บทที่ 2

ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

เวอร์ชวลไลเซชัน (virtualization) สามารถเกิดขึ้นได้ที่หลายระดับ แต่เทคโนโลยีเวอร์ชวลไลเซชันระดับซิสเต็มส์ (system virtualization) ที่สำคัญคือเกสโอเอส (guest operating system) หรือเซิร์ฟเวอร์เวอร์ชวลไลเซชัน เกสโอเอสเวอร์ชวลไลเซชันคือซอฟต์แวร์เลเยอร์ที่มีความสามารถที่จะติดต่อกับทรัพยากรจริงของเครื่อง เพื่อทำให้ทรัพยากรเหล่านั้นสามารถถูกใช้งานโดยเวอร์ชวลแมชชีนต่างๆได้ในเวลาเดียวกัน เทคโนโลยีเวอร์ชวลไลเซชันของเกสโอเอสมี 2 ประเภท ประเภทแรกคืออีมูเลเตอร์ (emulator) ซึ่งเป็นซอฟต์แวร์เลเยอร์ที่ใช้เพื่อจำลองเครื่องจริงขึ้นมาบนระบบปฏิบัติการเดิมที่รันอยู่บนฮาร์ดแวร์โฮส ประเภทที่สองคือไฮเปอร์ไวเซอร์ (hypervisor) ซึ่งเป็นซอฟต์แวร์ที่รันอยู่บนฮาร์ดแวร์โดยตรง (Ruest & Ruest, 2009, p. xix) ด้วยเทคโนโลยีนี้ ทำให้เราสามารถจัดเก็บสถานะการทำงานของเวอร์ชวลแมชชีนลงสู่ไฟล์และรีเซ็ทสภาพแวดล้อมเหล่านั้นให้กลับมาสู่สภาพเดิมภายหลังได้โดยง่าย นอกจากนี้เวอร์ชวลไลเซชันยังมีประโยชน์อื่นๆอีกหลายอย่างรวมทั้งช่วยแก้ปัญหาส่วนใหญ่เกี่ยวกับการจัดการแอพพลิเคชันได้อีกด้วย

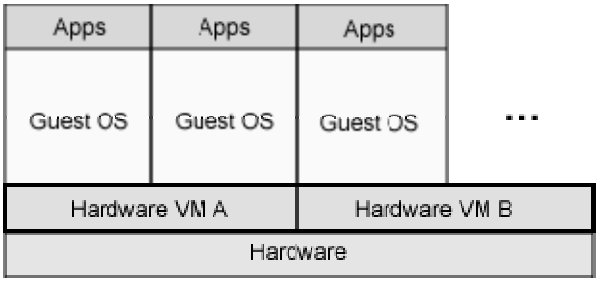
ประเภทของเวอร์ชวลไลเซชัน

วิธีการทำเวอร์ชวลไลเซชันมีหลายวิธีแตกต่างกันไป โดยทั่วไปวิธีการทำเวอร์ชวลไลเซชันในลินุกซ์มี 3 วิธี คือ ฮาร์ดแวร์อีมูเลชัน (hardware emulation) ฟูลเวอร์ชวลไลเซชัน (full virtualization) และพาราเวอร์ชวลไลเซชัน (paravirtualization) (Jones, 2006)

1. ฮาร์ดแวร์อีมูเลชัน (Hardware emulation)

ในวิธีการนี้ ฮาร์ดแวร์เวอร์ชวลแมชชีนถูกสร้างขึ้นบนระบบโฮสเพื่อจำลองฮาร์ดแวร์ที่ต้องการ ดังที่แสดงในภาพที่ 2.1

ภาพที่ 2.1
ฮาร์ดแวร์อีมูเลชัน



ที่มา: “Virtual Linux,” โดย Jones, M. T., 2006, from http://www.ibm.com/developerworks/library/l-linuxvirt/index.html.

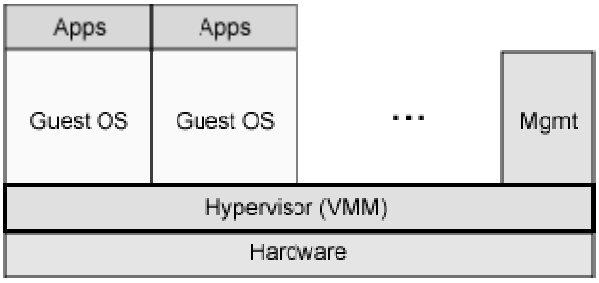
การใช้ฮาร์ดแวร์อีมูเลชันที่สำคัญอย่างหนึ่งคือการใช้เพื่อพัฒนาเฟิร์มแวร์และฮาร์ดแวร์ โดยแทนที่จะรอให้มีฮาร์ดแวร์จริง ผู้พัฒนาเฟิร์มแวร์สามารถใช้ฮาร์ดแวร์เวอร์ชวลแมชชีนเป้าหมายในการทำให้โค้ดจริงสามารถใช้งานได้ในระบบการจำลอง ปัญหาหลักของฮาร์ดแวร์อีมูเลชันคือมันอาจจะช้ามาก เนื่องจากทุกคำสั่งต้องถูกจำลองบนฮาร์ดแวร์จริง ซึ่งอาจทำให้เครื่องช้าลงถึง 100 เท่าได้ สำหรับการจำลองระดับสูงที่รวมทั้งความถูกต้องของไซเคิล ซีพียูไปป์ไลน์ที่ถูกจำลอง และการแคชการกระทำต่างๆ อาจจะทำให้เครื่องช้าลงถึง 1000 เท่าได้ (Jones, 2006)

ข้อดีของฮาร์ดแวร์อีมูเลชันคือสามารถใช้รันระบบปฏิบัติการสำหรับเครื่อง PowerPC บนโฮสที่มีโปรเซสเซอร์ ARM ได้ และยังสามารถรันเวอร์ชวลแมชชีนได้หลายเครื่องโดยแต่ละเครื่องจำลองโปรเซสเซอร์ที่แตกต่างกัน

2. **ฟูลเวอร์ชวลไลเซชัน** (Full virtualization)

ฟูลเวอร์ชวลไลเซชัน หรือ เนทีฟเวอร์ชวลไลเซชัน (native virtualization) คือวิธีการทำเวอร์ชวลไลเซชันที่ใช้เวอร์ชวลแมชชีนเป็นสื่อกลางระหว่างเกสโอเอสและเนทีฟฮาร์ดแวร์ ในภาพที่ 2.2 VMM เป็นสื่อกลางระหว่างเกสโอเอสและฮาร์ดแวร์จริง โดยคำสั่งที่ถูกป้องกันบางคำสั่งต้องถูกดักและจัดการภายในไฮเปอร์ไวเซอร์ (hypervisor) เนื่องจากฮาร์ดแวร์จริงไม่ได้เป็นของระบบปฏิบัติการ แต่จะถูกแบ่งให้ใช้งานโดยระบบปฏิบัติการผ่านทางไฮเปอร์ไวเซอร์

ภาพที่ 2.2
ฟูลเวอร์ชวลไลเซชัน



ที่มา: “Virtual Linux,” โดย Jones, M. T., 2006, from http://www.ibm.com/developerworks/library/l-linuxvirt/index.html.

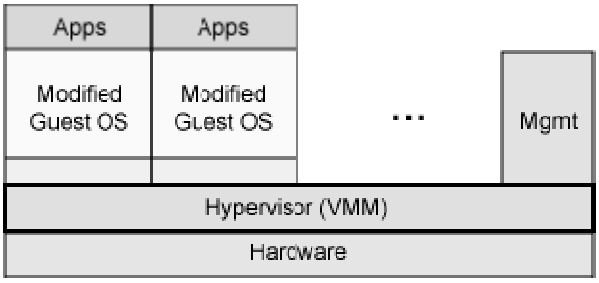
ฟูลเวอร์ชวลไลเซชันเร็วกว่าฮาร์ดแวร์อีมูเลชัน แต่ประสิทธิภาพน้อยกว่าการทำงานบนฮาร์ดแวร์จริงเพราะมีตัวกลางคือไฮเปอร์ไวเซอร์ ข้อดีที่สำคัญของฟูลเวอร์ชวลไลเซชันคือระบบปฏิบัติการสามารถรันได้โดยไม่ต้องถูกแก้ไข โดยมีเงื่อนไขเพียงอย่างเดียวคือระบบปฏิบัติการต้องใช้ชุดคำสั่งที่รันบนฮาร์ดแวร์จริง

3. **พาราเวอร์ชวลไลเซชัน** (Paravirtualization)

วิธีการนี้ใช้ไฮเปอร์ไวเซอร์สำหรับการใช้งานฮาร์ดแวร์จริงร่วมกัน แต่ต้องมีการแก้ไขเปลี่ยนแปลงระบบปฏิบัติการเกสโดยแทรกโค้ดเพื่อจัดการการทำเวอร์ชวลไลเซชันเข้าไปในระบบปฏิบัติการเกส ดังภาพที่ 2.3 วิธีการนี้ทำให้ไม่จำเป็นต้องทำการคอมไพล์โค้ดในขณะที่เกสทำงานหรือการดักจับคำสั่งใดๆเหมือนแบบฟูลเวอร์ชวลไลเซชัน เนื่องจากโค้ดสำหรับจัดการเวอร์ชวลไลเซชันได้รับการเพิ่มเติมลงในโค้ดของระบบปฏิบัติการเกสแล้ว

ข้อเสียของพาราเวอร์ชวลไลเซชันคือเกสโอเอสจำเป็นต้องถูกแก้ไขสำหรับไฮเปอร์ไวเซอร์ แต่มีประสิทธิภาพใกล้เคียงกับระบบจริง ทั้งฟูลเวอร์ชวลไลเซชันและพาราเวอร์ชวลไลเซชันสามารถสนับสนุนเวอร์ชวลแมชชีนหลายเครื่องที่มีระบบปฏิบัติการเกสแตกต่างกันได้ในเวลาเดียวกัน

ภาพที่ 2.3
พาราเวอร์ชวลไลเซชัน



ที่มา: “Virtual Linux,” โดย Jones, M. T., 2006, from http://www.ibm.com/developerworks/library/l-linuxvirt/index.html.

เวอร์ชวลแมชชีน (Virtual Machine)

ซิสเต็มส์เวอร์ชวลแมชชีนช่วยให้ฮาร์ดแวร์แพลตฟอร์มของโฮสเครื่องหนึ่งสามารถสนับสนุนเกสโอเอสหลายระบบได้ในเวลาเดียวกัน ด้วยเทคโนโลยีเวอร์ชวลแมชชีน ผู้ใช้สามารถรันระบบปฏิบัติการที่แตกต่างกันได้บนฮาร์ดแวร์เดียวกัน ความสามารถที่สำคัญของเทคโนโลยีซิสเต็มส์เวอร์ชวลแมชชีนคือการแยกออกจากกัน (isolation) ของระบบต่างๆที่รันอยู่ในเวลาเดียวกันบนฮาร์ดแวร์แพลตฟอร์มเดียวกัน นั่นคือถ้าเกสโอเอสระบบหนึ่งเกิดความผิดพลาดขึ้น ซอฟต์แวร์ที่รันอยู่บนเกสระบบอื่นจะไม่ได้รับผลกระทบไปด้วย โดยหลักในซิสเต็มส์เวอร์ชวลแมชชีน VMM จะแบ่งทรัพยากรฮาร์ดแวร์ระหว่างเกสโอเอสต่างๆ เช่น ดิสก์เวอร์ชวลไลเซชัน โดย VMM จะมีการใช้งานและจัดการทรัพยากรฮาร์ดแวร์ทั้งหมด เกสโอเอสและแอพพลิเคชันโปรเซสของมันจะถูกจัดการภายใต้การควบคุมของ VMM เมื่อเกสโอเอสทำคำสั่งพิเศษของระบบหรือทำงานที่ติดต่อกับทรัพยากรฮาร์ดแวร์โดยตรง VMM จะจับการทำงานนั้น ตรวจสอบความถูกต้องและทำงานนั้นแทนเกส โดยที่ซอฟต์แวร์ของเกสไม่รู้เกี่ยวกับการทำงานนี้ (Smith & Ravi, 2005, p. 36)

VMs ถูกสร้างขึ้นจากองค์ประกอบที่แตกต่างกัน ดังนี้ (Ruest & Ruest, 2009, pp. 30-32)

- คอนฟิกไฟล์ (configuration file) คือไฟล์ที่ประกอบด้วยข้อมูลเกี่ยวกับการตั้งค่าต่างๆสำหรับเวอร์ชวลแมชชีน ได้แก่ ขนาด RAM จำนวนโปรเซสเซอร์ จำนวนและประเภทของ

