รหัสโครงการ SUT7-705-47-12-45

รายงานการวิจัย

การออกแบบระบบคลัสเตอร์สำหรับกลุ่มเครื่องแม่ข่ายโปรแกรมประยุกต์

(Design of Application Server Clusters)

ได้รับทุน อุดหนุนการวิจัยจาก
มหาวิทยาลัยเทคโนโลยีสุรนารี

ผลงานวิจัยเป็นความรับผิดชอบของหัวหน้าโครงการวิจัยแต่เพียงผู้เดียว

รหัสโครงการ SUT7-705-47-12-45

รายงานการวิจัย

การออกแบบระบบคลัสเตอร์สำหรับกลุ่มเครื่องแม่ข่ายโปรแกรมประยุกต์
(Design of Application Server Clusters)

คณะผู้วิจัย

หัวหน้าโครงการ

อาจารย์ ชาญวิทย์ แก้วกสิ

สาขาวิชาวิศวกรรมคอมพิวเตอร์

สำนักวิชาวิศวกรรมศาสตร์

มหาวิทยาลัยเทคโนโลยีสุรนารี

ได้รับทุนอุดหนุนการวิจัยจากมหาวิทยาลัยเทคโนโลยีสุรนารี ปีงบประมาณ พ.ศ. 2547

ผลงานวิจัยเป็นความรับผิดชอบของหัวหน้าโครงการวิจัยแต่เพียงผู้เดียว

กันยายน 2548

กิตติกรรมประกาศ

ขอขอบคุณบุคคลทุกท่านที่ไม่สามารถเอ่ยนามได้ครบถ้วน ซึ่งมีส่วนร่วมช่วยเหลือการวิจัยครั้งนี้ในทุก ๆ ด้าน ทั้งการศึกษาระบบคลัสเตอร์ การพัฒนาระบบทดสอบ การทดสอบ และงานเอกสาร การวิจัยครั้งนี้ได้รับทุนอุดหนุนการวิจัยนักวิจัยรุ่นใหม่ จากมหาวิทยาลัยเทคโนโลยีสุรนารี ปีงบประมาณ 2547

บทคัดย่อ

ได้ศึกษากลไกการสื่อสารระหว่างโหนดในระบบคลัสเตอร์โปรแกรมแม่ข่ายโปรแกรมประยุกต์ JBoss ที่สนับสนุนการทำงานของ Enterprise JavaBeans พบว่า การสื่อสารระหว่างโหนดในคลัสเตอร์เป็นชนิด n-n และ การปรับแต่งการสื่อสารระหว่างโหนดด้วยโปรโตคอลชนิด TCP ให้ความเร็วสูงกว่าชนิด UDP ในกรณีคลัสเตอร์ขนาด 2 และ 4 โหนด

Abstract

This work studies a mechanism of communication system using in JBoss application server which supports Enterprise JavaBeans clustering. It is clearly observed from the experiment that the communication among nodes in a JBoss cluster is n-n. The experimental results also show that the TCP configuration for a cluster layer yields better performance than UDP in the case of 2- and 4-node cluster.

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

สารบัญภาพ

บทที่ 1
บทนำ

1. ความสำคัญ ที่มาของปัญหาที่ทำการวิจัย

ระบบการทำงานบนคอมพิวเตอร์ในปัจจุบันมีการใช้งานเพิ่มมากขึ้น เครื่องแม่ข่ายที่ทำการรองรับจำนวนผู้ใช้ที่มีปริมาณเปลี่ยนแปลงไปในช่วงเวลาต่างๆกันจึงจำเป็นต้องสามารถขยายตัวได้ เพื่อตอบสนองความต้องการดังกล่าว ดังนั้นการนำระบบคลัสเตอร์สำหรับโปรแกรมแม่ข่ายโปรแกรมประยุกต์เข้ามาช่วย จึงมีความสำคัญต่อการขยายตัวและการปรับเปลี่ยนขนาดของระบบ ทำให้สามารถตอบสนองต่อปริมาณผู้ใช้ที่เพิ่มมากขึ้น โดยที่สามารถใช้ทรัพยากรที่มีอยู่เดิมให้คุ้มค่า

มีการพัฒนาซอฟต์แวร์ระบบคลัสเตอร์เชิงพาณิชย์เพื่อใช้กับระบบงาน Enterprise JavaBean (EJB) จากหลายบริษัท เช่น WebLogic ของ BEA, WebSphere ของ IBM, และ BES ของ Borland เป็นต้น ซอฟต์แวร์ดังกล่าวมีราคาสูงและต้องจ่ายค่าลิขสิทธิ์เพื่อใช้งานต่อหน่วยประมวลผล (per CPU Licensing) เมื่อพิจารณาถึงค่าใช้จ่าย ที่สูงเพิ่มขึ้นตามจำนวนเครื่องที่นำมาสร้างระบบคลัสเตอร์แล้วนั้น ทางเลือกของการใช้งานซอฟต์แวร์แบบต้นรหัสเปิด (open source) เพื่อสร้างระบบคลัสเตอร์จึงมีความเหมาะสม ซึ่งในงานวิจัยนี้จะนำซอฟต์แวร์ JBoss จาก JBoss.org มาใช้เพื่อพัฒนาระบบคลัสเตอร์

อย่างไรก็ตาม มีรายงานการเปรียบเทียบประสิทธิภาพของ JBoss กับระบบเชิงพาณิชย์อื่น ๆ พบว่าประสิทธิภาพของ JBoss ต่ำกว่า ดังนั้นเทคนิควิธีในการปรับปรุงประสิทธิภาพเมื่อนำ JBoss ไปใช้เพื่อสร้างระบบคลัสเตอร์เป็นสิ่งจำเป็นที่ต้องได้รับการพัฒนาเพื่อให้ได้ระบบที่ใช้รองรับระบบงานขนาดใหญ่ต่อไป

2. วัตถุประสงค์ของโครงการวิจัย

2.1 เพื่อพัฒนาต้นแบบของระบบคลัสเตอร์เครื่องแม่ข่ายโปรแกรมประยุกต์

2.2 เพื่อพัฒนาเทคนิควิธีการปรับปรุงประสิทธิภาพของระบบคลัสเตอร์

3. ขอบเขตของการวิจัย

3.1 การวิจัยครอบคลุมเฉพาะการสร้างระบบคลัสเตอร์สำหรับโปรแกรมแม่ข่ายโปรแกรมประยุต์ของ JAVAขนาด 4 โหนด

3.2 การทดลองทำบนระบบปฏิบัติการ Windows 2000 และ RedHat Linux

3.3 เครื่องแม่ข่ายที่นำมาสร้างระบบคลัสเตอร์คือเครื่อง Pentium 133 MHz ที่มีหน่วยความจำ 128 Mb และ Ethernet Adapter 10 Mb/sec (เนื่องจากจำนวนอุปกรณ์เครื่องแม่ข่ายไม่พอเพียงสำหรับใช้ทดลอง ดังนั้นจึงปรับเครื่องแม่ข่ายเป็นเครื่อง Pentium 400 MHz ที่มีหน่วยความจำ 256 Mb และ Ethernet Adapter 10/100 Mb/sec แทน)

4. ขั้นตอนการวิจัย

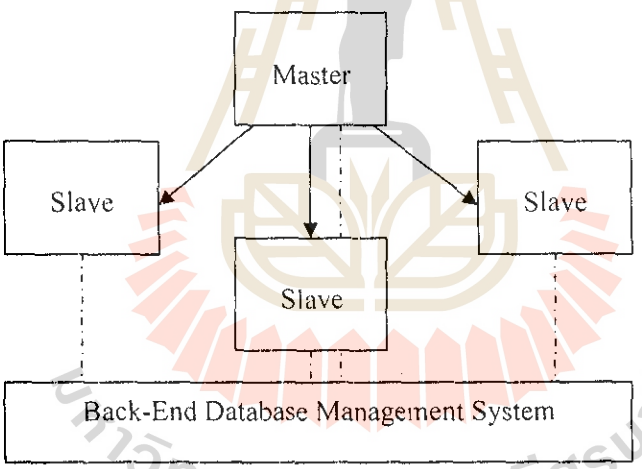
4.1 ศึกษาการใช้งานและติดตั้ง JBoss EJB Container

4.2 สร้างระบบคลัสเตอร์ตามรูปที่ 1

4.3 สร้างระบบงานสำหรับทดสอบ

4.4 พัฒนาเทคนิควิธีการปรับปรุงประสิทธิภาพของระบบคลัสเตอร์

4.5 ทดลองใช้งานระบบงาน และปรับแต่งเพื่อเพิ่มประสิทธิภาพของระบบคลัสเตอร์



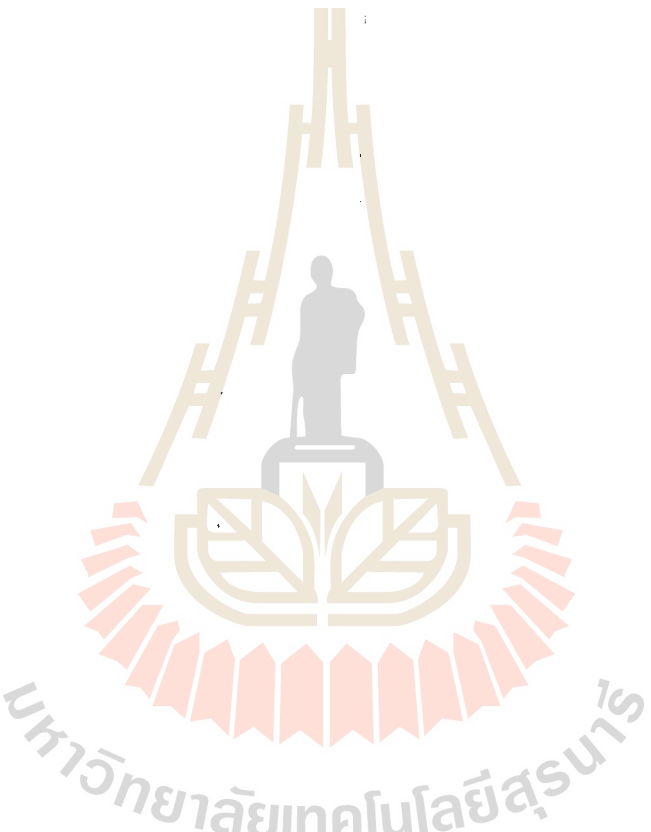
รูปที่ 1. แบบจำลองระบบคลัสเตอร์ขนาด 4 โหนด

5. ประโยชน์ที่คาดว่าจะได้รับ

5.1 เพื่อนำเครื่องคอมพิวเตอร์ที่มีสมรรถนะต่ำมาพัฒนาเพื่อให้รองรับระบบที่มีประสิทธิภาพสูงได้

5.2 เพื่อเป็นต้นแบบการพัฒนาระบบสาธารณูปโภคพื้นฐานทางสารสนเทศ ให้กับมหาวิทยาลัยเทคโนโลยีสุรนารี หรือ หน่วยงานใด ๆ ที่ต้องการสร้างระบบงานด้วยโปรแกรมแม่ข่ายโปรแกรมประยุกต์

