับ Compression Server สำหรับหน่วยความจำที่เทรด dtx ถ่ายโอนมา และจะทำการบีบอัดหน่วยความจำที่เทรด io ถ่ายโอนมาในขั้นตอนที่ 3 ด้วย Blosc ซึ่งอาศัยเทรดในการบีบอัดข้อมูลและมีกลไกการใช้ shuffle ซึ่งเป็นชุดคำสั่ง SIMD ทำการจัดเรียงหน่วยความจำก่อนนำไปผ่านเครื่องมือในการบีบอัด bloscLZ, LZ4, LZ4hc, Snappy และ Zlib ส่วน Memory Server ที่เครื่องปลายทางอาศัยเทรดในการถอดรหัสหน่วยความจำที่ได้รับจาก decompression server กลับด้วยเครื่องมือบีบอัดเดียวกับที่ใช้ที่เครื่องต้นทาง



ภาพที่ 3.4 แสดงการทำงานของการอพยพแบบคงสถานะการทำงาน TLM-BZ

กลไกการทำงานของ TLM-BZ นั้น มี 3 ขั้นตอน (ตามภาพที่ 3.4) ดังนี้

ขั้นตอนที่ 1 (stage 1) ที่คอมพิวเตอร์เสมือนต้นทาง ทำการสร้าง เทรด mtx และ dtx หลังจากนั้น เทรด io จะทำการจะสร้างอาร์เรย์ของ dirty bit ส่วนเทรด mtx จะสร้างการเชื่อมต่อกับ Z-Server และ เทรด dtx จะสร้างการเชื่อมต่อกับ Memory Server [การทำงานขั้นตอนนี้จะเหมือนกับแบบ TLM-XZ]

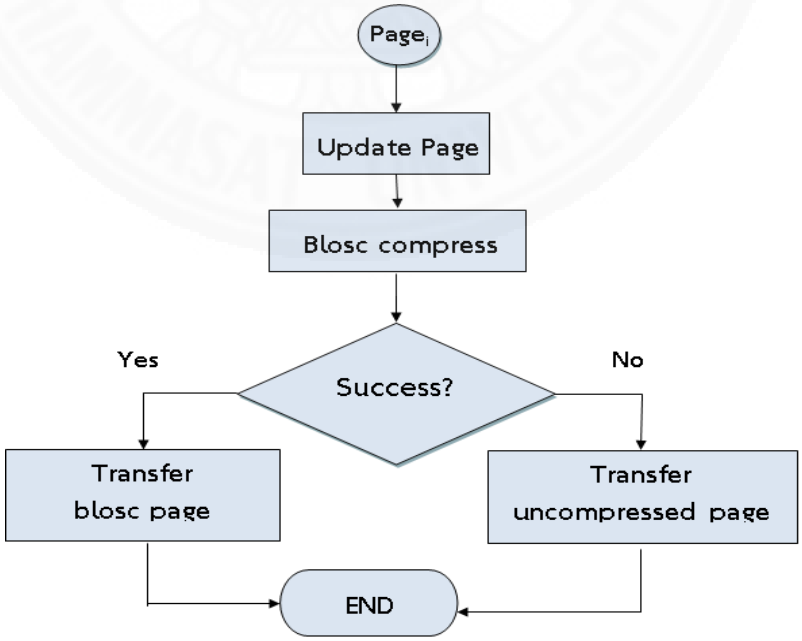
ขั้นตอนที่ 2 (stage 2) เทรด mtx จะเริ่มถ่ายโอนหน่วยความจำ (non-dirty pages) ที่ 0 ถึง หน่วยความจำสุดท้ายไปยัง Compression Server เพื่อทำการบีบอัดด้วย LZ4 และ

ถ่ายโอนต่อไปยัง Decompression Server ที่เครื่องปลายทาง ถอดรหัสหน่วยความจำด้วย LZ4 และถ่ายโอนต่อไปยัง คอมพิวเตอร์เสมือนปลายทาง โดยมี io thread ในการรับหน่วยความจำ

ในขณะที่ เทรด dtx ถ่ายโอนหน่วยความจำที่เปลี่ยนแปลง (dirty page) โดยในทุกๆ 1 วินาทีนั้น เทรด io จะอัพเดทอาร์เรย์ของ dirty bit และ เทรด dtx จะถ่ายโอนหน่วยความจำจากการอ่านอาร์เรย์ของ dirty bit ไปยัง Memory Server และเพียงถ่ายโอนข้อมูลหน่วยความจำต่อไปยัง Compression Server บีบอัดและถ่ายโอนต่อไปยัง De-compression Server ที่เครื่องปลายทางถอดรหัสหน่วยความจำและถ่ายโอนต่อไปยัง Memory Server ถ่ายโอนต่อไปยังเทรด drx ที่คอมพิวเตอร์เสมือนปลายทาง

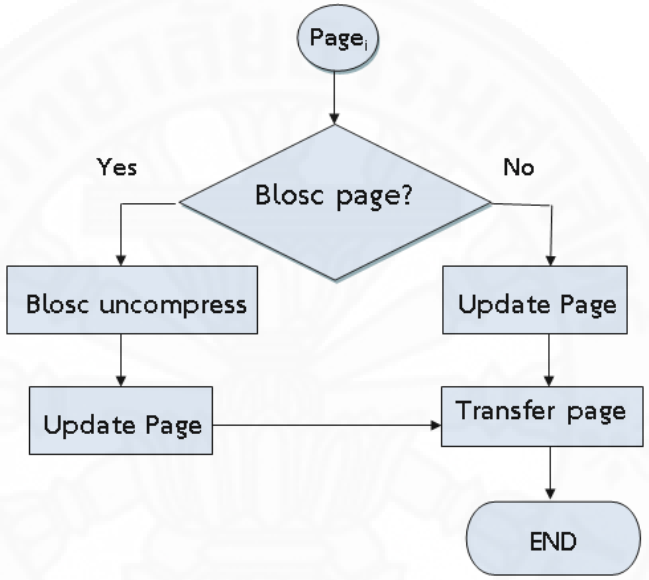
ขั้นตอนที่ 3 (stage 3) เมื่อเทรด mtx ทำการสแกนและถ่ายโอนหน่วยความจำครบแล้ว คอมพิวเตอร์เสมือนต้นทางจะถูกสั่งให้หยุดการทำงาน และ เทรด io จะทำการถ่ายโอนข้อมูลที่เหลืออยู่ทั้งหมด $P_{remaining}$ ไปให้ Memory Server ซึ่งมี Blosc ทำหน้าที่บีบอัดหน่วยความจำทีละเพจ เมื่อบีบอัดแล้วจึงถ่ายโอนเพจที่ถูกบีบอัด $P_{compressed}$ ไปยัง Compression Server โดย LZ4 LZ4 บีบอัดข้อมูลอีกครั้งแต่เป็นการบีบอัดข้อมูลแบบสตรีมมิ่งมีขนาดบล็อก 64 KB ต่อจากนั้นจึงถ่ายโอนข้อมูลไปยัง Decompression Server ที่เครื่องปลายทางถอดรหัสหน่วยความจำและถ่ายโอนหน่วยความจำนั้นไปยัง Memory Server ซึ่งมี Blosc ทำหน้าที่ถอดรหัสข้อมูลกลับและถ่ายโอนหน่วยความจำต่อไปยังเทรด io ที่คอมพิวเตอร์เสมือน

3.1.2.1 กลไกการทำงานของ Memory Server



ภาพที่ 3.5 กลไกการทำงานของ memory server ที่เครื่องต้นทางสำหรับ TLM-BZ

การอพยพแบบคงสถานะการทำงาน TLM-Z แบบที่มีการเพิ่ม Blosc มีกลไกการทำงานตามภาพที่ 3.5 คือ Memory Server ที่เครื่องต้นทาง เมื่อได้รับหน่วยความจำเพจจะทำการอัพเดทหน่วยความจำเพจนั้น และบีบอัดข้อมูลด้วย blosc แล้วตรวจเช็คว่า blosc สามารถบีบอัดหน่วยความจำเพจนั้นสำเร็จหรือไม่ หากสำเร็จ Memory Server จะถ่ายโอนหน่วยความจำเพจที่ถูกบีบอัดด้วย Blosc (blosc page) ให้กับ Compression Server หากว่า บีบอัดหน่วยความจำเพจนั้นไม่สำเร็จ Memory Server จะถ่ายโอนหน่วยความจำเพจแบบที่ไม่ถูกบีบอัดไปให้ Compression Server

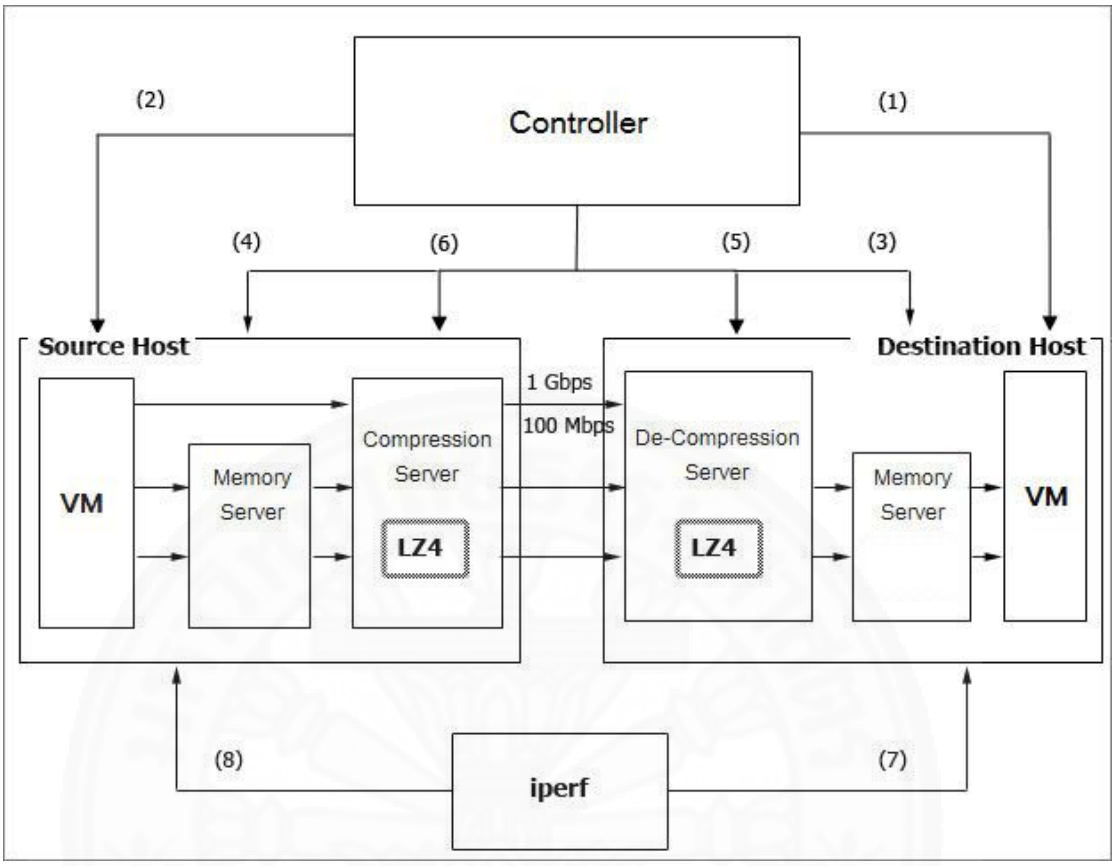


ภาพที่ 3.6 กลไกการทำงานของ memory server ที่เครื่องปลายทางสำหรับ TLM-BZ

ส่วน Memory Server ที่เครื่องปลายทางตามภาพที่ 3.6 มีกลไกดังนี้ Memory Server เช็คหน่วยความจำเพจที่ได้รับว่าเป็น blosc page หรือไม่ หากไม่เป็น จะอัพเดทข้อมูลหน่วยความจำนั้นและถ่ายโอนหน่วยความจำนั้นไปยังคอมพิวเตอร์เสมือนปลายทาง แต่หากว่าเพจที่ได้รับมาเป็น blosc page จำเป็นต้องถอดรหัสข้อมูลกลับด้วย blosc แล้วจึงอัพเดทหน่วยความจำเพจนั้น และถ่ายโอนหน่วยความจำเพจนั้นไปยังคอมพิวเตอร์เสมือนปลายทาง

3.2 การออกแบบระบบ

ผู้วิจัยได้ผนวก Memory Server เข้ากับระบบ TLM-Z โดยมีสถาปัตยกรรมดังแสดงในภาพที่ 3.7 ซึ่งประกอบด้วย เครื่องคอมพิวเตอร์จำนวน 4 เครื่อง



ภาพที่ 3.7 สถาปัตยกรรมของเฟรมเวิร์คสำหรับการบีบอัดสองชั้น

1. เครื่องควบคุม (Controller) มีซีพียู Intel Xeon E5530 2.4 GHz มีหน่วยประมวลผล 8 คอร์ หน่วยความจำ 16 GB ติดตั้งระบบปฏิบัติการลินุกซ์ Ubuntu 12.04 x86_64 LTS ทำหน้าที่ในการควบคุมระบบ

2. เครื่องต้นทาง (Source Host) มีซีพียู AMD Opteron 2.1 GHz มีหน่วยประมวลผล 12 คอร์ หน่วยความจำ 48 GB ติดตั้งระบบปฏิบัติการลินุกซ์ Ubuntu 12.04 x86_64 ทำหน้าที่เป็นคอมพิวเตอร์เสมือนต้นทาง เชื่อมต่อกับ Memory Server ซึ่งทำหน้าที่บีบอัดข้อมูลในระดับหน่วยความจำ และถ่ายโอนหน่วยความจำไปยัง Compression Server ซึ่งทำหน้าที่ในการบีบอัดหน่วยความจำแบบ Stream

3. เครื่องปลายทาง (Destination Host) มีซีพียู AMD Opteron 2.1 GHz มีหน่วยประมวลผล 12 คอร์ หน่วยความจำ 48 GB ติดตั้งระบบปฏิบัติการลินุกซ์ Ubuntu 12.04 x86_64 ทำหน้าที่เป็นคอมพิวเตอร์เสมือนปลายทาง โดยมี Decompression Server ซึ่งเชื่อมต่อกับ Compression Server ผ่านเครือข่ายความเร็ว 1 Gbps และ 100 Mbps ทำหน้าที่ถอดรหัสหน่วยความจำเดิมแบบ Stream กลับมา และถ่ายโอนหน่วยความจำไปยัง Memory server เพื่อ

ถอดรหัสหน่วยความจำเดิมในระดับหน่วยความจำเพจกลับ และจึงถ่ายโอนไปที่คอมพิวเตอร์เสมือนปลายทาง

4. iperf ซึ่งจะทำหน้าที่ในการตรวจจับความเร็วของเครือข่าย และเก็บข้อมูลลงไฟล์

ระบบจะเริ่มทำงานด้วยการสั่งงานของ ตัวควบคุมระบบ ซึ่งมีขั้นตอนดังนี้

1. สั่งให้คอมพิวเตอร์เสมือนบนเครื่องปลายทาง (Destination Host) เริ่มทำงาน
2. สั่งให้คอมพิวเตอร์เสมือนบนเครื่องต้นทาง (Source Host) เริ่มทำงาน
3. เริ่มการทำงานของ Memory Server ที่เครื่องปลายทาง
4. เริ่มการทำงานของ Memory Server ที่เครื่องต้นทาง
5. เริ่มการทำงานของ Decompression Server บนเครื่องปลายทาง
6. เริ่มการทำงานของ Compression Server บนเครื่องปลายทาง
7. iperf เริ่มการตรวจวัดความเร็วเครือข่ายของ server
8. iperf เริ่มการตรวจวัดความเร็วเครือข่าย client

3.3 วิธีการทดลอง

ในการทดลองเบื้องต้นนี้ ใช้เกสโอเอส Scientific Linux 6.5 x86_64 ซึ่งกำหนดให้คอมพิวเตอร์เสมือนรัน NAS Parallel Benchmarks (NPB) จำนวน 7 โปรแกรม ได้แก่ bt.C, cg.C, is.C, lu.C, mg.C, sp.C และ ua.C ซึ่งแต่ละโปรแกรมกำหนดให้ใช้ซีพียู 8 คอร์ หน่วยความจำขนาด 8 GB ส่วนไฟล์ดิสก์อิมเมจมีขนาด 500 เมกะไบต์ และถือว่าไฟล์ดิสก์อิมเมจที่เก็บบนเครื่องต้นทางจะต้องเป็นชุดเดียวกับเครื่องปลายทาง และเพื่อให้เป็นไปตามเงื่อนไขดังกล่าวนั้น ในการทดลองนี้ ได้เลือกวิธีการแชร์ผ่าน NFS

สำหรับการกำหนดค่าพารามิเตอร์ของ TLM-Z นั้น ได้กำหนด tlm interval (ช่วงเวลาที่ io thread อัพเดท dirty bit) ไว้ที่ 1 วินาที กำหนดระยะเวลาในการเริ่มการอพยพแบบคงสถานะการทำงานหลังจากให้โปรแกรมทำงานไว้ที่ 20 วินาที เครือข่ายความเร็วกำหนดไว้ที่ 1 Gbps และ 100 Mbps เครื่องแม่ข่ายในการบีบอัดข้อมูล Z-Server ใช้เทคนิค LZ4 บล็อกข้อมูลขนาด 64 KB โดยใช้การบีบอัดที่ระดับ 1 คือระดับที่เน้นความเร็ว ซึ่งเป็นค่าพารามิเตอร์ที่ดีสุด (พิทักษ์ แท่นแก้ว, 2557)

ส่วนค่าพารามิเตอร์สำหรับ Blosc ได้แก่ เครื่องมือในการบีบอัด BloscLZ, LZ4, LZ4hc, Snappy และ Zlib เทรดจำนวน 2 เทรด และ 4 เทรด กำหนดระดับการบีบอัดไว้ที่ 6 ใช้ filter แบบ byte-wise shuffle และ bit-wise shuffle

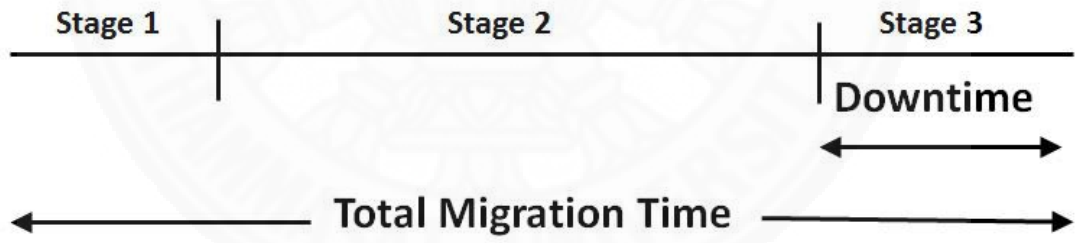
3.4 การวัดประสิทธิภาพ

ประเมินประสิทธิภาพของ TLM-XZ และ TLM-BZ เทียบกับ TLM และ TLM-Z โดยใช้ชุดโปรแกรม NPB เวอร์ชัน 3.3.1 คลาส C จำนวน 7 โปรแกรม ในเรื่องต่อไปนี้

1. วัดประสิทธิภาพการบีบอัดของเครื่องแม่ข่าย Z-Server (ใช้วิธีบีบอัด LZ4) ด้วย Space Saving ซึ่งเป็นการวัดอัตราส่วนของข้อมูลที่ถูกบีบอัดกับข้อมูลที่ยังไม่ถูกบีบอัด สามารถบอกได้ว่า การบีบอัดมีประสิทธิภาพมากเพียงใดจากการประหยัดพื้นที่ของแบนด์วิดท์ในการถ่ายโอนข้อมูลหน่วยความจำ

2. เวลารวมสำหรับการอพยพ (Total Migration Time) โดยนับตั้งแต่ ที่ io thread เริ่มทำงานขั้นตอนที่ 1 จนกระทั่งคอมพิวเตอร์เสมือนต้นทางถูกอพยพไปเริ่มทำงานต่อที่คอมพิวเตอร์เสมือนปลายทาง

3. เวลาที่คอมพิวเตอร์เสมือนหยุดทำงาน (Downtime) เมื่อเทรด mtx ถ่ายโอนหน่วยความจำจากคอมพิวเตอร์เสมือนต้นทางไปยังเครื่องปลายทางเรียบร้อย Downtime จะเริ่มนับตั้งแต่เวลาที่หยุดคอมพิวเตอร์เสมือนต้นทาง ไปจนถึงเวลาที่เทรด io ถ่ายโอนหน่วยความจำที่เหลืออยู่ทั้งหมดไปยังเครื่องปลายทางเสร็จสิ้น



ภาพที่ 3.8 แสดงความหมายของเวลาหยุดการทำงานและเวลารวมสำหรับการอพยพ

3.5 เบนซ์มาร์ก

ชุดโปรแกรมเบนมาร์กใช้ NPB ซึ่งเป็นชุดโปรแกรมขนาดเล็ก ซึ่งถูกออกแบบไว้เพื่อใช้ในการประเมินประสิทธิภาพของเครื่องซุปเปอร์คอมพิวเตอร์ที่ทำงานแบบขนาน โดยเบนมาร์กชุดนี้ถูกนำไปใช้กับโปรแกรมอย่างเช่น MPI และ OpenMP

NPB Benchmarks นี้มีหลายคลาส ตั้งแต่ คลาส S ซึ่งมีขนาดเล็กใช้ในการทดสอบที่ต้องการผลอย่างรวดเร็ว คลาส W ซึ่งก็มีขนาดเล็ก ส่วนคลาส A, B, C นั้นเป็นชุดทดสอบมาตรฐาน โดย คลาส B จะมีขนาดใหญ่กว่า คลาส A 4 เท่า และ คลาส C มีขนาดใหญ่กว่า คลาส B 4 เท่า เช่นกัน ส่วนคลาส D, E, F นั้นจะเป็นชุดทดสอบที่มีขนาดใหญ่ ขึ้นอีก 16 เท่าจากแต่ละคลาสก่อนหน้า ดังนั้น การวิจัยนี้ จึงเลือกคลาส C ซึ่งมีขนาดมาตรฐานมาใช้ในการทดลอง โดยในงานวิจัยเลือกใช้โปรแกรมที่เกี่ยวกับ พลศาสตร์ของไหลเชิงคำนวณ (Computational Fluid Dynamics) ซึ่งเป็นการใช้คอมพิวเตอร์สำหรับการวิเคราะห์ปัญหาทางด้านพลศาสตร์ของไหล (จอมภพ แววศักดิ์, 2549, 32-42) ดังต่อไปนี้

โปรแกรมที่เกี่ยวข้องกับแก่นของระบบปฏิบัติกลาง (kernels)ได้แก่

(1) IS – เป็นโปรแกรมเรียงลำดับจำนวนเต็ม ซึ่งมีการเข้าใช้หน่วยความจำแบบสุ่ม

(2) CG – เป็นโปรแกรมในการแก้ปัญหาเกี่ยวกับตัวเลขสำหรับระบบสมการเชิงเส้น ซึ่งมีการติดต่อและใช้งานหน่วยความไม่สม่ำเสมอนัก

(3) MG – เป็นโปรแกรมการคำนวณเกี่ยวโครงข่ายของรูปทรง ซึ่งมีการใช้หน่วยประมวลผลมาก

โปรแกรมรหัสเทียม (pseudo applications) ได้แก่

(1) BT – เป็นโปรแกรมช่วยคำนวณเมตริกซ์เกี่ยวกับบล็อก

(2) SP - เป็นโปรแกรมช่วยคำนวณเมตริกซ์เกี่ยวกับสเกลาร์

(3) LU – เป็นโปรแกรมช่วยคำนวณเมตริกซ์เกี่ยวกับขอบเขตบนและล่าง

โปรแกรมสำหรับการคำนวณแบบไม่มีโครงสร้าง การเคลื่อนย้ายข้อมูล อินพุธเอาท์พุธแบบขนาน ได้แก่

(1) UA – เป็นโปรแกรมแก้ปัญหาเกี่ยวกับรูปทรงที่ไม่มีโครงสร้าง ซึ่งมีการเรียกใช้หน่วยความจำเป็นครั้งคราว

บทที่ 4
ผลการวิจัยและอภิปรายผล

ในงานวิจัยนี้มีวัตถุประสงค์ที่จะเพิ่มประสิทธิภาพการอพยพแบบคงสถานะการทำงานของคอมพิวเตอร์เสมือนแบบเทรดด้วยการบีบอัด 2 ชั้น ชั้นแรกบีบอัดในระดับหน่วยความจำเพจด้วย Memory Server ชั้นที่สองบีบอัดแบบ Stream ด้วย LZ4 โดยนำเสนอการอพยพแบบคงสถานะการทำงานของคอมพิวเตอร์เสมือน 2 วิธี คือ TLM-XZ และ TLM-BZ

