บทที่ 2

งานวิจัยและทฤษฎีที่เกี่ยวข้อง

2.1 งานวิจัยที่เกี่ยวข้อง

กระบวนการทนทานต่อความผิดพร่องสามารถจำแนกออกเป็น 2 วิธีการใหญ่ ๆ คือ การใช้ฮาร์ดแวร์สำรอง (hardware redundancy) และการใช้ซอฟต์แวร์ (software approach) สำหรับการใช้ฮาร์ดแวร์สำรองนั้นเป็นวิธีการที่ตรงไปตรงมาคือ การใช้ฮาร์ดแวร์อีกชุดหนึ่งทำงานขนานไปกับระบบหลักโดยปรับตั้งให้ฮาร์ดแวร์ทั้งสองชุดมีสำเนาข้อมูลเหมือนกัน และเมื่อเกิดความผิดพลาดจนระบบหลักไม่สามารถปฏิบัติงานต่อไปได้ระบบสำรองจะถูกปรับให้เข้ามาทำหน้าที่แทนระบบหลักโดยที่ผู้ใช้จะไม่สามารถทราบได้ว่ามีการสับเปลี่ยนระบบเกิดขึ้น ข้อดีของวิธีการนี้คือ ให้สมรรถนะการทำงานที่ดีเยี่ยม แต่ข้อด้อยคือ มีภาระค่าใช้จ่ายสูงจนยากต่อการนำไปปฏิบัติจริง จึงนำไปสู่การคิดค้นวิธีการเพิ่มการทนทานต่อความผิดพร่องผ่านซอฟต์แวร์ข้อดีของวิธีการนี้คือ ไม่จำเป็นต้องใช้ฮาร์ดแวร์สำรองทำให้มีค่าใช้จ่ายต่ำ แต่ข้อด้อยคือ ความยากในการสร้างขึ้นใช้งาน และจากการทบทวนงานประพันธ์ต่าง ๆ ผู้วิจัยสามารถจำแนกการเช็คพอยต์ด้วยซอฟต์แวร์ออกเป็นระดับโพรเซส ระดับเคอร์เนล และระดับไฮเปอร์ไวเซอร์ สำหรับงานวิจัยชิ้นนี้เลือกพัฒนาเช็คพอยต์/รีโคเวอร์รีโพรโทคอลในระดับของไฮเปอร์ไวเซอร์ซึ่งจะกล่าวในรายละเอียดภายหลัง

ถึงแม้ว่าการแก้ไขซอร์สโค๊ด (source code) เป็นวิธีการที่ตรงไปตรงมาสำหรับการทำเช็คพอยต์โปรแกรมประยุกต์ แต่ก็เห็นได้ชัดเจนว่ามันสร้างภาระให้กับโปรแกรมเมอร์เป็นอย่างมาก และมีแนวโน้มที่จะเกิดจุดบกพร่อง (bug) ต่าง ๆ มากมาย ดังนั้นจึงมีการศึกษาเทคนิคต่าง ๆ ขึ้นมาเพื่อบรรเทาภาระดังกล่าวลง งานลิบซีเคพีที (Libckpt) (Plank, Beck, Kingsley, & Li, 1995, pp. 213-223) เป็นเช็คพอยต์ไลบรารีที่อนุญาตให้โปรแกรมประยุกต์สามารถเช็คพอยต์ได้โดยเพียงแค่เพิ่มไลบรารีลงไปบนซอร์สโค๊ดและรีคอมไพล์ (recompile) โปรแกรมเข้ากับ libckpt.a ซึ่งจากจุดเริ่มต้นนี้เองจึงนำไปสู่การปรับปรุงวิธีการสร้างเช็คพอยต์ให้มีความโปร่งใสมากขึ้น คอนดอร์ (Condor) (Litzkow & Solomon, 1999, pp. 163-164) สามารถทำการเช็คพอยต์ได้โดยการเชื่อมโยง (relink) อ๊อบเจคไฟล์ (object file) ของโปรแกรมเข้ากับ condor_syscall_lib.a และไม่มีความจำเป็นต้องแก้ไขซอร์สโค๊ด ซึ่งแน่นอนว่าวิธีการจัดเก็บสเตทของโปรแกรมประยุกต์

ในระดับของยูสเซอร์สเปซ (user space) มีข้อจำกัดในการเข้าถึงข้อมูลหลายอย่างทำให้มีความพยายามคิดค้นเทคนิคในการสร้างเช็คพอยต์ในระดับเคอร์เนลสเปซ (kernel space) จนได้ผลลัพธ์ออกมาคือ บีแอลซีอาร์ (BLCR) (Duell, 2002, p. 1) และ แครค (CRAK) (Zhong & Nieh, 2001, p. 1) ซึ่งมีความโปร่งใสต่อโปรแกรมประยุกต์โดยแท้จริงบนเคอร์เนลที่ได้รับการออกแบบมาเป็นพิเศษ

แต่อย่างไรก็ตามการเช็คพอยต์ก็สามารถทำได้ด้วยไฮเปอร์ไวเซอร์ซึ่งมีความโดดเด่นกว่าวิธีทั้งหมดที่กล่าวมาคือ ไม่ต้องการดัดแปลงโปรแกรมประยุกต์หรือเคอร์เนลของระบบปฏิบัติการใด ๆ ซึ่งมีความโปร่งใสต่อโปรแกรมเมอร์โดยแท้จริง เมื่อพิจารณาไฮเปอร์ไวเซอร์ต่าง ๆ ในปัจจุบันนี้สามารถจำแนกออกเป็นสามกลุ่มใหญ่คือ ฟูลเวอร์ชวลไลเซชัน (full virtualization) พาราเวอร์ชวลไลเซชัน (paravirtualization) และโอเอสเลเวลเวอร์ชวลไลเซชัน (os-level virtualization) คีมู (QEMU) (Bellard, 2005, pp. 41-46) เป็นไฮเปอร์ไวเซอร์แบบฟูลเวอร์ชวลไลเซชันที่มีความสามารถติดตั้งระบบปฏิบัติการและซอฟต์แวร์ที่มีอยู่ได้โดยไม่จำเป็นต้องทำการดัดแปลงใด ๆ และจากการวัดสมรรถนะการประมวลผลแสดงให้เห็นว่าไฮเปอร์ไวเซอร์ชนิดนี้มีโอเวอร์เฮดในการทำงานสูงทำให้เกิดงานวิจัยต่าง ๆ ที่มุ่งเป้าเพื่อเพิ่มสมรรถนะให้แก่ไฮเปอร์ไวเซอร์ขึ้นจำนวนหนึ่ง เคคีมู (KQEMU) เป็นเคอร์เนลโมดูลสำหรับเพิ่มสมรรถนะการประมวลผลของคีมูมันสามารถทำงานกับคีมูได้ทันที เซน (Xen) (Dragovic et al., 2003, pp. 164-177) มีมุมมองว่าการจำลองสถาปัตยกรรมของเครื่องเสมือนให้เหมือนกับเครื่องจริงนั้นก่อให้เกิดโอเวอร์เฮดโดยไม่จำเป็น พวกเขาจึงมีความพยายามปรับเปลี่ยนมันเพื่อให้ได้สถาปัตยกรรมใหม่ที่ออกแบบมาสำหรับเครื่องเสมือนโดยเฉพาะจึงทำให้เกิดพาราเวอร์ชวลไลเซชันขึ้นมา และจากความพยายามนี้เองทำให้ได้เครื่องเสมือนที่มีประสิทธิภาพสูงขึ้นแต่ก็ต้องแลกมาด้วยกับเงื่อนไขที่ว่าจำเป็นต้องดัดแปลงเกสโอเอสเสียก่อนจึงจะนำมาใช้งานกับเซนได้ ในทำนองเดียวกันโอเพนวีซี (OpenVZ) (Kolyshkin, 2007, pp. 3-6) มีแนวคิดว่าควรจะให้เกสโอเอสและโฮสโอเอสใช้เคอร์เนลร่วมกันทำให้ไม่เกิดโอเวอร์เฮดในระดับเกสเคอร์เนลซึ่งเทคนิคนี้นอกจากมีความจำเป็นต้องดัดแปลเกสโอเอสแล้วยังต้องดัดแปลงโฮสเคอร์เนลเพื่อให้สามารถจัดเตรียมเวอร์ชวลไลเซชันได้ด้วย เพราะเทคโนโลยีเวอร์ชวลไลเซชันกำลังได้รับความนิยมเพิ่มขึ้นเรื่อย ๆ ทำให้ผู้ผลิตโพรเซสเซอร์มีการคิดค้นสถาปัตยกรรมใหม่เพื่อสนับสนุนเวอร์ชวลไลเซชันโดยเฉพาะเช่น อินเทล-วีที (Intel-VT) และ เอเอ็ม-ดีวี (AMD-V) ซึ่งเควีเอ็ม (KVM) (Quramnet, 2006, p. 2) เป็นตัวอย่างหนึ่งของเครื่องเสมือนแบบฟูลเวอร์ชวลไลเซชันที่ใช้ประโยชน์จากสถาปัตยกรรมดังกล่าว