เน็ตเวิร์คอินเตอร์เฟสการ์ด (NICs) และจำนวนและประเภทของเวอร์ชวลดิสก์ โดยแต่ละครั้งที่สร้างเวอร์ชวลแมชชีนเครื่องใหม่ คอนฟิกไฟล์ของเวอร์ชวลแมชชีนเครื่องนั้นจะถูกสร้างขึ้น ซึ่งไฟล์นี้จะบอกเวอร์ชวลไลเซชันซอฟต์แวร์ว่าจะจัดสรรทรัพยากรจริงจากโฮสให้กับเวอร์ชวลแมชชีนได้อย่างไร โดยการระบุตำแหน่งที่ฮาร์ดดิสก์ไฟล์อยู่ ขนาด RAM ที่จะใช้ วิธีการโต้ตอบกับเน็ตเวิร์คอะแด็ปเตอร์การ์ด และโปรเซสเซอร์ตัวไหนบ้างที่จะใช้

- ฮาร์ดดิสก์ไฟล์ คือไฟล์ที่ประกอบด้วยข้อมูลที่โดยปกติจะมีอยู่ในฮาร์ดดิสก์จริง แต่ละครั้งที่สร้างเวอร์ชวลแมชชีน เวอร์ชวลไลเซชันซอฟต์แวร์จะสร้างเวอร์ชวลฮาร์ดดิสก์ขึ้นมา นั่นคือไฟล์ที่จะทำงานเหมือนกับดิสก์ที่มีเซ็กเตอร์ทั่วไป เมื่อติดตั้งระบบปฏิบัติการบนเวอร์ชวลแมชชีน มันจะถูกใส่เข้าไปในไฟล์นี้ และเหมือนกับระบบจริง คือแต่ละเวอร์ชวลแมชชีนสามารถมีดิสก์ไฟล์ได้หลายไฟล์ เนื่องจากมันจำลองฮาร์ดดิสก์ขึ้นมา โดยปกติไฟล์นี้จึงสำคัญในด้านขนาด โดยระบบสามารถเริ่มต้นด้วยไฟล์ขนาดเล็ก และค่อยๆเพิ่มขนาดขึ้นเมื่อเนื้อหาใหม่ถูกใส่เข้าไปในเวอร์ชวลแมชชีน
- ไฟล์สถานะของเวอร์ชวลแมชชีน เช่นเดียวกับเครื่องจริง เวอร์ชวลแมชชีนสนับสนุนโหมดปฏิบัติการที่คล้ายกับสแตนบายหรือไฮเบอร์เนชัน ในแง่ของเวอร์ชวลไลเซชันหมายถึงการหยุดชั่วคราวหรือค้างการทำงานไว้เหมือนกับการบันทึกสถานะของเครื่อง เมื่อเครื่องถูกค้างการทำงานไว้ชั่วคราว สถานะที่ถูกหยุดของมันจะถูกบันทึกลงไปในไฟล์ เนื่องจากมีเพียงสถานะของเครื่องเท่านั้น โดยปกติไฟล์นี้จึงเล็กกว่าฮาร์ดดิสก์ไฟล์
- ไฟล์อื่นๆ คือไฟล์ที่ประกอบด้วยล็อกและข้อมูลที่เกี่ยวกับเวอร์ชวลแมชชีนอื่นๆ

สถาปัตยกรรมของเวอร์ชวลแมชชีน

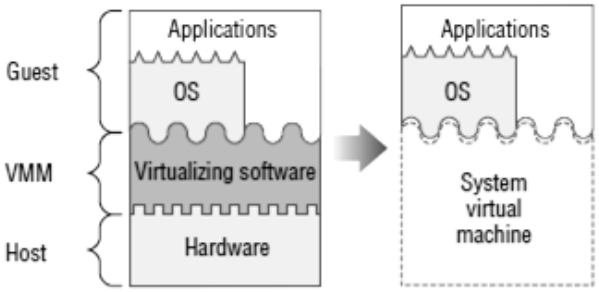
จากมุมมองของโปรเซสที่เอ็กซิคิวต์ยูเซอร์โปรแกรม แมชชีนจะประกอบไปด้วยพื้นที่แอดเดรสของหน่วยความจำที่จัดไว้ให้กับโปรเซสพร้อมกับคำสั่งระดับยูเซอร์และรีจิสเตอร์ที่ทำให้สามารถเอ็กซิคิวต์โค้ดที่เป็นของโปรเซสได้ ไอโอของเครื่องจะมองเห็นได้โดยผ่านทางระบบปฏิบัติการเท่านั้น และวิธีการเดียวที่โปรเซสสามารถโต้ตอบกับระบบไอโอคือผ่านทางซิสเต็มส์คอล จากมุมมองของระบบปฏิบัติการและแอพพลิเคชันของมัน ระบบทั้งระบบจะรันบนเครื่องจริง ซึ่งระบบคือสภาพแวดล้อมของการเอ็กซิคิวต์ที่สามารถสนับสนุนโปรเซสจำนวนมากได้ในเวลาเดียวกัน โปรเซสเหล่านี้จะใช้ไฟล์ซิสเต็มส์และทรัพยากรไอโออื่นๆร่วมกัน โดยระบบจะจัดสรรหน่วยความจำจริงและทรัพยากรไอโอให้กับโปรเซสต่างๆ และยอมให้โปรเซสสามารถ

โต้ตอบกับทรัพยากรของมันได้ เนื่องจากมีมุมมองของโปรเซสและซิสเต็มส์ที่มีต่อ “แมชชีน” เวอร์ชวลแมชชีนจึงมีสองแบบคือแบบโปรเซสและซิสเต็มส์ โปรเซสเวอร์ชวลแมชชีน (process VM) คือเวอร์ชวลแพลตฟอร์มที่เอ็กซิคิวต์โปรเซสใดโปรเซสหนึ่ง โดย VM ประเภทนี้มีอยู่เพื่อช่วยเหลือโปรเซสเท่านั้น โดยจะถูกสร้างขึ้นเมื่อโปรเซสถูกสร้างขึ้นและจะจบการทำงานเมื่อโปรเซสจบการทำงาน (ตัวอย่างเช่น Java Virtual Machine)

ในทางตรงข้าม ซิสเต็มส์เวอร์ชวลแมชชีน (system VM) จะสร้างสภาพแวดล้อมของระบบที่สมบูรณ์และคงอยู่ตลอด ซึ่งสนับสนุนระบบปฏิบัติการพร้อมกับยูเซอร์โปรเซสจำนวนมากของมัน และช่วยให้เกสโอเอส (guest operating system) สามารถใช้งานทรัพยากรที่เป็นเวอร์ชวลฮาร์ดแวร์ซึ่งรวมทั้งเน็ตเวิร์ค ไอโอ และอาจรวมถึงกราฟฟิคยูเซอร์อินเตอร์เฟสพร้อมกับโปรเซสเซอร์และหน่วยความจำด้วย โปรเซสหรือซิสเต็มส์ที่รันบน VM คือเกส (guest) และแพลตฟอร์มจริงที่สนับสนุน VM คือโฮส (host) ซอฟต์แวร์เวอร์ชวลไลซ์ที่อิมพลีเมนต์โปรเซสเวอร์ชวลแมชชีนมักตั้งชื่อว่ารันไทม์ ซึ่งมาจากรันไทม์ซอฟต์แวร์ (runtime software) ส่วนซอฟต์แวร์เวอร์ชวลไลซ์ในซิสเต็มส์เวอร์ชวลแมชชีนโดยทั่วไปหมายถึงเวอร์ชวลแมชชีนมอนิเตอร์ (Virtual Machine Monitor หรือ VMM)

ภาพที่ 2.4

การแปลง ISA ของซอฟต์แวร์เวอร์ชวลไลซ์ในซิสเต็มส์เวอร์ชวลแมชชีน



ที่มา: “The architecture of virtual machines,” โดย Smith, J. E., & Ravi, N., 2005, *Computer, 38*(5), น. 34.

จากภาพที่ 2.4 ในซิสเต็มส์เวอร์ชวลแมชชีน ซอฟต์แวร์ที่ทำเวอร์ชวลไลซ์จะอยู่ระหว่างฮาร์ดแวร์ของโฮสและซอฟต์แวร์ของเกส VMM จะจำลอง ISA ของฮาร์ดแวร์ เพื่อที่ซอฟต์แวร์ของเกสจะได้สามารถเอ็กซิคิวต์ ISA ที่แตกต่างจากที่อิมพลีเมนต์อยู่บนโฮสได้ อย่างไรก็

ตาม ในหลายแอพพลิเคชันซิสเต็มส์เวอร์ชวลแมชชีน VMM ไม่ได้ทำการจำลองคำสั่ง แต่หน้าที่หลักของมันคือจัดทรัพยากรฮาร์ดแวร์ที่ถูกเวอร์ชวลไลซ์ให้กับเกส (Smith & Ravi, 2005, p. 34)

จากมุมมองของผู้ใช้ ซิสเต็มส์เวอร์ชวลแมชชีนส่วนใหญ่จะทำงานได้เหมือนกัน แต่แตกต่างกันในรายละเอียดการอิมพลีเมนต์ วิธีการแบบเวอร์ชวลแมชชีนมอนิเตอร์ ซึ่งเป็นการวาง VMM ไว้บนฮาร์ดแวร์จริงและให้เวอร์ชวลแมชชีนอยู่ด้านบน โดย VMM จะรันในโหมดที่มีสิทธิสูงสุด ขณะที่เกสซิสเต็มส์ทั้งหมดจะรันด้วยสิทธิที่ต่ำกว่า เพื่อให้ VMM สามารถแทรกแซงและจำลองการกระทำของเกสโอเอสทั้งหมดที่จะใช้งานหรือจัดการทรัพยากรฮาร์ดแวร์ได้

การอิมพลีเมนต์ซิสเต็มส์เวอร์ชวลแมชชีนแบบโฮสเวอร์ชวลแมชชีน (host virtual machine) จะสร้างซอฟต์แวร์เวอร์ชวลไลซ์ด้านบนโฮสโอเอส ข้อดีของโฮสเวอร์ชวลแมชชีนคือผู้ใช้จะติดตั้งมันเหมือนกับเป็นแอพพลิเคชันโปรแกรมปกติ นอกจากนั้น ซอฟต์แวร์เวอร์ชวลไลซ์ยังสามารถอาศัยโฮสโอเอสให้จัดดีไวซ์ไดรเวอร์และเซอร์วิสระดับล่างอื่นๆให้แทนที่จะอาศัย VMM

ในซิสเต็มส์เวอร์ชวลแมชชีนปกติ ซิสเต็มส์ซอฟต์แวร์ของทั้งโฮสและเกสรวมทั้งแอพพลิเคชันซอฟต์แวร์จะใช้ ISA เดียวกับฮาร์ดแวร์จริง อย่างไรก็ตาม ในบางสถานการณ์ ระบบโฮสและเกสไม่ได้มี ISA เดียวกัน ดังนั้น เวอร์ชวลแมชชีนแบบทั้งระบบสามารถแก้ปัญหานี้ได้โดยการเวอร์ชวลไลซ์ซอฟต์แวร์ทั้งหมดซึ่งรวมทั้งระบบปฏิบัติการและแอพพลิเคชันด้วย เนื่องจาก ISA แตกต่างกัน เวอร์ชวลแมชชีนจึงต้องจำลองโค้ดทั้งแอพพลิเคชันและระบบปฏิบัติการ เวอร์ชวลแมชชีนซอฟต์แวร์จะเอ็กซิคิวต์เหมือนกับเป็นแอพพลิเคชันโปรแกรมที่ได้รับการสนับสนุนโดยโฮสโอเอสและไม่ใช้โอเปอเรชัน ISA ของระบบ

เมื่อโฮสแพลตฟอร์มเป็นมัลติโปรเซสเซอร์ขนาดใหญ่ที่ใช้งานหน่วยความจำร่วมกัน จุดประสงค์ที่สำคัญคือการแบ่งระบบขนาดใหญ่ออกเป็นระบบมัลติโปรเซสเซอร์ที่เล็กลงหลายๆ ระบบโดยการกระจายทรัพยากรฮาร์ดแวร์ของระบบใหญ่ ด้วยการแบ่งแบบฟิสิคัล ทรัพยากรจริงที่เวอร์ชวลซิสเต็มส์หนึ่งใช้จะแยกออกจากทรัพยากรที่ถูกใช้โดยเวอร์ชวลซิสเต็มส์อื่นๆ การแบ่งแบบฟิสิคัลนี้ทำให้มีระดับของการแยกออกจากกันสูง ดังนั้น ปัญหาของซอฟต์แวร์หรือข้อผิดพลาดของฮาร์ดแวร์บนพื้นที่แบ่งส่วนหนึ่งจะไม่มีผลกระทบต่อโปรแกรมในพื้นที่แบ่งส่วนอื่น ส่วนการแบ่งแบบโลจิคอล ทรัพยากรฮาร์ดแวร์จริงจะถูกแบ่งเวลาระหว่างพื้นที่แบ่งต่างๆ ซึ่งทำให้การใช้งานทรัพยากรระบบดีขึ้น แต่จะสูญเสียประโยชน์บางอย่างของการแยกออกจากกันของฮาร์ดแวร์ไป โดยปกติ เทคนิคการแบ่งพื้นที่ทั้งสองแบบใช้ซอฟต์แวร์หรือเฟิร์มแวร์พิเศษที่มีการดัดแปลงฮาร์ดแวร์เฉพาะกับพื้นที่แบ่งที่ต้องการ