ในงานวิจัยนี้ทำการทดลองการอพยพแบบคงสถานะการทำงานแบบ TLM-XZ และ TLM-BZ ผ่านเครือข่ายที่มีแบนด์วิดท์ 1 Gbps และ 100 Mbps โดยวัดประสิทธิภาพจากเวลาที่หยุดการทำงานของคอมพิวเตอร์เสมือน (Downtime) และเวลารวมสำหรับการอพยพ (Total Migration Time) เปรียบเทียบกับการอพยพแบบคงสถานะการทำงานแบบ TLM และ TLM-Z เป็นเปอร์เซ็นต์

หลักการในการคำนวณประสิทธิภาพของ TLM-XZ เมื่อเปรียบเทียบกับ TLM คือ

(TLM – TLM-XZ) * 100 / TLM

หลักการในการคำนวณประสิทธิภาพของ TLM-XZ เมื่อเปรียบเทียบกับ TLM-Z คือ

(TLM-Z – TLM-XZ) * 100 / TLM-Z

การอพยพแบบใดใช้เวลาที่หยุดการทำงานและเวลารวมสำหรับการอพยพน้อยกว่าแสดงว่า มีประสิทธิภาพดีกว่า ตัวอย่างเช่น เมื่อวัดจากเวลาหยุดคอมพิวเตอร์ TLM-XZ มีประสิทธิภาพดีกว่า TLM 2% แปลว่า TLM-XZ ใช้เวลาในการหยุดคอมพิวเตอร์เสมือนน้อยกว่า TLM เป็นเวลา 2% และหากว่า ค่าที่ได้เป็นลบ เช่น -2% นั้นหมายความว่า TLM-XZ ใช้เวลาในการหยุดคอมพิวเตอร์เสมือนนานกว่า TLM 2%

4.1 การอพยพแบบคงสถานะการทำงานแบบ TLM-XZ

ทดสอบประสิทธิภาพของการอพยพแบบ TLM-XZ โดยมีสมมติฐานว่า

(1) การเข้ารหัสแบบเดลตาด้วยวิธี XOR ทำให้เวลาหยุดการทำงานและเวลารวมสำหรับการอพยพของ TLM-XZ น้อยกว่า TLM และ TLM-Z

(2) ประสิทธิภาพของการอพยพแบบคงสถานะการทำงานแบบ TLM-XZ บนระบบเครือข่ายที่มีแบนด์วิดท์ 100 Mbps ดีกว่า 1 Gbps

(3) การเข้ารหัสแบบเดลตาด้วยวิธี XOR ช่วยเพิ่มประสิทธิภาพการบีบอัดข้อมูลของ Z-Server

4.1.1 วัดประสิทธิภาพจากเวลาที่หยุดการทำงานของคอมพิวเตอร์เสมือน (Downtime) และเวลารวมสำหรับการอพยพ (Total Migration Time)

วัดประสิทธิภาพการอพยพแบบคงสถานะการทำงานแบบ TLM-XZ เปรียบเทียบกับ TLM และ TLM-Z สำหรับโปรแกรมทางวิทยาศาสตร์โดยใช้คลาส C 7 โปรแกรมได้แก่ BT.C, CG.C, IS.C, LU.C, MG.C, SP.C UA.C แสดงเวลาหยุดการทำงานและเวลารวมสำหรับการอพยพเป็นวินาที โดยค่าที่น้อยกว่า แสดงว่า มีประสิทธิภาพที่ดีกว่า และค่าที่แสดงนี้ได้จากการหาค่าเฉลี่ยจากผลการทดลอง 5 ครั้ง โดยแยกการทดลองออกตามแบนด์วิดท์

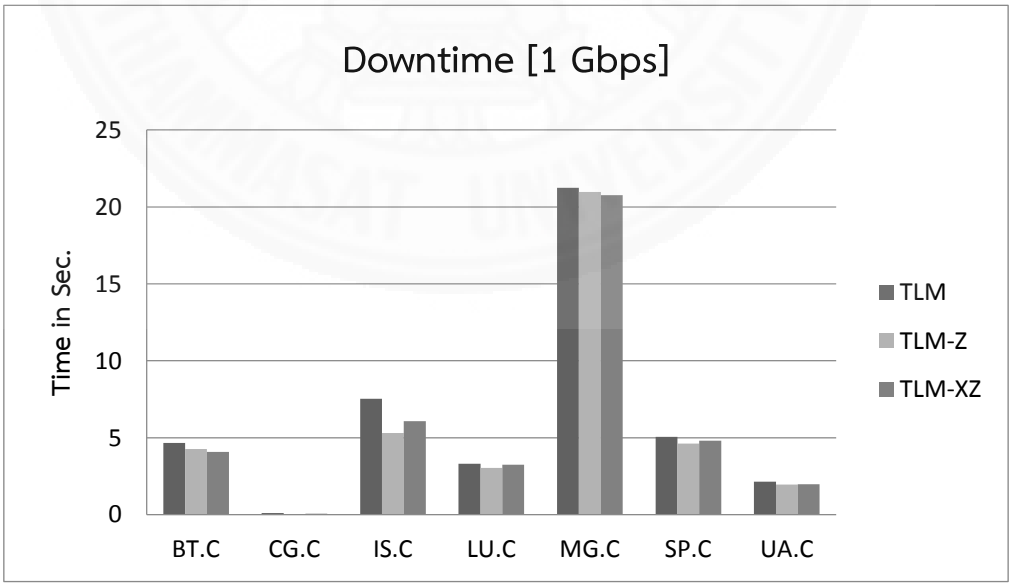
4.1.1.1 ทดลองบนเครือข่ายที่มีแบนด์วิดท์ 1 Gbps

ตารางที่ 4.1 เปรียบเทียบ Downtime และ Total Migration time ระหว่าง TLM, TLM-Z และ TLM-XZ บนเครือข่ายที่มีแบนด์วิดท์ 1 Gbps

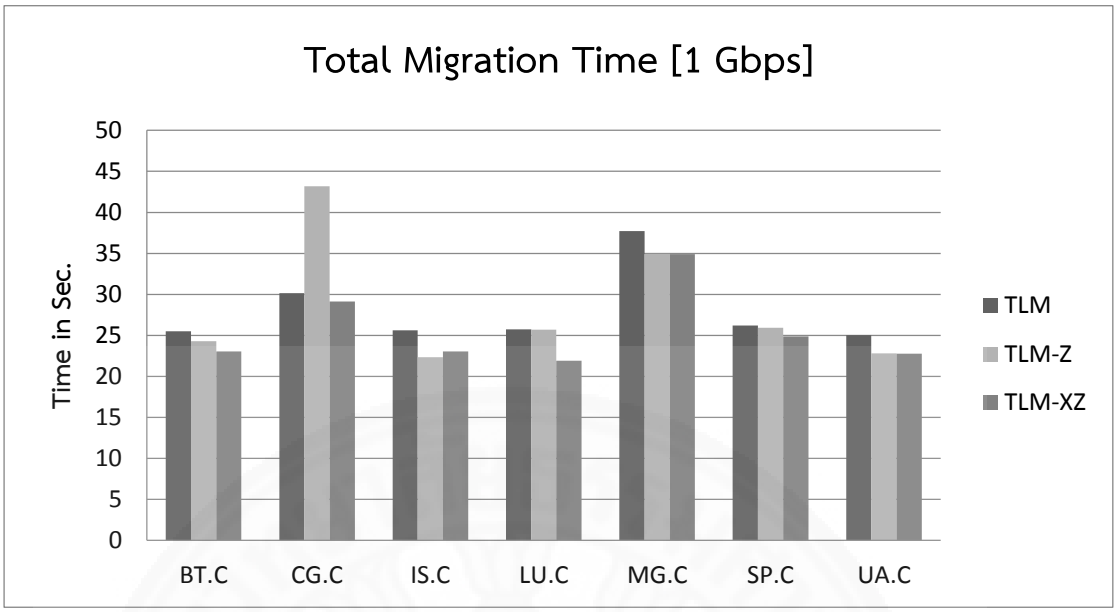
BenchMark	Downtime (Sec.)			Total Migrataion time (Sec.)		
	TLM	TLM-Z	TLM-XZ	TLM	TLM-Z	TLM-XZ
BT.C	4.66	4.26	4.09	25.51	24.28	23.05
CG.C	0.11	0.04	0.06	30.16	43.19	29.12
IS.C	7.54	5.30	6.08	25.64	22.36	23.04
LU.C	3.31	3.05	3.25	25.74	25.69	21.90
MG.C	21.24	20.98	20.76	37.71	34.95	34.89
SP.C	5.05	4.62	4.82	26.22	25.93	24.88
UA.C	2.14	1.96	1.97	25.04	22.81	22.78

ตารางที่ 4.2 เปรียบเทียบประสิทธิภาพของ TLM-Z และ TLM-XZ เมื่อเทียบกับ TLM และ TLM-XZ เมื่อเทียบกับ TLM-Z ในรูปแบบเปอร์เซ็นต์บนเครือข่ายที่มีแบนด์วิดท์ 1 Gbps

Bench Mark	Downtime			Migration		
	TLM-Z ต่อ TLM	TLM-XZ ต่อ TLM	TLM-XZ ต่อ TLM-Z	TLM-Z ต่อ TLM	TLM-XZ ต่อ TLM	TLM-XZ ต่อ TLM-Z
BT.C	9%	12%	4%	5%	10%	5%
CG.C	60%	48%	-32%	-43%	3%	33%
IS.C	30%	19%	-15%	13%	10%	-3%
LU.C	8%	2%	-6%	0%	15%	15%
MG.C	1%	2%	1%	7%	7%	0%
SP.C	9%	5%	-4%	1%	5%	4%
UA.C	9%	8%	-1%	9%	9%	0%



ภาพที่ 4.1 เปรียบเทียบ Downtime ระหว่าง TLM, TLM-Z และ TLM-XZ บนเครือข่ายที่มีแบนด์วิดท์ 1 Gbps



ภาพที่ 4.2 เปรียบเทียบ Total Migration Time ระหว่าง TLM, TLM-Z และ TLM-XZ บนเครือข่ายที่มีแบนด์วิดท์ 1 Gbps

การทดลองการอพยพแบบคงสถานะการทำงานของคอมพิวเตอร์เสมือนบนเครือข่ายที่มีแบนด์วิดท์ขนาด 1 Gbps (จากตารางที่ 4.1-ตารางที่ 4.2 และภาพที่ 4.1-ภาพที่ 4.2) โดยวัดประสิทธิภาพจาก Downtime พบว่า TLM-XZ สามารถทำให้ประสิทธิภาพของการอพยพสำหรับโปรแกรม BT.C และ MG.C ดีขึ้นเพียงเล็กน้อยเท่านั้น โดยดีกว่า TLM 12% และ 2% ตามลำดับ และดีกว่า TLM-Z 4% และ 1% ตามลำดับ อย่างไรก็ตาม TLM-XZ มีประสิทธิภาพแย่กว่า TLM-Z มาก สำหรับโปรแกรม CG.C, IS.C, LU.C, SP.C โดยที่ CG.C ดีกว่า TLM 48%, 19%, 2% และ 5% ตามลำดับ แต่แย่กว่า TLM-Z 32%, 15%, 6% และ 4% TLM-XZ มีประสิทธิภาพแย่กว่า TLM-Z เพียงเล็กน้อยสำหรับโปรแกรม UA.C ซึ่งดีกว่า TLM 8% แต่แย่กว่า TLM-Z เพียง 1% เท่านั้น

วัดประสิทธิภาพจาก Total Migration Time พบว่า TLM-XZ ทำให้ประสิทธิภาพของการอพยพสำหรับโปรแกรม BT.C, CG.C, LU.C และ SP.C ดีขึ้นกว่า TLM และ TLM-Z และมีประสิทธิภาพดีกว่า TLM แต่แย่กว่า TLM-Z สำหรับโปรแกรม IS.C ในขณะที่มีประสิทธิภาพเท่ากันกับ TLM-Z สำหรับ MG.C และ UA.C อย่างไรก็ตาม TLM-Z สำหรับโปรแกรม CG.C กลับมี Total Migration Time ที่แย่กว่าทั้ง TLM และ TLM-XZ มาก แย่กว่า TLM ถึง 43 % เลยทีเดียว

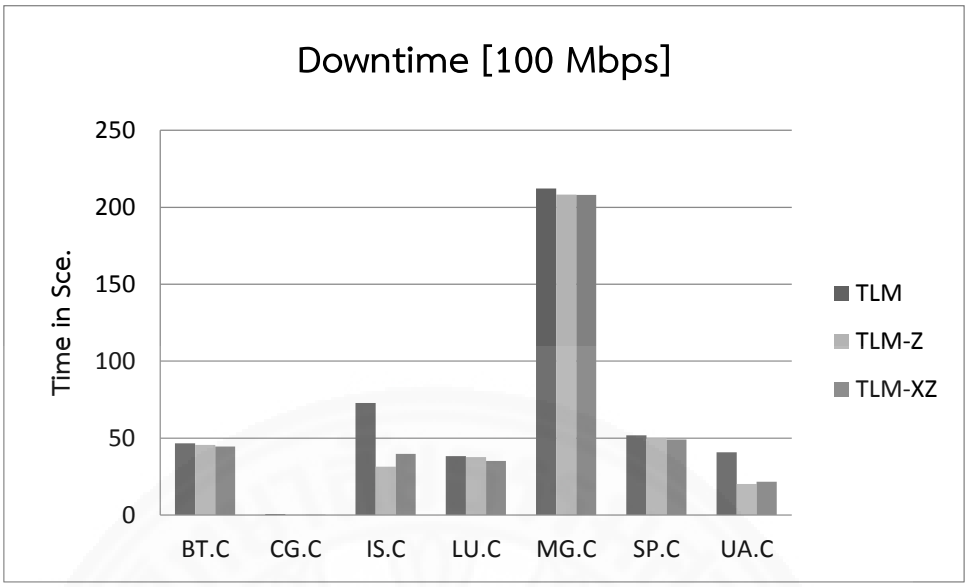
4.1.1.2 ทดลองบนเครือข่ายที่มีแบนด์วิดท์ 100 Mbps

ตารางที่ 4.3 เปรียบเทียบ Downtime และ Total Migration time ระหว่าง TLM, TLM-Z และ TLM-XZ บนเครือข่ายที่มีแบนด์วิดท์ 100 Mbps

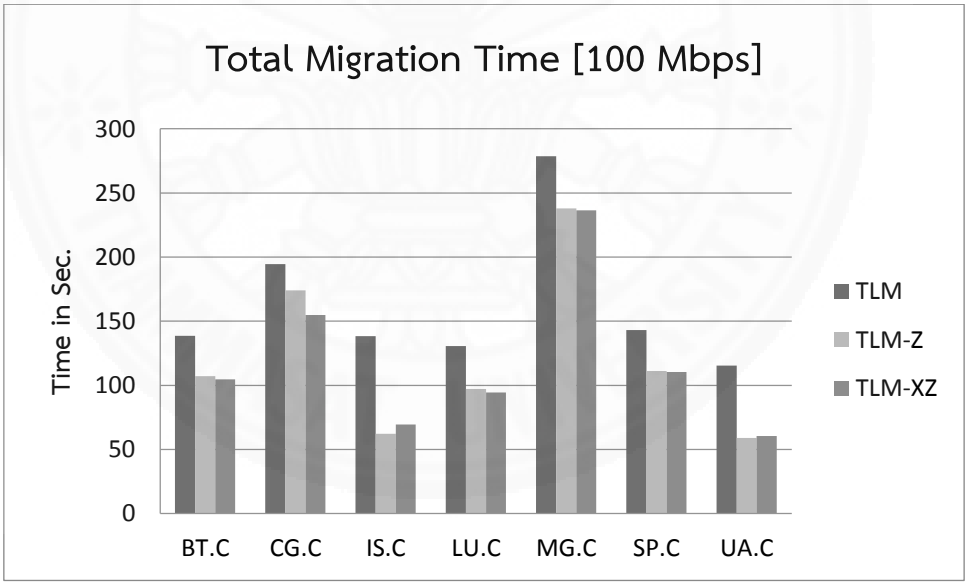
BenchMark	Downtime (Sec.)			Total Migrataion time (Sec.)		
	TLM	TLM-Z	TLM-XZ	TLM	TLM-Z	TLM-XZ
BT.C	46.67	45.59	44.55	138.65	107.25	104.67
CG.C	0.62	0.29	0.19	194.58	174.15	154.71
IS.C	72.83	31.52	39.77	138.28	62.11	69.37
LU.C	38.40	37.79	35.21	130.59	97.26	94.39
MG.C	212.31	208.21	208.14	278.68	238.03	236.43
SP.C	51.88	49.94	49.06	143.05	111.16	110.38
UA.C	40.75	20.15	21.73	115.31	58.87	60.46

ตารางที่ 4.4 เปรียบเทียบประสิทธิภาพของ TLM-Z และ TLM-XZ เมื่อเทียบกับ TLM และ TLM-XZ เมื่อเทียบกับ TLM-Z ในรูปแบบเปอร์เซ็นต์บนเครือข่ายที่มีแบนด์วิดท์ 100 Mbps

Bench Mark	Downtime			Migration		
	TLM-Z ต่อ TLM	TLM-XZ ต่อ TLM	TLM-XZ ต่อ TLM-Z	TLM-Z ต่อ TLM	TLM-XZ ต่อ TLM	TLM-XZ ต่อ TLM-Z
BT.C	2%	5%	2%	23%	25%	2%
CG.C	54%	69%	32%	10%	20%	11%
IS.C	57%	45%	-26%	55%	50%	-12%
LU.C	2%	8%	7%	26%	28%	3%
MG.C	2%	2%	0%	15%	15%	1%
SP.C	4%	5%	2%	22%	23%	1%
UA.C	51%	47%	-8%	49%	48%	-3%



ภาพที่ 4.3 เปรียบเทียบ Downtime ระหว่าง TLM, TLM-Z และ TLM-XZ บนเครือข่ายที่มีแบนด์วิดท์ 100 Mbps



ภาพที่ 4.4 เปรียบเทียบ Total Migration Time ระหว่าง TLM, TLM-Z และ TLM-XZ บนเครือข่ายที่มีแบนด์วิดท์ 100 Mbps

การทดลองการอพยพแบบคงสถานะการทำงานของคอมพิวเตอร์เสมือนบนเครือข่ายที่มีแบนด์วิดท์ขนาด 100 Mbps (จากตารางที่ 4.3-ตารางที่ 4.4 และภาพที่ 4.3-ภาพที่ 4.4) โดยวัดประสิทธิภาพจาก Downtime พบว่า TLM-XZ

สามารถทำให้ประสิทธิภาพของการอพยพสำหรับโปรแกรม BT.C, CG.C LU.C, และ SP.C ดีขึ้นกว่า TLM 5%, 69%, 8% และ 5% ตามลำดับ ดีกว่า TLM-Z 2%, 32%, 7% และ 2% ตามลำดับ มีประสิทธิภาพดีกว่า TLM แต่แย่กว่า TLM-Z สำหรับโปรแกรม IS.C และ UA. โดยดีกว่า TLM 57% และ 51% ตามลำดับ แต่แย่กว่า TLM-Z 26% และ 8% ตามลำดับ มีประสิทธิภาพดีกว่า TLM และเท่ากันกับ TLM-Z สำหรับโปรแกรม MG.C

วัดประสิทธิภาพจาก Total Migration Time พบว่า TLM-XZ ทำให้ประสิทธิภาพของการอพยพสำหรับโปรแกรม BT.C, CG.C, LU.C และ SP.C ดีขึ้นกว่า TLM และ TLM-Z และมีประสิทธิภาพดีกว่า TLM แต่แย่กว่า TLM-Z สำหรับโปรแกรม IS.C และ UA.C ในขณะที่มี TLM-Z ประสิทธิภาพดีกว่า TLM แต่เท่ากันกับ TLM-Z สำหรับ MG.C

การเข้ารหัสแบบเดลตาด้วยวิธี XOR ไม่สามารถทำให้ประสิทธิภาพการอพยพแบบคงสถานะการทำงานของคอมพิวเตอร์เสมือนที่ใช้หน่วยประมวลและหน่วยความจำมากผ่านเครือข่ายแบนด์วิดท์ 1 Gbps ดีขึ้น และสามารถทำให้ประสิทธิภาพดีขึ้นเพียงเล็กน้อยเมื่ออพยพผ่านเครือข่ายแบนด์วิดท์ 100 Mbps สำหรับบางโปรแกรม โดยมีประสิทธิภาพเพิ่มขึ้นมากที่สุดสำหรับโปรแกรม LU.C เพิ่มขึ้น 7% อย่างไรก็ตาม กลับทำให้ประสิทธิภาพแย่ลงมากสำหรับโปรแกรม IS.C ลดลงถึง 26% เมื่อเทียบกับ TLM-Z

4.1.2 วัดประสิทธิภาพการบีบอัดข้อมูลของ LZ4

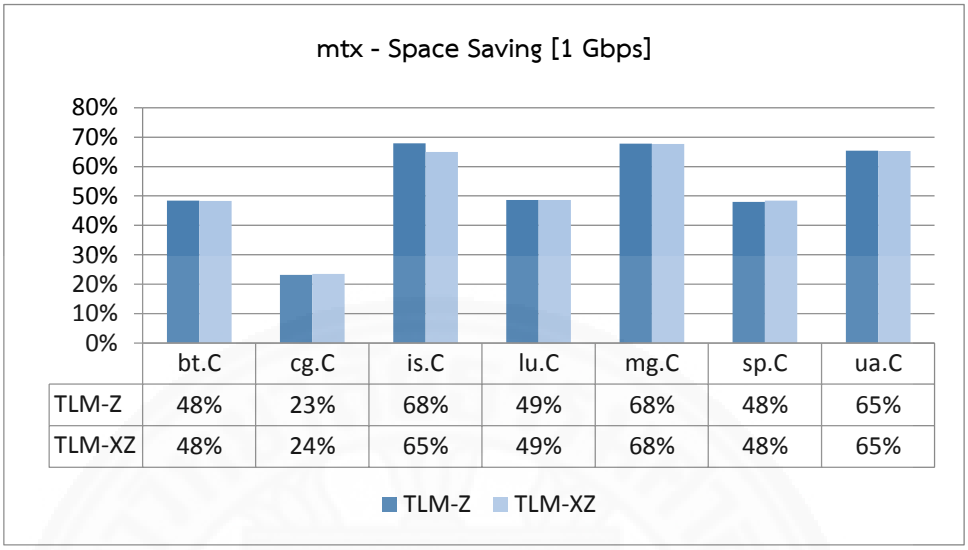
วัดความสามารถในการบีบอัดข้อมูลของ LZ4 เพื่อประเมินว่า การเข้ารหัสแบบเดลตาด้วยวิธี XOR ทำให้ประสิทธิการบีบอัดข้อมูลของ LZ4 ดีขึ้นหรือไม่ โดยวัดประสิทธิภาพการบีบอัดข้อมูลของ LZ4 สำหรับข้อมูลหน่วยความจำที่เทรด mtx, dtx ถ่ายโอนไปยังเครื่องแม่ข่าย Z-Server และ เทรด io ถ่ายโอนในช่วงที่คอมพิวเตอร์เสมือนหยุดการทำงาน (S3) เปรียบเทียบระหว่าง TLM และ TLM-Z และแบ่งการวัดประสิทธิภาพตามแบนด์วิดท์

วัดประสิทธิภาพด้วย Space Saving (Data compression ratio, Wikipedia)

$$\text{Space Saving} = 1 - \frac{\text{Compressed Size}}{\text{Uncompressed Size}}$$

โดยค่าที่มากกว่า แสดงว่า LZ4 สามารถบีบอัดข้อมูลได้ดีกว่าและสามารถช่วยลดปริมาณข้อมูลที่ส่งผ่านแบนด์วิดท์ได้มากกว่า

