ไลฟ์ไมเกรชั่นแบบเทรดบนเวอร์ชวลแมชชีนด้วยเทคนิคการเชื่อมต่อเครือข่ายแบบขนาน

Thread-based Live Migration of Virtual Machines Using Parallel Network Connections

พิทักษ์ แท่นแก้ว [1] และ กษิดิศ ชาญเชี่ยว [2]
ภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยธรรมศาสตร์
99 หมู่ 18 ถ.พหลโยธิน ต.คลองหนึ่ง อ.คลองหลวง จ.ปทุมธานี 12121 ประเทศไทย
Email: [1] 5209035194@student.cs.tu.ac.th, [2] kasidit@cs.tu.ac.th

บทคัดย่อ

บทความนี้นำเสนอวิธีการไลฟ์ไมเกรชั่นของเวอร์ชวลแมชชีนแบบใหม่เรียกว่า ทีเอลเอ็ม (TLM) ที่ทำให้การเคลื่อนย้ายเวอร์ชวลแมชชีนระหว่างเครื่องคอมพิวเตอร์จริงสองเครื่องเป็นไปได้อย่างมีประสิทธิภาพ วิธีการทีเอลเอ็มนำเสนอการออกแบบใหม่สองประการ ได้แก่ 1) การใช้เทรดเพื่อส่งสถานะของเวอร์ชวลแมชชีนไปยังเครื่องคอมพิวเตอร์ปลายทางในขณะที่เวอร์ชวลแมชชีนประมวลผล และ 2) มีการส่งข้อมูลแบบขนานเพื่อให้การเคลื่อนย้ายเวอร์ชวลแมชชีนเร็วขึ้น ระบบต้นแบบของทีเอลเอ็มได้รับพัฒนาขึ้นบนโอเพ่นซอร์สซอฟต์แวร์เควีเอ็ม (Kernel-based Virtual Machine) และทดสอบกับเวอร์ชวลแมชชีนขนาดใหญ่ที่กำลังประมวลผล NAS Paralllel Benchmark ผู้วิจัยพบว่าทีเอลเอ็มมีประสิทธิภาพมากกว่าวิธีการไมเกรชั่นเดิมของเควีเอ็มทั้งแบบไลฟ์ไมเกรชั่นและแบบหยุดแล้วไมเกรชั่นหลายเท่า

คำสำคัญ: เวอร์ชวลแมชชีน ไลฟ์ไมเกรชั่น การเชื่อมต่อเครือข่ายแบบขนาน

Abstract

This paper describes a novel Thread-based Live Migration (TLM) of virtual machines where additional threads are created to facilitate the state transfer of a virtual machine over parallel TCP connections concurrently with the virtual machine's execution. We have implemented TLM on the Open Source Kernel-based Virtual Machine (KVM) software and conducted preliminary experiments by migrating a large SMP virtual machine running computation and memory intensive NAS Parallel Benchmarks. The results indicate significantly better performance of TLM over the traditional live and non-live migration mechanisms.

keywords: virtual machine, live migration, parallel network connections

1. บทนำ

ในระบบประมวลผลแบบกลุ่มเมฆ (Cloud Computing) ซึ่งประกอบไปด้วย การใช้งานเวอร์ชวลแมชชีนจำนวนมาก ความสามารถในการไลฟ์ไมเกรชั่น หรือการเคลื่อนย้ายเวอร์ชวลแมชชีนในขณะที่เวอร์ชวลแมชชีนกำลังประมวลผลและมีการขัดจังหวะการทำงานของเวอร์ชวลแมชชีนน้อยที่สุดนั้นเป็นความสามารถที่เป็นประโยชน์หลายประการอาทิ เช่น

- เพิ่มประสิทธิภาพของแอพพลิเคชันที่ปฏิบัติงานอยู่บนเวอร์ชวลแมชชีนโดยไมเกรชั่นไปยังเครื่องคอมพิวเตอร์จริงที่มีประสิทธิภาพสูงขึ้น หรือไมเกรชั่นไปยังเครื่องที่อยู่ใกล้กับฐานข้อมูล
- การเพิ่มความทนทานต่อความผิดพร่อง (Fault Tolerance) โดยไมเกรชั่นจากเครื่องคอมพิวเตอร์ที่มีอุปกรณ์บางส่วนเสียหายไปยังเครื่องที่ทำงานปกติ
- ช่วยการบริหารการใช้งานทรัพยากรภายในดาต้าเซ็นเตอร์ให้มีประสิทธิภาพโดยไมเกรชั่นเวอร์ชวลแมชชีนที่อาจกระจายอยู่บนเครื่องคอมพิวเตอร์หลาย ๆ เครื่องให้ไปอยู่บนเครื่องเดียวเพื่อประหยัดพลังงาน

อย่างไรก็ตาม วิธีการไลฟ์ไมเกรชั่นในปัจจุบันยังมีข้อจำกัดและไม่สามารถปฏิบัติงานได้อย่างมีประสิทธิภาพเมื่อเวอร์ชวลแมชชีนเป็นแบบ SMP ที่ใช้ซีพียูคอร์แบบขนานและมีขนาดของหน่วยความจำขนาดใหญ่ซึ่งเป็นลักษณะของเวอร์ชวลแมชชีนที่มีให้บริการกับลูกค้าตามปกติในดาต้าเซ็นเตอร์ [1] ดังที่แสดงในตารางที่ 1 ซึ่งแบบ Extra Large และ Double และ Quadruple Extra Large นั้นใช้สำหรับแอพพลิเคชันที่เน้นการใช้งานฐานข้อมูลและการแคชข้อมูลในหน่วนความจำ ส่วนแบบ Cluster นั้นใช้ในการประมวลผลสมรรถนะสูง