ปัญหาเรื่องการทนทานต่อความผิดพร่องถูกทำให้ซับซ้อนมากขึ้นบนโปรแกรมประยุกต์แบบขนาน (parallel application) และเทคนิคหนึ่งที่ได้รับความนิยมสำหรับโปรแกรมชนิดนี้คือการทำโคออดิเนตเช็คพอยต์บนเอ็มพีไอ (MPI: Message Passing Interface) โคเช็ค(CoCheck) (Stellner, 1996, pp. 526-531) เป็นความพยายามเช็คพอยต์โปรแกรมประยุกต์แบบขนานโดยมุ่งเป้าหมายให้ไม่ยึดติดกับเอ็มพีไอที่ทำงานอยู่ แต่การใช้งานจำเป็นต้องแทนที่ฟังก์ชันบางตัวด้วยฟังก์ชันใหม่ของโคเช็คทำให้วิธีการนี้สร้างภาระระดับหนึ่งให้แก่โปรแกรมเมอร์ จากงานนี้แสดงให้เห็นว่าความพยายามไม่ยึดติดกับเอ็มพีไออาจจะไม่ใช่ทางเลือกที่ดีนักในการบรรลุความโปร่งใสงานต่อมาเช่น แลม/เอ็มพีไอ (LAM/MPI) (Sankaran et al., 2003, pp. 479-493) และเอ็มพีไอซีเอชวี (MPICHV) (Coti et al., 2006, pp. 1-13) เลือกที่จะสร้างเช็คพอยต์บนเอ็มพีไอของตัวเองเพื่อให้บรรลุคุณสมบัติความโปร่งใส จากการทบทวนงานวิจัยต่าง ๆ ทำให้ผู้วิจัยเชื่อว่างานเช็คพอยต์โปรแกรมประยุกต์แบบขนานในระดับโพรเซสนั้นได้รับการปรับปรุงและพัฒนามาเป็นอย่างดีแล้ว แต่เมื่อมองถึงในระดับไฮเปอร์ไวเซอร์นั้นผู้วิจัยพบว่ายังเป็นเรื่องที่ใหม่และท้าทาย ตัวอย่างงานวิจัยในระดับนี้คือ จีแอลพี (GLP) (Scarpazza et al., 2007, pp. 74-83) ซึ่งเลือกใช้เซนเป็นไฮเปอร์ไวเซอร์ งานชิ้นนี้มีระดับความโปร่งใสที่น้อยเพราะจากการออกแบบแสดงให้เห็นว่ามันต้องการใช้โกลบอลโคออดิเนเตอร์ (global coordinator) หลาย ๆ ตัวที่ทำงานบนทั้งโฮสโอเอสและเกสโอเอสเพื่อประสานงานเช็คพอยต์ผ่านเครือข่ายอินฟินิแบนด์ (infiniband) ไปจนถึงการยึดติดกับเออาร์เอ็มซีไอ (ARMCI) ไลบรารี่ของมัน ส่วนไวโอลิน (VIOLIN) (Kangarlou, Xu, Ruth, Eugster, 2007, pp. 1-6) เป็นงานที่สามารถเช็คพอยต์/รีโคเวอร์รีในระดับไฮเปอร์ไวเซอร์และเป็นอิสระต่อโอเอสและโปรแกรมประยุกต์ต่าง ๆ แต่งานนี้เกิดปัญหาเฟรมข้อมูลสูญหายและแอพพลิเคชันไทม์เอาต์ทำให้มีโอเวอร์เฮดของการจัดส่งเฟรมข้อมูลซ้ำเป็นจำนวนมาก ดังนั้นเป้าหมายหลักของวิทยานิพนธ์ชิ้นนี้คือ การออกแบบวิธีการเช็คพอยต์ที่โปร่งใสต่อโปรแกรมประยุกต์และเกสโอเอสในระดับของไฮเปอร์ไวเซอร์หรืออีกนัยหนึ่งคือ กระบวนการทั้งหมดถูกกระทำบนไฮเปอร์ไวเซอร์เพียงอย่างเดียว รวมทั้งคาดหวังว่างานชิ้นนี้ควรเป็นอิสระจากเอ็มพีไอค่ายต่าง ๆ เนื่องมาจากการทำงานในระดับไฮเปอร์ไวเซอร์ ไปจนถึงแก้ไขปัญหาที่พบในไวโอลิน

2.2 โกลบอลสเตท

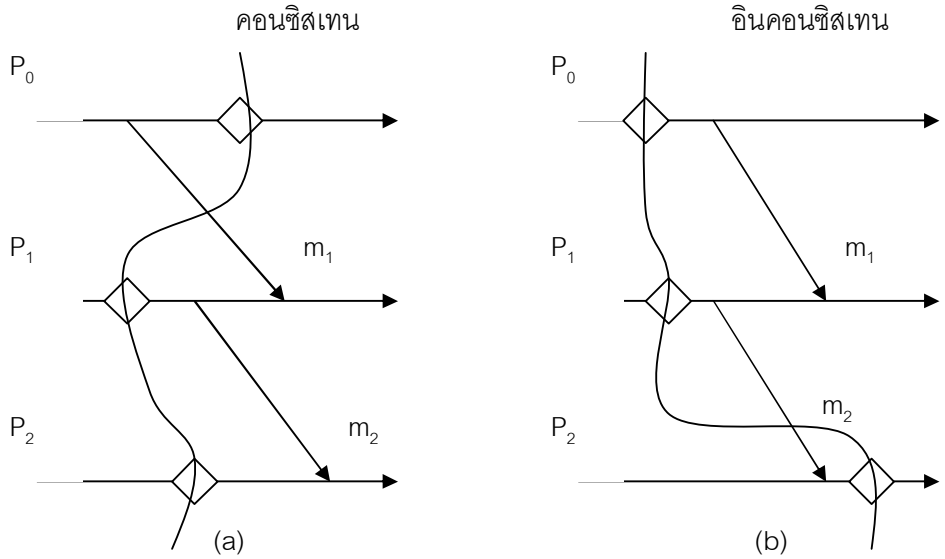
โดยภาพรวมนั้นการเช็คพอยต์คือ การจัดเก็บสเตทของโปรแกรมประยุกต์ลงบนอุปกรณ์เก็บข้อมูลที่มีเสถียรภาพ (stable storage) เพื่อป้องกันสเตทสูญหายเมื่อเกิดความ