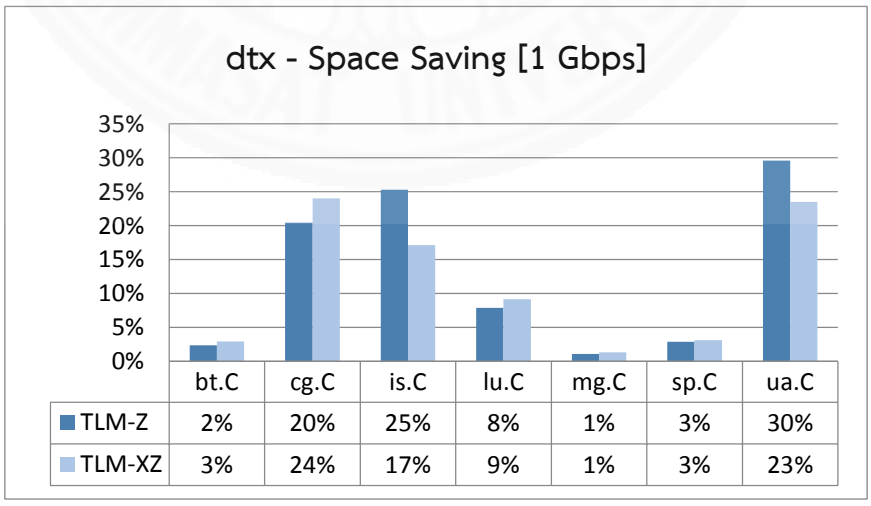
4.1.2.1 ทดลองบนเครือข่ายที่มีแบนด์วิดท์ 1 Gbps



	bt.C	cg.C	is.C	lu.C	mg.C	sp.C	ua.C
TLM-Z	48%	23%	68%	49%	68%	48%	65%
TLM-XZ	48%	24%	65%	49%	68%	48%	65%

ภาพที่ 4.5 เปรียบเทียบประสิทธิภาพการบีบอัดข้อมูลของ LZ4 ของเทรด mtx สำหรับการอพยพแบบคงสถานะการทำงานแบบ TLM-Z และ TLM-XZ ผ่านเครือข่ายแบนด์วิดท์ 1 Gbps

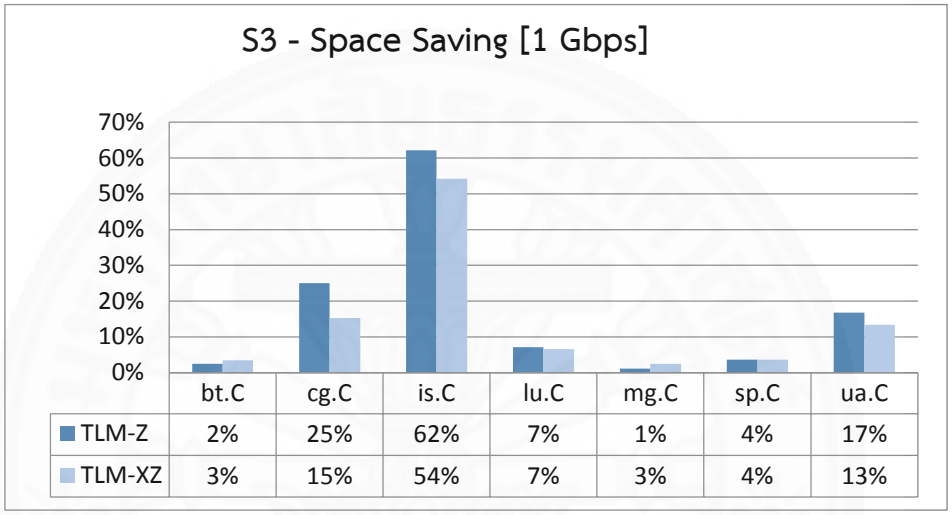
จากภาพที่ 4.5 พบว่า ประสิทธิภาพของการบีบอัดของ LZ4 ต่อหน่วยความจำที่เทรด mtx ถ่ายโอนผ่านเครือข่ายแบนด์วิดท์ 1 Gbps ของการอพยพแบบคงสถานะการทำงานทั้งสองแบบแทบไม่ต่างกันเลย มีเพียง IS.C เท่านั้น ที่มีประสิทธิภาพลดลงในการอพยพแบบคงสถานะการทำงานแบบ TLM-XZ โดยลดลงไป 3%



	bt.C	cg.C	is.C	lu.C	mg.C	sp.C	ua.C
TLM-Z	2%	20%	25%	8%	1%	3%	30%
TLM-XZ	3%	24%	17%	9%	1%	3%	23%

ภาพที่ 4.6 เปรียบเทียบประสิทธิภาพการบีบอัดข้อมูลของ LZ4 ของเทรด dtx สำหรับการอพยพแบบคงสถานะการทำงานแบบ TLM-Z และ TLM-XZ ผ่านเครือข่ายแบนด์วิดท์ 1 Gbps

จากภาพที่ 4.6 พบว่า ประสิทธิภาพของการบีบอัดข้อมูลของ LZ4 ต่อหน่วยความจำที่เทรด dtx ถ่ายโอนผ่านเครือข่ายแบนด์วิดท์ 1 Gbps ของการอพยพแบบคงสถานะการทำงานทั้งสองแบบมีประสิทธิภาพค่อนข้างสูงสำหรับบางโปรแกรม ซึ่งได้แก่ CG.C, IS.C และ UA.C และพบว่า ประสิทธิภาพของการบีบอัดสำหรับการอพยพแบบคงสถานะการทำงานแบบ TLM-XZ ดีกว่า TLM-Z สำหรับโปรแกรม BT.C, CG.C และ LU.C แต่ประสิทธิภาพของการบีบอัดสำหรับ TLM-XZ แยกกว่า TLM-Z สำหรับโปรแกรม IS.C และ UA.C

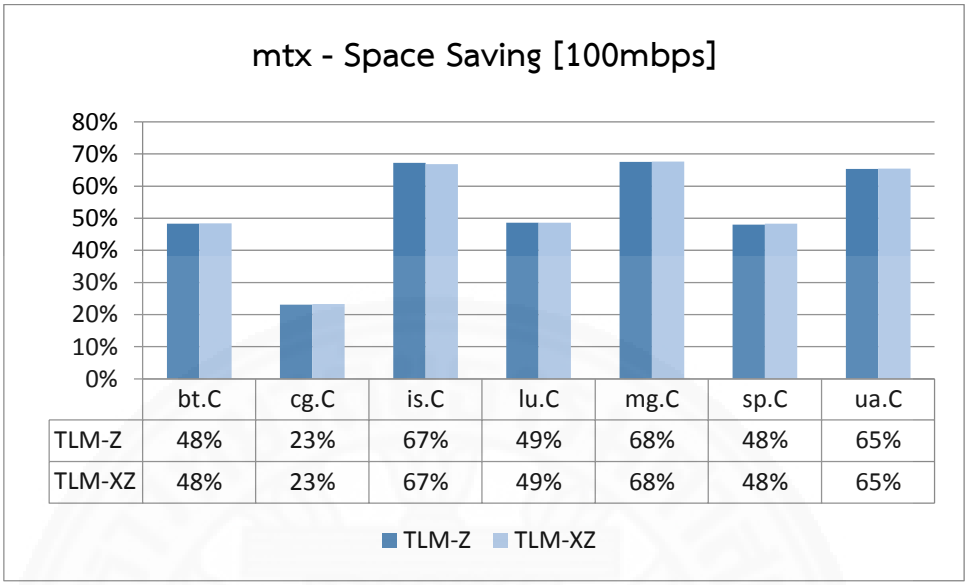


	bt.C	cg.C	is.C	lu.C	mg.C	sp.C	ua.C
TLM-Z	2%	25%	62%	7%	1%	4%	17%
TLM-XZ	3%	15%	54%	7%	3%	4%	13%

ภาพที่ 4.7 เปรียบเทียบประสิทธิภาพการบีบอัดข้อมูลของ LZ4 ของช่วง S3 สำหรับการอพยพแบบคงสถานะการทำงานแบบ TLM-Z และ TLM-XZ ผ่านเครือข่ายแบนด์วิดท์ 1 Gbps

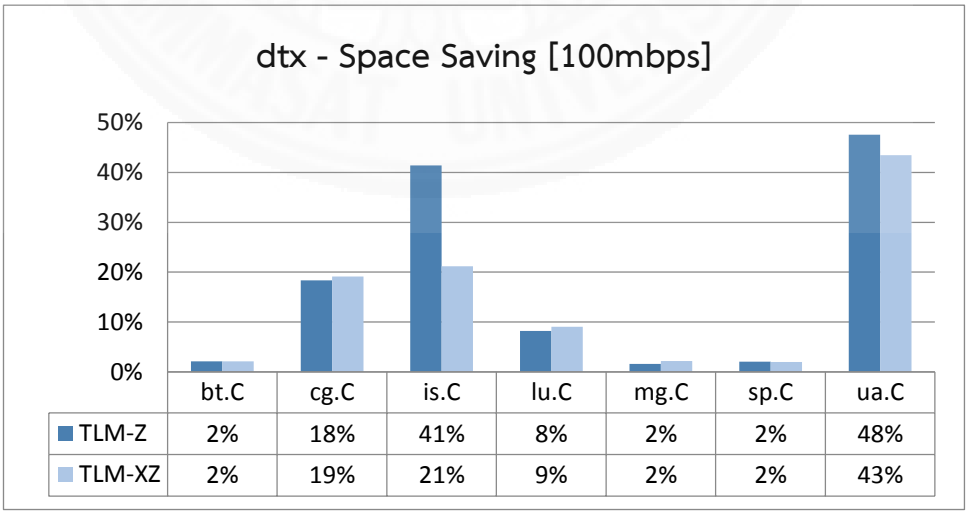
จากภาพที่ 4.7 พบว่า ประสิทธิภาพการบีดอัดข้อมูลของ LZ4 ต่อหน่วยความจำที่เทรด io ถ่ายโอนผ่านเครือข่ายแบนด์วิดท์ 1 Gbps ในช่วงที่คอมพิวเตอร์เสมือนหยุดการทำงานสำหรับการอพยพแบบคงสถานะการทำงานทั้งสองแบบมีประสิทธิภาพค่อนข้างสูงมากสำหรับโปรแกรม IS.C และมีประสิทธิภาพค่อนข้างสูงสำหรับโปรแกรม CG.C และ UA.C และพบว่าประสิทธิภาพของการบีบอัดสำหรับการอพยพแบบคงสถานะการทำงานแบบ TLM-XZ ดีกว่า TLM-Z สำหรับโปรแกรม BT.C และ MG.C แต่ประสิทธิภาพของการบีบอัดสำหรับ TLM-XZ แย่กว่า TLM-Z สำหรับโปรแกรม CG.C, IS.C และ UA.C

4.1.2.2 ทดลองบนเครือข่ายที่มีแบนด์วิดท์ 100 Mbps



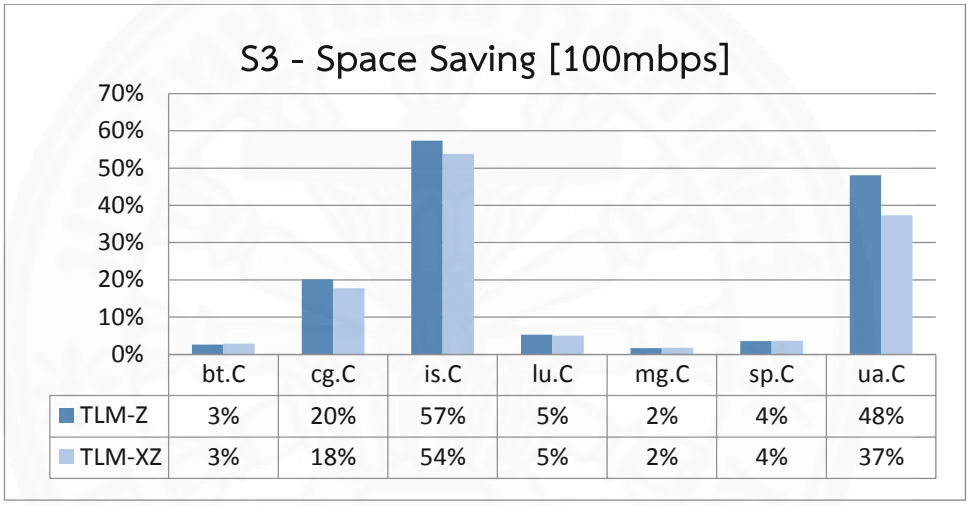
ภาพที่ 4.8 เปรียบเทียบประสิทธิภาพการบีบอัดข้อมูลของ LZ4 ของเทรด mtx สำหรับการอพยพแบบคงสถานะการทำงานแบบ TLM-Z และ TLM-XZ ผ่านเครือข่ายแบนด์วิดท์ 100 Mbps

จากภาพที่ 4.8 พบว่า ประสิทธิภาพของการบีบอัดของ LZ4 ต่อหน่วยความจำที่เทรด mtx ถ่ายโอนผ่านเครือข่ายแบนด์วิดท์ 100 Mbps ของการอพยพแบบคงสถานะการทำงานทั้งสองแบบไม่ต่างกันเลย



ภาพที่ 4.9 เปรียบเทียบประสิทธิภาพการบีบอัดข้อมูลของ LZ4 ของเทรด dtx สำหรับการอพยพแบบคงสถานะการทำงานแบบ TLM-Z และ TLM-XZ ผ่านเครือข่ายแบนด์วิดท์ 100 Mbps

จากภาพที่ 4.9 พบว่า ประสิทธิภาพของการบีบอัดข้อมูลของ LZ4 ต่อหน่วยความจำที่เทรด dtx ถ่ายโอนผ่านเครือข่ายแบนด์วิดท์ 100 Mbps ของการอพยพแบบคงสถานะการทำงานทั้งสองแบบมีประสิทธิภาพเท่ากันและสามารถบีบอัดได้เพียงเล็กน้อยสำหรับโปรแกรม BT.C, MG.C และ SP.C และการบีบอัดสำหรับการอพยพทั้งสองแบบมีประสิทธิภาพค่อนข้างสูงสำหรับโปรแกรม CG.C และมีประสิทธิภาพสูงมากสำหรับโปรแกรม UA.C แต่ประสิทธิภาพการบีบอัดของ LZ4 สำหรับการอพยพแบบ TLM-Z สูงกว่าแบบ TLM-XZ และประสิทธิภาพการบีบอัดของ LZ4 ในการอพยพแบบ TLM-Z มีประสิทธิภาพสูงมากสำหรับโปรแกรม IS.C สูงถึง 41% แต่กลับมีประสิทธิภาพเพียง 21% สำหรับการอพยพแบบ TLM-XZ



	bt.C	cg.C	is.C	lu.C	mg.C	sp.C	ua.C
TLM-Z	3%	20%	57%	5%	2%	4%	48%
TLM-XZ	3%	18%	54%	5%	2%	4%	37%

ภาพที่ 4.10 เปรียบเทียบประสิทธิภาพการบีบอัดข้อมูลของ LZ4 ของช่วง S3 สำหรับการอพยพแบบคงสถานะการทำงานแบบ TLM-Z และ TLM-XZ ผ่านเครือข่ายแบนด์วิดท์ 100 Mbps

จากภาพที่ 4.10 พบว่า ประสิทธิภาพของการบีบอัดข้อมูลของ LZ4 ต่อหน่วยความจำที่เทรด io ถ่ายโอนผ่านเครือข่ายแบนด์วิดท์ 100 Mbps ในช่วงที่คอมพิวเตอร์เสมือนหยุดการทำงานของการอพยพแบบคงสถานะการทำงานทั้งสองแบบมีประสิทธิภาพเท่ากันและสามารถบีบอัดได้เพียงเล็กน้อยสำหรับโปรแกรม BT.C, LU.C, MG.C และ SP.C และการบีบอัดสำหรับการอพยพทั้งสองแบบมีประสิทธิภาพสูงมากสำหรับโปรแกรม IS.C และ UA.C และมีประสิทธิภาพค่อนข้างสูงสำหรับโปรแกรม CG.C อย่างไรก็ตาม ประสิทธิภาพการบีบอัดสำหรับการอพยพแบบ TLM-Z สูงกว่าแบบ TLM-XZ โดยสูงกว่าสำหรับ CG.C 2%, IS.C 3% และ สูงกว่า UA.C ถึง 11%

เมื่อเปรียบเทียบประสิทธิภาพการบีบอัดข้อมูลของ LZ4 สำหรับการอพยพแบบคงสถานะการทำงานผ่านเครือข่ายแบนด์วิดท์ 1 Gbps และ 100 Mbps นั้น พบว่า ประสิทธิภาพการ

บีบอัดต่อหน่วยความจำที่เทรด mtx ถ่ายโอนนั้นแทบไม่ต่างกันเลย แต่ประสิทธิภาพการบีบอัดต่อหน่วยความจำที่เทรด dtx ถ่ายโอน กลับสูงขึ้นมากเมื่อเครือข่ายมีแบนด์วิดท์ลดลงสำหรับโปรแกรม IS.C และ UA.C และที่เทรด io ถ่ายโอนในช่วงหยุดคอมพิวเตอร์เสมือนสูงขึ้นมากสำหรับโปรแกรม UA.C

จากการทดลองที่คาดว่าการเข้ารหัสแบบเดลตาด้วยวิธี XOR จะทำให้ LZ4 มีประสิทธิภาพในการบีบอัดข้อมูลมากขึ้น พบว่า การเข้ารหัสแบบเดลตาด้วยวิธี XOR ไม่ได้ช่วยให้ประสิทธิภาพของการบีบอัดสูงขึ้น แต่หากทำให้ประสิทธิภาพการบีบอัดข้อมูลของ LZ4 ลดลงสำหรับโปรแกรม IS.C และ UA.C

4.1.3 การวิเคราะห์ลักษณะของหน่วยความจำ

จากผลการทดลองข้างต้นนั้น พบว่า การเข้ารหัสแบบเดลตาด้วยวิธี XOR ไม่ได้ช่วยเพิ่มประสิทธิภาพของการอพยพแบบคงสถานะการทำงานแบบ TLM-Z อย่างที่ได้ตั้งสมมติฐานไว้ และยังทำให้ประสิทธิภาพการบีบอัดข้อมูลของ LZ4 ลดลงสำหรับโปรแกรม IS.C และ UA.C ดังนั้น ผู้วิจัยจึงทำการวิเคราะห์ข้อมูลหน่วยความจำเพิ่มเติม เพื่อหาคำตอบว่าเพราะสาเหตุใดการเข้ารหัสด้วยวิธี XOR ทำให้ประสิทธิภาพของการอพยพแบบคงสถานะการทำงานของคอมพิวเตอร์เสมือนที่รันโปรแกรม NPB ลดลงและหาแนวทางในการเพิ่มประสิทธิภาพ โดยวิเคราะห์แยกตามเครือข่าย

4.1.3.1 วิเคราะห์ค่าศูนย์ที่เรียงต่อกัน (Zero Sequence)

การเข้ารหัสแบบเดลตาด้วยวิธี XOR นั้น หากว่า ข้อมูลเหมือนกัน จะได้ค่าที่เป็น 0 ดังนั้นการวิเคราะห์ค่าที่เป็น 0 สามารถบอกได้ว่า ข้อมูลมีความเหมือนกันมากเพียงใดในแต่ละโปรแกรมและมีข้อมูลที่เหมือนกันอยู่ติดกันมากเพียงใด โดยการนับค่าที่เป็น 0 ที่อยู่ติดกัน 1 byte, 2 bytes, 4 bytes และ 8 bytes โดยที่ Total XOR Bytes คือ จำนวนไบต์ทั้งหมดที่ผ่านการเข้ารหัสแบบเดลตาด้วยวิธี XOR และคำนวณหาเปอร์เซ็นต์ของค่าที่เป็น 0 ต่อจำนวนไบต์ทั้งหมดที่ผ่านการทำ XOR โดยใช้หลักการคำนวณค่าเปอร์เซ็นต์ดังนี้

1 byte : จำนวนครั้งที่พบค่า 0 ติดกัน 1 byte / Total XOR Bytes

2 bytes: จำนวนครั้งที่พบค่า 0 ติดกัน 2 byte / Total XOR 2 Bytes

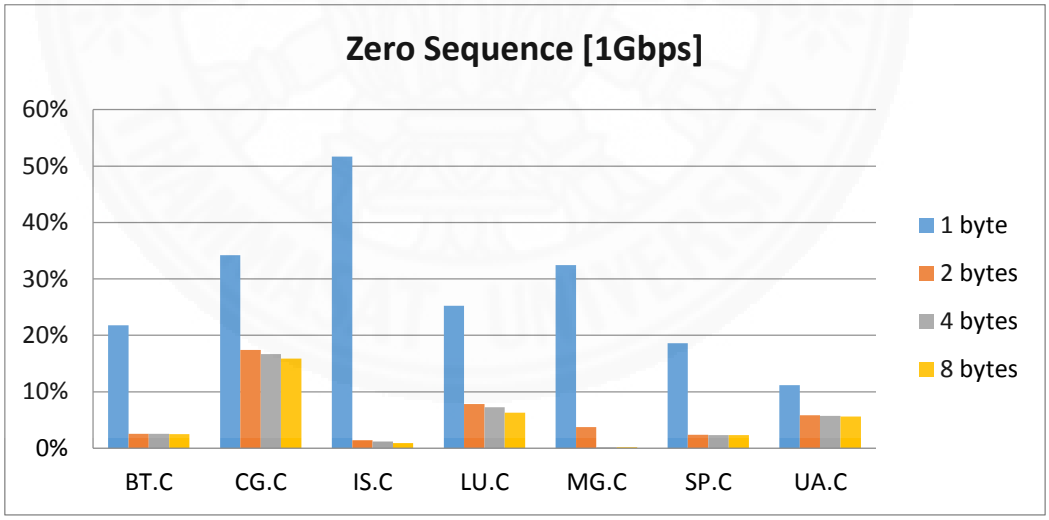
4 bytes: จำนวนครั้งที่พบค่า 0 ติดกัน 4 byte / Total XOR 4 Bytes

8 bytes: จำนวนครั้งที่พบค่า 0 ติดกัน 4 byte / Total XOR 8 Bytes

(1) การอพยพแบบคงสถานะการทำงานบนเครือข่าย 1 Gbps

ตารางที่ 4.5 แสดงจำนวนครั้งในการพบค่า 0 ที่อยู่ติดกัน 1 byte, 2 bytes, 4 bytes และ 8 bytes สำหรับการอพยพบนเครือข่าย 1 Gbps

Benchmark	1 byte	2 bytes	4 bytes	8 bytes	Total XOR Bytes
BT.C	220,024,655	12,962,967	6,423,070	3,181,633	1,011,212,288
CG.C	52,833,257	13,458,889	6,445,550	3,063,562	154,419,200
IS.C	461,407,393	6,338,035	2,613,307	1,038,459	893,145,088
LU.C	259,477,527	40,149,213	18,685,407	8,103,336	1,027,567,616
MG.C	398,480,338	22,872,362	604,420	299,926	1,227,177,984
SP.C	213,293,833	13,503,439	6,693,452	3,315,598	1,146,773,504
UA.C	78,907,706	20,546,445	10,124,029	4,953,765	705,232,896



ภาพที่ 4.11 อัตราส่วนคิดเป็น % ในการพบค่า 0 ที่อยู่ติดกัน 1 byte, 2 bytes, 4 bytes และ 8 bytes สำหรับการอพยพบนเครือข่าย 1 Gbps

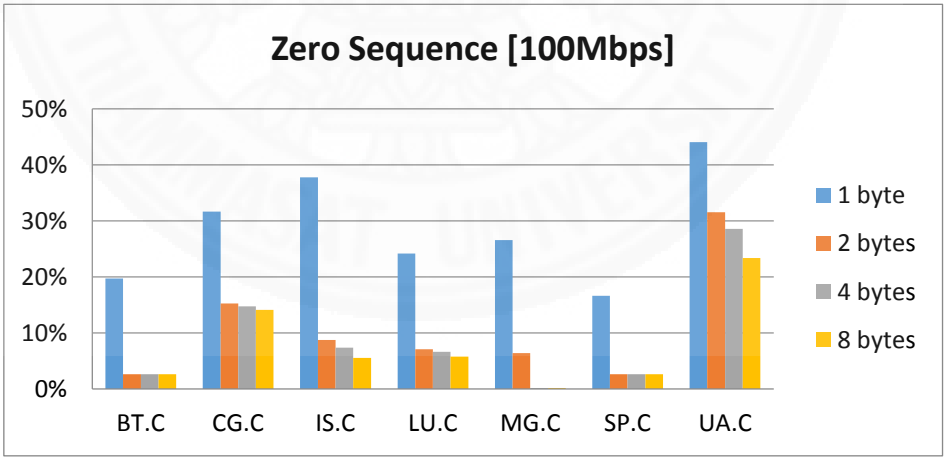
จากตารางที่ 4.5 แสดงจำนวนครั้งในการพบค่า 0 ที่อยู่ติดกัน 1 byte, 2 bytes, 4 bytes และ 8 bytes สำหรับการอพยพบนเครือข่าย 1 Gbps และ ภาพที่ 4.11 แสดงให้เห็นว่าโปรแกรม IS.C มีหน่วยความจำที่มีค่า 0 อยู่ติดกัน 1 byte มากที่สุด เกินกว่า 50% รองลงมา