การทำไลฟ์ไมเกรชั่นแบบเดิมจะใช้เวลานานมากหรืออาจใช้ไม่ได้เลยสำหรับเวอร์ชวลแมชชีนที่มีปริมาณการประมวลผลมากและมีขนาดของหน่วยความจำมาก ดังนั้นการไมเกรชั่นจึงต้องหันไปใช้วิธีหยุดการทำงานของเวอร์ชวลแมชชีนแล้วส่งสถานะไปยังคอมพิวเตอร์ปลายทางซึ่งเป็นทางเลือกที่ยอมรับได้สำหรับเวอร์ชวลแมชชีนที่รัน Batch Jobs แต่ไม่เหมาะสมกับเวอร์ชวลแมชชีนที่ทำงานเป็นฐานข้อมูลหรือแคชข้อมูลหรือแอพพลิเคชันที่ต้องการการตอบสนอง

ตารางที่ 1 ชนิดของเวอร์ชวลแมชชีนของบริษัท Amazon

Instant Types	No. of Vitual Core	Memory Size (Gigabytes)	Disk Space (Gigabytes)
Large	2	7.5	850
Extra Large	4	15	1,690
Double Extra Large	4	34.2	850
Quadruple Extra Large	8	68.4	1,690
Cluster Quad Extra Large	8	23	1,690
Cluster Eight Extra Large	16	60.5	3,370

งานวิจัยนี้นำเสนอการทำไลฟ์ไมเกรชั่นแบบใหม่ที่สามารถเคลื่อนย้ายเวอร์ชวลแมชชีนแบบ SMP ขนาดใหญ่ได้ในขณะที่ทำงานและมีการขัดจังหวะการทำงานของเวอร์ชวลแมชชีนน้อยกว่าการไมเกรชั่นแบบเดิมหลายเท่า โดยนำทีเอลซี (TLC) [7] ซึ่งใช้เวอร์ชวลไลเซชั่นซอฟต์แวร์ที่เป็นโอเพ่นซอร์สชื่อเควีเอ็ม มาพัฒนาต่อ ทีเอลซีนั้นใช้วิธีสร้างเทรดขึ้นมาเพิ่มเพื่อทำหน้าที่เช็คพอยน์ โดยเวอร์ชวลแมชชีนและเทรดดังกล่าว จะทำงานไปพร้อม ๆ กัน

สำหรับวิธีการใหม่หรือทีเอลเอ็ม ได้รับการออกแบบให้มีสามเทรด เทรดแรกทำหน้าที่สแกนหน่วยความจำและส่งไปยังโฮสปลายทาง เทรดที่สองเลือกส่งเฉพาะข้อมูลที่เกิดขึ้นใหม่ เทรดที่สามคอยส่งข้อมูลที่เหลือ โดยเทรดแรกและเทรดที่สองจะส่งข้อมูลในลักษณะขนานกันไป

จากการทดลองกับเวอร์ชวลแมชชีนที่ประมวลผลโปรแกรม mg Class D และ sp Class D ของ NAS Parallel Benchmarks พบว่าทีเอลเอ็มสามารถทำไลฟ์ไมเกรชั่นได้ภายในเวลาจำกัดและสามารถวัดความคืบหน้าได้ ในขณะที่วิธีการเดิมใช้เวลานานมากจนกว่า Benchmark จะทำงานจบจึงไมเกรท นอกจากนั้นทีเอลเอ็มมีระยะเวลาการขัดจังหวะน้อยกว่าการไมเกรชั่นแบบหยุดแล้วจึงไมเกรทเดิมของเควีเอ็มถึง 7 เท่าและ 5.2 เท่าสำหรับโปรแกรม mg Class D และ sp Class D

จากผลการทดลองเบื้องต้นพบว่า ทีเอลเอ็ม ซึ่งมีการส่งข้อมูลแบบขนานในลักษณะพื้นฐาน ให้ผลที่ดีกว่างานอื่น ๆ ก่อนหน้านี้ ผู้วิจัยมีความคิดว่า การเพิ่มการเชื่อมต่อในการส่งข้อมูลให้มากขึ้น น่าจะทำให้การทำงานของไลฟ์ไมเกรชั่นเร็วขึ้นอีก

ผู้วิจัยจึงทำการพัฒนาไลบรารีสำหรับการเชื่อมต่อเครือข่ายแบบขนานขึ้นมา เพื่อทดสอบสมมติฐานข้างต้น และได้ออกแบบการทดลองต่างหาก ด้วยการส่งข้อมูลโดยใช้การเชื่อมต่อแบบต่าง ๆ เพื่อทดสอบความเป็นไปได้ ศึกษาแนวโน้ม ก่อนจะนำไปประยุกต์ใช้กับทีเอลเอ็ม ให้มีการทำงานแบบขนานมากขึ้น ซึ่งเป็นงานที่จะทำต่อไปในอนาคต

2. ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

การทำไลฟ์ไมเกรชั่นของเวอร์ชวลแมชชีนนั้นได้รับการพัฒนาขึ้นบนไฮเปอร์ไวเซอร์ได้แก่ Xen [2] และ Vmotion ของ Vmware [3] และ KVM [4] ซึ่งมีหลักการที่คล้ายกันคือการไมเกรชั่นแบบ Pre-Copy ซึ่งจะทยอยส่งข้อมูลเป็นระยะจากเวอร์ชวลแมชชีนต้นทางไปยังปลายทางโดยที่ในการส่งเพจหน่วยความจำแต่ละครั้งจะส่งไม่มากเพื่อไม่ให้ขัดจังหวะการทำงานของเวอร์ชวลแมชชีนมากเกินไปและทำเช่นนี้ต่อไปเรื่อย ๆ จนกระทั่งจำนวน Dirty เพจที่จำเป็นต้องส่งเหลืออยู่น้อยแล้วจึงหยุดเวอร์ชวลแมชชีนเพื่อส่งสถานะที่เหลือไปทั้งหมดแล้วโอนการทำงานไปยังเวอร์ชวลแมชชีนปลายทาง ปัญหาของวิธีการนี้คือถ้าเวอร์ชวลแมชชีนเป็นแบบ SMP และมีการเขียนหน่วยความจำเป็นปริมาณมากวิธีนี้จะต้องใช้เวลานานมากจนกว่าจะเหลือ Dirty เพจน้อยแล้วจบได้