ผิดพลาดและข้อมูลสเตทนี้เองจะถูกใช้สำหรับรีโคเวอร์รีโปรแกรมในภายหลัง ซึ่งการจัดเก็บสเตทนั้นถ้าหากเป็นโปรแกรมที่ทำงานอยู่บนเครื่องเพียงเครื่องเดียวก็สามารถทำได้ง่ายและตรงไปตรงมา แต่จะยากและซับซ้อนมากขึ้นในโปรแกรมประยุกต์แบบขนานที่มีการแตกออกเป็นโพรเซสย่อยและกระจัดกระจายไปบนเครื่องต่าง ๆ รวมไปถึงมีเรื่องเครือข่ายเข้ามาเกี่ยวข้อง

โกลบอลสเตท (global state) หมายถึง สเตทของโพรเซสต่าง ๆ ในโปรแกรมประยุกต์แบบขนานร่วมกับสเตทของช่องสื่อสาร ซึ่งจากความซับซ้อนของตัวโปรแกรมเองจึงทำให้มีโอกาสที่จะเกิดโกลบอทสเตทซึ่งใช้งานไม่ได้และสเตทชนิดนี้ถูกเรียกว่า อินคอนซิสเทนโกลบอลสเตท (inconsistent global state) ส่วนสเตทที่ใช้งานได้จะถูกเรียกว่า คอนซิสเทนโกลบอลสเตท (consistent global state) จากการศึกษาของ (Chandy & Lamport, 1985, p. 69) ได้ชี้ให้เห็นคุณสมบัติสำคัญที่ใช้แยกแยะโกลบอลสเตทว่าเป็นชนิดใดดังนี้ ถ้าภายในโกลบอลสเตทบอกว่าโพรเซสหนึ่งได้รับเมสเซส ในส่วนข้อมูลของผู้ส่งเมสเซสก็ต้องบันทึกเอาไว้ว่าเมสเซสถูกส่งออกไปแล้วจึงจะถือว่าเสทนั้นเป็นโกลบอลสเตท จากภาพที่ 2.1(a) เป็นคอนซิสเทนโกลบอลสเตทเพราะมีการบันทึกว่าเมสเซส m_1 ที่ถูกส่งออกไปแต่ยังไม่ถูกรับ 2.1(b) เป็นอินคอนซิสเทนโกลบอลสเตทเพราะว่าโพรเซส P_2 แสดงให้เห็นว่าได้รับ m_2 แต่สเตทของโพรเซส P_1 บอกว่าไม่ได้ส่ง m_2 ออกไป

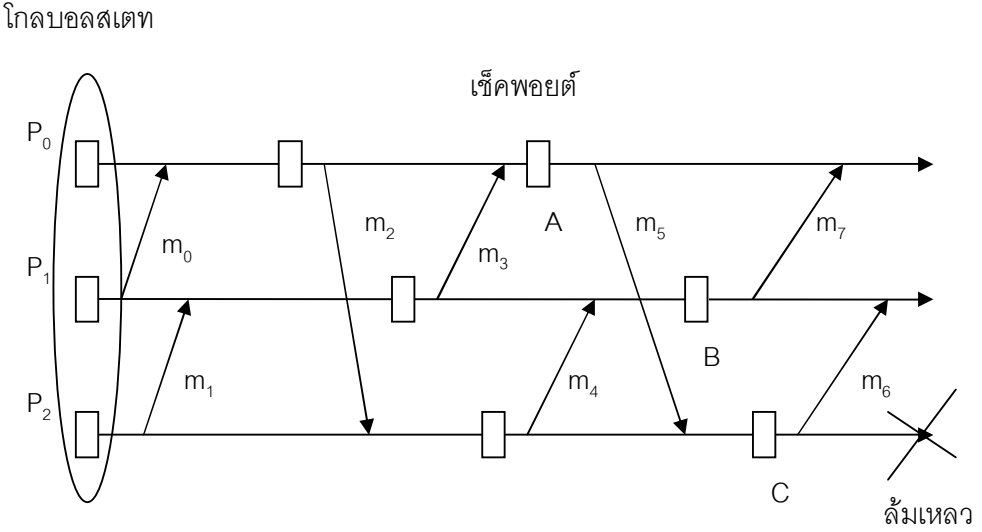
ภาพที่ 2.1

ตัวอย่างคอนซิสเทนโกลบอลสเตทและอินคอนซิสเทนโกลบอลสเตท



ภาพที่ 2.2

โรลแบ็คพรอพเพเกชัน โกลบอลสเตท และโดมิโนเอฟเฟค

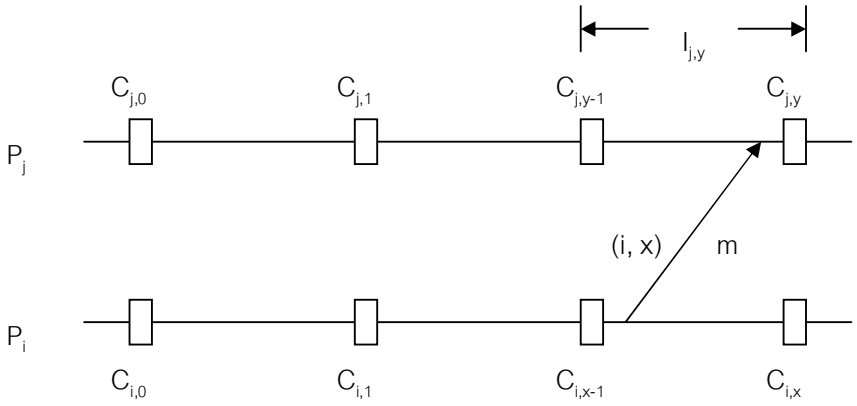


การค้นหาคอนซิสเทนโกลบอลสเตทหรืออีกนัยนึงคือ การค้นหาการขึ้นต่อกันระหว่างโปรเซส (interprocess dependency) เป็นสิ่งจำเป็นเพื่อเพิ่มโอกาสในการรีโคเวอร์รีให้สูงมากขึ้นและลดปริมาณงานเช็คพอยต์ซ้ำซ้อนเพื่อประหยัดพื้นที่เก็บข้อมูล คุณลักษณะการขึ้นต่อกันระหว่างโปรเซส (inter-process dependency) เป็นปัญหาสำคัญที่ทำให้โพรโทคอลซับซ้อน เพราะคุณลักษณะนี้อาจบังคับให้โพรเซสที่ยังทำงานได้และมีความเกี่ยวข้องกับโพรเซสที่ล้มเหลวต้องถูกโรลแบ็ค (rollback) ซึ่งเหตุการณ์นี้มีชื่อว่าโรลแบ็คพรอพเพเกชัน (rollback propagation) ตัวอย่างสถานการณ์ของโรลแบ็คพรอพเพเกชันคือ สถานการณ์ที่ผู้ส่งเมสเซส A จำเป็นต้องโรลแบ็คไปสู่สเตทก่อนการส่งเมสเซส A ผลที่ตามมาคือ ผู้รับเมสเซส A ต้องโรลแบ็คไปสู่สเตทก่อนการรับเมสเซส A ไม่เช่นนั้นสเตทของผู้รับและผู้ส่งจะไม่สอดคล้องกัน โดยแสดงออกมาว่าเมสเซส A ถูกรับแล้วแต่ผู้ส่งยังไม่ได้ส่งเมสเซส A ออกมาซึ่งเป็นสิ่งที่เป็นไปไม่ได้ในการทำงานจริง นอกจากความไม่สอดคล้องเฉพาะโพรเซสคู่หนึ่งก็ยังมีความเป็นไปได้ที่โรลแบ็คพรอเพเกชันจะเกิดขึ้นอย่างต่อเนื่องจนเป็นผลให้ทุกโพรเซสต้องเริ่มทำงานใหม่ตั้งแต่ต้น ซึ่งเหตุการณ์ลักษณะดังกล่าวถูกเรียกว่าโดมิโนเอฟเฟค (domino effect) (Randell , 1975, p. 8) ภาพที่ 2.2 แสดงภาพการทำงานของสามโพรเซสซึ่งแต่ละโพรเซสทำเช็คพอยต์อย่างอิสระดังแสดงโดยแท่งสีเหลี่ยมแต่ละโพรเซสเริ่มต้นการทำงานโดยสร้างเช็คพอยต์หนึ่งครั้ง สมมติว่าโพรเซส P_2 ผิดพลาดและโรลแบ็คไปสู่เช็คพอยต์ C จะส่งผลให้การส่งเมสเซส m_6 ไม่เกิดขึ้น ดังนั้น P_1 ต้องโรลแบ็คไปสู่

