การจัดแบ่งแบนด์วิดท์ของไอโอแบบปรับเปลี่ยนสำหรับสภาวะแบบเสมือน

นายภาสกร สุขีพจน์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2559

ADAPTIVE I/O BANDWIDTH ALLOCATION FOR VIRTUALIZATION ENVIRONMENT

Mr. Phasakorn Sukheepoj

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Science Program in Computer Science

Department of Computer Engineering

Faculty of Engineering

Chulalongkorn University

Academic Year 2016

หัวข้อวิทยานิพนธ์	การจัดแบ่งแบนด์วิดท์ของไอโอแบบปรับเปลี่ยนสำหรับสภาวะแบบเสมือน
โดย	นายภาสกร สุขีพจน์
สาขาวิชา	วิทยาศาสตร์คอมพิวเตอร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	ผู้ช่วยศาสตราจารย์ ดร.ณัฐวุฒิ หนูไพโรจน์

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

.....คณบดีคณะวิศวกรรมศาสตร์

(รองศาสตราจารย์ ดร.สุพจน์ เตชวรสินสกุล)

คณะกรรมการสอบวิทยานิพนธ์

.....ประธานกรรมการ

(ผู้ช่วยศาสตราจารย์ ดร.วีระ เหมืองสิน)

.....อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก

(ผู้ช่วยศาสตราจารย์ ดร.ณัฐวุฒิ หนูไพโรจน์)

.....กรรมการ

(ผู้ช่วยศาสตราจารย์ ดร.เกริก ภิรมย์โสภา)

.....กรรมการภายนอกมหาวิทยาลัย

(ดร.กาญจนา ศิลาวราเวทย์)

ภาสกร สุขีพจน์ : การจัดแบ่งแบนด์วิดท์ของไอโอแบบปรับเปลี่ยนสำหรับสภาวะแบบเสมือน (ADAPTIVE I/O BANDWIDTH ALLOCATION FOR VIRTUALIZATION ENVIRONMENT) อ.ที่ปรึกษาวิทยานิพนธ์หลัก: ผศ. ดร.ณัฐวุฒิ หนูไพโรจน์, 44 หน้า.

ในอุตสาหกรรมคอมพิวเตอร์ปัจจุบัน เทคโนโลยีเวอชวลไลเซชั่นยังคงเป็นสิ่งที่ดึงดูดความสนใจเป็นอย่างมาก ด้วยเทคโนโลยีนี้ทำให้เครื่องแม่ข่ายเดี่ยวได้รับความสามารถในการดำเนินระบบปฏิบัติการหลายระบบอย่างต่อเนื่อง และเป็นอิสระ ซึ่งเป็นการใช้งานทรัพยากรอย่างเต็มที่ และยังเป็นการลดค่าใช้จ่ายในการซ่อมบำรุง เนื่องจากมีการใช้ฮาร์ดแวร์ที่น้อยกว่าเมื่อเทียบกับระบบที่มีเครื่องบริการจำนวนมาก อย่างไรก็ตามการใช้งานทรัพยากรหน่วยเก็บข้อมูลร่วมกันในระบบแบบเสมือนยังคงเป็นปัญหาที่ท้าทาย ซึ่งดูเหมือนว่าการต่อสู้แย่งชิงทรัพยากรของหน่วยเก็บข้อมูลจะมีค่าที่สูงมากเมื่อนำไปเปรียบเทียบกับการแย่งชิงทรัพยากรในหน่วยประมวลผลกลาง และหน่วยความจำ ซึ่งนำไปสู่เวลาในการดำเนินการของไอโอที่ไม่แน่นอน ยิ่งไปกว่านั้นไฮเปอร์ไวเซอร์ไม่สามารถจัดสรรทรัพยากรหน่วยเก็บข้อมูลที่เพียงพอให้กับโดเมนแต่ละโดเมน ดังนั้นระบบจึงไม่สามารถประคับประคองคุณภาพในการให้บริการเอาไว้ได้ การบริหารจัดการทรัพยากรหน่วยเก็บข้อมูลที่มีประสิทธิภาพ และมีความยืดหยุ่นจึงเป็นแนวทางที่สำคัญ ในการพัฒนาประสิทธิภาพของหน่วยเก็บข้อมูลในระบบแบบเสมือน ในงานวิจัยนี้ได้นำเสนอการจัดแบ่งแบนด์วิดท์ของไอโอแบบปรับเปลี่ยนสำหรับสภาวะเสมือนบนแพลตฟอร์มเซ็น โดยการนำแนวคิดของการจัดการ และควบคุมปริมาณการสื่อสารข้อมูลมาประยุกต์ใช้ในระบบควบคุมแบนด์วิดท์สำหรับใช้ในการแจกจ่ายทรัพยากรดิสก์ โดยการประกันแบนด์วิดท์ต่ำสุด และสูงสุดที่แต่ละโดเมนจะได้รับ กรอบของงานนี้ดำเนินการอยู่บนพื้นที่ของผู้ใช้งาน และทำการสื่อสารกับส่วนต่อประสานของเคอร์เนลของระบบปฏิบัติการเพื่อประสิทธิภาพในการประมวลผล โดยกลุ่มควบคุมของลินุกซ์ได้ถูกนำมาใช้เพื่อสร้างคลาสของทรัพยากรสำหรับแต่ละโดเมนเกสต์ และนำระบบการจัดตารางคิวอย่างยุติธรรมของลินุกซ์มาใช้เพื่อควบคุมการจัดแบ่งที่เท่าเทียมในกลุ่มของโดเมน เนื่องด้วยคลาสของทรัพยากรจะถูกจัดลำดับความสำคัญออกเป็น 10 ระดับ แบนด์วิดท์ที่เหลืออยู่ของระบบจะถูกนำไปเพิ่มให้กับคลาสที่มีลำดับความสำคัญสูงที่สุดก่อน จากนั้นจึงแบ่งให้กับคลาสที่มีลำดับความสำคัญรองลงมา งานวิจัยนี้แสดงให้เห็นว่าระบบจัดแบ่งแบนด์วิดท์สามารถประคับประคองแบนด์วิดท์ไอโอของหน่วยเก็บข้อมูล และรับประกันการใช้ทรัพยากรอย่างเต็มประสิทธิภาพของระบบได้อีกด้วย

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต .....

สาขาวิชา วิทยาศาสตร์คอมพิวเตอร์ ลายมือชื่อ อ.ที่ปรึกษาหลัก .....

ปีการศึกษา 2559

5670335721 : MAJOR COMPUTER SCIENCE

KEYWORDS: BANDWIDTH CONTROL / I/O BANDWIDTH / VIRTUALIZATION

PHASAKORN SUKHEEPOJ: ADAPTIVE I/ O BANDWIDTH ALLOCATION FOR VIRTUALIZATION ENVIRONMENT. ADVISOR: ASST. PROF. NATAWUT NUPAIROJ, Ph.D., 44 pp.

Server virtualization continues to draw attention to the computer industry today. With virtualization, a single physical server grants an ability to operate multiple services and operating systems independently and simultaneously. It improves the utilization of system resources and also reduces maintenance cost since less hardware is required as compared to the system with multiple servers. However, sharing disk I/O resources in virtualized systems still be a challenging issue. The contention of disk I/O resource seems higher in comparison with CPU and memory contention which leads to uncertain I/O completion time. Moreover, the hypervisor cannot provide sufficient disk I/O resources to the particular domain, so we cannot maintain the quality of service (QoS) of them. Efficient and flexible disk I/O resource management is an important approach to improve disk I/O performance in virtualized systems. In this study, we propose an Adaptive I/O Bandwidth Allocation for Virtualization Environment on Xen platform which selects the Traffic policing and shaping concept to implement the bandwidth control and distribute the disk I/O resource. The framework operates as a user space daemon and communicates with the disk scheduler via kernel interfaces of the operating system by a guarantee of minimum bandwidth and limitation of maximum bandwidth to ensure high-throughput processing. We use the Linux cgroup to create resources class for each guest domain and the Linux CFQ scheduler to build fairness control over the domain group. Since the resource classes are divided into ten priority classes, the remaining bandwidth of the system will be taken by the highest priority class, following by the lower priority classes. The proposed framework is particularly useful in maintaining the disk I/O bandwidth of the specific domain and also guarantees the efficient use of system resources.

Department: Computer Engineering Student's Signature .....

Field of Study: Computer Science Advisor's Signature .....

Academic Year: 2016

กิตติกรรมประกาศ

ในการทำงานวิจัยครั้งนี้ ผู้วิจัยขอกราบขอบพระคุณ ผู้ช่วยศาสตราจารย์ ดร. ณัฐวุฒิ หนูไพโรจน์ อาจารย์ที่ปรึกษาวิทยานิพนธ์ เป็นอย่างสูง ที่กรุณาให้คำปรึกษา แนะนำแนวทาง ลำดับขั้นตอนในการวิจัย ทั้งยังให้ความรู้ที่เป็นประโยชน์ต่อการทำงานวิจัย ตลอดจนแนวทางในการแก้ไขปัญหาที่เกิดขึ้นระหว่างการทำวิจัยด้วยดีมาตลอด

ขอขอบพระคุณ ผู้ช่วยศาสตราจารย์ ดร. วีระ เหมืองสิน ประธานกรรมการการสอบวิทยานิพนธ์ ร่วมด้วย ผู้ช่วยศาสตราจารย์ ดร. เกริก ภิรมย์โสภา และ ดร.กาญจนา ศิลาวราเวทย์ กรรมการสอบวิทยานิพนธ์ที่ได้ให้ความกรุณาสละเวลา เพื่อมอบคำแนะที่เป็นประโยชน์ในการทำวิทยานิพนธ์ครั้งนี้

ขอขอบพระคุณคณาจารย์ภาควิชาวิศวกรรมคอมพิวเตอร์ทุกท่าน ที่ได้กรุณาให้ความรู้ คำชี้แนะ ที่เป็นประโชน์ และขอขอบคุณเพื่อนนิสิต ที่คอยผลักดัน และช่วยเหลือในหลายๆ ด้าน

สุดท้ายนี้ต้องขอขอบพระคุณบิดา มารดา และครอบครัว ผู้สนับสนุนในทุกด้าน คอยส่งเสริม มอบความห่วงใย และกำลังใจที่ดีเสมอมา

สารบัญ

หน้า

หน้า

สารบัญตาราง

สารบัญรูปภาพ

บทที่ 1
บทนำ

1.1 ความเป็นมา และความสำคัญของปัญหา

ปัจจุบันเทคโนโลยีเวอร์ชวลไลเซชั่น (virtualization) มีการเติบโตขึ้นอย่างรวดเร็ว คอมพิวเตอร์เสมือน (virtual machine) ได้ถูกนำมาใช้งานอย่างแพร่หลายทั้งในอุตสาหกรรม และแวดวงการศึกษา เนื่องจากเวอร์ชวลไลเซชั่นสามารถจำลองสภาพแวดล้อมให้เสมือนมีคอมพิวเตอร์หลายเครื่องทำงานอยู่ในเครื่องเดียวกันได้ ด้วยการจำลองอุปกรณ์และซอฟต์แวร์เสมือนขึ้น โดยแต่ละระบบมีทรัพยากรแบบเสมือนที่เป็นอิสระต่อกัน คอมพิวเตอร์เสมือนแต่ละเครื่องจึงสามารถมีระบบปฏิบัติการ และซอฟต์แวร์เป็นของตนเองโดยอิสระ นำมาสู่ความสะดวกในการบริหารจัดการและสามารถใช้ทรัพยากรที่มีอยู่อย่างจำกัดให้คุ้มค่ามากที่สุด [1]

ถึงแม้ว่าจะเทคโนโลยีเวอร์ชวลไลเซชั่นจะมีข้อดีอยู่มาก แต่ก็ยังมีข้อจำกัดอยู่บางประการโดยเฉพาะอย่างยิ่งไอโอของหน่วยเก็บข้อมูล ซึ่งเป็นสาเหตุหนึ่งที่นำมาสู่คอขวดของประสิทธิภาพ จากการที่ทรัพยากรหน่วยบันทึกข้อมูลถูกใช้งานร่วมกันโดยโปรแกรมประยุกต์ที่หลากหลาย รวมทั้งระบบปฏิบัติการที่มีมากกว่าหนึ่ง [2-4] ดังนั้น จึงต้องมีนโยบายเพื่อจำกัดการเข้าถึงอย่างเหมาะสม ด้วยเหตุนี้การจัดการหน่วยเก็บข้อมูลภายใน Virtual Machine Monitor (VMM) [5] จึงเข้ามามีบทบาทสำคัญในการกำหนดความเท่าเทียมกัน และประสิทธิภาพของระบบแบบเวอร์ชวลไลซ์ [6-8] แต่อย่างไรก็ตามการนำ Fair scheduler [9] เช่น NOOP, CFQ, Anticipatory และ Deadline มาใช้ในการจัดสรรอุปกรณ์บล็อก สำหรับโดเมนเสมือน อัลกอริทึมจะเพิกเฉยต่อความประสงค์ของผู้ใช้งาน ยกตัวอย่างในกรณีที่ต้องการให้ Guest domain บางโดเมนมีแบนด์วิดท์สำหรับการอ่านข้อมูลที่มากขึ้น แต่ไม่สามารถที่จะจัดสรรได้โดยตรง เนื่องจากมีโดเมนอื่นๆ ต้องการแย่งชิงทรัพยากรในส่วนนี้เช่นกัน นำไปสู่การจัดสรรทรัพยากรที่ด้อยประสิทธิภาพ และไม่สอดคล้องกับความประสงค์ของผู้ใช้งาน งานวิจัยที่ผ่านมาจึงมุ่งเน้นในด้านการออกแบบ และพัฒนาอัลกอริทึม โดยการนำแนวคิดที่หลากหลายมาประยุกต์ใช้ [10-12] เช่น การใช้ Schedule Token และ Time Slice เพื่อทำการแบ่งสัดส่วนของ disk I/O bandwidth ในคอมพิวเตอร์เสมือนจำนวนมากตามลำดับความสำคัญ หรือตามความต้องการไอโอในเวลาดำเนินการ [13, 14], วิธีการจัดตารางสำหรับไอโอของหน่วยเก็บข้อมูลโดยใช้ Leaky bucket และ Weighted fair queue algorithm เพื่อทำการควบคุมการ allocation ของทรัพยากร disk [15], อัลกอริทึมในการจัดสรรอุปกรณ์บล็อกเพื่อความเที่ยงธรรม และรักษาคุณภาพในการให้บริการ [16, 17] แต่อย่างไรก็ตาม งานวิจัยที่ผ่านมายังคงมีข้อจำกัดอยู่บางประการโดยเฉพาะอย่างยิ่งการที่ Guest Domain ไม่สามารถเข้าใช้งานแบนวิดท์ของหน่วยเก็บข้อมูล ได้อย่าง

เต็มที่เนื่องจากมีการสำรองแบนวิดท์เอาไว้ กล่าวคือเมื่อคงเหลือโดเมนที่ทำงานอยู่เพียงโดเมนเดียว ในขณะที่โดเมนอื่นๆ เสร็จสิ้นงานของตนแล้วเปลี่ยนสถานะเป็นหยุดนิ่ง แต่โดเมนที่ทำงานอยู่เพียงตัวเดียวนั้น สามารถใช้ทรัพยากรหน่วยบันทึกข้อมูล ได้เพียงเท่ากับที่ได้กำหนดสัดส่วนเอาไว้เท่านั้น จึงทำให้ขาดความยืดหยุ่นในการใช้ทรัพยากรอย่างเต็มสมรรถภาพ ซึ่งนับว่าเป็นการลดประสิทธิภาพการทำงานโดยรวมของระบบ และความสามารถในการรองรับปริมาณงานที่มากขึ้นด้วย ดังนั้นการจัดสรรแบนด์วิดท์ไอโอสำหรับหน่วยเก็บข้อมูลอย่างเหมาะสม จึงมีความสำคัญเป็นลำดับต้นๆ

งานวิจัยชิ้นนี้จะนำเสนอการจัดการ แบนด์วิดท์ไอโอสำหรับหน่วยเก็บข้อมูลบนระบบแบบเสมือน โดยมีจุดมุ่งหมายให้คอมพิวเตอร์เสมือนสามารถใช้ทรัพยากรดิสก์ได้อย่างเต็มประสิทธิภาพ พร้อมทั้งรับประกันคุณภาพในการให้บริการ และประสิทธิภาพของแต่ละคอมพิวเตอร์เสมือน โดยทำการพัฒนาระบบควบคุม เพื่อจำกัดการเข้าถึงทรัพยากรหน่วยเก็บข้อมูลสำหรับโดเมนเสมือน ที่มีความยืดหยุ่น และปรับเปลี่ยนได้ โดยนำหลักการของ การควบคุมปริมาณการสื่อสารของแพ็กเกตข้อมูล (Traffic shaping) แบบ Hierarchical Token Bucket ซึ่งมีความยืดหยุ่นกว่างานวิจัยอื่นๆ และระบบคิว (Queuing discipline) มาประยุกต์ใช้ในส่วนของอัลกอริทึมเพื่อควบคุมการจัดสรรทรัพยากรของหน่วยเก็บข้อมูล

1.2 วัตถุประสงค์ของงานวิจัย

1.2.1 เพื่อทำการออกแบบและพัฒนาระบบจัดการแบนด์วิดท์ไอโอ แบบปรับเปลี่ยนสำหรับหน่วยเก็บข้อมูลบนระบบแบบเสมือน

1.2.2 เพื่อพัฒนาระบบที่ใช้สำหรับควบคุมอัตราการ อ่าน-เขียนข้อมูลเมื่อมีการใช้งานคอมพิวเตอร์เสมือนพร้อมๆ กัน โดยยังคงรักษาคุณภาพในการให้บริการเอาไว้

1.2.3 เพื่อพัฒนาระบบควบคุมการใช้ทรัพยากรของหน่วยเก็บข้อมูลได้ โดยไม่ต้องทำการแก้ไข kernel ของระบบ

1.2.4 เพื่อวิเคราะห์ และทำการวัดประสิทธิภาพของระบบจัดการ แบนด์วิดท์ไอโอสำหรับหน่วยเก็บข้อมูลบนระบบแบบเสมือน