งานวิจัยทางด้านไมเกรชันมีจุดมุ่งหมายเดียวกันคือการพยายามพัฒนาวิธีการที่ทำให้การไมเกรชันมีประสิทธิภาพและมีการขัดจังหวะการทำงานของเวอชวลแมชชีนที่น้อยที่สุดเท่าที่จะทำได้ งานวิจัยของ [5] ใช้เทคนิค Delta Compression เพื่อลดขนาดของข้อมูลและเพิ่มประสิทธิภาพการทำไลฟ์ไมเกรชันสำหรับแอพพลิเคชันสำหรับองค์กรซึ่งยังมีการใช้งานหน่วยความจำและปริมาณการประมวลผลน้อยกว่า NPB Benchmark ที่ใช้ในการทดลองนี้ แต่อย่างไรก็ตามการทำ Delta Compression อาจนำมาใช้เพื่อเสริมประสิทธิภาพของทีเอลเอ็มได้ในอนาคต งานวิจัย [6] เสนอวิธีการไมเกรชันแบบ Post-Copy โดยย้ายการทำงานของเวอร์ชวลแมชชีนไปยังคอมพิวเตอร์ปลายทางก่อนแล้วค่อยทยอยเคลื่อนย้ายเพจหน่วยความจำที่จำเป็นต้องใช้ในการประมวลผล On-Demand ในกรณีที่มีความจำเป็นต้องใช้หน่วยความจำมากอย่างต่อเนื่องบนเวอร์ชวลแมชชีนแบบ SMP วิธีนี้อาจทำให้เวอร์ชวลแมชชีนต้องหยุดการทำงานเป็นเวลานานได้

3. การดำเนินการวิจัย

ประกอบด้วย การออกแบบไลฟ์ไมเกรชั่นแบบเทรด การพัฒนาไลบรารีการเชื่อมต่อแบบขนาน และงานที่จะทำต่อไปในอนาคตคือการนำสองอย่างข้างต้นมาประยุกต์ใช้งานร่วมกัน

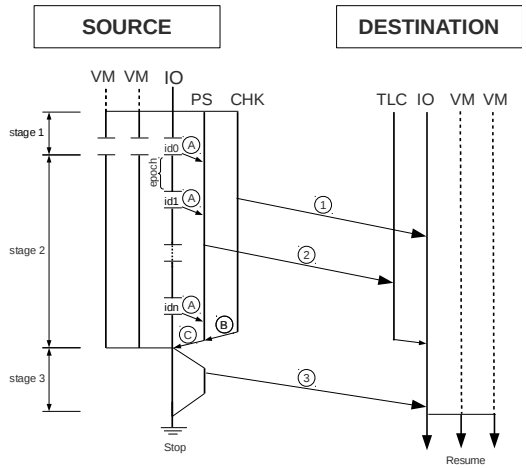
3.1 ไลฟ์ไมเกรชั่นแบบเทรด

สถาปัตยกรรมไลฟ์ไมเกรชั่นแบบเทรด (TLM - Thread-based Live Migration) ได้รับการออกแบบโดยอาศัยหลัก Pre-Copy ประกอบไปด้วยเครื่องโฮสต้นทางและปลายทาง โดยที่เครื่องโฮสต้นทางจะเป็นเครื่องที่รันเวอร์ชวลแมชชีน ที่จะทำไลฟ์ไมเกรชั่นไปยังเครื่องโฮสปลายทาง ทางฝั่งต้นทางประกอบด้วย เทรดการทำงานของเวอร์ชวลแมชชีน (VM) เทรดไอโอ (IO) เทรดเพจเซฟเวอร์ (PS) และ เทรดเช็คพอยน์ (CHK) ฝั่งปลายทางประกอบด้วยเทรดการทำงานของเวอร์ชวลแมชชีน (VM) เทรดที่รอรับข้อมูล (IO) และเทรดทีแอลซี (TLC)

จากรูปที่ 1 เมื่อเครื่องโฮสต้นทางได้รับคำสั่งให้ทำไมเกรชั่น (stage 1) จะมีหยุดการทำงานของเวอร์ชวลแมชชีน ทำการกำหนดค่าเริ่มต้นต่าง ๆ สร้างเทรดเพจเซฟเวอร์และเทรดเช็คพอยน์ เทรดไอโอจะทำการตรวจสอบ dirty ที่เกิดขึ้น และแจ้งผลให้กับเทรดเพจเซฟเวอร์ (ลูกศร A)

ซึ่งใน stage 1 นี้จะยังไม่มี dirty

เมื่อเข้าสู่ stage 2 เทรดเวอร์ชวลแมชชีนและเทรดไอโอจะเริ่มทำงานต่อ เทรดเช็คพอยน์จะทำการอ่านหน่วยความจำของเวอร์ชวลแมชชีนและทยอยส่งไปยังโฮสปลายทาง (ลูกศร 1) สำหรับเทรดไอโอจะมีการหยุดทำงานเป็นช่วง ๆ ตามระยะเวลา epoch เพื่อตรวจสอบ dirty ที่เกิดขึ้นและแจ้งผลให้กับเทรดเพจเซฟเวอร์ (ลูกศร A) เพื่อให้เทรดเพจเซฟเวอร์ส่งข้อมูลเฉพาะ dirty ไปยังโฮสปลายทาง (ลูกศร 2) ทางฝั่งโฮสปลายทางมีสองเทรดที่คอยรับข้อมูล และเขียนข้อมูลลงหน่วยความจำโดยให้ลำดับความสำคัญกับข้อมูลที่ได้รับจากเทรดเพจเซฟเวอร์มากกว่าเทรดเช็คพอยน์สำหรับกรณีที่มีการเขียนข้อมูลลงตำแหน่งเดียวกัน