เช็คพอยต์ B เพื่อยกเลิกการรับเมสเซส m_6 ซึ่งจากตรงนี้ P_1 ต้องยกเลิกการส่งเมสเซส m_7 และบังคับให้ P_0 ต้องโรลแบ็ค และเหตุการณ์โรลแบ็คพรอพเพเกชันนี้จะดำเนินต่อไปเรื่อย ๆ จนทำให้เกิดโดมิโนเอฟเฟค เป็นผลให้ระบบต้องกลับไปสู่จุดเริ่มต้นของการคำนวณอันเนื่องมาจากความผิดพลาดของ P_2 เพียงโปรเซสเดียว

(Elnozahy, Alvisi, Wang, & Johnson, 2002, p. 7) ได้ทำการศึกษาเทคนิคต่าง ๆ ในการค้นหาคอนซิสเทนโกลบอลสเตทโดยจำแนกเทคนิคที่ใช้ออกเป็น อันโคออดิเนตเช็คพอยต์ (uncoordinated checkpoint) โคออดิเนตเช็คพอยต์ (coordinated checkpoint) และคอมมิวนิเคชันอินดิวส์เช็คพอยต์ (communication-induced checkpoint)

ภาพที่ 2.3
การขึ้นต่อกันของเช็คพอยต์ระหว่างโพรเซส



2.3 อันโคออดิเนตเช็คพอยต์

โดยภาพรวมอันโคออดิเนตเช็คพอยต์อนุญาตให้แต่ละโพรเซสตัดสินใจสร้างเช็คพอยต์อย่างอิสระ ข้อดีของวิธีนี้คือ สามารถกำหนดนโยบายการสร้างเช็คพอยต์ให้กับแต่ละโพรเซสแยกกันได้เช่น เช็คพอยต์เมื่อปริมาณข้อมูลที่ต้องบันทึกน้อยที่สุด เมื่อปริมาณข้อมูลที่ต้องบันทึกมากที่สุดหรือผ่านไประยะเวลาหนึ่งเป็นต้น แต่การสร้างเช็คพอยต์วิธีนี้มีข้อเสียคือ โอกาสเกิดโดมิโนเอฟเฟคและเกิดเช็คพอยต์ที่ไม่เป็นประโยชน์สูง เช็คพอยต์ที่ไม่เป็นประโยชน์คือเช็คพอยต์ที่ไม่มีทางเป็นส่วนหนึ่งของคอนซิสเทนโกลบอลสเตท จึงเป็นผลให้โพรเซสต้องเสียงาน

ไปโดยเปล่าประโยชน์ ข้อด้อยต่อมาคือ มีความต้องการเก็บเช็คพอยต์หลายอันเพราะไม่สามารถทราบได้ว่าเช็คพอยต์ใดจะเป็นส่วนหนึ่งของคอนซิสเทนโกลบอลสเตท

อันโคออดิเนตเช็คพอยต์ใช้การจดบันทึกการขึ้นต่อกันระหว่างโพรเซสลงไปบนเช็คพอยต์เพื่อใช้เป็นข้อมูลสำหรับคำนวณหาคอนซิสเทนโกลบอลสเตท โดย (Bhargava & Lian, 1988, pp. 3-12) เสนอวิธีบ่งชี้การขึ้นต่อกันไว้ดังนี้ กำหนดให้ $C_{i,x}$ คือ ดรรชนีเช็คพอยต์ที่ (checkpoint index) x ของโพรเซส P_i และ $I_{i,x}$ คือ ช่วงระหว่างเช็คพอยต์ (checkpoint interval) ของ $C_{i,x-1}$ กับ $C_{i,x}$ ดังภาพที่ 2.3 แสดงโพรเซส P_i ที่ช่วงเวลา $I_{i,x}$ ทำการแนบ (piggyback) คู่ลำดับ (I,x) ไปกับเมสเซส m ต่อมาโพรเซส P_j ได้รับ m ในช่วง $I_{j,y}$ ผลคือ โพรเซส P_j จะบันทึกว่ามีการขึ้นต่อกันจาก $I_{i,x}$ ถึง $I_{j,y}$ ลงไปในพื้นที่เก็บข้อมูลพร้อมกับสร้างเช็คพอยต์ที่ $C_{j,y}$

ในกรณีที่มีความผิดพลาดหนึ่งเกิดขึ้นโพรเซสที่มีหน้าที่กู้คืน (restoring process) จะเริ่มต้นกระบวนการรีโคเวอร์รีโดยแพร่กระจายรีเควสเมสเซสเพื่อขอข้อมูลการขึ้นต่อกันออกไป เมื่อโพรเซสของโปรแกรมประยุกต์ได้รับเมสเซสดังกล่าวมันจะหยุดการดำเนินงานปัจจุบันและตอบข้อมูลการขึ้นต่อกันที่มีอยู่กลับไป เมื่อได้รับข้อมูลการขึ้นต่อกันมาจากโพรเซสที่ถูกร้องขอทั้งหมดแล้วโพรเซสที่มีหน้าที่กู้คืนจะทำการคำนวณหาคอนซิสเทนโกลบอลสเตทและเผยแพร่รีโคเวอร์รีเมสเซสที่บรรจุข้อมูลเช็คพอยต์ที่จะให้โพรเซสอ่านกลับมา เมื่อโพรเซสได้รับข้อมูลรีโคเวอร์รีและตรวจพบว่าสถานะปัจจุบันตรงกับข้อมูลเช็คพอยต์อยู่แล้วโพรเซสนั้นจะสามารถทำงานต่อได้ทันที แต่ถ้าไม่ใช่โพรเซสต้องโรลแบ็คไปยังเช็คพอยต์ที่ถูกระบุในเมสเซส สำหรับวิธีการค้นหาคอนซิสเทนโกลบอลสเตทสองวิธีหลักคือ โรลแบ็คดีเพนเดนซีกราฟ (rollback dependency graph) และเช็คพอยต์กราฟ (checkpoint graph)

2.3.1 **โรลแบ็คดีเพนเดนซีกราฟ**

วิธีการหนึ่งสำหรับค้นหาคอนซิสเทนโกลบอลสเตทคือ โรลแบ็คดีเพนเดนซีกราฟ (Bhargava & Lian, 1988, pp. 3-12) ซึ่งกำหนดให้แต่ละโหนดของกราฟแทนเช็คพอยต์หนึ่งและมีเส้นเชื่อมระบุทิศทาง (directed edge) จาก $C_{i,x}$ ไป $C_{j,y}$ ถ้ากรณีใดต่อไปนี้เป็นจริง