ในบทถัดไป บทที่ 2 กล่าวถึงพื้นฐานความรู้เกี่ยวกับคลัสเตอร์ และ EJB บทที่ 3 กล่าวถึงการทำคลัสเตอร์สำหรับ EJB ในบทที่ 4 ได้อธิบายระบบงานที่ใช้ในการทดสอบและการปรับแต่ง บทที่ 5 ผลการวิจัย แสดงผลการปรับแต่งและการทดสอบ และบทสุดท้าย บทที่ 6 ได้ทำการสรุป อภิปรายผลการทดสอบรวมทั้งข้อเสนอแนะ



บทที่ 2

คลัสเตอร์ และ Enterprise JavaBeans

1. คลัสเตอร์

คลัสเตอร์ เป็นกลุ่มของโหนดที่มีจุดมุ่งหมายร่วมกัน โหนดสามารถเป็นคอมพิวเตอร์หรือเป็นเครื่องแม่ข่ายเครื่องหนึ่ง ๆ ในชุดโปรแกรมแม่ข่ายทั่วไป โหนดในคลัสเตอร์มีจุดมุ่งหมายร่วมกันอยู่ 2 ประการ คือ เพื่อให้เกิดความทนทานต่อการล่มของระบบ และ เพื่อกระจายงานผ่านทางการเรพลิเคชั่น โดยปกติแล้วคลัสเตอร์จะมีความสามารถหลายอย่างเพื่อสนันสนุนกันให้บรรลุทั้งสองจุดมุ่งหมาย

"ความสามารถในการบริการ" เป็นสัดส่วนโดยตรงทางเวลากับบริการแต่ละตัวที่สามารถเข้าถึงได้และมีเวลาการตอบสนองที่ดีหรือคาดเดาได้ คำว่า high availability ใช้สำหรับอธิบายสัดส่วนที่สูงของ "การบริการได้" อย่างไรก็ตาม สัดส่วนดังกล่าวขึ้นกับสภาพโดยรอบ ตัวอย่างเช่น ค่า HA สำหรับระบบวิกฤติในเครื่องบินย่อมมีตัวเลขสูงกว่าค่า HA สำหรับเวปไซต์ทั่วไป สัดส่วนของ HA กำหนดได้โดย จำนวนเวลาที่ระบบล่มที่อนุญาตให้เกิดขึ้นได้มากที่สุด ในช่วงเวลาหนึ่งๆ

ตารางที่ 1: แสดงตัวอย่างของ เวลาที่ระบบล่มที่อนุญาตให้เกิดขึ้นได้มากที่สุด ต่อปี ซึ่งขึ้นกับสัดส่วนของ HA

(ที่มา: JBoss 2004)

สัดส่วนค่า HA	เวลาที่ระบบล่มที่อนุญาตให้เกิดขึ้นได้มากที่สุด ต่อปี	
98 %	7.3	วัน
99 %	87.6	ชั่วโมง
99.5 %	43.8	ชั่วโมง
99.9 %	8.76	ชั่วโมง
99.95 %	4.38	ชั่วโมง
99.99 %	53	นาที
99.999 %	5.25	นาที
99.9999 %	31	วินาที
99.99999 %	3.1	วินาที

จากตารางที่ 1, หน้าที่ 4 แสดงให้เห็นว่า สัดส่วนของ HA มีความสัมพันธ์กับ เวลาที่ระบบล่มที่อนุญาตให้เกิดขึ้น ส่วนค่าใช้จ่ายในการดูแลระบบนั้นโดยทั่วไปไม่ได้เป็นสัดส่วนกันกับ HA เช่น การปรับ HA จาก 99% เป็น 99.99% โดยปกติแล้วจะมีค่าใช้จ่ายสูงกว่าการปรับจาก ที่ 98% ไปยัง 99% มาก แม้ว่าค่า % ของความแตกต่างจะดูเท่ากัน – คือเพิ่มขึ้น 1 % ตัวอย่างเช่น ระบบธุรกิจที่ต้องบริการลูกค้า 24 ชั่วโมง โดยปกติจะต้องการค่า HA ที่ระดับ 99.999 %

2. ความสามารถพื้นฐานของคลัสเตอร์

2.1 ความคงทนต่อความผิดพลาด

ความคงทนต่อความผิดพลาด หมายถึงค่า HA ที่สูง ถึงแม้ว่าข้อมูลที่อยู่ในการบริการที่ HA สูง จะไม่มีความจำเป็นว่าจะถูกต้องแม่นยำ แต่ทว่าบริการที่คงทนต่อการล่ม จะประกันความปกติของพฤติกรรมของระบบ และรู้แน่ชัดต่อจำนวนครั้งและชนิดของความผิดพลาดที่จะเกิดขึ้น นั่นคือในบางระบบจึงต้องการเพียงความเป็น HA (ตัวอย่างเช่น บริการสารบัญ ซึ่งมักจะประกอบไปด้วยข้อมูลที่คงที่)

2.2 การกระจายภาระงาน

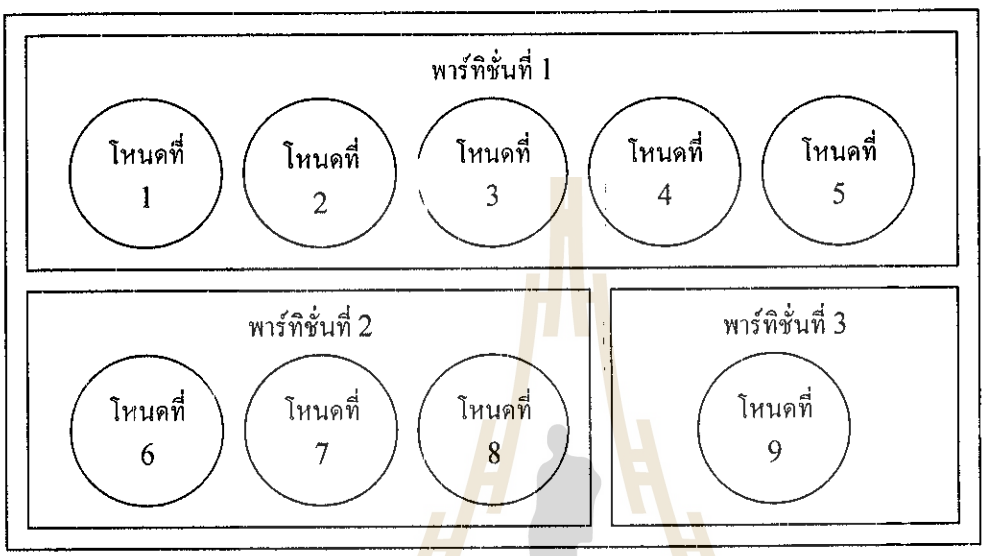
การกระจายภาระงาน หรือ Load balancing เป็นการทำงานในคลัสเตอร์เพื่อให้ได้ระบบประสิทธิภาพสูง โดยการจัดการการร้องขอไปยัง แต่ละต่างๆอย่างรวดเร็ว แต่การกระจายภาระงานที่ดีไม่ได้หมายความว่าของระดับความทนทานต่อการล่มหรือการให้บริการได้ของระบบจะสูงขึ้น ตัวอย่างเช่น เวปไซต์หนึ่งอาจใช้ฟาร์มของเครื่องแม่ข่ายเพื่อสร้างหน้าเวปโดยมีข้อมูลปกติเก็บอยู่ในระบบฐานข้อมูล ถ้าไม่เกิดคอขวดขึ้นที่ระบบฐานข้อมูล การกระจายภาระงานจะให้ประสิทธิภาพสูงอย่างชัดเจน อย่างไรก็ตาม ระบบฐานข้อมูลเดี่ยวจะทำให้เกิด "จุดล่ม" การพังของเครื่องสำหรับฐานข้อมูลเพียงเครื่องเดียวจะทำให้แม่ข่ายทั้งหมดไร้ประโยชน์ทันที ในกรณีนี้ การเพิ่มจำนวนเครื่องแม่ข่ายในฟาร์มไม่ได้เพิ่มค่า HA

ในบางระบบมีทั้งความทนทานต่อการล่ม ที่ทำให้ค่า HA สูง และ การกระจายภาระงาน ซึ่งทำให้เกิด ความสามารถในการขยายระบบ (scalability)

2.3 สถาปัตยกรรมคลัสเตอร์

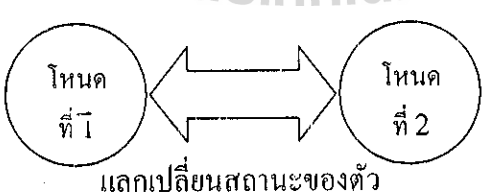
ในส่วนนี้จะกล่าวถึงสถาปัตยกรรมของระบบคลัสเตอร์ ซึ่งจะเฉพาะเจาะจงกับสถาปัตยกรรมโปรแกรมแม่ข่ายโปรแกรมประยุกต์สำหรับจาวา (Java Application Server)

คลัสเตอร์คือกลุ่มของโหนดที่มีขนาดใด ๆ กลุ่มย่อยของโหนดสามารถเรียกได้ว่าเป็น พาร์ทิชั่น นั่นคือเราจะสามารถมีพาร์ทิชั่นได้มากกว่าหนึ่งพาร์ทิชั่นในคลัสเตอร์



รูปที่ 2. ระบบพาร์ทิชั่น

จากรูปที่ 2 จะเห็นได้ว่าในระบบหนึ่ง ๆ ที่ใช้งานคลัสเตอร์ เราสามารถมีพาร์ทิชั่นได้มากกว่าหนึ่งพาร์ทิชั่น ในกรณีที่อยู่ในเครือข่ายเดียวกัน แต่ละพาร์ทิชั่นจะต้องมีชื่อของตัวเองที่ไม่ซ้ำกับพาร์ทิชั่นอื่น ในงานวิจัยนี้เราศึกษาระบบคลัสเตอร์ใน JBoss ซึ่งความสามารถในการติดต่อกันระหว่างแต่ละโหนดในพาร์ทิชั่นนั้น JBoss ใช้รหัสของ JGroups เป็นตัวจัดการ โหนดภายในพาร์ทิชั่นเดียวกันจะทำงานเพื่อเป้าหมายเดียวกัน ยกตัวอย่างเช่น การกระจายภาระงาน ตามที่กล่าวมาแล้วเป็นต้น การแบ่งให้โหนดอยู่ในพาร์ทิชั่นก็เพื่อให้สามารถช่วยกันทำงานให้ได้ประสิทธิภาพยิ่งขึ้น



รูปที่ 3. การสำรองค่าวัตถุและตัวแปรในหน่วยความจำ ในคลัสเตอร์ขนาด 2 โหนด

รูปที่ 3, หน้าที่ 6 แสดงการสำรองข้อมูลวัตถุและตัวแปรสำหรับวัตถุที่ต้องการคงสถานะไว้ในกรณีที่โหนดใดโหนดหนึ่งพังลง ระบบงานก็จะยังดำเนินต่อไปได้เป็นปกติ เพราะสถานะของวัตถุจะยังสามารถเข้าถึงได้จากโหนดที่เหลืออยู่