1.3 ขั้นตอนการวิจัย

1.3.1 ศึกษางานวิจัยที่เกี่ยวข้อง โดยทำการศึกษาเกี่ยวกับหลักการ และอัลกอริทึมสำหรับการควบคุม I/O แบนด์วิดท์ ต่างๆ รวมไปถึงการ ออกแบบระบบในสภาพแวดล้อมแบบเวอชวลไลเซชั่น

1.3.2 ศึกษาโครงสร้างของ Xen Hypervisor รวมไปถึงเครื่องมือที่ใช้ในการเฝ้าสังเกต และการจำลองโหลดของงาน เพื่อใช้ในการทำการวัดเปรียบเทียบสมรรถนะสำหรับระบบที่จัดทำขึ้น เช่น FIO (Flexible I/O tester) ที่สามารถจำลองโหลดของงาน ได้หลากหลายประเภท เป็นต้น

1.3.3 ทำการออกแบบระบบ และเขียนโปรแกรมที่สอดคล้องเพิ่มเติมให้กับระบบ

1.3.4 นำระบบที่สร้างขึ้นไปทำการทดลองใช้งานจริง เพื่อทดสอบความสามารถ และประสิทธิภาพในการควบคุม แบนด์วิดท์ไอโอของหน่วยเก็บข้อมูล โดยทำการวัดค่า IOPS และแบนด์วิดท์ของแต่ละโดเมน เพื่อทำการเปรียบเทียบว่าระบบสามารถทำงานได้ตามที่ได้มีการออกแบบมาหรือไม่ และเกิดสิ่งผิดปกติภายใต้สภาวะการใช้งานจริงหรือเปล่า

1.3.5 นำผลการทดลองที่ได้มาสรุปผล โดยวิเคราะห์ว่าระบบที่ทำการสร้างขึ้นมานั้นมีความสามารถในการบริหารจัดการแบนด์วิดท์บนระบบแบบเสมือนได้ และมีประสิทธิภาพเป็นที่น่าพอใจหรือไม่ พร้อมทั้งอธิบายถึงผลที่เกิดขึ้นว่ามีความแตกต่างมากน้อยเพียงใดกับงานวิจัยอื่นๆ

1.4 ขอบเขตการวิจัย

1.4.1 ทำการสร้างและทดสอบระบบจัดการแบนด์วิดท์ บนระบบปฏิบัติการในตระกูลยูนิกซ์ โดยเลือกใช้ Ubuntu 16.04 กับ Linux kernel 4.4

1.4.2 ทำการสร้างและทดสอบระบบเวอร์ชวลไลเซชั่นแบบ Operating System-Based Virtualization เท่านั้น โดยมี Host เป็นลินุกซ์

1.4.3 เลือกใช้ Xen hypervisor เป็นซอฟต์แวร์เวชวลไลเชชัน และเลือกใช้สถาปัตยกรรมแบบ Para-virtualization

1.4.4 ทำการสร้างและทดสอบระบบบนแม่ข่ายเดี่ยว ซึ่งประกอบด้วยโดเมน (Guest domain) จำนวน 4 โดเมน และโดเมนควบคุม (Dom0) จำนวน 1 โดเมน

1.4.5 ในการทดสอบระบบใช้ FIO (Flexible I/O tester) เป็นตัวจำลองงาน โดยใช้การทดสอบการเขียน-อ่านตามลำดับ (Sequential Read-write)

1.4.6 ออกแบบ และพัฒนาระบบโดยใช้ภาษา Python 2.7.13 ร่วมกับ Shell script

1.4.7 ระบบเฝ้าสังเกต (monitor) ใช้ Bandwidth Monitor NG (bwm-ng) และ iostat ในการตรวจสอบการใช้งานทรัพยากรของระบบ

1.5 ประโยชน์ได้รับจากการวิจัย

1.5.1 เข้าใจถึงขั้นตอนวิธีในการควบคุมแบนด์วิดท์ไอโอของหน่วยเก็บข้อมูลบน Xen

1.5.2 ได้รับความรู้เกี่ยวกับ อัลกอริทึมที่ใช้ในการจัดลำดับการทำงานของดิสก์ รวมทั้งข้อดี ข้อเสีย และประสิทธิภาพในภาระงานที่แตกต่างกัน

1.5.3 ผู้ใช้งานสามารถควบคุมอัตราการแย่งชิงทรัพยากรหน่วยเก็บข้อมูล ได้อย่างมีเสถียรภาพ และมีประสิทธิภาพ โดยไม่ต้องทำการแก้ไขเคอร์เนลของระบบ

1.5.4 สามารถนำหลักการจัดการ แบนด์วิดท์ไอโอของหน่วยเก็บข้อมูล ไปใช้งานได้อย่างเหมาะสม ไม่ว่าจะเป็นการทดลอง หรือนำไปใช้งานในสภาวะจริง

1.5.5 สามารถนำระบบควบคุมแบนด์วิดท์ไอโอไปประยุกต์ใช้ร่วมกับระบบควบคุม หรืออัลกอริทึมอื่นๆ ที่อยู่ในระดับชั้นลึกลงไปได้ เพื่อเพิ่มประสิทธิภาพในการจัดการระบบ

1.5.6 การประยุกต์ใช้งานกลุ่มควบคุมนั้น นอกจากจะทำงานร่วมกับ Xen บนระบบลินุกซ์ได้แล้วยังสามารถนำไปประยุกต์ใช้กับระบบ ที่มีฐานของระบบเป็นลินุกซ์ได้ เช่น KVM และ Docker

1.6 งานวิจัยที่ได้รับการตีพิมพ์

ส่วนหนึ่งของวิทยานิพนธ์นี้ได้รับการตีพิมพ์ในรายงานสืบเนื่องจากการประชุมวิชาการระดับนานาชาติเรื่อง “Adaptive I/O Bandwidth Allocation for Virtualization Environment on Xen platform” โดยได้รับการตอบรับตีพิมพ์ในงานประชุมวิชาชื่อ 2nd International Conference on Intelligent Information Processing (ICIIP 2017) ซึ่งจัดขึ้นที่ประเทศไทย ระหว่างวันที่ 17 ถึง 18 กรกฎาคม 2560 ณ โรงแรมโนโวเทลกรุงเทพ สยามสแควร์

บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

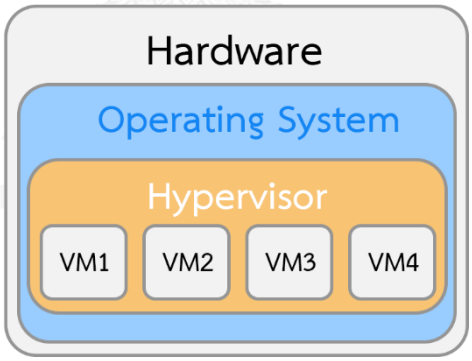
2.1 Virtualization

Virtualization คือ การจำลองเครื่องเสมือนด้วยซอฟต์แวร์ ที่ทำให้คอมพิวเตอร์หนึ่งเครื่องสามารถทำงานเป็นเครื่องเสมือนหลายๆ ระบบได้ โดยแต่ละรบบมีทรัพยากรหน่วยความจำ, ฮาร์ดดิสก์ และอุปกรณ์เน็ตเวิร์คเสมือนที่เป็นอิสระต่อกัน เครื่องเสมือนแต่ละเครื่องจึงสามารถมีระบบปฏิบัติการ และซอฟต์แวร์เป็นของตน เองอย่างอิสระ โดยและแต่ละ operating system จะถูกติดตั้งสู่เครื่อง Virtual machine แต่ละเครื่องในระบบ

2.1.1 Virtualization Method

2.1.1.1 Operating System-based Virtualization

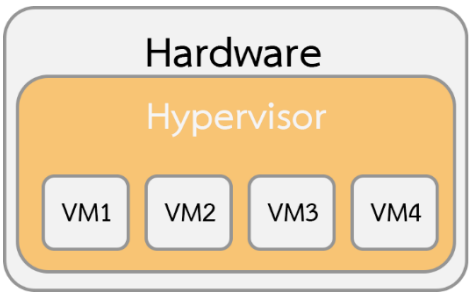
การใช้ software เพื่ออนุญาตให้ระบบสามารถสั่งการให้ OS หลายระบบทำงานพร้อม ๆ กันได้ ซึ่งระบบนี้จะต้องมี Host OS เพื่อทำการติดตั้ง virtualization software แสดงดังรูปที่ 1



รูปที่ 1 โครงสร้างของ Hardware-based Virtualization

2.1.1.2 Hardware-based Virtualization

เป็นระบบที่ไม่จำเป็นต้องใช้ operating system ในการติดตั้ง โดยจะเป็นการติดตั้ง virtualization software โดยตรงลงบนฮาร์ดแวร์ของ Host เพื่อให้ VM สามารถสื่อสารกับฮาร์ดแวร์โดยไม่ต้องผ่านตัวกลางอย่าง Host OS นั่นเอง

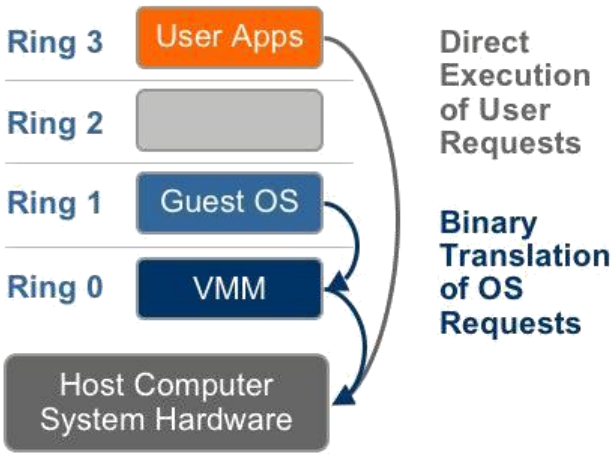


รูปที่ 2 โครงสร้างของ Hardware-based Virtualization

2.1.2 Virtualization Techniques [6]

2.1.2.1 Full Virtualization

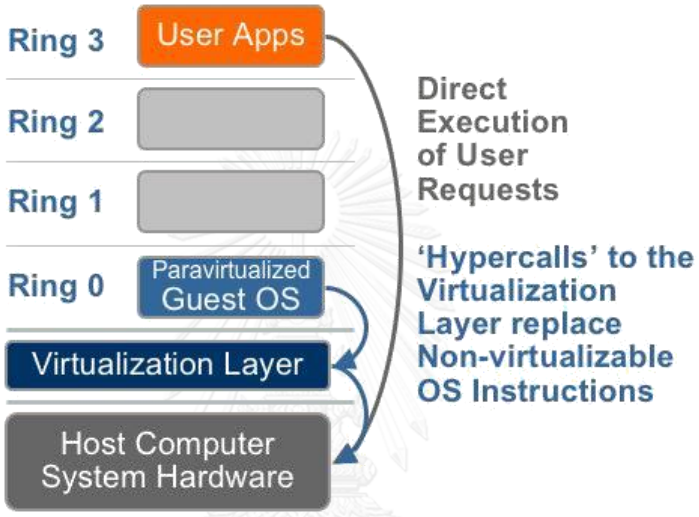
เทคนิคนี้เป็นการสร้างชุดคำสั่งที่ใช้แปลง Kernel Code ของระบบปฏิบัติการที่จะเรียกใช้บริการของฮาร์ดแวร์ แล้วนำไปวางดักไว้ก่อนที่ระบบปฏิบัติการจะติดต่อกับฮาร์ดแวร์ ซึ่งโดยปกติจะนำไปวางแทนที่ระบบปฏิบัติการในชั้นที่ 0 แล้วเลื่อนการทำงานของระบบปฏิบัติการมาอยู่ในชั้นที่ 1 โดยเมื่อระบบปฏิบัติการที่เป็น Guest OS ต้องการจะติดต่อกับฮาร์ดแวร์จะถูกแปลงคำสั่งโดยใช้ชุดคำสั่งใหม่ที่เป็นการทำงานแบบ Virtualization เป็นตัวจัดการ แสดงดังรูปที่ 3 เทคนิคนี้มีข้อดีคือไม่ต้องเปลี่ยนแปลงโครงสร้างภายในของระบบปฏิบัติการ ทั้งระบบปฏิบัติการหลักและระบบปฏิบัติการเสมือนทำให้การโอนย้าย Virtual Machine ระหว่างฮาร์ดแวร์จริงทำได้ง่าย แต่ก็มีข้อเสียคือจะเกิดค่า Overhead ในการทำงานที่สูงอันเนื่องมาจากการแปลงคำสั่ง โปรแกรมที่ใช้เทคโนโลยีแบบนี้ได้แก่ VMWare Workstation และ Microsoft Virtual Server



รูปที่ 3 เทคนิคแบบ Full-Virtualization

2.1.2.2 Para-virtualization

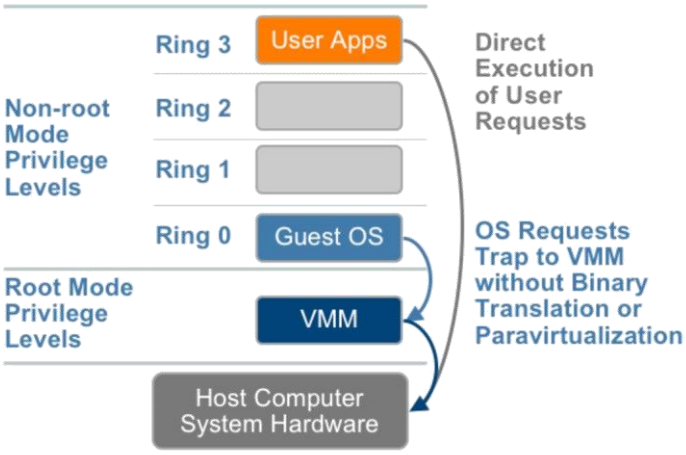
เป็นเทคนิคที่ทำให้Guest OS และ Hypervisor ทำงานติดต่อกันได้ดีมากขึ้นด้วยการปรับแต่ง Kernel ของ Guest OS ให้สามารถเรียกใช้คำสั่งผ่านHypervisor ได้โดยตรงโดยไม่ต้องมีการแปลงคำสั่งให้ได้ประสิทธิภาพใกล้เคียงการทำงานแบบปกติแต่ว่าข้อเสียก็คือเทคนิคแบบนี้สามารถใช้ได้กับเฉพาะระบบปฏิบัติการที่สามารถดัดแปลง Kernel Code ให้รองรับการทำงานแบบ Paravirtualization ได้เท่านั้นโปรแกรมที่ใช้เทคโนโลยีแบบนี้ได้แก่ Xen



รูปที่ 4 เทคนิคแบบ Para-virtualization

2.1.2.3 Hardware Assisted Virtualization

เป็นเทคนิคที่ถูกนำเสนอโดยผู้ผลิต CPU สองค่ายใหญ่คือ Intel (VT) และ AMD(AMD-V) โดยทั้งสองค่ายสร้างรูปแบบการทำงานของ CPU ที่อนุญาตให้ Virtualization Layer สามารถ Run อยู่ในชั้นพิเศษที่ต่ำกว่าชั้นที่0 สามารถดักจับคำสั่งที่เรียกใช้บริการ ฮาร์ดแวร์จาก Guest OS ที่ทำงานอยู่บน Layer ที่สูงกว่าได้โดยตรง ซึ่งไม่ต้องผ่านการทำ Binary Translation หรือ Paravirtualization มาก่อน

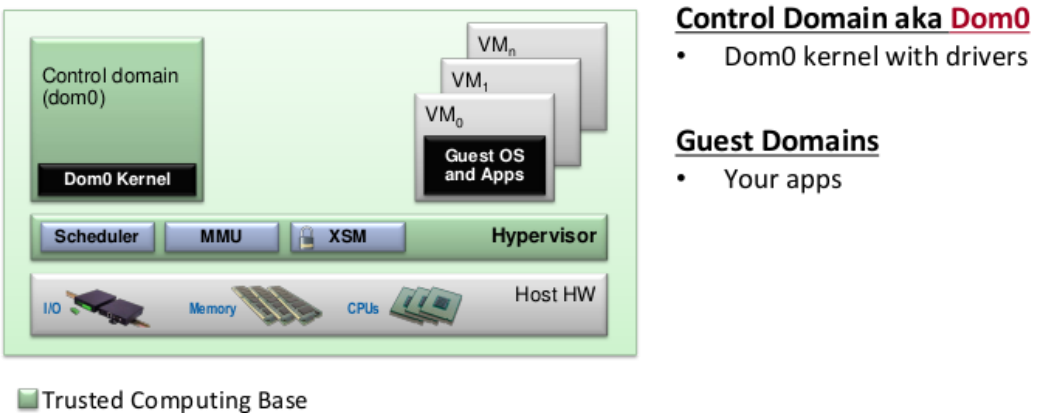


รูปที่ 5 เทคนิคแบบ Hardware Assisted Virtualization

2.1.3 Xen Hypervisor

คือระบบซอฟต์แวร์ที่อนุญาตให้มีการใช้ระบบปฏิบัติการแบบเสมือนได้หลายเครื่องพร้อมกันบนเครื่องกายภาพเพียงเครื่องเดียว โดยเฉพาะอย่างยิ่งโครงการที่ใช้ hypervisor ประเภท 1 (bare-metal) ซึ่งหมายความว่าจะทำงานได้โดยตรงแทนที่จะเป็นภายในระบบปฏิบัติการ นอกจากนั้น Xen ยังสนับสนุนทั้ง para-virtualization และ full virtualization อีกด้วย

เครื่องเสมือนที่ทำงานบน Xen Hypervisor เรียกว่า "โดเมน" และโดเมนพิเศษที่เรียกว่า Dom0 จะรับผิดชอบในการควบคุม hypervisor และเริ่มระบบปฏิบัติการอื่น ๆ ของ guest domain หรือ domUs เนื่องจากโดเมนเหล่านี้ไม่สิทธิพิเศษ ในแง่ที่ว่าพวกเขาไม่สามารถควบคุม hypervisor หรือเริ่มต้น และหยุดโดเมนอื่น ๆ ได้ โครงสร้างของ Xen แสดงในรูปที่ 6