รูปที่ 1 สถาปัตยกรรมไลฟ์ไมเกรชั่นแบบเทรด

เมื่อเทรดเช็คพอยน์ทำงานเสร็จจะส่งสัญญาณให้กับเทรดเพจเซฟเวอร์เพื่อให้หยุดทำงาน จากนั้นเทรดเพจเซฟเวอร์ก็จะส่งสัญญาณให้กับเทรดไอโอ เพื่อให้หยุดการทำงานของเวอร์ชวลแมชชีน เมื่อเข้าสู่ stage 3 โฮสต้นทางจะทำการส่งข้อมูลที่เหลือไปยังโฮสปลายทาง (ลูกศร 3) เมื่อส่งครบแล้วจึงหยุดการทำงาน ทางฝั่งโฮสปลายทางเมื่อได้รับข้อมูลครบแล้วจะทำการสั่งให้เวอร์ชวลแมชชีนทำงานต่อ

3.2 ไลบรารีสำหรับการเชื่อมต่อแบบขนาน

ผู้ทำการวิจัย ได้พัฒนาไลบรารีสำหรับอำนวยความสะดวกในการเชื่อมต่อเครือข่ายแบบขนานโดยใช้โปรโตคอลทีซีพี ตามโมเดลไคลเอนต์-เซิร์ฟเวอร์ (Client-Server Model) มีเป้าหมายเพื่อนำไปใช้กับการทำไลฟ์ไมเกรชันแบบเทรด จำแนกเป็น 2 ส่วนดังนี้

```
#define ADDRLEN 16
#define PORTLEN 16

typedef struct {
    char addr[ADDRLEN];
    char port[PORTLEN];
} mr_addr_port;
```

รูปที่ 2 โครงสร้างข้อมูลสำหรับการเชื่อมต่อแบบขนาน

3.2.1 โครงสร้างข้อมูล ชื่อ mr_addr_port ใช้เก็บข้อมูลชุดของหมายเลขไอพีและหมายเลขพอร์ต ของเครื่องเซิร์ฟเวอร์ โดยการใช้งานจะประกาศตัวแปรเป็นแบบอะเรย์

```
mr_addr_port *mr_get_opt(const char *filename, int *n);
int *mr_get_fd_list(int n);
void mr_server_listen(int *fd, mr_addr_port *ap, int n);
int *mr_server_accept(int *fd, int n);
void mr_client_connect(int *fd, mr_addr_port *ap, int n);
```

รูปที่ 3 ฟังก์ชันสำหรับการเชื่อมต่อแบบขนาน

3.2.2 ฟังก์ชันการทำงาน ประกอบด้วยฟังก์ชันหลัก 5 ฟังก์ชัน ดังนี้

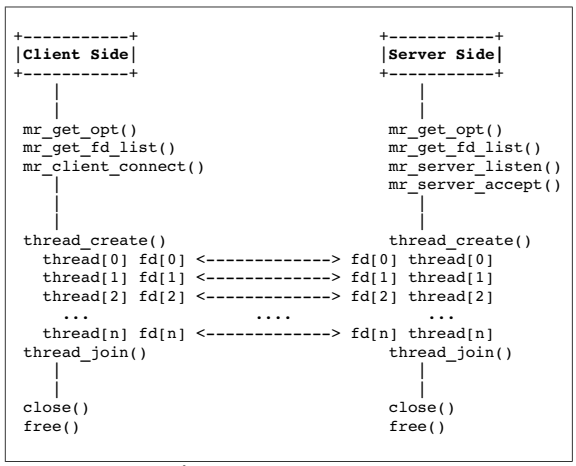
mr_get_opt(filename, n) ทำหน้าที่อ่านค่าจากไฟล์คอนฟิก ที่เก็บข้อมูลชุดของหมายเลขไอพี และหมายเลขพอร์ตของเครื่องเซิร์ฟเวอร์ เมื่อจบการทำงานจะคืนค่าเป็นอะเรย์ของ mr_addr_port และ n จะถูกกำหนดให้มีค่าเท่ากับจำนวนหมายเลขไอพีและหมายเลขพอร์ต

mr_get_fd_list(n) ทำหน้าที่สร้างซ็อกเก็ตสำหรับการเชื่อมต่อ โดยคืนค่าเป็นอะเรย์ของไฟล์เดสคริปเตอร์ (File Descriptor) อะเรย์ที่สร้างมีขนาดเท่ากับ n

mr_server_listen(fd, ap, n) ทำการไบด์ (Bind) หมายเลขไอพีและหมายเลขพอร์ต เข้ากับไฟล์เดสคริปเตอร์ที่ได้จาก mr_get_fd_list() และปรับสถานะของซ็อกเก็ตให้เป็นการรอรับการเชื่อมต่อ ฟังก์ชันนี้ใช้เฉพาะทางฝั่งเซิร์ฟเวอร์เท่านั้น

mr_server_accept(fd, n) รอรับการเชื่อมต่อจากไคลเอนต์ จนครบ จำนวน n การเชื่อมต่อ เมื่อจบการทำงานแล้วจะคืนค่าเป็นอะเรย์ของไฟล์เดสคริปเตอร์ที่ใช้ติดต่อสื่อสารกับไคลเอนต์ ฟังก์ชันนี้ใช้เฉพาะทางฝั่งเซิร์ฟเวอร์เท่านั้น

mr_client_connect(fd, ap, n) ทำการเชื่อมต่อไปยังเซิร์ฟเวอร์จนครบจำนวน n การเชื่อมต่อ หลังจากจบการทำงานแล้ว ไคลเอนต์สามารถติดต่อสื่อสารกับเซิร์ฟเวอร์ได้ ผ่านอะเรย์ของไฟล์เดสคริปเตอร์ ฟังก์ชันนี้ใช้เฉพาะทางฝั่งไคลเอนต์เท่านั้น