คือ CG.C และ MG.C แต่ IS.C กลับมีค่า 0 ที่อยู่ติดกัน 2 bytes, 4 bytes และ 8 bytes น้อยมาก โดยมี 0 ติดกันน้อยที่สุด และ MG.C กลับมีค่า 0 ที่อยู่ติดกัน 3 bytes และ 4 bytes น้อยที่สุด

(2) การอพยพแบบคงสถานะการทำงานบนเครือข่าย 100 Mbps

ตารางที่ 4.6 แสดงจำนวนครั้งในการพบค่า 0 ที่อยู่ติดกัน 1 byte, 2 bytes, 4 bytes และ 8 bytes สำหรับการอพยพบนเครือข่าย 100 Mbps

Benchmark	1 byte	2 bytes	4 bytes	8 bytes	Total XOR Bytes
BT.C	82,961,295	5,542,956	2,768,685	1,382,847	420,909,056
CG.C	280,351,364	67,513,422	32,609,863	15,612,224	885,084,160
IS.C	42,739,314	4,944,429	2,085,292	783,275	113,147,904
LU.C	112,794,933	16,484,042	7,707,761	3,363,062	467,136,512
MG.C	61,809,412	7,455,444	103,136	49,573	232,697,856
SP.C	70,601,508	5,511,106	2,752,288	1,374,436	425,111,552
UA.C	224,573,616	80,382,078	36,396,672	14,901,085	509,693,952



ภาพที่ 4.12 อัตราส่วนคิดเป็น % ในการพบค่า 0 ที่อยู่ติดกัน 1 byte, 2 bytes, 4 bytes และ 8 bytes สำหรับการอพยพบนเครือข่าย 100 Mbps

จากตารางที่ 4.6 และ ภาพที่ 4.12 แสดงให้เห็นว่าโปรแกรม UA.C มีหน่วยความจำที่มีค่า 0 อยู่ติดกัน 1 byte, 2 bytes, 4 bytes และ 8 bytes จำนวนมาก ในขณะที่

โปรแกรม IS.C มีค่า 0 อยู่ติดกัน 1 byte จำนวนมาก แต่กลับมีค่า 0 อยู่ติดกัน 2 bytes, 4 bytes และ 8 bytes ไม่มากนัก

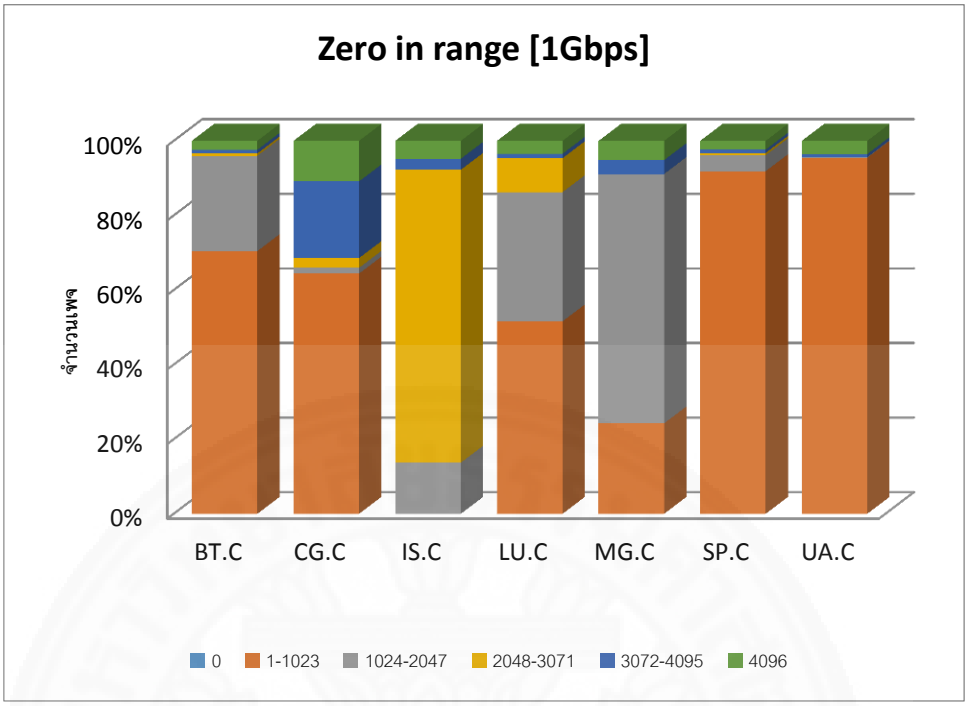
4.1.3.2 วิเคราะห์ค่าศูนย์ในแต่ละเพจ

วิเคราะห์หน่วยความจำเพจว่า ในแต่ละเพจนั้นมีข้อมูลที่เหมือนกันมากน้อยเพียงใด จากการนับจำนวนครั้งที่พบ 0 ที่ได้จากการทำ XOR ในแต่ละเพจว่ามีจำนวนเท่าไหร่ และจัดแสดงจำนวนเพจในลักษณะเป็นช่วงข้อมูล

(1) การอพยพแบบคงสถานะการทำงานบนเครือข่าย 1 Gbps

ตารางที่ 4.7 จำนวนเพจที่มีค่า 0 จากการทำการ XOR อยู่ในช่วงต่างๆ สำหรับการอพยพบนเครือข่าย 1 Gbps

Benchmark	0	1-1023	1024-2047	2048-3071	3072-4095	4096	Total XOR Pages
BT.C	0	173,901	63,101	1,541	2,397	5,938	246,878
CG.C	0	24,281	587	985	7,749	4,098	37,700
IS.C	0	0	29,898	171,233	6,471	10,451	218,053
LU.C	0	129,451	86,914	22,859	2,950	8,697	250,871
MG.C	0	72,665	200,279	15	11,185	15,460	299,604
SP.C	0	257,237	11,972	1,729	2,423	6,613	279,974
UA.C	0	164,365	232	25	1,437	6,117	172,176



ภาพที่ 4.13 สัดส่วนจำนวนเพจที่มีค่า 0 จากการทำการ XOR ที่อยู่ในช่วงต่างๆ ต่อจำนวนเพจทั้งหมดที่ทำ XOR สำหรับการอพยพบนเครือข่าย 1 Gbps

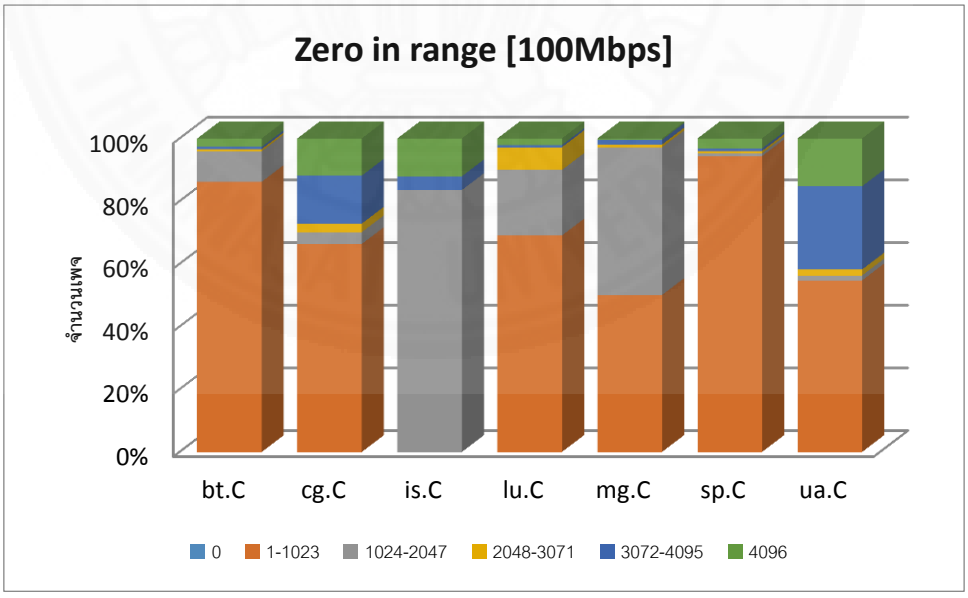
จากตารางที่ 4.7 และ ภาพที่ 4.13 พบว่า มีเพจที่ผ่านการทำ XOR แล้วได้ค่า 0 เท่ากับ 4096 นั่นแปลว่า หน่วยความจำเพจเวอร์ชันก่อนหน้ากับเวอร์ชันปัจจุบันเหมือนกัน ไม่มีความแตกต่างกันเลย โดยที่ MG.C มีหน่วยความจำเพจที่ไม่มีการเปลี่ยนแปลงเลยมากที่สุด รองลงมาคือ IS.C, LU.C, SP.C, UA.C, BT.C และ CG.C ตามลำดับ เมื่อเพจไม่มีการเปลี่ยนแปลงก็ไม่จำเป็นที่จะต้องถ่ายโอนเพจเหล่านั้นซ้ำอีก

นอกจากนี้ ยังพบว่า โปรแกรมส่วนใหญ่มีข้อมูลที่เหมือนกันน้อยซึ่งจะอยู่ในช่วง 1-1023 ในขณะที่โปรแกรม MG.C มีข้อมูลหน่วยความจำที่เหมือนกันส่วนใหญ่อยู่ในช่วง 1024-2047 และโปรแกรม IS.C มีลักษณะข้อมูลที่ค่อนข้างเหมือนกันซึ่งอยู่ในช่วง 2048-3071

(2) การอพยพแบบคงสถานะการทำงานบนเครือข่าย 100 Mbps

ตารางที่ 4.8 จำนวนเพจที่มีค่า 0 จากการทำการ XOR อยู่ในช่วงต่างๆ สำหรับการอพยพบนเครือข่าย 100 Mbps

Benchmark	0	1-1023	1024-2047	2048-3071	3072-4095	4096	Total XOR Pages
BT.C	0	88,571	10,017	685	719	2,769	102,761
CG.C	0	143,352	8,179	5,864	33,083	25,607	216,085
IS.C	0	0	23,074	2	1,214	3,334	27,624
LU.C	0	78,738	23,843	8,278	749	2,439	114,047
MG.C	0	28,431	26,767	511	837	265	56,811
SP.C	0	97,928	942	766	804	3,347	103,787
UA.C	0	68,123	1,934	2,377	33,027	18,976	124,437



ภาพที่ 4.14 สัดส่วนจำนวนเพจที่มีค่า 0 จากการทำการ XOR ที่อยู่ในช่วงต่างๆ ต่อจำนวนเพจทั้งหมดที่ทำ XOR สำหรับการอพยพบนเครือข่าย 100 Mbps

จาก ตารางที่ 4.8 และภาพที่ 4.14 พบว่า โปรแกรม CG.C มีหน่วยความจำเพจที่ไม่มีการเปลี่ยนแปลงเลยมากที่สุด รองลงมาคือ UA.C ในขณะที่โปรแกรม LU.C, SP.C, BT.C และ IS.C มีจำนวนเพจที่ไม่เปลี่ยนแปลงใกล้เคียงกัน แต่ MG.C กลับมีเพียงแค่ 265 เพจเท่านั้นที่ไม่มีการเปลี่ยนแปลง

นอกจากนี้ ยังพบว่า โปรแกรมส่วนใหญ่มีข้อมูลที่เหมือนกันน้อยซึ่งจะอยู่ในช่วง 1-1023 ในขณะที่โปรแกรม MG.C และ IS.C มีข้อมูลหน่วยความจำที่เหมือนกันส่วนใหญ่อยู่ในช่วง 1024-2047 และโปรแกรม UA.C หน่วยความจำเพจที่ค่อนข้างเหมือนกัน โดยดูจากจำนวนเพจที่ไม่เปลี่ยนแปลงและเพจที่ไม่เปลี่ยนแปลงอยู่ในช่วง 3072-4095 รวมกันเกือบ 50%

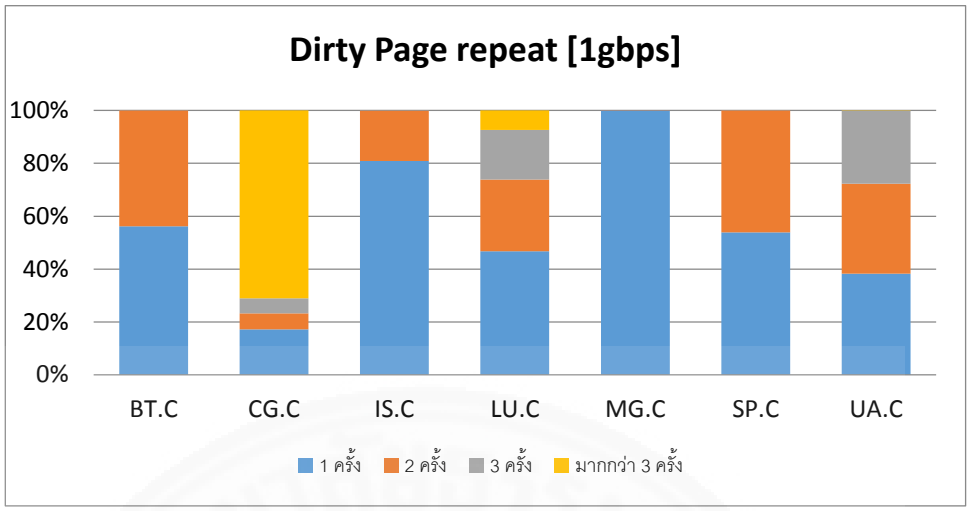
4.1.3.3 วิเคราะห์การเกิดซ้ำของหน่วยความจำเพจ

การอพยพแบบคงสถานะการทำงานของคอมพิวเตอร์เสมือนนั้นอพยพข้อมูลหน่วยความจำเพจในขณะที่คอมพิวเตอร์เสมือนกำลังทำงานอยู่ โดยเทรด dtx มีหน้าที่ถ่ายโอนหน่วยความจำที่เปลี่ยนแปลง ซึ่งหน่วยความจำเพจที่มีการเปลี่ยนแปลงนี้คอมพิวเตอร์เสมือนใช้งานและเปลี่ยนแปลงค่าอยู่ตลอดเวลา ทำให้เกิดการเปลี่ยนแปลงซ้ำที่หน่วยความจำเพจเดิม และการถ่ายโอนหน่วยความจำเพจเดิมซ้ำหลายครั้ง ก็เป็นการสิ้นเปลืองแบนด์วิดท์ ดังนั้นผู้วิจัยได้ทำการวิเคราะห์การเกิดซ้ำของข้อมูลหน่วยความจำเพจสำหรับโปรแกรม NPB โดยการนับว่า ในแต่ละโปรแกรมมีหน่วยความจำเพจเท่าไหร่ที่มีการเปลี่ยนแปลงซ้ำที่เพจเดิม 1 ครั้ง 2 ครั้ง 3 ครั้ง หรือมากกว่า 3 ครั้ง โดยวิเคราะห์แยกตามแบนด์วิดท์

(1) การอพยพแบบคงสถานะการทำงานบนเครือข่าย 1 Gbps

ตารางที่ 4.9 แสดงจำนวนเพจที่เปลี่ยนแปลงซ้ำที่เพจเดิมของแต่ละโปรแกรมสำหรับการอพยพบนเครือข่าย 1 Gbps

Benchmark	1 ครั้ง	2 ครั้ง	3 ครั้ง	มากกว่า 3 ครั้ง	XOR pages
BT.C	138,831	108,038	9	0	246,878
CG.C	6,503	2,267	2,121	26,809	37,700
IS.C	176,409	41,498	146	0	218,053
LU.C	117,309	68,173	47,012	18,377	250,871
MG.C	299,597	7	0	0	299,604
SP.C	150,930	129,036	8	0	279,974
UA.C	65,978	58,509	47,680	9	172,176



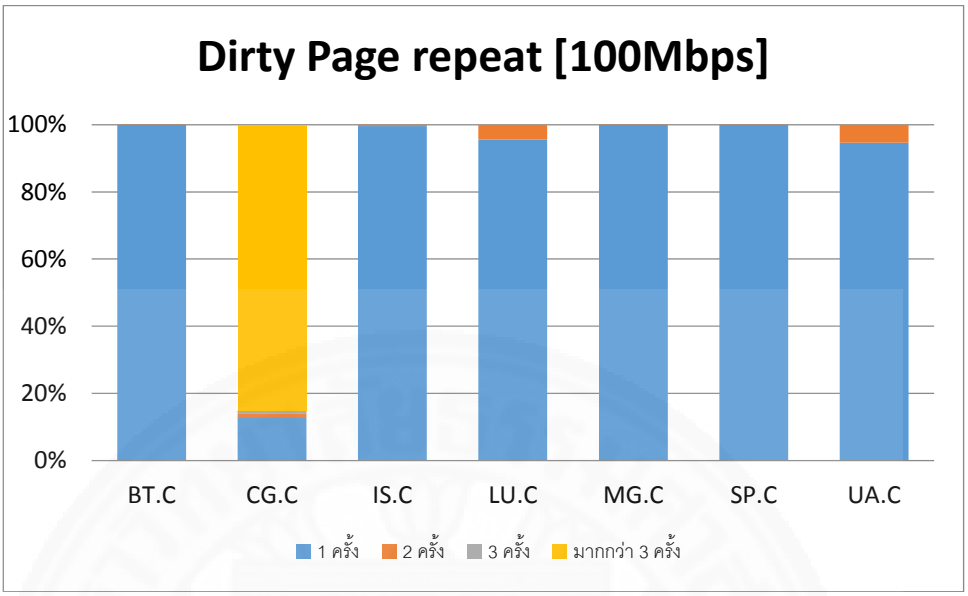
ภาพที่ 4.15 สัดส่วนจำนวนเพจที่เปลี่ยนแปลงซ้ำต่อจำนวนเพจทั้งหมดที่ทำ XOR สำหรับการอพยพบนเครือข่าย 1 Gbps

จากตารางที่ 4.9 และ ภาพที่ 4.15 พบว่าโปรแกรม CG.C และ LU.C นั้นมีการเปลี่ยนแปลงที่หน่วยความจำเพจเดิมมากกว่า 3 ครั้ง โดยที่โปรแกรม CG.C มีสัดส่วนของหน่วยความเพจเปลี่ยนแปลงซ้ำที่เพจเดิมมากกว่า 70% ในขณะที่โปรแกรม UA.C มีการเปลี่ยนแปลงซ้ำที่หน่วยความจำเพจเดิมมากสุดแค่ 3 ครั้ง ส่วนโปรแกรม BT.C, IS.C และ SP.C มีการเปลี่ยนแปลงที่เพจเดิมมากสุดแค่ 2 ครั้ง ส่วน MG.C เปลี่ยนแปลงซ้ำที่เพจเดิมแค่ 1 ครั้ง ข้อมูลตรงส่วนนี้สามารถนำไปใช้เป็นแนวทางประกอบในการทำ page reordering ได้

(2) การอพยพแบบคงสถานะการทำงานบนเครือข่าย 100 Mbps

ตารางที่ 4.10 แสดงจำนวนเพจที่เปลี่ยนแปลงซ้ำที่เพจเดิมของแต่ละโปรแกรมสำหรับการอพยพบนเครือข่าย 100 Mbps

Benchmark	1 ครั้ง	2 ครั้ง	3 ครั้ง	มากกว่า 3 ครั้ง	XOR pages
BT.C	102,734	27	0	0	102,761
CG.C	28,038	2,098	2,245	183,704	216,085
IS.C	27,517	107	0	0	27,624
LU.C	109,061	4,984	2	0	114,047
MG.C	56,796	15	0	0	56,811
SP.C	103,756	31	0	0	103,787
UA.C	117,810	6,621	6	0	124,437



ภาพที่ 4.16 สัดส่วนจำนวนเพจที่เปลี่ยนแปลงซ้ำต่อจำนวนเพจทั้งหมดที่ทำ XOR สำหรับการอพยพบนเครือข่าย 100 Mbps

จากตารางที่ 4.10ตารางที่ 4.9 และ ภาพที่ 4.16 พบว่าการอพยพแบบคงสถานะการทำงานของคอมพิวเตอร์เสมือนผ่านเครือข่ายแบนด์วิดท์ 100 Mbps นั้น ทำให้การเปลี่ยนแปลงหน่วยความจำเพจซ้ำที่เพจเดิมลดน้อยลง โดยโปรแกรมส่วนใหญ่มีการเปลี่ยนแปลงซ้ำเพียงแค่ครั้งเดียวเป็นส่วนใหญ่ จะมีเพียงโปรแกรม CG.C เท่านั้นที่มีการเปลี่ยนแปลงหน่วยความจำเพจซ้ำที่เพจเดิมมากกว่า 3 ครั้ง และมีอัตราส่วนสูงถึง 80%

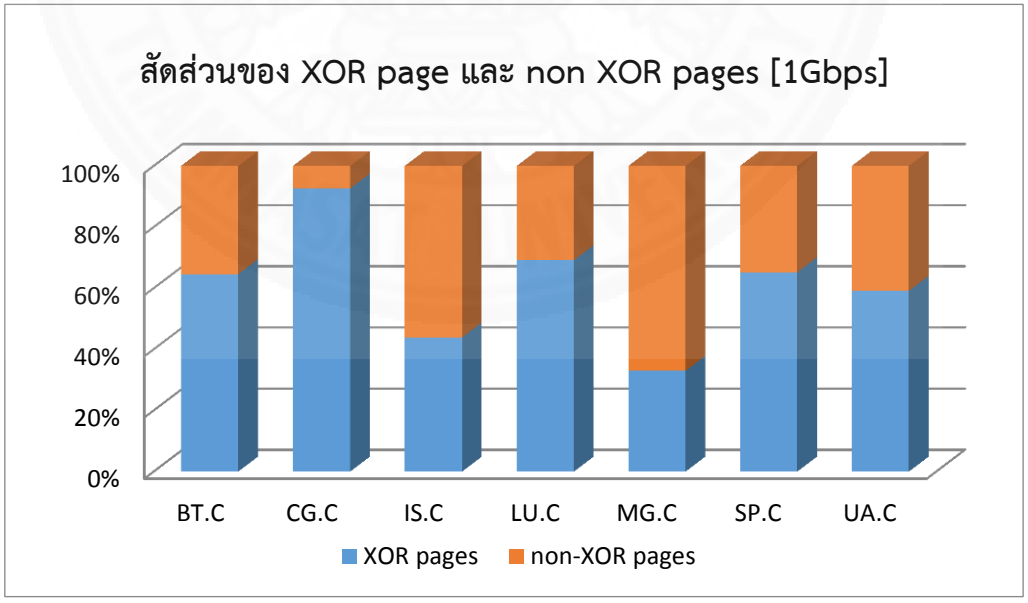
4.1.3.4 วิเคราะห์สัดส่วนของเพจที่ผ่านการทำ XOR

วิเคราะห์หน่วยความจำเพจที่ผ่านการเข้ารหัสแบบเดลตาด้วยวิธี XOR ของแต่ละโปรแกรม โดยวิเคราะห์แยกเป็นหน่วยความจำที่เทรด dtx ถ่ายโอนมาที่ Memory Server และที่เทรด io ถ่ายโอนในช่วงที่คอมพิวเตอร์เสมือนหยุดทำงาน โดย receiving page คือ หน่วยความจำเพจทั้งหมดที่ Memory Server ได้รับ ส่วน XOR pages คือ หน่วยความจำเพจที่ผ่านการเข้ารหัสแบบเดลตาด้วยวิธี XOR และคำนวณสัดส่วนของเพจที่ผ่านการทำ XOR ต่อจำนวนเพจที่ได้รับทั้งหมด

(1) การอพยพแบบคงสถานะการทำงานบนเครือข่าย 1 Gbps

ตารางที่ 4.11 สัดส่วนของเพจที่ผ่านการทำ XOR ต่อจำนวนเพจทั้งหมดที่ได้รับของแต่ละโปรแกรม สำหรับการอพยพบนเครือข่าย 1 Gbps

Bench mark	dtx			s3			Total		
	receiving pages	Xor pages	Xor/ receiving	receiving pages	Xor pages	Xor/ receiving	receiving pages	Xor pages	Xor/ receiving
BT.C	247,078	112,204	45.41%	134,985	134,674	99.77%	382,063	246,878	64.62%
CG.C	39,200	36,212	92.38%	1,490	1,488	99.87%	40,690	37,700	92.65%
IS.C	253,775	79,493	31.32%	243,138	138,560	56.99%	496,913	218,053	43.88%
LU.C	275,313	163,541	59.40%	87,335	87,330	99.99%	362,648	250,871	69.18%
MG.C	320,044	8	0.00%	588,416	299,596	50.92%	908,460	299,604	32.98%
SP.C	286,784	137,709	48.02%	142,583	142,265	99.78%	429,367	279,974	65.21%
UA.C	216,727	117,536	54.23%	74,485	54,640	73.36%	291,212	172,176	59.12%