2.4 การค้นหาโหนด

กระบวนการสำคัญอย่างหนึ่งในการสร้างคลัสเตอร์คือความสามารถในการค้นหาโหนดในคลัสเตอร์บางระบบนั้นไม่จำเป็นต้องมีรูปแบบคลัสเตอร์ที่ตายตัว เมื่อกำหนดพาร์ทิชั่นและตั้งชื่อให้แล้วโหนดใด ๆ สามารถเข้าร่วมหรือออกจากพาร์ทิชั่นนั้น ๆ เมื่อใดก็ได้ ซึ่งจะเป็นผลทำให้รูปแบบของคลัสเตอร์เปลี่ยนไปได้เสมอ ลูกข่ายก็จะสามารถเข้าใช้คลัสเตอร์ได้อย่างปกติ โดยข้อมูลของรูปแบบคลัสเตอร์ใหม่จะส่งจากระบบคลัสเตอร์ไปยังลูกข่ายโดยอัตโนมัติ

2.5 การติดต่อในเครือข่าย

จากที่กล่าวมาแล้วในข้างต้น โหนดในระบบคลัสเตอร์จะต้องทำงานร่วมกัน นั่นคือ โหนดเหล่านั้นจะสามารถติดต่อกันได้ เพื่อที่จะตรวจสอบการเข้าร่วมของโหนดใหม่และการพังของโหนดที่มีอยู่, เพื่อที่จะแลกเปลี่ยนสถานะของวัตถุและตัวแปรซึ่งกันและกัน, และเพื่อที่จะทำงานใด ๆ ไปพร้อม ๆ กัน โดยงานเหล่านี้จะทำในระดับเครือข่ายโดยใช้ชุดรหัสของเฟรมเวิร์ค JGroups

การทำงานของ JGroups จะอยู่บนแนวคิดของ Channel หรือ "ช่องทางติดต่อ" โดยช่องทางนี้จะหมายถึงวิธีการที่สามารถทำให้โหนดที่ *เข้าร่วม* กลุ่มกับโหนดอื่น ๆ สามารถ *แลกเปลี่ยน* ข้อความถึงกันได้ทั้งในลักษณะ unicast หรือ multicast

โดยสรุป การค้นหาโหนดและการติดต่อในเครือข่ายเป็นปัจจัยสำคัญที่ต้องมีในระบบคลัสเตอร์ และควรจะอนุญาตให้ปรับแต่งได้อย่างยืดหยุ่นเพื่อให้เข้ากันได้กับระบบเครือข่ายชนิดต่าง ๆ เช่น เครือข่ายท้องถิ่น (LAN) หรือเครือข่ายกว้าง (WAN) เป็นต้น

3. Enterprise JavaBeans

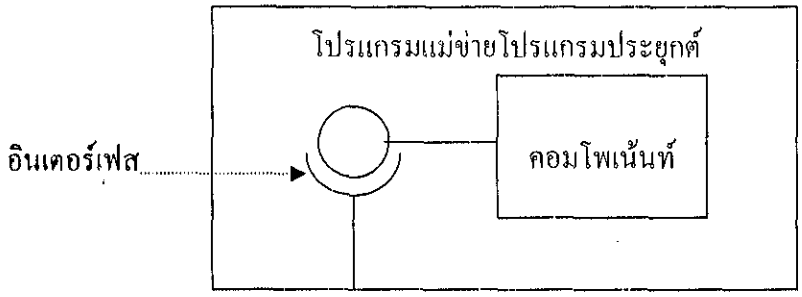
Enterprise JavaBeans (EJB) เป็นสถาปัตยกรรมคอมโพเน้นท์ฝั่งแม่ข่ายที่ช่วยให้กระบวนการพัฒนาโปรแกรมระดับเอ็นเตอร์ไพรส์ (enterprise-class application) ที่ประกอบไปด้วยคอมโพเน้นท์แบบกระจาย (distributed component) ทำได้ง่ายขึ้น EJB สร้างขึ้นมาเพื่อใช้สำหรับภาษา Java ประโยชน์ในการใช้งาน EJB คือทำให้สามารถสร้างโปรแกรมที่เชื่อถือได้ ปลอดภัย และพัฒนาต่อได้ง่าย โดยไม่มีความจำเป็นในการพัฒนาเฟรมเวิร์คที่ซับซ้อนเพื่อจัดการคอมโพเน้นท์แบบกระจายขึ้นเอง EJB นั้นตั้งใจจะมีไว้เพื่อให้สามารถพัฒนาคอมโพเน้นท์ฝั่งแม่ข่ายได้

อย่างรวดเร็วโดยการพัฒนา EJB คอมโพเน้นท์ด้วยภาษา Java และสามารถนำระบบที่พัฒนาขึ้นไปใช้งานในระบบพื้นฐานเดียวกัน ที่เป็นโปรแกรมแม่ข่ายซึ่งรองรับมาตรฐาน EJB และด้วยการออกแบบ EJB ให้เป็นมาตรฐาน ทำให้ตัวคอมโพเน้นท์และระบบงานสามารถใช้งานได้โดยไม่ยึดติดกับโปรแกรมแม่ข่ายของบริษัทใดบริษัทหนึ่ง

ระบบแบบกระจาย (Distributed system) คือระบบขนาดใหญ่ที่สามารถแบ่งส่วนประกอบในระบบออกจากกันเป็นชิ้นย่อย ๆ หรือ คอมโพเน้นท์ ได้ อีกทั้งยังแยกออกแบบระดับชั้น (layer) ที่เป็นอิสระและไม่ขึ้นต่อกัน ความต้องการในการพัฒนาระบบแบบกระจายทำให้เกิด *โปรแกรมแม่ข่ายโปรแกรมประยุกต์* (Application Server) หรือเรียกกันโดยทั่วไปว่า *ตัวกลาง* (middleware – มิดเดิ้ลแวร์) ที่ผ่านมาองค์กรต่าง ๆ ต่างก็สร้างตัวกลางขึ้นมาใช้งานเอง อย่างไรก็ตาม เมื่อจำนวนลูกข่ายเพิ่มมากขึ้นและต้องการระบบงานที่ใช้ตัวกลางกันในระดับคลัสเตอร์ ทำให้มีแนวคิดในการสร้าง โปรแกรมแม่ข่ายโปรแกรมประยุกต์ที่เป็นมาตรฐานขึ้น เพื่อให้ผู้พัฒนาระบบไม่จำเป็นต้องกังวลเกี่ยวกับสาธารณูปโภคพื้นฐานของแม่ข่ายโปรแกรมประยุกต์ จึงมีเวลามากขึ้นในการแก้ปัญหาเชิงตรรกทางธุรกิจให้กับระบบ

3.1 สถาปัตยกรรมของคอมโพเน้นท์

ปัญหาในยุคแรกของการพัฒนาระบบในโปรแกรมแม่ข่ายโปรแกรมประยุกต์ที่ยังไม่มีมาตรฐานคือ การพัฒนาระบบแล้วทำให้ต้นรหัสยึดติดกับตัวโปรแกรมแม่ข่าย สถาปัตยกรรมคอมโพเน้นท์ของ Java จึงออกแบบมาเพื่อให้ตัวคอมโพเน้นท์สามารถย้ายจากโปรแกรมแม่ข่ายของบริษัทหนึ่งไปยังอีกบริษัทหนึ่งได้โดยที่ไม่ต้องแก้ไขและคอมไพล์ต้นรหัสใหม่ แนวคิดดังกล่าวโยงไปหากระบวนการประกาศ *ข้อตกลง* ในการโปรแกรมเชิงวัตถุ หรือ อินเตอร์เฟส (interface) โดยจะกำหนดกลุ่มของอินเตอร์เฟสระหว่างคอมโพเน้นท์และโปรแกรมแม่ข่ายโปรแกรมประยุกต์ไว้ตั้งแต่แรก การตั้งข้อตกลงดังกล่าวทำให้สามารถย้ายคอมโพเน้นท์ไปทำงานบนโปรแกรมแม่ข่ายใด ๆ ก็ได้ที่มีมาตรฐานตามข้อตกลง และข้อตกลงนี้เองคือสถาปัตยกรรมคอมโพเน้นท์ของ EJB รูปที่ 4, หน้าที่ 9 แสดงภาพคอมโพเน้นท์ที่อยู่ในโปรแกรมแม่ข่ายโปรแกรมประยุกต์ซึ่งเชื่อมต่อกันอยู่ด้วย อินเตอร์เฟสตามข้อตกลง



รูปที่ 4. คอมโพเน้นท์ในโปรแกรมแม่ข่าย

3.2 การใช้ EJB เป็น คอมโพเน้นท์ระดับธุรกิจ

โดยทั่วไประบบงานจะแยกได้ออกเป็น 3 ระดับชั้นดังต่อไปนี้ 1. ระดับแสดงผล (Presentation Layer) 2. ระดับตรรกเชิงธุรกิจ (Business Logic Layer) และ 3. ระดับวัตถุเชิงธุรกิจ (Business Object Layer) โดย EJB จะรับทำหน้าที่ในส่วนที่ 2 และ 3

ในระดับแสดงผลนั้น เป็นการทำงานของระบบ*ด้านลูกข่าย* (client-side) ที่เกี่ยวข้องกับผู้ใช้โดยตรง เช่น โปรแกรมลูกข่าย หรือ ตัวแสดงหน้าเว็บ โปรแกรมในกลุ่มนี้ที่รับผิดชอบเกี่ยวกับเว็บอาจถูกเรียกว่า *ระดับชั้นเว็บ* (Web-tier) ก็ได้ ในทางตรงข้าม EJB ซึ่งมีหน้าที่ในระดับที่ 2 และ 3 ไม่ได้ทำงานที่เกี่ยวข้องกับด้านลูกข่าย คอมโพเน้นท์ที่สร้างเป็น EJB จึงมักเรียกว่า คอมโพเน้นท์*ด้านแม่ข่าย* (server-side) โดยหมายถึงการทำงานบนฝั่งแม่ข่าย มีหน้าที่เกี่ยวกับการประมวลผลที่ซับซ้อน หรือ ทำทรานแซ็คชั่นทางธุรกิจจำนวนมาก เป็นต้น คอมโพเน้นท์ด้านแม่ข่ายมักมีความจำเป็นที่ต้องทำงานได้ตลอดเวลา ทนต่อความผิดพลาด มีระบบจัดการทรานแซ็คชั่น สนับสนุนผู้ใช้หลายคน และต้องมีความปลอดภัยของการขนย้ายข้อมูล เป็นต้น โปรแกรมแม่ข่ายโปรแกรมประยุกต์จะเป็นตัวให้บริการนี้แก่คอมโพเน้นท์ EJB รวมถึงกระบวนการจัดการ EJB ขณะที่ระบบทำงานอยู่ นั่นหมายถึง EJB จะรับหน้าที่ในการแก้ปัญหาตรรกเชิงธุรกิจอย่างเดียวดังที่กล่าวไว้แล้ว และโดยปกติ EJB จะใช้ในงานต่อไปนี้

1. การทำงานตามตรรกเชิงธุรกิจ
2. การเข้าถึงข้อมูลในฐานข้อมูล
3. การเข้าถึงข้อมูลในระบบภายนอก

นั่นหมายความว่าการใช้งาน EJB ต้องการลูกข่ายประเภทใดประเภทหนึ่ง อย่างเช่น ลูกข่ายแบบหนา (Think Client) เวบชนิดพลวัต (Dynamic Web) หรือ ลูกข่ายเวบเซอร์วิส (Web Service Client) เป็นต้น

3.3 เทคโนโลยี Java EE

EJB เป็นเพียงส่วนหนึ่งของ Java Platform, Enterprise Edition หรือ Java EE โดยตัว Java EE เป็นข้อกำหนดในการสร้างซอฟต์แวร์ระดับเอ็นเตอร์ไพรส์ของภาษา Java ที่ทำให้เกิดมาตรฐานในกลุ่มบริษัทผู้สร้างซอฟต์แวร์ตัวกลาง Java EE นั้นประกอบไปด้วย