รูปที่ 4 รูปแบบการใช้งานไลบรารี

รูปแบบการใช้นำไลบรารีมาใช้งานจะเป็นในลักษณะมัลติเทรด ดังแสดงในรูปที่ 4

3.3 การปรับปรุงไลฟ์ไมเกรชั่นแบบเทรด

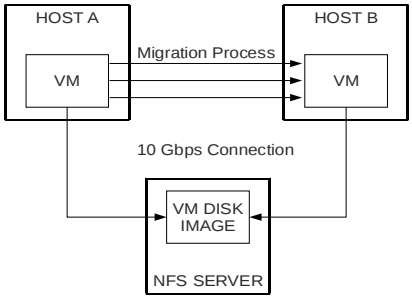
ในอนาคตผู้วิจัย จะดำเนินการปรับปรุงไลฟ์ไมเกรชั่นแบบเทรดให้มีประสิทธิภาพมากขึ้นอีก โดยมีเป้าหมายเพื่อลดระยะเวลาที่เวอร์ชวลแมชชีนหยุดการทำงาน (Downtime = $Time_{stage\ 3}$) และลดระยะเวลาทั้งหมดในกระบวนการไมเกรชั่น (Elapsed Time = $Time_{stage\ 1}$ + $Time_{stage\ 2}$ + $Time_{stage\ 3}$) ด้วยการเพิ่มการทำงานแบบขนานให้มากขึ้น โดยนำไลบรารีที่ได้รับการพัฒนา มาประยุกต์ใช้กับไลฟ์ไมเกรชั่นแบบเทรด

4. การทดลองเบื้องต้น

การทดลองประกอบไปด้วย 2 ส่วน คือ การทดลองไลฟ์ไมเกรชั่นแบบเทรด (TLM) และ การทดลองเพื่อศึกษาความเป็นไปได้ในการใช้ไลบรารีที่พัฒนาขึ้นมา

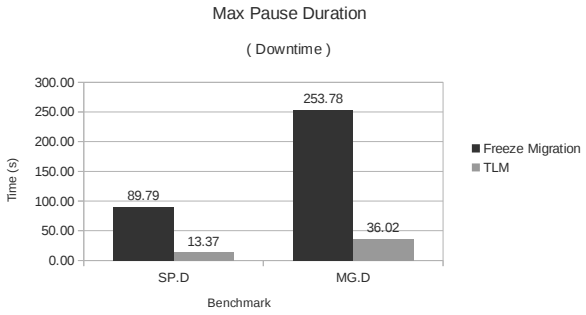
4.1 การทดลองไลฟ์ไมเกรชั่นแบบเทรด

ในการทดลองนี้ ใช้เครื่องคอมพิวเตอร์จำนวน 3 เครื่อง แต่ละเครื่องใช้หน่วยประมวลผลกลางแบบ AMD Opteron Processor 6172 ความเร็ว 2.1 GHz มี 12 แกนประมวลผล (Core) ขนาดหน่วยความจำหลัก 48 GB ขนาดความจุของฮาร์ดดิสก์ 900 GB ระบบปฏิบัติการเป็น Ubuntu 11.10 เคอร์เนล 3.0.0-12-server ติดตั้งโมดูล kvm-kmod-3.0b ทั้ง 3 เครื่องเชื่อมต่อกันด้วยเครือข่ายที่มีความเร็ว 10 Gbps ใช้เก็บข้อมูลฮาร์ดดิสก์ในรูปแบบไฟล์อิมเมจของเวอร์ชวลแมชชีน ด้วยการแชร์ไฟล์ให้กับเครื่องโฮสผ่านโปรโตคอลเอ็นเอฟเอส (NFS) จำนวน 1 เครื่อง และอีก 2 เครื่องใช้งานเป็นโฮสของเวอร์ชวลแมชชีน สร้างเวอร์ชวลแมชชีนที่มีหน่วยประมวลผลกลาง 8 แกนประมวลผล ขนาดหน่วยความจำหลัก 16 GB (สำหรับเบนช์มาร์ก SP.D) และ 36 GB (สำหรับเบนช์มาร์ก MG.D) ขนาดความจุของฮาร์ดดิสก์ 10 GB ระบบปฏิบัติการเกสต์โอเอสคือ Ubuntu 10.04.3 LTS เคอร์เนล 2.6.32-33-generic



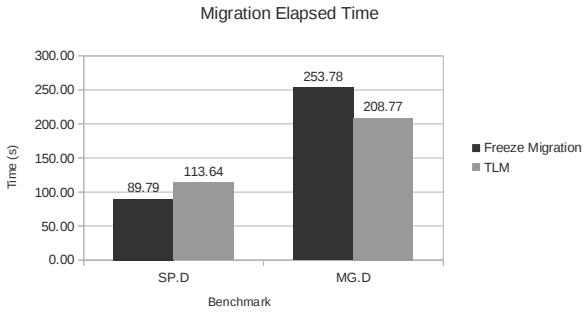
รูปที่ 5 โครงสร้างของระบบที่ใช้ในการทดลองไลฟ์ไมเกรชั่น

การประเมินประสิทธิภาพทำโดยการเปรียบเทียบเวลาที่ใช้ในการทำงานแบบโอเอ็มพี (OMP) ของโปรแกรม NAS Parellel Benchmark (NPB) เวอร์ชัน 3.3 เลือกเบนช์มาร์ก SP.D และ MG.D ซึ่งมีการใช้หน่วยความจำขนาด 12.1 GB และ 27.3 GB ตามลำดับ ทำการเปรียบเทียบระหว่างการใช้ไลฟ์ไมเกรชั่นแบบเทรด (TLM) ไลฟ์ไมเกรชั่นแบบดั้งเดิม และแบบหยุดแล้วไมเกรชั่น