ภาพที่ 4.17 สัดส่วนของเพจที่มีการทำ XOR และเพจที่ไม่ผ่านการทำ XOR ของแต่ละโปรแกรม สำหรับการอพยพบนเครือข่าย 1 Gbps

จากตารางที่ 4.11 และ ภาพที่ 4.17 พบว่า ในช่วงที่คอมพิวเตอร์เสมือนกำลังทำงานอยู่นั้น โปรแกรม CG.C มีสัดส่วนของเพจที่ผ่านการเข้ารหัสด้วย XOR สูงมากถึง 92% แต่ MG.C กลับแทบจะไม่มีเพจที่ผ่านการเข้ารหัสด้วย XOR เลย ในขณะที่โปรแกรมอื่นมีสัดส่วนของเพจที่ผ่านการเข้ารหัสด้วย XOR ประมาณ 50% ยกเว้น IS.C ซึ่งมีสัดส่วนของเพจที่ผ่านการเข้ารหัสด้วย XOR เพียง 31% เท่านั้น

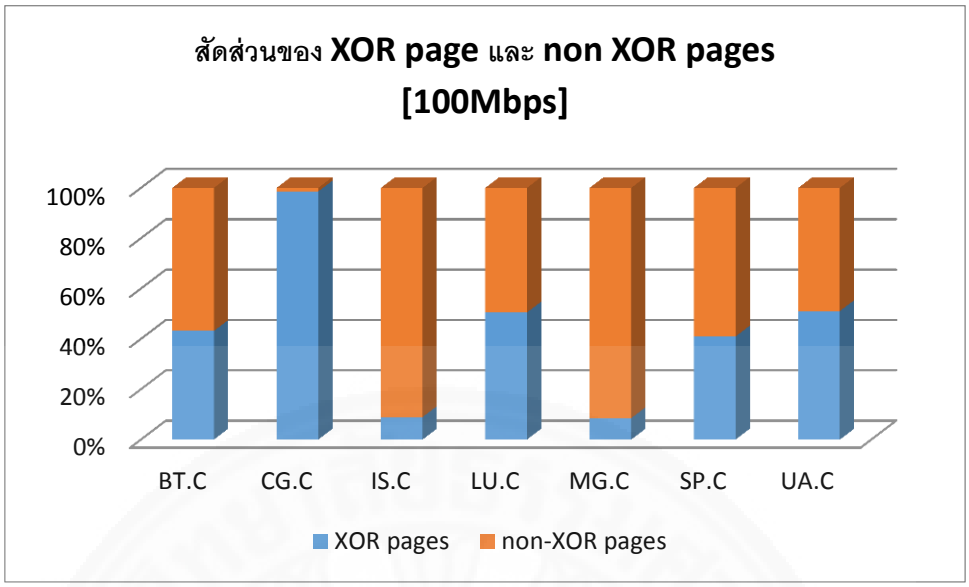
ส่วนในช่วงที่คอมพิวเตอร์เสมือนหยุดทำงาน โปรแกรม BT.C, CG.C, LU.C และ SP.C มีสัดส่วนของเพจที่ผ่านการเข้ารหัสด้วย XOR เกือบ 100% ส่วน UA.C มีสัดส่วน 73% ในขณะที่ IS.C และ MG.C มีสัดส่วนของเพจที่ผ่านการเข้ารหัสด้วย XOR เพียง 50% เท่านั้น

เมื่อพิจารณาโดยรวม มีเพียงโปรแกรม CG.C ที่มีสัดส่วนของเพจที่ผ่านการเข้ารหัสด้วย XOR สูงถึง 93% ในขณะที่ LU.C, SP.C, BT.C, UA.C มีสัดส่วนอยู่ระหว่าง 59%-69% และ IS.C มีสัดส่วน 44% ส่วน MG.C มีสัดส่วนโดยรวมน้อยที่สุดเพียงแค่ 33% เท่านั้น

(2) การอพยพแบบคงสถานะการทำงานบนเครือข่าย 100 Mbps

ตารางที่ 4.12 สัดส่วนของเพจที่ผ่านการทำ XOR ต่อจำนวนเพจทั้งหมดที่ได้รับของแต่ละโปรแกรมสำหรับการอพยพบนเครือข่าย 100 Mbps

Bench mark	dtx			s3			Total		
	receiving pages	xor pages	xor/ receiving	receiving pages	xor pages	xor/ receiving	receiving pages	xor pages	xor/ receiving
BT.C	103,437	28	0.03%	134,581	102,733	76.34%	238,018	102,761	43.17%
CG.C	217,433	214,352	98.58%	1,737	1,733	99.77%	219,170	216,085	98.59%
IS.C	69,009	210	0.30%	239,048	27,414	11.47%	308,057	27,624	8.97%
LU.C	115,578	5,893	5.10%	110,279	108,154	98.07%	225,857	114,047	50.50%
M.G.C	56,820	16	0.03%	608,968	56,795	9.33%	665,788	56,811	8.53%
SP.C	103,789	31	0.03%	149,550	103,756	69.38%	253,339	103,787	40.97%
UA.C	125,721	6,679	5.31%	117,897	117,758	99.88%	243,618	124,437	51.08%



ภาพที่ 4.18 สัดส่วนของเพจที่มีการทำ XOR และหน้าที่ไม่ผ่านการทำ XOR ของแต่ละโปรแกรม สำหรับการอพยพบนเครือข่าย 100 Mbps

จากตารางที่ 4.12 และ ภาพที่ 4.18 พบว่า ในช่วงที่คอมพิวเตอร์เสมือนกำลังทำงานอยู่นั้น มีเพียงโปรแกรม CG.C เท่านั้นที่มีสัดส่วนของเพจที่ผ่านการเข้ารหัสด้วย XOR สูงและสูงมากถึง 99% ในขณะที่โปรแกรม LU.C และ UA.C มีสัดส่วนของเพจที่ผ่านการเข้ารหัสด้วย XOR เพียงแค่ 5% ส่วนโปรแกรมอื่นมีสัดส่วนของเพจที่ผ่านการเข้ารหัสด้วย XOR เพียงน้อยนิดเท่านั้นไม่ถึง 1%

ส่วนในช่วงที่คอมพิวเตอร์เสมือนหยุดทำงาน โปรแกรม CG.C, LU.C และ UA.C มีสัดส่วนของเพจที่ผ่านการเข้ารหัสด้วย XOR เกือบ 100% ในขณะที่ BT.C และ SP.C มีสัดส่วนของเพจที่ผ่านการเข้ารหัสด้วย XOR ค่อนข้างสูง แต่โปรแกรม IS.C และ MG.C กลับมีสัดส่วนของเพจที่ผ่านการเข้ารหัสด้วย XOR เพียงเล็กน้อยเท่านั้น

เมื่อพิจารณาโดยรวม มีเพียงโปรแกรม CG.C ที่มีสัดส่วนของเพจที่ผ่านการเข้ารหัสด้วย XOR สูงถึง 99% ในขณะที่ LU.C, SP.C, BT.C, UA.C มีสัดส่วนอยู่ระหว่าง 40%-51% แต่โปรแกรม IS.C และ MG.C มีสัดส่วนโดยรวมน้อยมากเพียงแค่ 9% เท่านั้น

4.2 การอพยพแบบคงสถานะการทำงานแบบ TLM-BZ

การอพยพแบบคงสถานะการทำงานแบบ TLM-BZ ด้วยการเพิ่มการบีบอัดข้อมูลในระดับหน่วยความจำเพจด้วย Blosc ซึ่งอาศัยการสร้างเทรดมาช่วยบีบอัดและมีการทำ filter ก่อนการบีบอัด โดยการจัดเรียงลำดับไบต์ (shuffle) หรือ เรียงลำดับบิต (bitshuffle) เพื่อทำให้การบีบอัดมีประสิทธิมากขึ้นและเร็วขึ้น โดยสามารถเลือกใช้วิธีการบีบอัดดังต่อไปนี้ BloscLZ, LZ4, LZ4hc, Snappy และ Zlib และสามารถกำหนดระดับการบีบอัดได้ตั้งแต่ 0 (ไม่บีบอัด) ถึง 9 โดยระดับที่ 1 เน้นความเร็วในการบีบอัดซึ่งจะทำให้อัตราการบีบอัดลดลง ในขณะที่ระดับที่ 9 เน้นอัตราการบีบอัดซึ่งทำให้ความเร็วในการบีบอัดลดลง เนื่องจากหน่วยความจำในระดับเพจมีขนาดเพียง 4 KB ซึ่งมีขนาดเล็กมาก ในบางครั้งเมื่อบีบอัดข้อมูลแล้วจะทำให้ข้อมูลมีขนาดที่ใหญ่ขึ้น ซึ่งเมื่อเป็นเช่นนั้น Blosc จะยกเลิกการบีบอัดเพจนั้น ดังนั้นในงานวิจัยนี้จึงเลือกระดับการบีบอัดที่ระดับ 6 โดยใช้เพียง 2 เทรด และใช้การเรียงลำดับบิต โดยกำหนด typesize ซึ่งเป็นขนาดของข้อมูลเชิงเดี่ยว (atomic type) ไว้ที่ 8

ทดสอบประสิทธิภาพของการอพยพแบบ TLM-BZ โดยมีสมมติฐานว่า

(1) การเพิ่มการบีบอัดข้อมูลด้วย Blosc ทำให้เวลาหยุดการทำงานและเวลารวมสำหรับการอพยพของ TLM-BZ น้อยกว่า TLM และ TLM-Z

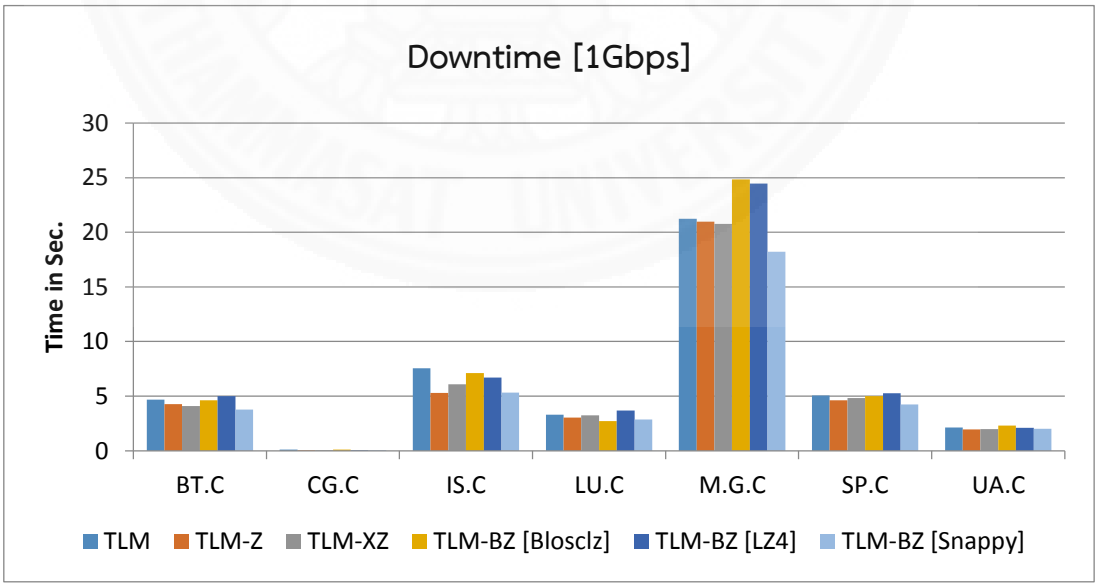
(2) การเพิ่มการบีบอัดข้อมูลด้วย Blosc ทำให้ประสิทธิภาพของการอพยพแบบคงสถานะการทำงานบนระบบเครือข่าย 100 Mbps ดีกว่า 1 Gbps

4.2.1 ทดลองบนเครือข่ายที่มีแบนด์วิดท์ 1 Gbps

ตารางที่ 4.13 เปรียบเทียบประสิทธิภาพด้วย Downtime ของ TLM, TLM-Z, TLM-XZ และ TLM-BZ บนเครือข่ายที่มีแบนด์วิดท์ 1 Gbps

Benchmark	Downtime [1gbps] (Time in Sec.)							
	TLM	TLM-Z	TLM-XZ	TLM-BZ**				
				blosclz	lz4	lz4hc	snappy	zlib
BT.C	4.66	4.26	4.09	4.63	5.01	38.24	3.78	128.94
CG.C	0.11	0.04	0.06	0.09	0.05	0.51	0.04	1.54
IS.C	7.54	5.30	6.08	7.11	6.71	65.34	5.32	149.43
LU.C	3.31	3.05	3.25	2.70	3.68	27.82	2.85	87.92
M.G.C	21.24	20.98	20.76	24.84	24.48	179.75	18.22	579.38
SP.C	5.05	4.62	4.82	4.99	5.27	38.84	4.25	128.73
UA.C	2.14	1.96	1.97	2.30	2.10	15.70	2.02	58.49

** TLM-BZ Parameter: 2 threads, clevel 6, bitshuffle, typesize 8



ภาพที่ 4.19 เปรียบเทียบประสิทธิภาพด้วย Downtime ของ TLM, TLM-Z, TLM-XZ และ TLM-BZ บนเครือข่ายที่มีแบนด์วิดท์ 1 Gbps

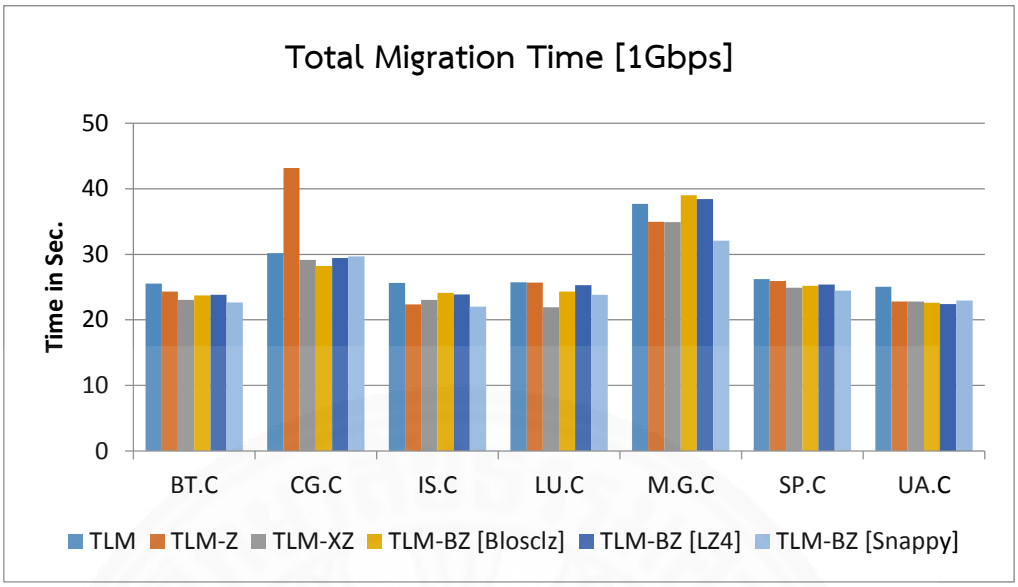
การวัดประสิทธิด้วยเวลาหยุดการทำงานของคอมพิวเตอร์เสมือน จากตารางที่ 4.13 พบว่า Snappy เป็นวิธีบีบอัดที่ดีสุดสำหรับ TLM-BZ บนเครือข่ายที่มีแบนด์วิดท์ 1 Gbps สามารถทำให้เวลาหยุดการทำงานลดลงสำหรับโปรแกรม BT.C, MG.C และ SP.C เมื่อเทียบกับ TLM และ TLM-Z โดยลดลงเมื่อเทียบกับ TLM 19%, 14%, และ 16% ตามลำดับ และลดลงเมื่อเทียบกับ TLM-Z 11%, 13% และ 8% ตามลำดับ ในขณะที่ทำให้เวลาหยุดการทำงานเพิ่มขึ้น 3% สำหรับโปรแกรม UA.C เมื่อเทียบกับ TLM-Z

ในการทดลองนี้ยังพบอีกว่า Zlib และ LZ4hc ใช้เวลายาวนานกว่าวิธีบีบอัดวิธีอื่นมาก เนื่องจาก Zlib และ LZ4hc เป็นวิธีการบีบอัดที่มุ่งเน้นเพื่อให้ได้อัตราการบีบอัดที่สูง ในขณะที่วิธีบีบอัด BloscLZ, LZ4 และ Snappy มุ่งเน้นความเร็วในการบีบอัด จึงสรุปได้ว่า วิธีบีบอัดที่เน้นความเร็วเหมาะสมกับลักษณะของหน่วยความจำมากกว่าวิธีบีบอัดที่เน้นอัตราการบีบอัดที่สูง

ตารางที่ 4.14 เปรียบเทียบประสิทธิภาพด้วย Total Migration Time ของ TLM, TLM-Z, TLM-XZ และ TLM-BZ บนเครือข่ายที่มีแบนด์วิดท์ 1 Gbps

Benchmark	Total Migration Time [1gbps] (Time in sec.)							
	TLM	TLM-Z	TLM-XZ	TLM-BZ**				
				blosclz	lz4	lz4hc	snappy	zlib
BT.C	25.51	24.28	23.05	23.73	23.82	57.18	22.65	148.03
CG.C	30.16	43.19	29.12	28.21	29.44	29.90	29.69	30.71
IS.C	25.64	22.36	23.04	24.11	23.85	82.27	22.03	166.31
LU.C	25.74	25.69	21.90	24.29	25.30	49.53	23.82	108.36
M.G.C	37.71	34.95	34.89	39.03	38.45	194.10	32.09	593.40
SP.C	26.22	25.93	24.88	25.20	25.38	58.68	24.46	149.04
UA.C	25.04	22.81	22.78	22.58	22.39	35.60	22.96	78.61

** TLM-BZ Parameter: 2 threads, clevel 6, bitshuffle, typesize 8



ภาพที่ 4.20 เปรียบเทียบประสิทธิภาพด้วย Total Migration Time ของ TLM, TLM-Z, TLM-XZ และ TLM-BZ บนเครือข่ายที่มีแบนด์วิดท์ 1 Gbps

การวัดประสิทธิด้วยเวลารวมสำหรับอพยพ จากตารางที่ 4.14 พบว่า Snappy เป็นวิธีบีบอัดที่ดีสุดสำหรับ TLM-BZ บนเครือข่ายที่มีแบนด์วิดท์ 1 Gbps สามารถทำให้เวลารวมสำหรับอพยพ ลดลงสำหรับโปรแกรม BT.C, IS.C, MG.C และ SP.C เมื่อเทียบกับ TLM และ TLM-Z โดยลดลงเมื่อเทียบกับ TLM 11%, 14%, 15%, และ 7% ตามลำดับ และลดลงเมื่อเทียบกับ TLM-Z 7%, 1%, 8% และ 6% ตามลำดับ

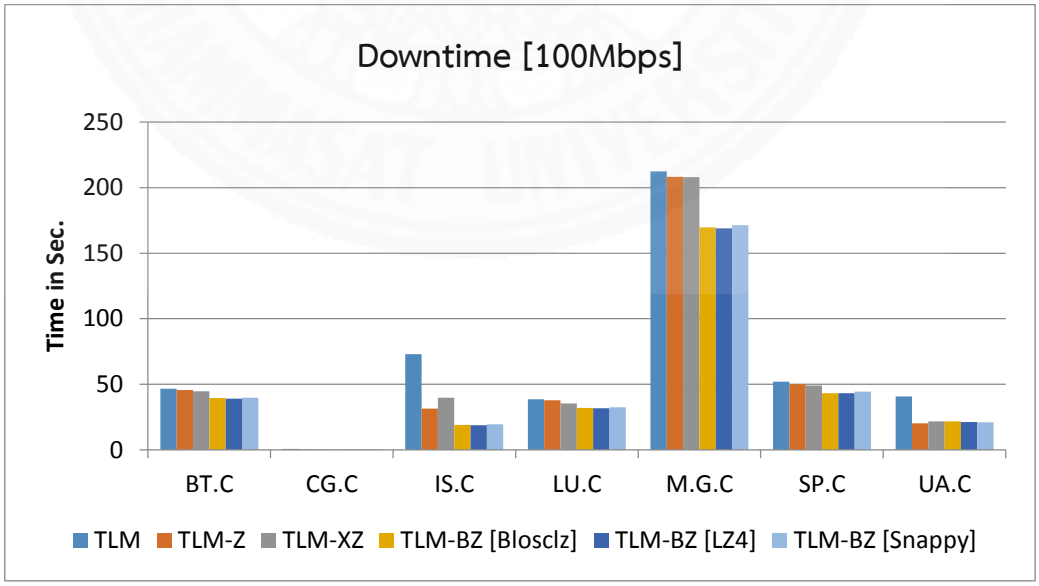
สรุปการวัดประสิทธิภาพการอพยพแบบคงสถานะการทำงานของคอมพิวเตอร์เสมือนด้วยเวลาหยุดการทำงานและเวลารวมสำหรับการอพยพบนเครือข่ายที่มีแบนด์วิดท์ 1 Gbps พบว่า Snappy เป็นวิธีบีบอัดที่ดีที่สุด สามารถทำให้ประสิทธิภาพการอพยพแบบ TLM-BZ ดีขึ้นกว่าของ TLM และ TLM-Z สำหรับโปรแกรม BT.C, CG.C, MG.C และ SP.C

4.2.2 ทดลองบนเครือข่ายที่มีแบนด์วิดท์ 100 Mbps

ตารางที่ 4.15 เปรียบเทียบประสิทธิภาพด้วย Downtime ของ TLM, TLM-Z, TLM-XZ และ TLM-BZ บนเครือข่ายที่มีแบนด์วิดท์ 100 Mbps

Benchmark	Downtime [100mbps] (Time in Sec.)							
	TLM	TLM-Z	TLM-XZ	TLM-BZ**				
				blosclz	lz4	lz4hc	snappy	zlib
BT.C	46.67	45.59	44.55	39.43	39.06	39.41	39.61	129.03
CG.C	0.62	0.29	0.19	0.18	0.25	0.56	0.26	1.82
IS.C	72.83	31.52	39.77	19.04	18.80	69.06	19.41	161.22
LU.C	38.40	37.79	35.21	31.86	31.76	29.50	32.35	105.16
M.G.C	212.31	208.21	208.14	169.68	168.99	183.59	171.27	579.52
SP.C	51.88	49.94	49.06	43.13	43.06	43.38	44.29	143.84
UA.C	40.75	20.15	21.73	21.57	21.23	33.18	20.85	99.83

** TLM-BZ Parameter: 2 threads, clevel 6, bitshuffle, typesize 8



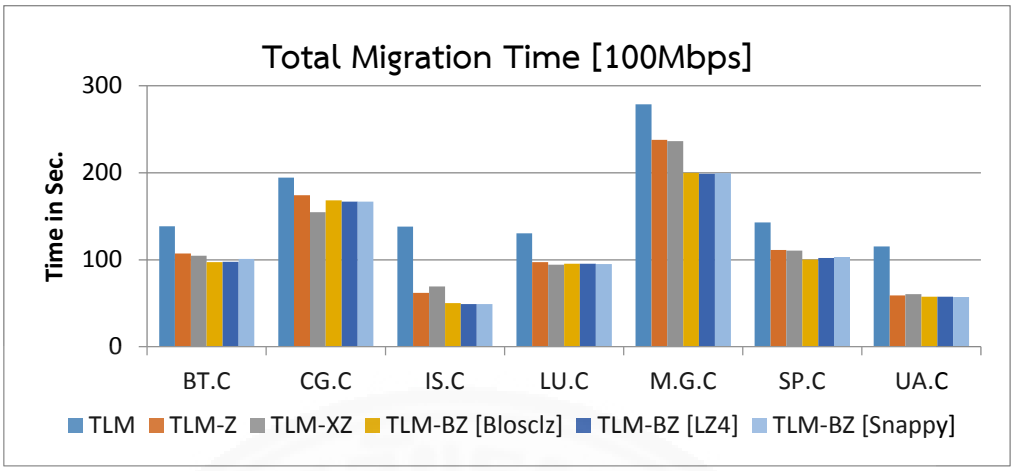
ภาพที่ 4.21 เปรียบเทียบประสิทธิภาพด้วย Downtime ของ TLM, TLM-Z, TLM-XZ และ TLM-BZ บนเครือข่ายที่มีแบนด์วิดท์ 100 Mbps