1. ข้อกำหนด
2. ชุดทดสอบ
3. อิมพลีเมนเทชั่นอ้างอิง

ที่ผ่านมา Java EE รู้จักกันในชื่อของ Java 2 Platform Enterprise Edition หรือ J2EE แต่เมื่อมีการพัฒนา Java Development Kit ออกมาใหม่จึงได้เปลี่ยนชื่อ จาก J2EE เป็น Java EE และสำหรับ EJB ที่กล่าวถึงในงานวิจัยนี้เป็น EJB รุ่น 2.1 ซึ่งเป็นส่วนหนึ่งของข้อกำหนด J2EE รุ่น 1.4

เทคโนโลยีที่เป็นส่วนประกอบของ J2EE 1.4 มีดังต่อไปนี้

1. Enterprise JavaBeans (EJB)
2. Java API for XML RPC (JAX-RPC)
3. Java Remote Method Invocation (RMI) และ RMI-IIOP
4. Java Naming and Directory Interface (JNDI)
5. Java Database Connectivity
6. Java Transaction API (JTA) และ Java Transaction Service (JTS)
7. Java Messaging Service (JMS)
8. Java Servlet
9. JavaServer Pages
10. Java IDL
11. JavaMail
12. J2EE Connector Architecture (JCA)
13. Java API for XML Parsing (JAXP)
14. Java Authentication and Authorization Service (JAAS)

3.4 EJB พื้นฐาน

ส่วนนี้จะกว่าถึงความแตกต่างกันของ EJB แต่ละประเภท และส่วนประกอบของ EJB ซึ่งได้แก่ คลาสของ EJB (class of EJB), อินเตอร์เฟสไกล (remote interface), อินเตอร์เฟสท้องถิ่น (local interface), วัตถุ EJB (EJB object), วัตถุท้องถิ่น (local object), อินเตอร์เฟสโฮม (home interface), วัตถุโฮม (home object), ตัวอธิบายการดีพลอย (deployment descriptor) และ ไฟล์ EJB-jar

เทคโนโลยี EJB อยู่บนพื้นฐานของ เทคโนโลยี 2 อย่างคือ RMI-IIOP และ JNDI ใน เบื้องต้นจะเป็นการอธิบายพื้นฐานของเทคโนโลยี RMI-IIOP และ JNDI

3.5 Java RMI-IIOP

Java RMI-IIOP มีชื่อเต็มว่า Java Remote Method Invocation over the Internet Inter-ORB Protocol เป็นกลไกพื้นฐานที่ใช้ใน J2EE เพื่อทำการเรียกใช้เมธอดห่างไกล ใช้สำหรับสร้างวัตถุแบบกระจาย ทำให้วัตถุติดต่อกันได้ทั้งในระดับหน่วยความจำเดียวกัน ต่างเวอร์ช่วลแมชชีน หรือ ต่างเครื่อง (ข้ามเครือข่ายหรือกระบวนการของเครือข่าย)

การเรียกกระบวนความระยะไกล (Remote Procedure Call) หรือ RPC เป็นการเรียกใช้กระบวนความจากโปรเซสซึ่งอยู่บนเครื่องหนึ่งไปยังอีกเครื่องหนึ่ง การใช้งาน RPC ทำให้คงตัวกระบวนความไว้ที่เครื่องใด ๆ แต่สามารถเรียกใช้ก้นข้ามโปรเซสหรือข้ามเครื่องได้

การเรียกใช้เมธอดระยะไกล (Remote Method Invocation) ในแพลตฟอร์ม Java ใช้แนวคิดของ RPC ดังกล่าวและปรับปรุงเพื่อให้สื่อสารได้ใน ระดับวัตถุ ซึ่งเป็นวัตถุแบบกระจาย การใช้ RMI-IIOP จึงทำให้ไม่เพียงแต่เรียกใช้กระบวนความอย่างเดียวเท่านั้น หากยังเรียกใช้เมธอดในวัตถุระยะไกลได้อีกด้วย ซึ่งทำให้สามารถพัฒนารหัสที่ต้องการใช้งานบนเครือข่ายด้วยแนวคิดเชิงวัตถุทั้งการสืบทอด การปิดซ่อน เป็นต้น RMI พัฒนาขึ้นโดยช่วยแก้ปัญหาพื้นฐานในการเรียกกระบวนความระยะไกล ดังต่อไปนี้

1. การมาร์แชลและการอันมาร์แชล (Marshalling and unmarshalling)

กลไกของ RMI จะอนุญาตให้ส่งพารามิเตอร์ไปยังเมธอดระยะไกลได้ทั้งในรูปแบบของชนิดข้อมูลพื้นฐานและชนิดข้อมูลแบบวัตถุผ่านทางเครือข่าย แต่อาจเป็นไปได้ว่าเครื่องระยะไกลอาจใช้รูปแบบข้อมูลที่อยู่ในระบบที่ต่างกัน หรือแม้แต่มีกลไกการจัดการวัตถุด้วยวิธีที่ต่างกัน เช่น เครื่องระยะไกลมีระบบจัดการหน่วยความจำที่ต่างไปจากเครื่องต้นทาง เป็นต้น RMI จึงจำเป็นต้องพึ่งพากระบวนการ Marshalling และ Unmarshalling ซึ่งทำหน้าที่ประมวลผลพารามิเตอร์ของเมธอดเพื่อให้สามารถใช้ข้อมูลเหล่านั้นบนเครื่องระยะไกลที่มีสภาวะแวดล้อม (ระบบปฏิบัติการ, หน่วยประมวลผล, ฮาร์ดแวร์อื่น ๆ) ต่างจากเครื่องต้นทางได้

2. รูปแบบการส่งค่าพารามิเตอร์ (Parameter Passing Conventions)

การส่งค่าพารามิเตอร์แบ่งออกได้เป็น 2 ลักษณะใหญ่ ๆ คือ การส่งค่า และ การส่งค่าอ้างอิง เมื่อเกิดการส่งค่าแบบปกติ ค่าที่ส่งไปยังเมธอดระยะไกลคือสำเนาของค่าจากต้นทาง นั้นหมายถึงการเปลี่ยนแปลงค่าที่ปลายทางจะไม่เกิดผลกระทบแต่อย่างใดต่อค่าในเครื่องต้นทาง ในขณะที่พารามิเตอร์ที่ส่งไปด้วยการส่งค่าอ้างอิงนั้นจะเกิดการเปลี่ยนแปลงค่าได้เมื่อมีการแก้ไขค่าที่เครื่องระยะไกล

3. ผลกระทบจากความไม่มีเสถียรภาพของเครือข่ายและเวอร์ช่วลแมชชีน (Network or Virtual Machine Instability)

การพังของเวอร์ช่วลแมชชีนในการใช้งานในระบบเดียว ทำให้โปรแกรมประยุกต์ทั้งหมดที่ทำงานอยู่บนเวอร์ช่วลแมชชีนนั้นพังไปด้วย แต่ในกรณีของโปรแกรมประยุกต์แบบกระจาย ซึ่งมีเวอร์ช่วลแมชชีนหลายตัวทำงานอยู่ด้วยกันนั้น การพังของแมชชีนตัวใดตัวหนึ่งไม่ได้ทำให้ระบบล่มทั้งระบบ นั้นคือการติดต่อระหว่างแมชชีนด้วย RMI ย่อมมีผลกระทบจากความไม่เสถียรดังกล่าว อย่างไรก็ตาม RMI ได้มีการออกแบบให้ทำงานบนมาตรฐานซึ่งทำให้สามารถแจ้งข้อผิดพลาดที่เกี่ยวข้องกับความไม่เสถียรของระบบได้ โดยผู้พัฒนาไม่ต้องจัดการข้อผิดพลาดเหล่านั้นด้วยตนเอง

3.6 ชนิดของ EJB

ชนิดของ EJB แบ่งออกได้เป็น 3 ชนิด

3.6.1 Session Bean

Session Bean ใช้สำหรับจำลองแบบการะบวนการเชิงธุรกิจ ซึ่งมักจะอยู่ในรูปของการกระทำ เช่น การคำนวณ การเข้าถึงฐานข้อมูล การเรียกใช้บริการจากระบบงานอื่น หรือการเรียกใช้งาน Bean ตัวอื่น

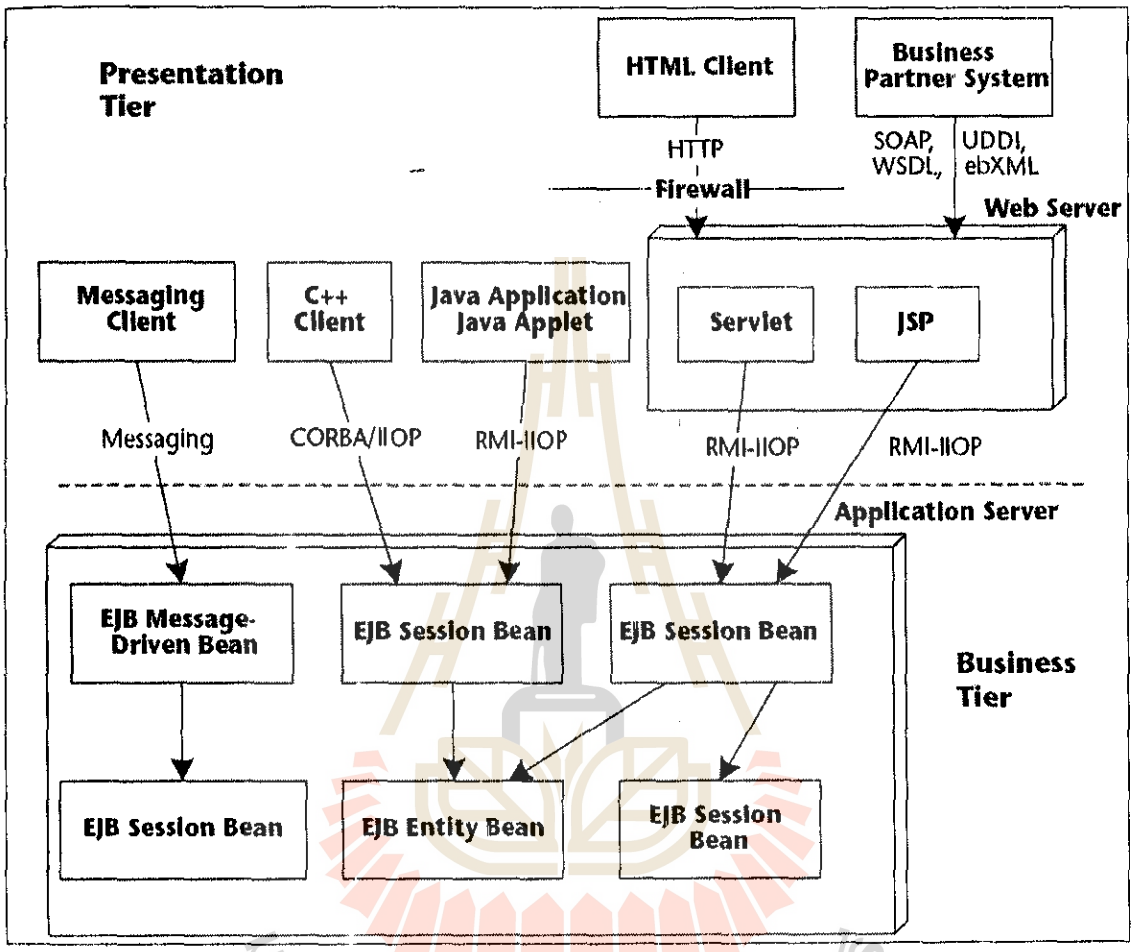
3.6.2 Entity Bean

Entity Bean ใช้สำหรับจำลองข้อมูลทางธุรกิจ เป็นวัตถุสำหรับเก็บข้อมูล กลไกการทำงานพื้นฐานของ Entity Bean คือการแคชข้อมูลจากระบบฐานข้อมูลเพื่อให้ลูกข่ายสามารถเข้าถึงข้อมูลได้อย่างมีประสิทธิภาพเพิ่มขึ้น ตัวอย่างเช่น ข้อมูลสินค้า, การสั่งซื้อ, พนักงาน เป็นต้น