รูปที่ 6 ระยะเวลาที่เวอร์ชวลแมชชีนหยุดการทำงาน

จากผลการทดลองเบื้องต้น ในรูปที่ 6 แสดงระยะเวลาที่เวอร์ชวลแมชชีนหยุดการทำงาน (Downtime) เพื่อทำการคัดลอกข้อมูลไปยังเครื่องโฮสปลายทาง จะเห็นได้ว่า ไลฟ์ไมเกรชั่นแบบเทรดใช้เวลาน้อยกว่าแบบหยุดแล้วไมเกรชั่น ทั้งสองเบนช์มาร์ก โดยหยุดเวอร์ชวลแมชชีนเป็นเวลา 13.37 วินาที สำหรับเบนช์มาร์ก SP.D และหยุด 36.02 วินาที สำหรับเบนช์มาร์ก MG.D ในขณะที่แบบหยุดแล้วไมเกรชั่นใช้เวลา 89.79 วินาที และ 253.78 ตามลำดับ ที่เป็นเช่นนี้เพราะทีเอลเอ็มใช้สองเทรดในการส่งข้อมูลไปล่วงหน้า โดยทำงานขนานกัน ทำให้การหยุดการทำงานเพื่อส่งข้อมูลในช่วงสุดท้ายใช้เวลาน้อย

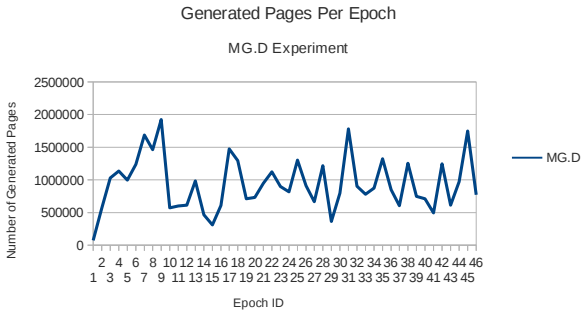


รูปที่ 7 ระยะเวลาทั้งหมดในกระบวนการไมเกรชั่น

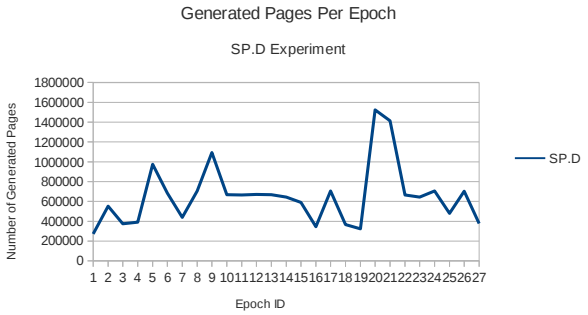
ผลการทดลองในรูปที่ 7 แสดงระยะเวลาตั้งแต่เริ่มต้นไมเกรชั่นจนจบ (Elapsed Time) จะเห็นได้ว่าในการทดลองที่ใช้เบนช์มาร์ก SP.D ไมเกรชันแบบเดิมใช้เวลา 89.79 วินาที ซึ่งน้อยกว่าไลฟ์ไมเกรชั่นแบบเทรดที่ใช้เวลา 113.64 วินาที แต่ในการทดลองที่ใช้เบนช์มาร์ก MG.D ไมเกรชันแบบเดิมใช้เวลา 253.78 วินาที ซึ่งมากกว่าไลฟ์ไมเกรชันแบบเทรดที่ใช้เวลา 208.77 วินาที

ในการทดลองนี้ไลฟ์ไมเกรชั่นแบบเดิมใช้เวลานานมากเพราะเบนช์มาร์กมีการเขียนหน่วยความจำเป็นจำนวนมาก (เป็นแสนเป็นล้านเพจ) อยู่ตลอดเวลา จะทำการไมเกรชั่นสำเร็จก็ต่อเมื่อ เบนช์มาร์กทำงานจบแล้ว

จากรูปที่ 8 และ 9 เป็นกราฟแสดงปริมาณของเพจ Dirty ที่ถูกสร้างขึ้น จะเห็นได้ว่ากราฟทั้งสองมีความแตกต่างกัน ทั้งนี้ขึ้นอยู่กับแอพพลิเคชันที่นำมาใช้ โดย MG.D มีการสร้างเพจ Dirty เยอะกว่า SP.D มีช่วงของข้อมูลกว้างกว่า และมีจำนวน Epoch มากกว่าด้วย