การวัดประสิทธิด้วยเวลาหยุดการทำงานของคอมพิวเตอร์เสมือนสำหรับ TLM-BZ บนเครือข่ายที่มีแบนด์วิดท์ 100 Mbps จากตารางที่ 4.15 พบว่า BloscLZ และ LZ4 มีประสิทธิภาพในการบีบอัดใกล้เคียงกันและเป็นวิธีบีบอัดที่ดีที่สุดสำหรับโปรแกรม BT.C, IS.C, LU.C, MG.C และ SP.C โดยทำให้เวลาหยุดการทำงานของคอมพิวเตอร์เสมือนน้อยกว่า TLM 16%, 74%, 17%, 20% และ 17% ตามลำดับ และน้อยกว่า TLM-Z 14%, 40%, 16%, 19% และ 14% ตามลำดับ BloscLZ มีประสิทธิภาพในบีบอัดดีกว่า LZ4 สำหรับโปรแกรม CG.C โดยทำให้เวลาหยุดการทำงานของคอมพิวเตอร์เสมือนน้อยกว่า TLM 72% และน้อยกว่า TLM-Z 39% ในขณะที่ LZ4hc มีประสิทธิภาพในการบีบอัดที่ดีที่สุดสำหรับโปรแกรม LU.C โดยทำให้เวลาหยุดการทำงานของคอมพิวเตอร์เสมือนน้อยกว่า TLM 23% และน้อยกว่า TLM-Z 22% อย่างไรก็ตาม TLM-BZ ไม่สามารถทำให้ประสิทธิภาพของการอพยพสำหรับโปรแกรม UA.C ดีขึ้นกว่า TLM-Z หากแต่ว่า ยังคงดีกว่า TLM

ตารางที่ 4.16 เปรียบเทียบประสิทธิภาพด้วย Total Migration Time ของ TLM, TLM-Z, TLM-XZ และ TLM-BZ บนเครือข่ายที่มีแบนด์วิดท์ 100 Mbps

Benchmark	Total Migration Time [100Mbps] (Time in sec.)							
	TLM	TLM-Z	TLM-XZ	TLM-BZ**				
				blosclz	lz4	lz4hc	snappy	zlib
BT.C	138.65	107.25	104.67	97.38	97.58	100.46	100.80	188.62
CG.C	194.58	174.15	154.71	168.29	166.74	165.08	167.03	167.87
IS.C	138.28	62.11	69.37	50.21	49.00	98.49	49.23	190.35
LU.C	130.59	97.26	94.39	95.32	95.44	98.56	95.25	168.71
M.G.C	278.68	238.03	236.43	200.19	198.90	210.68	199.57	608.70
SP.C	143.05	111.16	110.38	100.15	101.99	107.38	103.27	201.04
UA.C	115.31	58.87	60.46	57.38	57.45	72.18	57.27	136.42

** TLM-BZ Parameter: 2 threads, clevel 6, bitshuffle, typesize 8



ภาพที่ 4.22 เปรียบเทียบประสิทธิภาพด้วย Total Migration Time ของ TLM, TLM-Z, TLM-XZ และ TLM-BZ บนเครือข่ายที่มีแบนด์วิดท์ 100 Mbps

การวัดประสิทธิด้วยเวลารวมสำหรับการอพยพของคอมพิวเตอร์เสมือนสำหรับ TLM-BZ บนเครือข่ายที่มีแบนด์วิดท์ 100 Mbps จากตารางที่ 4.16 พบว่า BloscLZ และ LZ4 มีประสิทธิภาพในการบีบอัดใกล้เคียงกันและเป็นวิธีบีบอัดที่ดีที่สุดสำหรับทุกโปรแกรม โดยทำให้เวลาเวลารวมสำหรับการอพยพสำหรับ BT.C, CG.C , IS.C, LU.C, MG.C, SP.C และ UA.C น้อยกว่า TLM 30%, 14%, 65%, 27%, 29%, 29% และ 3% ตามลำดับ และน้อยกว่า TLM-Z 9%, 4%, 21%, 2%, 16%, 10% และ 3% ตามลำดับ

สรุปการวัดประสิทธิภาพการอพยพแบบคงสถานะการทำงานของคอมพิวเตอร์เสมือนด้วยเวลาหยุดการทำงานและเวลารวมสำหรับการอพยพบนเครือข่ายแบนด์วิดท์ 100 Mbps พบว่า BloscLZ และ LZ4 มีประสิทธิภาพในการบีบอัดใกล้เคียงกัน และ LZ4 เป็นวิธีบีบอัดที่ดีที่สุด สามารถทำให้ประสิทธิภาพการอพยพแบบ TLM-BZ ดีขึ้นกว่าของ TLM และ TLM-Z สำหรับโปรแกรมทุกโปรแกรมยกเว้น UA.C โดย LZ4hc กลับทำให้ TLM-BZ มีเวลาหยุดทำงานของคอมพิวเตอร์น้อยที่สุดสำหรับโปรแกรม LU.C

การอพยพแบบคงสถานะการทำงานของคอมพิวเตอร์เสมือนแบบ TLM-BZ บนเครือข่ายแบนด์วิดท์ 1 Gbps ด้วยวิธีบีบอัด Snappy โดยใช้เทรด 2 เทรด และการเรียงลำดับบิต ทำให้เวลาหยุดการทำงานและเวลารวมสำหรับการอพยพของ TLM-BZ น้อยกว่า TLM และ TLM-Z สำหรับโปรแกรม BT.C, CG.C, MG.C และ SP.C และทำให้เวลาหยุดการทำงานและเวลารวมสำหรับการอพยพบนเครือข่ายแบนด์วิดท์ 100 Mbps ของ TLM-BZ น้อยกว่า TLM และ TLM-Z ด้วยวิธีบีบอัด LZ4 สำหรับทุกโปรแกรมยกเว้น UA.C อีกทั้ง TLM-BZ ทำให้ประสิทธิภาพของการอพยพแบบคงสถานะการทำงานบนระบบเครือข่าย 100 Mbps ดีกว่า 1 Gbps

บทที่ 5
สรุปผลการวิจัยและข้อเสนอแนะ

5.1 สรุปผลการวิจัย

งานวิจัยมีวัตถุประสงค์เพื่อเพิ่มประสิทธิภาพของการอพยพแบบคงสถานะการทำงานของคอมพิวเตอร์เสมือนแบบเทรดด้วยการบีบอัดข้อมูล โดยปรับปรุงการอพยพแบบคงสถานะการทำงานแบบ TLM-Z โดยการนำเสนอ Framework สำหรับการบีบอัดแบบใหม่ที่เพิ่ม Memory Server เข้ามาระหว่างคอมพิวเตอร์เสมือนและเครื่องแม่ข่าย Z-Server เพื่อจัดการบีบอัดหน่วยความจำในระดับเพจ 2 วิธี ได้แก่

(1) การเข้ารหัสแบบเดลตาด้วยวิธี XOR (TLM-XZ)

(2) การเพิ่มการบีบอัดข้อมูลด้วย Blosc (TLM-BZ)

โดยมีสมมุติฐานดังนี้

(1) การเข้ารหัสแบบเดลตาด้วยวิธี XOR ทำให้เวลาหยุดการทำงานและเวลารวมสำหรับการอพยพของ TLM-XZ น้อยกว่า TLM และ TLM-Z

(2) การเข้ารหัสแบบเดลตาด้วยวิธี XOR ทำให้ประสิทธิภาพของการอพยพแบบคงสถานะการทำงานบนระบบเครือข่าย 100 Mbps ดีกว่า 1 Gbps

(3) การเข้ารหัสแบบเดลตาด้วยวิธี XOR ช่วยเพิ่มประสิทธิภาพการบีบอัดข้อมูลของ Z-Server

(4) การเพิ่มการบีบอัดข้อมูลระดับหน่วยความจำเพจด้วย Blosc ทำให้เวลาหยุดการทำงานและเวลารวมสำหรับการอพยพของ TLM-BZ น้อยกว่า TLM และ TLM-Z

(5) การเพิ่มการบีบอัดข้อมูลด้วย Blosc ทำให้ประสิทธิภาพของการอพยพแบบคงสถานะการทำงานบนระบบเครือข่าย 100 Mbps ดีกว่า 1 Gbps

ทำการทดลองโดยอพยพเครื่องคอมพิวเตอร์เสมือนที่รันโปรแกรมแบบ OpenMP จาก NPB 7 โปรแกรม ได้แก่ BT.C, CG.C, IS.C, LU.C, MG.C, SP.C และ UA.C ซึ่งเป็นโปรแกรมประมวลผลทางวิทยาศาสตร์ที่ใช้หน่วยความจำและหน่วยประมวลผลมาก ผ่านระบบเครือข่ายที่มีแบนด์วิดท์ 1 Gbps และ 100 Mbps วัดประสิทธิภาพของการอพยพด้วยเวลารวมสำหรับการอพยพและหยุดการทำงานของคอมพิวเตอร์เสมือน

ผลการวิจัยพบว่า

(1) การบีบอัดข้อมูลด้วย LZ4 สามารถเพิ่มประสิทธิภาพของการอพยพแบบคงสถานะการทำงานได้ดีกว่าแบบที่ไม่มีการบีบอัด

(2) การอพยพแบบคงสถานะการทำงานผ่านเครือข่ายแบนด์วิดท์ 1 Gbps TLM-XZ มีเวลาหยุดการทำงานและเวลารวมสำหรับการอพยพน้อยกว่า TLM และ TLM-Z สำหรับโปรแกรม BT.C และ MG.C แต่กลับมีเวลาหยุดการทำงานและเวลารวมสำหรับการอพยพมากกว่า TLM-Z สำหรับโปรแกรม CG.C, IS.C, LU.C, SP.C และ UA.C

(3) การอพยพแบบคงสถานะการทำงานผ่านเครือข่ายที่มีแบนด์วิดท์ 100 Mbps แบบ TLM-XZ มีเวลาหยุดการทำงานและเวลารวมสำหรับการอพยพน้อยกว่า TLM และ TLM-Z สำหรับโปรแกรม BT.C CG.C, LU.C, MG.C และ SP.C แต่กลับมีเวลาหยุดการทำงานและเวลารวมสำหรับการอพยพมากกว่า TLM-Z สำหรับโปรแกรม IS.C 26% และ UA.C 8%

(4) ประสิทธิภาพการอพยพแบบคงสถานะการทำงานแบบ TLM-XZ ผ่านเครือข่ายที่มีแบนด์วิดท์ 100 Mbps ดีกว่าเครือข่ายที่มีแบนด์วิดท์ 1 Gbps

(5) จากการวิเคราะห์ลักษณะของหน่วยความจำเพจพบว่า การเข้ารหัสแบบเดลตาด้วยวิธี XOR ไม่สามารถทำให้ประสิทธิภาพการบีบอัดของ LZ4 ดีขึ้น สำหรับโปรแกรม IS.C เนื่องจากข้อมูลหน่วยความจำเพจเวอร์ชันปัจจุบันเหมือนกับข้อมูลเวอร์ชันก่อนหน้าค่อนข้างมาก แต่ข้อมูลหน่วยความจำเพจที่เหมือนกันนั้นกลับไม่ได้อยู่ติดกันเลย ในขณะที่ LZ4 ค้นหารูปแบบที่ซ้ำกันเป็นจำนวน 4 ไบต์ อีกทั้งการถ่ายโอนข้อมูลหน่วยความจำผ่าน Memory Server ยังส่งผลต่อการบีบอัดหน่วยความจำเพจของโปรแกรม IS.C และ UA.C เมื่อถ่ายโอนข้อมูลหน่วยความจำเพจผ่านเครือข่ายที่มีแบนด์วิดท์ลดลงยิ่งทำให้ประสิทธิภาพการบีบอัดข้อมูลของ LZ4 ลดลงมากยิ่งขึ้น

(6) จากการวิเคราะห์ผลของการเข้ารหัสแบบเดลตาด้วยวิธี XOR ต่อประสิทธิภาพในการบีบอัดข้อมูลของ LZ4 ด้วย Space Saving พบว่า การเข้ารหัสข้อมูลแบบเดลตาด้วยวิธี XOR ทำให้ประสิทธิภาพการบีบอัดข้อมูลของ LZ4 ลดลงสำหรับโปรแกรม IS.C และ UA.C

(7) การอพยพแบบคงสถานะการทำงานแบบ TLM-BZ ผ่านเครือข่าย 1 Gbps ด้วยวิธีบีบอัด Snappy มีประสิทธิภาพดีกว่า TLM สำหรับโปรแกรม BT.C 15%, CG.C 68%, MG.C 14% และ SP.C 16% และดีกว่า TLM-Z สำหรับโปรแกรม BT.C, CG.C, MG.C และ SP.C ดีขึ้น 7%, 19%, 13% และ 8% ตามลำดับ และการอพยพแบบคงสถานะการทำงานแบบ TLM-BZ ผ่านเครือข่ายแบนด์วิดท์ 100 Mbps ด้วยวิธีบีบอัด LZ4 มีประสิทธิภาพดีกว่า TLM และ TLM-Z สำหรับทุกโปรแกรมยกเว้น UA.C

(8) ประสิทธิภาพการอพยพแบบคงสถานะการทำงานแบบ TLM-BZ ผ่านเครือข่ายที่มีแบนด์วิดท์ 100 Mbps ดีกว่าเครือข่ายที่มีแบนด์วิดท์ 1 Gbps

5.2 การอภิปรายและข้อเสนอแนะ

จากการทดลองด้วยการเข้ารหัสแบบเดลตาด้วยวิธี XOR เพื่อลดปริมาณหน่วยความจำด้วยการถ่ายโอนหน่วยความจำเฉพาะที่แตกต่างเท่านั้นผ่านเครือข่าย พบว่า การอพยพแบบคงสถานะการทำงานของคอมพิวเตอร์เสมือนแบบ TLM-XZ ผ่านเครือข่ายที่มีแบนด์วิดท์ 100 Mbps มีประสิทธิภาพดีกว่าแบบ TLM-Z ในบางโปรแกรมเพียงเล็กน้อย แต่ทำให้ประสิทธิภาพของบางโปรแกรมแย่ลงค่อนข้างมาก ดังนั้น การเข้ารหัสแบบเดลตาด้วยวิธี XOR จึงไม่ทำให้การบีบอัดข้อมูลด้วย LZ4 มีประสิทธิภาพมากขึ้น เนื่องจากการเข้ารหัสข้อมูลแบบเดลตาด้วยการวิธี XOR มีค่าภาระงาน (overhead) ที่เพิ่มขึ้น อีกทั้งการบีบอัดข้อมูลของ LZ4 เป็นการบีบอัดแบบใช้ dictionary ในการค้นหาการเกิดรูปแบบที่ซ้ำ การเข้ารหัสข้อมูลแบบเดลตาด้วยการวิธี XOR น่าจะทำให้รูปแบบการเกิดซ้ำของข้อมูลลดลง

ผลจากการวิเคราะห์ข้อมูลด้วยวิธี XOR พบว่า หน่วยความจำที่เปลี่ยนแปลงมีลักษณะไม่แตกต่างจากหน่วยความจำเดิมมากนัก อีกทั้งหน่วยความจำที่เทรด dtx ถ่ายโอนสำหรับบางโปรแกรมนั้นแทบจะไม่มีการเปลี่ยนแปลงเลย และบางโปรแกรมมีการเปลี่ยนแปลงซ้ำที่เพจเดิมโดยรวมแล้วไม่ถึง 50% อย่างไรก็ตาม งานวิจัยนี้ไม่ได้ออกแบบให้ Memory Server รับหน่วยความจำจากเทรด mtx การใช้กลไกในการเปรียบเทียบหน่วยความจำจึงยังไม่สามารถทำได้อย่างมีประสิทธิภาพมากเพียงพอ ดังนั้นจึงยังคงมีการถ่ายโอนหน่วยความจำซ้ำๆ ผ่านเครือข่าย นอกจากนี้ การวิเคราะห์หน่วยความจำด้วยวิธี XOR ทำให้ทราบว่า หน่วยความจำมีความแตกต่างกันมากน้อยแค่ไหนและหน่วยความจำมีการเปลี่ยนแปลงที่เพจเดิมซ้ำมากน้อยเพียงใดในแต่ละโปรแกรม และหน่วยความจำนั้นมีค่า 0 อยู่ติดกันเป็นจำนวนมากเท่าไหร่ ซึ่งทำให้ทราบลักษณะของข้อมูลอย่างคร่าวๆ ได้ สามารถนำข้อมูลที่ได้นี้ไปต่อยอดพัฒนาประสิทธิการอพยพแบบคงสถานะการทำงานได้

จากการทดลองการบีบอัดข้อมูลด้วย Blosc เพื่อลดปริมาณหน่วยความจำที่ถ่ายโอนผ่านเครือข่าย พบว่า การบีบอัดข้อมูลในระดับหน่วยความจำเพจด้วย Blosc ช่วยทำให้ประสิทธิภาพการอพยพแบบคงสถานะการทำงานดีขึ้นเมื่ออพยพคอมพิวเตอร์เสมือนผ่านเครือข่าย 100 Mbps และวิธีบีบอัดที่เหมาะสมกับการอพยพนั้นควรเน้นที่ความเร็วในการบีบอัดเป็นหลัก เนื่องจากหน่วยความจำในระดับเพจมีขนาดเพียง 4 KB เท่านั้น ซึ่งเป็นข้อมูลที่มีขนาดเล็กสำหรับการบีบอัด การเกิดรูปแบบที่ซ้ำมีน้อยกว่าข้อมูลขนาดใหญ่ ทำให้ไม่สามารถบีบอัดข้อมูลได้อย่างมีประสิทธิภาพ ในการทดลองยังพบว่า มีหน่วยความจำเพจจำนวนมากที่เมื่อถูกบีบอัดแล้วมีขนาดใหญ่กว่าขนาดเดิม ทำให้เสียเวลาในการบีบอัด อีกทั้งการเพิ่มจำนวนเทรดเพื่อทำหน้าที่ในการบีบอัด เมื่อข้อมูลมีลักษณะที่ไม่เอื้อต่อการบีบอัด ทำให้มีภาระงานที่เกิดขึ้นมาอย่างไรประโยชน์ อย่างไรก็ตาม การใช้ filter เช่น

shuffle และ bitshuffle ซึ่งใช้ความสามารถจากสถาปัตยกรรมฮาร์ดแวร์ ในการจัดเรียงข้อมูลก่อนการบีบอัดสามารถช่วยให้ประสิทธิภาพของการบีบอัดดีขึ้น

5.3 แนวทางการวิจัยในอนาคต

เนื่องจากงานวิจัยนี้เพิ่มการบีบอัดในระดับหน่วยความจำเพจด้วย Blosc สำหรับช่วงที่คอมพิวเตอร์เสมือนหยุดการทำงานเท่านั้น ดังนั้นการนำเทคนิคการบีบอัดด้วย Blosc มาเพิ่มในช่วงที่ dtx ถ่ายโอนข้อมูลน่าจะช่วยทำให้ประสิทธิภาพของการอพยพดีขึ้น อีกทั้งการออกแบบให้เทรด mtx เปิดช่องทางการถ่ายโอนข้อมูลของ non-dirty pages ผ่านเข้ามาใน Memory Server เพื่อให้มีข้อมูลมากเพียงพอในการเปรียบเทียบ และใช้วิธีการจัดลำดับในการถ่ายโอนเพจใหม่ (page reordering) โดยการกำหนด page priority ให้เพจที่มีการเปลี่ยนแปลงซ้ำบ่อยครั้ง ถูกถ่ายโอนไปทีหลังเพื่อลดการถ่ายโอนข้อมูลซ้ำผ่านเครือข่าย นอกจากนี้ มีวิธีการบีบอัดแบบใหม่ที่เน้นความเร็วในการบีบอัดมากขึ้นอย่างเช่น LzTurbo (LzTurbo - World’s fastest compressor, 2015) หรือ Zstd (Zstandard - Fast real-time compression algorithm, n.d.) ซึ่งในอนาคตจะถูกนำมาเพิ่มเป็นอีกวิธีบีบอัดของ Blosc หรือใช้ bloscpack มาแทน LZ4 สำหรับการบีบอัดของเครื่องแม่ข่าย Z-server ได้เช่นกัน

รายการอ้างอิง

หนังสือและบทความในหนังสือ

ลัญฉกร วุฒิสิทธิกุลกิจ และคณะ. (2549). *เทคโนโลยีการบีบอัดข้อมูลเบื้องต้น*. กรุงเทพฯ: จุฬาลงกรณ์มหาวิทยาลัย.

บทความวารสาร

จอมภพ แววศักดิ์. (2549) พลศาสตรของไหลเชิงคำนวณ. *วารสารวิทยาศาสตรทักษิณ*, 3(1), 32-42

วิทยานิพนธ์

พิทักษ์ แท่นแก้ว. (2557). ทีแอลเอ็มแซด: ไลฟ์ไมเกรชันแบบเทรดบนเวอร์ชวลแมชชีนโดยใช้เครื่องแม่ข่ายสำหรับบีบอัดข้อมูล. (วิทยานิพนธ์ปริญญามหาบัณฑิต). มหาวิทยาลัยธรรมศาสตร์, คณะวิทยาศาสตร์และเทคโนโลยี.

ธวัชชัย มัทนัง. (2556). การเปรียบเทียบประสิทธิภาพของการส่งข้อมูลบนโครงข่ายก่อนและหลังการบีบอัดข้อมูล. (วิทยานิพนธ์ปริญญามหาบัณฑิต). มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าธนบุรี, คณะเทคโนโลยีสารสนเทศ.

สื่ออิเล็กทรอนิกส์

การบีบอัดข้อมูลตามขั้นตอนวิธีของ Jacob Ziv และ Abraham Lempel (LZ77). (n.d.) สืบค้นเมื่อวันที่ 20 มิถุนายม 2559 จาก http://angsila.cs.buu.ac.th/~jira/CSunplugged/Activity3/LZ77.doc

ชนิดา เรืองศิริวัฒนกุล. (n.d.) *เทคโนโลยีการบีบอัดข้อมูล Data Compression.* เอกสารประกอบการสอนวิชาเทคโนโลยีสารสนเทศ, อุตรดิตถ์ : มหาวิทยาลัยราชภัฏอุตรดิตถ์. สืบค้นจาก http://202.29.52.57/~chanida/Multimedia/Slide/chapter_11_data-compress.ppt

เทคโนโลยี Virtualization. (n.d.) สืบค้นเมื่อวันที่ 6 กุมภาพันธ์ 2559, จาก http://www.mat.co.th/th/products/si_consulting/virtual/

อนุพงษ์ บรรจงการ, วิทยา คงหาเพชร, ศุภกิจ พฤกษอรุณ, วรา วราวิทย์ และศรเทพ วรรณรัตน์. (2551). *การศึกษาประสิทธิภาพการอพยพระบบคอมพิวเตอร์ใน Xen สำหรับคำนวณผลประสิทธิภาพสูงพื้นฐาน*. The 12th National Computer Science and Engineering Conference, ประจำปี พ.ศ. 2551 (หน้า 505-511). พัทยา: NCSEC. สืบค้นจาก http://www.ee.kmutnb.ac.th/files/NCSEC/P182_Preliminary%20Study%20of%20Xen%20Migration%20Performance%20for%20High%20Performance%20Computing%20Applications.pdf

อดิศร ขาวสังข์. (2556) *แนะนำ KVM Virtualization ใน Linux* สืบค้นเมื่อวันที่ 6 กุมภาพันธ์ 2559, จาก http://www.itmanage.info/technology/VM_Cloud/KVM_introduction/KVM_Introduction.pdf

Icesoluction. (n.d.). Virtualization คืออะไร. สืบค้นเมื่อวันที่ 6 กุมภาพันธ์ 2559, จาก http://www.icesolution.com/th/solutions/ผลิตภัณฑ์-โซลูชั่น-ซิมพลีเวอร์ชวลไลเซชั่น-ไอซ์โซลูชั่น/318-what-is-virtualization.html

Sira Nokyoongthong. (2552) *มารู้จักเทคโนโลยี Virtualization* สืบค้นเมื่อวันที่ 6 กุมภาพันธ์ 2559, จาก http://thaiopensource.org/มารู้จักเทคโนโลยี-virtualization/

Articles

Chanchio, K., & Thaenkaew, P. (2014). Time-Bound, Thread-Based Live Migration of Virtual Machines. *The 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* (pp. 364-373).

Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I., & Warfield, A. (2005). Live Migration of Virtual Machines. *The 2nd Conference on Symposium on Networked Systems Design & Implementation Volume 2* (pp. 273-286).

Pountain D. (1987). Run-length encoding. Byte, 12(6): (pp. 317-319).

Svärd, P., Hudzia, B., Tordsson, J., & Elmroth, E. (2011). Evaluation of Delta Compression Techniques for Efficient Live Migration of Large Virtual Machines.

Proceedings of *The 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments* (pp. 111-120).

Svärd, P., Tordsson, J., Hudzia, B., & Elmroth, E. (2011). High performance live migration through dynamic page transfer reordering and compression. In Proc. of 2011 3rd IEEE International Conference on Cloud Computing Technology and Science (pp. 542-548).

Electronic Media

Alted, F. (2014). Blosc Talk from PyData. Retrieved May 5, 2016, from http://www.slideshare.net/PyData/blosc-py-data-2014

Alted, F. (2015). New 'bitshuffle' filter. Retrieved May 5, 2016, from http://blosc.org/blog/new-bitshuffle-filter.html

Alted, F. (2016). Blosc, an extremely fast, multi-threaded, meta-compressor library. Retrieved April 09 2016, from http://www.blosc.org/

An Overview of Virtualization Techniques. (n.d., updated 2016 May 29.). Retrieved June 06, 2016, from http://www.virtuatopia.com/index.php/An_Overview_of_Virtualization_Techniques

Binary delta compression. (n.d.). Retrieved February 6, 2016, from https://en.wikipedia.org/wiki/Binary_delta_compression

Chanchio K. (2016, Jan 12). How to read TLMZ performance report. Retrieved January 14 2016, from http://sciencecloud-community.cs.tu.ac.th/?p=198

Collet Y. (2015, Jun 29). LZ4 - Extremely fast compression. GitHub. Retrieved March 27 2016, from https://github.com/Cyan4973/lz4

Collet Y. (2015). Zstandard - Fast real-time compression algorithm. GitHub. Retrieved June 6, 2016, from https://github.com/Cyan4973/zstd

Data compression ratio. (n.d.). In Wikipedia, the free encyclopedia. Retrieved March 27 2016, from https://en.wikipedia.org/wiki/Data_compression_ratio

Dunbar, J. (n.d., updated 2016 March 21). NASA Advanced Supercomputing Division. (n.d.). NAS Parallel Benchmarks. Retrieved March 27, 2016, from http://www.nas.nasa.gov/publications/npb.html

Google. (n.d.). Snappy, a fast compressor/decompressor. GitHub. Retrieved March 27, 2016, from https://github.com/google/snappy

Hidayat, A. (n.d.). FastLZ - lightning-fast compression library Retrieved March 27, 2016, from http://fastlz.org/

Hudzia, B. (2011). Enhancing Live Migration Process for CPU and/or memory intensive VMs running Enterprise applications. Retrieved February 6, 2016, from http://www.linux-kvm.org/images/c/cb/2011-forum-kvm_hudzia.pdf

LZ4 Block Format Description. (n.d.). Retrieved February 6, 2016, from https://github.com/Cyan4973/lz4/blob/master/lz4_Block_format.md

LZ4 (compression algorithm) . (n.d.). Retrieved February 6, 2016, from https://en.wikipedia.org/wiki/LZ4_(compression_algorithm)

LZ4 explained. (n.d.). Retrieved February 6, 2016, from http://fastcompression.blogspot.com/2011/05/lz4-explained.html

LzTurbo - World's fastest compressor. (2015) Retrieved June 6, 2016, from https://sites.google.com/site/powturbo/

Shah, A. (2015, March 24). Live Migrating QEMU-KVM Virtual Machines. Retrieved June 6, 2016, from http://developers.redhat.com/blog/2015/03/24/live-migrating-qemu-kvm-virtual-machines/

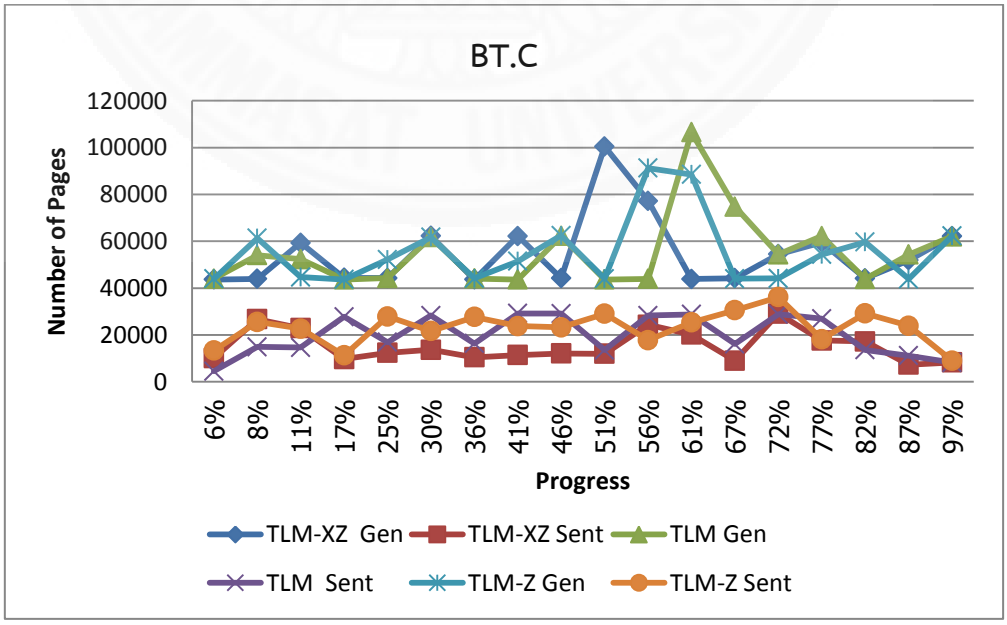
ภาคผนวก

ภาคผนวก ก
รูปแบบการใช้หน่วยความจำของเบนมาร์ก

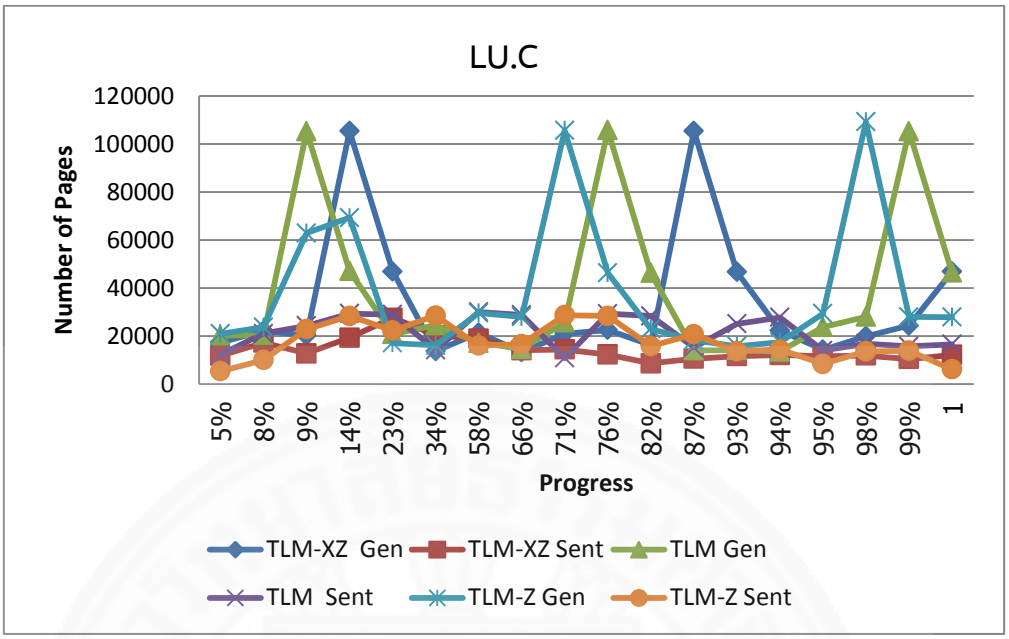
โปรแกรม NPB ที่ใช้ในการทดลองเป็นโปรแกรมทางวิทยาศาสตร์ที่ใช้หน่วยประมวลผลและหน่วยความจำมาก ในขณะที่ในการทดลองนี้อพยพคอมพิวเตอร์เสมือนขณะที่คอมพิวเตอร์เสมือนนั้นทำงานอยู่ โดยอพยพผ่านเครือข่ายแบนด์วิดท์ 1 Gbps และ 100 Mbps ดังนั้นเพื่อให้เห็นภาพผู้วิจัยจึงแนบภาพประกอบการรูปแบบการใช้หน่วยความจำของแต่ละโปรแกรมเบนมาร์ก และการหน่วยความจำเพจที่ถูกถ่ายโอนในขณะอพยพคอมพิวเตอร์เสมือนเฉพาะเครือข่าย 1 Gpbs เนื่องจากรูปแบบการใช้หน่วยความจำนั้นจะเหมือนกัน จะแตกต่างกันเพียงแค่ปริมาณหน่วยความจำที่ถูกถ่ายโอนสำหรับแบนด์วิดท์ 100 Mbps จะน้อยกว่าเท่านั้น

ในการอพยพแบบคงสถานะการทำงานของคอมพิวเตอร์เสมือนแบบเทรด (TLM) จะถ่ายโอนหน่วยความจำเพจของคอมพิวเตอร์เสมือนตั้งแต่เพจที่ 0 ถึงเพจสุดท้ายแล้วจึงหยุดการทำงาน ดังนั้นในภาพประกอบจึงแสดงความคืบหน้า (Progress) ตัวอย่างเช่น 30% นั้นแสดงว่า ได้มีการอพยพไปแล้ว 30% เมื่อถ่ายโอนไปจนครบคือ 100% คอมพิวเตอร์เสมือนจะถูกสั่งให้หยุดการทำงาน

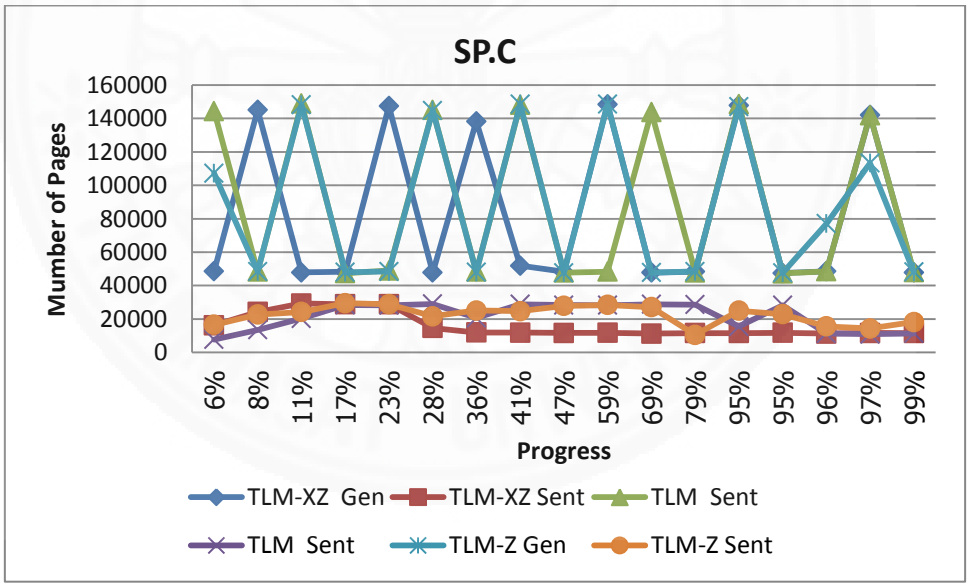
ส่วน Gen หมายถึง จำนวนของหน่วยความจำเพจที่เปลี่ยนแปลงที่เกิดขึ้นใหม่จากการประมวลผลของคอมพิวเตอร์เสมือน (Chanchio, 2016). ส่วน Sent หมายถึง หน่วยความจำเพจที่ถูกถ่ายโอนไปยังคอมพิวเตอร์เสมือนปลายทาง



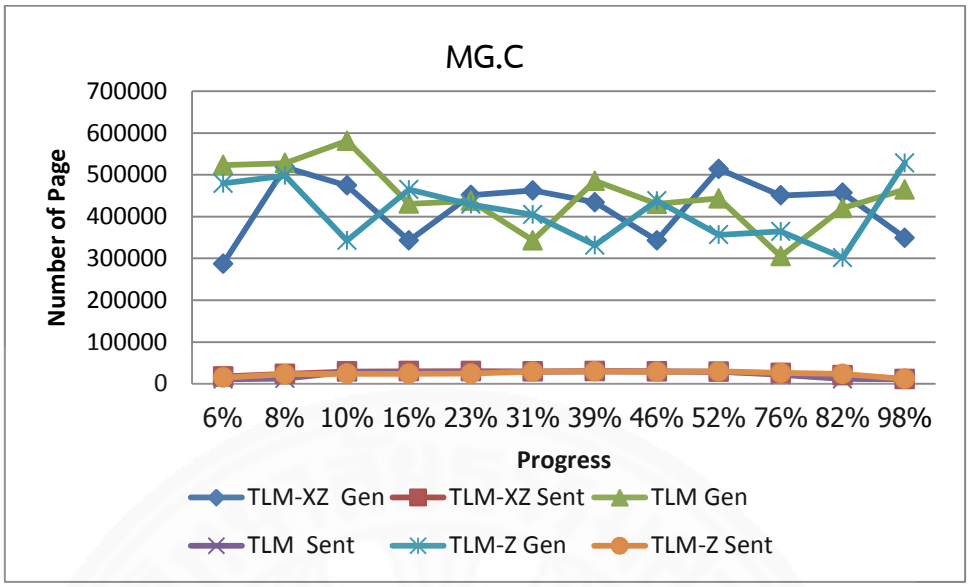
ภาพที่ ก.1 แสดง Memory access Pattern และหน่วยความจำที่ถูกถ่ายโอนของโปรแกรม BT.C



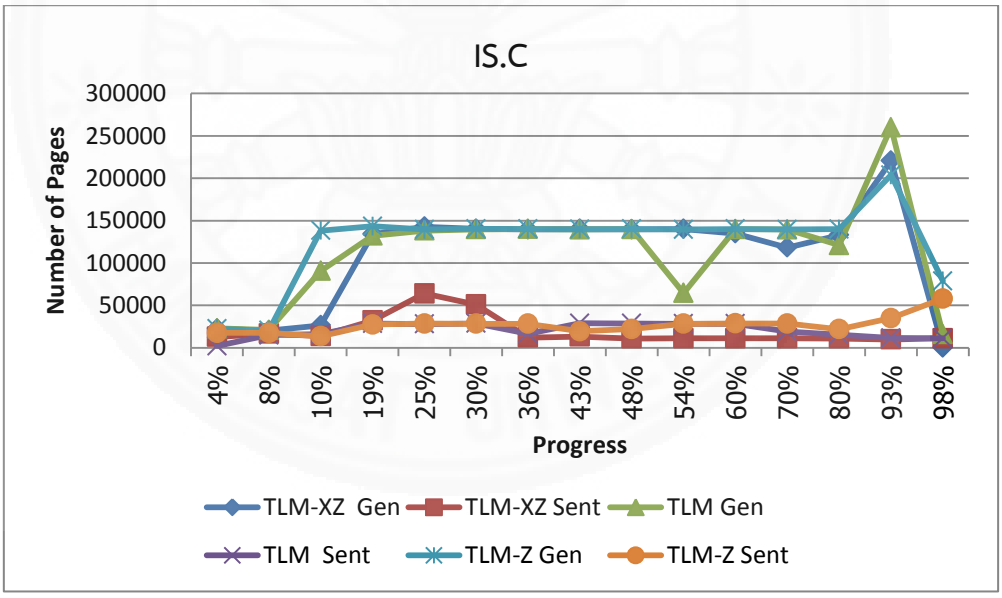
ภาพที่ ก.2 แสดง Memory access Pattern และหน่วยความจำที่ถูกถ่ายโอนของโปรแกรม LU.C



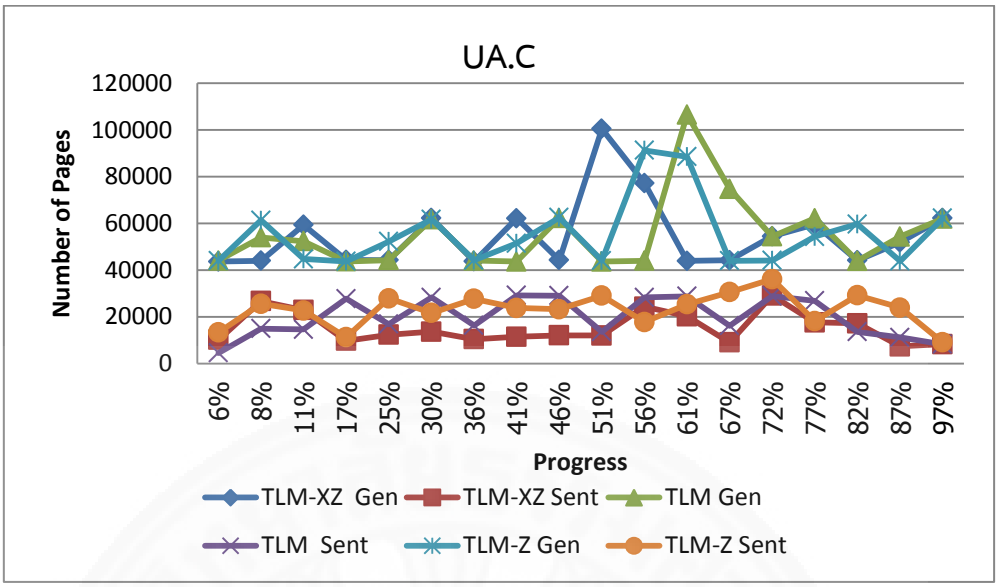
ภาพที่ ก.3 แสดง Memory access Pattern และหน่วยความจำที่ถูกถ่ายโอนของโปรแกรม SP.C



ภาพที่ ก.4 แสดง Memory access Pattern และหน่วยความจำที่ถูกถ่ายโอนของโปรแกรม MG.C



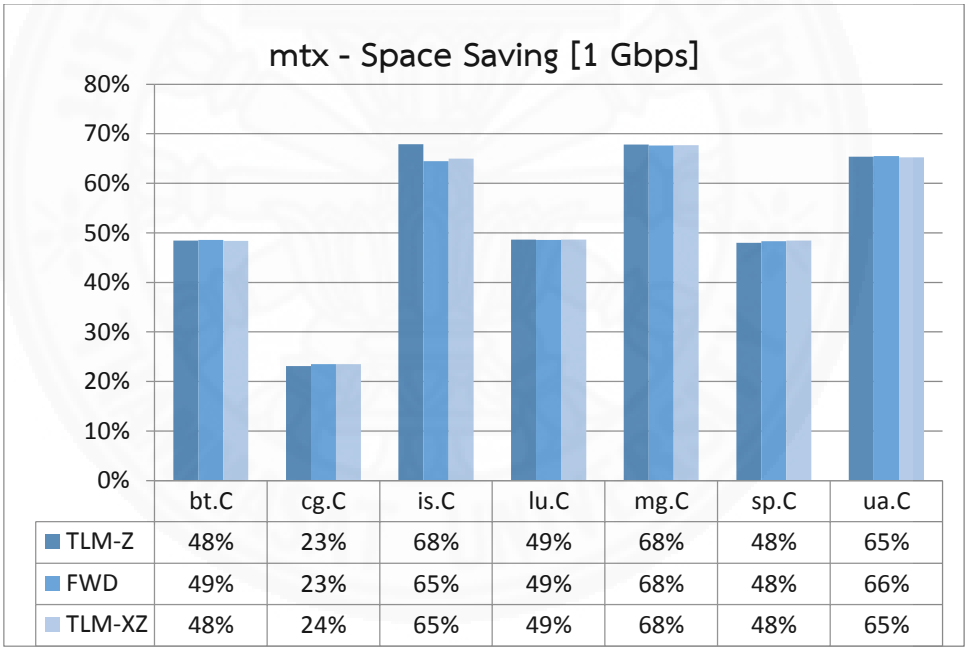
ภาพที่ ก.5 แสดง Memory access Pattern และหน่วยความจำที่ถูกถ่ายโอนของโปรแกรม IS.C



ภาพที่ ก.6 แสดง Memory access Pattern และหน่วยความจำที่ถูกถ่ายโอนของโปรแกรม UA.C

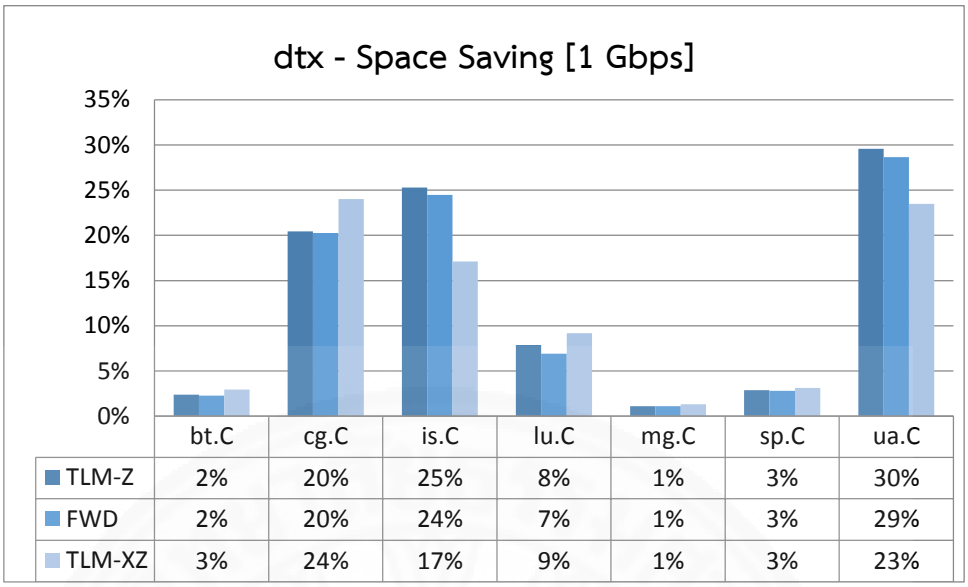
ภาคผนวก ข
ประสิทธิภาพการบีบอัดข้อมูลของ LZ4 สำหรับการ Forward

ในงานวิจัยนี้ได้สร้าง Framework แบบใหม่ขึ้นด้วยการเพิ่ม Memory Server และกำหนดให้เทรด dtx และเทรด io ในช่วงที่คอมพิวเตอร์เสมือนหยุดการทำงาน (S3) ถ่ายโอนข้อมูลจากคอมพิวเตอร์เสมือนต้นทางไปยัง Memory Server ดังนั้น ผู้วิจัยจึงทดสอบประสิทธิภาพของ LZ4 ในกรณีที่ Memory Server ทำหน้าที่เพียงถ่ายโอนข้อมูลต่อ (Forward) ไปยังเครื่องแม่ข่าย Z-Server ซึ่งมี LZ4 ทำหน้าที่บีบอัดข้อมูลหน่วยความจำเพื่อดูผลกระทบที่เกิดขึ้น วัดประสิทธิภาพด้วย space saving โดยวัดประสิทธิภาพเพียงการอพยพผ่านเครือข่าย 1 Gbps เท่านั้น โดยวัดเปรียบเทียบกับ TLM-Z และ TLM-XZ