3.6.3 Message-Driven Bean

Message-Driven Bean คล้ายกับ Session Bean ตรงที่เป็นการกระทำเหมือนกัน สิ่งที่ต่างกันก็คือ การเรียกใช้ Message-Driven Bean ทำได้โดยการส่ง *ข้อความ* แทนการเรียกใช้โดยตรง

รูปที่ 5, page 14 แสดงการเรียกใช้ EJB ชนิดต่าง ๆ ในระบบคอมโพเน้นท์ EJB จากลูกข่ายต่างชนิดกัน เช่น HTML Client, Java Application, Java Applet และ Messaging Client



รูปที่ 5. แสดงการเรียกใช้ระบบคอมโพเน้นท์ EJB จากลูกข่าย (ที่มา Mastering EJB 3rd Edition)

บทที่ 3

การทำคลัสเตอร์สำหรับ EJB ใน JBoss

1. คลัสเตอร์สำหรับ Stateless Session Bean

การทำคลัสเตอร์สำหรับ Stateless Session Bean เป็นกรณีที่ง่ายที่สุด คือ ไม่มีสถานะของวัตถุ ในที่นี้คือ Bean มาเกี่ยวข้อง และสามารถเรียกใช้ผ่านกระบวนการกระจายภาระงาน (load balancing) บนโหนดใด ๆ ที่อยู่ภายในคลัสเตอร์เดียวกันได้ คือโหนดที่มีการระบุความเป็นคลัสเตอร์ที่ Bean นี้ติดตั้งอยู่ ในกรณีของ JBoss เพื่อให้ Bean ที่ต้องการเป็นวัตถุที่ใช้งานได้ในระบบคลัสเตอร์ ต้องทำการแก้ไขไฟล์ตัวอธิบาย "jboss.xml" เพื่อเพิ่มแท็ก <clustered> เข้าไป สำหรับโปรแกรมแม่ข่ายอื่น ๆ อาจมีการตั้งค่าความเป็นคลัสเตอร์ด้วยวิธีการที่แตกต่างกัน

```
<jboss>
  <enterprise-beans>
    <session>
      <ejb-name>sut.StatelessSession</ejb-name>
      <jndi-name>sut.StatelessSession</jndi-name>
      <clustered>True</clustered>
      <cluster-config>
        <partition-name>DefaultPartition</partition-name>
        <home-load-balance-policy>
          org.jboss.ha.framework.interfaces.RoundRobin
        </home-load-balance-policy>
        <bean-load-balance-policy>
          org.jboss.ha.framework.interfaces.RoundRobin
        </bean-load-balance-policy>
      </cluster-config>
    </session>
  </enterprise-beans>
</jboss>
```

รูปที่ 6. การตั้งค่า Stateless Session Bean ให้มีความเป็นคลัสเตอร์

ในการปรับแต่งตัวอธิบาย Bean ของ JBoss ในรูปที่ 6 แท็ก<clustered> เป็นแท็กที่ต้องระบุเพื่อบ่งบอกว่า Bean นั้นจะทำงานเป็นคลัสเตอร์ แท็กอื่น ๆ ทั้งหมดที่เป็นสมาชิกย่อยอยู่ภายใต้แท็ก <cluster-config> เป็นค่าตัวเลือก และค่าโดยปริยายจะเป็นไปตามตัวอย่างการปรับแต่งข้างต้น

แท็ก <partition-name> ใช้เพื่อพิจารณาว่า Bean ดังกล่าวจะเป็นของคลัสเตอร์ใด หากไม่ระบุค่าโดยปริยายจะเป็นพาร์ทิชั่นโดยปริยายของระบบคลัสเตอร์ปัจจุบัน

ค่าของแท็ก <home-load-balance-policy> บ่งบอกชื่อคลาสที่จะใช้โดย home proxy เพื่อกระจายภาระการเรียกใช้วัตถุของ Stateless Session Bean ที่ทำบนโหนดแต่ละโหนดของคลัสเตอร์ โดยปกติแล้ว proxy จะกระจายการเรียกใช้แบบ round-robin สำหรับวิธีการกระจายภาระงานแบบอื่นนั้น ผู้ใช้คลัสเตอร์อาจจะพัฒนาขึ้นเองได้ หรือใช้คลาส org.jboss.ha.framework.interfaces.FirstAvailable สำหรับการใช้งานโหนดแรกสุดของคลัสเตอร์ที่ว่างอยู่และจะใช้ไปเรื่อย ๆ จนกว่าโหนดนั้นจะล่มจึงเปลี่ยนโหนด

แท็ก <bean-load-balance-policy> บ่งบอกว่า คลาสนี้ถูกใช้โดยตัวแทนระยะไกล (remote proxy) เพื่อสมดุลการเรียกใช้ ซึ่งทำบนโหนดของคลัสเตอร์ โดยปกติแล้วตัวแทนจะสมดุลการเรียกใช้โดยใช้วิธีการแบบ round-robin และแท็ก <home-load-balance-policy> ก็ใช้วิธีการตั้งค่าแบบเดียวกัน

2. คลัสเตอร์สำหรับ Stateful Session Bean

การทำคลัสเตอร์สำหรับ Stateful Session Bean ซับซ้อนกว่าการทำคลัสเตอร์แบบ Stateless Bean มากเนื่องจากกระบวนการนี้ต้องจัดการกับสถานะของวัตถุ และในการสร้างระบบจัดการสถานะของ Stateful Session Bean โดยทั่วไป จะไม่ใช้ระบบฐานข้อมูลหรือกลไกอื่น ๆ เพื่อทำการคัดลอกและ กระจายสถานะ ของ Bean ซึ่งกระบวนการนี้จะใช้วิธีการคัดลอกสถานะของ Bean ผ่านหน่วยความจำ (in-memory replication) ระหว่างโหนด โดยสถานะของ Stateful Session Bean ทั้งหมดจะถูกคัดลอกและทำให้ตรงกันกับสำเนาของแต่ละ Bean ข้ามคลัสเตอร์ในทุก ๆ ครั้งที่สถานะของ Bean เปลี่ยน

ในการจัดการสถานะของ Stateful Session Bean ใน JBoss จะต้องใช้บริการระดับคลัสเตอร์ คือ JBoss Mbean ที่มีชื่อว่า HASessionState รูปที่ 7 แสดงการประกาศใช้ HASessionState สำหรับตารางที่ 2 แสดงคุณสมบัติที่ปรับแต่งได้ของ HASessionState

```
<mbean
    code="org.jboss.ha.hasessionstate.server.HASessionStateService"
    name="jboss:service=HASessionState">
</mbean>
```

รูปที่ 7. การประกาศใช้ HASessionState

ตารางที่ 2: คุณสมบัติที่ปรับแต่งได้ของ HASessionState

ชื่อคุณสมบัติ	ต้องมีหรือไม่	ค่าโดยปริยาย	คำอธิบาย
JndiName	มีหรือไม่ก็ได้	/HAPartition/Default	ชื่อ JNDI อ้างอิงของวัตถุ HASessionState
PartitionName	มีหรือไม่ก็ได้	DefaultPartition	ชื่อของพาร์ทิชั่นที่ HASessionState ตัวปัจจุบันจะทำงานด้วย
BeanCleaningDelay	มีหรือไม่ก็ได้	30*60*1000 (30 นาที)	ตัวเลขระบุเวลาเป็นมิลลิวินาทีเพื่อล้างข้อมูลสถานะ Bean ที่ไม่ได้รับการคัดลอกอย่างถูกต้องในระบบคลัสเตอร์

เพื่อให้ Stateful Session Bean ที่ต้องการเป็นวัตถุที่ใช้งานได้ในระบบคลัสเตอร์ ต้องทำการแก้ไขไฟล์ตัวอธิบาย "jboss.xml"

```
<jboss>
  <enterprise-beans>
    <session>
      <ejb-name>sut.StatefulSession</ejb-name>
      <jndi-name>sut.StatefulSession</jndi-name>
      <clustered>True</clustered>
      <cluster-config>
        <partition-name>DefaultPartition</partition-name>
          <home-load-balance-policy>
          org.jboss.ha.framework.interfaces.RoundRobin
        </home-load-balance-policy>
        <bean-load-balance-policy>
          org.jboss.ha.framework.interfaces.FirstAvailable
        </bean-load-balance-policy>
        <session-state-manager-jndi-name>
          /HASessionState/Default
          </session-state-manager-jndi-name>
      </cluster-config>
    </session>
  </enterprise-beans>
</jboss>
```

รูปที่ 8. แสดงการตั้งค่าโดยปริยายของ **Stateful Session Bean** ใน **JBoss**

จากรูปที่ 8, หน้าที่ 16 ในการปรับแต่งตัวอธิบาย Bean แท็ก <clustered> เป็นแท็กที่ต้องระบุเพื่อบ่งบอกว่า Bean นั้นจะทำงานเป็นคลัสเตอร์ แท็กอื่น ๆ ทั้งหมดที่เป็นสมาชิกย่อยอยู่ภายใต้แท็ก <cluster-config> เป็นค่าตัวเลือก และค่าโดยปริยายจะเป็นไปตามตัวอย่างการปรับแต่งข้างต้น

แท็ก <session-state-manager-jndi-name> ใช้เพื่อตั้งชื่อ JNDI name ของตัวบริการ HASessionState ที่ใช้โดย Bean นี้ ซึ่งจะใช้ค่าโดยปริยาย JNDI ของ HASessionState สำหรับรายละเอียดของแท็กที่เหลือจะเหมือนกับ Stateless Session Bean

3. คลัสเตอร์สำหรับ Entity Bean

การทำคลัสเตอร์สำหรับ Entity Bean จะต้องแก้ไขตัวอธิบายใน "jboss.xml" เพื่อให้มีแท็ก <clustered> แสดงอยู่ รูปที่ 9 แสดงค่าโดยปริยายของการทำ Entity Bean ให้มีความเป็นคลัสเตอร์

```
<jboss>
  <enterprise-beans>
    <entity>
      <ejb-name>sut.EnterpriseEntity</ejb-name>
      <jndi-name>sut.EnterpriseEntity</jndi-name>
      <clustered>True</clustered>
      <cluster-config>
        <partition-name>DefaultPartition</partition-name>
        <home-load-balance-policy>
          org.jboss.ha.framework.interfaces.RoundRobin
        </home-load-balance-policy>
        <bean-load-balance-policy>
          org.jboss.ha.framework.interfaces.FirstAvailable
        </bean-load-balance-policy>
      </cluster-config>
    </entity>
  </enterprise-beans>
</jboss>
```