(1) $i \neq j$, และเมสเซส m ถูกส่งจาก $C_{i,x}$ ไป $C_{j,y}$ หรือ

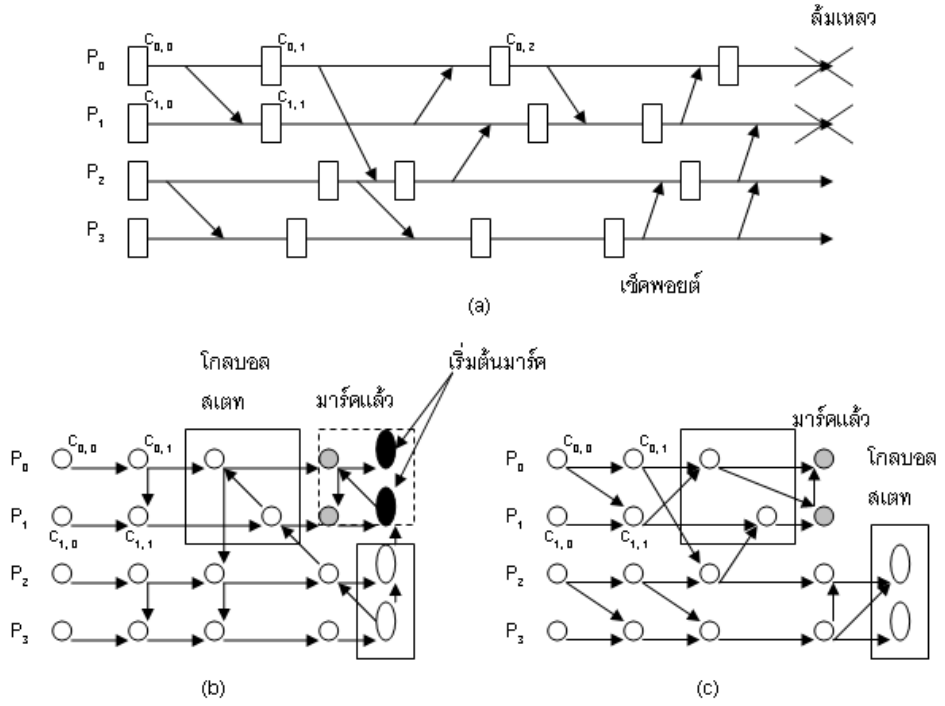
(2) $i = j$ และ $y = x + 1$

สาเหตุที่วิธีนี้มีชื่อว่าโรลแบ็คดีเพนเดนซีกราฟมาจากการสังเกตที่พบว่า ถ้ามีเส้นเชื่อมหนึ่งจาก $C_{i,x}$ ไป $C_{j,y}$ และเกิดความผิดพลาดที่บังคับให้ $C_{i,x}$ ต้องโรลแบ็คผลคือ $C_{j,y}$ ต้องถูกโรลแบ็คด้วย

สมมติให้มีโปรเซส P_0, P_1, P_2, และ P_3 มีการแลกเปลี่ยนเมสเซสไปจนกระทั่ง P_0 และ P_1 เกิดความผิดพลาดดังภาพที่ 2.4(a) กระบวนการกู้ระบบของโรลแบ็คดีเพนเดนซีกราฟจะเริ่มต้นขึ้นโดยการทำเครื่องหมายตรงตำแหน่งที่เกิดความผิดพลาด (วงรีสีดำ) จากนั้นวิเคราะห์ความเข้าถึงได้ (reachability) โดยเริ่มจากตำแหน่งที่ถูกทำเครื่องหมาย จากนั้นกำหนดให้โหนดล่าสุดที่ไม่ถูกทำเครื่องหมายเป็นส่วนหนึ่งของโกลบอลสเตทดังภาพที่ 2.4(b) สุดท้ายให้โพรเซสทั้งหมดโรลแบ็คไปยังเช็คพอยต์ดังกล่าว

ภาพที่ 2.4

(a) การส่งเมสเซส, (b) โรลแบ็คดีเพนเดนซีกราฟ, (c) เช็คพอยต์กราฟ



2.3.2 เช็คพอยต์กราฟ

วิธีการที่สองสำหรับการค้นหาคอนซิสเทนโกลบอลสเตทคือ เช็คพอยต์กราฟ (Wang, 1993) วิธีนี้เก็บการขึ้นต่อกันแตกต่างจากโรลแบ็คดีเพนเดนซีกราฟดังนี้ เมื่อเมสเซสถูกส่งจาก $C_{i,x}$

และถูกรับใน $C_{j,y}$ ให้เขียนเส้นเชื่อมระบุทิศทางมาจาก $C_{i,x-1}$ ไปสู่ $C_{j,y}$ ดังแสดงในภาพที่ 2.4(c) และวิธีการคำนวณคอนซิสเทนโกลบอลสเตทของเช็คพอยต์กราฟเป็นดังนี้ เริ่มต้นให้ตัดโหนดที่ล้มเหลวของโพรเซสออก จากนั้นใช้ขั้นตอนวิธีโรลแบ็คพรอพเพเกชัน (rollback propagation algorithm) กับเช็คพอยต์กราฟเพื่อทำให้โพรเซสทั้งหมดจะถูกโรลแบ็คไปสู่โกลบอลสเตทที่เป็นปัจจุบันที่สุดดังภาพ

จากรายละเอียดที่ผ่านมาทั้งหมดแสดงให้เห็นอย่างชัดเจนว่าการทำเช็คพอยต์แบบไม่ประสานงานก่อให้เกิดเช็คพอยต์ที่ไม่เป็นส่วนหนึ่งของโกลบอลสเตทเป็นจำนวนมาก ดังนั้นจึงจำเป็นต้องทำการเก็บขยะ (garbage collection) ซึ่งเป็นการนำพื้นที่กลับมาใช้ประโยชน์ใหม่ ด้วยเทคนิคของโรลแบ็คดีเพนเดนซีกราฟและเช็คพอยต์กราฟทำให้เก็บขยะได้ง่ายเพราะว่าเมื่อค้นพบโกลบอลสเตทที่เป็นปัจจุบันที่สุดแล้ว สามารถลบเช็คพอยต์ที่ไม่อยู่โกลบอลสเตทออกไปได้ทันทีงาน (Wang, Chung, & Fuchs, 1995, pp. 1-9) ได้แสดงให้เห็นว่าจำนวนเช็คพอยต์ที่มีประโยชน์สูงสุดที่ต้องถูกจัดเก็บบนอุปกรณ์สำรองข้อมูลจะไม่มากไปกว่า (N(N+1)/2) สุดท้ายสิ่งที่ควรระลึกไว้เสมอคือ เทคนิคนี้มีโอกาสเกิดโดมิโนเอฟเฟคได้สูงที่สุดเมื่อเทียบกับสองเทคนิคที่เหลือ

2.4 โคออดิเนตเช็คพอยต์

เมื่อพิจารณาจากอันโคออดิเนตเช็คพอยต์แล้วจะเห็นได้ชัดเจนว่าปัญหาหนึ่งของมันคือ การค้นหาคอนซิสเทนโกลบอลสเตทภายหลังจากมีความผิดพลาดเกิดขึ้น ทำให้มีโอกาสเกิดโดมิโนเอฟเฟคและเช็คพอยต์สูญเปล่าสูง ดังนั้นวิธีการของโคออดิเนตเช็คพอยต์ก็คือ การค้นหาคอนซิสเทนโกลบอลสเตทตั้งแต่แรก โดยบังคับให้ทุก ๆ โพรเซสต้องร่วมมือกันเพื่อสร้างคอนซิสเทนโกลบอลสเตทขึ้น วิธีนี้ทำให้การกู้ระบบง่ายกว่าการปล่อยให้สร้างเช็คพอยต์อย่างอิสระและไม่หวั่นไหวง่ายต่อโดมิโนเอฟเฟค เพราะวิธีนี้ให้การรับประกันว่าจะต้องมีคอนซิสเทนโกลบอลสเตทเสมอยิ่งไปกว่านั้นยังไม่ต้องทำการเก็บขยะ เนื่องจากโพรเซสจะทำนุบำรุงเฉพาะเช็คพอยต์ล่าสุดเท่านั้น แต่ข้อด้อยของวิธีนี้คือ ต้องสูญเสียเวลาจำนวนหนึ่งไปกับการประสานงานระหว่างโพรเซสเพื่อสร้างเช็คพอยต์