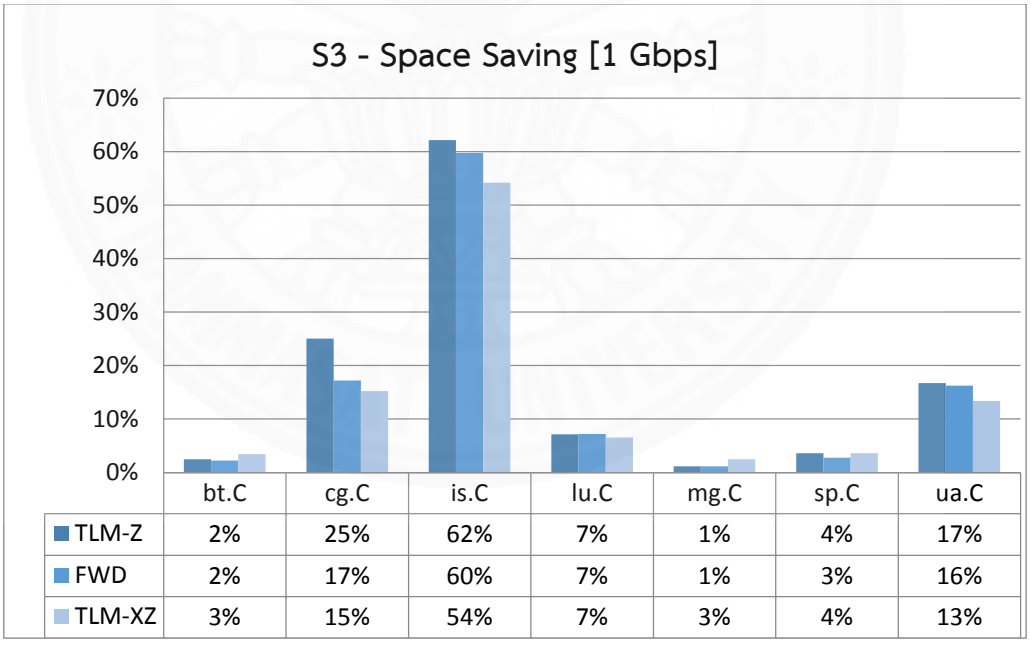
	bt.C	cg.C	is.C	lu.C	mg.C	sp.C	ua.C
TLM-Z	48%	23%	68%	49%	68%	48%	65%
FWD	49%	23%	65%	49%	68%	48%	66%
TLM-XZ	48%	24%	65%	49%	68%	48%	65%

ภาพที่ ข.1 เปรียบเทียบประสิทธิภาพการบีบอัดข้อมูลของ LZ4 ของเทรด mtx สำหรับการอพยพแบบคงสถานะการทำงานแบบ TLM-Z, FWD และ TLM-XZ ผ่านเครือข่ายแบนด์วิดท์ 1 Gbps



	bt.C	cg.C	is.C	lu.C	mg.C	sp.C	ua.C
TLM-Z	2%	20%	25%	8%	1%	3%	30%
FWD	2%	20%	24%	7%	1%	3%	29%
TLM-XZ	3%	24%	17%	9%	1%	3%	23%

ภาพที่ ข.2 เปรียบเทียบประสิทธิภาพการบีบอัดข้อมูลของ LZ4 ของเทรด dtx สำหรับการอพยพแบบคงสถานะการทำงานแบบ TLM-Z, FWD และ TLM-XZ ผ่านเครือข่ายแบนด์วิดท์ 1 Gbps



	bt.C	cg.C	is.C	lu.C	mg.C	sp.C	ua.C
TLM-Z	2%	25%	62%	7%	1%	4%	17%
FWD	2%	17%	60%	7%	1%	3%	16%
TLM-XZ	3%	15%	54%	7%	3%	4%	13%

ภาพที่ ข.3 เปรียบเทียบประสิทธิภาพการบีบอัดข้อมูลของ LZ4 ของช่วง S3 สำหรับการอพยพแบบคงสถานะการทำงานแบบ TLM-Z และ TLM-XZ ผ่านเครือข่ายแบนด์วิดท์ 1 Gbps

ภาคผนวก ค

ผลการทดลองการอพยพแบบ TLM-BZ

ตาราง ค.1 เปรียบเทียบประสิทธิภาพด้วย Downtime ของ TLM, TLM-Z และ TLM-BZ บนเครือข่ายที่มีแบนด์วิดท์ 1 Gbps โดยการเรียงลำดับการบีบอัดแบบไบต์ก่อนการบีบอัด

Benchmark	Downtime [1gbps] (Time in Sec.)							
	TLM	TLM-Z	TLM-XZ	TLM-BZ**				
				blosclz	lz4	lz4hc	snappy	zlib
BT.C	4.66	4.26	4.09	5.95	4.16	34.27	3.77	120.41
CG.C	0.11	0.04	0.06	0.07	0.10	0.49	0.03	1.29
IS.C	7.54	5.30	6.08	6.95	5.54	51.56	5.26	151.30
LU.C	3.31	3.05	3.25	2.89	3.88	21.63	2.64	96.24
M.G.C	21.24	20.98	20.76	29.58	20.81	182.20	18.29	578.24
SP.C	5.05	4.62	4.82	6.57	4.43	35.77	4.25	114.04
UA.C	2.14	1.96	1.97	2.69	2.10	17.93	2.13	59.34

** TLM-BZ Parameter: clevel6, byteshuffle, typesize 8, 2threads

ตาราง ค.2 เปรียบเทียบประสิทธิภาพด้วย Downtime ของ TLM, TLM-Z และ TLM-BZ บนเครือข่ายที่มีแบนด์วิดท์ 1 Gbps โดยไม่เรียงลำดับก่อนการบีบอัด

Benchmark	Downtime [1gbps] (Time in Sec.)							
	TLM	TLM-Z	TLM-XZ	TLM-BZ**				
				blosclz	lz4	lz4hc	snappy	zlib
BT.C	4.66	4.26	4.09	5.92	4.39	34.53	4.41	130.09
CG.C	0.11	0.04	0.06	0.09	0.05	34.53	4.41	1.28
IS.C	7.54	5.30	6.08	7.57	7.14	0.48	0.05	171.41
LU.C	3.31	3.05	3.25	4.10	3.31	56.91	6.49	101.20
M.G.C	21.24	20.98	20.76	29.40	21.17	167.25	21.52	590.23
SP.C	5.05	4.62	4.82	6.08	4.89	38.60	4.24	120.46
UA.C	2.14	1.96	1.97	2.47	1.90	16.86	2.13	70.38

** TLM-BZ Parameter: clevel6, noshuffle, typesize 8, 2threads

ตาราง ค.3 เปรียบเทียบประสิทธิภาพด้วย Downtime ของ TLM, TLM-Z และ TLM-BZ บนเครือข่ายที่มีแบนด์วิดท์ 1 Gbps โดยเรียงลำดับข้อมูลแบบบิตก่อนการบีบอัดและใช้เทรด 4 เทรด

Benchmark	Downtime [1gbps] (Time in Sec.)							
	TLM	TLM-Z	TLM-XZ	TLM-BZ**				
				blosclz	lz4	lz4hc	snappy	zlib
BT.C	4.66	4.26	4.09	6.22	4.61	35.76	4.22	134.77
CG.C	0.11	0.04	0.06	0.06	0.08	0.58	0.14	1.36
IS.C	7.54	5.30	6.08	7.65	6.38	67.21	5.78	159.79
LU.C	3.31	3.05	3.25	5.12	4.39	29.48	2.79	141.91
M.G.C	21.24	20.98	20.76	31.87	23.02	177.92	17.89	580.55
SP.C	5.05	4.62	4.82	7.34	5.01	36.90	4.46	128.58
UA.C	2.14	1.96	1.97	3.08	2.30	15.73	1.97	60.84

** TLM-BZ Parameter: clevel6, bitshuffle, typesize 8, 4threads

ตาราง ค.4 เปรียบเทียบประสิทธิภาพด้วย Total Migration Time ของ TLM, TLM-Z และ TLM-BZ บนเครือข่ายที่มีแบนด์วิดท์ 1 Gbps โดยเรียงลำดับข้อมูลแบบไบต์ก่อนการบีบอัด

Benchmark	Total Migration Time [1gbps] (Time in sec.)							
	TLM	TLM-Z	TLM-XZ	TLM-BZ**				
				blosclz	lz4	lz4hc	snappy	zlib
BT.C	25.51	24.28	23.05	25.49	23.45	54.82	23.18	139.28
CG.C	30.16	43.19	29.12	28.78	28.87	29.46	28.81	31.10
IS.C	25.64	22.36	23.04	23.70	22.21	68.63	22.12	168.69
LU.C	25.74	25.69	21.90	24.32	25.04	43.50	23.61	115.18
M.G.C	37.71	34.95	34.89	43.47	34.66	196.10	32.28	593.58
SP.C	26.22	25.93	24.88	26.57	24.63	55.47	23.95	134.73
UA.C	25.04	22.81	22.78	23.02	20.85	38.78	22.57	81.14

** TLM-BZ Parameter: clevel6, byteshuffle, typesize 8, 2threads

ตาราง ค.5 เปรียบเทียบประสิทธิภาพด้วย Total Migration Time ของ TLM, TLM-Z และ TLM-BZ บนเครือข่ายที่มีแบนด์วิดท์ 1 Gbps โดยไม่เรียงลำดับข้อมูลก่อนการบีบอัด

Benchmark	Total Migration Time [1gbps] (Time in sec.)							
	TLM	TLM-Z	TLM-XZ	TLM-BZ**				
				blosclz	lz4	lz4hc	snappy	zlib
BT.C	25.51	24.28	23.05	25.08	23.81	54.67	23.82	140.30
CG.C	30.16	43.19	29.12	29.26	28.55	30.14	29.25	33.29
IS.C	25.64	22.36	23.04	24.45	23.98	73.81	23.33	170.39
LU.C	25.74	25.69	21.90	25.76	25.09	48.56	23.67	118.38
M.G.C	37.71	34.95	34.89	43.93	35.57	181.83	35.09	600.92
SP.C	26.22	25.93	24.88	25.62	25.10	58.85	24.57	143.76
UA.C	25.04	22.81	22.78	22.77	22.60	37.89	22.16	90.67

** TLM-BZ Parameter: clevel6, noshuffle, typesize 8, 2threads

ตาราง ค.6 เปรียบเทียบประสิทธิภาพด้วย Total Migration Time ของ TLM, TLM-Z และ TLM-BZ บนเครือข่ายที่มีแบนด์วิดท์ 1 Gbps โดยเรียงลำดับข้อมูลแบบบิตก่อนการบีบอัด และใช้เทรด 4 เทรด

Benchmark	Total Migration Time [1gbps] (Time in sec.)							
	TLM	TLM-Z	TLM-XZ	TLM-BZ**				
				blosclz	lz4	lz4hc	snappy	zlib
BT.C	25.51	24.28	23.05	25.91	23.99	54.80	23.13	153.09
CG.C	30.16	43.19	29.12	28.94	28.80	30.03	29.37	29.61
IS.C	25.64	22.36	23.04	24.51	23.68	84.10	22.55	177.33
LU.C	25.74	25.69	21.90	25.98	24.72	51.75	25.38	160.51
M.G.C	37.71	34.95	34.89	46.38	37.09	192.10	31.98	594.55
SP.C	26.22	25.93	24.88	27.59	24.57	56.51	23.75	148.52
UA.C	25.04	22.81	22.78	22.41	23.08	35.45	22.53	82.49

** TLM-BZ Parameter: clevel6, bitshuffle, typesize 8, 4threads

ตาราง ค.7 เปรียบเทียบประสิทธิภาพด้วย Downtime ของ TLM, TLM-Z และ TLM-BZ บนเครือข่ายที่มีแบนด์วิดท์ 100 Mbps โดยเรียงลำดับข้อมูลแบบไบต์ก่อนการบีบอัด

Benchmark	Downtime [100mbps] (Time in Sec.)							
	TLM	TLM-Z	TLM-XZ	TLM-BZ**				
				blosclz	lz4	lz4hc	snappy	zlib
BT.C	46.67	45.59	44.55	38.07	38.28	38.23	40.15	128.79
CG.C	0.62	0.29	0.19	0.32	0.23	0.50	0.19	1.65
IS.C	72.83	31.52	39.77	22.22	22.48	57.53	22.66	159.10
LU.C	38.40	37.79	35.21	30.70	30.93	31.93	32.20	99.65
M.G.C	212.31	208.21	208.14	172.65	172.68	187.24	180.05	567.11
SP.C	51.88	49.94	49.06	42.02	41.90	44.92	45.13	141.43
UA.C	40.75	20.15	21.73	20.11	19.78	33.19	20.44	95.51

** TLM-BZ Parameter: clevel6, byteshuffle, typesize 8, 2threads

ตาราง ค.8 เปรียบเทียบประสิทธิภาพด้วย Downtime ของ TLM, TLM-Z และ TLM-BZ บนเครือข่ายที่มีแบนด์วิดท์ 100 Mbps โดยไม่เรียงลำดับข้อมูลก่อนการบีบอัด

Benchmark	Downtime [100mbps] (Time in Sec.)							
	TLM	TLM-Z	TLM-XZ	TLM-BZ**				
				blosclz	lz4	lz4hc	snappy	zlib
BT.C	46.67	45.59	44.55	45.46	45.70	45.85	45.79	136.09
CG.C	0.62	0.29	0.19	0.44	0.48	0.73	0.17	1.64
IS.C	72.83	31.52	39.77	31.48	62.70	62.46	34.32	181.13
LU.C	38.40	37.79	35.21	72.48	34.99	37.96	33.72	107.17
M.G.C	212.31	208.21	208.14	208.53	208.74	208.80	208.34	618.78
SP.C	51.88	49.94	49.06	50.11	50.19	50.33	50.29	149.27
UA.C	40.75	20.15	21.73	20.12	35.52	35.99	20.15	99.65

** TLM-BZ Parameter: clevel6, noshuffle, typesize 8, 2threads

ตาราง ค.9 เปรียบเทียบประสิทธิภาพด้วย Downtime ของ TLM, TLM-Z และ TLM-BZ บนเครือข่ายที่มีแบนด์วิดท์ 100 Mbps โดยเรียงลำดับข้อมูลแบบบิตก่อนการบีบอัดและใช้เทรด 4 เทรด

Benchmark	Downtime [100mbps] (Time in Sec.)							
	TLM	TLM-Z	TLM-XZ	TLM-BZ**				
				blosclz	lz4	lz4hc	snappy	zlib
BT.C	46.67	45.59	44.55	38.98	39.35	39.09	39.55	130.66
CG.C	0.62	0.29	0.19	0.42	0.30	0.62	0.24	1.55
IS.C	72.83	31.52	39.77	20.94	19.38	67.03	18.87	166.56
LU.C	38.40	37.79	35.21	31.79	32.47	34.73	31.33	104.55
M.G.C	212.31	208.21	208.14	169.95	168.46	185.16	172.03	581.54
SP.C	51.88	49.94	49.06	43.09	43.28	43.50	44.20	142.44
UA.C	40.75	20.15	21.73	21.61	21.70	30.59	21.69	100.24

** TLM-BZ Parameter: clevel6, bitshuffle, typesize 8, 4threads

ตาราง ค.10 เปรียบเทียบประสิทธิภาพด้วย Total Migration Time ของ TLM, TLM-Z และ TLM-BZ บนเครือข่ายที่มีแบนด์วิดท์ 100 Mbps โดยเรียงลำดับข้อมูลแบบไบต์ก่อนการบีบอัด

Benchmark	Total Migration Time [100Mbps] (Time in sec.)							
	TLM	TLM-Z	TLM-XZ	TLM-BZ**				
				blosclz	lz4	lz4hc	snappy	zlib
BT.C	138.65	107.25	104.67	99.22	99.00	99.66	101.01	177.47
CG.C	194.58	174.15	154.71	166.97	168.01	169.33	166.28	166.08
IS.C	138.28	62.11	69.37	52.33	51.74	88.08	51.31	190.34
LU.C	130.59	97.26	94.39	94.92	94.73	95.84	91.81	158.59
M.G.C	278.68	238.03	236.43	201.96	202.60	217.18	208.28	598.49
SP.C	143.05	111.16	110.38	102.94	104.34	105.95	107.39	203.56
UA.C	115.31	58.87	60.46	57.81	55.74	69.82	58.72	133.22

** TLM-BZ Parameter: clevel6, byteshuffle, typesize 8, 2threads

ตาราง ค.11 เปรียบเทียบประสิทธิภาพด้วย Total Migration Time ของ TLM, TLM-Z และ TLM-BZ บนเครือข่ายที่มีแบนด์วิดท์ 100 Mbps โดยไม่เรียงลำดับข้อมูลก่อนการบีบอัด

Benchmark	Total Migration Time [100Mbps] (Time in sec.)							
	TLM	TLM-Z	TLM-XZ	TLM-BZ**				
				blosclz	lz4	lz4hc	snappy	zlib
BT.C	138.65	107.25	104.67	106.91	106.39	106.51	106.33	188.37
CG.C	194.58	174.15	154.71	167.85	167.51	168.09	165.86	171.20
IS.C	138.28	62.11	69.37	62.91	92.69	92.77	83.53	211.61
LU.C	130.59	97.26	94.39	87.39	96.39	96.66	75.31	168.76
M.G.C	278.68	238.03	236.43	237.35	237.09	238.10	237.21	649.74
SP.C	143.05	111.16	110.38	113.52	110.11	111.56	112.04	209.49
UA.C	115.31	58.87	60.46	53.65	69.78	74.49	59.02	137.40

** TLM-BZ Parameter: clevel6, noshuffle, typesize 8, 2threads

ตาราง ค.12 เปรียบเทียบประสิทธิภาพด้วย Total Migration Time ของ TLM, TLM-Z และ TLM-BZ บนเครือข่ายที่มีแบนด์วิดท์ 100 Mbps โดยเรียงลำดับข้อมูลแบบบิตก่อนการบีบอัดและใช้เทรด 4 เทรด

Benchmark	Total Migration Time [100Mbps] (Time in sec.)							
	TLM	TLM-Z	TLM-XZ	TLM-BZ**				
				blosclz	lz4	lz4hc	snappy	zlib
BT.C	138.65	107.25	104.67	95.78	99.91	97.51	100.09	191.44
CG.C	194.58	174.15	154.71	166.33	167.81	167.14	166.67	167.89
IS.C	138.28	62.11	69.37	49.11	49.62	95.55	49.38	196.56
LU.C	130.59	97.26	94.39	95.90	95.77	95.41	94.58	167.42
M.G.C	278.68	238.03	236.43	198.30	198.12	215.64	199.03	608.20
SP.C	143.05	111.16	110.38	101.68	103.78	103.83	105.81	204.08
UA.C	115.31	58.87	60.46	58.90	60.03	64.74	58.49	138.48

** TLM-BZ Parameter: clevel6, bitshuffle, typesize 8, 4threads

การวิเคราะห์ลักษณะหน่วยความจำด้วย XOR พบว่า มีหน่วยความจำเพจเวอร์ชันก่อนหน้าเหมือนกันกับเวอร์ชันปัจจุบันเลย ผู้วิจัยจึงลองทดสอบกลไก memcmp ซึ่งเป็นกลไกในการตรวจเช็คหน่วยความจำเพจที่เหมือนกันแทนการเข้ารหัสแบบเดลตาด้วย XOR

ตาราง ค.13 เปรียบเทียบประสิทธิภาพด้วย Downtime ของ TLM, TLM-Z และ TLM-BZ บนเครือข่ายที่มีแบนด์วิดท์ 1Gbps โดยเพิ่มกลไก memcmp

Benchmark	Downtime [1gbps] (Time in Sec.)							
	TLM	TLM-Z	TLM-XZ	TLM-BZ**				
				blosclz	lz4	lz4hc	snappy	zlib
BT.C	4.66	4.26	4.09	6.15	4.65	34.41	3.90	123.68
CG.C	0.11	0.04	0.06	0.05	0.07	0.44	0.04	1.29
IS.C	7.54	5.30	6.08	8.26	6.69	63.62	5.55	134.24
LU.C	3.31	3.05	3.25	3.36	3.07	24.48	2.74	56.44
M.G.C	21.24	20.98	20.76	31.32	23.69	178.87	18.93	594.62
SP.C	5.05	4.62	4.82	6.96	4.71	41.36	4.46	115.89
UA.C	2.14	1.96	1.97	2.86	2.06	14.08	1.92	63.02

ตาราง ค.14 เปรียบเทียบประสิทธิภาพด้วย Total Migration Time ของ TLM, TLM-Z และ TLM-BZ บนเครือข่ายที่มีแบนด์วิดท์ 1Gbps โดยเพิ่มกลไก memcmp

Benchmark	Total Migration Time [1gbps] (Time in sec.)							
	TLM	TLM-Z	TLM-XZ	TLM-BZ**				
				blosclz	lz4	lz4hc	snappy	zlib
BT.C	25.51	24.28	23.05	25.91	23.84	53.21	23.13	140.30
CG.C	30.16	43.19	29.12	28.80	29.15	29.43	28.75	33.29
IS.C	25.64	22.36	23.04	25.12	23.24	80.19	22.74	170.39
LU.C	25.74	25.69	21.90	25.17	24.72	46.49	24.67	118.38
M.G.C	37.71	34.95	34.89	45.50	37.66	192.44	33.21	600.92
SP.C	26.22	25.93	24.88	26.57	24.58	60.22	25.43	143.76
UA.C	25.04	22.81	22.78	23.84	21.59	34.00	21.83	90.67

** TLM-BZ Parameter: memcmp, clevel6, bitshuffle, typesize 8, 2threads

ตาราง ค.15 เปรียบเทียบประสิทธิภาพด้วย Downtime ของ TLM, TLM-Z และ TLM-BZ บนเครือข่ายที่มีแบนด์วิดท์ 100 Mbps โดยเพิ่มกลไก memcmp

Benchmark	Downtime [100mbps] (Time in Sec.)							
	TLM	TLM-Z	TLM-XZ	TLM-BZ**				
				blosclz	lz4	lz4hc	snappy	zlib
BT.C	46.67	45.59	44.55	38.65	38.88	39.13	59.58	127.07
CG.C	0.62	0.29	0.19	0.16	0.19	0.58	0.25	1.38
IS.C	72.83	31.52	39.77	19.61	20.21	72.58	18.12	165.14
LU.C	38.40	37.79	35.21	35.56	35.06	34.66	31.81	100.13
M.G.C	212.31	208.21	208.14	169.72	170.66	186.44	171.93	582.04
SP.C	51.88	49.94	49.06	42.87	42.58	42.65	43.92	136.87
UA.C	40.75	20.15	21.73	21.62	21.65	29.10	19.43	87.27

** TLM-BZ Parameter: memcmp, clevel6, bitshuffle, typesize 8, 2threads

ตาราง ค.16 เปรียบเทียบประสิทธิภาพด้วย Total Migration Time ของ TLM, TLM-Z และ TLM-BZ บนเครือข่ายที่มีแบนด์วิดท์ 100 Mbps โดยเพิ่มกลไก memcmp

Benchmark	Total Migration Time [100Mbps] (Time in sec.)							
	TLM	TLM-Z	TLM-XZ	TLM-BZ**				
				blosclz	lz4	lz4hc	snappy	zlib
BT.C	138.65	107.25	104.67	98.24	99.17	100.15	99.46	187.37
CG.C	194.58	174.15	154.71	160.52	159.84	160.63	158.61	162.54
IS.C	138.28	62.11	69.37	50.67	48.92	103.13	48.75	195.79
LU.C	130.59	97.26	94.39	92.97	90.11	94.98	93.64	164.91
M.G.C	278.68	238.03	236.43	199.92	198.44	216.37	199.94	611.90
SP.C	143.05	111.16	110.38	100.88	104.83	103.05	104.38	199.34
UA.C	115.31	58.87	60.46	59.34	60.54	66.95	57.09	125.36

** TLM-BZ Parameter: memcmp, clevel6, bitshuffle, typesize 8, 2threads

ประวัติผู้เขียน

ชื่อ	นางสาวสาวิตรี พันวินิต
วันเดือนปีเกิด	9 กันยายน 2521
วุฒิการศึกษา	ปีการศึกษา 2542: เศรษฐศาสตร์บัณฑิต มหาวิทยาลัยธรรมศาสตร์
ตำแหน่ง	Web Developer บริษัท Digitalendpoint Co.,Ltd.

ผลงานทางวิชาการ

สาวิตรี พันวินิต และกษิดิศ ชาญเชี่ยว. (2559). การวิเคราะห์ประสิทธิภาพการบีบอัดข้อมูลสำหรับการอพยพแบบคงสถานะการทำงานของคอมพิวเตอร์เสมือนแบบเทรด. การประชุมวิชาการและเสนอผลงานวิจัยระดับชาติและนานาชาติ ICMSIT 2016 : International Conference on Management Science, Innovation, and Technology 2016 ระหว่างวันที่ 17 มิถุนายน 2559, หน้า 103-114

ประสบการณ์ทำงาน

2556 – ปัจจุบัน ตำแหน่ง Web Developer
บริษัท Digitalendpoint Co., Ltd.
2554 – 2556 ตำแหน่ง Web Developer
บริษัท Asiamediastudio Co., Ltd.
2554 ตำแหน่ง Web Developer
บริษัท Brain Box Asia Co., Ltd.
2549-2550 ตำแหน่ง Web Developer
บริษัท Venda Software Development Co., Ltd.
2548 ตำแหน่ง Web Coordinator
บริษัท Net Bigstar Co., Ltd.
2543-2547 ตำแหน่ง Web Coordinator
บริษัท Nation Multimedia Group Public Co., Ltd.