รูปที่ 9. แสดงการตั้งค่าโดยปริยายของ Entity Bean ในระบบคลัสเตอร์ของ JBoss

ในการปรับแต่งตัวอธิบาย Bean แท็ก <clustered> เป็นแท็กที่ต้องระบุเพื่อบ่งบอกว่า Bean นั้นจะทำงานเป็นคลัสเตอร์ แท็กอื่น ๆ ทั้งหมดที่เป็นสมาชิกย่อยอยู่ภายใต้แท็ก <cluster-config> เป็นค่าตัวเลือก และค่าโดยปริยายจะเป็นไปตามตัวอย่างการปรับแต่งข้างต้น สำหรับรายละเอียดของแท็กที่เหลือจะเหมือนกับ Stateless Session Bean

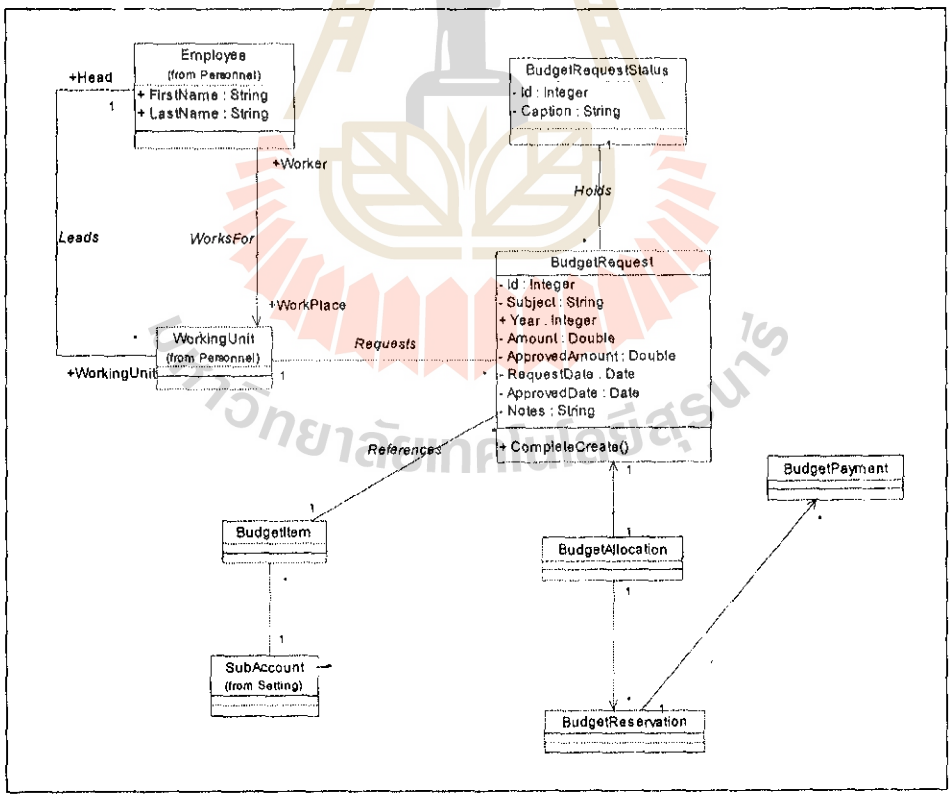
บทที่ 4

ระบบงานทดสอบและวิธีการปรับแต่ง

ระบบที่สร้างขึ้นเพื่อใช้ทดสอบคลัสเตอร์เป็นระบบงานที่ออกแบบโดยใช้ EJB เป็นส่วนหนึ่งของระบบจำลองการทำงานของระบบ MIS ของมหาวิทยาลัยเทคโนโลยีสุรนารี โดยส่วนที่นำมาใช้ทดสอบระบบคลัสเตอร์ เป็นระบบการร้องขอและจัดการงบประมาณ

1. การออกแบบ

การออกแบบในส่วนที่เป็น Entity Bean มีคลาส *BudgetRequest* เป็นคลาสสำหรับเก็บข้อมูลการร้องของบประมาณ *BudgetRequestStatus* เป็นคลาสสำหรับเก็บข้อมูลสถานะของการร้องของงบประมาณ คลาส *WorkingUnit* แทนข้อมูลหน่วยงานที่ร้องของบประมาณ วัตถุของ BudgetRequest แต่ละวัตถุจะอ้างอิงถึงข้อมูลงบประมาณอ้างอิงและส่วนอื่น ๆ ซึ่งละไว้เพื่อลดความซับซ้อนในการอธิบายระบบงาน และคลาสทั้งหมดจะอยู่ใน package th.ac.sut.mis.budget ในรูปที่ 4.1 จะแสดงแผนผังคลาส UML ของระบบงบประมาณ



รูปที่ 10. แผนผัง UML สำหรับระบบทดสอบ

2. การอิมพลีเม้นต์ EJB

2.1. Entity Bean และ ข้อมูล

ได้สร้าง Entity Bean ต่อไปนี้ครอบตารางในฐานข้อมูล

- th.ac.sut.mis.budget.BudgetRequest
  ครอบตาราง BUDGET_REQ ในระบบฐานข้อมูล
- th.ac.sut.mis.personnel.WorkingUnit
  ครอบตาราง WORK_UNIT ในระบบฐานข้อมูล
- th.ac.sut.mis.budget.BudgetRequestStatus
  ครอบตาราง BUDGET_REQ_STATUS ในระบบฐานข้อมูล

เพื่อให้สะดวกต่อการคำนวณประสิทธิภาพ ผู้วิจัยได้เพิ่มสดมภ์ข้อมูลที่ไม่ได้เกี่ยวข้องกับตรรกเชิงธุรกิจเข้าไปในวัตถุจริงที่ใช้ในการทดลอง ให้มีข้อมูลแถวละ 1024 ไบต์

2.2. Stateless Session Bean และบริการ

สำหรับในระดับชั้นของการบริการแบบไม่มีสถานะ ได้ใช้ Stateless Session Bean ในการจำลองการบริการดังนี้

```
package th.ac.sut.mis.budget.interfaces;
…
public interface IBudgetRequestService {
   public List findByYear(int year);
   public List findByWorkingUnitAndYear(int wid, int year);
}
```

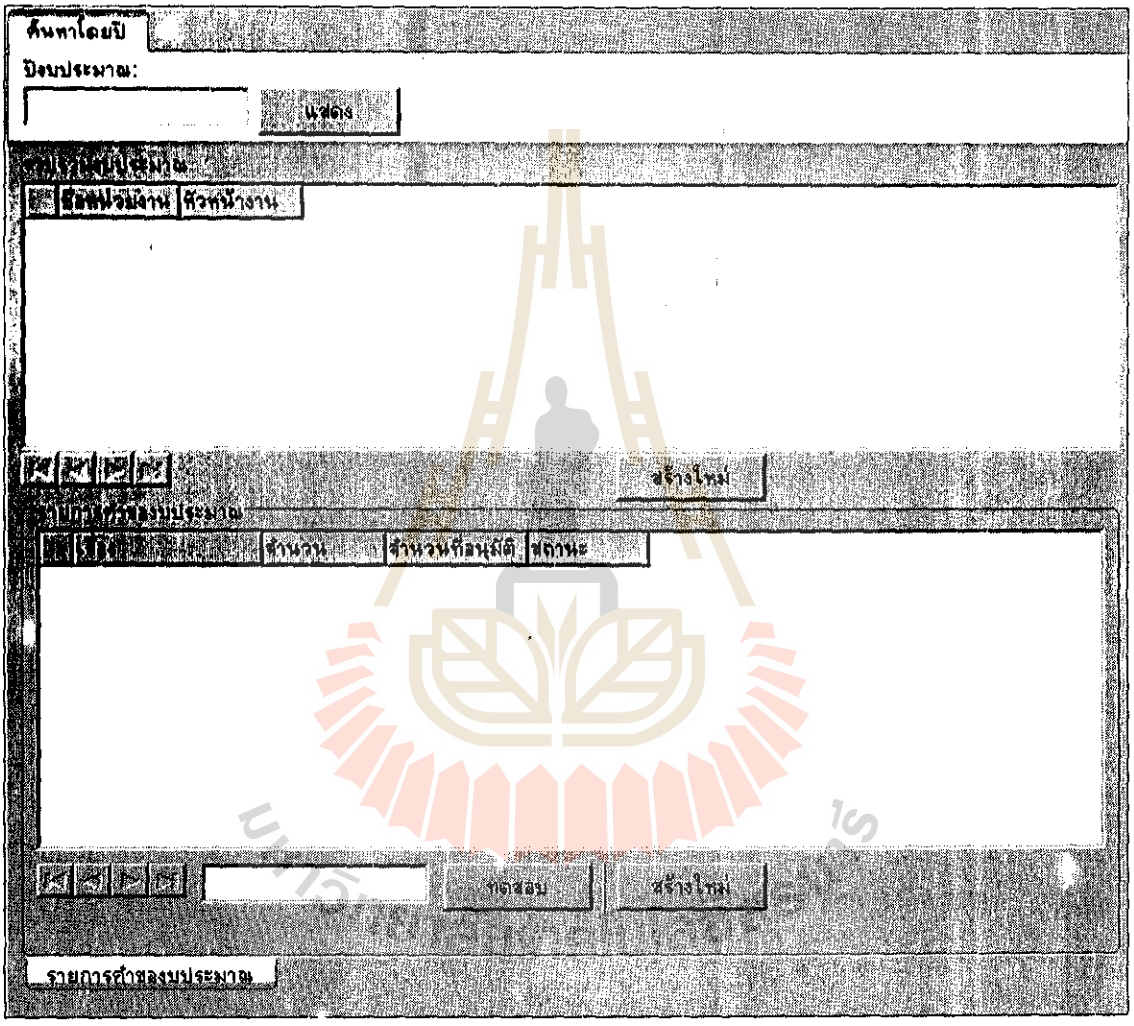
2.3. Stateful Session Bean และตรรกเชิงธุรกิจ

สำหรับในระดับชั้นของการบริการแบบมีสถานะ ได้ใช้ Stateful Session Bean ในการจำลองการบริการดังนี้