รูปที่ 6 สถาปัตยกรรมพื้นฐานของ Xen Hypervisor

2.2 Scheduling algorithm

2.2.1 Disk Scheduling Algorithms

2.2.1.1 First Come -First Serve (FCFS)

วิธีนี้เป็นวิธีที่เรียบง่ายที่สุด process ใดที่มาถึงก่อนและอยู่ในคิวก่อน จะมีสิทธิ์ได้ใช้งานก่อน โดยหัวอ่านจะเคลื่อนไปอ่านข้อมูลที่ cylinder ตามลำดับของ process ที่อยู่ในคิว ข้อดีของวิธีนี้ก็คือ ทุก ๆ process จะได้รับบริการทั้งหมด แต่ข้อเสียก็คือ process ที่อยู่ในคิวท้าย ๆ จะต้องรอเป็นเวลานานมาก

2.2.1.2 Shortest Seek Time First (SSTF)

วิธีนี้จะดูว่าในบรรดา process ที่รอใช้งาน disk อยู่นั้น process ที่มีตำแหน่งของข้อมูลที่ต้องการอ่านอยู่ที่ cylinder ที่อยู่ใกล้กับหัวอ่านมากที่สุด ก็จะเลือก process นั้นให้ได้รับบริการก่อน วิธีนี้จะทำให้หัวอ่านมีระยะทางในการเคลื่อนจากตำแหน่งหนึ่งไปยังอีกตำแหน่งที่สั้นที่สุด หัวอ่านเคลื่อนไปหาตำแหน่งที่อยู่กับตำแหน่งที่หัวอ่านอยู่ครั้งสุดท้ายมากที่สุด ซึ่งจะทำให้หัวอ่านมีระยะทางในเคลื่อนน้อยที่สุด ข้อดีของวิธีนี้ก็คือทำให้มี bandwidth เพิ่มมากขึ้น แต่ข้อเสียก็คืออาจจะมีบาง process ที่ไม่ได้รับบริการเลย ทำให้เกิดการรอคอยอย่างไม่มีที่สิ้นสุด (starvation)

2.2.1.3 Elevator (SCAN)

วิธีนี้หัวอ่านจะอ่านกราดงานที่อยู่ในรายทางไปในทิศทางเดียวกันจนสุดที่ปลายของ cylinder สุดท้าย จากนั้นก็จะเริ่มอ่านกราดงานที่อยู่ในรายทางกลับมาในอีกทิศทางหนึ่ง ดังนั้น process ที่มีตำแหน่งของข้อมูลที่ต้องการอ่านอยู่ในทิศทางของการกราดของหัวอ่านจะได้รับการอ่านทั้งหมด ข้อดีของวิธีก็คือ process ที่อยู่ในคิวจะได้รับการบริการทั้งหมด ข้อเสียก็คืองานที่อยู่ที่ปลายอีกด้านหนึ่งของการอ่านจะต้องรอเป็นเวลานานมาก จนกว่างานอื่น ๆ จะได้รับการบริการทั้งหมดก่อนจึงจะได้รับการบริการ วิธี SCAN บางครั้งถูกเรียกว่าวิธี elevator เนื่องจากมีการทำงานเหมือนลิฟต์ที่จะให้บริการไปในทิศทางหนึ่งจนสิ้นสุดก่อนจึงจะเริ่มกลับมาให้บริการในอีกทิศทางหนึ่ง

2.2.1.4 Circular Scan (C-SCAN)

วิธีนี้เป็นวิธีที่ดัดแปลงจากวิธี SCAN โดยเมื่อหัวอ่านได้อ่านกราดงานตามรายทางไปจนถึง cylinder สุดท้ายแล้ว แทนที่จะเริ่มอ่านกราดกลับมาในอีกทิศทางหนึ่ง หัวอ่านจะกระโดดข้ามกลับมาที่ตำแหน่งแรกของ cylinder ทันทีแล้วจึงเริ่มอ่านกราดงานของที่อยู่รายทาง (มองเหมือนตำแหน่งสุดท้ายกับตำแหน่งเริ่มต้นอยู่ติดกันเป็นวงกลม) ข้อดีของวิธีนี้ก็คืองานทั้งหมดจะได้รับการบริการและงานที่อยู่ปลายขอบก็จะได้รับการบริการโดยไม่ต้องรอนานจนเกินไป แต่ข้อเสียก็คือหัวอ่านต้องเคลื่อนที่เป็นระยะทางไกลมาก (more seek time)

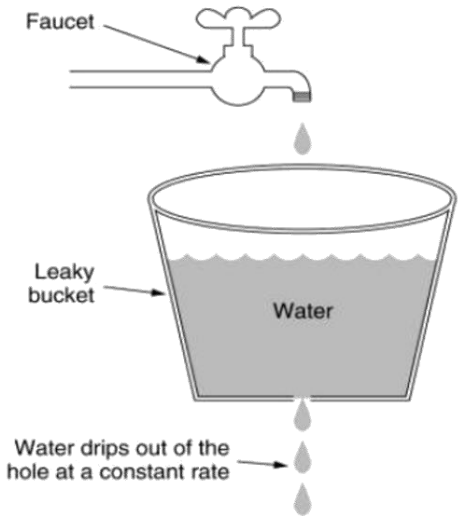
2.2.1.5 C-LOOK

วิธีนี้คล้าย ๆ กับวิธี C-SCAN แต่เมื่อหัวอ่านได้อ่านกราดไปจนถึงตำแหน่งที่มีงานอยู่ในตำแหน่งสุดท้ายแล้ว หัวอ่านจะไม่เคลื่อนที่ต่อไปแต่จะหยุดอยู่แค่ตำแหน่งนั้นแล้วกระโดดย้อนกลับมาหาตำแหน่งแรกของฝั่งตรงกันข้ามที่มีงานรออยู่ โดยไม่ไปถึงตำแหน่งสุดท้ายของฝั่งตรงกันข้ามเช่นเดียวกัน ข้อดีของวิธีนี้ก็คืองานทั้งหมดจะได้รับการบริการและงานที่อยู่ปลายขอบก็จะได้รับการบริการโดยไม่ต้องรอนานจนเกินไปและใช้เวลาในการเคลื่อนที่ของหัวอ่านน้อยกว่าวิธี C-SCAN ข้อเสียของวิธีนี้ก็คือต้องมีการกำหนดตัวแปรและการคำนวณล่วงหน้าที่ซับซ้อน

2.2.2 Traffic Shaping Algorithm

2.2.2.1 Leaky bucket [18]

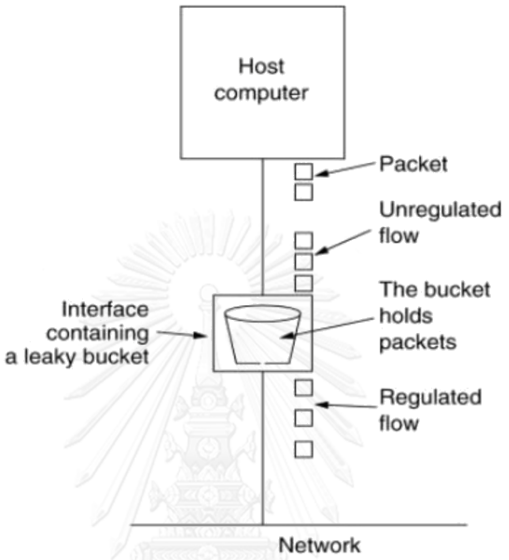
กลไกของการควบคุมแบบ Leaky Bucket ได้ถูกนำมาใช้งานในส่วนของกลไกของ Policing Mechanism บนเครือข่ายความเร็วสูงที่ต้องการแบนวิดธ์ที่มากและต้องการในเรื่องการรับประกันคุณภาพของการให้บริการ (Quality of Service) อีกด้วยซึ่งกลไกเช่นนี้จะช่วยทำให้แน่ใจได้ว่าข้อมูลที่เดินทางมาจากต้นทางนั้นจะไม่เดินทางเข้ามาในอัตราที่เร็วเกินกว่าที่กำหนดเอาไว้ซึ่งขนาดของบัฟเฟอร์นั้นอาจเปรียบได้กับขนาดของถังน้ำที่มีความจุเท่ากับ N โดยถ้าหากเฟรมข้อมูลได้เดินทางเข้ามาจนกระทั่งเกินกว่าความจุที่มีแล้ว เฟรมเหล่านั้นก็จะถูกทิ้งออกไป นั่นคือ กลไกการควบคุมนี้จะปล่อย ทราฟฟิคข้อมูลออกมาในอัตราที่คงที่ตามที่กำหนด ซึ่งวิธีการนี้ได้ถูกนำไปใช้กันโดยทั่วไปในการควบคุมปล่อยทราฟฟิคข้อมูลบนเครือข่ายความเร็วสูง



รูปที่ 7 แบบจำลองถังน้ำรั่ว

2.2.2.2 Token bucket [19]

เป็นอัลกอริทึมที่ใช้ในการควบคุมแพ็คเก็ตบนเครือข่ายคอมพิวเตอร์และเครือข่ายการสื่อสารโทรคมนาคม โดยยอมให้คิวส่งข้อมูลได้คงที่เช่นเดียวกับ leaky bucket แต่จะนับเป็น Byte ที่ส่ง สามารถยอมให้ส่งข้อมูลแบบ Burst ได้บ้าง แต่ค่าเฉลี่ยต้องไม่เกินค่าที่กำหนด โดยมีการกำหนด Token ให้กับผู้ส่งเป็นระยะ



รูปที่ 8 แบบจำลองถังโทเคน

2.2.3 Linux's Disk Scheduling Algorithm [20]

2.2.3.1 NOOP Scheduler

เป็น scheduler ที่เรียบง่ายที่สุดสำหรับ I/O schedulers โดยใช้หลักกการผสาน request เพื่อปรับปรุง throughput ให้ดีขึ้น ทุก requests จะถูกกำหนดคิวแบบ FIFO โดยไม่ลำดับความสำคัญสำหรับการ execution ซึ่งเป็นทางเลือกหนึ่งที่ดีสำหรับระบบที่ใช้หน่วยเก็บข้อมูลแบบ solid-state

2.2.3.2 Deadline Scheduler

นำ service deadline มาปรับใช้กับแต่ละ request ที่เข้ามาประมวลผลคิวสำหรับบริการถูกจัดลำดับความสำคัญด้วย deadline expiration เป็นทางเลือกที่ดีสำหรับแอปพลิเคชันแบบ real-time, databases และแอปพลิเคชันแบบ disk-intensive

2.2.3.3 Anticipatory Scheduler

ประมาณการณ์ว่า I/O request ที่เสร็จสิ้นแล้ว จะถูกตามด้วย request เพิ่มเติมจาก block ที่อยู่ใกล้กันก่อนที่จะย้ายไปยังคิวถัดไป ข้อดีคือเพิ่ม throughput ของดิสก์ แต่ก็มีอัตราเสี่ยงในการเพิ่มขึ้นอย่างเล็กน้อยของ latency

2.2.3.4 Completely Fair Queuing Scheduler (CFQ)

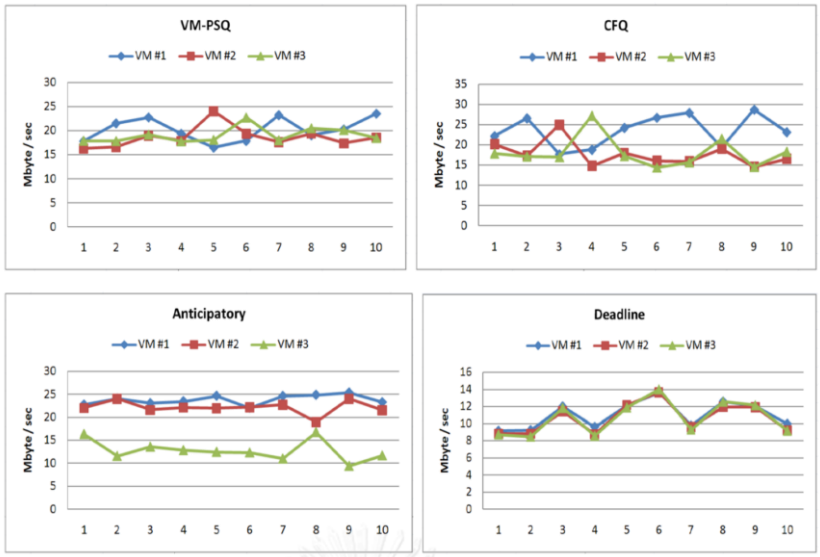
มีหน้าที่จัดการ process ที่มีการแย่งกันใช้ทรัพยากรด้วยความเท่าเทียม โดยแต่ละ process จะถูกกำหนด request queue แบบแยกกัน และ time slice สำหรับการเข้าถึง ดิสก์ ซึ่งเหมาะกับแอปพลิเคชันที่หลากหลาย โดยเฉพาะอย่างยิ่งระบบที่มีผู้ใช้งานหลายคน

2.3 งานวิจัยที่เกี่ยวข้อง

2.3.1 Proportional disk I/O bandwidth management for server virtualization environment

เป็นโครงการที่นำหลักการของ Schedule Token และ Time Slice มาประยุกต์ใช้เพื่อทำการแบ่งสัดส่วนแบนด์วิดท์ไอโอของดิสก์ ในหมู่ VM จำนวนมากตามลำดับความสำคัญ หรือตามความต้องการ I/O ใน runtime โดยใช้ชื่อว่า VM-PSQ [13] ส่วนของตัวจัดตารางนั้นถูกพัฒนาขึ้นในลำดับชั้นของ I/O scheduler layer เพื่อรับประกันประสิทธิภาพในการให้บริการสำหรับแต่ละ VM

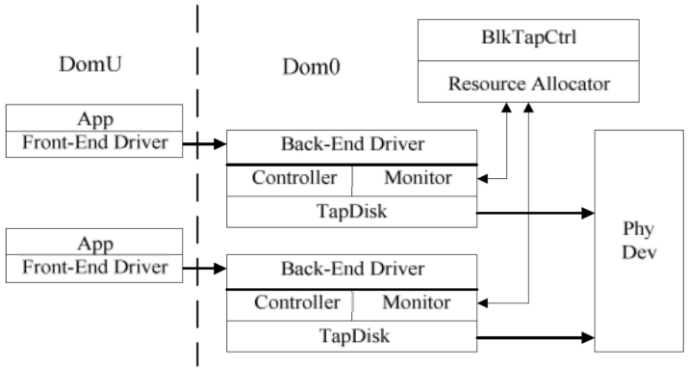
ขั้นตอนในการทำงานของระบบจัดคิวจะแบ่งสถานะออกเป็น 2 ส่วน คือ คิวที่พร้อมใช้งาน (active queue) และคิวที่ว่างงาน (idle queue) โดยคำร้องขอที่มีจำนวน token มากที่สุดจะได้รับการจัดคิวไว้หน้าสุดในคิวที่พร้อมใช้งาน เพื่อรับบริการก่อน ในการดำเนินการแต่ละรอบจำนวน token ของแต่ละกระบวนการจะถูกลดลงในทุกรอบ เมื่อ token ถูกลดลงจนเหลือศูนย์ กระบวนการนั้นจะถูกจัดไว้ในคิวที่ว่างงานเพื่อรอให้ คิวที่พร้อมใช้งานดำเนินการจนหมดก่อน และเมื่อ token รวมหมดลง ระบบจะทำการคืนค่าตั้งต้นให้กับทุก process เพื่อดำเนินการต่อไป ผลลัพธ์เมื่อเปรียบเทียบกับ fair share scheduling เป็นดัง *รูปที่ 9* สังเกตุได้ว่า VM-PSQ ให้ แบนด์วิดท์ที่ใกล้เคียงกับ CFQ มากที่สุดแต่มีค่าความผันผวนที่น้อยกว่าอย่างเห็นได้ชัด



รูปที่ 9 การเปรียบเทียบประสิทธิภาพของ VM-PSQ กับอัลกอริทึมในการจัดตารางของลินุกซ์

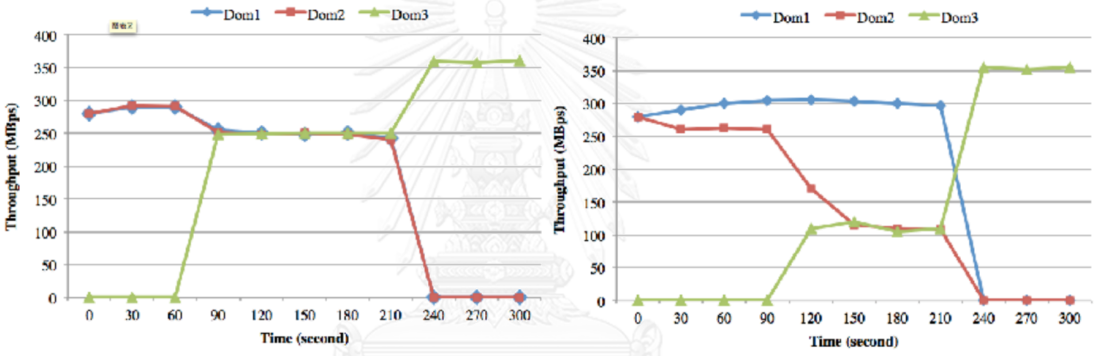
2.3.2 Weighted Fairness Resource Allocation of Disks in XEN

งานวิจัยนี้นำเสนออัลกอริทึมในการจัดสรรอุปกรณ์บล็อคเพื่อความเที่ยงธรรม และรักษาคุณภาพในการให้บริการ [17] โดยอัลกอริทึมตัวนี้มีแนวคิดที่ว่า เมื่อทำการแบ่งพื้นที่สำรองสำหรับ Guest domain แล้วถ้ามันไม่สามารถใช้แบนด์วิดท์ที่ถูกแบ่งเอาไว้ภายในหน่วยเวลาที่กำหนด อัลกอริทึมจะจัดสรร แบนด์วิดท์ให้กับโดเมนตัวนั้นลดลง ทางด้านแบนด์วิดท์ที่เหลือจะถูกกระจายไปยังโดเมน ที่มีความต้องการมากกว่า ทำให้ virtual domain มีโอกาศที่จะใช้งานทรัพยากรทางกายภาพได้อย่างเต็มที่ และหลีกเลี่ยงการจัดการที่ด้อยประสิทธิภาพ