การใช้งานฟังก์ชันและการรันบนสถาปัตยกรรมที่แตกต่างกันได้เป็นเป้าหมายของซิสเต็มส์เวอร์ชวลแมชชีนส่วนใหญ่ที่ถูกอิมพลีเมนต์บนฮาร์ดแวร์ที่ถูกพัฒนาขึ้นสำหรับ ISA มาตรฐาน ในทางตรงข้าม เวอร์ชวลแมชชีนแบบโคดีไซน์ (codesigned virtual machine) อิมพลีเมนต์ ISA ใหม่ที่มีเป้าหมายที่การปรับปรุงประสิทธิภาพและการใช้พลังงานให้ดีขึ้น โดยอาจมีการสร้าง ISA ของโฮสใหม่ทั้งหมดหรือเพิ่มขยาย ISA ที่มีอยู่ก็ได้ เวอร์ชวลแมชชีนแบบนี้ไม่มีแอพพลิเคชัน ISA จริง แต่ VMM จะกลายเป็นส่วนหนึ่งของการอิมพลีเมนต์ฮาร์ดแวร์ มีหน้าที่อย่างเดียวคือจำลอง ISA ของเกส โดย VMM จะอยู่ในพื้นที่ของหน่วยความจำที่ถูกซ่อนจากซอฟต์แวร์ทั้งหมด ซึ่งรวมทั้งตัวแปลงไบนารีที่เปลี่ยนคำสั่งเกสให้เป็นคำสั่ง ISA ของโฮสและแคชคำสั่งเหล่านั้นเก็บไว้ในพื้นที่ของหน่วยความจำที่ถูกซ่อนไว้ (Smith & Ravi, 2005, pp. 36-37)

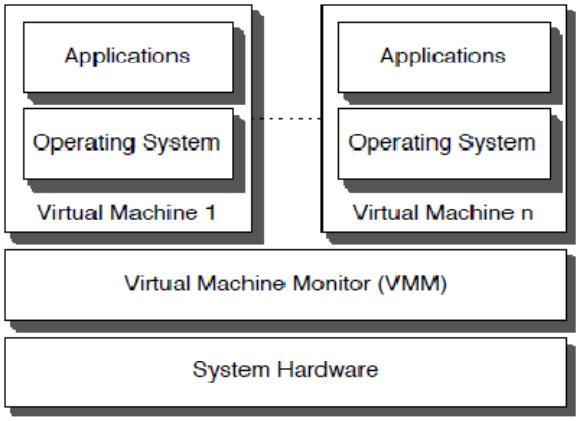
เวอร์ชวลแมชชีนมอนิเตอร์ (Virtual Machine Monitor)

ในปลายปี 1960 เวอร์ชวลแมชชีนมอนิเตอร์กลายเป็นซอฟต์แวร์เลเยอร์ที่แบ่งฮาร์ดแวร์แพลตฟอร์มออกเป็นเวอร์ชวลแมชชีนหลายเครื่อง แต่ละเวอร์ชวลแมชชีนเหล่านี้มีความใกล้เคียงกับเครื่องจริงเพียงพอที่จะรันซอฟต์แวร์ที่มีอยู่ได้โดยไม่ต้องเปลี่ยนแปลงแก้ไขซอฟต์แวร์ อย่างไรก็ตาม ในช่วงปี 1980 และ 1990 ระบบปฏิบัติการแบบมัลติทาสกิ้งและการลดลงของราคาฮาร์ดแวร์ทำให้คุณค่าของ VMM ลดลง สถาปัตยกรรมคอมพิวเตอร์ไม่มีฮาร์ดแวร์ที่จำเป็นในการอิมพลีเมนต์ VMM ได้อย่างมีประสิทธิภาพ ข้ามเข้าสู่ปี 2005 VMM กลายเป็นประเด็นที่ได้รับความสนใจอีกครั้งในวงการการศึกษาและอุตสาหกรรม บริษัทผู้ผลิตที่สำคัญหลายแห่งได้พัฒนาเทคโนโลยีเวอร์ชวลแมชชีนของตนเองขึ้น Intel, AMD, Sun Microsystems, และ IBM ได้พัฒนาเทคนิคเวอร์ชวลไลเซชันที่มีเป้าหมายทางการตลาดที่มีเงินทุนหลายพันล้าน ส่วนในด้านงานวิจัย นักวิจัยได้พยายามพัฒนาวิธีการที่ใช้เวอร์ชวลแมชชีนเพื่อแก้ปัญหาเกี่ยวกับการเคลื่อนที่ ความปลอดภัย และความสามารถในการจัดการ ในปี 1990 นักวิจัยที่ Stanford University เริ่มมองไปที่ศักยภาพของเวอร์ชวลแมชชีนในการแก้ไขข้อจำกัดของฮาร์ดแวร์และระบบปฏิบัติการ ซึ่งปัญหาเกิดจากเครื่อง MPP (Massively Parallel Processing) ที่ยากที่จะโปรแกรมและไม่สามารถรันระบบปฏิบัติการที่มีอยู่ได้ แต่ด้วยเวอร์ชวลแมชชีน จะสามารถทำให้สถาปัตยกรรมที่ยากจะควบคุมเหล่านี้ดูคล้ายกับแพลตฟอร์มที่มีอยู่เพื่อให้สามารถใช้ระบบปฏิบัติการต่างๆได้ (Rosenblum & Garfinkel, 2005, p. 39) จากภาพที่ 2.5 เวอร์ชวลแมชชีนมอนิเตอร์ (VMM) คือส่วนหนึ่งของซอฟต์แวร์ที่ดูแลเกสเวอร์ชวลแมชชีน โดย VMM มักจะหมายถึงโฮส และเวอร์ชวล

แมชชีนหมายถึงเกส VMM สร้างเวอร์ชวลโปรเซสเซอร์และส่วนอื่นๆของอุปกรณ์ระบบที่ถูกเวอร์ชวลไลซ์ เช่น อุปกรณ์ไอโอ ที่เก็บข้อมูล หน่วยความจำ นอกจากนี้ VMM ยังทำให้มีการแยกออกจากกันระหว่างเวอร์ชวลแมชชีนที่มันจะต้องจัดการ ดังนั้นปัญหาที่เกิดขึ้นในเวอร์ชวลแมชชีนเครื่องหนึ่งจะไม่มีผลกระทบต่ออีกเครื่องหนึ่ง (Rose, 2004, p. 3)

ภาพที่ 2.5

ความสัมพันธ์ระหว่างเวอร์ชวลแมชชีนมอนิเตอร์และเวอร์ชวลแมชชีน



ที่มา: “Survey of System Virtualization Techniques,” โดย Rose, R., 2004, from http://www.robertwrose.com/vita/rose-virtualization.pdf, น. 3.

การอิมพลีเมนต์ VMM

VMM ต้องมีฮาร์ดแวร์อินเตอร์เฟสให้กับซอฟต์แวร์ในเวอร์ชวลแมชชีนที่เทียบเท่ากับฮาร์ดแวร์จริงและรักษาการควบคุมเครื่องและความสามารถที่จะเข้าไปแทรกเมื่อมีการเข้าถึงฮาร์ดแวร์ โดยมีเทคนิคหลายเทคนิคที่สามารถทำได้ ซึ่งเป้าหมายของการออกแบบ VMM ที่สำคัญคือ ความเข้ากันได้ ประสิทธิภาพ และความง่าย ประโยชน์ที่สำคัญที่สุดของ VMM คือความเข้ากันได้หรือความสามารถในการรันซอฟต์แวร์เก่าได้ ส่วนในเรื่องของประสิทธิภาพนั้น โอเวอร์เฮดของเวอร์ชวลไลเซชันจะต้องไม่มากจนเกินไปและอยู่ในระดับที่ยอมรับได้ การวัดโอเวอร์เฮดของเวอร์ชวลไลเซชันคือการรันเวอร์ชวลแมชชีนที่ความเร็วเท่ากับที่ซอฟต์แวร์จะรันบนเครื่องจริง ส่วนความง่ายเป็นเรื่องสำคัญเนื่องจากความล้มเหลวของ VMM จะทำให้เวอร์ชวลแมชชีนทั้งหมดที่รัน

อยู่บนคอมพิวเตอร์ล้มเหลวไปด้วย(Rosenblum & Garfinkel, 2005, pp. 41-43) ดังนั้น VMM ควรที่จะได้รับการออกแบบให้ง่ายและไม่มีการทำงานที่ซับซ้อนจนเกินไป

การอิมพลีเมนต์ VMM นั้นแบ่งออกเป็น 3 ส่วนใหญ่ๆ ได้แก่ ซีพียูเวอร์ชวลไลเซชัน เมมโมรีเวอร์ชวลไลเซชัน และไอโอเวอร์ชวลไลเซชัน

1. **ซีพียูเวอร์ชวลไลเซชัน** (CPU virtualization)

สถาปัตยกรรมซีพียูจะสามารถเวอร์ชวลไลซ์ได้ถ้ามันสนับสนุนเทคนิค VMM พื้นฐานเกี่ยวกับการเอ็กซิคิวต์โดยตรง หมายถึงการเอ็กซิคิวต์เวอร์ชวลแมชชีนบนเครื่องจริงในขณะที่ยอมให้ VMM ยังคงมีการควบคุมซีพียูอย่างเต็มที่ การอิมพลีเมนต์การเอ็กซิคิวต์โดยตรงพื้นฐานนี้จะต้องมีโค้ดปกติและโค้ดพิเศษ (เคอเนลระบบปฏิบัติการ) ของเวอร์ชวลแมชชีนที่รันอยู่ในโหมดปกติของซีพียูในขณะที่ VMM รันอยู่ในโหมดพิเศษ ดังนั้น เมื่อเวอร์ชวลแมชชีนพยายามที่จะทำโอเปอเรชันพิเศษ ซีพียูจะทำการแทรปภายใน VMM ซึ่งจำลองโอเปอเรชันพิเศษบนสถานะของเวอร์ชวลแมชชีนที่ VMM จัดการ

ตัวอย่างการจัดการคำสั่งของ VMM คือการออกคำสั่งที่ทำให้อินเทอร์รัพท์ใช้การไม่ได้ ซึ่งเกสโอเอสไม่สามารถทำได้ เนื่องจากเป็นการไม่ปลอดภัยเพราะจะทำให้ VMM ไม่สามารถกลับมาควบคุมซีพียูได้ โดย VMM จะแทรปโอเปอเรชันเพื่อทำให้อินเทอร์รัพท์ใช้การไม่ได้ และจากนั้นจึงบันทึกว่าอินเทอร์รัพท์นั้นไม่สามารถใช้ได้สำหรับเวอร์ชวลแมชชีนนั้น แล้วจึงเลื่อนการส่งอินเทอร์รัพท์ที่ตามมาให้เวอร์ชวลแมชชีนออกไปจนกว่ามันจะทำให้อินเทอร์รัพท์ใช้การได้อีกครั้ง ดังนั้น สิ่งสำคัญในการทำให้สถาปัตยกรรมสามารถเวอร์ชวลไลซ์ได้คือการมีแทรปซีแมนติกที่ยอมให้ VMM ใช้ซีพียูเพื่อเอ็กซิคิวต์เวอร์ชวลแมชชีนได้โดยตรง มีความปลอดภัย และเวอร์ชวลแมชชีนไม่สามารถมองเห็นได้ ซึ่งทำให้ VMM สามารถใช้การเอ็กซิคิวต์โดยตรงเพื่อสร้างการจำลองเครื่องจริงขึ้นมาสำหรับซอฟต์แวร์ที่รันอยู่ภายในเวอร์ชวลแมชชีนได้

2. **เมมโมรีเวอร์ชวลไลเซชัน** (Memory virtualization)

เทคนิคการอิมพลีเมนต์แบบเดิมสำหรับการเวอร์ชวลไลซ์หน่วยความจำคือการให้ VMM คอยเก็บแชโดว์ของโครงสร้างข้อมูลของการจัดการหน่วยความจำของเวอร์ชวลแมชชีนไว้ โครงสร้างข้อมูลนี้คือแชโดว์เพจเทเบิล (shadow page table) ซึ่งยอมให้ VMM สามารถควบคุมได้ว่าเพจใดบ้างของหน่วยความจำของเครื่องที่มีให้สำหรับเวอร์ชวลแมชชีน