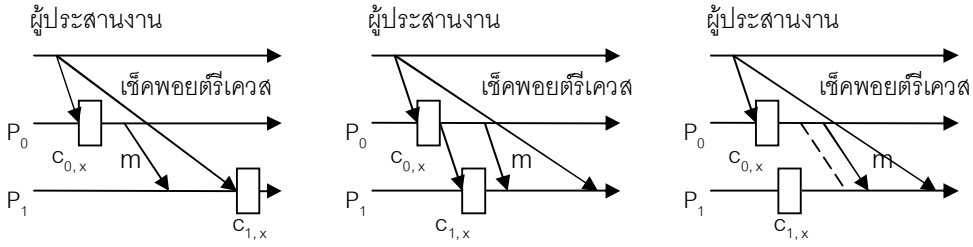
2.4.1 โคออดิเนตเช็คพอยต์แบบปิดกั้นการสื่อสาร

(Tamir & Sequim, 1984, pp. 32-44) เสนอให้ปิดกั้นการสื่อสารในระหว่างการประสานงานเพื่อสร้างคอนซิสเทนโกลบอลสเตท เทคนิคนี้เริ่มต้นโดยให้โพรเซสที่ทำหน้าที่

ประสานงานหรือผู้ประสานงาน (coordinator) เผยแพร่เช็คพอยต์รีเควส (checkpoint request) ไปสู่โพรเซสของโปรแกรมประยุกต์ทั้งหมดเพื่อแจ้งถึงการเริ่มต้นการสร้างเช็คพอยต์ เมื่อโพรเซสได้รับรีเควสการทำงานทุกอย่างจะถูกหยุดลงและทำการชะล้าง (flush) ช่องสื่อสารทั้งหมด จากนั้นจะสร้างเช็คพอยต์เบื้องต้น (tentative checkpoint) ต่อมาส่งการตอบรับ (acknowledgement) กลับสู่ผู้ประสานงาน เมื่อผู้ประสานงานได้รับการตอบรับจากโพรเซสครบทั้งหมดแล้วแล้วจะเผยแพร่เช็คพอยต์คอมมิท (checkpoint commit) เพื่อแจ้งให้โพรเซสอื่นทราบว่าการสร้างเช็คพอยต์เสร็จสมบูรณ์และทำการลบเช็คพอยต์เก่า จากนั้นจะปรับให้เช็คพอยต์เบื้องต้นกลายเป็นเช็คพอยต์ล่าสุดและโพรเซสจะได้รับอิสระในการทำงานตามปกติ

ภาพที่ 2.5

การสร้างเช็คพอยต์แบบโคออดิเนตโดยไม่ปิดกั้นช่องสื่อสาร (a) อินคอนซิสเทนโกลบอลสเตท (b) ช่องสื่อสารแบบไฟโฟ; (c) ช่องสื่อสารไม่ใช่ไฟโฟ (เส้นประแสดงถึงเช็คพอยต์รีเควสถูกแนบไปด้วย)



2.4.2 โคออดิเนตเช็คพอยต์แบบไม่ปิดกั้นการสื่อสาร

ปัญหาหลักของโคออดิเนตเช็คพอยต์คือ การป้องกันโพรเซสไม่ให้รับเมสเซสของแอพพลิเคชันที่สามารถทำให้เกิดอินคอนซิสเทนโกลบอลสเตท พิจารณาตัวอย่างในภาพที่ 2.5(a), ซึ่งเมสเซส m ถูกส่งโดย P_0 ภายหลังจากได้รับเช็คพอยต์รีเควสจากผู้ประสานงาน สำหรับตอนนี้สมมติว่า m ไปถึง P_1 ก่อนที่โพรเซสนี้จะได้รับเช็คพอยต์รีเควสจึงส่งผลให้เกิดอินคอนซิสเทนขึ้น เพราะว่าเช็คพอยต์ $C_{1,x}$ บอกว่ามีการรับเมสเซส m จาก P_0 แต่ $C_{0,x}$ กลับไม่ได้แสดงให้เห็นว่ามีการส่งเมสเซสจาก P_0 ซึ่งเหตุการณ์ลักษณะนี้สามารถแก้ไขได้ในกรณีที่ช่องทางการสื่อสารที่ใช้เป็นไฟโฟ โดยบังคับให้แต่ละโพรเซสส่งเช็คพอยต์รีเควสออกไปก่อนที่จะส่งเมสเซสแรกหลังการสร้างเช็คพอยต์ดังภาพที่ 2.5(b) ตัวอย่างหนึ่งของโคออดิเนเช็คพอยต์โพรโทคอลแบบไม่ปิดกั้นการ

สื่อสารคือ การกระจายแสนปช๊อต (Chandy & Lamport, 1985, pp. 63-75) โพรโทคอลนี้ใช้มาร์คเกอร์เป็นคำร้องขอให้สร้างเช็คพอยต์ โดยผู้ประสานงานจะเริ่มต้นสร้างเช็คพอยต์ก่อนจากนั้นจึงแพร่กระจายมาร์คเกอร์ไปสู่โพรเซสทั้งหมด ทันทีที่โพรเซสได้รับมาร์คเกอร์มันจะสร้างเช็คพอยต์และเผยแพร่มาร์คเกอร์ที่ได้รับไปสู่โปรเซสทั้งหมดก่อนที่จะเริ่มส่งเมสเซสของแอพพลิเคชันซึ่งโพรโทคอลนี้ทำงานบนสมมติฐานคือ ช่องทางการสื่อสารเชื่อถือได้และเป็นแบบเข้าก่อนออกก่อน ในกรณีที่ช่องทางการสื่อสารไม่เป็นไฟโฟสามารถแก้ไขได้โดยแนบมาร์คเกอร์ไปกับเมสเซสแรกของแอพพลิเคชันภายหลังการเช็คพอยต์ โดยบังคับให้ทำเช็คพอยต์เมื่อดรรชนีการเช็คพอยต์ของผู้รับมีค่าต่ำกว่าดรรชนีเช็คพอยต์ที่ถูกแนบไปกับเมสเซสดังแสดงในภาพที่ 2.5(c) (Elnozahy, Johnson, & Zwaenepoel, 1992, pp. 39-47)

2.4.3 โคออดิเนตเช็คพอยต์ด้วยนาฬิกา

การปรับนาฬิกาของโปรเซสทั้งหมดให้ใกล้เคียงกันหรือเท่ากันสามารถใช้ทำโคออดิเนตเช็คพอยต์ได้ (Cristian & Jahanian, 1991, pp. 12-20) ด้วยนาฬิกาทำให้สามารถกำหนดจุดเวลาให้โพรเซสทั้งหมดเริ่มทำเช็คพอยต์ในเวลาใกล้เคียงกันโดยไม่ต้องมีผู้ประสานงานและไม่ต้องมีการแลกเปลี่ยนเมสเซสเมื่อแต่ละโพรเซสทำเช็คพอยต์แล้วจะต้องรอคอยเป็นระยะเวลารวมของค่าความคลาดเคลื่อนสูงสุดระหว่างเวลาของนาฬิกาและเวลาสูงสุดในการตรวจจับความผิดพลาดที่เกิดกับโพรเซสหนึ่งในระบบ กรณีเกิดความผิดพลาดและสามารถตรวจพบได้ภายในเวลาที่กำหนดการสร้างเช็คพอยต์จะถูกยกเลิก