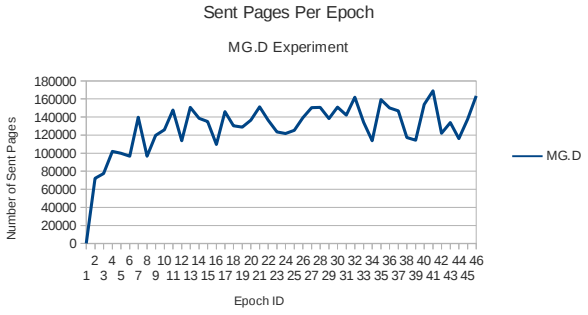


รูปที่ 8 จำนวนเพจที่สร้างขึ้นในแต่ละ Epoch สำหรับ MG.D

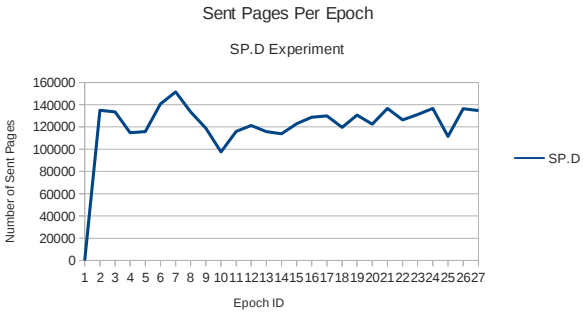


รูปที่ 9 จำนวนเพจที่สร้างขึ้นในแต่ละ Epoch สำหรับ SP.D

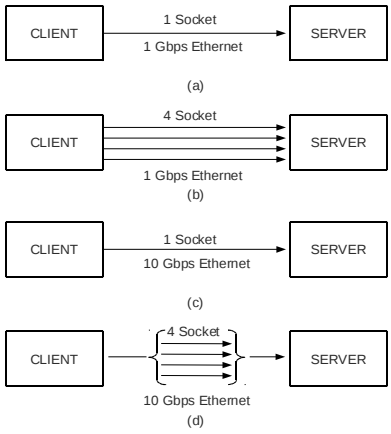
รูปที่ 10 และ 11 แสดงปริมาณของเพจ Dirty ที่เทรดเพจเซฟเวอร์ส่งไปยังเครื่องโฮสปลายทาง โดย MG.D มีการส่งข้อมูลเพจมากกว่า SP.D และมีช่วงของข้อมูลกว้างกว่าด้วย



รูปที่ 10 จำนวนเพจที่เทรดเพจเซฟเวอร์ส่งออกไป สำหรับ MG.D



รูปที่ 11 จำนวนเพจที่เทรดเพจเซฟเวอร์ส่งออกไป สำหรับ MG.D



รูปที่ 12 การทดลองเพื่อศึกษาความเป็นไปได้ในการใช้งานไลบรารี

(a) แบบ 1 ทีซีพีช็อกเก็ต บน 1 ฟิสิคัล 1 Gpbs อีเทอร์เน็ต

(b) แบบ 4 ทีซีพีช็อกเก็ต บน 4 ฟิสิคัล 1 Gbps อีเทอร์เน็ต

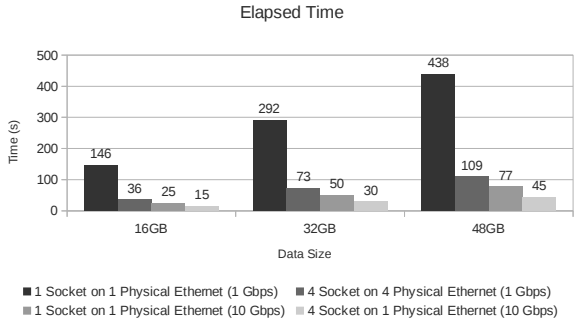
(c) แบบ 1 ทีซีพีช็อกเก็ต บน 1 ฟิสิคัล 10 Gbps อีเทอร์เน็ต

(d) แบบ 4 ทีซีพีช็อกเก็ต บน 1 ฟิสิคัล 10 Gbps อีเทอร์เน็ต

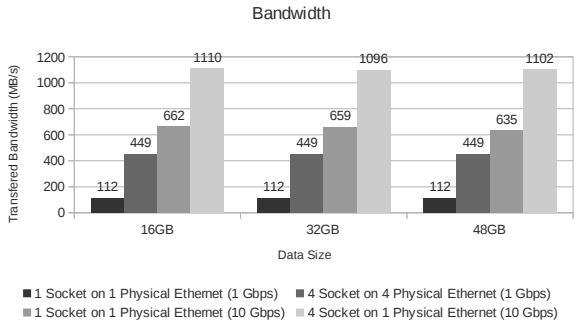
4.2 การทดลองเพื่อศึกษาความเป็นไปได้ในการใช้งานไลบรารี

การทดลองนี้ใช้เครื่องคอมพิวเตอร์จำนวน 2 เครื่อง แต่ละเครื่องใช้หน่วยประมวลผลกลางแบบ AMD Opteron Processor 6172 ความเร็ว 2.1 GHz มี 12 แกนประมวลผล (Core) ขนาดหน่วยความจำหลัก 48 GB ขนาดความจุของฮาร์ดดิสก์ 900 GB ระบบปฏิบัติการเป็น Ubuntu 11.10 เคอร์เนล 3.0.0-12-server เชื่อมต่อกันด้วยเครือข่ายที่มีความเร็ว 1 Gbps และ 10 Gbps เครื่องที่ 1 เป็นไคลเอนต์ เครื่องที่ 2 เป็นเซิร์ฟเวอร์ ในการทดลองนี้ไคลเอนต์มีหน้าที่ส่งข้อมูลขนาดต่าง ๆ ให้กับเซิร์ฟเวอร์ โดยใช้การเชื่อมต่อหลาย ๆ แบบ ขนาดของข้อมูลที่ใช้ในการทดลองได้แก่ 16 GB 32GB และ 48 GB ตามลำดับ การเชื่อมต่อที่ใช้ในการทดลองได้แก่ 1) แบบ 1 ทีซีพีช็อกเก็ต บน 1 ฟิสิคัล 1 Gpbs อีเทอร์เน็ต 2) แบบ 4 ทีซีพีช็อกเก็ต บน 4 ฟิสิคัล 1 Gbps อีเทอร์เน็ต 3) แบบ 1 ทีซีพีช็อกเก็ต บน 1 ฟิสิคัล 10 Gbps อีเทอร์เน็ต และ 4) แบบ 4 ทีซีพีช็อกเก็ต บน 1 ฟิสิคัล 10 Gbps อีเทอร์เน็ต การประเมินประสิทธิภาพทำโดยการเปรียบเทียบเวลาที่ไคลเอนต์ใช้ส่งข้อมูลให้กับเซิร์ฟเวอร์ ในแต่ละ