รูปที่ 10 ตำแหน่งของระบบบควบคุมแบนด์วิดท์กับ backend driver

ระบบที่ใช้ในการแบ่งสรรทรัพยากรจัดทำขึ้นในส่วนของ backend driver ใน Dom0 โดยมีองค์ประกอบอยู่ 3 ส่วนด้วยกัน ได้แก่ Controller, Monitor และ Allocator แสดงดัง *รูปที่ 10* Controller และ Monitor ถูกวางไว้ในใน tapdisk ของแต่ละโดเมน เพื่อ implement การควบคุมทรัพยากร และ ตอบกลับสถานะไปยัง Allocator ซึ่ง Allocator จะมีแค่หนึ่งตัวที่ blktapctrl โดยมีหน้าที่ทำการพิจารณาการจัดสรรทรัพยากรด้วยสถานะที่ได้รับจากแต่ละ Guest domain. Controller และ Monitor นับเป็น process เดียวกันโดยแลกเปลี่ยนข้อมูลผ่าน global variables ส่วน Allocator จะทำงานเป็นอีก process หนึ่งโดยสื่อสารกับ tapdisk ผ่าน message queue โดยผลลัพธ์เมื่อทำการเปรียบเทียบกับการ allocation แบบไม่กำหนด policy แสดงใน *รูปที่ 11* พบว่าสามารถควบคุม throughput ได้เป็นไปตามหลักการ แต่เมื่อเหลือเพียง domain เดียวที่ยังทำงานอยู่ พบว่ามันสามารถใช้ bandwidth ได้เพียงที่สำรองเอาไว้เท่านั้น



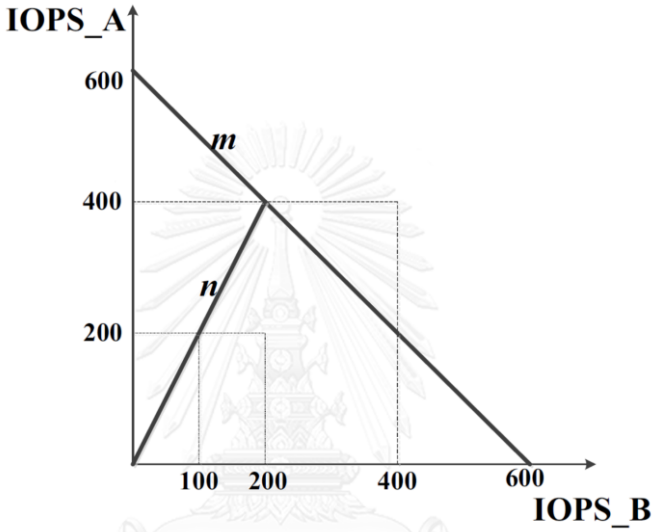
(a) With no weighted protection (b) With weighted allocation

รูปที่ 11 เปรียบเทียบการจัดสรรทรัพยากรโดยการถ่วงน้ำหนัก

2.3.3 IO QoS: A New Disk I/O Scheduler Module with QoS Guarantee for Cloud Platform

เป็นงานวิจัยที่ถูกพัฒนาขึ้นในปี 2012 [15] โดยมีแนวคิดในการทำ Scheduling สำหรับ Disk I/O โดยใช้ Leaky bucket และอัลกอริทึม Weighted fair queue เพื่อทำการควบคุมการ จัดสรรทรัพยากรดิสก์ ซึ่งอัลกอริทึมนี้ได้ทำการฝังลงในเคอร์เนลของลินุกซ์ พร้อมกับผลกระทบเพียงเล็กน้อยต่อระบบโดยรวม โดยหลักในการควบคุมจะใช้การจำกัดความสัมพันธ์ (Constrained relationship) ระหว่าง Throttle policy และ Weight policy แสดงความสัมพันธ์ใน *รูปที่ 12* ในกรณีนี้สมมติว่าอุปกรณ์พื้นฐานนี้ สามารถรองรับได้ 600 IOPS โดยมีดิสก์เสมือนสองตัวคือ A และ B ซึ่งน้ำหนักถูกตั้งไว้ที่ 2:1 และค่า throttle เท่ากับ 400 IOPS:400 IOPS ภายใต้เส้น m คือ สมรรถนะของอุปกรณ์พื้นฐาน (Underlying device capacity) ส่วนเส้นประทั้งสองเส้นคือการตั้งค่าของ throttle policy ภายในพื้นที่นี้ จำนวน I/O operation ของ A และ B เป็นไปตามเส้น n (weight policy settings)

เมื่อทั้ง A และ B เริ่มร้องขอ 200 IOPS พร้อมกัน หลังจาก A เสร็จสิ้นการดำเนินการแล้ว พบว่า B ดำเนินการไปเพียง 100 แต่ว่าผลรวมการดำเนินการของไอโอ ยังมีค่าต่ำกว่า capacity ด้วยเหตุนี้ B จึงเข้ายึดครองทรัพยากรเพื่อทำให้อีก 100 I/O operation เสร็จสมบูรณ์ ซึ่งพบว่า A และ B สามารถทำงานให้เสร็จสิ้นลงได้ โดยใช้ throughput อย่างเต็มที่ โดยเมื่อทำการประเมินผลฟังก์ชันต่างๆ พบว่าสามารถประกันคุณภาพในการให้บริการ โดยสูญเสียประสิทธิภาพลงเพียง 1% เท่านั้นในขณะที่โหลดโมดูลไปยังระบบ



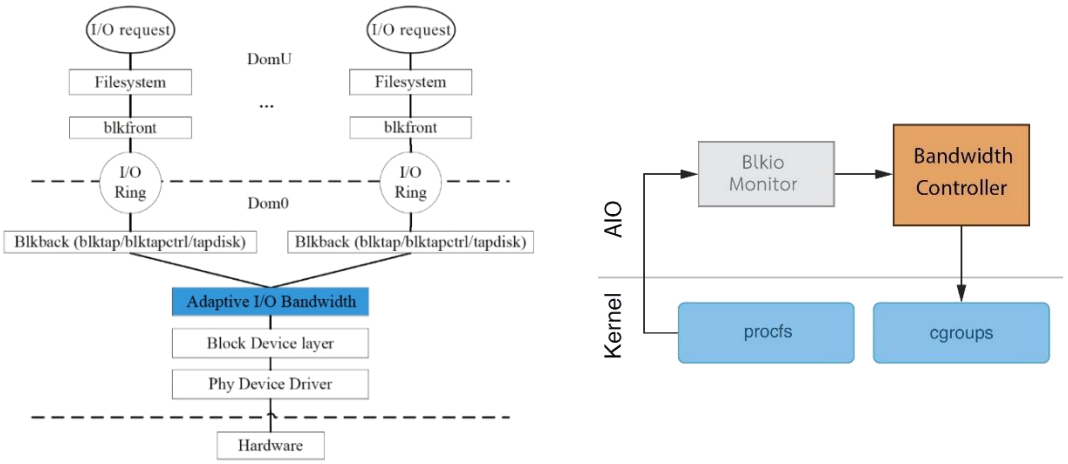
รูปที่ 12 การจำกัดความสัมพันธ์

บทที่ 3
กระบวนการควบคุมแบนด์วิดท์ไอโอของหน่วยเก็บข้อมูล

งานวิจัยนี้นำเสนอกระบวนการในการควบคุมแบนด์วิดท์ไอโอของหน่วยเก็บข้อมูล ระหว่างคอมพิวเตอร์เสมือนบนเครื่องแม่ข่ายเดี่ยวเพื่อศึกษาวิธีการในการควบคุมอัตราการเข้าถึงทรัพยากรหน่วยเก็บข้อมูลบนระบบแบบเวอร์ชวลไลซ์ และทำการวิเคราะห์ปัจจัยที่มีผลกระทบต่อ แบนด์วิดท์ไอโอของระบบ โดยมีเป้าหมายที่จะทำให้ระบบสามารถใช้งานทรัพยากรได้อย่างเต็มประสิทธิภาพ มีความยืดหยุ่นในการบริหารจัดการ และสามารถลดปัญหาจากการแย่งชิงทรัพยากรที่ยังคงเกิดขึ้นอยู่ในปัจจุบัน

3.1 กระบวนการไหลของไอโอ

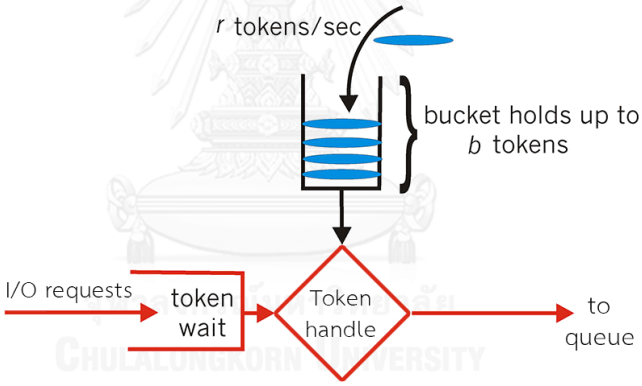
รูปที่ 13 แสดงสภาพแวดล้อมของ I/O ในระบบเวอร์ชวลไลซ์ด้วย Xen โดยแต่ละ VM จะดำเนินการร้องขอ I/O operation ผ่าน frontend driver ใน Guest domain และ backend driver ใน Host domain (Dom0) จากนั้นจึงส่ง storage operation มายังระบบจัดการแบนด์วิดท์ เพื่อกำหนดสิทธิในการเข้าถึงหน่วยเก็บข้อมูล และจัดคิวในการดำเนินการตามลำดับ โดยระบบนี้ทำหน้าที่เป็นส่วนหนึ่งของ Dom0 เพื่อควบคุมการ allocation ของทรัพยากรหน่วยเก็บข้อมูล (physical disk resources) ของ Host ให้แก่คอมพิวเตอร์เสมือนที่กำลังดำเนินการอยู่ในขณะนั้น ระบบที่นำเสนอจะพิจารณาการใช้งานหน่วยเก็บข้อมูล ของระบบพร้อมทั้งการกำหนดค่าควบคุมโดยผู้ใช้งาน เพื่อประสิทธิภาพในการใช้ทรัพยากรในกลุ่มของคอมพิวเตอร์เสมือน



รูปที่ 13 แผนภาพแสดงลำดับขั้นตอนของกระบวนการไอโอใน Xen

3.2 นโยบายในการจัดตาราง

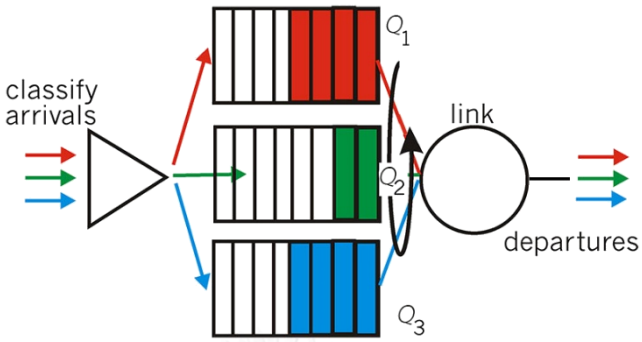
จากการศึกษาในเรื่องระบบคิว และการควบคุมปริมาณการสื่อสารของแพ็กเกตข้อมูล จึงนำ Hierarchical Token Bucket (HTB) [21] มาใช้ในการกำหนดคิวของ I/O request สำหรับแต่ละ VM เนื่องจากการใช้ HTB จะมีส่วนช่วยเพิ่มความยืดหยุ่นในการจัดสรรแบนด์วิดท์จากการขอยืมโทเคน และจำกัดการแย่งชิงแบนด์วิดท์ของเหล่า VM ได้โดยง่าย ทั้งนี้จากงานวิจัยที่ผ่านมาพบว่า Token bucket ยังสามารถยอมให้มีการ burst สำหรับ traffic ในบางช่วงเวลาอีกด้วย โดยแต่ละ pending I/O request จะใช้หนึ่งโทเคนเพื่อส่งคำขอไปยังคิวเพื่อประมวลผล เมื่อไรก็ตามที่ VM ใช้ token จนหมด request ที่ถูกส่งมา ต้องรอการเติม token ใหม่จึงจะสามารถส่งคำขอไปยังคิวได้ ลำดับการทำงานแสดงดัง รูปที่ 14 ซึ่ง Token Bucket ประกอบไปด้วย ถังโทเคนที่สามารจุได้ b tokens โดยโทเคนจะถูกเติมลงที่อัตรา r tokens ต่อวินาที โดยถังเก็บจะไม่สามารถรองรับโทเคนได้มากกว่า b อีกนัยหนึ่งคือ แพ็กเกตจะคอยจนกระทั่ง มีปริมาณโทเคนที่เพียงพอในถัง มันจึงจะทำการส่งไปยังคิว



รูปที่ 14 Token Bucket Policer

HTB สนับสนุน TB หลายตัว และจัดตารางโดยใช้ Deficit Round Robin scheduler (DRR) เพื่อให้แต่ละคิวได้รับการบริการอย่างทั่วถึง โดย DRR แสดงให้เห็นถึงรุ่นที่มีการปรับปรุงของ WRR (Weighted Round Robin) สิ่งที่เหนือกว่าจากการปรับปรุง WRR คือ มันสามารถจัดการกับแพ็กเกตที่มีขนาดแตกต่างกันรวมทั้ง การให้บริการคิวโดยขึ้นอยู่กับค่า deficit ค่าควอนตัมจะถูกนำมาคำนวณด้วย โดยในแต่ละคิวอาจจะมีค่าควอนตัมที่แตกต่างกัน ในแต่ละคลาสจะสามารถรับปริมาณของ service ที่แตกต่างกันได้ในแต่ละช่วงเวลา โดยคลาส i ถูกกำหนดน้ำหนักด้วย W_i ในระหว่างช่วงเวลาใดๆ ที่คลาส i ทำการส่ง I/O request จะมีการรับประกันว่าจะได้รับการ service เท่ากับ $Q_i/(\sum Q_j)$

ซึ่ง $\sum Q_j$ จะถูกแทนที่ด้วย class ทั้งหมดที่มีการ queue I/O request การดำเนินการแสดงดัง *รูปที่ 15*



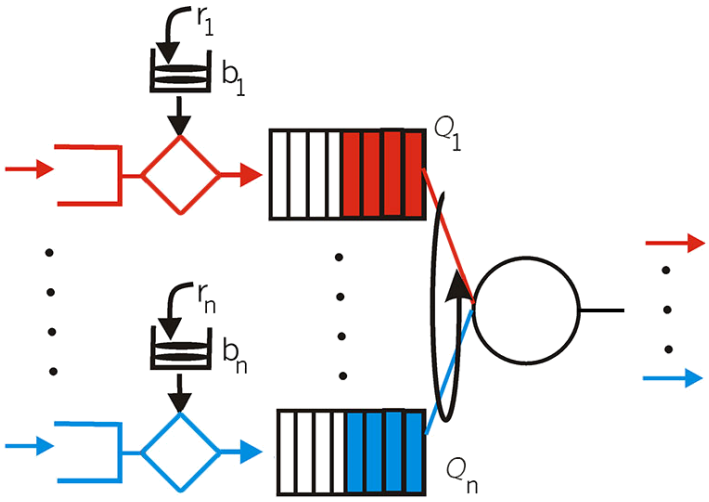
รูปที่ 15 Deficit Round Robin (DRR)

HTB ใช้แนวคิดของ tokens buckets พร้อมด้วยระบบแบบ class-based hierarchical เพื่อการควบคุมแพ็กเกตที่มีความละเอียดซับซ้อน โดยยอมให้ผู้ใช้งานกำหนดคุณลักษณะของโทเคน และถังเก็บ พร้อมทั้งสามารถกำหนด rate และ ceil โดย Rate คืออัตราการส่งข้อมูลที่แน่นอน ในขณะที่ Ceil คืออัตราการส่งข้อมูลสูงสุด ใน HTB คลาสจะถูกกำหนดแทนด้วยต้นไม้ตามความสัมพันธ์ของ traffic aggregations มีเพียง leaf class เท่านั้นที่มีคิวสำหรับบัฟเฟอร์แพ็กเกตที่เป็นของคลาสนั้นๆ คลาสลูก (children class) จะยืมโทเคนจาก parent ของมันเมื่อการไหลของแพ็กเกตถึงอัตราที่จำกัด โดยคลาสลูกจะพยายามขอยืมโทเคนอย่างต่อเนื่องจนกว่าจะถึงค่า ceil สำหรับในแต่ละคลาส c อัตราที่อนุญาตจะถูกกำหนดดังนี้

$$R_c = \min(CR_c,\ AR_c + B_c) \tag{1}$$

$$B_c = \begin{cases} \dfrac{Q_c R_p}{\sum_{i \epsilon D(p)} Q_i \text{ where } P_i = P_c} & \text{if } \min_{i \epsilon D(p)}[P_i] \geq P_c \\ 0 & \text{Otherwise} \end{cases} \tag{2}$$

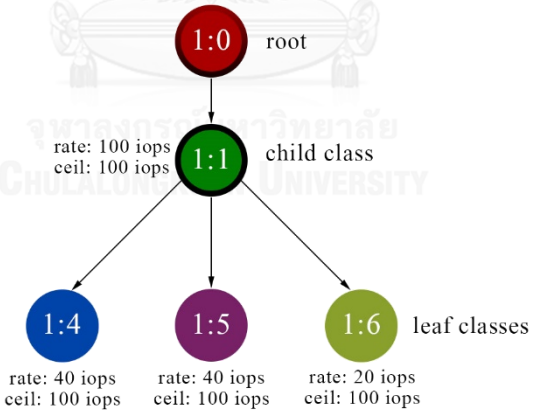
เมื่อ Rc คือเรทที่อนุญาตของคลาส c ในขณะที่ CRc, ARc และ Bc คือเรทการส่งข้อมูลสูงสุด, เรทที่แน่นอน และเรทการขอยืมของคลาสตามลำดับ ส่วนความหมายของ Bc ในสมการที่ 2 คือการลำดับความสำคัญในคิว เมื่อไม่มี parent ค่า Bc = 0 ส่วนที่เกินอัตรา (Rp) ที่ยืมมาจาก parent จะถูกแบ่งไปยัง backlogged leave ที่มีการจัดลำดับความสำคัญสูงสุดตามค่าควอนตัม (Qi) โดยการดำเนินการจะเป็นไปตาม รูปที่ 16