เมื่อระบบปฏิบัติการที่รันอยู่ในเวอร์ชวลแมชชีนสร้างแมพพิงในเพจเทเบิลของมัน VMM จะตรวจหาความเปลี่ยนแปลงและสร้างแมพพิงในแชโดว์เพจเทเบิลซึ่งชี้ไปที่ตำแหน่งของเพจจริงในฮาร์ดแวร์เมมโมรี เมื่อเวอร์ชวลแมชชีนกำลังเอ็กซิคิวต์ ฮาร์ดแวร์จะใช้แชโดว์เพจเทเบิลสำหรับการแปลงหน่วยความจำ เพื่อที่ VMM จะได้สามารถควบคุมหน่วยความจำที่แต่ละเวอร์ชวลแมชชีนกำลังใช้อยู่ได้ เหมือนกับระบบเวอร์ชวลเมมโมรีของระบบปฏิบัติการ คือ VMM สามารถเพจเวอร์ชวลแมชชีนไปที่ดิสก์ได้ เพื่อที่หน่วยความจำที่ถูกจัดสรรให้กับเวอร์ชวลแมชชีนจะได้สามารถขยายขนาดหน่วยความจำจริงของฮาร์ดแวร์ได้ จากการที่เวอร์ชวลแมชชีนสามารถมีหน่วยความจำของเครื่องเกินกว่าปกติได้นี้ ทำให้โหลดงานของเวอร์ชวลแมชชีนต้องการฮาร์ดแวร์น้อยลง VMM สามารถควบคุมได้ว่าหน่วยความจำขนาดเท่าไรที่แต่ละเวอร์ชวลแมชชีนจะได้รับตามสิ่งที่มันต้องการ

3. **ไอโอเวอร์ชวลไลเซชัน** (I/O virtualization)

เมื่อ 30 ปีก่อน ระบบไอโอของเมนเฟรม IBM ใช้สถาปัตยกรรมที่อาศัยแชนแนลในการเข้าถึงอุปกรณ์ไอโอ โดยการสื่อสารกับแชนแนลโปรเซสเซอร์ที่แยกออกมาต่างหาก ซึ่งการใช้แชนแนลโปรเซสเซอร์ ทำให้ VMM สามารถส่งการเข้าถึงอุปกรณ์ไอโอให้เวอร์ชวลแมชชีนได้โดยตรง เป็นผลให้มีโอเวอร์เฮดของเวอร์ชวลไลเซชันสำหรับไอโอน้อยมาก แทนที่จะสื่อสารกับอุปกรณ์โดยใช้การแทรปภายใน VMM ซอฟต์แวร์ในเวอร์ชวลแมชชีนสามารถอ่านและเขียนอุปกรณ์ได้โดยตรง

เวอร์ชวลไลเซชันซอฟต์แวร์และการทำไลฟ์ไมเกรชัน

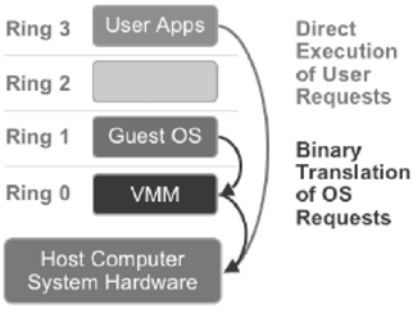
1. **วีเอ็มแวร์** (VMware)

วีเอ็มแวร์ (VMware) คือซอฟต์แวร์ทางการค้าที่ใช้เทคนิคฟูลเวอร์ชวลไลเซชัน โดยมีไฮเปอร์ไวเซอร์อยู่ระหว่างเกสโอเอสและฮาร์ดแวร์จริง ซึ่งจะยอมให้ระบบปฏิบัติการใดๆสามารถรันบนฮาร์ดแวร์ได้โดยที่ไม่ต้องมีความรู้เกี่ยวกับเกสโอเอสอื่นๆ และวีเอ็มแวร์ยังทำเวอร์ชวลไลซ์ไอโอฮาร์ดแวร์และใส่ไดรเวอร์เข้าไปในไฮเปอร์ไวเซอร์เพื่อให้อุปกรณ์มีประสิทธิภาพสูงด้วย สภาพแวดล้อมที่ถูกเวอร์ชวลไลซ์ทั้งหมดจะถูกเก็บเป็นไฟล์ นั่นคือทั้งระบบ (รวมทั้งเกสโอเอส,

VM, และเวอร์ชวลฮาร์ดแวร์) สามารถถูกไมเกรทไปที่โฮสใหม่เพื่อกระจายโหลดให้เท่ากันได้โดยง่ายและรวดเร็ว (Jones, 2006)

ภาพที่ 2.6

วิธีการแปลงไบนารีในการเวอร์ชวลไลซ์สถาปัตยกรรม x86 ของวีเอ็มแวร์



ที่มา: “Understanding Full Virtualization, Paravirtualization, and Hardware Assist,” โดย VMware, Inc., 2007a, *White Paper*, น. 4.

วีเอ็มแวร์สามารถเวอร์ชวลไลซ์ระบบปฏิบัติการ x86 ได้โดยใช้การรวมเทคนิคการแปลงไบนารีและการเอ็กซิคิวต์โดยตรงเข้าด้วยกัน ในภาพที่ 2.6 วิธีการนี้จะแปลงเคอเนลโค้ดเพื่อให้แทนที่คำสั่งที่ไม่สามารถเวอร์ชวลไลซ์ได้ด้วยลำดับคำสั่งใหม่ที่มีผลต่อเวอร์ชวลฮาร์ดแวร์ ในขณะที่โค้ดระดับยูเซอร์จะถูกเอ็กซิคิวต์ได้โดยตรงบนโปรเซสเซอร์เพื่อให้เวอร์ชวลไลเซชันมีประสิทธิภาพสูง แต่ละเวอร์ชวลแมชชีนมอนิเตอร์จะให้เซอร์วิสของระบบต่างๆทั้งหมดแก่แต่ละเวอร์ชวลแมชชีน (VM) ซึ่งรวมถึงเวอร์ชวลไบออส, เวอร์ชวลดีไวซ์ และการจัดการหน่วยความจำที่ถูกเวอร์ชวลไลซ์ การรวมการแปลงไบนารีและการเอ็กซิคิวต์โดยตรงเข้าด้วยกันนี้ทำให้ระบบเป็นฟูลเวอร์ชวลไลเซชัน ขณะที่เกสโอเอสถูกแยกออกจากฮาร์ดแวร์จริงอย่างสมบูรณ์โดยเวอร์ชวลไลเซชันเลเยอร์ เกสโอเอสจะไม่รู้ว่ามันกำลังถูกเวอร์ชวลไลซ์และไม่จำเป็นต้องมีการแก้ไขเปลี่ยนแปลง ฟูลเวอร์ชวลไลเซชันเป็นวิธีการที่ไม่ต้องการความช่วยเหลือจากฮาร์ดแวร์หรือระบบปฏิบัติการในการเวอร์ชวลไลซ์คำสั่งต่างๆ ไฮเปอร์ไวเซอร์จะแปลงคำสั่งระบบปฏิบัติการทั้งหมดขณะกำลังทำงาน และแคชผลลัพธ์ไว้เพื่อใช้ในอนาคต ในขณะที่คำสั่งระดับยูเซอร์จะรันโดยไม่ได้ถูกแก้ไขที่ความเร็วแท้จริงของเครื่อง ฟูลเวอร์ชวลไลเซชันทำให้เวอร์ชวลแมชชีนมีความเป็นอิสระจากกันและมีความปลอดภัย และทำให้การไมเกรทและการพอร์ตง่ายขึ้นโดยที่เกสโอเอส

เดิมสามารถรันแบบเวอร์ชวลไลซ์หรือรันบนฮาร์ดแวร์จริงได้ (VMware, 2007a, p. 4) โดย ESX Server ของวีเอ็มแวร์คือเครื่องมือที่ใช้สร้างแพลตฟอร์มที่ทำให้สามารถออกแบบ อิมพลีเมนต์ และจัดการโครงสร้างภายในเสมือนที่สมบูรณ์ได้ (Muller, & Wilson, 2005, p. 1)

การทำไลฟ์ไมเกรชันของเวอร์ชวลแมชชีนจากเซิร์ฟเวอร์เครื่องหนึ่งไปอีกเครื่องหนึ่งด้วย VMotion สามารถทำได้โดยใช้เทคโนโลยี 3 แบบ ดังนี้ (VMware, 2007b, p.1)

1. สถานะของเวอร์ชวลแมชชีนทั้งหมดจะถูกรวมเป็นไฟล์ชุดหนึ่งที่ถูกเก็บอยู่บนที่เก็บข้อมูลร่วมกัน เช่น Fibre Channel หรือ iSCSI Storage Area Network (SAN) หรือ Network Attached Storage (NAS) โดยกลุ่ม Virtual Machine File System (VMFS) ของวีเอ็มแวร์จะยอมให้ ESX Server หลายเครื่องสามารถเข้าถึงเวอร์ชวลแมชชีนไฟล์เดียวกันพร้อมๆกันได้

2. แอคทีฟเมมโมรีและสถานะการทำงานของเวอร์ชวลแมชชีนจะถูกส่งข้ามเน็ตเวิร์คความเร็วสูงไปอย่างรวดเร็ว ซึ่งทำให้เวอร์ชวลแมชชีนสามารถเปลี่ยนจากการรันบนเครื่อง ESX Server ต้นทางไปที่เครื่อง ESX Server ปลายทางได้โดยทันที ผู้ใช้จะไม่รับรู้ถึงช่วงของการย้าย VM เนื่องจาก VMotion จะคอยติดตามการทำงานของหน่วยความจำในบิตแมพ ทันทีที่หน่วยความจำและสถานะของระบบทั้งหมดได้ถูกคัดลอกไปที่ ESX Server เป้าหมายแล้ว VMotion จะหยุดการทำงานของเวอร์ชวลแมชชีนต้นทาง คัดลอกบิตแมพไปที่เครื่อง ESX Server เป้าหมาย และเรียกเวอร์ชวลแมชชีนขึ้นมาทำงานต่อบนเครื่อง ESX Server เป้าหมาย กระบวนการทั้งหมดนี้ใช้เวลาน้อยกว่า 2s บนกิกะบิตอีเทอร์เน็ตเน็ตเวิร์ค

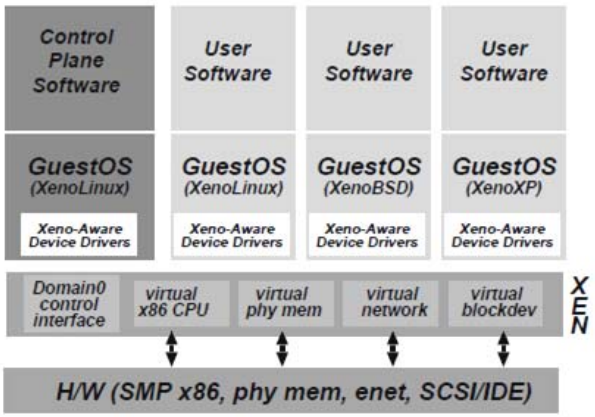
3. เน็ตเวิร์คที่ถูกใช้โดยเวอร์ชวลแมชชีนจะถูกเวอร์ชวลไลซ์โดยเครื่อง ESX Server ด้วยเพื่อให้แน่ใจว่าหลังจากการไมเกรทแล้ว ไอดีของเวอร์ชวลแมชชีนเน็ตเวิร์คและการเชื่อมต่อเน็ตเวิร์คจะยังคงอยู่ VMotion จัดการเวอร์ชวลแมคแอดเดรสแบบเป็นส่วนหนึ่งของโปรเซส ทันทีที่เครื่องปลายทางถูกเรียกให้ทำงาน VMotion จะปิงเน็ตเวิร์คเราเตอร์เพื่อให้แน่ใจว่ามันรู้เกี่ยวกับตำแหน่งใหม่ของเวอร์ชวลแมคแอดเดรส เนื่องจากการไมเกรทเวอร์ชวลแมชชีนด้วย VMotion จะเก็บรักษาสถานะการทำงาน เน็ตเวิร์คไอดี และการเชื่อมต่อเน็ตเวิร์คไว้ จึงทำให้เครื่องไม่ต้องหยุดการทำงานและไม่รบกวนผู้ใช้

2. เซน (Xen)

เซน (Xen) คือเวอร์ชวลแมชชีนมอนิเตอร์ (VMM) จัดการทรัพยากรที่ทำให้สามารถใช้แอพพลิเคชันต่างๆได้ เช่น การรวมเซิร์ฟเวอร์หลายๆระบบมาไว้ที่เซิร์ฟเวอร์เดียว, เครื่องมือสำหรับ