ขนาด และในแต่ละแบบของการเชื่อมต่อ จากนั้นทำการคำนวณและเปรียบเทียบค่าแบนด์วิดท์ จากปริมาณข้อมูลที่ใช้ส่งต่อเวลาที่ใช้ส่ง



รูปที่ 13 เวลาที่ไคลเอนต์ส่งข้อมูลให้กับเซิร์ฟเวอร์



รูปที่ 14 แบนด์วิดท์ระหว่างไคลเอนต์และเซิร์ฟเวอร์

จากผลการทดลองพบว่า การส่งข้อมูลผ่าน 10 Gbps Ethernet ที่ใช้การเชื่อมต่อแบบขนานจะมีประสิทธิภาพสูงสุด นั่นคือใช้เวลาในการส่งข้อมูลน้อยที่สุดและมีแบนด์วิดท์สูงสุด โดยใช้เวลา 15 วินาทีในการส่งข้อมูลขนาด 16 GB ใช้เวลา 30 วินาทีในการส่งข้อมูลขนาด 32 GB และใช้เวลา 45 วินาทีในการส่งข้อมูลขนาด 48 GB มีแบนด์วิดท์ประมาณ 1.1 GB/s (8.8 Gbps)

ในกรณีที่ระบบไม่มี 10 Gbps Ethernet สามารถใช้การเชื่อมต่อแบบขนานกับ 1 Gbps Ethernet แทนได้ จากผลการทดลองพบว่า การส่งข้อมูลผ่านการเชื่อมต่อแบบขนาน จำนวน 4 การเชื่อมต่อ ใช้เวลาน้อยกว่าและมีแบนด์วิดท์สูงกว่าการส่งข้อมูลผ่านการเชื่อมต่อแบบ 1 การเชื่อมต่อ 4 เท่า

ผลการทดลองข้างต้น เป็นการช่วยเพิ่มความมั่นใจ ในการที่จะนำไลบรารีการเชื่อมต่อเครือข่ายแบบขนานมาใช้กับการทำไลฟ์ไมเกรชั่น โดยมีแนวโน้มว่า การทำไลฟ์ไมเกรชั่นจะใช้เวลาน้อยลงเมื่อใช้วิธีส่งข้อมูลแบบขนาน

6. สรุปผลการทดลองและงานที่จะทำต่อไป

งานวิจัยนี้ทำการปรับปรุงประสิทธิภาพของการทำไมเกรชั่นให้รองรับเวอร์ชวลแมชชีนที่กำลังรันแอพพลิเคชั่นที่เน้นการใช้งานหน่วยความจำ โดยวิธีไลฟ์ไมเกรชั่นแบบเดิมนั้น ไม่สามารถทำสำเร็จได้ในขณะที่กำลังรันแอพพลิเคชั่นดังกล่าว อีกวิธีคือหยุดแล้วไมเกรชั่น วิธีนี้จะต้องหยุดการทำงานของเวอร์ชวลแมชชีนเป็นเวลานาน ผู้วิจัยขอนำเสนอวิธีไลฟ์ไมเกรชั่นแบบที่ใช้เทรดมาช่วยในการย้ายข้อมูลบางส่วนไปยังเครื่องปลายทางล่วงหน้า ในขณะเดียวกันกับที่เครื่องเวอร์ชวลแมชชีนทำงาน เพื่อให้การหยุดการทำงานของเวอร์ชวลแมชชีน ในขั้นตอนสุดท้ายของการไมเกรชั่น ใช้เวลาน้อยกว่าวิธีอื่น การประเมินประสิทธิภาพทำโดยใช้โปรแกรม NAS Parallel Benchmarks

ผลการทดลองพบว่าวิธีการทำไลฟ์ไมเกรชั่นแบบเทรด ใช้เวลาหยุดการทำงานของเวอร์ชวลแมชชีนน้อยกว่าวิธีหยุดแล้วไมเกรชั่น 5 ถึง 7 เท่า

จากผลการทดลองเบื้องต้นเพื่อศึกษาความเป็นไปได้ในการใช้งานไลบรารีพบว่า วิธีการส่งข้อมูลโดยใช้การเชื่อมต่อแบบขนาน จะช่วยให้ส่งข้อมูลได้เร็วขึ้น ใช้เวลาน้อยลง ดังนั้น งานในขั้นต่อไปคือ การนำไลบรารีที่ได้ทำการพัฒนาเพื่อใช้ในการเชื่อมต่อแบบขนาน มาประยุกต์ใช้กับไลฟ์ไมเกรชั่นแบบเทรด โดยมีความคาดหวังว่า การย้ายข้อมูลไปยังเครื่องปลายทางแบบขนานนั้น จะช่วยให้การหยุดการทำงานของเวอร์ชวลแมชชีนใช้เวลาน้อยลง และจะช่วยให้ระยะเวลาทั้งหมดในกระบวนการไมเกรชั่นลดลงด้วย

7. เอกสารอ้างอิง

[1] http://aws.amazon.com/ec2/instance-types/

[2] Christopher Clarket al. Live migration of virtual machines. In Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation - Volume 2 (NSDI'05), Vol. 2. USENIX Association, Berkeley, CA, USA, 273-286.

[3] vmware, http://www.vmware.com/files/pdf/VMware-VMotion-DS-EN.pdf

[4] http://www.linux-kvm.org/page/Migration

[5] Petter Svärd, Benoit Hudzia, Johan Tordsson, Erik Elmroth, Evaluation of delta compression techniques for efficient live migration of large virtual machines, VEE 2011

[6] Michael R. Hines, Umesh Deshpande, and Kartik Gopalan. 2009. Post-copy live migration of virtual machines. SIGOPS Oper. Syst. Rev. 43, 3 (July 2009), 14-26.

[7] Vasinee Siripoonya and Kasidit Chanchio. Thread-Based Live Checkpointing of Virtual Machines. IEEE NCA 2011, pp.155-162