2.4.4 คุณลักษณะของเครือข่าย

เนื่องจากโพรโทคอลนี้มีการประสานงานกันเพื่อแลกเปลี่ยนข้อมูลระหว่างโพรเซสทำให้เครือข่ายที่เลือกใช้มีผลต่อความสำเร็จในการสร้างเช็คพอยต์ ในกรณีที่ช่องสื่อสารเป็นแบบเชื่อถือได้ทำให้บางโพรเซสมีความจำเป็นต้องบันทึกข้อมูลเมสเซสที่ตกค้างเข้าไปกับเช็คพอยต์ด้วย พิจารณากรณีที่โพรเซส p ส่งเมสเซส m ก่อนการสร้างเช็คพอยต์และ m เดินทางไปถึงโพรเซส q หลังจากที่ q ทำเช็คพอยต์แล้ว ในกรณีนี้ p ควรบันทึกลงในเช็คพอยต์ว่ามีการส่งเมสเซส m ออกไปและ q ควรบันทึกลงเช็คพอยต์ว่ายังไม่ได้รับเมสเซส m ดังนั้นถ้าความผิดพลาดเกิดขึ้นทำให้ทั้ง p และ q ต้องโรลแบ็คไปยังเช็คพอยต์ดังกล่าวและไม่มีการจัดเก็บเมสเซส m เอาไว้จะส่งผลให้ความเชื่อถือได้ของช่องสื่อสารสูญเสียไป เพื่อหลีกเลี่ยงปัญหานี้จึงต้องบังคับให้เมสเซสที่อยู่ใน

ระหว่างการเดินทางต้องถูกบันทึกไว้กับเช็คพอยต์ของผู้รับปลายทาง แต่ถ้าหากช่องสื่อสารไม่เป็นแบบเชื่อถือได้จะทำให้ไม่จำเป็นต้องบันทึกเมสเซสที่กำลังเดินทางเพราะเช็คพอยต์จะเป็นส่วนหนึ่งของโกลบอลสเตทด้วยสมมติฐานของช่องสื่อสารเองโดยอัตโนมัติ (มองว่าการสูญเสียเมสเซส m กรณีที่ระบบล้มเหลวและเริ่มต้นใหม่จะมีผลเหมือนกับการสูญเสียเมสเซสเนื่องจากช่องสื่อสารล้มเหลวในการทำงานปกติ)

2.4.5 อัตราการขยายตัว

การทำโคออดิเนตเช็คพอยต์มีข้อกำหนดว่าทุก ๆ โพรเซสต้องร่วมมือกันซึ่งข้อกำหนดนี้มีความเกี่ยวข้องโดยตรงการปรับขนาดได้ (scaleable) ของระบบ ดังนั้นเพื่อให้การปรับขนาดได้ของระบบมีสูงจึงจำเป็นต้องลดจำนวนโพรเซสที่มีส่วนร่วมในการเช็คพอยต์ให้น้อยที่สุด (Koo & Toueg, 1987, pp. 23-31) เสนอโพรโทคอลแบบสองเฟสที่สามารถจำกัดจำนวนโพรเซสที่ต้องร่วมทำเช็คพอยต์ให้น้อยที่สุดไว้ดังนี้ เฟสที่หนึ่งให้ผู้ประสานงานค้นหาโพรเซสที่มีการติดต่อกันหลังจากการเช็คพอยต์ครั้งสุดท้ายและส่งคำร้องไปหาโปรเซสเหล่านั้นเพื่อให้ทำการค้นหาโพรเซสอื่นที่มีการติดต่อหลังการเช็คพอยต์ครั้งสุดท้าย ซึ่งกระบวนการนี้จะเกิดขึ้นอย่างต่อเนื่องไปจนกระทั่งไม่สามารถค้นหาโพรเซสเพิ่มเติมได้อีก ในเฟสที่สองโพรเซสทั้งหมดที่ถูกตรวจพบในเฟสหนึ่งจะทำการบล็อกการส่งเมสเซสเพื่อดำเนินการสร้างเช็คพอยต์

2.5 คอมมิวนิเคชันอินดิวส์เช็คพอยต์

วิธีการหลีกเลี่ยงโดมิโนเอฟเฟคอีกวิธีหนึ่งที่ไม่จำเป็นต้องให้ทุก ๆ โพรเซสประสานงานร่วมกันคือ คอมมิวนิเคชันอินดิวส์เช็คพอยต์ (Communication-Induced Checkpoint:CIC) โดยภาพรวมวิธีการนี้จำแนกเช็คพอยต์ออกเป็นสองชนิดคือ โลคอล (local) และฟอร์ส (forced) ซึ่งโพรเซสมีสิทธิสร้างโลคอลเช็คพอยต์ได้อย่างอิสระ ส่วนฟอร์สเช็คพอยต์นั้นโพรเซสจำเป็นต้องทำเพื่อปรับปรุงโกลบอลสเตทให้เป็นปัจจุบันมากที่สุดหรืออีกนัยหนึ่งฟอร์สเช็คพอยต์มีไว้เพื่อป้องกันการสร้างเช็คพอยต์ที่ไม่เป็นประโยชน์เพราะว่าเช็คพอยต์ดังกล่าวไม่มีทางเป็นส่วนหนึ่งของคอนซิสเทนโกลบอลสเตท

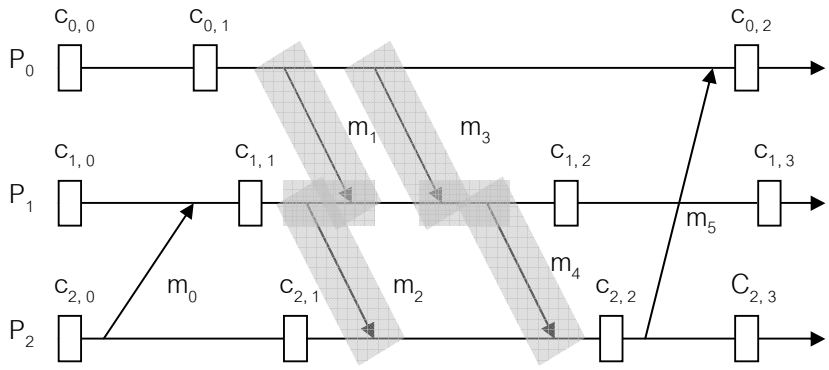
โพรโทคอลนี้สามารถสร้างเช็คพอยต์ได้โดยไม่ต้องมีการแลกเปลี่ยนเมสเซสใด ๆ แต่มันใช้การแนบข้อมูลที่เกี่ยวข้องกับสร้างเช็คพอยต์ไปบนเมสเซสของโปรแกรมประยุกต์โดยตรงเพื่อให้ผู้รับเมสเซสใช้ข้อมูลดังกล่าวในการตัดสินใจว่าจะทำฟอร์สเช็คพอยต์หรือไม่ โดยผู้รับจะวิเคราะห์

รูปแบบของการสื่อสารและการสร้างเช็คพอยต์ในอดีตว่าจะทำให้เกิดเช็คพอยต์สูญเปล่าหรือไม่ ในกรณีที่พบผู้รับจะทำฟอร์สเช็คพอยต์เพื่อทำลายรูปแบบและเช็คพอยต์สูญเปล่าดังกล่าว

คอมมิวนิเคชันอินดิวส์เช็คพอยต์สามารถแบ่งออกเป็นสองประเภทคือ การสร้างเช็คพอยต์ด้วยแบบจำลอง (model-based checkpointing) ที่มีการเก็บข้อมูลโครงสร้างของเช็คพอยต์และการสื่อสารเพื่อใช้หลีกเลี่ยงการสร้างเช็คพอยต์เปล่าประโยชน์ (Wang, 1997, pp. 456-468) อีกประเภทหนึ่งคือ การสร้างเช็คพอยต์ด้วยดรรชนี (index-based checkpointing) ซึ่งใช้วิธีกำหนดตราเวลา (timestamp) ให้กับโลคอลเช็คพอยต์และฟอร์สเช็คพอยต์จากนั้นใช้ตราเวลาดังกล่าวคำนวณหาคอนซิสเทนโกลบอลสเตท