โฮสติ้ง, เว็บเซอร์วิสแบบกระจาย, แพลตฟอร์มการคำนวณที่มีความปลอดภัย, และการเคลื่อนย้ายแอพพลิเคชัน เซนใช้พาราเวอร์ชวลไลเซชัน โดยไฮเปอร์ไวเซอร์และระบบปฏิบัติการทำงานร่วมกันบนเวอร์ชวลไลเซชัน ซึ่งทำให้ประสิทธิภาพดีขึ้นถึงแม้ว่าจะต้องมีการแก้ไขเกสโอเอส (Barham, et al., 2003, pp. 164-165) ด้วยเทคโนโลยีนี้ เวอร์ชวลเซิร์ฟเวอร์และไฮเปอร์ไวเซอร์จะทำงานร่วมกันเพื่อให้ได้ประสิทธิภาพสูงสำหรับการเวอร์ชวลไลซ์ไอโอ ซีพียู และหน่วยความจำ (Citrix Systems, 2005a) เซนไฮเปอร์ไวเซอร์ (Xen hypervisor) ทำเซอร์วิสการจัดการในเวอร์ชวลแมชชีนที่ควบคุมการใช้งานฮาร์ดแวร์, การสื่อสารของเวอร์ชวลแมชชีน, การสตาร์ท/การหยุดเวอร์ชวลแมชชีน, และเซอร์วิสที่จำเป็นอื่นๆ การจัดการเวอร์ชวลแมชชีนเหล่านี้จะเริ่มต้นขึ้นระหว่างที่บูทระบบ และจัดการอุปกรณ์ทั้งหมดอย่างปลอดภัย (Citrix Systems, 2005b)

ภาพที่ 2.7
โครงสร้างของเครื่องที่รันเซนไฮเปอร์ไวเซอร์



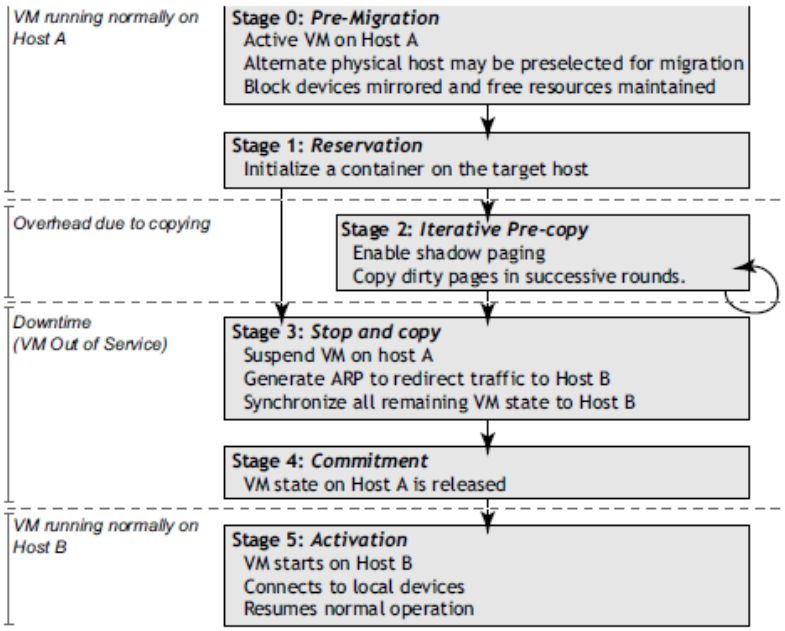
ที่มา: “Xen and the art of virtualization,” โดย Barham, et al., 2003, *Proceedings of the nineteenth ACM symposium on Operating systems principles*, น. 168.

โครงสร้างของระบบเซนแสดงในภาพที่ 2.7 โดเมนจะถูกสร้างขึ้นเมื่อบูทระบบซึ่งจะได้รับอนุญาตให้ใช้คอนโทรลอินเตอร์เฟสได้ โดเมนเริ่มต้นนี้เรียกว่า Domain0 มีหน้าที่ดูแลซอฟต์แวร์การจัดการระดับแอพพลิเคชัน คอนโทรลอินเตอร์เฟสช่วยให้สามารถสร้างและจบการทำงานโดเมนอื่นๆได้ และสามารถควบคุมพารามิเตอร์การจัดลำดับงานที่เกี่ยวข้อง, การจัดสรรฟิสิคัลเมมโมรี และการใช้งานฟิสิคัลดิสก์ของเครื่องและอุปกรณ์เน็ตเวิร์ค (Barham, et al., 2003, pp. 168)

สำหรับการอิมพลีเมนต์ไลฟ์ไมเกรชันในเซนไฮเปอร์ไวเซอร์ การเคลื่อนย้ายหน่วยความจำของ VM จากเครื่องหนึ่งไปอีกเครื่องหนึ่งมีหลายวิธี อย่างไรก็ตาม เมื่อ VM กำลังรันไลฟ์เซอร์วิสอยู่ การเคลื่อนย้ายนี้จะต้องเกิดขึ้นในลักษณะที่มีความสมดุลระหว่างความต้องการที่จะลดการหยุดการทำงานของเครื่องและเวลาที่ทำไมเกรชันทั้งหมดให้น้อยที่สุด ขั้นตอนที่เกิดขึ้นเมื่อไมเกรทโอเอสแสดงในภาพที่ 2.8 โดยใช้วิธีการเดิมในการจัดการไมเกรชันเพื่อให้มีความปลอดภัยและการจัดการความผิดพลาด โดยมองกระบวนการไมเกรชันว่าเป็นการทำงานโต้ตอบกันระหว่างเครื่องสองเครื่องที่เกี่ยวข้องกัน วิธีการนี้สามารถจัดการกับความผิดพลาดได้เนื่องจากมีอย่างน้อยหนึ่งโฮสที่มีอิมเมจของ VM ที่สอดคล้องกันตลอดเวลาในระหว่างไมเกรชัน โดยมีเงื่อนไขว่าโฮสเดิมจะยังคงทำงานได้เหมือนเดิมจนกระทั่งไมเกรชันคอมมิท ดังนั้น ถ้ามีความผิดพลาดเกิดขึ้นในระหว่างไมเกรชันบนโฮสใหม่หรือระบบเน็ตเวิร์ค การไมเกรทจะถูกยกเลิก VM อาจถูกหยุดและเรียกให้ทำงานต่อบนโฮสนั้นโดยที่ไม่มีความเสี่ยงที่จะเกิดความผิดพลาด (Clark, et al., 2005, pp. 275-277)

ภาพที่ 2.8

ไลฟ์ไมเกรชันในเซนไฮเปอร์ไวเซอร์



ที่มา: “Live Migration of Virtual Machines,” โดย Clark, et al., 2005, *the 2nd Symposium on Networked Systems Design & Implementation*, น. 276.

จากภาพที่ 2.8 การทำเวอร์ชวลแมชชีนไมเกรชันมีขั้นตอนดังนี้

ขั้นตอนที่ 0: Pre-Migration เพื่อให้การไมเกรท VM ที่ทำงานอยู่บนฟิสิคัลโฮส A เป็นไปได้อย่างรวดเร็ว อาจจะต้องมีการเลือกโฮสเป้าหมายไว้ล่วงหน้า เพื่อจะได้มีการยืนยันทรัพยากรที่จำเป็นในการรับไมเกรชัน

ขั้นตอนที่ 1: Reservation มีการร้องขอให้ไมเกรทโอเอสจากโฮส A ไปที่โฮส B โดยเริ่มต้นจะทำการยืนยันว่าทรัพยากรที่จำเป็นมีอยู่บนเครื่อง B และจองพื้นที่สำหรับ VM ความผิดพลาดที่เกิดขึ้นในขั้นนี้จะทำให้ VM ทำงานต่อบนเครื่อง A โดยที่ไม่ได้รับผลกระทบใดๆ

ขั้นตอนที่ 2: Iterative Pre-Copy ในระหว่างการทำงานรอบแรก เพจทั้งหมดจะถูกย้ายจากเครื่อง A ไป B รอบต่อไปจะคัดลอกเฉพาะเพจที่มีการแก้ไขในระหว่างช่วงการย้ายเพจก่อนหน้านี้

ขั้นตอนที่ 3: Stop-and-Copy หยุดการทำงานของโอเอสที่รันอยู่ที่เครื่อง A และย้ายเส้นทางเน็ตเวิร์คของมันไปที่เครื่อง B จากนั้นสถานะของซีพียูและเมมโมรีเพจที่ไม่สอดคล้องที่เหลืออยู่จะถูกย้ายไป เมื่อสิ้นสุดขั้นตอนนี้จะมีสำเนาที่เหมือนกันของ VM ที่ถูกหยุดอยู่ที่ทั้งเครื่อง A และ B สำเนาที่เครื่อง A จะยังคงเป็นสำเนาหลักและจะถูกเรียกขึ้นมาทำงานในกรณีเกิดความผิดพลาด

ขั้นตอนที่ 4: Commitment โฮส B บอกโฮส A ว่ามันได้รับโอเอสอิมเมจที่สอดคล้องกันเรียบร้อยแล้ว โฮส A รับรู้ว่าเมสเซจนี้เป็นข้อตกลงของการทำไมเกรชัน โดยโฮส A อาจจบการทำงานของ VM เดิม และโฮส B จะกลายเป็นโฮสหลักแทน

ขั้นตอนที่ 5: Activation VM ที่ถูกไมเกรทบนเครื่อง B จะทำงานต่อจากจุดที่หยุดในขั้นตอนที่ 3

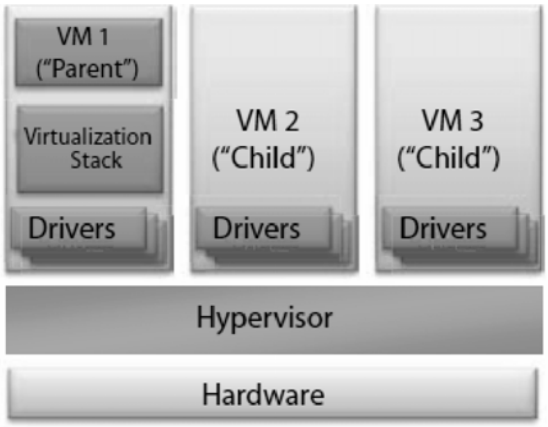
3. **ไฮเปอร์-วี** (Hyper-V)

ไฮเปอร์ไวเซอร์ Hyper-V เป็นเลเยอร์บางๆของโค้ดที่อยู่บนฮาร์ดแวร์ ซึ่งได้รับการพัฒนาภายใต้ Microsoft Security Development Lifecycle ไมโครซอฟท์พัฒนาไฮเปอร์-วีโดยใช้สถาปัตยกรรมไมโครเคอเนล ดังภาพที่ 2.9 ในโมเดลนี้ ไฮเปอร์ไวเซอร์เป็นเลเยอร์บางๆระหว่างเกสและฮาร์ดแวร์ ซึ่งมีการทำงานที่ช่วยขยายเวอร์ชวลไลเซชันให้โปรเซสเซอร์ นอกจากนี้ ไมโครเคอเนลไฮเปอร์ไวเซอร์ยังสนับสนุนฮาร์ดแวร์ได้มากขึ้น เนื่องจากผู้ผลิตฮาร์ดแวร์ส่วนใหญ่สร้างไดรเวอร์ระบบปฏิบัติการอยู่แล้วและไม่จำเป็นต้องสร้างไฮเปอร์ไวเซอร์ไดรเวอร์ต่างหากอีก ด้วยเกสที่

ใช้ไดรเวอร์ของมันเอง ขนาดของส่วนประกอบกลไกการรักษาความปลอดภัยหรือ TCB (trusted computing base) จะลดลงเนื่องจากเกสไม่ได้มีเส้นทางที่ผ่านไดรเวอร์ของ “Parent” (Microsoft Corporation, 2008a, pp. 21-22)

ภาพที่ 2.9

ไมโครเคอเนลไฮเปอร์ไวเซอร์ (Microkernel Hypervisor)



ที่มา: “Microsoft Virtualization from Data Center to Desktop,” โดย Microsoft Corporation, 2008a, *White Paper*, น. 22.