รูปที่ 16 Hierarchical Token Bucket flows

3.3 ตัวอย่างการทำงานของอัลกอริทึม

สันนิษฐานว่าอุปกรณ์ดิสก์ของเราสามารถรองรับได้ 100 IOPS ที่บล็อคขนาด 64KB ซึ่งมี 3 VM กำลังขอใช้งานดิสก์ โดยมีการกำหนดสิทธิที่ระดับเดียวกัน และมีอัตราสูงสุดที่สามารถรองรับได้สำหรับแต่ละโดเมนเท่ากับ 100 IOPS การจำกัด disk bandwidth เป็นไปตาม รูปที่ 17



Domain ID	Limit-at (IOPS)	Max-limit (IOPS)
DOM1 (1:4)	40	100
DOM2 (1:5)	40	
DOM3 (1:6)	20	

รูปที่ 17 ตัวอย่าง HTB เมื่ออัตราส่วนในการขอยืม rate เท่ากัน

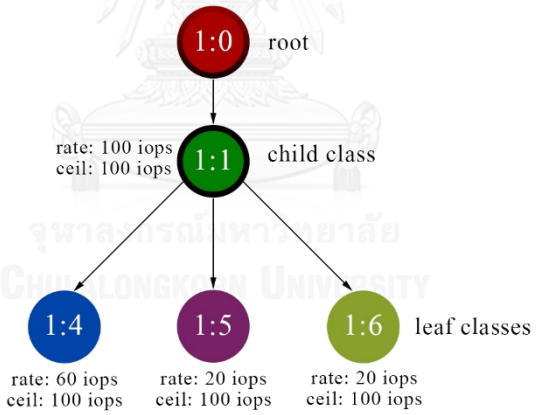
ตัวอย่าง Dom3 หยุดการเข้าถึงดิสก์ในขณะที่ Dom1 และ Dom2 ยังคงเรียกใช้งานดิสก์อยู่

กรณีที่ 1 อัตราส่วนในการขอยืม rate ของแต่ละโดเมนมีค่า*เท่ากัน* (รูปที่ 17)

ขณะที่ Dom1 และ Dom2 กำลังขอ rate ที่เพิ่มขึ้นจากคลาส 1:1 ซึ่งพบว่า Dom3 ไม่มีการใช้งานดิสก์อยู่ ดังนั้นทั้งสองโดเมนจึงขอยืมแบนด์วิดท์ผ่านคลาส 1:1 โดยแบ่งในอัตราที่เท่ากัน เนื่องมาจากอัตราส่วน rate ของ ทั้งสองโดเมนนั้นมีค่าเท่ากัน ในที่นี้ Dom1 และ Dom3 จะได้รับแบนด์วิดท์เพิ่มขึ้นเป็น 50 IOPS สำหรับในแต่ละโดเมน

กรณีที่ 2 อัตราส่วนในการขอยืม rate ของแต่ละโดเมนมีค่า*ไม่เท่ากัน* (รูปที่ 18)

ในกรณีนี้อัตราส่วน rate ของ ทั้งสองโดเมนไม่เท่ากัน เมื่อมีการขอยืมแบนด์วิดท์แต่ละคลาสจะได้รับตามสัดส่วนของตน ในที่นี้ Dom1 จะได้รับแบนด์วิดท์เพิ่มเป็น 75 IOPS ในขณะที่ Dom3 ได้รับเพิ่มเป็น 25 IOPS ตามอัตราส่วน 3:1

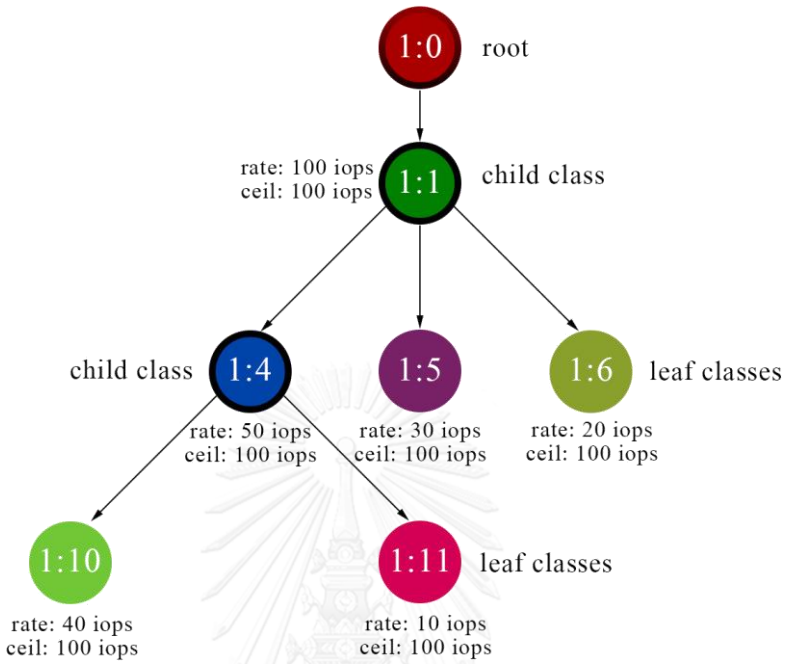


รูปที่ 18 ตัวอย่าง HTB เมื่ออัตราส่วนในการขอยืม rate ไม่เท่ากัน

กรณีที่ 3 มีการแบ่งคลาสย่อยเพิ่ม (รูปที่ 19)

ในกรณีนี้จะคล้ายกับในกรณีที่1 เพียงแต่มีการกำหนดคลาสย่อยเพิ่มเติม โดย rate ของ Dom1 และ Dom2 จะถูกกำหนดโดยคลาส 1:4 (50 IOPS) แต่ยังสามารถมีอัตราการใช้งานแบนด์วิดท์สูงสุดได้ถึง 100 IOPS เมื่อมีการใช้งานเพียงแค่โดเมนเดียว ยกตัวอย่างคือ Dom1 มีอัตราที่กำหนดอยู่ที่ 40 IOPS แต่มีความต้องการที่มากขึ้น ซึ่งในขณะนั้นไม่มีโดเมนใดที่กำลังใช้งานอยู่เลย Dom1 จึงขอยืมแบนด์วิดท์จากคลาส 1:4 ที่อัตรา 10 IOPS ทำให้ในขณะนั้น Dom1 มีแบนด์วิดท์อยู่ที่ 50 IOPS แต่ว่ายังมี

ความต้องการที่เพิ่มขึ้นอีก คลาส 1:4 จึงขอยืมจากคลาส 1:1 มาอีกทอดหนึ่ง เพื่อให้ Dom1 สามารถใช้งานแบนด์วิดท์ได้เต็มที่ (100 IOPS) เมื่อโดเมนอื่นๆ ไม่มีการใช้งานนั่นเอง



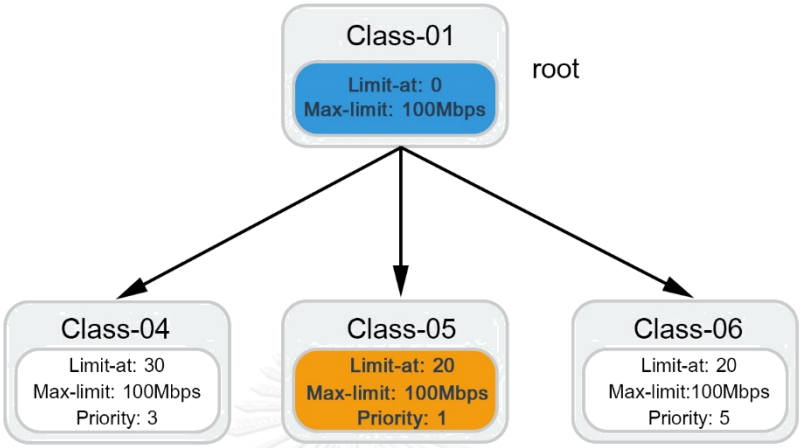
รูปที่ 19 ตัวอย่าง HTB แบบแบ่งคลาสเพิ่ม

ลำดับขั้นตอนการดำเนินการ

1. สั่งเขียนไฟล์จาก Dom1 ขนาด 1GB ด้วย block size ขนาด 256KB
2. การดำเนินการจะถูกส่งต่อไปยังยัง Dom0 ด้วย frontend driver
3. Backend driver รับ I/O request จาก Dom0 แล้วจึงส่งมายังระบบจัดการแบนด์วิดท์
4. ถ้ามีโทเคนเพียงพอสำหรับบล็อกขนาด 256 KB โทเคนจะถูกหักออก แล้วทำการส่ง request ไปยังคิวเพื่อจัดลำดับในการดำเนินการ
5. จากนั้น request จะถูกส่งไปยัง Block Device Layer เพื่อสื่อสารกับหน่วยเก็บข้อมูลในการเขียนไฟล์ตามลำดับ

3.4 ตัวอย่างการทำงานของอัลกอริทึมเมื่อมีการกำหนดลำดับความสำคัญ

3.4.1 กรณีที่1 แบ่งโดเมนออกเป็น 3 คลาส (รูปที่ 20)



Class ID	Limit-at (Mbps)	Max-limit (Mbps)	Priority
Class-01 (root)	0	100	-
Class-04 (leaf)	30	100	3
Class-05 (leaf)	20	100	1
Class-06 (leaf)	20	100	5

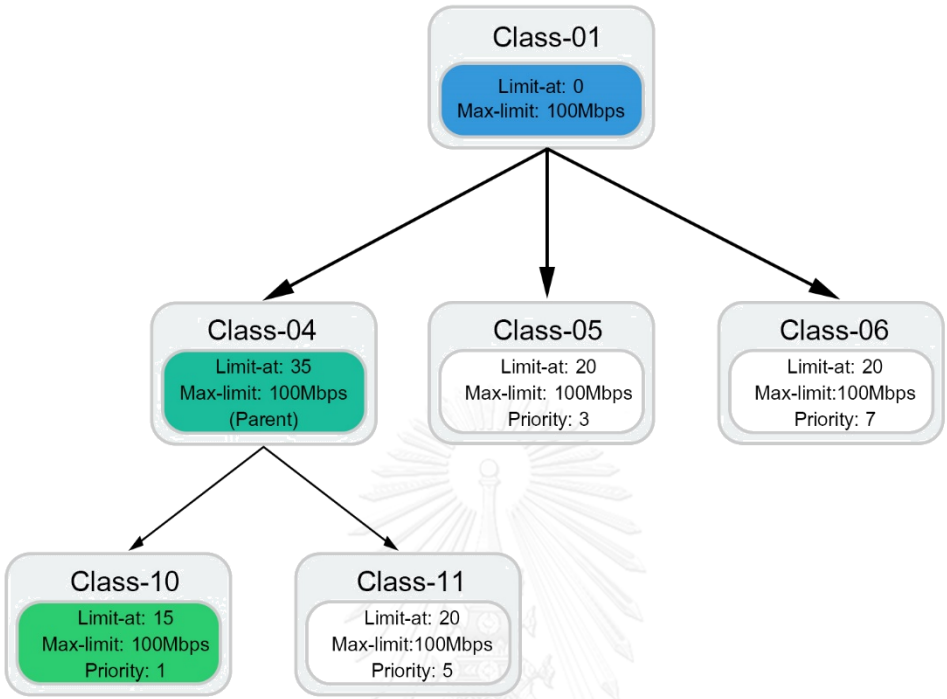
รูปที่ 20 ตัวอย่างการแบ่งคลาสเมื่อมีการจัดลำดับความสำคัญของโดเมน

ผลลัพธ์ของกรณีที่ 1 แบบมีการจัดลำดับความสำคัญ

- Class-04 จะได้รับ 30Mbps
- Class-05 จะได้รับ 50Mbps
- Class-06 จะได้รับ 20Mbps

เมื่อได้รับแบนด์วิดท์ที่กำหนดจากการตั้งค่า ระบบจะมอบแบนด์วิดท์ที่ยังไม่ถูกใช้งาน หรือคงเหลืออยู่ให้กับคลาสที่มีการจัดลำดับความสำคัญมากที่สุด ซึ่งในกรณีนี้คือ Class-05 โดยจะได้รับแบด์วิดท์เพิ่มขึ้นอีก 30 Mbps ทำให้ Class-10 มีแบนด์วิดท์เพิ่มขึ้นเป็น 50Mbps เมื่อคลาสนี้ไม่ถูกใช้งานก็จะมอบทรัพยากรคืนให้แก่ระบบ จากนั้นจึงนำทรัพยากรที่คงเหลือนี้ส่งต่อให้กับโดเมนที่มีความต้องการใช้งานซึ่งมีการจัดลำดับความสำคัญสูงที่สุดในเวลานั้นต่อไป

3.4.2 กรณีที่ 2 แบ่งโดเมนออกเป็น 4 คลาสโดยมีการเพิ่มลำดับชั้นด้วยคลาสแม่ (รูปที่ 21)



Class ID	Limit-at (Mbps)	Max-limit (Mbps)	Priority
Class-01 (root)	0	100	-
Class-04 (parent)	40	100	-
Class-05 (leaf)	20	100	3
Class-06 (leaf)	20	100	7
Class-10 (leaf-inner)	15	100	1
Class-11 (leaf-inner)	20	100	5

รูปที่ 21 ตัวอย่างการแบ่งคลาสเมื่อมีการจัดลำดับความสำคัญของโดเมนแบบมีคลาสแม่

ผลลัพธ์ของกรณีที่ 2 แบบมีการจัดลำดับความสำคัญ

- Class-05 จะได้รับ 20Mbps
- Class-06 จะได้รับ 20Mbps
- Class-10 จะได้รับ 40Mbps
- Class-11 จะได้รับ 20Mbps

หลังจากแต่ละคลาสได้รับแบนด์วิดท์ตามที่กำหนดเอาไว้แล้ว ระบบจะมอบแบนด์วิดท์ที่ยังไม่ถูกใช้งาน หรือคงเหลืออยู่ให้กับคลาสที่มีการจัดลำดับความสำคัญมากที่สุดก่อนเสมอ ซึ่งในกรณีนี้ Class-10 จากที่ถูกกำหนดไว้ 15Mbps จะได้รับแบนด์วิดท์เพิ่มขึ้นเป็น 40Mbps เนื่องจากเมื่อทุกคลาสได้รับแบนด์วิดท์ครบคามที่ถูกกำหนดแล้วระบบยังมีทรัพยากรเหลืออยู่อีก 25Mbps ดังนั้นแบนด์วิดท์ที่เหลือนี้จะถูกมอบให้กับคลาสที่มีการจะดลำดับความสำคัญสูงที่สุดนั่นก็คือ Class-10 ซึ่งมีค่า priority อยู่ที่ 1 นั่นเอง

3.5 ขั้นตอนการดำเนินการของระบบจัดแบ่งแบนด์วิดท์ของไอโอแบบปรับเปลี่ยนได้

3.5.1 ขั้นตอนตรวจสอบการตั้งค่า

3.5.1.1 ในกรณีที่ไม่มีไฟล์การตั้งค่า หรือต้องการตั้งค่าใหม่ ระบบจะเริ่มให้ผู้ใช้งานกำหนดค่าแบนด์วิดท์สูงสุดของระบบ จากนั้นจึงกำหนดค่า อัตราขั้นต่ำ, อัตราสูงสุด และ ค่าลำดับความสำคัญ สำหรับแต่ละโดเมน แล้วจึงทำการบันทึกลงในไฟล์การตั้งค่า

3.5.1.2 ในกรณีที่มีไฟล์การตั้งค่าระบบจะกำหนดตัวแปรที่ใช้โดยรับข้อมูลจากไฟล์ที่ได้ถูกกำหนดค่าเอาไว้

3.5.2 ขั้นตอนกำหนดค่าเริ่มต้นให้กับ cgroup

3.5.2.1 ใช้อัตราขั้นต่ำในการส่งค่าไปยังกลุ่มควบคุม ซึ่งถูกกำหนดเอาไว้สำหรับในแต่ละโดเมนในขั้นตอนของการตั้งค่า

3.5.3 ขั้นตอนการเฝ้าสังเกตอัตราการเข้าถึงข้อมูล

3.5.3.1 ในส่วนของระบบเฝ้าสังเกต ตรวจสอบการใช้งานทรัพยากรหน่วยเก็บข้อมูลของระบบ ณ เวลาปัจจุบัน โดยใช้ bwm-ng

3.5.3.2 นั้นจึงนำผลที่วัดได้ได้ไปทำการวิเคราะห์ในขั้นตอนต่อไป โดยลักษณะข้อมูลของ bwm-ng จะมีลักษณะตามรูปที่ 24

3.5.4 ขั้นตอนการกำหนดสถานะให้กับโดเมน

3.5.4.1 ทำการตรวจสอบสถานะของโดเมนจากระบบเฝ้าสังเกต จากข้อ 4.4.3 โดยแบ่งออกเป็น 2 สถานะคือ สถานะว่างงาน (idle) และสถานะดำเนินการ (active)

3.5.4.2 สถานะดำเนินการ จะถูกกำหนดให้กับโดเมนที่มีการใช้งานทรัพยากรหน่วยเก็บข้อมูลอยู่ในขณะนั้น หรือมีการเขียน-อ่านเกิดขึ้นโดยมีอัตราไม่ต่ำกว่า 5 เปอร์เซ็นต์ของอัตราต่ำสุด

3.5.4.3 สถานะว่างงาน จะถูกกำหนดให้กับโดเมนที่ไม่มีการใช้งานทรัพยากรหน่วยเก็บข้อมูลอยู่ในขณะนั้น หรือมีการเขียน-อ่านเกิดขึ้นโดยมีอัตราที่ต่ำกว่า 5 เปอร์เซ็นต์ของอัตราต่ำสุด ซึ่งการเปลี่ยนสถานะเป็น idle นั้น โดเมนจะต้องถูกพิจารณาว่ามีสถานะเป็น idle จากการเก็บข้อมูลในข้อ 4.4.3 เป็นจำนวน 5 รอบการทำงาน หรือ 0.5 วินาที เพื่อยืนยันว่าโดเมนนั้นไม่ต้องการเข้าถึงทรัพยากรหน่วยเก็บข้อมูลอีก ณ เวลานั้น