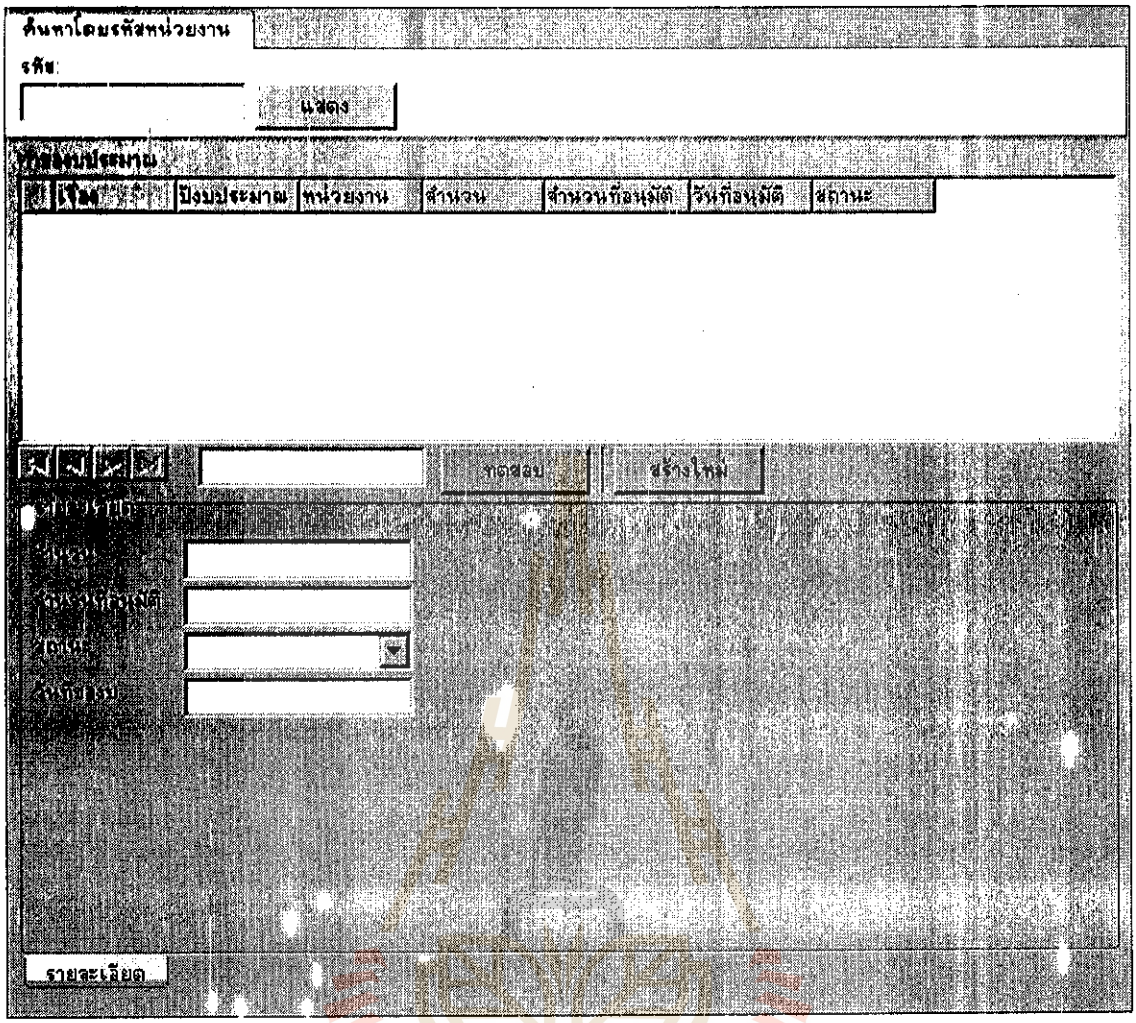
```
package th.ac.sut.mis.budget.interfaces;
…
public interface IBudgetRequestProcess {
   public void addBudgetForWorkingUnit(BudgetRequest b,int wid);
   public void processBudgetRequests(int wid);
}
```

2.4. โปรแกรมลูกข่ายต้นแบบ

ส่วนหนึ่งของโปรแกรมต้นแบบลูกข่ายได้แสดงไว้ในรูปที่ 11 และ 12, หน้าที่ 21 สำหรับรูปที่ 11 แสดงหน้าจอรายการคำของบประมาณ ซึ่งเป็นการแสดงรายละเอียดของข้อมูลจาก Entity Bean *WorkingUnit* และ *BudgetRequest* ในขณะที่รูปที่ 12, หน้าที่ 21 แสดงหน้าจอที่สามารถใช้แก้ไขรายละเอียดของ *BudgetRequest* ได้



รูปที่ 11. หน้าจอแสดงรายการคำของบประมาณ



รูปที่ 12. หน้าจอแสดงรายละเอียดการของบประมาณ

เนื่องจากงานวิจัยเน้นกระบวนการพัฒนาและทดสอบคลัสเตอร์ จึงละรายละเอียดการอธิบายการพัฒนาลูกข่ายไว้

3. การปรับแต่งคลัสเตอร์

ได้ปรับแต่งระบบสื่อสาร JGroups ซึ่งใช้เป็นระบบสื่อสารหลักระหว่างโหนดในพาร์ทิชั่นของระบบคลัสเตอร์ของ JBoss พาร์ทิชั่นเป็นส่วนประกอบพื้นฐานของการสร้างคลัสเตอร์ โดยแต่ละโหนดในคลัสเตอร์จำเป็นต้องอยู่ในอย่างน้อย 1 พาร์ทิชั่น โดยปกติจะสามารถเลือกให้ JBoss ที่ใช้ในงานวิจัยนี้ ทำงานได้ 3 รูปแบบคือ minimal, default หรือ all แต่ระบบคลัสเตอร์จะทำงานได้เฉพาะในรูปแบบ all เท่านั้น ในกรณีของระบบคลัสเตอร์ของ JBoss ต้องทำการตั้งค่าการใช้งานคลัส

เตอร์ใน “cluster-service.xml” และอยู่ในโฟลเดอร์ “deploy” ใต้โฟลเดอร์ของรูปแบบ all ใน “cluster-service.xml” จะประกาศค่าโดยปริยายของพาร์ทิชั่นเป็น “DefaultPartition” และทำการโหลด MBean สำหรับจัดการพาร์ทิชั่นรวมทั้งตั้งค่าเริ่มต้นให้ JGroups และส่วนจัดการอื่น ๆ ของระบบคลัสเตอร์ การนิยามระบบคลัสเตอร์ด้วย MBean ใน JBoss ทำได้ดังนี้ต่อไปนี้

```
<mbean code="org.jboss.ha.framework.server.ClusterPartition'
       name="jboss:service=DefaultPartition"
/>
```

จุดสำคัญภายใน cluster-service.xml คือแอทริบิวต์ PartitionConfig ซึ่งเป็นสายอักขระที่มีรูปแบบ XML เพื่อใช้อธิบายและปรับแต่งค่า JGroups โปรโตคอลสแต็ก ค่าตั้งต้นโดยปริยายของสายอักขระนี้เป็นดังรูปที่ 13, หน้าที่ 23 ถ้าเครื่องแม่ข่ายใช้ระบบปฏิบัติการ Microsoft Windows (2000, XP และ 2003) จะต้องปรับแต่งค่าในรูปที่ 13 เพื่อหลบข้อผิดพลาดของความสามารถ MediaSense ที่มีผลกระทบกับ multicasting และในการตั้งค่าด้านบนต้องตั้งให้ค่า loopback เป็น true

```
<Config>
<UDP mcast_addr="228.1.2.3"
     mcast_port="45566"
     ip_ttl="64"
     ip_mcast="true"
     mcast_send_buf_size="150000" mcast_recv_buf_size="80000"
     ucast_send_buf_size="150000" ucast_recv_buf_size="80000"
     loopback="false"
/>
<PING
     timeout="2000"
     num_initial_members="3"
     up_thread="true" down_thread="true"
/>
<MERGE2
     min_interval="5000"
     max_interval="10000"
/>
<FD shun="true" up_thread="true" down_thread="true" />
<VERIFY_SUSPECT
    timeout="1500"
    up_thread="true"
    down_thread="true"
/>
<pbcast.STABLE
    desired_avg_gossip="20000"
    up_thread="true" down_thread="true"
/>
<pbcast.NAKACK
    gc_lag="50"
    retransmit_timeout="300,600,1200,2400,4800"
    up_thread="true" down_thread="true"
/>
<UNICAST
    timeout="5000" window_size="100"
    min_threshold="10" down_thread="true"
/>
<FRAG
    frag_size="8192" down_thread="true"
    up_thread="true"
/>
<pbcast.GMS
    join_timeout="5000" join_retry_timeout="2000"
    shun="true" print_local_addr="true"
/>
<pbcast.STATE_TRANSFER
    up_thread="true" down_thread="true"
/>
</Config>
```

รูปที่ 13. รายละเอียดของสายอักขระเชื่อมต่อโดยปริยาย ของ **JGroups** โปรโตคอลสแต็ก

3.1. การปรับแต่งแบบ UDP

การปรับแต่งแบบ UDP ให้ความเร็วในบางสภาพแวดล้อมสูงกว่าแบบ TCP เนื่องจากใช้ความสามารถ multicast ของเครือข่าย อย่างไรก็ตามความเชื่อถือได้จะต่ำกว่าโปรโตคอล TCP ได้ทำการปรับแต่ง JGroups โปรโตคอลสแต็กแบบ UDP ดังต่อไปนี้

```
<Config>
<UDP bind_addr="bindAddress"
     mcast_addr="228.1.2.3"
     mcast_port="45566"
     ip_ttl="32"
     loopback="true" <!-- set loopback=fase for Linux -->
/>
<PING timeout="3000"
      num_initial_members="6"
/>
<MERGE2/>
<FD   timeout="5000" />
<VERIFY_SUSPECT
      timeout="1500"
/>
<pbcast.NAKACK
     gc_lag="10"
     retransmit_timeout="3000"
/>
<pbcast.STABLE
     desired_avg_gossip="10000"
/>
<UNICAST timeout="5000"
/>
<pbcast.GMS
     join_timeout="5000"
     join_retry_timeout="2000"
     shun="false"
     print_local_addr="false"
/>
<FC
     max_credits="200000"
     min_credits="5200"
     down_thread="false"
/>
<FRAG/>
</Config>
```

3.2. การปรับแต่งแบบ TCP

การปรับแต่งแบบ TCP ใช้ความน่าเชื่อถือของ TCP ชดเชยความเร็วที่อาจลดไป ในงานวิจัยได้ปรับแต่ง JGroups โปรโตคอลสแต็ก ให้ใช้ระบบสื่อสารภายในคลัสเตอร์ด้วยโปรโตคอล TCP ดังต่อไปนี้

```
<Config>
<TCP bind_addr="bindAddress"
     loopback="true"
/>
<TCPPING
     initial_hosts="192.168.0.1[7800]"
/>
<MERGE2
     min_interval="5000"
     max_interval="10000"
/>
<FD timeout="5000"/>
<VERIFY_SUSPECT
    timeout="1500"
/>
<FRAG />
<GMS
    join_timeout="3000" join_retry_timeout="2000"
    shun="true" print_local_addr="true"
/>
</Config>
```