วิธีการของไมโครเคอเนลทำให้ไฮเปอร์ไวเซอร์มีขนาดเล็กและไม่ต้องมีโค้ดพิเศษรันอยู่ภายในไฮเปอร์ไวเซอร์ โดยไฮเปอร์ไวเซอร์จะยอมให้ระบบปฏิบัติการหลายระบบรันโดยไม่ต้องถูกแก้ไขในเวลาเดียวกันได้ หลังจากที่ติดตั้งไฮเปอร์-วี ระบบปฏิบัติการทั้งหมดบนเครื่องคอมพิวเตอร์จะรันเป็นเวอร์ชวลแมชชีน แม้แต่ระบบ Windows Server 2008 ที่ใช้เพื่อสร้างและจัดการเวอร์ชวลแมชชีนก็เป็นเวอร์ชวลแมชชีนด้วย ซึ่งเรียกว่า management operating system (Solution Accelerators, 2009, p. 9)

ไลฟ์ไมเกรชันของไฮเปอร์-วี ถูกรวมเข้าไปอยู่ใน Windows Server 2008 R2 Hyper-V และ Microsoft Hyper-V Server 2008 R2 ทำให้สามารถเคลื่อนย้าย VM ที่กำลังรันอยู่จากฟิสิคัลโฮสที่มีไฮเปอร์-วีเครื่องหนึ่งไปยังอีกเครื่องหนึ่งโดยที่ไม่มีการรบกวนเซอร์วิสหรือหยุดการทำงานของเวอร์ชวลแมชชีน การไมเกรทเกิดขึ้นโดยการคัดลอกหน่วยความจำของ VM ที่จะไมเกรทล่วงหน้าไปที่เครื่องปลายทาง ไลฟ์ไมเกรชันจะลดเวลาการย้าย VM ให้น้อยที่สุด โดยผู้ดูแลระบบหรือสคริปต์ที่เริ่มต้นไลฟ์ไมเกรชันสามารถควบคุมได้ว่าคอมพิวเตอร์เครื่องไหนที่เป็นเครื่อง

ปลายทางสำหรับไลฟ์ไมเกรชัน เกสโอเอสของ VM ที่ไมเกรทจะไม่รู้ว่าเกิดไมเกรชันขึ้น ดังนั้นจึงไม่จำเป็นต้องมีการคอนฟิกพิเศษสำหรับเกสโอเอส (Microsoft Corporation, 2008b, pp. 4-5)

4. **คีมู** (QEMU)

คีมู (QEMU) คือแมชชีนอีมูเลเตอร์ที่สามารถรันระบบปฏิบัติการเป้าหมายที่ไม่ได้ถูกแก้ไข (เช่น วินโดวส์หรือลินุกซ์) และแอพพลิเคชันของมันทั้งหมดในเวอร์ชวลแมชชีน ตัวคีมูจะรันบนระบบปฏิบัติการที่เป็นโฮสได้หลายระบบ เช่น ลินุกซ์ (Linux), วินโดวส์ (Windows) และแมคโอเอสเท็น (Mac OS X) โดยซีพียูของโฮสและเกสอาจจะแตกต่างกันได้ การใช้คีมูโดยหลักคือเพื่อรันระบบปฏิบัติการหนึ่งบนอีกระบบหนึ่ง เช่น วินโดวส์บนลินุกซ์ หรือลินุกซ์บนวินโดวส์ การใช้อีกอย่างหนึ่งคือเพื่อแก้ไขข้อผิดคลาดของโค้ด เนื่องจากเวอร์ชวลแมชชีนสามารถถูกหยุดได้โดยง่ายและสถานะของมันสามารถถูกตรวจสอบ บันทึก และเรียกให้ทำงานต่อได้ นอกจากนั้น อุปกรณ์ระบบฝังตัวเฉพาะบางอย่างสามารถถูกจำลองได้โดยการใส่แมชชีนเดสคริปชันใหม่และอุปกรณ์ที่ถูกจำลองใหม่เข้าไปได้ และคีมูยังได้รวมยูเซอร์โหมดอีมูเลเตอร์ที่เฉพาะกับลินุกซ์เข้าไปด้วย โดยเป็นส่วนหนึ่งของแมชชีนอีมูเลเตอร์ที่รันลินุกซ์โปรเซสสำหรับซีพียูเป้าหมายบนอีกซีพียูหนึ่ง ซึ่งโดยหลักจะถูกใช้เพื่อทดสอบผลของคอมไพเลอร์ข้ามแพลตฟอร์ม หรือเพื่อทดสอบซีพียูอีมูเลเตอร์โดยที่ไม่ต้องสตาร์ทเวอร์ชวลแมชชีนทั้งเครื่องขึ้นมา คีมูประกอบด้วยระบบย่อยหลายระบบ ดังนี้ (Bellard, 2005, pp. 41-42)

- ซีพียูอีมูเลเตอร์ (x86, PowerPC, ARM และ Sparc)
- อุปกรณ์ที่ถูกจำลอง (เช่น วีจีเอดิสเพลย์, ซีเรียลพอร์ต 16450, เม้าส์และคีบอร์ด PS/2, ไอดีอีฮาร์ดดิสก์, เน็ตเวิร์คการ์ด NE2000)
- ดีไวซ์ทั่วไป (เช่น บล็อกดีไวซ์, คาแรกเตอร์ดีไวซ์, เน็ตเวิร์คดีไวซ์) ที่ใช้เพื่อต่ออุปกรณ์ที่ถูกจำลองเข้ากับอุปกรณ์ของโฮสที่สอดคล้องกัน
- แมชชีนเดสคริปชัน (เช่น PC, PowerMac, Sun4m) ที่เป็นตัวแทนของอุปกรณ์ที่ถูกจำลอง
- ดีบักเกอร์
- ยูเซอร์อินเตอร์เฟส

5. เควีเอ็ม (KVM)

ระบบเคอเนลเบสเวอร์ชวลแมชชีน (Kernel-based Virtual Machine) หรือเควีเอ็ม (KVM, 2009) คือระบบซอฟต์แวร์ที่สร้างมาจากพื้นฐานของระบบคีมู (QEMU) และทำงานแบบฟูลเวอร์ชวลไลเซชัน (full virtualization) สำหรับลินุกซ์บนฮาร์ดแวร์ x86 ที่มีการเพิ่มเวอร์ชวลไลเซชันเข้าไป (Intel VT หรือ AMD-V) (Kivity, Kamay, Laor, Lublin, & Liguori, 2007, pp. 225-229; Uhlig, et al., 2005, pp. 48–56) เควีเอ็มประกอบด้วยเคอเนลโมดูลคือ kvm.ko ที่มีระบบพื้นฐานของเวอร์ชวลไลเซชันหลักที่สำคัญและโมดูลที่เฉพาะเจาะจงกับโปรเซสเซอร์คือ kvm-intel.ko (สำหรับโปรเซสเซอร์ของ Intel) หรือ kvm-amd.ko (สำหรับโปรเซสเซอร์ของ AMD) และนอกจากนี้ยังมีโค้ดของคีมูที่ได้รับการแก้ไขรวมอยู่ด้วย

เควีเอ็มเป็นไฮเปอร์ไวเซอร์แรกที่กลายเป็นส่วนหนึ่งของลินุกซ์เคอเนล (2.6.20) และได้รับการพัฒนาโดย Avi Kivity ผ่านทาง Qumranet ในตอนเริ่มต้น ซึ่งปัจจุบันเป็นของ Red Hat (Jones, 2009, p. 4)

ฮาร์ดแวร์ x86 เป็นสถาปัตยกรรมที่ยากที่จะทำเวอร์ชวลไลซ์ บางคำสั่งที่มีสถานะพิเศษจะไม่ทำการแทรปเมื่อถูกเอ็กซิคิวต์ในยูเซอร์โหมด เช่น popf และสถานะพิเศษบางอย่างก็ยากที่จะซ่อน เช่น ระดับสิทธิที่โปรเซสเซอร์กำลังทำงานอยู่หรือ cpl เนื่องจากผู้ผลิตฮาร์ดแวร์ (Intel และ AMD) เห็นความสำคัญของเวอร์ชวลไลเซชัน จึงได้เพิ่มการขยายสถาปัตยกรรม x86 ที่ทำให้สามารถทำเวอร์ชวลไลเซชันได้ง่ายขึ้น การสนับสนุนการทำเวอร์ชวลไลเซชันในระดับฮาร์ดแวร์มีหลักการดังนี้

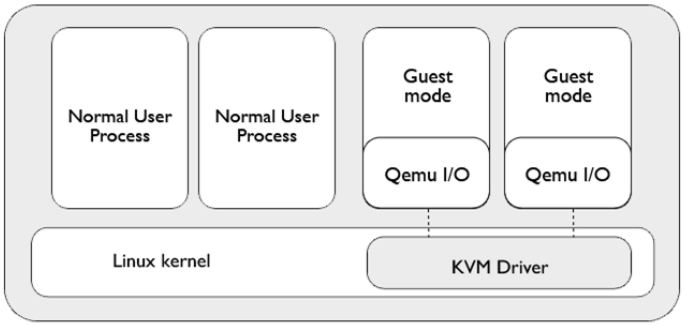
- โหมดปฏิบัติการเกส โปรเซสเซอร์สามารถเปลี่ยนเข้าไปอยู่ในเกสโหมดได้ ซึ่งมีระดับสิทธิตามปกติทั้งหมดของโหมดปฏิบัติการปกติ ยกเว้นซิสเต็มส์ซอฟต์แวร์สามารถเลือกร้องขอได้ว่าจะให้ทำการแทรปคำสั่งหรือการเข้าถึงรีจิสเตอร์ใดบ้าง
- โปรเซสเซอร์สามารถสลับสถานะฮาร์ดแวร์ เมื่อสลับเข้าหรือออกจากเกสโหมด ฮาร์ดแวร์จะเปลี่ยนคอนโทรลรีจิสเตอร์ที่มีผลต่อโหมดปฏิบัติการโปรเซสเซอร์ เช่นเดียวกับเซกเมนต์รีจิสเตอร์ และพอยเตอร์คำสั่ง เพื่อที่การย้ายการควบคุมจะได้ทำได้อย่างมีประสิทธิภาพ
- การรายงานผลสาเหตุของการออกจากโหมด เมื่อมีการเปลี่ยนจากเกสโหมดกลับไปที่โฮสโหมด ฮาร์ดแวร์จะรายงานสาเหตุของการเปลี่ยนโหมด เพื่อที่ซอฟต์แวร์จะได้สามารถทำการกระทำที่เหมาะสม

ในเควีเอ็ม เกสโอเอสจะถูกสตาร์ทในยูเซอร์สเปช แต่ละเกสโอเอสคือโปรเซสหนึ่งของโฮสโอเอส (หรือไฮเปอร์ไวเซอร์) ส่วนล่างสุดคือฮาร์ดแวร์แพลตฟอร์มที่มีความสามารถในการทำเวอร์ชวลไลเซชัน ส่วนที่รันบนฮาร์ดแวร์จริงคือไฮเปอร์ไวเซอร์ (ลินุกซ์เคอเนลที่มีเควีเอ็มโมดูล) ไฮเปอร์ไวเซอร์นี้จะดูเหมือนลินุกซ์เคอเนลปกติที่สามารถรันแอพพลิเคชันใดลงไปก็ได้ แต่เคอเนลนี้สามารถสนับสนุนเกสโอเอสได้ด้วยซึ่งถูกโหลดขึ้นมาผ่านทางเควีเอ็มยูทิลิตี้ นอกจากนี้ เกสโอเอสยังสามารถสนับสนุนแอพพลิเคชันเดียวกับที่โฮสโอเอสสนับสนุนได้ด้วย (Jones, 2007)

5.1 สถาปัตยกรรม kvm

ภายใต้เควีเอ็มเคอเนลโมดูล เวอร์ชวลแมชชีนถูกสร้างขึ้นโดยการเปิดดีไวซ์โหนด (`/dev/kvm`) โดยเกส (guest) มีหน่วยความจำของมันเอง ซึ่งแยกออกจากยูเซอร์สเปชโปรเซสที่สร้างมันขึ้นมา แต่เวอร์ชวลซีพียูไม่ได้ถูกจัดลำดับการทำงานบนตัวมันเอง ในภาพที่ 2.10 โดยการใส่ความสามารถเวอร์ชวลไลเซชันเข้าไปที่ลินุกซ์เคอเนล ซึ่งทำหน้าที่เป็นไฮเปอร์ไวเซอร์ ทุกๆ เวอร์ชวลแมชชีนก็คือลินุกซ์โปรเซสปกติที่ถูกจัดลำดับงานโดยตัวจัดลำดับงานมาตรฐานของลินุกซ์ และหน่วยความจำของมันก็จะถูกจัดสรรโดยตัวจัดสรรหน่วยความจำของลินุกซ์ โดยปกติ ลินุกซ์โปรเซสมีโหมดของการเอ็กซิคิวต์ 2 โหมด คือเคอเนลและยูเซอร์ ในโมเดลนี้ เควีเอ็มได้เพิ่มโหมดที่สามเข้าไปคือเกสโหมด ซึ่งจะมีทั้งโหมดเคอเนลและยูเซอร์ของมันเอง การรวมความสามารถของไฮเปอร์ไวเซอร์เข้าไปในลินุกซ์เคอเนลของโฮสในรูปแบบของโมดูลที่โหลดขึ้นมาได้นี้ ทำให้สภาพแวดล้อมของการเวอร์ชวลไลซ์สามารถจัดการได้ง่ายขึ้นและทำให้ประสิทธิภาพดีขึ้นได้ (Qumranet, 2006, pp. 3-5)

ภาพที่ 2.10
สถาปัตยกรรมของเควีเอ็ม



ที่มา: “KVM - Kernel-based Virtualization Machine,” โดย Qumranet Inc., 2006, *White paper*, น. 3.