ภาพที่ 2.6

แซดพาธและแซดไซเคิล



2.5.1 แซดพาธและแซดไซเคิล

แซดพาธ (Z-path) คือ เส้นทางการส่งเมสเซสที่มีเช็คพอยต์สองอันเกี่ยวข้อง (Netzer and Xu, 1995) ให้ $\Rightarrow$ แทนความสัมพันธ์เกิดขึ้นก่อน (happened before relation) ของแลมพอร์ต (Lamport, 1978, หน้าที่ 559) ให้ $C_{i,x}$ แทนเช็คพอยต์ลำดับที่ x ของโพรเซส P_i และให้คำจำกัดความส่วนของการทำงานระหว่างเช็คพอยต์สองอันที่ติดกันบนโพรเซสเดียวกันว่าเป็นช่วงระหว่างเช็คพอยต์ เมื่อกำหนดเช็คพอยต์ $C_{i,x}$ และ $C_{j,y}$ จะพบแซดพาธเกิดขึ้นระหว่างเช็คพอยต์ทั้งสองก็ต่อเมื่อเงื่อนไขหนึ่งจากสองข้อต่อไปนี้เป็นจริง:

1. $x < y$ และ $i = j$; หรือ
2. มีเส้นทางของเมสเซส $[m_0, m_1, \ldots, m_n]$, $n \geq 0$ ที่ทำให้:
   - $C_{i,x} \Longrightarrow send_i(m_0)$;
   - $\forall I < n$, ถ้า $deliver_k(m_I)$ และ $send_k(m_{I+1})$ ในช่วงระหว่างเช็คพอยต์เดียวกัน, หรือไม่ก็ $deliver_k(m_I) \Longrightarrow send_k(m_{I+1})$ เพียงอย่างใดอย่างหนึ่ง; และ
   - $deliver_j(m_n) \Longrightarrow C_{j,y}$

ซึ่ง $send_i$ และ $deliver_i$ คือ เหตุการณ์เกี่ยวกับการสื่อสาร (communication event) ที่เกิดขึ้นมาจากโพรเซส P_i ในภาพที่ 2.6 [m1, m2] และ [m3, m4] คือ ตัวอย่างแซดพาธระหว่างเช็คพอยต์ $C_{0,1}$ และ $C_{2,2}$

แซดไซเคิล (Z-cycle) คือ แซดพาธที่เริ่มต้นและจบที่เช็คพอยต์เดียวกัน ในภาพที่ 2.6 [m5, m3, m4] คือ แซดไซเคิลที่เริ่มต้นและจบที่เช็คพอยต์ $C_{2,2}$ ซึ่ง (Netzer & Xu, 1995, pp. 165-169) ทำการพิสูจน์แล้วว่าเช็คพอยต์จะไม่มีประโยชน์ก็ต่อเมื่อมันเป็นส่วนหนึ่งของแซดไซเคิล ดังนั้นวิธีการหนึ่งในการหลีกเลี่ยงการสร้างเช็คพอยต์ที่เปล่าประโยชน์คือ การรับประกันว่าแซดพาธจะไม่มีทางกลายเป็นแซดไซเคิล

2.5.2 การสร้างเช็คพอยต์ด้วยแบบจำลอง

โพรโทคอลนี้กำหนดแบบจำลองของการติดต่อสื่อสารและการสร้างเช็คพอยต์รูปแบบต่าง ๆ เพื่อใช้สำหรับตรวจจับแซดไซเคิลโดยใช้การแก้ปัญหาแบบฮิวริสติก (heuristic) เมื่อตรวจพบว่ามีความเป็นไปได้ที่จะเกิดแซดไซเคิลโพรโทคอลนี้จะบังคับให้โพรเซสทำฟอร์สเช็คพอยต์ แต่เนื่องจากโพรเซสไม่มีข้อมูลโกลบอลสเตทที่สมบูรณ์และเป็นปัจจุบัน จึงเป็นผลให้โพรโทคอลนี้มักสร้างฟอร์สเช็คพอยต์ที่มากเกินความจำเป็น

(Russell, 1980, pp. 183-194) เสนอแบบจำลองของการติดต่อสื่อสารหนึ่งที่มีความสามารถป้องกันโดมิโนเอฟเฟคได้อย่างสมบูรณ์แบบ โดยกำหนดให้เหตุการณ์รับเมสเซส (message receiving event) ต้องมาก่อนเหตุการณ์ส่งเมสเซส (message sending event) เสมอในทุก ๆ ช่วงระหว่างเช็คพอยต์ ดังนั้นแบบจำลองนี้จึงต้องทำเช็คพอยต์หนึ่งครั้งก่อนหน้าที่จะมีการรับเมสเซสเพื่อแยกเหตุการณ์รับเมสเซสไปไว้ในอีกช่วงระหว่างเช็คพอยต์หนึ่ง ส่วนอีกแบบจำลอง

หนึ่ง (Barlett, 1981, pp. 22-29) ที่ใช้หลีกเลี่ยงโดมิโนเอฟเฟคได้นั้นเสนอข้อบังคับที่ตรงกันข้ามกันคือ ให้ทำเช็คพอยต์ก่อนหน้าเหตุการณ์ส่งเมสเซส

2.5.3 การสร้างเช็คพอยต์ด้วยดรรชนี

โพรโทคอลนี้ให้การรับประกันผ่านฟอร์สเช็คพอยต์ว่า (1) ถ้าหากมีเช็คพอยต์ $C_{i,m}$ และ $C_{j,n}$ ซึ่ง $C_{i,m} \Longrightarrow C_{j,n}$ ดังนั้น $ts(C_{j,n}) \geq ts(C_{i,m})$, ซึ่ง ts(c) คือ ตราเวลาของเช็คพอยต์ c และ (2) โลคอลเช็คพอยต์ซึ่งมาทีหลังต้องมีตราเวลามากกว่าอันที่มาก่อน ซึ่งตราเวลาดังกล่าวจะถูกแนบไปกับเมสเซสของโปรแกรมประยุกต์เพื่อช่วยให้ผู้รับตัดสินใจว่าควรทำฟอร์สเช็คพอยต์เมื่อไร (Briatico, Ciuffoletti, & Simoncini 1984, pp. 207-215) เสนอโพรโทคอลที่บังคับให้ทำฟอร์สเช็คพอยต์เมื่อดรรชนีที่แนบมากับเมสเซสมีค่ามากกว่าดรรชนีของโพรเซสรวมไปถึงยังให้การรับประกันว่าในคอนซิสเทนโกลบอลสเตทนั้นเช็คพอยต์ต่าง ๆ จะมีดรรชนีเดียวกันถึงแม้จะอยู่คนละโพรเซส

2.5.4 สมรรถนะ

ในทางทฤษฎีโพรโทคอลนี้ควรให้สมรรถนะเหนือกว่าโคออดิเนตเช็คพอยต์และมีสมรรถนะใกล้เคียงกับอันโคออดิเนตเช็คพอยต์เนื่องจากการให้อิสระในการสร้างเช็คพอยต์ เมื่อพิจารณาในแง่ของปริมาณเช็คพอยต์โพรโทคอลนี้จะสร้างเช็คพอยต์มากกว่าโคออดิเนตเช็คพอยต์แต่น้อยกว่าอันโคออดิเนตเช็คพอยต์ เพราะจากเงื่อนไขในการสร้างฟอร์สเช็คพอยต์อาจส่งผลให้สร้างเช็คพอยต์มากเกินความจำเป็น