3.5.5 ขั้นตอนการตรวจสอบแบนด์วิดท์ส่วนกลาง

3.5.5.1 ต่อมาจึงทำการคำนวณหาค่าแบนด์วิดท์ส่วนกลาง เพื่อนำไปเพิ่มให้โดเมนที่ต้องการใช้งาน โดยนำแบนด์วิดท์ของระบบทั้งหมดหักออกด้วยแบนด์วิดท์ปัจจุบัน ของโดเมนที่ยังดำเนินการอยู่

3.5.6 ขั้นตอนการตรวจสอบลำดับความสำคัญของโดเมน

3.5.6.1 ตรวจสอบโดเมนที่มีค่าลำดับความสำคัญ (priority) สูงที่สุด ที่มีสถานะดำเนินการ (active) จากนั้นเปลี่ยนสถานะของโดเมนเหล่านั้นเป็น eager

3.5.7 ขั้นตอนการปรับค่าแบนด์วิดท์ของโดเมน

3.5.7.1 ทำการปรับค่าแบนด์วิดท์ปัจจุบัน ให้กับโดเมนที่มีสถานะ eager โดยเพิ่มจากค่าแบนด์วิดท์ส่วนกลางที่ได้จากข้อ 4.4.5

3.5.8 ขั้นตอนการปรับอัตราที่เปลี่ยนแปลงใน cgroup

3.5.8.1 ถ้าโดเมนมีมีสถานะเป็น “eager” หรือได้รับแบนด์วิดท์เพิ่มขึ้นจากส่วนกลาง ระบบจะส่งการตั้งค่าแบนด์วิดท์ปัจจุบันไปยังกลุ่มควบคุม เพื่อปรับแบนด์วิดท์ที่ได้รับเพิ่มจากส่วนกลางให้แก่โดเมนนั้น

3.5.8.2 ถ้าโดเมนไม่มีการใช้งานดิสก์ และได้รับการเปลี่ยนสถานะเป็น “idle” ระบบจะส่งการตั้งค่า อัตราขั้นต่ำไปยังกลุ่มควบคุมเพื่อเป็นค่าเริ่มต้นในการดำเนินการครั้งต่อไปของโดเมน

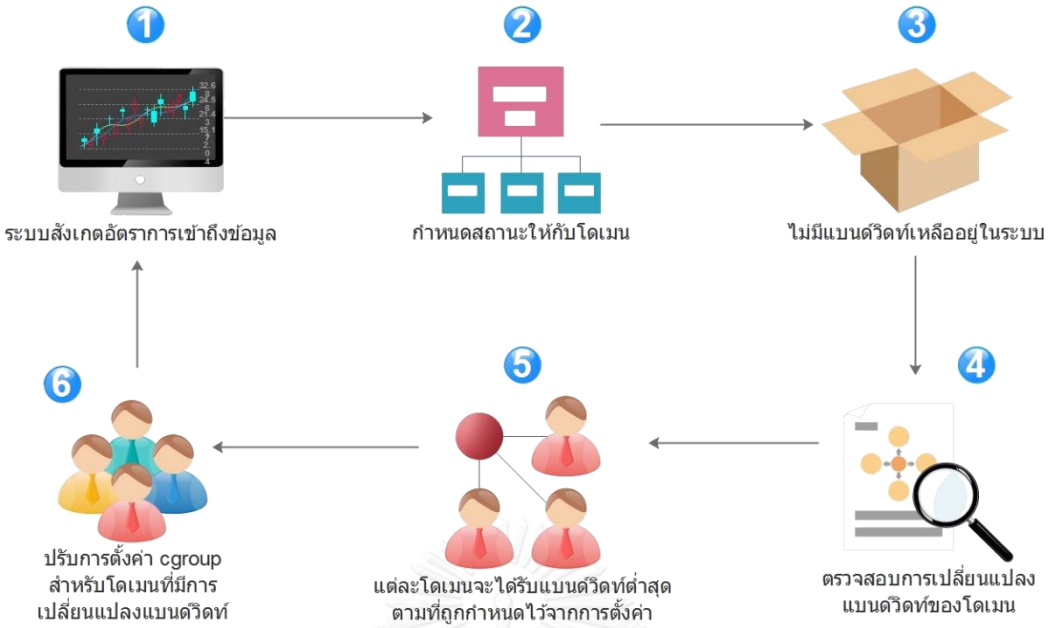
3.5.8.3 หลังจากนั้นระบบจะเริ่มสังเกตอัตตราการเข้าถึงข้อมูลอีกครั้งในหัวข้อ 4.4.3 โดยระบบสามารถแสดง log การทำงานดัง รูปที่ 25

อัลกอริทึมที่ใช้ในการจัดแบ่งแบนด์วิดท์ของ AIO แสดงในรูปที่ 22 ลำดับขั้นตอนการทำงานของ AIO เมื่อมีแบนด์วิดท์เหลือในระบบ และไม่มีแบนด์วิดท์เหลือในระบบแสดงในรูปที่ 23 และ 24 ตามลำดับ

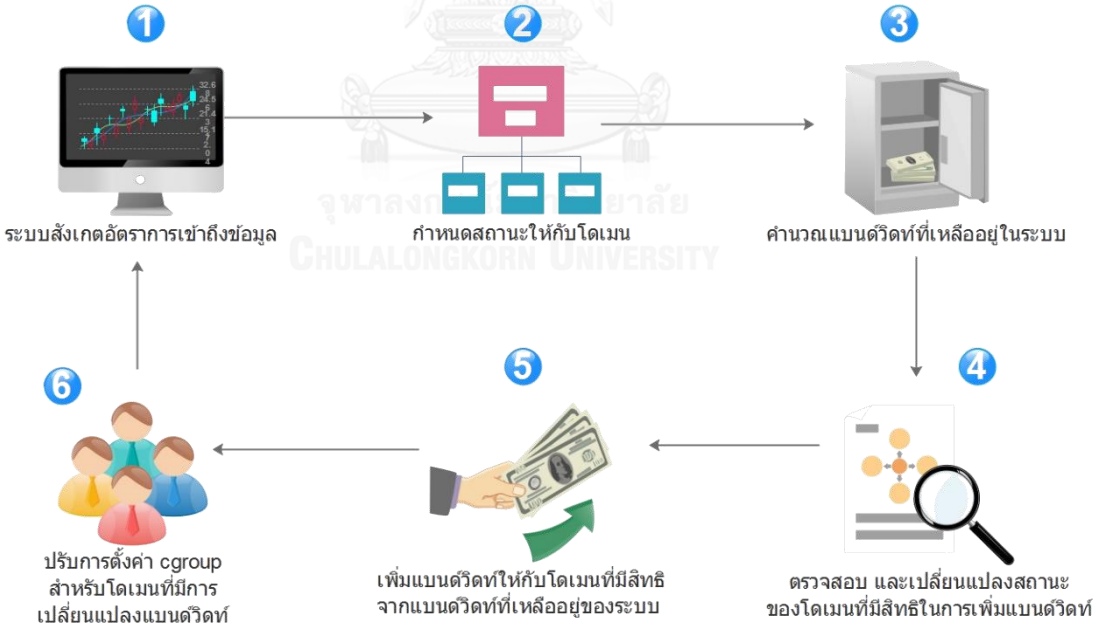
Algorithm 1: AIO Algorithms

```
for u ← 1 to totalDomain do
    dom[u].bandwidth ← monitor[u].bandwidth
for u ← 1 to totalDomain do
    if dom[u].write = 0 or dom[u].write < dom[u].error then
        if dom[u].idleCount < 5 then
            dom[u].idleCount ← dom[u].idleCount + 1
        else
            dom[u].status ← "idle"
            dom[u].current ← 0
    else
        dom[u].status ← "active"
        dom[u].current ← dom[u].bandwidth
        dom[u].idleCount ← 1
for u ← 1 to totalDomain do
    if dom[u].priority < root.priority and dom[u].status! = "idle" then
        root.priority ← dom[u].priority
for u ← 1 to totalDomain do
    root.eagerCount ← eagerCount + 1
if eagerCount ≠ 0 then
    root.distribute ← root.remaining/eagerCount
for u ← 1 to totalDomain do
    if status = "eager" then
        dom[u].tempRate ← dom[u].rate + root.distribute
    else if status = "active" then
        dom[u].tempRate ← dom[u].rate
    else
        dom[u].tempRate ← 0
for u ← 1 to totalDomain do
    if cgroup[u].rate ≠ dom[u].rate and dom[u].tempRate = 0 then
        SET cgroup[u].rate ← dom[u].rate
    else if dom[u].tempRate ≠ 0 and dom[u].tempRate ≠ cgroup[u].rate then
        SET cgroup[u].rate ← dom[u].tempRate
```

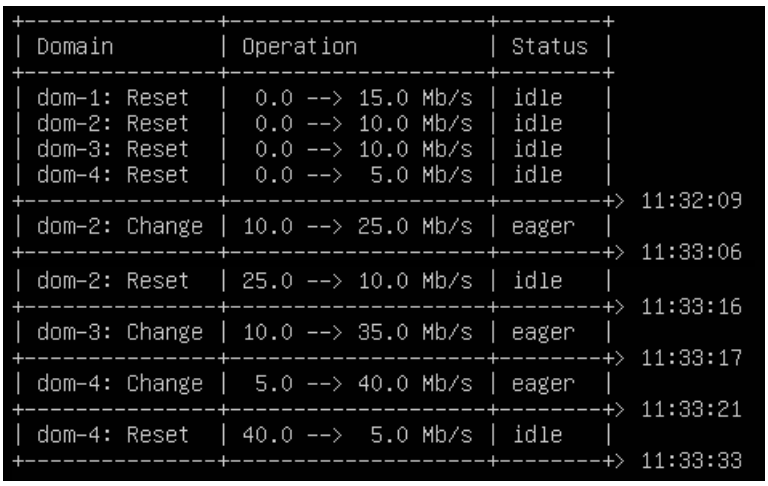
รูปที่ 22 แสดงอัลกอริทึมในการควบคุมแบนด์วิดท์ของ AIO



รูปที่ 23 แสดงลำดับขั้นตอนการทำงานของ AIO เมื่อไม่มีแบนด์วิดท์เหลือในระบบ



รูปที่ 24 แสดงลำดับขั้นตอนการทำงานของ AIO เมื่อมีแบนด์วิดท์เหลือในระบบ



รูปที่ 25 แสดง log การทำงานของระบบ AIO

3.6 ตัวชี้วัดที่ใช้ในการทดสอบ

เนื่องจากระบบควาบคุม I/O แบนด์วิดท์นี้ต้องทำการวัดประสิทธิภาพของระบบหน่วยเก็บข้อมูลซึ่งสามารถวัดผลจาก Input/output Operations Per Second และ Bandwidth โดยประเมินการเข้าถึงตำแหน่งแบบ random access และ แบบ sequential success มีรายละเอียดดัง *ตารางที่ 1*

ตัวชี้วัด	ความหมาย	การคำนวณ
Input/output Operations Per Second	จำนวนคำสั่งอ่านเขียนที่ทำได้ต่อวินาที	$\left(\frac{\text{MBps Throughput}}{\text{KB per I/O}}\right) \times 1024$
Disk bandwidth	อัตราเร็วในการเขียนอ่านที่ทำได้ต่อวินาที	$\frac{\text{IOPS} \times \text{KB per I/O}}{1024}$

ตารางที่ 1 ตัวชี้วัดของระบบควบคุม Disk I/O Bandwidth

บทที่ 4
การทดลองและผลการทดลอง

ระบบจัดแบ่งแบนด์วิดท์ของไอโอแบบปรับเปลี่ยนได้ (AIO) ทำงานเป็นโปรแกรมบนพื้นที่ของผู้ใช้งาน (user space) โดยการจับคู่ โดเมนเกสต์ กับคลาสของ cgroup โดยมีการกำหนดแบนด์วิดท์ที่เฉพาะเจาะจง และสามารถปรับเปลี่ยนได้จนถึงค่าสูงสุดที่กำหนด โดยนำแนวคิดจาก HTB มาประยุกต์ใช้ โดยระบบจะทำการเฝ้าสังเกตการณ์ใช้ทรัพยากรหน่วยเก็บข้อมูลของโดเมนเกสต์ เพื่อนำไปพิจารณาการเปลี่ยนสถานะการทำงาน แล้วยังใช้ในการวิเคราะห์ว่าโดเมนใดจะได้รับการกำหนดแบนด์วิดท์มากขึ้น จากนั้นก็ส่งคำขอไปยังชั้นเคอร์เนลผ่านอินเทอร์เฟซของ cgroup ซึ่งระบบนี้จะใช้ CFQ เป็นกลไกในการจัดตารางการใช้ทรัพยากรของดิสก์ เนื่องจากมีความเข้ากันได้ในการทำงานร่วมกับคลาสที่กำหนดจากกลุ่มควบคุม รวมถึงการจัดลำดับความสำคัญของกระบวนการที่ใช้ในงานวิจัยนี้อีกด้วย โดยมีใจความสำคัญ และวิธีการดำเนินงาน ดังต่อไปนี้

4.1 สภาพแวดล้อมที่ใช้ในการพัฒนางานวิจัย

เครื่องคอมพิวเตอร์ที่ใช้ในการพัฒนาประกอบไปด้วยโดเมนควบคุม (Dom0) จำนวน 1 โดเมน และโดเมนเกสต์ (DomU) จำนวน 4 โดเมน มีรายละเอียดดังต่อไปนี้

4.1.1 ระบบฮาร์ดแวร์

CPU:	Intel Xeon Processor E3-1220v2, 3.10 GHz, 8MB Cache
Memory:	8 GB (4x2) DDR3 1333MHz
Hard Drive:	500GB 3.5-inch 7.2K RPM SATA II

4.1.2 โดเมนควบคุม

Operating System:	Ubuntu 16.04.2 LTS
Kernel:	Linux 4.4.0-83-generic (x86_64)
Xen Version:	4.9.0
Python Version:	2.7.13

4.1.3 โดเมนเกสต์

Operating System:	Ubuntu 14.04.3 LTS
Kernel :	Linux 4.4.0-83-generic (x86_64)
Logical Volume Size:	50GB (10GB x 4)
Memory:	2048 GB (512GB x 4)
vCPU:	4 (1 x 4)

4.2 ขั้นตอนการติดตั้งระบบ

4.2.1 ติดตั้งระบบปฏิบัติการ Ubuntu 16.04.2 LTS

4.2.2 ทำการติดตั้ง และอัพเกรดแพคเกจที่จำเป็น

4.2.3 ทำการติดตั้ง และตั้งค่า Xen 4.9.0

4.2.4 ตั้งค่า Logical Volume สำหรับการติดตั้งโดเมนเกสต์

4.2.5 ติดตั้งระบบปฏิบัติการ 14.04.3 LTS ให้กับโดเมนเกสต์จำนวน 4 โดเมน

4.2.6 อัพเกรดเคอร์เนลของโดเมนเกสต์เป็นรุ่น 4.4.0-83-generic

4.2.7 ติดตั้ง FIO (Flexible I/O tester) เพื่อใช้ในการจำลอง workload

4.2.8 รัน FIO ในโหมดเซิร์ฟเวอร์บนเกสต์โดเมนทั้ง 4 โดเมน

4.3 ขั้นตอนการเตรียมความพร้อมของระบบ และเริ่มต้นการทดสอบ

4.3.1 ตรวจสอบความพร้อมใช้งานของโดเมนทั้งหมด

4.3.2 เริ่มต้น FIO บนโดเมนเกสต์ที่ต้องการทดสอบด้วยโหมด server เพื่อรอคำสั่งจากฝั่ง client ในการดำเนินการ

4.3.3 ทำการตั้งค่ากลุ่มควบคุมจาก backend-driver ของโดเมนเกสต์ ในที่นี้ได้ใช้ระบบที่จัดทำขึ้นมาเพิ่มเติมเพื่อใช้ในการบริหารจัดการ

4.3.4 เริ่มต้นระบบ AIO โดยมีการดำเนินการตามหัวข้อที่ 4.4

4.3.5 ทำการทดสอบโดยการจ่ายงานด้วย FIO ในโหมด client .ให้แก่โดเมนที่ต้องการ โดยสามารถเลือกที่จะจ่ายงานประเภทเดียวกัน หรือแตกต่างกันได้

4.4 ผลการทดลองของระบบจัดแบ่งแบนด์วิดท์ของไอโอแบบปรับเปลี่ยนได้

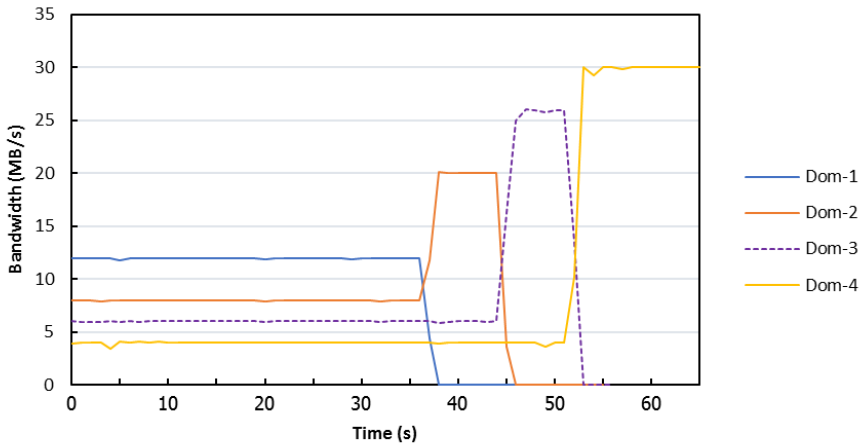
4.4.1 การทดสอบการทำงานเบื้องต้นของระบบจัดแบ่งแบนด์วิดท์