1. ดีไวซ์โหนด /dev/kvm

เควีเอ็มไดรเวอร์มีโครงสร้างเป็นลินุกซ์คาแรกเตอร์ดีไวซ์ตามปกติ ซึ่งมีดีไวซ์โหนด /dev/kvm ที่สามารถใช้งานได้โดยยูเซอร์สเปช เพื่อสร้างและรันเวอร์ชวลแมชชีนผ่านทางการเรียก ioctl()

การทำงานของ /dev/kvm มีดังนี้

- การสร้างเวอร์ชวลแมชชีนเครื่องใหม่ขึ้นมา
- การจัดสรรหน่วยความจำให้แก่เวอร์ชวลแมชชีน
- การอ่านและการเขียนเวอร์ชวลซีพียูรีจิสเตอร์
- การใส่อินเทอร์รัพท์เข้าไปในเวอร์ชวลซีพียู
- การรันเวอร์ชวลซีพียู

2. การจัดการกับความแตกต่างของสถาปัตยกรรมชุดคำสั่ง

เนื่องจากไม่มีมาตรฐานของการเพิ่มเวอร์ชวลไลเซชันในฮาร์ดแวร์ โปรเซสเซอร์ของ Intel และ AMD มีคำสั่งสำหรับใช้ในการสนับสนุนฮาร์ดแวร์เวอร์ชวลไลเซชันที่แตกต่างกัน รวมทั้งมีซีแมนติกและความสามารถแตกต่างกัน เควีเอ็มจัดการความแตกต่างเหล่านี้ในวิธีการของลินุกซ์โดยการใส่ฟังก์ชันพอยเตอร์เวกเตอร์ kvm_arch_ops และใช้การเรียกฟังก์ชันที่กำหนดไว้เมื่อใดก็ตามที่มีการทำงานที่ขึ้นอยู่กับสถาปัตยกรรม ฟังก์ชันการทำงานพื้นฐานของเควีเอ็มอยู่ในโมดูล kvm.ko ส่วนฟังก์ชันการทำงานที่เฉพาะเจาะจงกับสถาปัตยกรรมจะอยู่ในโมดูล kvm-intel.ko และ kvm-amd.ko

5.2 การเวอร์ชวลไลซ์ MMU

ด้วยโปรเซสเซอร์ x86 ที่มีระบบเวอร์ชวลเมมโมรี การแปลงเวอร์ชวลแอดเดรสให้เป็นฟิสิคัลแอดเดรสทำโดย Memory Management Unit หรือ MMU ซึ่งประกอบด้วย

- เพจเทเบิล ซึ่งทำการเข้ารหัสการแปลงจากเวอร์ชวลไปเป็นฟิสิคัล
- กลไกที่ทำการเตือนซิสเต็มส์ซอฟต์แวร์เกี่ยวกับการแปลงที่ขาดหายไป (page faults)
- Translation Lookaside Buffer (TLB) คือแคชที่อยู่บนชิพที่ช่วยเพิ่มความเร็วในการค้นหาของเพจเทเบิล
- คำสั่งสำหรับการสลับฐานการแปลงเพื่อที่จะจัดให้มีพื้นที่แอดเดรสที่เป็นอิสระจากกัน
- คำสั่งสำหรับการจัดการ TLB

ปัญหาที่สำคัญคือ MMU มีระดับการแปลงเพียงหนึ่งระดับ (guest virtual → guest physical) แต่ไม่มีระดับการแปลงที่สองที่จำเป็นสำหรับเวอร์ชวลไลเซชัน (guest physical → host physical) การแก้ปัญหาทำได้โดยการใช้ความสามารถของฮาร์ดแวร์เวอร์ชวลไลเซชัน ซึ่งจะเข้ารหัสการแปลงโดยรวมการแปลงสองระดับเข้าด้วยกัน (guest virtual → host physical) ในขณะที่จำลองการโต้ตอบของฮาร์ดแวร์กับเพจเทเบิลเดิมที่เกสมีให้ แชโดว์เพจเทเบิลจะถูกสร้างขึ้นแบบเพิ่มขนาดขึ้นเรื่อยๆ โดยเริ่มจากเทเบิลว่าง และเมื่อความผิดพลาดของการแปลงถูกรายงานไปที่โฮส ข้อมูลที่หายไปจะถูกใส่เพิ่มเข้ามา

ปัญหาหลักของการใช้แชโดว์เพจเทเบิลคือการรักษาความสอดคล้องระหว่างเกสเพจเทเบิลและแชโดว์เพจเทเบิล เมื่อใดก็ตามที่เกสเขียนเพจเทเบิล จะต้องเขียนการเปลี่ยนแปลงนั้นบนแชโดว์เพจเทเบิลด้วย

1. การอิมพลีเมนต์ Virtual TLB

เวอร์ชันเริ่มแรกของอัลกอริทึมแชโดว์เพจเทเบิลในเควีเอ็มใช้วิธีง่ายๆตรงไปตรงมา โดยอัลกอริทึมนี้สร้างบนพื้นฐานความจริงที่ว่าเกสต้องใช้คำสั่งการจัดการ TLB เพื่อซิงโครไนซ์ TLB กับเพจเทเบิลของมัน ดังนั้นเราจึงใช้วิธีการดักจับคำสั่งเหล่านี้และเปลี่ยนแปลงค่าของแชโดว์เพจเทเบิลนอกเหนือจากผลการทำงานบน TLB ตามปกติ

2. การแคช Virtual MMU

เพื่อที่จะปรับปรุงประสิทธิภาพของเกสให้ดีขึ้น การอิมพลีเมนต์ virtual mmu จะถูกขยายให้สามารถแคชเพจเทเบิลข้ามคอนเท็กซ์สวิตช์ได้ ซึ่งจะเพิ่มประสิทธิภาพได้อย่างมาก จากปัญหาที่การเขียนเพจเทเบิลของเกสไม่ได้ถูกดักจับโดยเวอร์ชวลไลเซชันฮาร์ดแวร์ ดังนั้นเพื่อที่จะได้รับข้อมูลเกี่ยวกับการเขียนของเกส จึงทำการป้องกันการเขียนเมมโมรีเพจของเกสที่มีการทำแชโดว์ไว้

5.3 **ไลฟ์ไมเกรชัน** (Live migration)

ไลฟ์ไมเกรชัน คือความสามารถในการย้ายเวอร์ชวลแมชชีนจากเครื่องหนึ่งไปอีกเครื่องหนึ่งโดยไม่ขัดจังหวะการทำงานของเกสเกิน 30ms เครื่องมือนี้ทำให้เวอร์ชวลแมชชีนสามารถย้ายที่ไปที่เครื่องอื่นได้เพื่อความเหมาะสมของจำนวนงานที่โหลดในแต่ละเครื่องและความต้องการด้านประสิทธิภาพ ไลฟ์ไมเกรชันทำงานโดยการคัดลอกหน่วยความจำของเกสไปที่เครื่องเป้าหมาย โดยทำงานขนานไปกับการทำงานของเกสตามปกติ ถ้าเกสเพจถูกแก้ไขหลังจากที่ได้ทำการคัดลอกไปแล้ว จะต้องทำการคัดลอกอีกครั้ง เมื่อถึงจุดสิ้นสุด เควีเอ็มจะมีเครื่องมือเรียกว่า dirty page log ซึ่งทำให้ยูเซอร์สเปซมีอิมเมจของเพจที่ถูกแก้ไขตั้งแต่การเรียก

ครั้งสุดท้าย โดยการทำงานภายในเควีเอ็มจะแมพเกสเพจว่า read-only และจะแมพสำหรับ write หลังจากที่มีการเขียนครั้งแรก ซึ่งจะมีจุดเฉพาะที่ทำให้เกิดการอัพเดทอิมเมจด้วย

ไลฟ์ไมเกรชันมีขั้นตอนการทำงานแบบวนลูป นั่นคือ แต่ละครั้งที่ทำการคัดลอกหน่วยความจำไปอีกเครื่องหนึ่ง เกสจะสร้างหน่วยความจำที่จะต้องคัดลอกเพิ่มขึ้น ดังนั้นจึงต้องกำหนดเกณฑ์ที่จะหยุดกระบวนการนี้ นั่นคือ

- มีการคัดลอกสองรอบซึ่งไม่จำเป็นต้องติดกัน ที่มีจำนวนหน่วยความจำที่ถูกคัดลอกเพิ่มขึ้นเมื่อเปรียบเทียบกับรอบก่อนหน้านี้ หรือ
- ผ่านไป 30 รอบ

อัลกอริทึมของไลฟ์ไมเกรชันในเควีเอ็ม มีดังนี้ (Lublin, & Liguori, 2007, pp. 10-14)

1. ไมเกรชันรีเควสท์มาถึง เมื่อไมเกรทเวอร์ชวลแมชชีนจากเครื่อง A ไป B
    - สร้างคำสั่งภายนอกขึ้นมา
    - เชื่อมต่อและส่งเฮดเดอร์
    - จัดสรรทรัพยากรและเซ็ตอัพ
2. ย้ายหน่วยความจำ
    - ย้ายเมมโมรีเพจทั้งหมดไปก่อน (รอบแรก)
    - สำหรับทุกๆรอบถัดไป ย้าย dirty page ทั้งหมดของรอบก่อนหน้านั้นจนกว่าจะเป็นไปตามเงื่อนไขที่กำหนดให้หยุด
3. หยุดเวอร์ชวลแมชชีน
4. ย้ายสถานะของเวอร์ชวลแมชชีน
    - แต่ละดีไวซ์ย้ายสถานะของมัน
    - รวมทั้งย้าย dirty page จากรอบสุดท้ายด้วย
5. ให้เวอร์ชวลแมชชีนทำงานต่อได้
    - บนเครื่องปลายทาง (B) ถ้าการไมเกรทสำเร็จ และส่ง (บรอดคาส) อีเทอร์เน็ตแพกเกตเพื่อประกาศตำแหน่งใหม่ของมัน
    - บนเครื่องโลคอลโฮส (A) ถ้าการไมเกรทล้มเหลว

โปรโตคอลสิ้นสุดไมเกรชัน (สำหรับ tcp://) ซึ่งป้องกันกรณีที่เกสรันต่อทั้งสองเครื่อง มีอัลกอริทึมดังนี้ (Lublin, & Liguori, 2007, p. 21)

- เครื่อง A ย้ายสถานะไปที่เครื่อง B และรอ ACK
- เครื่อง B รับสถานะ ส่ง ACK และรอ GO

- เครื่อง A รับ ACK และส่ง GO
- เครื่อง B รับ GO และทำงานต่อ
- กรณีไทม์เอาท์ใดๆ (เมสเซจสูญหาย) การไมเกรทจะล้มเหลว ส่วนในกรณีที่แย่ที่สุดคือเมสเซจ GO สูญหาย เวอร์ชวลแมชชีนจะไม่รันบนเครื่องใดเลย เครื่อง A จะหยุด และเครื่อง B จะจบการทำงาน

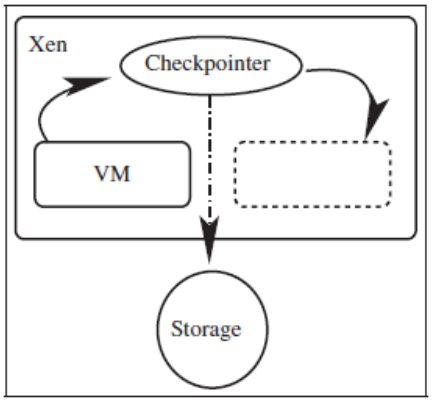
งานวิจัยที่เกี่ยวข้องกับการเช็คพอยน์เวอร์ชวลแมชชีนด้วยวิธีไลฟ์ไมเกรชัน

Ta-Shma, Laden, Ben-Yehuda และ Factor (2008, pp. 127-132) ได้ศึกษาเรื่องเวอร์ชวลแมชชีนไทม์ทราเวล (virtual machine time travel) ซึ่งเป็นวิธีการที่ทำให้สามารถย้อนสถานะของเวอร์ชวลแมชชีนให้กลับไปสู่สถานะก่อนหน้านี้ได้ ความสามารถนี้สามารถนำมาใช้ในการปรับปรุงความพร้อมใช้งาน (availability) ของเวอร์ชวลแมชชีนให้ดีขึ้นได้ โดยเสนอวิธีการเวอร์ชวลแมชชีนไทม์ทราเวลที่รวมความช่วยเหลือของที่เก็บข้อมูล CDP (Continuous Data Protection) เข้ากับการเช็คพอยน์เวอร์ชวลแมชชีนแบบไลฟ์ไมเกรชัน โดยวิธีการใหม่สำหรับ CDP จะทำให้สามารถย้อนสถานะของที่เก็บข้อมูลกลับไปสู่สถานะในอดีตได้อย่างมีประสิทธิภาพและเป็นไปได้ที่จะย้อนกลับการย้อนสถานะ ซึ่งทำการอิมพลีเมนต์กลไกการเช็คพอยน์แบบไลฟ์ไมเกรชันในเซน (Xen)