บทที่ 5
ผลการปรับแต่งและการทดสอบ

1. ผลการทดสอบชนิดของการสื่อสารระหว่างโหนด

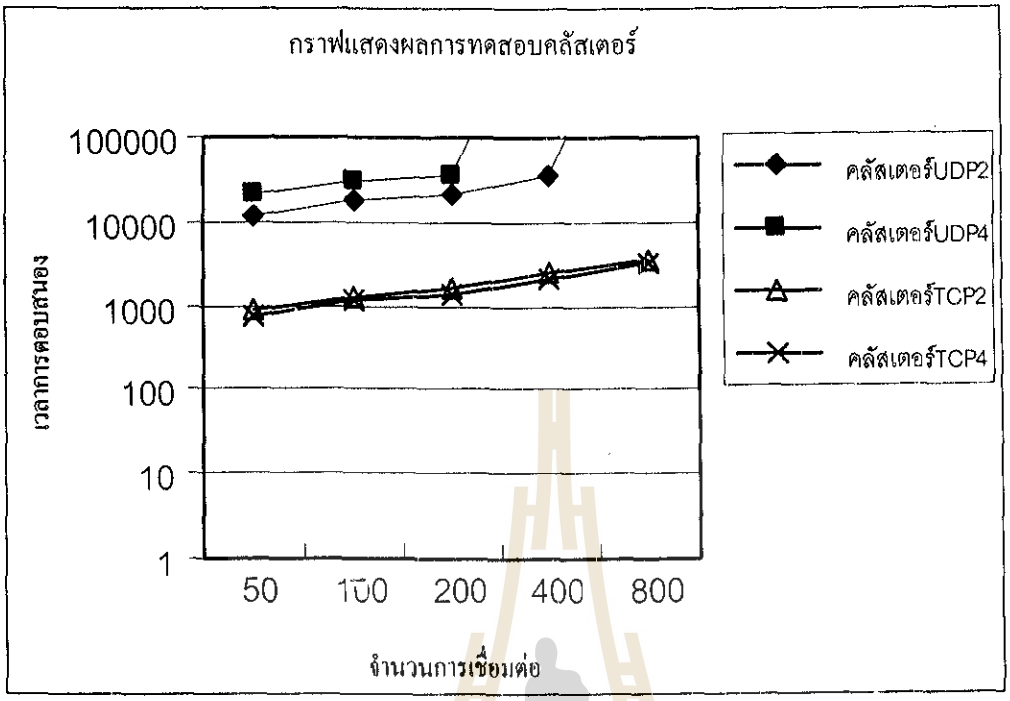
1.1เมื่อโปรแกรมลูกข่ายร้องขอข้อมูลไปยังแม่ข่ายจะไม่เกิดการทำงานในส่วนของการแลกเปลี่ยนสถานะของวัตถุภายในพาร์ทิชั่น ในทุก ๆ รูปแบบของคลัสเตอร์ (ชนิด 2 โหนด และ 4 โหนด)

1.2เมื่อโปรแกรมลูกข่ายทำการเปลี่ยนแปลงข้อมูลและส่งไปบันทึกยังฐานข้อมูลผ่านระบบคลัสเตอร์ ปรากฏว่า โหนดที่รับข้อมูลได้บันทึกสถานะของวัตถุลงฐานข้อมูล และส่งต่อข้อมูลไปยังโหนดอื่น ๆ *ทุก ๆ โหนด* ในพาร์ทิชั่นเดียวกัน

2. ผลกระทบของการปรับแต่งต่อประสิทธิภาพ

ทำการทดสอบโดยเปลี่ยนแปลงการปรับแต่ง 2 แบบคือแบบ UDP และ TCP บนคลัสเตอร์ขนาด 2 โหนด และ 4 โหนด โดยทดสอบบนระบบปฏิบัติการ Windows 2000 และ Redhat Linux จากนั้นหาค่าเฉลี่ยเวลาการตอบสนองระหว่าง 2 ระบบปฏิบัติการ ในการทดสอบได้ทำการเพิ่มจำนวนการเชื่อมต่อของลูกข่ายจาก 50 เป็น 100, 200, 400 และ 800 ตามลำดับ ข้อมูลที่ได้คือเวลาในการตอบสนองเฉลี่ยต่อการเรียกใช้จากลูกข่ายไปยังแม่ข่าย ผลการทดสอบเป็นไปตามรูปที่ 5.1 โดยเวลาในการตอบสนองแทนด้วยแกนตั้งและอยู่ในสเกลล็อก สำหรับสัญลักษณ์ในผลการทดสอบมีดังต่อไปนี้

- **คลัสเตอร์ UPD 2** หมายถึง JBoss คลัสเตอร์ขนาด 2 โหนดที่ใช้การปรับแต่ชนิด UDP
- **คลัสเตอร์ UPD 4** หมายถึง JBoss คลัสเตอร์ขนาด 4 โหนดที่ใช้การปรับแต่ชนิด UDP
- **คลัสเตอร์ TCP 2** หมายถึง JBoss คลัสเตอร์ขนาด 2 โหนดที่ใช้การปรับแต่ชนิด TCP
- **คลัสเตอร์ TPC 4** หมายถึง JBoss คลัสเตอร์ขนาด 4 โหนดที่ใช้การปรับแต่ชนิด TCP



รูปที่ 14. แสดงผลการทดสอบคลัสเตอร์

บทที่ 6
บทสรุป

1. ชนิดการสื่อสารภายในคลัสเตอร์ JBoss

จากการทดสอบระบบคลัสเตอร์สรุปได้ว่าการเชื่อมต่อและการสื่อสารภายในคลัสเตอร์ของ JBoss เป็นชนิด n-n นั่นคือ เมื่อโหนดใดโหนดหนึ่งในพาร์ทิชั่นมีการติดต่อจากลูกข่ายเพื่อปรับปรุงวัตถุข้อมูล โหนดนั้นจะทำการติดต่อไปยังทุกโหนดที่เหลือในพาร์ทิชั่นเพื่อทำการปรับปรุงสถานะสำเนาของวัตถุ เมื่อมีการติดต่อเข้ามาจากลูกข่ายจำนวนมากพร้อม ๆ กันในทุก ๆ โหนดในพาร์ทิชั่นหนึ่ง ๆ ทำให้ลักษณะการสื่อสารเพื่อแลกเปลี่ยนสถานะของวัตถุข้อมูลมีลักษณะเป็น n-n

2. การปรับแต่ง

การปรับแต่งคลัสเตอร์ด้วยโปรโตคอลสแต็กแบบ TCP ให้ผลดีกว่าการปรับแต่งด้วยโปรโตคอลสแต็กแบบ UDP อย่างมีนัยสำคัญ คือมีความเร็วในการตอบสนองสูงกว่าประมาณ 10-15 เท่า ทั้งนี้อาจขึ้นกับอุปกรณ์เครือข่ายเป็นสาเหตุหนึ่ง อีกประเด็นหนึ่งก็คือ กรณีของ UDP ซึ่งเป็นโปรโตคอลที่มีความเร็วสูงด้วยความสามารถทาง multicast แต่ความน่าเชื่อถือต่ำเมื่อเทียบกับ TCP อาจเกิดการสูญหายของก้อน ข้อมูลมากเกินไปในระหว่างการส่งมีความจำเป็นต้องส่งข้อมูลซ้ำหลายครั้ง จึงทำให้ประสิทธิภาพโดยรวมต่ำกว่าการปรับแต่งแบบ TCP เนื่องจากการทดลองเบื้องต้นชี้ให้เห็นว่าชนิดการสื่อสารภายใน JBoss เป็นแบบ n-n

ดังนั้นการปรับแต่งที่ควรใช้เพิ่มประสิทธิภาพระบบคลัสเตอร์ของ JBoss ขนาด 2 โหนด และ 4 โหนด คือ โปรโตคอลสแต็กแบบ TCP

3. ข้อเสนอแนะ

เนื่องจากอุปกรณ์เครือข่ายและระบบคอมพิวเตอร์ที่ใช้ในการทดสอบระบบคลัสเตอร์ และเป็นอุปกรณ์ที่ใช้งานประจำในห้องปฏิบัติการ ทำให้การทดสอบอาจมีความคลาดเคลื่อนได้ในหลายจุด เช่น คุณภาพของสายเคเบิ้ล สภาพของการ์ดเครือข่าย เป็นต้น

การทดสอบบนระบบปฏิบัติการสองชนิดคือ Windows 2000 และ Redhat Linux อาจให้ผลแตกต่างกันในระดับสังเกตได้ เนื่องจาก

1. เวอร์ช่วลแมชชีนภาษา Java ไม่ใช่ตัวเดียวกัน
2. ระบบจัดการเครือข่ายภายในระบบปฏิบัติการอาจมีโครงสร้างแตกต่างกัน

ดังนั้นการประยุกต์ใช้ระบบคลัสเตอร์ที่คาดหวังผลลัพธ์คงที่ ทุกโหนดในคลัสเตอร์ควรใช้ระบบปฏิบัติการเดียวกัน

การปรับปรุงประสิทธิภาพอื่น ๆ ของระบบคลัสเตอร์อาจทำได้โดย

1. ทำคลัสเตอร์ในระดับระบบฐานข้อมูล
2. เปลี่ยนระดับชั้นการสื่อสารภายในคลัสเตอร์โดยใช้รหัสที่เร็วขึ้น เช่น เปลี่ยน JGroups เป็น Spread Toolkits ซึ่งอาจติดกับแพล็ตฟอร์มใดแพล็ตฟอร์มหนึ่ง อย่างไรก็ตาม ความเร็วในการสื่อสารระหว่างโหนดอาจเพิ่มขึ้นอย่างเห็นได้ชัด
3. ปรับปรุงอุปกรณ์เครือข่าย
4. ปรับแก้ต้นรหัสของโปรแกรมแม่ข่ายโปรแกรมประยุกต์โดยเปลี่ยนระบบจัดการเครือข่ายจากเดิมคือ java.io เป็น java.nio
5. ใช้เทคโนโลยี Microreboot และ Recurive Recovery เพื่อเพิ่มความทนทานต่อการล่มของระบบโดยรวม
6. ใช้ตัวกระจายภาระงานระดับฮาร์ดแวร์

บรรณานุกรม

BEA Systems Inc. 2003. BEA WebLogic Server 8.1 (Online). Available URL: http://www.bea.com.

Borland Software Inc. 2003. Borland Enterprise Server AppServer Edition: Datasheet (Online). Available URL: http://www.borland.com/besappserver/pdf/besa51_datasheet.pdf

IBM Corp. 2003. WebSphere Application Server: Features and Benefits (Online). Available URL: http://www-306.ibm.com/software/webservers/appserv/was/features/

JBoss.org (2005). JBoss Support Forum: Performance Tuning (Online). Available URL: http://www.jboss.org/index.html?module=bb&op=viewforum&f=121

Roman, E., Sriganesh, R. P., and Brose, G., (2005) Mastering EJB 3rd Edition. Wiley and Sons.

Stark, S., Fleury, M., and Richards, N (2005). JBoss 4.0 – The Official Guide. Sams Publishing.

Sun Microsystem Inc. 2003. Enterprise JavaBeans Technology (Online). http://java.sun.com/products/ejb/

The ServerSide.com 2005, March. Application Server Matrix. http://www.theserverside.com/reviews/matrix.tss.

ประวัติผู้วิจัย

อาจารย์ชาญวิทย์ แก้วกสิ เป็นอาจารย์ประจำสาขาวิชาวิศวกรรมคอมพิวเตอร์ สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี จบการศึกษาระดับปริญญาตรี วิศวกรรมคอมพิวเตอร์ (เกียรตินิยมอันดับ 1) จากมหาวิทยาลัยเทคโนโลยีสุรนารี และระดับปริญญาโท วิศวกรรมคอมพิวเตอร์ จากจุฬาลงกรณ์มหาวิทยาลัย มีความเชี่ยวชาญทางวิศวกรรมซอฟต์แวร์เชิงวัตถุ (object-oriented) และเชิงลักษณะ (aspect-oriented) มีผลงานวิจัยทางวิศวกรรมซอฟต์แวร์ได้รับการตีพิมพ์ในระดับนานาชาติ และได้พัฒนาและเผยแพร่เครื่องมือเพื่อช่วยลดระยะเวลาในการพัฒนาซอฟต์แวร์อย่างต่อเนื่อง โดยเฉพาะซอฟต์แวร์บน Java และ .NET แพล็ตฟอร์ม