ในการทดสอบการทำงานขั้นต้น ได้ทำการทดลองโดยการกำหนดค่าแบนด์วิดท์สูงสุดไว้ที่ 30MB/s ซึ่งมีค่าต่ำกว่าค่าเฉลี่ยที่ระบบสามารถทำได้ เพื่อลดการแย่งชิงทรัพยากร และแสดงให้เก็นหลักการทำงานของระบบได้อย่างชัดเจน โดยมีการตั้งค่าตาม ตารางที่ 2

Class ID	Limit-at (MB/s)	Max-limit (MB/s)	Priority
Root	0	30	-
Dom-1	12	30	1
Dom-2	8	30	3
Dom-3	6	30	5
Dom-4	4	30	7

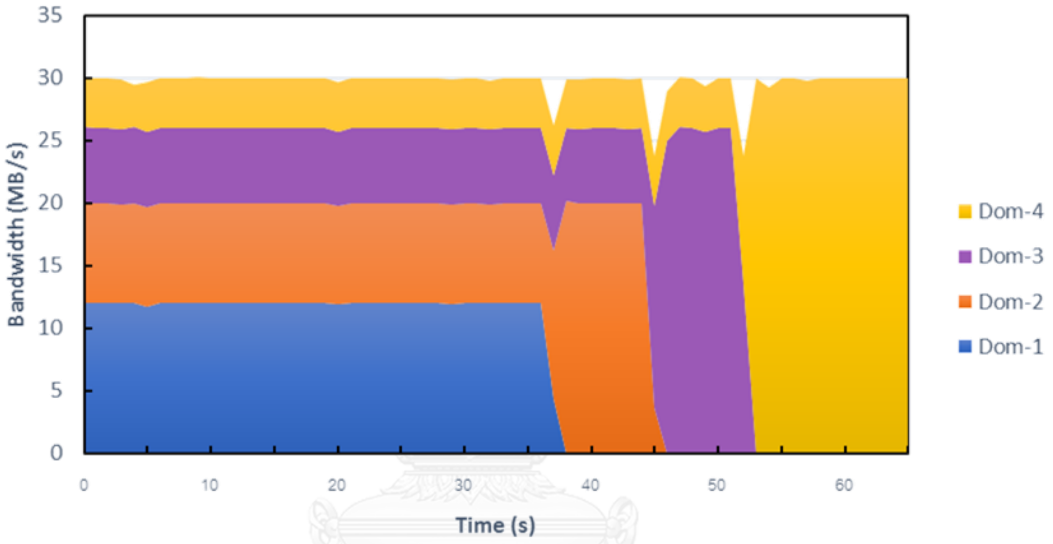
ตารางที่ 2 การตั้งค่าโดเมนสำหรับการทดสอบ การอ่านตามลำดับด้วยบล็อคขนาด 128k

จากผลการทดลองเบื้องต้นในรูปที่ 26 เป็นการทดสอบการอ่านแบบตามลำดับ (sequential-read) ด้วยขนาดของบล็อกที่ 128k โดยการจ่ายงานให้พร้อมกันทุกโดเมน เมื่อ Dom-1 ทำงานเสร็จสิ้นแบนด์วิดท์จะถูกคืนให้ระบบส่วนกลาง จากนั้นจึงมอบแบนด์วิดท์ที่เหลืออยู่ของระบบนี้ให้กับโดเมนที่มีค่าความสำคัญสูงที่สุดที่กำลังดำเนินการอยู่ในที่นี้คือ Dom-2 ทำให้โดเมนนี้มีอัตราการอ่านข้อมูลเพิ่มขึ้นเป็น 20 MB/s จากเดิมอยู่ที่ 8 Mb/s และเมื่อ Dom-2 ทำงานเสร็จสิ้นลง ก็จะส่งมอบแบนด์วิดท์ของตนคืนกลับให้ส่วนกลางเพื่อนำไปจ่ายให้กับโดเมนที่มีสิทธิใช้งานต่อไป โดยแสดง



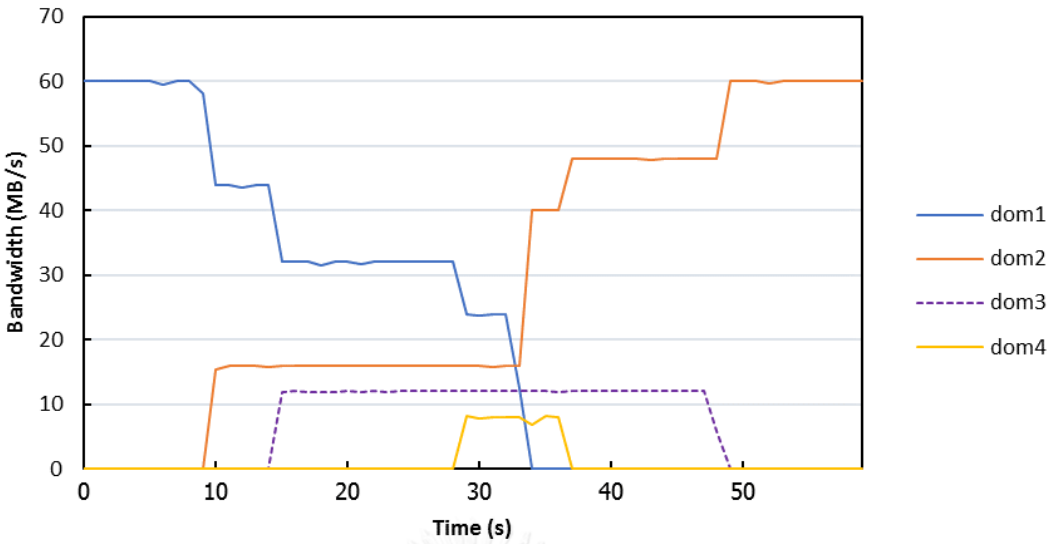
รูปที่ 26 การทดสอบการอ่านไฟล์ตามลำดับด้วยบล็อกขนาด 128k พร้อมกัน 4 โดเมน

การใช้งานแบนด์วิดท์โดยรวม ถูกแสดงในกราฟการซ้อนทับของรูปที่ 27 ซึ่งได้แสดงให้เห็นว่าระบบมีการใช้งานแบนด์วิดท์โดยรวมเป็นไปตามที่กำหนดซึ่งอยู่ในช่วง 30MB/s แต่จะมีการลดลงเพียงเล็กน้อยในช่วงที่โดเมนใดโดเมนหนึ่งหยุดการทำงาน เนื่องจากระบบมีการหน่วงเวลาในการจ่ายแบนด์วิดท์จากส่วนกลางให้กับโดเมนที่มีสิทธิได้รับเพิ่มเพื่อป้องกันการเพิ่มขึ้นเกินขีดจำกัดของระบบ ซึ่งอาจส่งผลให้โดเมนอื่นๆ ต้องลดแบนด์วิดท์ของตัวเองลงเพื่อให้ระบบสมารถทำงานต่อไปได้ และป้องกันการคำรวณที่ผิดพลาดซึ่งอาจส่งผลเสียให้แก่ระบบโดยรวมได้เช่นกัน



รูปที่ 27 กราฟซ้อนทับของการอ่านไฟล์ตามลำดับด้วยบล็อกขนาด 128k พร้อมกัน 4 โดเมน

ต่อมาได้ทำการทดลองโดยการเปลี่ยนการตั้งค่าให้กับแต่ละโดเมนที่ 1-4 เป็น 25, 15, 12 และ 8MB/s ตามลำดับแสดงในรูปที่ 28 โดยมีแบนด์วิดท์รวมอยู่ที่ 60Mb/s แล้วทำการทดสอบโดยการจ่ายงานในการอ่านให้ในแต่ละโดเมนโดยไม่พร้อมเพรียงกัน เริ่มต้นโดยให้ Dom-1 ทำงานเพียงโดเมนเดียวทำให้สามารถใช้ Bandwidth ของระบบได้ทั้งหมดที่ 60MB/s จากนั้นในวินาทีที่ 9 มีการจ่ายงานให้กับ Dom-2 ทำให้ Dom-1 ต้องคืนอันตราการอ่านให้กับ Dom-2 ที่ 15MB/s และเมื่อมีโดเมนอื่นเริ่มทำงานโดนเมนที่ได้รับสิทธิก็จะคืนแบนด์วิดท์ที่ตัวเองดึงมาใช้นั้นกลับสู่งส่วนกลางเพื่อนำไปจ่ายให้โดเมนที่เริ่มทำงาน ทำให้สามารถดำเนินการในอัตราที่ถูกกำหนดไว้ในการตั้งค่าได้ ซึ่งเป็นการรักษาเสถียรภาพของระบบอีกด้วย



รูปที่ 28 การทดสอบการอ่านไฟล์โดยกำหนดงานให้แต่ละโดเมนไม่พร้อมกัน

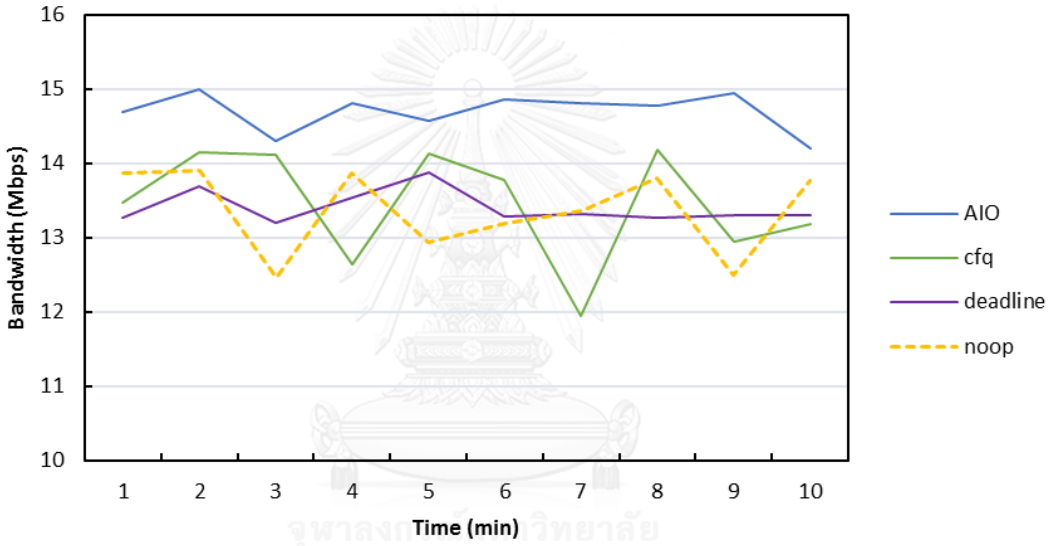
4.4.2 การทดสอบการอ่านตามลำดับอย่างต่อเนื่องด้วยบล็อคขนาด 512B

การทดลองนี้ได้ทำการสดสอบการอ่านด้วยบล็อกขนาดเล็กมาก โดยเลือกใช้ขนาด 512B เพื่อวัดว่าระบบสามารถทนต่อสภาวะ ที่มีการใช้ไอโอในปริมาณมากได้หรือไม่โดยมีการตั้งค่าโดเมนตามตารางที่ 3

Class ID	Limit-at (Mbps)	Max-limit (Mbps)	Priority
Root	0	15	-
Dom-1	6	15	1
Dom-2	4	15	3
Dom-3	3	15	5
Dom-4	2	15	7

ตารางที่ 3 การตั้งค่าโดเมนสำหรับการทดสอบ การอ่านตามลำดับด้วยบล็อคขนาด 512B

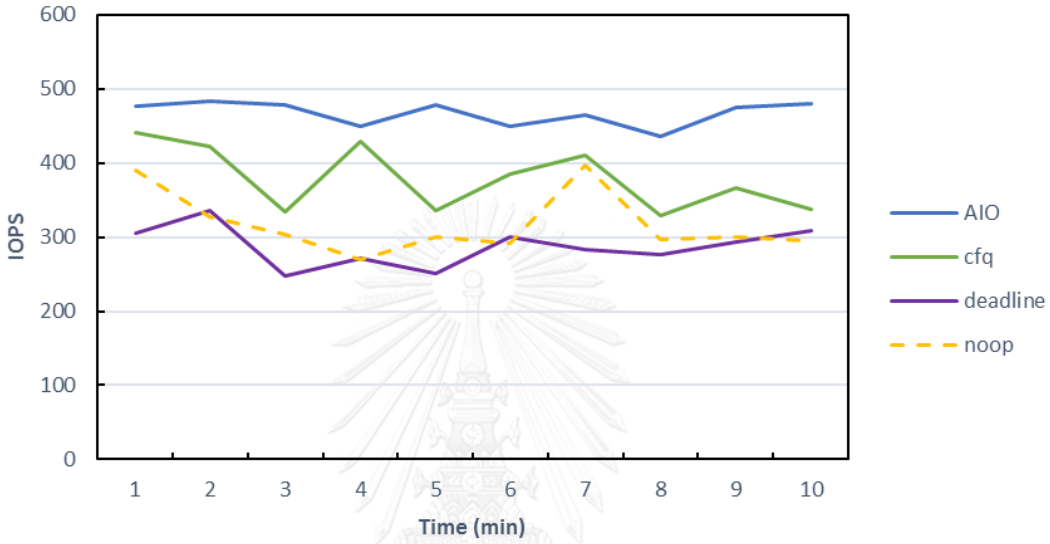
ในการทดลองนี้ได้ทำการเปรียบเทียบการจัดแบ่งแบนด์วิดท์ของ AIO เทียบกับ noop, deadline และ cfq ซึ่งเป็นอัลกอริทึมในการจัดตารางการใช้งานหน่วยเก็บข้อมูลของลินุกซ์ในเคอร์เนลปัจจุบัน (รุ่น 4.4.0 ณ เวลาที่ผู้เขียนทำงานวิจัย) ผลการทดลองแสดงดังรูปที่ 29 โดยผลลัพธ์แสดงให้เห็นว่าเมื่อมีการใช้ไอโอที่มีขนาดเล็กมากอย่าง 512B ทำให้ระบบถูกลดแบนด์วิดท์ลงอย่างเห็นได้ชัดจาก cfq, noop และ deadline ที่ไม่ได้มีการปรับแต่งอะไรเพิ่มเติม ซึ่งระบบของเรายังคงพยายามรักษาระดับแบนด์วิดท์ที่ได้กำหนดเอาไว้ถึงแม้ว่าจะไม่สามารถรักษาระดับไว้ที่ 15 Mbps แต่ยังคงค่าเฉลี่ยไว้ที่ 14.69Mbps ส่วน cfq, noop และ deadline มีค่าเฉลี่ยอยู่ที่ 13.45, 13.40 และ 13.36M ตามลำดับ แสดงให้เห็นว่าระบบสามารถรักษาเสถียรภาพได้ดีระดับหนึ่งเลยทีเดียว



รูปที่ 29 การเปรียบเทียบประสิทธิภาพในการอ่านตามลำดับต่อเนื่องด้วยบล็อคขนาด 512B

4.4.3 การทดสอบการอ่านตามลำดับอย่างต่อเนื่องด้วยบล็อคขนาด 64k

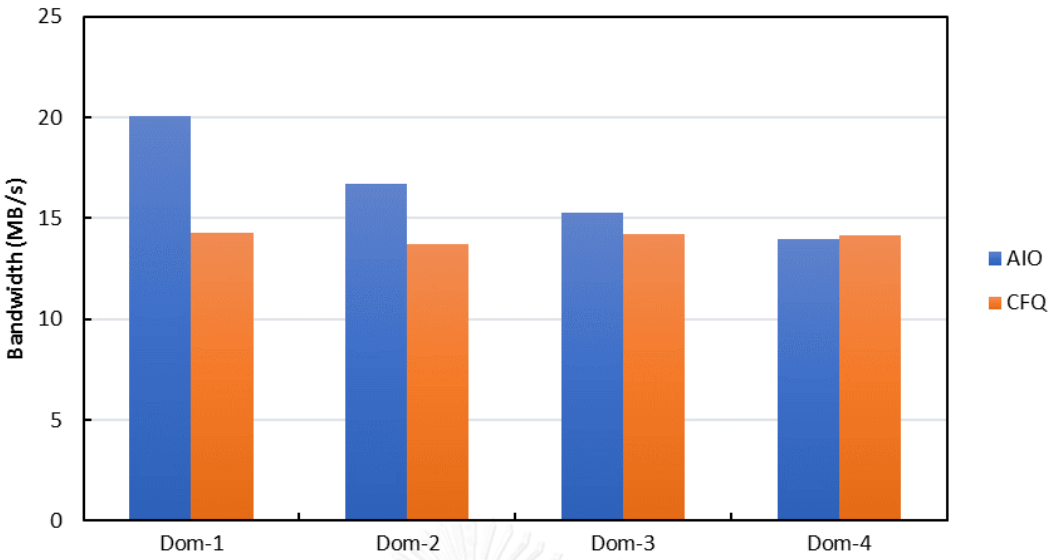
การทดลองต่อมาเป็นการทดสอบการอ่านตามลำดับด้วยบล็อกขนาดเล็กที่ 64k เพื่อทำการวัดค่า การดำเนินการไอโอต่อวินาที (IOPS) ในรูปที่ 30 โดยใช้การตั้งค่าตามตารางที่ 2 ผลลัพธ์แสดงให้เห็นว่าระบบ AIO ยังคงรักษาเสถียรภาพไว้ได้ดีระดับหนึ่งเมื่อเทียบกับอัลกอริทึมอื่น ที่มีการแกว่งของค่า IOPS อยู่ในช่วง 248 ถึง 440 IOPS



รูปที่ 30 การเปรียบเทียบประสิทธิภาพในการอ่านตามลำดับด้วยบล็อคขนาด 64k อย่างต่อเนื่อง