สถาปัตยกรรม CDP สำหรับการเช็คพอยน์นี้จะแยกเวอร์ชันปัจจุบันของส่วนที่เก็บข้อมูลที่เรียกว่า production device ออกจากเวอร์ชันอื่นๆที่ถูกเก็บอยู่ในส่วน repository device การแยกส่วนเช่นนี้จะช่วยให้สามารถอ่านข้อมูลได้อย่างรวดเร็วเนื่องจากไม่มีการเข้าถึงทางอ้อมใน production device โดยสิ่งที่จะต้องเสียไปคือข้อมูลจำเป็นที่จะต้องถูกเขียนหรือถูกคัดลอกไปที่ทั้ง production device และ repository device โดยบล็อกข้อมูลจะถูกคัดลอกไปที่ส่วน repository เฉพาะตอนที่มันกำลังจะถูกเขียนทับใน production device นั่นคือวิธีการของ copy-on-write หรือ COW

เมื่อผู้ใช้ต้องการย้อนระบบกลับไปสู่จุดใดจุดหนึ่งในเวลาก่อนหน้านี้ ส่วน production device จะต้องถูกอัพเดทเพื่อให้มีข้อมูลที่ถูกต้อง การคัดลอกเหล่านี้จะไม่บล็อกไอโอในระหว่างที่ทำงานนี้ การเขียนต่างๆจะถูกส่งไปที่ production device และการอ่านจะทำได้โดยตรงจากส่วน repository วิธีการนี้จะทำให้ระบบที่ทำการย้อนกลับดูเหมือนเกิดขึ้นในทันทีสำหรับผู้ใช้ ถึงแม้ว่าประสิทธิภาพจะลดลงบ้างในระหว่างที่ทำการคัดลอกข้อมูลอยู่เบื้องหลัง

ภาพที่ 2.11
การเช็คพอยน์โดยใช้โลคอลโฮสไลฟ์ไมเกรชัน



ที่มา: “Virtual machine time travel using continuous data protection and checkpointing,” โดย Ta-Shma, P., Laden, G., Ben-Yehuda, M., & Factor, M., 2008, *SIGOPS Oper. Syst. Rev.,* น. 131.

จากภาพที่ 2.11 การเช็คพอยน์อิมพลีเมนต์โดยการไมเกรทเวอร์ชวลแมชชีนไปที่ไฮเปอร์ไวเซอร์เดิม ในระหว่างการไมเกรทสำเนาของไมเกรชันบิตสตรีมจะถูกเขียนลงไฟล์ เช็คพอยน์เตอร์ในงานวิจัยนี้จะดักเส้นทางของไมเกรชันระหว่างเครื่องต้นทางและปลายทาง โดยทำตัวเหมือนเป็นเครื่องปลายทางให้กับเครื่องต้นทาง และเป็นเครื่องต้นทางให้กับเครื่องปลายทาง

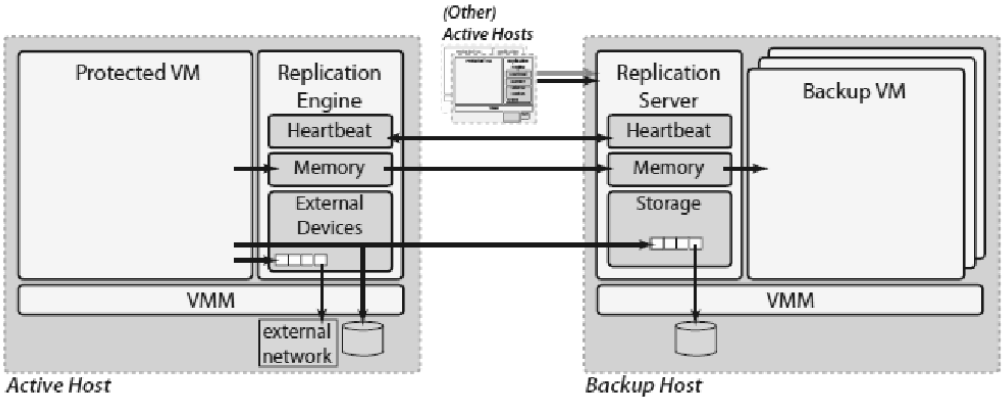
เมื่อเช็คพอยน์เตอร์เริ่มต้นทำงาน มันจะสร้างซ็อกเก็ตรอรับบนพอร์ตที่รู้จักกันดีและรอไมเกรชันรีเควสท์ ในการสร้างเวอร์ชวลแมชชีนเช็คพอยน์ ไลฟ์ไมเกรชันของเวอร์ชวลแมชชีนจะถูกส่งไปที่พอร์ตนั้นบนเครื่องเดิมที่เวอร์ชวลแมชชีนกำลังทำงานอยู่ เช็คพอยน์เตอร์จะทำไมเกรชันโปรโตคอลที่มีตัวเริ่มต้น และจะเริ่มต้นการไมเกรทกลับไปที่เซนบนเครื่องเดิม ขณะที่ข้อมูลการไมเกรทเริ่มไหลผ่านไป เช็คพอยน์เตอร์จะดักจับมัน เขียนสถานะชั่วคราวของเวอร์ชวลแมชชีนลงในดิสก์พร้อมกับส่งต่อกลับไปที่เซน ลักษณะเด่นของไลฟ์ไมเกรชันคือเวอร์ชวลแมชชีนยังคงรันต่อไปในขณะที่สถานะส่วนใหญ่ของมันถูกเคลื่อนย้ายไป การทำงานของมันจะถูกหยุดเพียงเล็กน้อยเท่านั้นเพื่อให้ได้มุมมองที่สอดคล้องของเฟรมที่มีการแก้ไขของหน่วยความจำสุดท้ายและสถานะของซีพียูรีจิสเตอร์

วิธีการเช็คพอยน์ที่ใช้เทคนิคไลฟ์ไมเกรชันของเซนในงานวิจัยนี้เป็นวิธีที่รวดเร็ว ซ่อนรายละเอียด และไม่รบกวนการทำงาน ซึ่งเซนได้เพิ่มวิธีการเช็คพอยน์นี้เข้าไปในส่วนของ xen-unstable ผ่านทางคำสั่ง “xm save –checkpoint”

Cully และคณะ (2008, pp. 1-14) ได้สร้างระบบซอฟต์แวร์ที่ทำให้ระบบปฏิบัติการและแอพพลิเคชันสามารถทำงานได้ต่อเนื่องตลอดเวลา (high availability) เรียกว่าระบบ Remus โดยระบบที่กำลังรันอยู่จะสามารถทำงานต่อบนโฮสอีกเครื่องหนึ่งได้เมื่อเกิดความผิดพลาดโดยมีดาวน์ไทม์เพียงแค่ไม่กี่วินาที วิธีการนี้อาศัยความสามารถของเวอร์ชวลไลเซชันในการไมเกรทเวอร์ชวลแมชชีนที่กำลังรันอยู่ระหว่างฟิสิคัลโฮส นั่นคือ ซอฟต์แวร์ที่ต้องการปกป้องจากความผิดพลาดจะรันอยู่ในเวอร์ชวลแมชชีน และขยายเทคนิคเพื่อทำสำเนาสแนปช็อตของระบบปฏิบัติการที่รันอยู่ทั้งหมดระหว่างฟิสิคัลแมชชีนสองเครื่อง นั่นคือ สถานะที่มีการเปลี่ยนแปลงจะถูกส่งไปที่โฮสสำรอง ซึ่งการทำสำเนาสามารถทำได้บ่อยมากถึงทุกๆ 25ms

ภาพที่ 2.12

สถาปัตยกรรมระดับสูงของ Remus



ที่มา: “Remus: High availability via asynchronous virtual machine replication,” โดย Cully, B., Lefebvre, G., Meyer, D. T., Feeley, M., Hutchinson, N. C., & Warfield, A., 2008, *USENIX Symposium on Networked Systems Design and Implementation (NSDI),* น. 4.

ภาพที่ 2.12 แสดงสถาปัตยกรรมของ Remus เริ่มด้วยการป้องกันเครื่องให้อยู่ภายในเวอร์ชวลแมชชีน การอิมพลีเมนต์ทำบนเซนเวอร์ชวลแมชชีนมอนิเตอร์ และขยายไลฟ์ไมเกรชันของเซนให้มีการทำเช็คพอยน์ Remus จะส่งเช็คพอยน์ของเวอร์ชวลแมชชีนที่กำลังทำงานไปที่ฟิสิคัล

โฮสสำรอง บนเครื่องสำรอง อิมเมจของเวอร์ชวลแมชชีนจะอยู่ในหน่วยความจำและอาจเริ่มต้นการเอ็กซิคิวต์ทันทีถ้าตรวจพบความผิดพลาดของระบบที่กำลังทำงาน ขั้นตอนการเช็คพอยน์มีดังนี้ (1) หยุดเวอร์ชวลแมชชีนที่กำลังรันอยู่และคัดลอกสถานะที่ถูกเปลี่ยนแปลงลงในบัฟเฟอร์ ขั้นตอนนี้คือขั้น stop-and-copy ของไลฟ์ไมเกรชันที่มีการปรับปรุงโดยใช้การทำไปป์ไลน์เช็คพอยน์โอเปอเรชันเพื่อให้สามารถเช็คพอยน์ได้หลายๆครั้ง ด้วยการเปลี่ยนแปลงสถานะที่ถูกเก็บไว้ในบัฟเฟอร์ เวอร์ชวลแมชชีนจะสามารถกลับมาทำงานต่อได้ (2) สถานะในบัฟเฟอร์จะถูกส่งและเก็บไว้ในหน่วยความจำบนโฮสสำรอง (3) ทันทีที่เครื่องปลายทางได้รับสถานะที่สมบูรณ์แล้ว มันจะส่งการยืนยันเช็คพอยน์ไปที่เครื่องเดิม และ (4) ปล่อยข้อมูลที่ถูกเก็บอยู่ในเน็ตเวิร์คบัฟเฟอร์

การเช็คพอยน์ถูกอิมพลีเมนต์บนกลไกสำหรับการทำไลฟ์ไมเกรชันของเซน ในระหว่างการไมเกรท การเขียนไปที่หน่วยความจำจะถูกดักจับและเพจที่ dirty จะถูกคัดลอกไปที่ตำแหน่งใหม่ซ้ำหลายๆรอบ และเมื่อถึงเวลาสิ้นสุดการคัดลอก เกสจะถูกหยุดและหน่วยความจำที่เหลือจะถูกคัดลอกไปพร้อมกับสถานะของซีพียู

Remus อิมพลีเมนต์การเช็คพอยน์เหมือนกับการเอ็กซิคิวต์ขั้นตอนสุดท้ายของไลฟ์ไมเกรชัน (stop-and-copy) ซ้ำๆ นั่นคือ แต่ละรอบ เกสจะถูกหยุดในขณะที่หน่วยความจำที่ถูกเปลี่ยนแปลงและสถานะของซีพียูจะถูกคัดลอกไปที่บัฟเฟอร์ จากนั้น เกสจะทำการเอ็กซิคิวต์ต่อบนเครื่องเดิมแทนที่จะทำบนเครื่องปลายทาง

การอิมพลีเมนต์เครื่องมือเช็คพอยน์ในเซนเหล่านี้อาศัยการทำงานของไลฟ์ไมเกรชันเนื่องจากเซนยังไม่มีการสนับสนุนที่เป็นส่วนสำคัญสำหรับการเช็คพอยน์เวอร์ชวลแมชชีน การบันทึกไฟล์สถานะจึงขึ้นอยู่กับข้อมูลที่ถูกไมเกรทระหว่างเครื่อง ในขณะที่โปรโตคอลไลฟ์ไมเกรชันในเควีเอ็มสามารถใช้ในการไมเกรทสถานะลงไฟล์ได้โดยตรง สำหรับงานวิจัยนี้ นอกจากการอิมพลีเมนต์วิธีการเช็คพอยน์ให้ใช้ประโยชน์จากโปรโตคอลไลฟ์ไมเกรชันที่มีอยู่แล้ว ยังได้ใช้การอิมพลีเมนต์แบบเทรดบนเครื่องมัลติคอร์เพื่อช่วยเพิ่มประสิทธิภาพของการเช็คพอยน์ด้วย ซึ่งเป็นการออกแบบที่ไม่พบในงานวิจัยอื่น