4.4.4 การเปรียบเทียบแบนด์วิดท์เฉลี่ยในแต่ละโดเมนเทียบกับ CFQ

สำหรับการทดลองนี้ดำเนินการโดยการเทียบค่าแบนด์วิดท์ในการทำงานโดยเฉลี่ยในแต่ละโดเมนระหว่าง AIO เทียบกับ cfq พบว่า ระบบจัดแบ่งแบนด์วิดท์ AIO มีค่าเฉลี่ยสูงกว่า cfq ในสามโดเมนแรก (Dom-1, Dom-2 และ Dom-3) แต่ในโดเมนที่ 4 มีค่าเฉลี่ยที่น้อยกว่า cfq เนื่องจาก Dom-4 เป็นโดเมนที่ถูกจัดลำดับความสำคัญน้อยที่สุด ทำให้ได้รับแบนด์วิดท์จากส่วนกลางหลังจากที่โดเมนอื่นที่มีการจัดลำดับความสำคัญสูงกว่าดำเนินการเสร็จสิ้นแล้วเท่านั้น จึงเป็นผลให้โดเมนที่ 4 มีค่าเฉลี่ยของแบนด์วิดท์ที่ต่ำกว่า cfq เพียงเล็กน้อย แสดงดังรูปที่ 31



รูปที่ 31 การเปรียบเทียบแบนด์วิดท์เฉลี่ยในแต่ละโดเมนระหว่า AIO เทียบกับ CFQ

4.4.5 การทดสอบการเขียนแบบตามลำดับด้วยบล็อคขนาด 512K

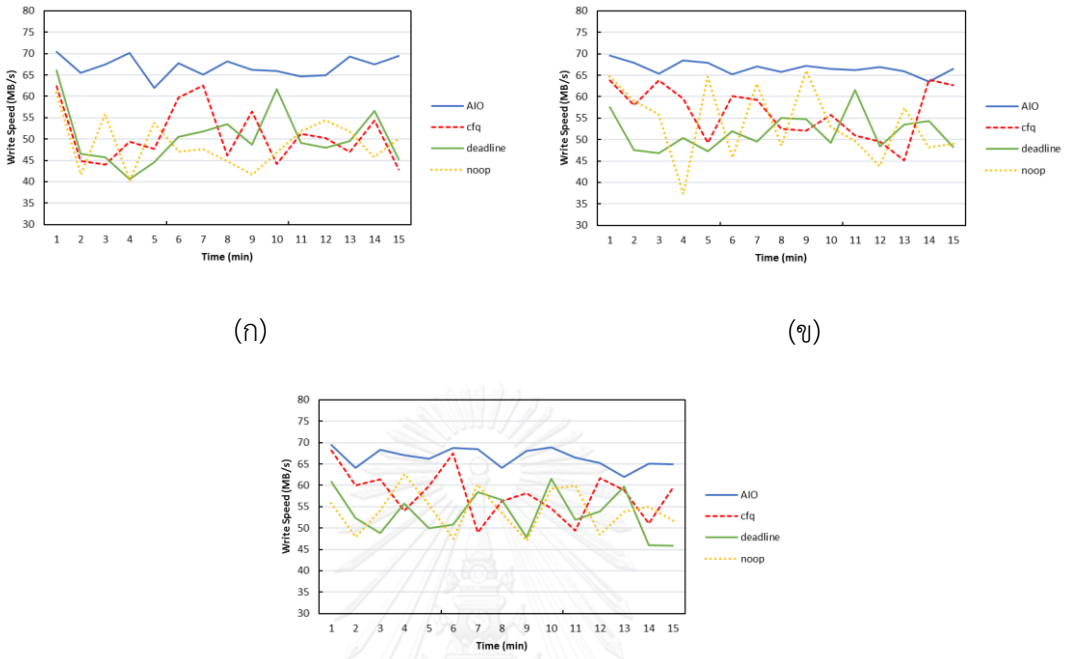
ในการทดลองนี้เป็นการสอบการเขียนไฟล์ขนาดใหญ่ด้วยบล็อกขนาด 512K เพื่อวัดประสิทธิภาพในการจัดการกับข้อมูลขนาดใหญ่ โดยทำการเขียนไฟล์อย่างต่อเนื่อง เพื่อวัดความเสถียรในการบริหารจัดการข้อมูลด้วย ซึ่งมีการตั้งค่าตาม ตารางที่ 4

Class ID	Limit-at (MB/s)	Max-limit (MB/s)	Priority
Root	0	70	-
Dom-1	30	70	1
Dom-2	15	70	3
Dom-3	15	70	5
Dom-4	10	70	7

ตารางที่ 4 การตั้งค่าโดเมนสำหรับการทดสอบอ่าน-เขียนตามลำดับด้วยบล็อคขนาด 512K

ผลการทดลองถูกแสดงรูปที่ 32 ขนาดของข้อมูลที่ใช้ทดสอบมี 3 ขนาดคือ 100MB ในรูปที่ 31(ก), 512MB ในรูปที่ 31(ข) และ 1GB ในรูปที่ 31(ค) โดยได้ทำการเปรียบเทียบกับ cfq, deadline และ noop ซึ่งผลการทดลองค่อนข้างมีความคล้ายคลึงกัน ทั้งนี้พบว่า AIO พยายามที่จะรักษาระดับแบนด์วิดท์เอาไว้ ถึงแม้ว่าจะมีการแย่งชิงทรัพยากรที่สูง จากโดเมนทั้ง 4 แต่เมื่อเทียบกับอัลกอริทึม

อื่นแล้ว AIO ยังคงมีความเสถียรในการจัดการข้อมูลที่มากกว่า ถึงแม้จะมีการแกว่งของแบนด์วิดท์อยู่บ้างแต่นับว่าเป็นผลที่ค่อนข้างน่าพอใจเลยทีเดียว



(ก) (ข)

(ค)

รูปที่ 32 การเปรียบเทียบประสิทธิภาพในการเขียนอย่างต่อเนื่องของระบบ AIO ด้วยขนาดของบล็อก 512k ใช้ไฟล์ ขนาด (ก) 100MB (ข) 500MB และ (ค) 1GB ในการทดสอบ

บทที่ 5
สรุปผลการวิจัย

5.1 สรุปผลการวิจัย

การนำเสนอการจัดแบ่งแบนด์วิดท์ของไอโอแบบปรับเปลี่ยนสำหรับสภาวะแบบเสมือน เป็นการนำเสนอแนวทางในการแก้ไขปัญหาที่เกิดจากการแย่งชิงทรัพยากรหน่วยเก็บข้อมูลในระบบแบบเสมือน ซึ่งการใช้อัลกอริทึมในการบริหารการเข้าถึงข้อมูลของลินุกซ์เพียงอย่างเดียวนั้น ยังไม่มีเสถียรภาพเพียงพอในการรองรับการเข้าถึงข้อมูลพร้อมกันจากหลายโดเมน ทำให้การแย่งชิงทรัพยากรนี้ส่งผลกระทบไปถึงระบบโดยรวม ซึ่งมีแนวโน้มที่จะทำให้ระบบมีประสิทธิภาพในการจัดการข้อมูลลดลงเป็นอย่างมาก ด้วยเหตุนี้ผู้วิจัยจึงได้นำเสนอกระบวนการในการจัดแบ่งแบนด์วิดท์สำหรับหน่วยเก็บข้อมูลขึ้น ซึ่งมีใจความสำคัญ และวิธีการดำเนินงานดังต่อไปนี้

ระบบจัดแบ่งแบนด์วิดท์นี้ถูกออกแบบขึ้นบนพื้นฐานที่ว่าต้องทำการปรับปรุงและพัฒนาระบบได้โดยไม่ต้องแก้ไขเคอร์เนล หรือปรับเปลี่ยนระบบนอกเหนือพื้นที่ของผู้ใช้งาน เพื่อให้การปรับปรุงแก้ไขสามารถทำได้โดยไม่ต้องอาศัยความชำนาญการในด้านระบบปฏิบัติการมากนัก โดยระบบนี้ได้นำกลุ่มควบคุมของลินุกซ์มาปรับใช้เพื่อทำการควบคุมระดับแบนด์วิดท์ของการเขียน และอ่านข้อมูลจากโดเมนเสมือน พร้อมทั้งนำหลักการของถังโทเคนแบบลำดับขั้นมาประยุกต์ใช้เพื่อให้การควบคุมแบนด์วิดท์สามารถปรับเปลี่ยนได้ตามลำดับความสำคัญของแต่ละโดเมน ซึ่งระบบได้มีการเฝ้าสังเกตการเข้าถึงข้อมูลของแต่ละโดเมนเพื่อนำมาวิเคราะห์ในการกำหนดปรับเปลี่ยนอัตราการเข้าถึงข้อมูล ให้เหมาะสมตามสภาวะการใช้งาน อีกทั้งยังช่วยลดการแย่งชิงทรัพยากรของระบบเนื่องจากมีการกำหนดอัตราสูงสุดที่ระบบสามารถดำเนินการได้ แต่ยังสามารถปรับเปลี่ยนในแต่ละโดเมนเพื่อให้เข้ากับสถานการณ์ได้

จากการทดลอง และวิเคราะห์ผลการทำงานของระบบ แสดงให้เห็นว่าระบบจัดแบ่งแบนด์วิดท์มีเสถียรภาพในการทำงานที่ดีอยู่ระดับหนึ่ง เมื่อเทียบกับการใช้อัลกอริทึมสำหรับจัดตารางการใช้งานดิสก์ในระบบลินุกซ์เพียงอย่างเดียว แล้วยังสามารถจัดการแบนด์วิดท์เมื่อมีการใช้งานทรัพยากรหน่วยเก็บข้อมูลจากหลายโดเมนในเวลาเดียวกันได้ นอกจากนี้ระบบดังกล่าวยังทำงานบนพื้นที่ของผู้ใช้งานจึงสามารถนำไปพัฒนาต่อยอดกับระบบที่มีลักษณะใกล้เคียงกันได้อีกด้วย

5.2 ข้อจำกัดของงานวิจัย

5.2.1 ผู้ใช้งานจะต้องกำหนดค่าแบนด์วิดท์ควบคุม แบนด์วิดท์สูงสุด และลำดับความสำคัญของแต่ละคลาส โดยระบบจะบันทึกการตั้งค่าไว้ในไฟล์การตั้งค่า

5.2.2 ถ้าผู้ใช้งานต้องการเปลี่ยนการตั้งค่าของระบบ ต้องทำการเริ่มโปรแกรมใหม่โดยใช้โหมดตั้งค่า เพื่อบันทึกข้อมูลใหม่

5.2.3 ในกรณีของโดเมนที่มีสถานะการทำงานที่ปกติ (active) เมื่อไม่ได้ใช้งานแบนด์วิดท์ถึงระดับที่ถูกกำหนดไว้จะไม่สามารถคืนแบนด์วิดท์ที่เหลือให้กับส่วนกลางได้ เนื่องจากถ้าโดเมนนั้นลดความต้องการลงแค่ชั่วครู่ ระบบจะไม่สามารถพิจารณาได้ว่า โดเมนนั้นจะมีความต้องการใช้งานมากกว่าจุดนั้นอีกเมื่อไร จนว่าสถานะของโดเมนนั้นจะถูกเปลี่ยนเป็นว่างงาน (idle) เพื่อกำหนดค่า rate ให้อีกครั้ง

5.2.4 ระบบไม่สามารถรองรับงานแบบผสมระหว่าง random และ sequential พร้อมกันได้ เนื่องจากไม่มี policy ในการปรับระบบควบคุมโดยเจาะจงประเภทของ workload

5.2.5 เมื่อผู้ใช้ไม่ต้องการใช้งานระบบควบคุมแบนด์วิดท์จะต้องตั้งค่า แบนด์วิดท์ควบคุม และแบนด์วิดท์สูงสุด ให้เป็น 0 หรือทำการลบ control group ที่เกี่ยวข้องออก เพื่อให้สามารถใช้งานระบบแบบดั้งเดิมได้

5.2.6 ไม่ได้ถูกตั้งให้เป็นค่าเริ่มต้นของระบบ โดยเมื่อทำการ reboot เครื่องทางกายภาพจะต้องทำการเริ่มโปรแกรมขึ้นมาอีกครั้ง โดยสามารถใช้ไฟล์การตั้งค่าที่ถูกบันทึกเอาไว้

5.3 งานวิจัยในอนาคต

เนื่องจากระบบนี้ดำเนินการอยู่บน user space จึงสามารถนำไปพัฒนา และประยุกต์ใช้กับระบบควบคุม หรืออัลกอริทึมอื่นๆ ที่อยู่ในระดับชั้นลึกลงไปได้ เพื่อเพิ่มประสิทธิภาพในการประมวลผลข้อมูลของระบบ นอกจากระบบนี้จะสามารถทำงานร่วมกับ xen hypervisor ได้แล้ว ยังสามารถนำไปปรับปรุงเพื่อใช้กับฐานของระบบที่เป็นลินุกซ์ได้อีกด้วย เช่น KVM และ Docker ทั้งนี้การพัฒนาระบบให้สามารถปรับตัวกับ workload ได้หลากหลายประเภท ยังคงมีความสำคัญเป็นอย่างยิ่ง เนื่องจากในการใช้งานจริงยังคงมีงานมากมายหลายประเภท ไม่ว่าจะเป็นการประมวลผลข้อมูลขนาดมหาศาล หรือแม้กระทั่งข้อมูลขนาดเล็กที่มีปริมาณมาก ซึ่งการประมวลผลข้อมูลเหล่านี้ล้วนเป็นตัวการลดประสิทธิภาพของระบบลงอย่างมากเลยทีเดียว

รายการอ้างอิง

1. Li, Y., W. Li, and C. Jiang. *A survey of virtual machine system: Current technology and future trends*. in *Electronic Commerce and Security (ISECS),* 2010 *Third International Symposium on*. 2010. IEEE.
2. Wu, J.C., S. Banachowski, and S.A. Brandt. *Hierarchical disk sharing for multimedia systems*. in *Proceedings of the international workshop on Network and operating systems support for digital audio and video*. 2005. ACM.
3. Kim, S., D. Kang, and J. Choi, *I/O Characteristics and Implications of Big Data Processing on Virtualized Environments.* Appl. Math, 2015. 9(2L): p. 591-598.
4. Shafer, J. *I/O virtualization bottlenecks in cloud computing today*. in *Proceedings of the 2nd conference on I/O virtualization*. 2010. USENIX Association.
5. Ongaro, D., A.L. Cox, and S. Rixner. *Scheduling I/O in virtual machine monitors*. in *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. 2008. ACM.
6. Kesavan, M., A. Gavrilovska, and K. Schwan. *On disk I/O scheduling in virtual machines*. in *Proceedings of the 2nd conference on I/O virtualization*. 2010. USENIX Association.
7. Zhang, B., et al. *A survey on i/o virtualization and optimization*. in *ChinaGrid Conference (ChinaGrid),* 2010 *Fifth Annual*. 2010. IEEE.
8. Seelam, S.R. and P.J. Teller. *Virtual I/O scheduler: a scheduler of schedulers for performance virtualization*. in *Proceedings of the 3rd international conference on Virtual execution environments*. 2007. ACM.
9. Ling, X., et al. *Efficient disk I/O scheduling with qos guarantee for xen-based hosting platforms*. in *Proceedings of the* 2012 12*th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid* 2012). 2012. IEEE Computer Society.

10. Malensek, M., S.L. Pallickara, and S. Pallickara. *Alleviation of disk I/O contention in virtualized settings for data-intensive computing*. in *Big Data Computing (BDC),* 2015 *IEEE/ACM* 2*nd International Symposium on*. 2015. IEEE.
11. Wu, S., et al., *iShare: Balancing I/O performance isolation and disk I/O efficiency in virtualized environments.* Concurrency and Computation: Practice and Experience, 2016. 28(2): p. 386-399.
12. Tan, H., et al., *VMCD: A Virtual Multi-Channel Disk I/O Scheduling Method for Virtual Machines.* IEEE Transactions on Services Computing, 2016. 9(6): p. 982-995.
13. Kang, D.-J., et al. *Proportional disk I/O bandwidth management for server virtualization environment*. in *Computer Science and Information Technology,* 2008. *ICCSIT'*08. *International Conference on*. 2008. IEEE.
14. Kang, D.-J., et al. *Range-bw: I/o scheduler for predicable disk i/o bandwidth*. in *Computer Engineering and Applications (ICCEA),* 2010 *Second International Conference on*. 2010. IEEE.
15. Wu, Y., B. Jia, and Z. Qi. *IO QoS: A New Disk I/O Scheduler Module with QoS Guarantee for Cloud Platform*. in *Information Science and Engineering (ISISE),* 2012 *International Symposium on*. 2012. IEEE.
16. Gulati, A., et al., *Efficient and adaptive proportional share I/O scheduling.* ACM SIGMETRICS Performance Evaluation Review, 2009. 37(2): p. 79-80.
17. Xiaojing, W., et al., *Weighted Fairness Resource Allocation of Disks in XEN.* International Journal on Cloud Computing: Services & Architecture, 2012.
18. Solomon, B., et al. *Leaky bucket model for autonomic control of distributed, collaborative systems*. in *Applied Computational Intelligence and Informatics (SACI),* 2013 *IEEE* 8*th International Symposium on*. 2013. IEEE.
19. Moraes, H.B. and P.R. Guardieiro. *Traffic policing mechanism based on the token bucket method for WiMax networks*. in *High Performance Computing and Communication &* 2012 *IEEE* 9*th International Conference on Embedded Software and Systems (HPCC-ICESS),* 2012 *IEEE* 14*th International Conference on*. 2012. IEEE.

20. Boutcher, D. and A. Chandra, *Does virtualization make disk scheduling passé?* ACM SIGOPS Operating Systems Review, 2010. 44(1): p. 20-24.

21. Lee, C.-H. and Y.-T. Kim. *QoS-aware hierarchical token bucket (QHTB) queuing disciplines for QoS-guaranteed Diffserv provisioning with optimized bandwidth utilization and priority-based preemption*. in *Information Networking (ICOIN),* 2013 *International Conference on*. 2013. IEEE.

ภาคผนวก

ประวัติผู้เขียนวิทยานิพนธ์

นายภาสกร สุขีพจน์ เกิดเมื่อวันที่ 3 กรกฎาคม พ.ศ. 2534 ที่จังหวัดกรุงเทพมหานคร สำเร็จการศึกษาระดับปริญญาบัณฑิต สาขาวิศวกรรมแมคคาทรอนิกส์ ภาควิชาวิศวกรรมการวัดและควบคุม คณะวิศวกรรมศาสตร์ จากสถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง และได้ศึกษาต่อในระดับมหาบัณฑิต สาขาวิทยาศาสตร์คอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย