ระบบการคำนวณแบบกระจายบนเครือข่ายเพียร์-ทู-เพียร์สำหรับกลุ่มงานการคำนวณโดยใช้โมเดลแบบซุปเปอร์เพียร์

นายเกษม ตริตระการ

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2547
ISBN 974-53-1474-9

A PEER-TO-PEER DISTRIBUTED COMPUTING FOR COOPERATIVE COMPUTATIONAL TASKS GROUPS BASED ON SUPER-PEER MODEL

Mr. Kasame Tritrakan

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering in Computer Engineering

Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University

Academic Year 2004
ISBN 974-53-1474-9

หัวข้อวิทยานิพนธ์	ระบบการคำนวณแบบกระจายบนเครือข่ายเพียร์-ทู-เพียร์สำหรับกลุ่มงานการคำนวณโดยใช้โมเดลแบบซุปเปอร์เพียร์
โดย	นายเกษม ตริตระการ
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
อาจารย์ที่ปรึกษา	อาจารย์ ดร.วีระ เหมืองสิน

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

..... คณบดีคณะวิศวกรรมศาสตร์
(ศาสตราจารย์ ดร.ดิเรก ลาวัณย์ศิริ)

คณะกรรมการสอบวิทยานิพนธ์

..... ประธานกรรมการ
(รองศาสตราจารย์ ดร.ประภาส จงสถิตวัฒนา)

..... อาจารย์ที่ปรึกษา
(อาจารย์ ดร.วีระ เหมืองสิน)

..... กรรมการ
(อาจารย์ ดร.ณัฐวุฒิ หนูไพโรจน์)

..... กรรมการ
(ผู้ช่วยศาสตราจารย์ ดร.ภุชงค์ อุทโยภาส)

เกษม ตริตระการ : ระบบการคำนวณแบบกระจายบนเครือข่ายเพียร์-ทู-เพียร์สำหรับกลุ่มงานการคำนวณโดยใช้โมเดลแบบซุปเปอร์เพียร์. (A PEER-TO-PEER DISTRIBUTED COMPUTING FOR COOPERATIVE COMPUTATIONAL TASKS GROUPS BASED ON SUPER-PEER MODEL) อ. ที่ปรึกษา : อ.ดร. วีระ เหมืองสิน, 77 หน้า. ISBN 974-53-1474-9.

ระบบการคำนวณแบบกระจายบนอินเตอร์เน็ตในปัจจุบันมีข้อจำกัดในด้านของความสามารถในการขยายระบบ โดยเฉพาะอย่างยิ่งเมื่อแอพพลิเคชั่นที่ทำงานบนระบบนั้นมีขนาดของข้อมูลที่ใหญ่เมื่อเทียบกันเวลาที่ใช้ในการคำนวณ เป็นเหตุให้ภาระของเครื่องเซิร์ฟเวอร์มากเกินไปโดยเฉพาะภาระในการรับ-ส่งข้อมูลระหว่างเครื่องเซิร์ฟเวอร์กับเครื่องไคลเอนต์

วิทยานิพนธ์ฉบับนี้มีจุดประสงค์เพื่อหาวิธีปรับปรุงการทำงานของระบบการคำนวณแบบกระจายบนเครือข่ายเพียร์-ทู-เพียร์ โดยเน้นที่ความสามารถในการขยายระบบและการรองรับแอพพลิเคชั่นที่มีสัดส่วนของขนาดข้อมูลต่อเวลาที่ใช้ในการคำนวณสูง

วิทยานิพนธ์ฉบับนี้ได้ทำการศึกษาวิเคราะห์ถึงปัจจัยที่มีผลต่อประสิทธิภาพความสามารถในการขยายระบบและขนาดของข้อมูลที่ระบบสามารถรองรับได้ โดยใช้โปรแกรมแบบจำลอง ซึ่งจากผลการทดลองแสดงให้เห็นว่าการสื่อสารแบบเพียร์-ทู-เพียร์ช่วยลดปริมาณการใช้แบนด์วิดธ์ของเครื่องเซิร์ฟเวอร์ลงเมื่อเทียบกับระบบที่ไม่ได้ใช้การสื่อสารแบบเพียร์-ทู-เพียร์ โดยปัจจัยที่มีผลต่อประสิทธิภาพที่สำคัญคือความเสถียรของเครือข่ายแบบเพียร์-ทู-เพียร์ของระบบ จึงได้ทำการออกแบบโครงสร้างของระบบโดยใช้โมเดลแบบซุปเปอร์เพียร์เพื่อแก้ปัญหาที่พบ จากนั้นได้พัฒนาระบบต้นแบบขึ้นโดยใช้โพรโตคอล JXTA เพื่อทดสอบแนวคิดที่นำเสนอ

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต .....

สาขาวิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่ออาจารย์ที่ปรึกษา .....

ปีการศึกษา 2547

457 02266 21 : MAJOR COMPUTER ENGINEERING

KEYWORD: PEER-TO-PEER COMMUNICATION / SUPER-PEER / HIGH PERFORMANCE COMPUTING / JXTA PROTOCOL

KASAME TRITRAKAN : A PEER-TO-PEER DISTRIBUTED COMPUTING FOR COOPERATIVE COMPUTATIONAL TASKS GROUPS BASED ON SUPER-PEER MODEL. THESIS ADVISOR : VEERA MUANGSIN, Ph.D., 77 pp. ISBN 974-53-1474-9.

At present, distributed computing systems on the Internet still have limitations on scalability. This is especially the case when the application consumes or produces a large amount of data. In the current client/server model, the server can be overloaded by the data to be transferred to and from computing machines.

This thesis aims to improve the scalability and capability to handle data intensive applications of a distributed computing system on a peer-to-peer network.

In this thesis, the evaluation of many factors that directly affect the performance of a distributed computing system is performed by simulation. The results of the study have shown that applying peer-to-peer communication can reduce the usage of server's bandwidth. The important factor that affects the performance of the system is the availability of peer-to-peer communication between job submitters and computing machines. Therefore, a super-peer model is employed to further enhance the performance of the system. A prototype of the proposed system is implemented by using the JXTA protocol.

Department Computer Engineering Student's signature .....

Field of study Computer Engineering Advisor's signature .....

Academic year 2004

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงไปได้ด้วยดี เนื่องจากผู้วิจัยได้รับคำแนะนำและคำปรึกษาต่างๆ ที่เป็นประโยชน์อย่างยิ่งของอาจารย์ ดร.วีระ เหมืองสิน อาจารย์ที่ปรึกษาวิทยานิพนธ์ ผู้วิจัยจึงขอกราบขอบพระคุณเป็นอย่างสูง

ขอกราบขอบพระคุณ บิดาและมารดาที่ช่วยให้กำลังใจและเตือนสติให้ผู้วิจัยกระทำในสิ่งที่ควรทำ และให้คำปรึกษาเมื่อมีปัญหา

ขอขอบคุณ นางสาวพีริยา คมสาคร และพี่สาวที่คอยให้กำลังใจและกดดันให้เร่งทำวิจัยอยู่ตลอดเวลา อีกทั้งเอาขนมอร่อยๆ มาให้เพื่อเป็นพลังในการทำวิจัย

สุดท้ายขอขอบคุณเพื่อนๆ นิสิตปริญญาโททุกคน ที่ได้คอยช่วยเหลือและตอบข้อสงสัยในการทำวิจัยแก่ผู้ทำวิจัย

สารบัญ

หน้า

บทคัดย่อภาษาไทย ง

บทคัดย่อภาษาอังกฤษ จ

กิตติกรรมประกาศ ฉ

สารบัญ ช

สารบัญตาราง ญ

สารบัญภาพ ฎ

บทที่ 1 บทนำ 1

1.1 ความเป็นมา 1

1.2 ปัญหาที่พบ 2

1.3 แนวทางการแก้ปัญหา 3

1.4 วัตถุประสงค์ 3

1.5 ขั้นตอนและวิธีการดำเนินงาน 3

1.6 ขอบเขตของงานวิจัย 3

1.7 ประโยชน์ที่คาดว่าจะได้รับจากงานวิจัย 4

1.8 โครงสร้างของวิทยานิพนธ์ 4

1.9 ผลงานที่ได้รับการตีพิมพ์จากวิทยานิพนธ์ 4

บทที่ 2 ทฤษฎีและงานวิจัยที่เกี่ยวข้อง 6

2.1 ระบบการคำนวณแบบกระจายแบบ Cycle Scavenger บนเครือข่ายท้องถิ่น 6

2.2 ระบบการคำนวณแบบกระจายบนอินเตอร์เน็ต 7

2.2.1 ระบบที่สร้างงานขึ้นมาจากแหล่งเดียว (Single site submission) 7

2.2.2 ระบบที่รับงานมาจากหลายแหล่ง (Multiple sites submission) 8

2.3 ระบบการคำนวณแบบกระจายบนเครือข่ายเพียร์-ทู-เพียร์ 12

2.4 เครือข่ายแบบเพียร์-ทู-เพียร์ 13

2.4.1 โมเดลแบบ Pure P2P 14

2.4.2 โมเดลแบบ Hybrid P2P 15

2.4.3 โมเดลแบบ Super peer 16

2.5 โพรโตคอล JXTA 16

สารบัญ (ต่อ)

หน้า

บทที่ 3 การวิเคราะห์ประสิทธิภาพของระบบการคำนวณแบบกระจายที่ใช้การสื่อสารแบบเพียร์-ทู-เพียร์ 18
3.1 ระบบการคำนวณแบบกระจายแบบไคลเอนต์/เซิร์ฟเวอร์ 18
3.2 ระบบการคำนวณแบบกระจายแบบเพียร์-ทู-เพียร์ 19
3.3 การศึกษาโดยใช้แบบจำลอง 21
3.4 รูปแบบการทดลอง 23
3.5 ผลการทดลองและวิเคราะห์ผล 23
3.5.1 ผลกระทบจากขนาดไฟล์เอาต์พุต 23
3.5.2 ผลกระทบจากความไม่เสถียรของเครือข่ายแบบเพียร์-ทู-เพียร์ 25
3.5.3 ผลกระทบจากจำนวนเครื่องที่รับงานไปประมวลผลในระบบ 26
3.5.4 ผลกระทบจากขนาดแบนด์วิดธ์ของเครื่องเซิร์ฟเวอร์ 28
3.6 สิ่งที่ได้รับจากการทดลอง 29

บทที่ 4 ระบบการคำนวณแบบกระจายบนเครือข่ายแบบเพียร์-ทู-เพียร์โดยใช้โมเดลแบบซุปเปอร์เพียร์ 30
4.1 การใช้โมเดลแบบซุปเปอร์เพียร์เพื่อเพิ่มประสิทธิภาพของระบบการคำนวณแบบกระจาย 30
4.2 โครงสร้างและรูปแบบการทำงานของระบบที่ใช้โมเดลแบบซุปเปอร์เพียร์ 31
4.2.1 Advertisement 32
4.2.2 SuperPeer 32
4.2.3 Volunteer 33
4.2.4 รูปแบบของงาน 34
4.3 การทำงานของระบบ 34
4.3.1 รูปแบบการทำงาน 34
4.3.2 การเลือก SuperPeer 37
4.4 การประเมินประสิทธิภาพ 38

บทที่ 5 ระบบต้นแบบ 39
5.1 รูปแบบการทำงานของระบบต้นแบบ 39

สารบัญ (ต่อ)

หน้า

5.2 ส่วนประกอบของระบบ 39
5.2.1 ส่วนประกอบของ SuperPeer 39
5.2.2 ส่วนประกอบของ Volunteer 41
5.2.3 Resource advertisement 43
5.2.4 Job advertisement 45
5.3 การส่งงานเข้าสู่ระบบ 46
5.3.1 วิธีการส่งงาน 46
5.3.2 วิธีดูสถานะของงาน 47
5.3.3 ผลลัพธ์ของงาน 48

บทที่ 6 สรุปผลการวิจัย 49
6.1 สรุปผลการวิจัย 49
6.2 ข้อเสนอแนะและแนวทางในการวิจัยต่อ 50

รายการอ้างอิง 51
ภาคผนวก 53
ภาคผนวก ก. โพรโตคอล JXTA 54
ภาคผนวก ข. ผลงานที่ได้รับการตีพิมพ์จากวิทยานิพนธ์ 64
ประวัติผู้เขียนวิทยานิพนธ์ 77

สารบัญตาราง

ตาราง หน้า

3.1 ค่าตัวแปรต่างๆ ที่ใช้ในการทดลองเพื่อวัดประสิทธิภาพของระบบ 22

5.1 ตัวอย่างข้อมูลที่อยู่ใน Resource advertisement 44

5.2 ตัวอย่างข้อมูลที่อยู่ใน Job advertisement 46

สารบัญภาพ

ภาพประกอบ หน้า

2.1 โครงสร้างของระบบ Condor 6
2.2 โครงสร้างของ SETI@home 8
2.3 โครงสร้างของ Frontier 9
2.4 โครงสร้างของ Javelin 10
2.5 ขั้นตอนการทำงานของ Javelin 11
2.6 โครงสร้างแบบไคลเอนต์/เซิร์ฟเวอร์ 13
2.7 โครงสร้างแบบเพียร์-ทู-เพียร์ 14
2.8 โมเดลแบบ Pure P2P 15
2.9 โมเดลแบบ Hybrid P2P 15
2.10 โครงสร้างแบบ Super peer 16
2.11 โครงสร้างของโพรโตคอล JXTA 17
3.1 รูปแบบการทำงานที่ใช้การสื่อสารแบบไคลเอนต์/เซิร์ฟเวอร์ 18
3.2 รูปแบบการทำงานที่ใช้การสื่อสารแบบเพียร์-ทู-เพียร์ 20
3.3 ผลกระทบของขนาดเอาต์พุตที่ค่า $P = 1.0$ 24
3.4 ผลกระทบของขนาดเอาต์พุตที่ค่า $0.0 \le P < 1.0$ 25
3.5 ผลกระทบจากความไม่เสถียรของเครือข่ายแบบเพียร์-ทู-เพียร์ 26
3.6 ผลกระทบของจำนวนเครื่องที่รับงานไปประมวลผลต่อค่า Response time เฉลี่ย 26
3.7 ผลกระทบของจำนวนเครื่องที่รับงานไปประมวลผลต่อปริมาณข้อมูลที่เครื่องเซิร์ฟเวอร์ต้องแบกรับ ณ เวลาหนึ่งๆ 27
3.8 ผลกระทบของค่าจำนวนเครื่องที่รับงานไปประมวลผลต่ออัตราเร็วในการส่งข้อมูลของระบบโดยเฉลี่ย 27
3.9 ผลกระทบของจำนวนเครื่องที่รับงานไปประมวลผลต่อความยาวเฉลี่ยของคิวงาน 27
3.10 ผลกระทบของขนาดแบนด์วิดธ์ของเครื่องเซิร์ฟเวอร์ 29
4.1 โครงสร้างโดยรวมของระบบที่ใช้โมเดลแบบซุปเปอร์เพียร์ 31
4.2 การทำงานในรูปแบบการทำงานที่ 1 35
4.3 การทำงานในรูปแบบการทำงานที่ 2 36
5.1 ส่วนประกอบของ SuperPeer 39
5.2 ส่วนประกอบของ Volunteer 41
5.3 การเลือกโปรแกรมถนอมหน้าจอ JScreenSaver 42
5.4 โปรแกรมถนอมหน้าจอ JScreenSaver 42

สารบัญภาพ (ต่อ)

ภาพประกอบ		หน้า
5.5	ตัวอย่างไฟล์ตั้งค่าระบบ	43
5.6	ตัวอย่างไฟล์แสดงรายละเอียดของงาน	45
5.7	โปรแกรมส่วนรับคำสั่งและคำสั่งเพื่อส่งงานเข้าสู่ระบบ	47
5.8	การดูสถานะของงาน	47
ก.1	Point-to-point and propagate pipes	57
ก.2	ตัวอย่าง Peer advertisement	58
ก.3	การกระจาย Request ใน JXTA	59
ก.4	ตัวอย่างโค้ดสำหรับสร้าง Pipe advertisement จากไฟล์	60
ก.5	ตัวอย่างโค้ดสำหรับการสร้าง Pipe advertisement ใหม่	60
ก.6	ตัวอย่างโค้ดสำหรับประกาศ Pipe advertisement ทั้งแบบ local และ remote	61
ก.7	ตัวอย่างโค้ดสำหรับหาเพียร์ที่ชื่อขึ้นต้นด้วยคำว่า “Peer”	61
ก.8	ตัวอย่างโค้ดสำหรับสร้าง Input pipe และตัว PipeMsgListener	62

บทที่ 1

บทนำ

1.1 ความเป็นมา

ในปัจจุบันความต้องการพลังการประมวลของเครื่องคอมพิวเตอร์ได้เพิ่มขึ้นอย่างรวดเร็ว ทั้งในการออกแบบทางวิศวกรรม, การวิเคราะห์ธุรกิจ, การสร้างภาพยนตร์อนิเมชัน และโดยเฉพาะยิ่งในการวิจัยด้านทางวิทยาศาสตร์ งานเหล่านี้ต้องใช้เวลาในการประมวลผลที่นานมาก รวมถึงต้องใช้หน่วยความจำหลักและหน่วยความจำสำรองขนาดใหญ่ ซึ่งเราเรียกงานเหล่านี้ว่า "*ปัญหาที่มีความท้าทายสูง (Grand Challenge Problems)*" จึงได้มีการสร้างเครื่องคอมพิวเตอร์สมรรถนะสูงหรือที่เรียกกันว่าซุปเปอร์คอมพิวเตอร์ขึ้นมาใช้กับงานดังกล่าว แต่เนื่องจากเครื่องซุปเปอร์คอมพิวเตอร์นั้นมีราคาและค่าใช้จ่ายในการดูแลรักษาที่สูงมาก จึงเกิดแนวคิดในการสร้างเครื่องคอมพิวเตอร์สมรรถนะสูงขึ้นโดยใช้เครื่องคอมพิวเตอร์ขนาดเล็กจำนวนมากมาทำงานร่วมกัน โดยแบ่งงานออกเป็นงานย่อยๆ เพื่อส่งไปให้เครื่องคอมพิวเตอร์เหล่านั้นช่วยกันประมวลผลเพื่อให้ได้ผลลัพธ์ที่เร็วขึ้น ซึ่งเรียกการทำงานแบบนี้ว่า "*การคำนวณแบบขนาน (Parallel computing)*" หรือ "*การคำนวณแบบกระจาย (Distributed computing)*"

แนวทางในการสร้างระบบการคำนวณแบบกระจายที่เป็นที่นิยมคือการใช้เครื่องคอมพิวเตอร์ที่ทำงานได้โดยลำพัง (Stand-alone computer) เช่น คอมพิวเตอร์ส่วนบุคคล (PC) หรือ เครื่องเวิร์กสเตชั่น (Workstation) ที่มีราคาถูกจำนวนมากมาช่วยกันประมวลผลงาน ซึ่งทำให้ต้นทุนของระบบดังกล่าวต่ำกว่าการใช้เครื่องซุปเปอร์คอมพิวเตอร์ที่มีสมรรถนะเท่ากัน

การสร้างระบบการคำนวณแบบกระจายโดยใช้เครื่องคอมพิวเตอร์ที่ทำงานได้โดยลำพังได้แบ่งออกเป็น 2 แนวคิดคือ แบบ *Dedicated Cluster* [1] ที่เครื่องคอมพิวเตอร์ในระบบมีทำหน้าที่ประมวลผลงานจากระบบเท่านั้น ตัวอย่างระบบแบบ Dedicated Cluster ที่เป็นที่นิยมมากที่สุดคือ Beowulf ของ NASA [2] เป็นต้น ส่วนอีกแนวคิดหนึ่งคือ แบบ *Cycle Scavenger* ซึ่งอาศัยพลังในการประมวลผลจากเครื่องคอมพิวเตอร์ส่วนบุคคลในขณะที่ไม่มีผู้ใช้งาน เช่นผู้ใช้อาจไปประชุม, รับประทานอาหาร หรือนอนหลับ เป็นต้น มาใช้ในการประมวลผลงานแทนที่จะปล่อยให้พลังในการประมวลผลเหล่านั้นสูญเสียไปโดยเปล่าประโยชน์ ซึ่งแนวคิดนี้สามารถช่วยประหยัดค่าใช้จ่ายทั้งการซื้อเครื่องและการดูแลรักษาระบบได้เป็นอย่างดี โดยแนวคิดนี้เริ่มมาจากโครงการ NOW (Network of workstations) [3] และ Condor [4] [5] ซึ่งเป็นระบบที่ใช้พลังการประมวลผลจากเครื่องคอมพิวเตอร์ส่วนบุคคลที่อยู่ภายในองค์กร

นอกจากนั้นแนวคิดการคำนวณแบบกระจายยังได้ขยายออกไปสู่อินเตอร์เน็ต หรือที่เรียกกันว่า "*การคำนวณผ่านอินเตอร์เน็ต (Internet computing)*" ซึ่งสามารถใช้พลังการประมวลผลจากเครื่องคอมพิวเตอร์ส่วนบุคคลที่มีอยู่มากมายบนเครือข่ายอินเตอร์เน็ตมาประมวลผลงานบางอย่าง ตัวอย่างระบบที่ทำงานแบบนี้ได้แก่ SETI@home [6] [7], FightAIDS@Home [8], Folding@Home [9], และ Distributed.net [10] เป็นต้น ระบบเหล่านี้จะประกอบด้วยเครื่องเซิร์ฟเวอร์กลางที่ทำหน้าที่สร้างงาน

จำนวนมาก และเครื่องคอมพิวเตอร์ส่วนบุคคลที่บริจาคทรัพยากรการคำนวณของตนให้แก่ระบบที่ติดตั้งโปรแกรมสำหรับดาวน์โหลดงานจากระบบมาประมวลผลเมื่อเครื่องไม่มีการใช้งานและส่งผลลัพธ์กลับไปยังเครื่องเซิร์ฟเวอร์กลาง ด้วยวิธีนี้จึงทำให้ SETI@home เป็นระบบคอมพิวเตอร์ที่มีสมรรถนะสูงที่สุดในโลก [6]

อย่างไรก็ตามเครื่องคอมพิวเตอร์ที่เข้าร่วมในระบบแบบ SETI@home จะเป็นฝ่ายที่รับงานมาประมวลผลเพียงฝ่ายเดียว ไม่สามารถส่งงานของตนไปให้เครื่องอื่นๆ ช่วยประมวลผลได้ และงานที่ทำรวมถึงโปรแกรมที่ใช้ก็จะมีเพียงแอพพลิเคชั่นเดียวเท่านั้น ซึ่งในการทำงานจริงนั้นมีผู้ใช้จำนวนมากที่ต้องการพลังในการประมวลผลเพื่อใช้กับงานของตนแต่ไม่สามารถสร้างระบบที่มีขนาดใหญ่และมีชื่อเสียงพอที่จะดึงดูดให้คนมาบริจาคทรัพยากรการคำนวณของตนให้แก่ตนได้ ดังนั้นเพื่อให้ผู้ใช้แต่ละคนสามารถส่งงานของตนไปให้เครื่องใดก็ได้ในระบบอินเตอร์เน็ตช่วยประมวลผล แนวคิดของเครือข่ายแบบเพียร์-ทู-เพียร์ [11] ซึ่งนิยมนำไปใช้กับแอพพลิเคชั่นสำหรับการแลกเปลี่ยนไฟล์ (File sharing) จึงได้ถูกนำมาใช้เรียกว่า "*การคำนวณแบบเพียร์-ทู-เพียร์ (Peer-to-Peer computing)*" [12] ระบบการคำนวณแบบกระจายบนเครือข่ายแบบเพียร์-ทู-เพียร์มีศักยภาพในการรวบรวมทรัพยากรของเครื่องคอมพิวเตอร์จำนวนมากเข้ามาทำงานร่วมกัน โดยเครื่องคอมพิวเตอร์เครื่องหนึ่งสามารถเป็นได้ทั้งผู้ที่ส่งงานเข้าสู่ระบบและผู้ที่รับงานจากระบบไปประมวลผล ระบบนี้จึงน่าจะเป็นประโยชน์อย่างมาก ตัวอย่างของระบบแบบนี้ได้แก่ Frontier [13], Javelin [14], และ JNGI [15] เป็นต้น ซึ่งจะกล่าวถึงโครงสร้างและรายละเอียดการทำงานของระบบเหล่านี้ในบทที่ 2

1.2 ปัญหาที่พบ

ระบบการคำนวณแบบกระจายบนเครือข่ายเพียร์-ทู-เพียร์ที่มีอยู่ในปัจจุบันมักจะมีเซิร์ฟเวอร์กลางที่ทำหน้าที่คอยรับงานจากผู้ใช้และกระจายงานไปยังเครื่องต่างๆ ในระบบ ดังนั้นหากระบบมีเครื่องคอมพิวเตอร์และผู้ใช้งานจำนวนมากๆ เครื่องเซิร์ฟเวอร์กลางจะต้องแบกรับภาระงาน (Load) ที่มากขึ้นไปด้วยทั้งในเรื่องของการจัดลำดับงาน (Scheduling), การติดต่อสื่อสารกับเครื่องคอมพิวเตอร์เครื่องอื่นๆ และที่สำคัญคือการทำหน้าที่เป็นตัวกลางในการรับ-ส่งไฟล์ข้อมูลระหว่างเครื่องคอมพิวเตอร์ในระบบ หากภาระงานดังกล่าวมากเกินกว่าความสามารถของเครื่องเซิร์ฟเวอร์ก็จะทำให้เกิดปัญหาคอขวดขึ้นที่เครื่องเซิร์ฟเวอร์ซึ่งทำให้ประสิทธิภาพในการทำงานของระบบลดลงอย่างรวดเร็ว ดังนั้นระบบจึงมีขีดจำกัดในด้านความสามารถในการรองรับการขยายขนาดของระบบและขนาดของไฟล์ข้อมูล

อีกปัญหาหนึ่งคือ ถ้าเครื่องเซิร์ฟเวอร์กลางเกิดเสียไม่สามารถให้บริการเครื่องคอมพิวเตอร์ในระบบได้ ก็จะทำให้ทั้งระบบหยุดการทำงานไปด้วย นั่นคือระบบมี Single point of failure

งานวิจัยมุ่งเน้นที่ปัญหาที่เกี่ยวข้องกับการขยายขนาดของระบบการคำนวณแบบกระจายบนเครือข่ายเพียร์-ทู-เพียร์ซึ่งมีสาเหตุจากการที่เครื่องเซิร์ฟเวอร์รับภาระงานมากเกินไป โดยเฉพาะอย่างยิ่ง

ภาระในการรับส่งข้อมูล และการรองรับแอพพลิเคชั่นที่มีขนาดของข้อมูลใหญ่ รวมถึงปัญหา Single point of failure ด้วย

1.3 แนวทางการแก้ปัญหา

เพื่อเป็นการแก้ปัญหาที่ได้กล่าวมาข้างต้น จึงต้องทำการวิเคราะห์ถึงปัญหาที่เกิดขึ้นบนระบบการคำนวณแบบกระจายบนเครือข่ายเพียร์-ทู-เพียร์ และทำการออกแบบโครงสร้างของระบบโดยการนำโมเดลแบบซุปเปอร์เพียร์เข้ามาใช้

1.4 วัตถุประสงค์

งานวิจัยนี้มีจุดประสงค์เพื่อต้องการวิเคราะห์ถึงปัญหาของระบบการคำนวณแบบกระจายบนเครือข่ายเพียร์-ทู-เพียร์ และทำการปรับปรุงระบบโดยใช้เทคนิคของซุปเปอร์เพียร์เพื่อแก้ปัญหาด้านความสามารถในการขยายระบบและไม่สามารถรองรับแอพพลิเคชั่นที่มีข้อมูลขนาดใหญ่ได้อันเนื่องมาจากภาระงานของเครื่องเซิร์ฟเวอร์ที่มากเกินไป

1.5 ขั้นตอนและวิธีการดำเนินงาน

1. ศึกษาทฤษฎีและงานวิจัยที่เกี่ยวข้อง
2. สร้างแบบจำลองระบบการคำนวณแบบกระจายบนเครือข่ายเพียร์-ทู-เพียร์
3. ทดสอบประสิทธิภาพการทำงานของระบบโดยใช้โปรแกรมแบบจำลอง
4. ออกแบบระบบโดยใช้โมเดลแบบซุปเปอร์เพียร์
5. สร้างระบบต้นแบบและทดสอบการทำงาน
6. สรุปผลการวิจัย
7. จัดทำวิทยานิพนธ์

1.6 ขอบเขตของงานวิจัย

1. จะใช้โปรแกรมแบบจำลอง (Simulator) ในการจำลองระบบที่มีขนาดใหญ่เพื่อวัดประสิทธิภาพการทำงาน
2. จะพัฒนาระบบต้นแบบขึ้นเพื่อทดสอบการทำงานตามแนวคิดที่นำเสนอ
3. การออกแบบระบบจะมุ่งเน้นการแก้ปัญหาการจัดการงานและการรับส่งข้อมูลมากกว่าประเด็นอื่นๆ เช่น ความปลอดภัย และความทนทานของระบบ

1.7 ประโยชน์ที่คาดว่าจะได้รับ

สามารถนำระบบที่ออกแบบไปสร้างระบบการคำนวณแบบกระจายบนเครือข่ายเพียร์-ทู-เพียร์ เพื่อนำไปใช้กับงานทางวิทยาศาสตร์หรือการคำนวณต่างๆ ที่ต้องการเครื่องคอมพิวเตอร์จำนวนมากมาช่วยในการประมวลผล โดยระบบที่ได้จะมีความสามารถในการขยายระบบและสามรถรองรับแอพพลิเคชั่นที่ข้อมูลมีขนาดใหญ่ได้

1.8 โครงสร้างของวิทยานิพนธ์

วิทยานิพนธ์ฉบับนี้แบ่งเนื้อหาออกเป็น 6 บทด้วยกันคือ

บทที่ 1 เป็นบทนำซึ่งจะกล่าวถึงความเป็นมาของระบบการคำนวณแบบกระจาย ปัญหาที่พบ แนวทางการแก้ปัญหาที่พบ วัตถุประสงค์ของวิทยานิพนธ์ ขั้นตอนในการดำเนินการวิจัย ขอบเขตของงานวิจัย และประโยชน์ที่คาดว่าจะได้รับ

บทที่ 2 เป็นการสรุปถึงทฤษฎีและงานวิจัยต่างๆ ที่เกี่ยวข้องได้แก่ ระบบการคำนวณแบบกระจายที่อาศัยพลังการประมวลผลจากเครื่องคอมพิวเตอร์ที่ไม่มีผู้ใช้งาน, ลักษณะโครงสร้างและรูปแบบการทำงานของเครือข่ายแบบเพียร์-ทู-เพียร์ และโพรโตคอล JXTA

บทที่ 3 เป็นการนำเสนอรูปแบบการประยุกต์ใช้การสื่อสารแบบเพียร์-ทู-เพียร์กับระบบการคำนวณแบบกระจายเพื่อเพิ่มประสิทธิภาพการทำงานของระบบ และได้แสดงถึงประสิทธิภาพที่เพิ่มขึ้นของของระบบการคำนวณแบบกระจายหลังจากการนำรูปแบบการสื่อสารแบบเพียร์-ทู-เพียร์ไปประยุกต์ใช้ และผลกระทบของปัจจัยต่างๆ ที่สำคัญต่อประสิทธิภาพการทำงาน

บทที่ 4 นำเสนอโครงสร้างแบบซุปเปอร์เพียร์ และรายละเอียดต่างๆ ของระบบที่ออกแบบในงานวิจัยนี้

บทที่ 5 นำเสนอการนำโครงสร้างที่ออกแบบในบทที่4 ไปอิมพลีเมนต์กับโพรโตคอล JXTA และภาษา Java

บทที่ 6 สรุปผลการวิจัย

1.9 ผลงานที่ได้รับการตีพิมพ์จากวิทยานิพนธ์

ส่วนหนึ่งของงานวิทยานิพนธ์นี้ได้รับการตีพิมพ์เป็นบทความทางวิชาการจำนวน 2 ฉบับ ดังนี้

1. หัวข้อ “การใช้การสื่อสารแบบเพียร์-ทู-เพียร์บนระบบการคำนวณแบบกระจายเพื่อลดปัญหาคอขวดบนเครื่องเซิร์ฟเวอร์ (Using Peer-to-Peer Communication for Distributed Computing System to Reduce Network Bottleneck on Server)” โดย นายเกษม ตริตระการ และ อ.ดร.วีระเหมืองสิน ในงานประชุมวิชาการ “The 8th National Computer Science and Engineering Conference (NCSEC 2004)” ซึ่งจัด

โดย คณะวิศวกรรมศาสตร์ มหาวิทยาลัยสงขลานครินทร์ ณ โรงแรม J.B. หาดใหญ่ ในวันที่ 21-22 ตุลาคม 2547

2. หัวข้อ “Using Peer-to-Peer Communication to Improve the Performance of Distributed Computing on the Internet” โดยนายเกษม ตริตระการ และ อ.ดร.วีระ เหมืองสิน ในงานประชุมวิชาการ “The IEEE 19th International Conference on Advanced Information Networking and Applications (AINA 2005)” ซึ่งจัดโดย IEEE Computer Society TCDP ณ Tamkang University ประเทศไต้หวัน ในวันที่ 28-30 มีนาคม 2548

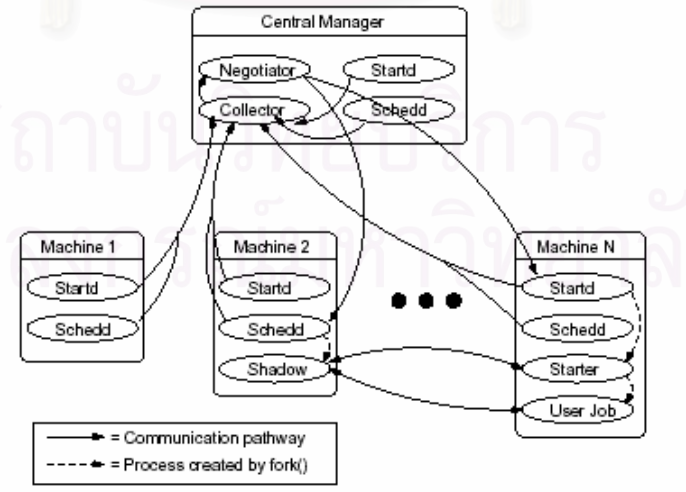
บทที่ 2

ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 ระบบการคำนวณแบบกระจายแบบ Cycle Scavenger บนเครือข่ายท้องถิ่น

ระบบการคำนวณแบบกระจายที่อาศัยพลังการประมวลผลจากเครื่องคอมพิวเตอร์ส่วนบุคคลในขณะที่ไม่มีผู้ใช้งานหรือที่เรียกว่า “Cycle Scavenger Distributed Computing System” เป็นระบบการคำนวณแบบกระจายแบบหนึ่งที่กำลังเป็นที่นิยมอย่างมากในปัจจุบัน เนื่องจากมีจุดเด่นในเรื่องของการประหยัดค่าใช้จ่ายในการซื้อเครื่องคอมพิวเตอร์และการดูแลรักษา แนวคิดในการใช้ประโยชน์จากเครื่องคอมพิวเตอร์ในขณะที่เครื่องว่างเกิดขึ้นมานานแล้ว โดยเริ่มจากโครงการ NOW (Network of workstations) [3] ซึ่งเป็นระบบที่มีการจัดลำดับงานแบบกลุ่มกระจาย (Distributed batch scheduling) ที่มุ่งเน้นให้ประสิทธิผล (Throughput) ของระบบสูงที่สุด โดยทำการรวบรวมพลังการประมวลผลที่สูญเสียไปจากการปล่อยให้เครื่องว่างของเครื่องคอมพิวเตอร์ที่อยู่ภายในเครือข่ายเดียวกันเพื่อใช้ในการประมวลผลงานที่ต้องใช้เครื่องคอมพิวเตอร์จำนวนมากๆ โดยมักมีเครื่องเซิร์ฟเวอร์กลางที่เรียกว่า Central manager ทำหน้าที่เป็นศูนย์กลางคอยทำหน้าที่รับงานจากผู้ใช้, จัดลำดับงาน, หาเครื่องที่เหมาะสมที่จะรับงานไปทำ, และส่งงานไปประมวลผลยังเครื่องที่เลือกไว้ ส่วนเครื่องไคลเอนต์จะมีโพรเซสที่คอยตรวจสอบว่าขณะนี้มีผู้ใช้งานอยู่หรือไม่ ถ้าพบว่าเครื่องว่างก็จะไปรับงานจากระบบมาประมวลผลและส่งผลลัพธ์กลับไปยังเครื่องเซิร์ฟเวอร์

ระบบในกลุ่มนี้ที่ได้รับความนิยมที่สุดคือ Condor [4] [5] ซึ่งมีการใช้เทคนิคการทำจุดตรวจสอบ (Check point) และการเคลื่อนย้ายโพรเซส (Process migration) เพื่อเพิ่มประสิทธิผลของระบบ



รูปที่ 2.1 โครงสร้างของระบบ Condor

ระบบเช่นนี้จะสามารถให้ประสิทธิผลที่ค่อนข้างสูง และลดค่าใช้จ่ายในการซื้อเครื่องคอมพิวเตอร์และค่าดูแลรักษาได้เป็นอย่างดี แต่อย่างไรก็ตามระบบสามารถรองรับการจัดการงานจากภายในเครือข่ายองค์กรเดียวกันเท่านั้น และมีปัญหา Single point of failure เนื่องจากหากเครื่องเซิร์ฟเวอร์ไม่สามารถให้บริการได้ ระบบก็จะไม่สามารถทำงานต่อไปได้ นอกจากนั้นในระบบ Condor จะมีไฟล์จุดตรวจสอบที่ใหญ่ (ประมาณ 100 MB) ทำให้ต้องทำงานภายใต้ระบบเครือข่ายแบบท้องถิ่นหรือเครือข่ายความเร็วสูงเท่านั้น ทำให้ระบบอย่างเช่น Condor จึงไม่เหมาะที่จะนำมาใช้บนเครือข่ายอินเตอร์เน็ต

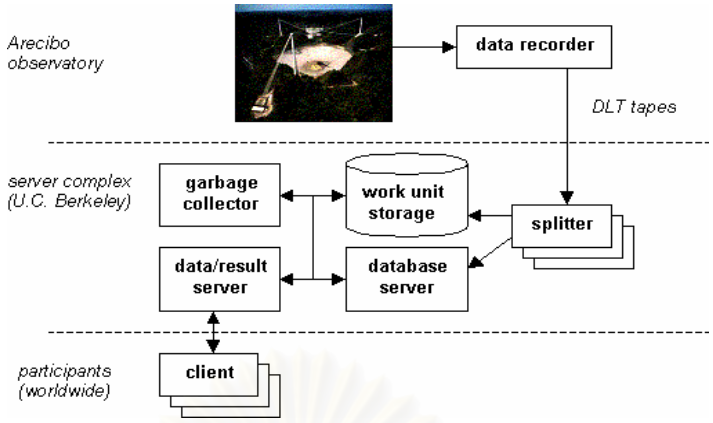
2.2 ระบบการคำนวณแบบกระจายบนอินเตอร์เน็ต

ระบบประเภทนี้จะอาศัยพลังการประมวลผลจากเครื่องคอมพิวเตอร์ที่อยู่ภายในเครือข่ายอินเตอร์เน็ต โดยสามารถแบ่งเป็นประเภทย่อยๆ ได้แก่

2.2.1 ระบบที่สร้างงานขึ้นมาจากแหล่งเดียว (Single site submission)

ระบบประเภทนี้จะมีแอพพลิเคชั่นที่เป็นที่ดึงดูดใจจนสามารถทำให้มีคนสนใจและต้องการบริจาคทรัพยากรของเครื่องตนให้แก่ระบบ ระบบประเภทนี้จะมีเครื่องเซิร์ฟเวอร์ที่ทำหน้าที่สร้างงานจำนวนมากและส่งแต่ละงานไปประมวลผลยังเครื่องสมาชิก ส่วนเครื่องสมาชิกจะมีหน้าที่รับงานจากเซิร์ฟเวอร์มาประมวลผลเท่านั้น โดยโครงการที่ประสบความสำเร็จและทำให้เกิดแนวคิดนี้คือ SETI@home [6] [7] ซึ่งเป็นโครงการที่มีจุดประสงค์เพื่อค้นหาสัญญาณที่คาดว่าจะมาจากสิ่งมีชีวิตนอกโลก โดยใช้พลังในการประมวลผลจากเครื่องคอมพิวเตอร์ส่วนบุคคลที่ไม่มีผู้ใช้งานผ่านระบบอินเตอร์เน็ต SETI@home มีโครงสร้างดังรูปที่ 2.2 โดยจะมีเครื่องเซิร์ฟเวอร์กลางที่ทำหน้าที่แบ่งข้อมูลสัญญาณวิทยุออกเป็นส่วนๆ ที่สามารถประมวลผลได้บนเครื่องคอมพิวเตอร์ส่วนบุคคลธรรมดา ผู้ใช้ที่เป็นอาสาสมัครจะต้องลงโปรแกรมที่ทำหน้าที่คล้ายๆ กับโปรแกรมถนอมหน้าจอ (Screen Saver) บนเครื่องของตน โดยโปรแกรมนี้จะทำงานอยู่เป็นโพรเซสเบื้องหลัง (Background process) เมื่อใดก็ตามที่เครื่องนั้นไม่มีผู้ใช้งานโปรแกรมนี้ก็จะทำการติดต่อกับเซิร์ฟเวอร์กลางของ SETI@home เพื่อทำการดาวน์โหลดชิ้นงาน (มีขนาดประมาณ 300 KB) มาประมวลผล และเมื่อประมวลผลเสร็จแล้วโปรแกรมก็จะติดต่อกับเซิร์ฟเวอร์กลางอีกครั้งเพื่อส่งผลลัพธ์กลับไปยังเซิร์ฟเวอร์และดาวน์โหลดชิ้นงานใหม่มาประมวลผลต่อไป ระบบ SETI@home จะเป็นระบบที่มีผู้สร้างงานเพียงคนเดียวคือเครื่องเซิร์ฟเวอร์ ซึ่งผู้ใช้ไม่สามารถส่งงานใดๆ ของตนเองเข้ามาประมวลผลยังระบบได้ ทำให้ระบบเหล่านี้มีความปลอดภัยจากโปรแกรมที่ประสงค์ร้าย (Malicious code) และมีปัญหาด้านความปลอดภัยอื่นๆ น้อย

จากความสำเร็จของโครงการ SETI@home ทำให้มีโครงการที่มีลักษณะการทำงานคล้ายกันเกิดขึ้นตามมาอีกเป็นจำนวนมาก เช่น FightAIDS@Home [8], Folding@Home [9], และ Distributed.net [10] เป็นต้น



รูปที่ 2.2 โครงสร้างของ SETI@home

อย่างไรก็ตาม เนื่องจากระบบ SETI@home เป็นระบบแบบ Single side submission และมีจำนวนเครื่องอาสาสมัครที่รับงานไปประมวลผลเป็นจำนวนมาก (ประมาณ 450,000 เครื่องในแต่ละเวลา) ทำให้ระบบพบกับปัญหาในเรื่องของความต้องการแบนด์วิดธ์ในการส่งงานและรับผลลัพธ์ ซึ่งในช่วง Peak time ระบบต้องใช้ขนาดแบนด์วิดธ์ถึง 70 Mbps ทำให้ระบบต้องเสียค่าใช้จ่ายด้านแบนด์วิดธ์มากกว่า 21,000 ดอลลาร์ต่อเดือน [16]

2.2.2 ระบบที่รับงานมาจากหลายแหล่ง (Multiple sites submission)

ในระบบประเภทนี้งานจะสามารถถูกส่งเข้ามาในระบบได้จากหลายแหล่งเนื่องจากระบบประเภทนี้จะอนุญาตให้สมาชิกสามารถส่งงานของตนเข้ามาประมวลผลในระบบได้ โดยงานที่สามารถส่งมายังระบบได้จะต้องเป็นงานที่อยู่ในรูปแบบตามเงื่อนไขของระบบเพื่อเป็นการป้องกันอันตรายจากโปรแกรมที่มุ่งประสงค์ร้ายสร้างความเสียหายให้แก่ระบบหรือเครื่องที่รับงานไปประมวลผล โดยอาจใช้วิธีการของ Java applets [17] หรือ Sandbox [18] ซึ่งวิธีนี้ก็มีข้อเสีย คือผู้ใช้ต้องเขียนโปรแกรมขึ้นมาใหม่ทั้งหมด ไม่สามารถใช้โปรแกรมที่มีอยู่แล้วได้ และการคำนวณทางวิทยาศาสตร์ก็มักจะไม่นิยมใช้ภาษา Java ในการเขียนโปรแกรมงานอีกด้วย

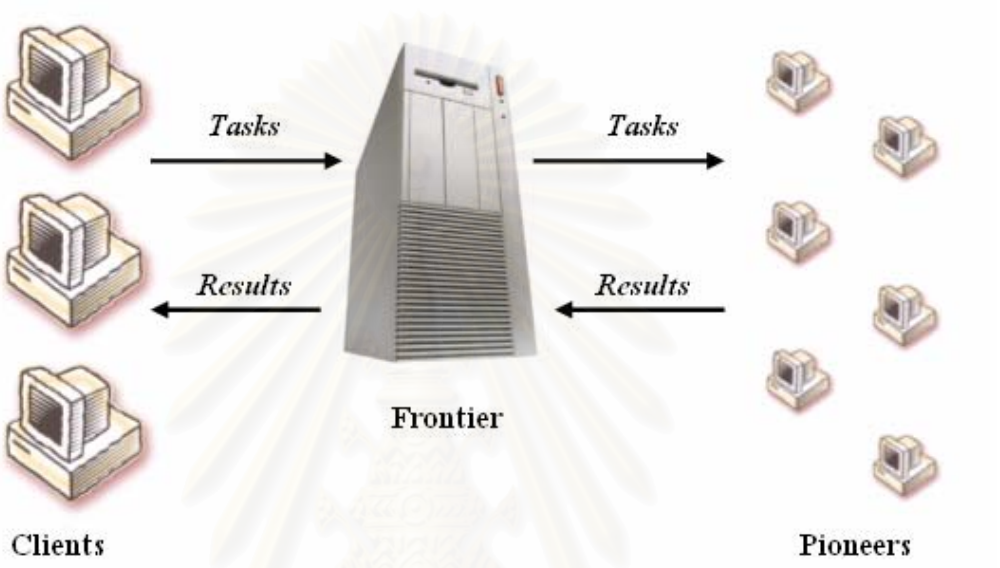
ระบบประเภทนี้ที่น่าสนใจคือ Frontier และ Javelin โดยแต่ละระบบมีโครงสร้างและรูปแบบการทำงานดังนี้

Frontier

Frontier [13] เป็นระบบที่ถูกพัฒนาขึ้นโดยบริษัท Parabon Computation โดยใช้ภาษา Java และมีการสร้าง Frontier Software Develop Kit (Frontier SDK) ที่ช่วยให้ผู้ใช้สามารถสร้างแอพพลิเคชั่นหรืองานที่จะมาใช้บน Frontier ได้ง่ายขึ้นโดย Frontier จะประกอบด้วยส่วนหลักๆ 3 ส่วน คือ

- **Client** : คือเครื่องที่ส่งงานมายังระบบ โดยจะส่งงานมาที่เครื่อง Frontier

- **Pioneer** : คือเครื่องคอมพิวเตอร์ส่วนบุคคลและมีโปรแกรม Pioneer ทำงานอยู่เป็นโพรเซสเบื้องหลัง ซึ่งจะเป็นเครื่องที่ให้ทรัพยากรการคำนวณแก่ระบบ
- **Frontier** : คือเครื่องเซิร์ฟเวอร์กลางของบริษัท Parabon Computation ที่ทำหน้าที่รับงานจากไคลเอนต์และส่งงานไปประมวลผลยังเครื่อง Pioneer และรับผลลัพธ์จากการประมวลผลจาก Pioneer ส่งกลับยังไคลเอนต์เจ้าของงาน



รูปที่ 2.3 โครงสร้างของ Frontier

การทำงานของ Frontier จะแบ่งออกเป็น 2 ส่วน คือเครื่อง Pioneer และเครื่อง Frontier ดังนี้

Pioneer

1. เมื่อเครื่อง Pioneer เชื่อมต่อกับอินเตอร์เน็ตก็จะส่งคำร้องขอและดาวน์โหลดงานจากเครื่อง Frontier มาเก็บไว้บนฮาร์ดดิสก์ของตน
2. เมื่อเครื่องว่างไม่มีผู้ใช้งาน Pioneer ก็จะนำงานที่เก็บไว้ขึ้นมาประมวลผล ถ้าในระหว่างที่ประมวลผล ผู้ใช้กลับมาใช้เครื่องก็จะหยุดงานนั้นไว้ประมวลผลต่อเมื่อเครื่องว่างในครั้งต่อไป
3. เมื่อเครื่อง Pioneer เชื่อมต่อกับอินเตอร์เน็ตอีกครั้ง Pioneer ก็จะส่งผลลัพธ์ที่ได้จากการประมวลผลกลับไปยังเครื่อง Frontier และดาวน์โหลดงานใหม่มาเก็บไว้เพื่อประมวลผลในเวลาที่เครื่องว่างต่อไป

Frontier

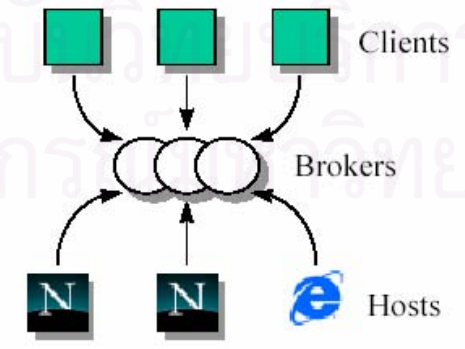
1. ไคลเอนต์ส่งงานไปยัง Frontier เซิร์ฟเวอร์ และเปิดให้เครื่อง Pioneer มาดาวน์โหลดงานไปประมวลผล

2. เมื่องานถูกประมวลผลเสร็จแล้วเครื่อง Pioneer ก็จะส่งผลลัพธ์กลับมายังเครื่อง Frontier เซิร์ฟเวอร์ จากนั้น Frontier เซิร์ฟเวอร์ก็จะส่งผลลัพธ์กลับยังไคลเอนต์ที่ส่งงานมา

Frontier ได้มีการรักษาความปลอดภัยให้แก่เครื่อง Pioneer โดย Frontier จะไม่อนุญาตให้งานใดๆสามารถเข้าถึงไฟล์ที่อยู่ภายในหรือเครือข่ายของเครื่อง Pioneer ได้ และมีการใช้ SSL (Secure Sockets Layer) ในการส่งงานไปยังเครื่อง Frontier เพื่อป้องกันไม่ให้มีใครสามารถขโมยงานของไคลเอนต์ไปใช้ได้ ด้วยประสิทธิภาพของเครื่องคอมพิวเตอร์ส่วนบุคคลจำนวนมากบนระบบทำให้ Frontier มีทรัพยากรจำนวนมากเพื่อใช้ในการทำงานที่ต้องการพลังในการประมวลผลสูงเช่นงานทางวิทยาศาสตร์ หรือปัญหาอื่นๆที่ซับซ้อนได้ แต่ปัญหาของ Frontier คือ ถ้าเมื่อเครื่อง Frontier เกิดเสียไม่สามารถทำงานได้จะทำให้ระบบหยุดการทำงานไปด้วย (มี Single point of failure) และผู้ใช้ต้องเขียนโปรแกรมสำหรับทำงานขึ้นมาใหม่โดยใช้ Frontier SDK จึงไม่สามารถใช้โปรแกรมที่มีอยู่แล้วได้

Javelin

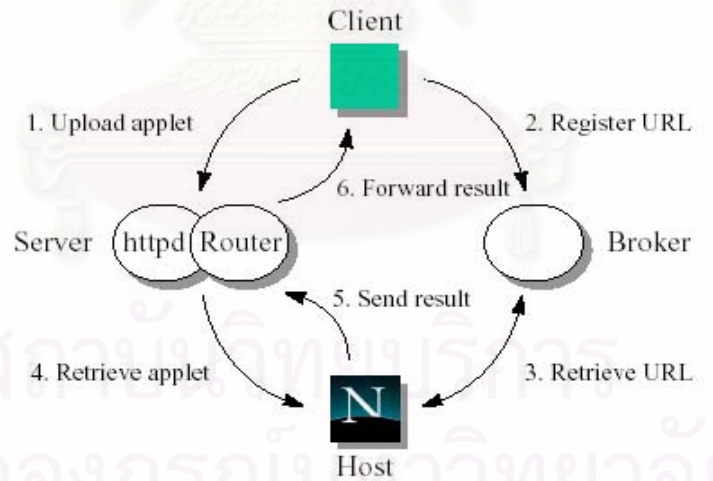
Javelin [14] เป็นระบบคำนวณแบบกระจายบนเครือข่ายอินเตอร์เน็ตที่อาศัยเทคโนโลยีของเว็บ และ Java applets ในการรับงานจากผู้ใช้และส่งไปประมวลผลบนเครื่องต่างๆที่อยู่ในระบบอินเตอร์เน็ต โดยลักษณะของงานที่จะนำมาประมวลผลใน Javelin จะอยู่ในรูปแบบของ Applets ที่ฝังอยู่ในเอกสาร HTML ที่สามารถอ้างถึงได้โดยใช้ URL เหมือนเว็บเพจทั่วๆไป โดยการใช้ Applets จะช่วยป้องกันคอมพิวเตอร์จากโค้ดที่มุ่งสร้างความเสียหาย ได้เป็นอย่างดีเนื่องจาก Applets จะไม่อนุญาตให้มีการอ่านหรือเขียนไฟล์ใดๆ บนเครื่องปลายทางได้ และการใช้งานที่ง่ายเพราะผู้ใช้ไม่จำเป็นต้องลงโปรแกรมใดเพิ่มเติมมีเพียงแค่เว็บบราวเซอร์ที่สนับสนุน Java ก็เพียงพอ โครงสร้างของ Javelin ประกอบด้วย 3 ส่วนหลักๆ ดังรูปที่ 2.4



รูปที่ 2.4 โครงสร้างของ Javelin

- **Client**: คือโพรเซสที่ต้องการทรัพยากรในการคำนวณ (โพรเซสที่ทำหน้าที่ส่งงานมายังระบบ) โดยเครื่องที่เป็น Client จะมี HTTP เซิร์ฟเวอร์อยู่เป็นโพรเซสเบื้องหลัง (Back ground process) หรือไม่ก็ได้ โดยถ้ามี HTTP เซิร์ฟเวอร์รันอยู่ Client นั้นก็จะเก็บหน้า Applets ของงานไว้ที่เครื่องของตน แต่ถ้าไม่มี HTTP เซิร์ฟเวอร์รันอยู่ก็จะต้องอัพโหลด (Upload) หน้า Applets นั้นไปยังเครื่องอื่นที่มี HTTP เซิร์ฟเวอร์รันอยู่ จากนั้นก็จะทำการลงทะเบียนงานโดยการส่ง URL ที่อ้างถึงงานนั้นไปยังตัว Broker เพื่อให้ระบบทราบถึงงานใหม่ที่เข้ามา
- **Host**: คือโพรเซสที่ให้ทรัพยากรในการคำนวณแก่ระบบซึ่งอยู่ในรูปของเว็บบราวเซอร์ที่สนับสนุน Java โดยถ้า Host ต้องการทำงานใดก็เพียงแค่เปิด URL ของงานนั้นขึ้นมา
- **Broker**: เป็นตัวกลางในการจัดหาทรัพยากรให้แก่ Client และหางานที่เหมาะสมให้แก่ Host โดยจะทำหน้าที่เป็นตัวจัดการ Host ที่มีอยู่ในระบบ, คิว (Queue) งาน และส่งงานที่เหมาะสมให้แต่ละ Host

รูปที่ 2.5 แสดงขั้นตอนการทำงานของ Javelin โดย HTTP เซิร์ฟเวอร์จะทำงาน 2 ฟังก์ชันหลักคือ 1) เก็บหน้า Applets ของงานและส่งให้แก่ Host 2) ส่งต่อผลลัพธ์ของงานกลับไปให้เครื่องที่เป็น Client เจ้าของงาน



รูปที่ 2.5 ขั้นตอนการทำงานของ Javelin

1. Client อัพโหลด Applets ที่ฝังอยู่ในหน้า HTML ไปบน HTTP เซิร์ฟเวอร์ ถ้า Client ใดรัน HTTP เซิร์ฟเวอร์นี้อยู่แล้วก็ข้ามขั้นตอนนี้ไป
2. Client ทำการลงทะเบียน URL ของงานกับ Broker
3. Host ทำการลงทะเบียนกับ Broker และรับ URL ของงาน

4. Host ดาวน์โหลดหน้า HTML จาก HTTP เซิร์ฟเวอร์และประมวลผล Applets ที่อยู่ในหน้า HTML นั้นโดยการเปิดหน้า HTML นั้นขึ้นมา
5. Host ส่งผลลัพธ์จากการประมวลผลกลับไปยังเซิร์ฟเวอร์
6. Client รับผลลัพธ์จากเซิร์ฟเวอร์

เนื่องจากงานในระบบ Javelin จะอยู่ในรูปแบบของ Applets ดังนั้นผู้ใช้จึงไม่สามารถนำโปรแกรมที่ตนมีอยู่แล้วนำไปใช้ได้ต้องเขียนงานขึ้นมาใหม่ให้อยู่ในรูปของ Applets อีกทั้งการใช้ Applets จะทำให้เกิดข้อจำกัดในเรื่องของประเภทของงาน โดย Javelin สามารถประมวลผลงานที่เป็นภาษา Java ที่ไม่ต้องมีการอ่านและเขียนไฟล์ได้เท่านั้น และผู้ใช้จำเป็นต้องเรียนรู้การเขียน Applets จึงจะสามารถเขียนงานและส่งงานไปประมวลผลยังระบบได้ทำให้ไม่สะดวกแก่การใช้งาน

2.3 ระบบการคำนวณแบบกระจายบนเครือข่ายเพียร์-ทู-เพียร์

ระบบประเภทนี้เครื่องคอมพิวเตอร์เครื่องหนึ่งสามารถเป็นได้ทั้งผู้ที่ส่งงานเข้ามายังระบบ และผู้ที่รับงานไปประมวลผล นอกจากนั้นเครื่องหนึ่งอาจทำหน้าที่เป็นเครื่องเซิร์ฟเวอร์หรือส่วนหนึ่งของเซิร์ฟเวอร์ที่ให้บริการเครื่องสมาชิกในระบบ ตัวอย่างระบบประเภทนี้ คือ JNGI

JNGI [15] เป็นระบบการคำนวณแบบเพียร์-ทู-เพียร์ที่มุ่งเน้นให้ระบบสามารถทำงานได้บนสภาพแวดล้อมที่ไม่คงที่ซึ่งแต่ละเพียร์สามารถเข้าและออกจากระบบได้ตลอดเวลาโดยไม่กระทบต่อการทำงานโดยรวม และระบบสามารถทำงานได้กับทุกๆแพลตฟอร์ม โดยได้มีการแบ่งเพียร์ออกเป็นกลุ่มๆ ตามฟังก์ชันการทำงานดังนี้

- **Monitor Group** : อยู่ระดับบนสุดที่ทำหน้าที่ประสานการทำงานทั้งหมดของระบบ เช่น จัดการเพียร์ที่ต้องการเข้ามายังระบบ และดูแลการส่งงานมาประมวลผลในระบบ
- **Worker Group** : ทำหน้าที่รับงานมาประมวลผล
- **Task dispatcher Group** : ทำหน้าที่ส่งงานไปยัง Worker Group
- **Repository Group** : ทำหน้าที่เก็บโปรแกรม, และข้อมูลต่างๆ ที่จำเป็นต่อการทำงานของงานในระบบ

แต่ละเพียร์สามารถเป็นสมาชิกกลุ่มได้หลายกลุ่ม แต่ไม่สามารถพร้อมกันได้ และแต่ละกลุ่มสามารถมีเพียร์เป็นสมาชิกได้หลายเพียร์เพื่อให้ระบบมีความทนทานและสามารถรองรับจำนวนผู้ใช้ได้เพิ่มขึ้น

การทำงานเริ่มต้นโดยเมื่อผู้ใช้ต้องการส่งงานมาประมวลผลในระบบในแต่ละงานจะมีการสร้าง Repository สำหรับงานนั้นขึ้นมาเพื่อเก็บโปรแกรมและข้อมูลที่เกี่ยวกับงานนั้นไว้ เมื่อ Worker ว่างก็จะทำการติดต่อกับ Task dispatcher เพื่อบอกถึงการมีเครื่องว่างรวมถึงบอกว่าเครื่องตนมีโปรแกรม

อะไรเก็บอยู่บ้าง จากนั้น Task dispatcher จะติดต่อไปยัง Repository เพื่อหางานที่เพียร์นั้นสามารถประมวลผลได้ เมื่อพบแล้วก็จะทำการส่งงานมายัง Worker เมื่อ Worker ทำการประมวลผลงานเสร็จแล้ว ผลลัพธ์ก็จะถูกส่งมายัง Task dispatcher เพื่อนำไปเก็บยัง Repository คอยเจ้าของงานมารับผลลัพธ์ไป โดยแต่ละงาน และแต่ละเพียร์จะมีเลขประจำตัวทำให้ง่ายต่อการหาเพียร์หรืองานที่ต้องการ

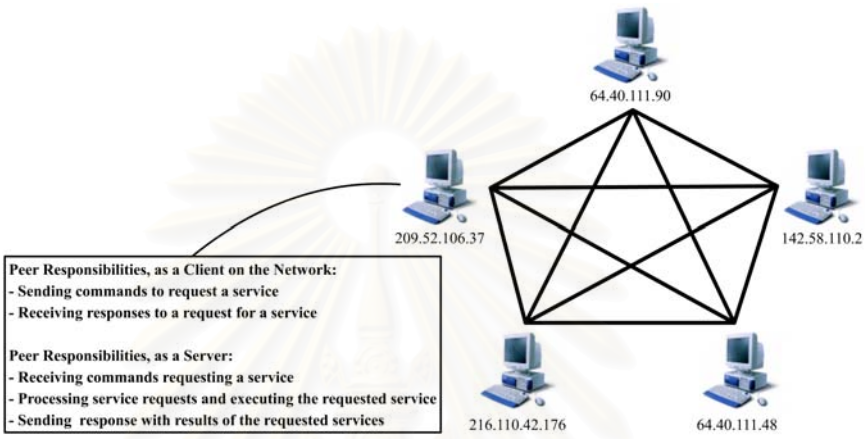
โปรแกรมและงานที่จะถูกนำมาใช้ใน JNGI ได้จะต้องถูกเขียนใหม่ให้อยู่ในรูปของ Java Thread เนื่องจากระบบทำงานโดยการส่ง Thread ของงานไปประมวลผลยังเครื่องอื่น ด้วย Remote Thread ซึ่งผู้ใช้ต้องศึกษาการเขียนโปรแกรมโดยใช้ Remote Thread จึงจะสามารถใช้งานระบบได้

2.4 เครือข่ายแบบเพียร์-ทู-เพียร์

ทุกวันนี้บริการส่วนใหญ่บนระบบอินเตอร์เน็ตจะใช้โครงสร้างแบบไคลเอนต์/เซิร์ฟเวอร์ ซึ่งเครื่องไคลเอนต์จะเชื่อมต่อกับเครื่องเซิร์ฟเวอร์ด้วยโพรโตคอลต่างๆ เช่น HTTP หรือ FTP เพื่อเรียกใช้บริการที่อยู่บนเครื่องเซิร์ฟเวอร์นั้น โดยการประมวลผลต่างๆ มักจะเกิดขึ้นที่เครื่องเซิร์ฟเวอร์เป็นส่วนใหญ่แต่กลับปล่อยให้เครื่องไคลเอนต์รับภาระไม่มากนัก รูปแบบการทำงานนี้ถูกเรียกว่า “Service-delivery model” อย่างไรก็ตาม โครงสร้างแบบไคลเอนต์/เซิร์ฟเวอร์นี้กลับมีข้อเสียที่สำคัญคือ เมื่อระบบมีจำนวนไคลเอนต์เพิ่มมากขึ้น ทรัพยากรของเครื่องเซิร์ฟเวอร์ที่มีอยู่ทั้งพลังในการประมวลผลและแบนด์วิดธ์อาจไม่สามารถรองรับการให้บริการเหล่านั้นได้อย่างเพียงพอจนทำให้เกิดปัญหาคอขวดขึ้นที่เซิร์ฟเวอร์ ในขณะที่ทรัพยากรต่างๆ ของเครื่องไคลเอนต์ทั้งพลังการประมวลผลและแบนด์วิดธ์กลับแทบไม่ถูกใช้งานเต็มที่เท่าที่ควรทั้งๆ ที่อัตราการเพิ่มขึ้นของประสิทธิภาพของเครื่องไคลเอนต์มีมากดังกฎของมัวร์

รูปที่ 2.6 โครงสร้างแบบไคลเอนต์/เซิร์ฟเวอร์

เพียร์-ทู-เพียร์ (Peer-to-peer, P2P) คือกลุ่มของระบบและแอพพลิเคชั่นที่มีการใช้ทรัพยากรที่กระจายอยู่บนเครือข่ายเพื่อทำงานบางอย่างร่วมกันในรูปแบบที่ไม่เป็นศูนย์กลาง (Decentralized) โดยทรัพยากรดังกล่าวรวมถึงพลังการประมวลผล, ข้อมูลและเนื้อที่เก็บข้อมูล, แบนด์วิดธ์ของเครือข่าย, และทรัพยากรบุคคล งานที่ทำอาจเป็นการคำนวณแบบกระจาย (Distributed computing), การแชร์ไฟล์ (File sharing), การทำงานร่วมกัน (Collaboration), และให้บริการอื่นๆ เป็นต้น



รูปที่ 2.7 โครงสร้างแบบเพียร์-ทู-เพียร์

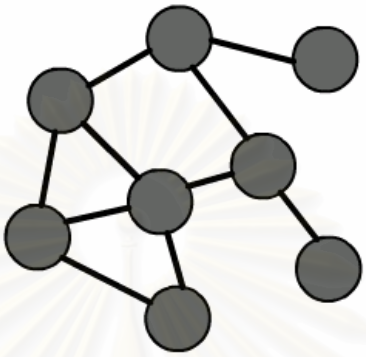
ระบบเพียร์-ทู-เพียร์เป็นตัวช่วยให้ระบบและแอพพลิเคชั่นต่างๆ สามารถดึงเอาทรัพยากรที่ไม่ได้ใช้งานของเครื่องคอมพิวเตอร์มาให้กับระบบเพื่อให้ระบบมีประสิทธิภาพเพิ่มขึ้นโดยไม่จำเป็นต้องเพิ่มประสิทธิภาพของเครื่องเซิร์ฟเวอร์ ในระบบเพียร์-ทู-เพียร์นั้น แต่ละเพียร์จะสามารถทำหน้าที่เป็นได้ทั้งเซิร์ฟเวอร์และไคลเอนต์ ดังนั้นหากมีเพียร์ใดออกไปจากระบบ ระบบก็ยังสามารถทำงานต่อไปได้เนื่องจากยังมีเพียร์อื่นที่ทำหน้าที่เป็นเซิร์ฟเวอร์แทนเพียร์ที่ออกไป

เครือข่ายแบบเพียร์-ทู-เพียร์นั้นสามารถแบ่งประเภทตามโครงสร้างออกได้เป็น 3 ประเภทใหญ่ๆ คือ Pure P2P, Hybrid P2P, และ Super Peer โดยในแต่ละประเภทจะมีข้อดีข้อเสียที่แตกต่างกัน

2.4.1 โมเดลแบบ Pure P2P

โมเดลแบบ Pure P2P จะมีลักษณะที่ตรงข้ามกับระบบแบบรวมศูนย์ โดยทุกๆ เพียร์สามารถติดต่อถึงกันได้โดยตรงโดยไม่ต้องผ่านเครื่องเซิร์ฟเวอร์ จุดเด่นของโมเดลแบบนี้คือความคงทนของระบบ (Fault tolerant) โดยถ้ามีเพียร์ใดเพียร์หนึ่งเสียหรือออกไปจากระบบก็จะไม่ส่งผลกระทบต่อระบบโดยรวม แต่โมเดลแบบนี้ก็มีข้อจำกัดตรงที่การควบคุมการไหลของข้อมูลในระบบทำได้ยาก ทำให้มักจะเกิดปัญหาการใช้แบนด์วิดธ์ที่สิ้นเปลือง และการจัดการเกี่ยวกับสมาชิกในระบบทำได้ยากเพราะโมเดลแบบนี้การยืนยันตัวผู้ใช้ (Authentication) จะทำได้ยาก ซึ่งอาจทำให้มีเพียร์ที่ไม่ประสงค์ดีเข้ามายังระบบเพื่อสร้างความเสียหายให้แก่ระบบหรือสมาชิกในระบบได้ ส่วนในด้านของความสามารถในการ

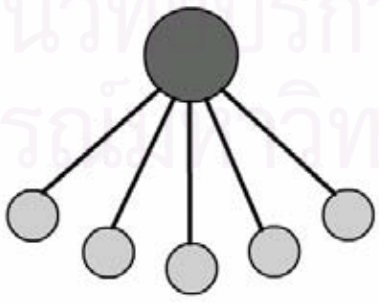
ขยายขนาดของระบบนั้นเป็นเรื่องยากที่จะบอกได้แน่นอนเพราะในทางทฤษฎีแล้วระบบสามารถขยายขนาดได้เรื่อยๆ ไม่จำกัด แต่ในทางปฏิบัติแล้วการที่จะทำให้เพียร์เหล่านั้นทำงานร่วมกันได้มักจะเกิดโสหุ้ยมากขึ้นตามขนาดของระบบซึ่งทำให้โมเดลแบบนี้มีความสามารถในการขยายระบบไม่ดีนัก ตัวอย่างเช่นระบบ Gnutella [19] เป็นต้น



รูปที่ 2.8 โมเดลแบบ Pure P2P

2.4.2 โมเดลแบบ Hybrid P2P

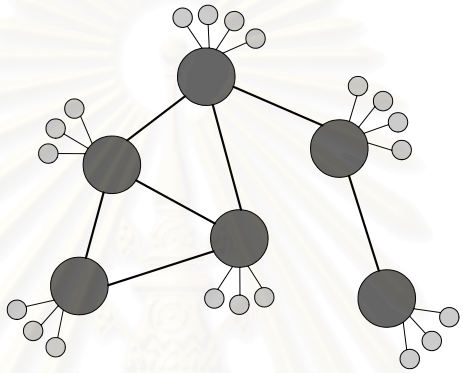
โมเดลแบบ Hybrid P2P จะเหมือนกับแบบรวมศูนย์คือมีเครื่องเซิร์ฟเวอร์กลางที่ทำหน้าที่ควบคุมการทำงานของระบบ แต่การส่งข้อมูลระหว่างเพียร์จะส่งถึงกันโดยตรงโดยไม่ผ่านเซิร์ฟเวอร์เช่นเดียวกับแบบ Pure P2P ทำให้ช่วยลดปัญหาด้านการจัดการระบบและเพิ่มความสามารถในการขยายขนาดของระบบเพราะโมเดลนี้จะมีการลดภาระของเครื่องเซิร์ฟเวอร์ลง เช่น ใน Napster [20] เครื่องเซิร์ฟเวอร์ทำหน้าที่เพียงดูแลฐานข้อมูลที่เก็บว่าไฟล์เพลงนี้ถูกเก็บอยู่ที่เครื่องใดและส่งข้อมูลนี้ให้เพียร์ที่ต้องการ จากนั้นเพียร์นั้นก็จะติดต่อและดาวน์โหลดไฟล์นั้นจากเครื่องต้นทางโดยตรงโดย เป็นต้น เนื่องจากภาระที่เครื่องเซิร์ฟเวอร์ต้องแบกรับน้อยลงทำให้โมเดลแบบนี้มีความสามารถในการขยายระบบที่ดีกว่าโมเดลแบบรวมศูนย์ แต่อย่างไรก็ตามถ้าเครื่องเซิร์ฟเวอร์เสีย ระบบก็จะไม่สามารถทำงานต่อไปได้



รูปที่ 2.9 โมเดลแบบ Hybrid P2P

2.4.3 โมเดลแบบ Super peer

โมเดลแบบ Super peer เป็นการรวมกันของโมเดลแบบ Pure P2P และ Hybrid P2P โดย Super peer คือเพียร์ที่ทำหน้าที่เหมือนเป็นเซิร์ฟเวอร์กลางในโครงสร้างแบบ Hybrid P2P ที่ทำหน้าที่ควบคุมการทำงานและการไหลของข้อมูลให้กับกลุ่มของเพียร์ในแต่ละกลุ่ม ในขณะที่แต่ละ Super peer จะเชื่อมต่อถึงกันด้วยโมเดลแบบ Pure P2P ดังนั้นในแต่ละ Super peer จะต้องมีโพรโตคอลในการติดต่อสื่อสารอยู่ 2 โพรโตคอลคือ โพรโตคอลในการติดต่อกับเพียร์ลูก และโพรโตคอลที่ใช้ในการติดต่อระหว่าง Super peer ด้วยกันเอง



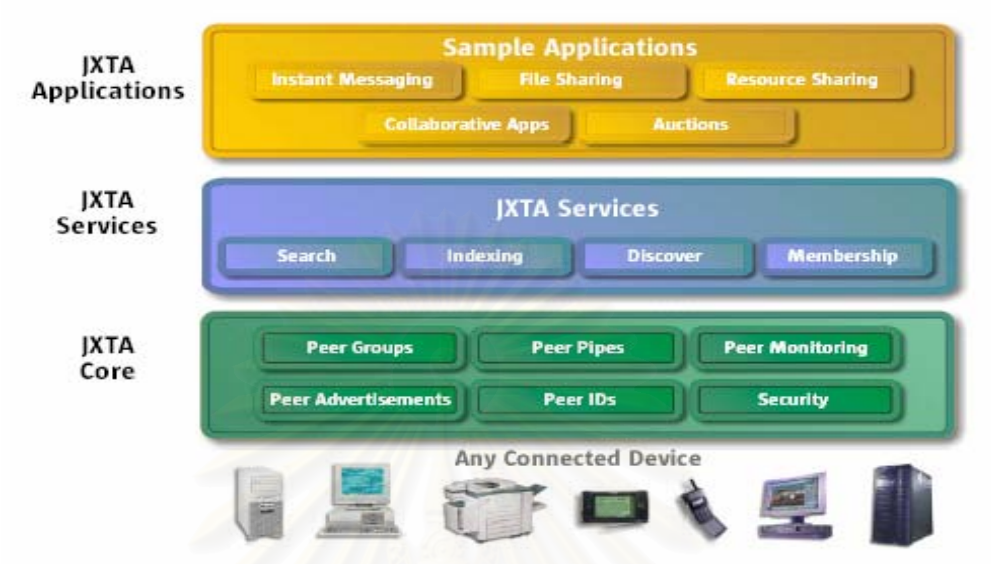
รูปที่ 2.10 โครงสร้างแบบ Super peer

โมเดลแบบ Super peer จะมีความสามารถในการขยายขนาดของระบบที่ดีมากเนื่องจากเมื่อจำนวนเพียร์ลูกในแต่ละ Super peer มากเกินไป ระบบก็จะคัดเลือกเพียร์ใหม่มาเป็น Super peer เพิ่ม และระบบมีความคงทนค่อนข้างสูงแม้ว่า Super peer ตัวใดเสียก็จะกระทบเฉพาะเพียร์ลูกของมันเท่านั้น โมเดลแบบนี้กำลังเป็นที่นิยมอย่างมากในปัจจุบันเนื่องจากสามารถรวมเอาข้อดีของโครงสร้างทั้งสองแบบข้างต้นมาไว้ด้วยกัน ในบางระบบอาจดัดแปลงให้ Super peer แต่ละตัวเชื่อมต่อกันด้วยโครงสร้างแบบอื่นก็ได้เช่นโครงสร้างแบบวงแหวน (Ring) เป็นต้น ตัวอย่างระบบที่ใช้โมเดลแบบ Super peer เช่น Kazaa [21] และระบบอื่นๆ ที่เคยใช้โมเดลแบบ Pure P2P เช่น Gnutella [19] และ FreeNet [22] เป็นต้น

2.5 โพรโตคอล JXTA

โพรโตคอล JXTA [23] [24] เป็นโพรโตคอลที่ออกแบบสำหรับการทำงานบนเครือข่ายแบบเพียร์-ทู-เพียร์ ที่ช่วยให้อุปกรณ์ต่างๆ บนเครือข่าย เช่น โทรศัพท์มือถือ, เครื่อง PDA, คอมพิวเตอร์โน้ตบุ๊ก, คอมพิวเตอร์ส่วนบุคคลจนถึงเครื่องคอมพิวเตอร์เซิร์ฟเวอร์ สามารถติดต่อสื่อสารและทำงานร่วมกันได้โดยไม่จำเป็นต้องมีศูนย์กลางคอยควบคุมการทำงานของระบบ โพรโตคอล JXTA ถูกออกแบบมาให้สามารถใช้ได้กับทุกภาษาคอมพิวเตอร์และทุกมาตรฐานการส่งข้อมูล ดังนั้นจึงมีการนำเอาโพรโตคอล JXTA ไปอิมพลีเมนต์ด้วยภาษาต่างๆ มากมาย เช่น C, Python, Perl และ Java โดยสามารถทำงานได้บนมาตรฐานการส่งข้อมูลแบบ TCP/IP, HTTP, Bluetooth, HomePNA และอื่นๆ โดยใน

งานวิจัยนี้จะใช้โพรโตคอล JXTA ที่อิมพลีเมนต์โดยภาษา Java เพื่อสามารถนำไปใช้ได้กับทุกๆ แพลตฟอร์ม ซึ่งจะทำงานอยู่บนมาตรฐานการส่งข้อมูลแบบ TCP/IP และ HTTP



รูปที่ 2.11 โครงสร้างของโพรโตคอล JXTA

รายละเอียดและการทำงานของโพรโตคอล JXTA จะกล่าวถึงโดยละเอียดภายหลังในภาคผนวก ก.

บทที่ 3

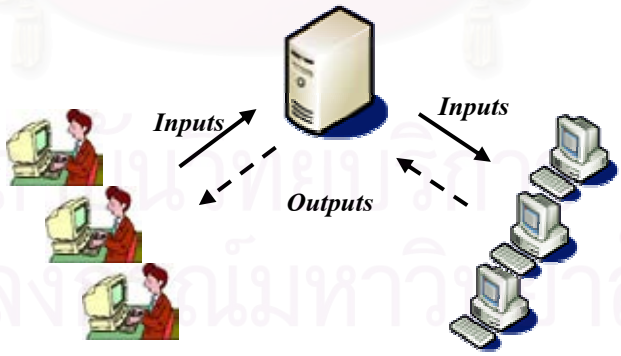
การวิเคราะห์ประสิทธิภาพของระบบการคำนวณแบบกระจาย ที่ใช้การสื่อสารแบบเพียร์-ทู-เพียร์

ในบทนี้จะกล่าวถึงปัญหาการเกิดคอขวดของเครือข่ายที่เครื่องเซิร์ฟเวอร์ในระบบที่ใช้โครงสร้างแบบไคลเอนต์/เซิร์ฟเวอร์ ซึ่งส่งผลให้ประสิทธิภาพการทำงานของระบบลดลงอย่างรวดเร็ว และจะกล่าวถึงวิธีการแก้ปัญหาดังกล่าวด้วยการนำรูปแบบการสื่อสารแบบเพียร์-ทู-เพียร์เข้ามาประยุกต์ใช้กับระบบเพื่อลดปัญหาคอขวดที่เครื่องเซิร์ฟเวอร์ ทำให้ประสิทธิภาพของระบบเพิ่มขึ้นโดยไม่จำเป็นต้องเสียค่าใช้จ่ายในการเพิ่มขนาดแบนด์วิดธ์ที่เครื่องเซิร์ฟเวอร์แต่อย่างใด

อย่างไรก็ตามในปัจจุบันยังขาดการศึกษาเกี่ยวกับประสิทธิภาพของการนำเอาการสื่อสารแบบเพียร์-ทู-เพียร์มาใช้กับการคำนวณแบบกระจาย โดยเฉพาะปัจจัยที่เป็นข้อจำกัดต่อการขยายระบบ และความเสถียร์ของเครือข่ายแบบเพียร์-ทู-เพียร์ อันเป็นจุดสำคัญของระบบแบบเพียร์-ทู-เพียร์

เพื่อที่จะพิสูจน์ถึงประสิทธิภาพการทำงานที่เพิ่มขึ้นจากการประยุกต์ใช้การสื่อสารแบบเพียร์-ทู-เพียร์เข้ากับระบบการคำนวณแบบกระจาย และผลกระทบของปัจจัยต่างๆ ต่อประสิทธิภาพของระบบ ผู้วิจัยจึงได้ใช้โปรแกรมแบบจำลอง (Simulator) เพื่อจำลองการทำงานของระบบและเปรียบเทียบประสิทธิภาพของระบบกับระบบแบบไคลเอนต์/เซิร์ฟเวอร์ รวมถึงได้มีการศึกษาถึงผลกระทบของปัจจัยต่างๆ ที่มีผลต่อประสิทธิภาพของระบบเพื่อเป็นแนวทางในการพัฒนาระบบต่อไป

3.1 ระบบการคำนวณแบบกระจายแบบไคลเอนต์/เซิร์ฟเวอร์



รูปที่ 3.1 รูปแบบการทำงานที่ใช้การสื่อสารแบบไคลเอนต์/เซิร์ฟเวอร์

โครงสร้างแบบไคลเอนต์/เซิร์ฟเวอร์ถูกนำมาใช้อย่างแพร่หลายเนื่องจากมีความสามารถในการควบคุมการทำงาน (Manageability) และการรักษาความปลอดภัยที่ดี โดยโครงสร้างแบบไคลเอนต์นั้นจะมีเครื่องเซิร์ฟเวอร์กลางที่เป็นศูนย์กลางในการให้บริการและติดต่อสื่อสารระหว่างไคลเอนต์ในระบบ ในระบบการคำนวณแบบกระจายนั้นเครื่องเซิร์ฟเวอร์จะทำหน้าที่จัดลำดับงาน (Scheduling), จับคู่

ระหว่างงานกับเครื่องที่เหมาะสม (Matching) และที่สำคัญคือเป็นตัวกลางในการรับ-ส่งระหว่างเจ้าของงานกับเครื่องที่รับงานไปประมวลผล ดังนั้นเมื่อจำนวนไคลเอนต์ในระบบหรือขนาดไฟล์ข้อมูลที่ส่งในระบบเพิ่มมากขึ้น จะส่งผลให้ปริมาณการใช้แบนด์วิดธ์ของเครื่องเซิร์ฟเวอร์ก็จะเพิ่มขึ้นด้วย และเมื่อเครื่องเซิร์ฟเวอร์ต้องแบกรับภาระการส่งข้อมูลที่มากเกินกว่าขนาดแบนด์วิดธ์ของเครื่องก็จะก่อให้เกิดปัญหาคอขวดขึ้น ซึ่งจะทำให้อัตราเร็วในการส่งข้อมูลไปยังหน่วยต่างๆ ในระบบช้าลง ทำให้เวลาที่ใช้ในการทำงานแต่ละชิ้นก็จะเพิ่มขึ้นไปด้วยดังสมการที่ 3.1

$$\textbf{Response time} = T_{in} + T_{queue} + T_{comp} + T_{out} \quad \textbf{(3.1)}$$

จากสมการที่ 3.1 เราจะพบว่าเมื่อจำนวนเครื่องที่รับงานไปประมวลผลมีมากขึ้น ถึงแม้ว่าจะเป็นการลดระยะเวลาในการคำนวณ (T_{comp}) แต่ถ้าแบนด์วิดธ์ของเซิร์ฟเวอร์ไม่สัมพันธ์กับจำนวนเครื่องที่รับงานไปประมวลผลจะส่งผลให้เกิดคอขวดขึ้นที่เซิร์ฟเวอร์ ซึ่งทำให้เวลาที่อินพุตถูกส่งไปยังเครื่องที่รับงานไปประมวลผล (T_{in}) เพิ่มขึ้น และเวลาที่เครื่องที่รับงานไปประมวลผลส่งคืนเอาต์พุตกลับไปยังเซิร์ฟเวอร์ (T_{out}) เพิ่มขึ้นด้วย ดังนั้นค่า *Response time* ก็จะเพิ่มขึ้น อีกทั้งเครื่องนั้นจะรับงานใหม่มาประมวลผลได้ช้าลงงานจึงค้างอยู่ในคิวมากและเวลาที่งานรออยู่ในคิว (T_{queue}) ก็จะเพิ่มขึ้น ดังนั้นโครงสร้างแบบไคลเอนต์/เซิร์ฟเวอร์จึงสามารถรองรับจำนวนเครื่องที่รับงานไปประมวลผลได้อย่างจำกัด

วิธีการแก้ปัญหาดังกล่าวอาจทำได้โดยการเพิ่มขนาดแบนด์วิดธ์ให้แก่เครื่องเซิร์ฟเวอร์ และการใช้เครื่องเซิร์ฟเวอร์หลายๆ เครื่อง (Multiple servers) [25] แต่ทั้งสองวิธีก็เป็นการเพิ่มค่าใช้จ่ายให้แก่ระบบอย่างมาก ดังนั้นแอพพลิเคชั่นที่เหมาะสมกับระบบแบบนี้จึงมักเป็นงานที่เน้นการคำนวณ (Computing intensive application) ที่มีขนาดของข้อมูลน้อยแต่ใช้เวลาในการประมวลผลมาก เพื่อความเหมาะสมในการนำไปใช้กับระบบที่ทำงานอยู่บนเครือข่ายอินเตอร์เน็ต [26]

3.2 ระบบการคำนวณแบบกระจายแบบเพียร์-ทู-เพียร์

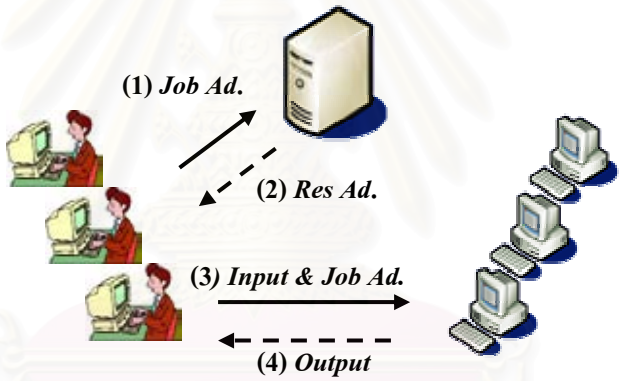
เพื่อให้รูปแบบการคำนวณแบบกระจายบนเครือข่ายอินเตอร์เน็ตเป็นที่แพร่หลายและทำได้ง่ายตั้งแต่องค์กรขนาดเล็กถึงขนาดกลางที่มีการลงทุนเริ่มต้นน้อย ไม่จำเป็นต้องใช้เครื่องเซิร์ฟเวอร์ที่มีขนาดแบนด์วิดธ์สูง และสามารถรองรับงานที่มีขนาดของข้อมูลมากๆ ได้ จึงจำเป็นต้องแก้ปัญหาปริมาณการใช้แบนด์วิดธ์ของเครื่องเซิร์ฟเวอร์โดยให้ประหยัดค่าใช้จ่ายมากที่สุด

ในเครือข่ายแบบเพียร์-ทู-เพียร์จะมีการส่งข้อมูลจากเครื่องหนึ่งไปยังอีกเครื่องหนึ่งโดยตรงโดยไม่ผ่านเครื่องเซิร์ฟเวอร์ ซึ่งเป็นการใช้แบนด์วิดธ์ของเครื่องไคลเอนต์ด้วยกันเอง ทำให้แบนด์วิดธ์โดยรวมของระบบจะขยายขึ้นตามจำนวนของเครื่องไคลเอนต์ ส่งผลให้ระบบสามารถรองรับจำนวนเครื่องได้เพิ่มมากขึ้นโดยไม่จำเป็นต้องเพิ่มขนาดแบนด์วิดธ์ของเครื่องเซิร์ฟเวอร์แต่อย่างใด

จากข้อดีของรูปแบบการสื่อสารแบบเพียร์-ทู-เพียร์ที่ได้กล่าวมาข้างต้น ผู้วิจัยจึงได้นำรูปแบบการสื่อสารแบบเพียร์-ทู-เพียร์มาประยุกต์ใช้กับระบบการคำนวณแบบกระจาย โดยในระบบแบบ

เพียร์-ทู-เพียร์นั้นจะพยายามส่งข้อมูลระหว่างเจ้าของงานและเครื่องที่รับงานไปประมวลผลโดยตรงให้มากที่สุดเท่าที่เป็นไปได้ โดยมีขั้นตอนการทำงานดังนี้รูปที่ 3.2

1. เจ้าของงานส่ง *Job Ad*. (*Job Advertisement*) ที่แสดงรายละเอียดของงาน เช่นความต้องการของเครื่องขั้นต่ำ, โปรแกรมที่ใช้, และคำสั่งที่ใช้ เป็นต้น ไปยังเซิร์ฟเวอร์เพื่อหาเครื่องที่เหมาะสมที่จะรับงานนั้นไปประมวลผล
2. ถ้าในขณะนั้นมีเครื่องที่สามารถรับงานไปประมวลผลได้ทันที เซิร์ฟเวอร์ก็จะส่ง *Res Ad*. (*Resource Advertisement*) ที่ประกอบด้วยข้อมูลที่จำเป็นในการเชื่อมต่อไปยังเครื่องที่จะรับงานไปประมวลผล กลับมายังเจ้าของงาน
3. เมื่อทั้งสองเครื่องสามารถเชื่อมต่อกันได้แล้ว อินพุต และ *Job A*d. ก็จะถูกส่งไปยังเครื่องที่รับงานไปประมวลผลโดยตรงโดยไม่ต้องผ่านเซิร์ฟเวอร์
4. เมื่องานถูกประมวลผลเสร็จแล้ว เครื่องที่รับงานไปประมวลผลจะใช้ข้อมูลที่อยู่ใน *Job Ad*. ทำการเชื่อมต่อไปยังเจ้าของงาน เพื่อส่งเอาต์พุตกลับไป



รูปที่ 3.2 รูปแบบการทำงานที่ใช้การสื่อสารแบบเพียร์-ทู-เพียร์

ด้วยเทคนิคนี้ทำให้ภาระในการส่งข้อมูลและการใช้แบนด์วิดธ์ของเซิร์ฟเวอร์ลดลง โดยข้อมูลสามารถส่งไปยังที่หมาย (เครื่องเจ้าของงาน หรือเครื่องที่รับงานไปประมวลผล) ได้โดยตรงโดยไม่ต้องใช้เซิร์ฟเวอร์เป็นตัวช่วยส่งต่อข้อมูล

อย่างไรก็ตามกรณีที่ไม่มีเครื่องว่างที่จะรับงานไปประมวลผลงานนั้นก็จะถูกเก็บไว้ในคิว ในกรณีนี้เครื่องเจ้าของงานจะได้รับแจ้งให้ส่งอินพุตไปเก็บไว้ที่เซิร์ฟเวอร์เพื่อป้องกันกรณีที่เจ้าของงานออกไปจากระบบทำให้ไม่มีอินพุตสำหรับการประมวลผล และเมื่อมีเครื่องที่สามารถรับงานนั้นได้แล้ว เซิร์ฟเวอร์ก็จะทำหน้าที่ส่งอินพุต และ *Job Ad*. ไปยังเครื่องที่รับงานไปประมวลผลนั้นแทน

ในบางครั้งเมื่องานเสร็จแต่เครื่องที่รับงานไปประมวลผลไม่สามารถติดต่อกับเครื่องเจ้าของงานเพื่อส่งเอาต์พุตโดยตรงได้ จึงจำเป็นต้องส่งเอาต์พุตไปยังเครื่องเซิร์ฟเวอร์เพื่อให้เครื่องเซิร์ฟเวอร์ช่วยส่งต่อให้กับเครื่องเจ้าของงานแทน ซึ่งเราเรียกกรณีเช่นนี้ว่าการเกิดความไม่เสถียรของเครือข่ายแบบเพียร์-ทู-เพียร์

3.3 การศึกษาโดยใช้แบบจำลอง

ในส่วนนี้ผู้วิจัยได้ทำการจำลองระบบที่มีการสื่อสารทั้งแบบไคลเอนต์/เซิร์ฟเวอร์และแบบที่นำการสื่อสารแบบเพียร์-ทู-เพียร์มาประยุกต์ใช้ เพื่อศึกษาถึงประสิทธิภาพในการทำงานและศึกษาถึงผลกระทบของปัจจัยต่างๆ ที่มีผลต่อประสิทธิภาพของระบบ ในการทดลองได้ใช้โปรแกรมจำลอง GridSim Toolkit 2.2 [27] จำลองระบบทั้ง 2 แบบขึ้น แต่เนื่องจาก GridSim Toolkit 2.2 ใช้รูปแบบของเครือข่ายการส่งข้อมูลแบบง่ายๆ (ไม่มี Network sharing) ซึ่งไม่ตรงกับลักษณะการทำงานของระบบในสภาพแวดล้อมจริง ผู้วิจัยจึงได้สร้างแบบจำลองของเครือข่ายขึ้นมาใหม่ให้สามารถจำลองการทำงานของระบบเครือข่ายได้ถูกต้องมากยิ่งขึ้น

เนื่องจากผู้วิจัยต้องการมุ่งเน้นที่ระบบขนาดเล็กที่สร้างขึ้นได้จากทรัพยากรที่มีอยู่อย่างแพร่หลายในปัจจุบัน ค่าตัวแปรต่างๆ ในการทดลองนี้จึงอ้างอิงตามคุณสมบัติของเซิร์ฟเวอร์และคอมพิวเตอร์ที่มีอยู่ทั่วไป

คุณลักษณะของเครื่องที่รับงานไปประมวลผล

เครื่องที่รับงานมาประมวลผลหมายถึงเครื่องคอมพิวเตอร์ของสมาชิกในระบบที่ยินดีรับงานมาประมวลผล โดยจำนวนเครื่องที่รับงานไปประมวลผลในการทดลองนี้จะหมายถึงจำนวนเครื่องในระบบที่สามารถรับงานมาประมวลผลได้ในขณะนั้น (ไม่รวมถึงเครื่องที่ไม่สามารถรับงานได้เนื่องจากเจ้าของเครื่องกำลังใช้งานอยู่) เครื่องคอมพิวเตอร์ในระบบจะมีความเร็วในการประมวลผลระหว่าง 250-350 MIPS (Million Instructions Per Second) ซึ่งอ้างอิงตามความเร็วของคอมพิวเตอร์ส่วนบุคคลที่มีความเร็วประมาณ 1 GHz หน่วยความจำหลัก 256 Mbytes [28] โดยกำหนดให้แต่ละเครื่องจะรับงานมาประมวลผลครั้งละ 1 งานเท่านั้น และเชื่อมต่ออินเตอร์เน็ตด้วยโมเด็มที่ส่งข้อมูลที่อัตราเร็ว 56 Kbit/s

คุณลักษณะของงาน

งานที่ใช้ในการทดลองนี้แต่ละงานจะเป็นอิสระต่อกัน (ไม่มีการสื่อสารระหว่างแต่ละงาน) แต่ละงานจะใช้ซีพียูเดียวในการประมวลผลและประมวลผลจนเสร็จโดยไม่มีการย้ายไปประมวลผลต่อยังเครื่องอื่น โดยงานจะเป็นงานที่มีขนาดอินพุตเล็กแต่เอาต์พุตมีขนาดใหญ่ซึ่งตรงกับโปรแกรมประยุกต์หลายๆ ประเภท ซึ่งจะกำหนดให้แต่ละงานจะประกอบด้วยอินพุตขนาด 0.5 Mbytes, *Job Ad.* ขนาด 5 Kbytes และเอาต์พุตขนาด 1 ถึง 3 Mbytes ขนาดของงาน (จำนวนคำสั่ง) จะมีการกระจายแบบเลขชี้กำลัง (Exponential distribution) ด้วยค่าเฉลี่ยเท่ากับ 1,620,000 MI (Million Instructions) ซึ่งจะใช้เวลาในการประมวลผลประมาณ 75-100 นาทีบนเครื่องที่ใช้ในการทดสอบ งานจะถูกส่งเข้ามายังระบบด้วยอัตราเฉลี่ยทุกๆ 60 วินาที โดยมีการกระจายแบบเลขชี้กำลังเช่นเดียวกัน

คุณลักษณะของเครื่องเซิร์ฟเวอร์

เครื่องเซิร์ฟเวอร์มีหน้าที่หลักในการจัดลำดับของงานและมอบหมายงานให้แก่เครื่องที่รับงานไปประมวลผลในระบบอัลกอริทึมที่ใช้ในการจัดลำดับงานคือ FCFS (First-Come-First-Serve) เนื่องจากอัลกอริทึมนี้ง่ายแก่การอิมพลีเมนต์และให้ประสิทธิภาพสูงในกรณีที่ไม่สามารถคาดเดาขนาดและการมาของงานได้ [29] และกำหนดให้เครื่องเซิร์ฟเวอร์มีแบนด์วิดธ์เท่ากับ 512 Kbit/s

คุณลักษณะของเครือข่าย

รูปแบบการสื่อสารนี้จะสามารถสามารถรองรับ Network sharing ได้ โดยมีสมมติฐานว่าเครื่องที่อยู่ในระบบจะมีการกระจายกันอยู่บนผู้ให้บริการอินเตอร์เน็ต (ISP) จำนวนมาก ซึ่งเป็นการกระจายภาระในการส่งข้อมูลออกไปเพื่อไม่ให้เกิดคอขวดขึ้นที่ตัวผู้ให้บริการอินเตอร์เน็ตซึ่งจะมีผลต่อแบนด์วิดธ์จริงที่ผู้รับบริการจะได้รับ ดังนั้นจึงมีสมมติฐานว่าแต่ละเครื่องสามารถส่งข้อมูลด้วยอัตราเร็วคงที่ตามที่กำหนดข้างต้น

ตารางที่ 3.1 แสดงค่าตัวแปรต่างๆ ที่ใช้ในการทดลองนี้ ซึ่งค่าตัวแปรที่เลือกมานี้เป็นเพียงรูปแบบหนึ่งที่เป็นไปได้ในเครือข่ายอินเตอร์เน็ตปัจจุบันเท่านั้น

ตารางที่ 3.1 ค่าตัวแปรต่างๆ ที่ใช้ในการทดลองเพื่อวัดประสิทธิภาพของระบบ

ตัวแปร	ค่าที่ใช้
จำนวนงานในระบบ (J)	5000
ขนาดไฟล์อินพุต (D_{in})	0.5 Mbytes
ขนาดไฟล์เอาต์พุต (D_{out})	1 - 3 Mbytes
ขนาด *Job Ad.* และ *Res Ad.*	5 Kbytes
จำนวนคำสั่งในแต่ละงาน	Exp(1,620,000) MI
ความเร็วในการประมวลผลของเครื่องที่รับงานไปประมวลผล	250 – 350 MIPS
ขนาดแบนด์วิดธ์ของเครื่องเซิร์ฟเวอร์ (S_{BW})	512 Kbit/s
ขนาดแบนด์วิดธ์ของเครื่องเจ้าของงานและเครื่องที่รับงานไปประมวลผล (U_{BW})	56 Kbit/s
ระยะเวลาที่งานต่อไปจะถูกส่งเข้ามาในระบบ (τ)	Exp(60) seconds

3.4 รูปแบบการทดลอง

ในระบบแบบเพียร์-ทู-เพียร์นั้นจะมีความไม่เสถียรของเครือข่ายเพียร์-ทู-เพียร์ ทำให้บางเพียร์อาจไม่สามารถติดต่อกับอีกเพียร์หนึ่งโดยตรงได้เนื่องด้วยข้อจำกัดบางอย่างเช่นระบบอยู่ภายใต้ Firewall หรือ NAT ดังนั้นในบางครั้งเมื่อเพียร์ที่รับงานไปประมวลผลทำการประมวลผลงานเสร็จแล้วแต่ไม่สามารถติดต่อกับเครื่องเจ้าของงานเพื่อส่งเอาต์พุตกลับไปโดยตรงได้ ในกรณีนี้เอาต์พุตจะต้องถูกส่งไปที่เครื่องเซิร์ฟเวอร์เพื่อให้เครื่องเซิร์ฟเวอร์ช่วยส่งเอาต์พุตนี้ต่อไปยังเครื่องเจ้าของงาน ดังนั้นถ้าระบบเกิดกรณีดังกล่าวมากขึ้นแล้วปริมาณการใช้แบนด์วิดธ์ของเครื่องเซิร์ฟเวอร์ก็จะเพิ่มมากขึ้นด้วยและจะกระทบต่อประสิทธิภาพของระบบ การศึกษาถึงผลกระทบของกรณีดังกล่าวจึงเป็นสิ่งสำคัญที่ต้องคำนึงถึง

เพื่อที่จะศึกษาถึงผลกระทบของความไม่เสถียรของเครือข่ายในระบบเพียร์-ทู-เพียร์ ค่าความน่าจะเป็น “*P*” จึงถูกกำหนดขึ้น ซึ่งแสดงถึงโอกาสที่การเชื่อมต่อแบบเพียร์-ทู-เพียร์ระหว่างเครื่องที่รับงานไปประมวลผลกับเครื่องเจ้าของงานจะสามารถเกิดขึ้นได้ ณ เวลาที่งานถูกประมวลผลเสร็จ ดังนั้นถ้าค่า $P = 1.0$ นั่นหมายความว่าเอาต์พุตไฟล์ทุกไฟล์จะสามารถส่งกลับไปยังเครื่องเจ้าของงานได้โดยตรงโดยไม่ต้องผ่านเครื่องเซิร์ฟเวอร์ ดังนั้นยิ่งค่า P มีค่าน้อยลงจะทำให้ปริมาณการใช้แบนด์วิดธ์ของเครื่องเซิร์ฟเวอร์เพิ่มมากขึ้นด้วย ในส่วนหลังจะทำการทดลองเพื่อศึกษาถึงผลกระทบของค่า P นี้เทียบกับโครงสร้างแบบไคลเอนต์/เซิร์ฟเวอร์

อีกปัจจัยหนึ่งที่มีความสำคัญมากต่อประสิทธิภาพของระบบนั่นคือจำนวนเครื่องที่รับงานไปประมวลผลในระบบ ในระบบเพียร์-ทู-เพียร์แบนด์วิดธ์โดยรวมของระบบจะขึ้นอยู่กับจำนวนเครื่องในระบบด้วย ยิ่งระบบมีจำนวนเครื่องมากแบนด์วิดธ์ของระบบก็จะมากขึ้นด้วย จึงจำเป็นที่จะต้องศึกษาถึงผลที่ได้รับเมื่อจำนวนเครื่องที่รับงานไปประมวลผลเพิ่มมากขึ้นด้วย

3.5 ผลการทดลองและวิเคราะห์ผล

ผู้วิจัยได้ทำการทดลองและวิเคราะห์ถึงผลกระทบของปัจจัยต่างๆ ที่มีผลกระทบต่อประสิทธิภาพการทำงานของระบบโดยตรง ซึ่งได้แก่ ขนาดไฟล์เอาต์พุต, ความเสถียรในระบบเพียร์-ทู-เพียร์ (ค่า *P*), จำนวนเครื่องที่รับงานไปประมวลผล และขนาดแบนด์วิดธ์ของเครื่องเซิร์ฟเวอร์

3.5.1 ผลกระทบจากขนาดไฟล์เอาต์พุต

ข้อมูลที่ส่งไปมาในระบบประกอบด้วย *Job Ad*, *Res Ad*, อินพุต, และเอาต์พุต ซึ่ง *Job Ad*, *Res Ad*, และอินพุตในการทดลองนี้มีขนาดมากเมื่อเปรียบเทียบกับเอาต์พุตไฟล์ ดังนั้นในการทดลองนี้ผู้วิจัยจึงได้ศึกษาเพียงผลกระทบของขนาดของเอาต์พุตไฟล์เท่านั้น (D_{out})

ขนาดของเอาต์พุตไฟล์จะถูกจำกัดในแต่ละระบบเพื่อป้องกันปัญหาคอขวดที่อาจเกิดขึ้นที่เครื่องเซิร์ฟเวอร์ ในระบบแบบไคลเอนต์/เซิร์ฟเวอร์นั้นขนาดเอาต์พุตไฟล์จะถูกจำกัดโดยขนาดแบนด์

วิดธ์ของเครื่องเซิร์ฟเวอร์ ส่วนในระบบแบบเพียร์-ทู-เพียร์จะถูกจำกัดโดยผลรวมของแบนด์วิดธ์ของสมาชิกในระบบและค่า P

$$P = 1.0$$

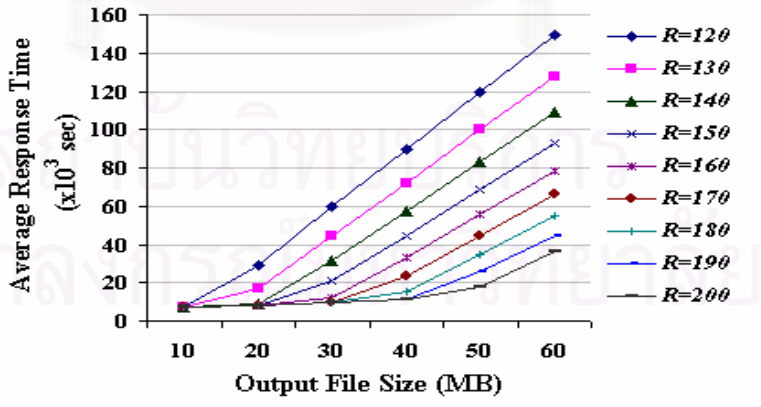
ถ้า P = 1.0 เอาต์พุตไฟล์ทุกตัวจะถูกส่งกลับโดยตรงไปยังเจ้าของงาน เครื่องเซิร์ฟเวอร์จึงแบกรับเพียง *Job Ad*, *Res Ad*, และอินพุตไฟล์บางตัวเท่านั้น ซึ่งล้วนมีขนาดเล็ก ดังนั้นปัญหาคอขวดจึงเกิดขึ้นเนื่องจากความล่าช้าในการส่งเอาต์พุตไฟล์ของเครื่องที่รับงานไปประมวลผลแทน การกระจายงานในระบบจะเปรียบได้กับรูปแบบของ Round-robin ดังนั้นแต่ละเครื่องจะได้รับงานใหม่ในอีก $\tau_R = R \bullet \tau$ วินาทีข้างหน้า (เมื่อ R คือจำนวนเครื่องที่รับงานไปประมวลผลในระบบ) ดังนั้นแต่ละงานจะต้องถูกประมวลผลให้เสร็จภายในระยะเวลา τ_R มิฉะนั้นปัญหาคอขวดจะเกิดขึ้น

$$\tau_R \leq T_{comp} + T_{comm} \qquad (3.2)$$

ดังนั้นคอขวดจะเกิดขึ้นถ้า

$$D_{out} \leq (\tau_R - T_{comp}) \bullet U_{BW} \qquad (3.3)$$

ระบบได้ถูกจำลองขึ้นด้วยค่าตัวแปรที่อยู่ในตารางที่ 1 เพื่อพิสูจน์สมการที่ 3.3 จากรูปที่ 3.3 เราพบว่าขีดจำกัดของขนาดเอาต์พุตไฟล์ใกล้เคียงกับค่าที่ได้จากสมการที่ 3.3 และจากรูปจะเห็นได้ว่ายิ่งจำนวนเครื่องที่รับงานไปประมวลผลในระบบมากขึ้นซึ่งหมายความว่าแบนด์วิดธ์โดยรวมในระบบก็มากขึ้นด้วย ทำให้ระบบสามารถรองรับขนาดเอาต์พุตไฟล์ได้มากขึ้น

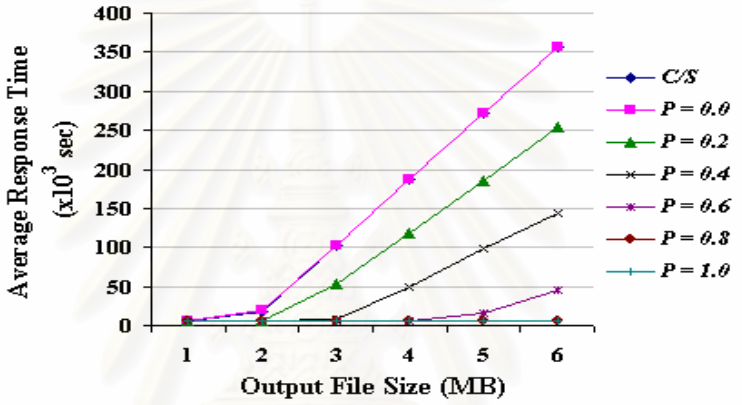


รูปที่ 3.3 ผลกระทบของขนาดเอาต์พุตที่ค่า P = 1.0

0.0 ≤ P < 1.0

เมื่อค่า P น้อยกว่า 1.0 แบนด์วิดธ์ของเครื่องเซิร์ฟเวอร์ (S_{BW}) จะเริ่มมีการถูกใช้งานมากขึ้นเพื่อส่งต่อเอาต์พุตไฟล์จากเครื่องที่รับงานไปประมวลผลไปยังเครื่องเจ้าของงาน ทำให้ขนาดแบนด์วิดธ์ของเครื่องเซิร์ฟเวอร์กลายเป็นปัจจัยหลักต่อประสิทธิภาพของระบบด้วย ขีดจำกัดของขนาดเอาต์พุตที่ไม่ทำให้เกิดคอขวดประมาณได้ตามสมการที่ 3.4

$$D_{out} \leq \left(\frac{1}{1-P}\right) \bullet \left(\frac{S_{BW}}{2}\right) \bullet \tau \qquad (3.4)$$

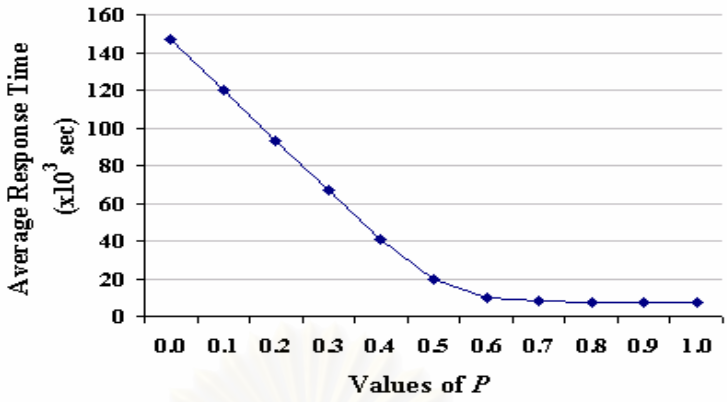


รูปที่ 3.4 ผลกระทบของขนาดเอาต์พุตที่ค่า $0.0 \leq P < 1.0$

รูปที่ 3.4 แสดงผลจากโปรแกรมจำลองที่จำลองระบบให้มีจำนวนเครื่องที่รับงานไปประมวลผล 150 เครื่อง และทำการเปลี่ยนค่า P และ D_{out} เพื่อดูว่าระบบจะสามารถรองรับขนาดเอาต์พุตไฟล์ได้เท่าใด ยิ่งระบบมีค่า P มาก (มีการใช้การสื่อสารแบบเพียร์-ทู-เพียร์มากขึ้น) ระบบจะสามารถรองรับขนาดเอาต์พุตไฟล์ได้มากขึ้นด้วย

3.5.2 ผลกระทบจากความไม่เสถียรของเครือข่ายแบบเพียร์-ทู-เพียร์

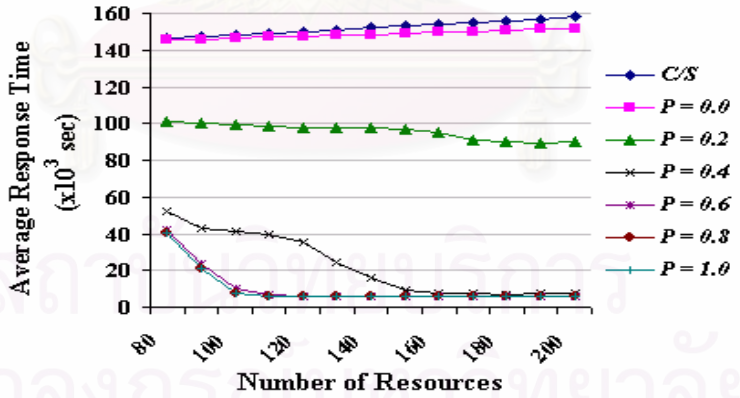
รูปที่ 3.5 แสดงถึงผลของความไม่เสถียรของเครือข่ายแบบเพียร์-ทู-เพียร์ที่มีต่อประสิทธิภาพของระบบตามตัวแปรในตารางที่ 1 และเครื่องที่รับงานไปประมวลผลจำนวน 100 เครื่อง จากรูปจะเห็นได้ว่าเมื่อค่า P เพิ่มขึ้นเพียงเล็กน้อยค่า Response time จะลดลงอย่างรวดเร็วจนคงที่เมื่อค่า P มีค่าเท่ากับ 0.6 นั่นหมายความว่าระบบมีค่า P = 0.6 ก็เพียงพอที่จะทำให้ระบบไม่เกิดปัญหาคอขวดได้ แสดงให้เห็นว่าการที่ระบบสามารถใช้รูปแบบการสื่อสารแบบเพียร์-ทู-เพียร์ได้เพียงบางส่วนก็สามารถช่วยแก้ปัญหาคอขวดของเครือข่ายที่เครื่องเซิร์ฟเวอร์ได้



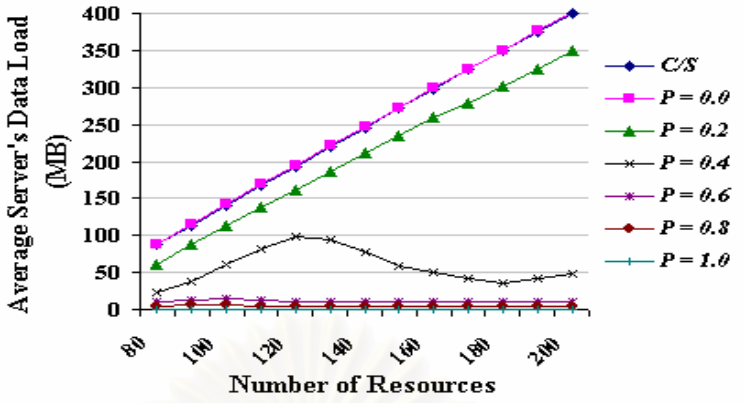
รูปที่ 3.5 ผลกระทบจากความไม่เสถียรของเครือข่ายแบบเพียร์-ทู-เพียร์

3.5.3 ผลกระทบจากจำนวนเครื่องที่รับงานไปประมวลผลในระบบ

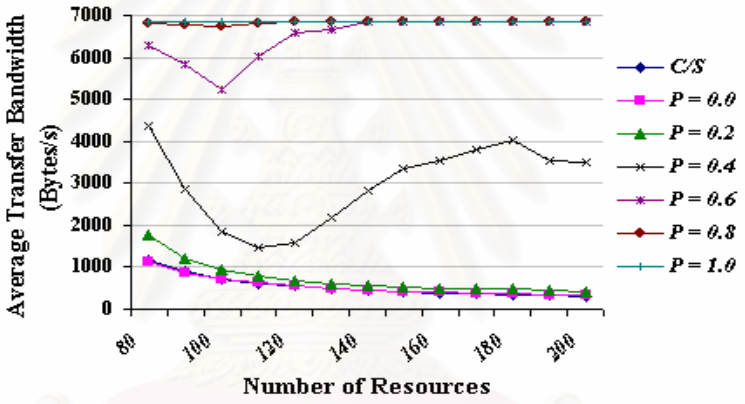
จำนวนเครื่องที่ระบบสามารถรองรับได้เป็นตัวบ่งชี้ถึงความสามารถในการขยายตัวของระบบเหล่านั้นได้ ในระบบแบบไคลเอนต์/เซิร์ฟเวอร์ถึงแม้ว่าการเพิ่มขึ้นของจำนวนเครื่องที่รับงานไปประมวลผลจะช่วยให้เวลาในการประมวลผล (T_{comp}) ลดลง แต่กลับเพิ่มปริมาณข้อมูลที่เครื่องเซิร์ฟเวอร์ต้องแบกรับซึ่งถ้าปริมาณข้อมูลนั้นมากเกินกว่าที่แบนด์วิดธ์ของเครื่องเซิร์ฟเวอร์จะรับได้ก็จะเกิดปัญหาคอขวดขึ้นซึ่งจะทำให้เวลาในการส่งข้อมูล (T_{comm}) มากขึ้นแบบทวีคูณส่งผลให้ค่า Response time ของแต่ละงานเพิ่มขึ้นแทนที่จะลดลง จำนวนเครื่องที่ระบบสามารถรองรับได้



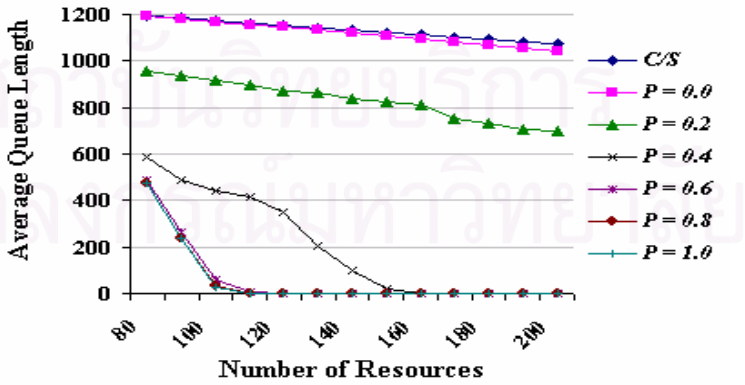
รูปที่ 3.6 ผลกระทบของจำนวนเครื่องที่รับงานไปประมวลผลต่อค่า Response time เฉลี่ย



รูปที่ 3.7 ผลกระทบของจำนวนเครื่องที่รับงานไปประมวลผลต่อ
ปริมาณข้อมูลที่เครื่องเซิร์ฟเวอร์ต้องแบกรับ ณ เวลาหนึ่งๆ



รูปที่ 3.8 ผลกระทบของค่าจำนวนเครื่องที่รับงานไปประมวลผลต่อ
อัตราเร็วในการส่งข้อมูลของระบบโดยเฉลี่ย



รูปที่ 3.9 ผลกระทบของจำนวนเครื่องที่รับงานไปประมวลผลต่อความยาวเฉลี่ยของคิวงาน

จากรูปจะเห็นได้ว่าในระบบแบบไคลเอนต์/เซิร์ฟเวอร์ยิ่งจำนวนเครื่องที่รับงานไปประมวลผลเพิ่มมากขึ้นแทนที่ค่า Response time ของแต่ละงานจะลดลงแต่กลับเพิ่มขึ้นดังในรูปที่ 3.6 เนื่องจากปัญหาคอขวดของแบนด์วิดธ์ซึ่งทำให้อัตราเร็วในการส่งข้อมูลไปยังเครื่องต่างๆ ในระบบช้าลงอย่างมาก ดังรูปที่ 3.8 ดังนั้นยิ่งระบบมีจำนวนเครื่องเพิ่มมากขึ้นประสิทธิภาพของระบบกลับแย่ลง ด้วยเหตุนี้ระบบแบบไคลเอนต์/เซิร์ฟเวอร์จึงมีความสามารถในการขยายระบบต่ำ เหตุการณ์เช่นนี้ก็จะเกิดขึ้นกับระบบแบบเพียร์-ทู-เพียร์ที่มีค่า P ต่ำๆ เช่น $P = 0.0$ และ $P = 0.2$ เช่นเดียวกัน

ในระบบแบบเพียร์-ทู-เพียร์ที่ค่า $P = 0.4$ มีสิ่งหนึ่งที่น่าสนใจเกิดขึ้น จากรูปที่ 3.7 จะพบว่าปริมาณข้อมูลที่เซิร์ฟเวอร์ต้องแบกรับในแต่ละเวลาสามารถพิจารณาแบ่งได้เป็น 3 ช่วงด้วยกัน ในช่วงแรก ($R = 80 – 120$) ปริมาณข้อมูลที่เซิร์ฟเวอร์จะเพิ่มขึ้นตามจำนวนเครื่องที่รับงานไปประมวลผลเช่นเดียวกับระบบแบบไคลเอนต์/เซิร์ฟเวอร์ เนื่องจากจำนวนเครื่องไม่เพียงพอต่องานงานที่เข้ามาทำให้เกิด Computational bottleneck ส่งผลให้อินพุตไฟล์จำนวนมากต้องถูกส่งไปเข้าคิวที่เซิร์ฟเวอร์ ในช่วงที่สอง ($R = 120 – 180$) ในช่วงนี้จำนวนเครื่องที่รับงานไปประมวลผลเริ่มเพียงต่อการเข้ามาของงานในระบบ ดังนั้นงานส่วนใหญ่จึงไม่ต้องอยู่ในคิว ความยาวคิวเฉลี่ยจึงลดลงอย่างรวดเร็วดังในรูปที่ 3.9 ปัญหาคอขวดที่เซิร์ฟเวอร์จึงบรรเทาลงไปและทำให้ค่า Response time ของงานลดลงอย่างรวดเร็วด้วย (รูปที่ 3.6) ในช่วงที่สาม ($R = 180 – 200$) ปริมาณข้อมูลที่เซิร์ฟเวอร์ต้องแบกรับเพิ่มขึ้นอีกครั้งจากปริมาณเอาต์พุตที่เครื่องที่รับงานไปประมวลผลส่งมายังเซิร์ฟเวอร์แต่เพิ่มขึ้นในอัตราที่ต่ำกว่าช่วงแรก

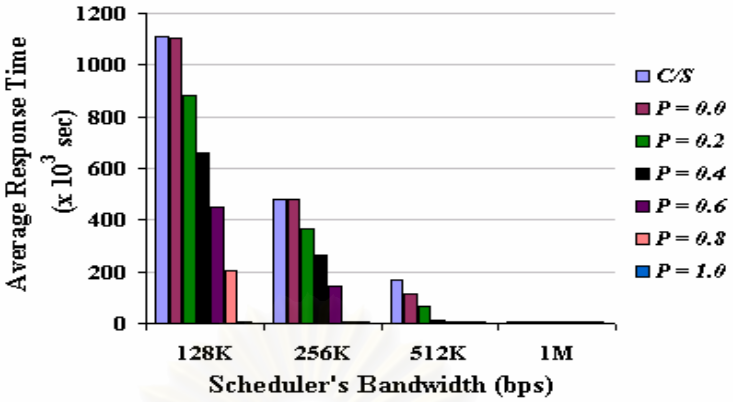
เมื่อค่า $P \geq 0.6$ ปัญหาคอขวดที่เครื่องเซิร์ฟเวอร์ได้หมดไปทำให้ระบบสามารถรองรับจำนวนเครื่องที่รับงานไปประมวลผลเพิ่มขึ้นได้เรื่อยๆ โดยไม่กระทบต่อประสิทธิภาพของระบบ

หลังจากที่นำการสื่อสารแบบเพียร์-ทู-เพียร์มาประยุกต์ใช้จะพบว่าค่า Response time ของงานลดลงเรื่อยๆ ตามจำนวนเครื่องที่รับงานไปประมวลผลในระบบ ระบบแบบเพียร์-ทู-เพียร์จึงมีความสามารถในการขยายระบบได้ดีกว่าระบบแบบไคลเอนต์/เซิร์ฟเวอร์

จำนวนเครื่องที่สามารถรองรับได้ในระบบแบบไคลเอนต์เซิร์ฟเวอร์และระบบแบบเพียร์-ทู-เพียร์ที่มีค่า P ไม่มาก จะขึ้นอยู่กับขนาดแบนด์วิดธ์ของเครื่องเซิร์ฟเวอร์เนื่องจากทั้งสองระบบนี้ไม่สามารถดึงเอาแบนด์วิดธ์ของเครื่องสมาชิกในระบบมาใช้ส่งข้อมูลได้อย่างมีประสิทธิภาพ จากการทดลองจำลองระบบโดยใช้ค่าตัวแปรตามตารางที่ 1

3.5.4 ผลกระทบจากขนาดแบนด์วิดธ์ของเครื่องเซิร์ฟเวอร์

จากที่ได้กว่ามาข้างต้นแล้วว่าประสิทธิภาพของระบบจะขึ้นอยู่กับขนาดแบนด์วิดธ์ของเครื่องเซิร์ฟเวอร์ด้วย ในส่วนนี้ผู้วิจัยจึงทำการจำลองระบบที่เครื่องเซิร์ฟเวอร์มีขนาดแบนด์วิดธ์ต่างๆ เพื่อศึกษาถึงผลกระทบที่มีต่อระบบ โดยให้ระบบมีเครื่องที่รับงานไปประมวลผลจำนวน 150 เครื่อง



รูปที่ 3.10 ผลกระทบของขนาดแบนด์วิดธ์ของเครื่องเซิร์ฟเวอร์

จากรูปที่ 3.10 เราพบว่าในระบบแบบไคลเอนต์/เซิร์ฟเวอร์เมื่อเครื่องเซิร์ฟเวอร์มีขนาดแบนด์วิดธ์ลดลงประสิทธิภาพของระบบจะลดลงอย่างมาก แต่เมื่อนำการสื่อสารแบบเพียร์-ทู-เพียร์มาประยุกต์ใช้ พบว่าปัญหาได้บรรเทาลง ยิ่งระบบมีค่า P มากขึ้น ความต้องการแบนด์วิดธ์ของเครื่องเซิร์ฟเวอร์ก็จะลดลงด้วย เช่นระบบที่มีค่า $P = 0.8$ เครื่องเซิร์ฟเวอร์มีแบนด์วิดธ์เพียง 256 Kbit/s ก็เพียงพอที่จะไม่ทำให้ระบบเกิดปัญหาคอขวดขึ้น

3.6 สิ่งที่ได้รับจากการทดลอง

จากการทดลองเราได้พบผลกระทบจากปัจจัยต่างๆ ต่อประสิทธิภาพของระบบคำนวณแบบกระจายที่เกิดกับระบบแบบไคลเอนต์/เซิร์ฟเวอร์ และพิสูจน์ได้ว่าการนำการสื่อสารแบบเพียร์-ทู-เพียร์มาประยุกต์ใช้กับระบบคำนวณแบบกระจายจะช่วยบรรเทาผลกระทบเหล่านั้นได้ และช่วยเพิ่มประสิทธิภาพการทำงานของระบบทั้งในด้านของการเพิ่มขนาดไฟล์ข้อมูล, เพิ่มความสามารถในการขยายระบบ, และลดความต้องการแบนด์วิดธ์ของเครื่องเซิร์ฟเวอร์

อย่างไรก็ตามหากระบบดังกล่าวต้องรองรับขนาดของข้อมูลที่มากเกินกว่าระบบจะสามารถรองรับและระบบมีความเสถียรของเครือข่ายเพียร์-ทู-เพียร์ต่ำ (ค่า P ต่ำ) ปัญหาคอขวดที่เครื่องเซิร์ฟเวอร์ก็จะยังคงเกิดขึ้น ดังนั้นจึงจำเป็นต้องออกแบบโครงสร้างให้สามารถป้องกันการเกิดปัญหาดังกล่าวได้อย่างแท้จริง

บทที่ 4

ระบบการคำนวณแบบกระจายบนเครือข่ายแบบเพียร์-ทู-เพียร์ โดยใช้โมเดลแบบซุปเปอร์เพียร์

จากการทดลองในบทที่ 3 แสดงให้เห็นว่าการนำรูปแบบการสื่อสารแบบเพียร์-ทู-เพียร์มาประยุกต์ใช้กับระบบการคำนวณแบบกระจาย สามารถช่วยลดปริมาณการใช้แบนด์วิดธ์ของเครื่องเซิร์ฟเวอร์ซึ่งเป็นต้นเหตุของปัญหาคอขวดและค่าใช้จ่ายด้านแบนด์วิดธ์ของเครื่องเซิร์ฟเวอร์ลงได้ และส่งผลให้ประสิทธิภาพการทำงานของระบบเพิ่มขึ้นอย่างเห็นได้ชัด

อย่างไรก็ตาม โครงสร้างของระบบในบทที่ 3 จะมีเครื่องเซิร์ฟเวอร์ที่คอยให้บริการเครื่องไคลเอนต์ในระบบเพียงเครื่องเดียว ซึ่งสามารถรองรับขนาดของไฟล์ข้อมูลได้ระดับหนึ่ง โดยขึ้นอยู่กับขนาดแบนด์วิดธ์ของเครื่องเซิร์ฟเวอร์และค่าความเสถียรของเครือข่ายเพียร์-ทู-เพียร์ของระบบ ดังสมการที่ 3.4 ซึ่งหากขนาดของไฟล์ข้อมูลมากเกินกว่าที่ระบบสามารถรองรับได้และเครือข่ายเพียร์-ทู-เพียร์มีความเสถียรต่ำ (ค่า P ต่ำ) ปัญหาคอขวดที่เครื่องเซิร์ฟเวอร์ก็จะยังคงเกิดขึ้น นอกจากนั้นหากเครื่องเซิร์ฟเวอร์ที่มีอยู่เพียงเครื่องเดียวไม่สามารถให้บริการได้ ระบบก็จะไม่สามารถทำงานต่อไปได้ด้วย นั่นคือการเกิดปัญหา Single point of failure

เพื่อเป็นการเพิ่มขีดความสามารถของระบบและแก้ปัญหา Single point of failure จึงนำโมเดลแบบซุปเปอร์เพียร์ (รูปที่ 2.10) ที่ได้รับการยอมรับว่าเป็นโมเดลที่มีประสิทธิภาพในการทำงานสูงกว่าโมเดลชนิดอื่นๆ [30] มาใช้เป็นโครงสร้างของระบบ

4.1 การใช้โมเดลแบบซุปเปอร์เพียร์เพื่อเพิ่มประสิทธิภาพของระบบการคำนวณแบบกระจาย

ซุปเปอร์เพียร์ (Super peer) คือ เครื่องที่ทำหน้าที่เป็นเซิร์ฟเวอร์ให้กับกลุ่มไคลเอนต์ โดยไคลเอนต์แต่ละตัวจะต้องอาศัยอยู่กับซุปเปอร์เพียร์ตัวใดตัวหนึ่ง โดยไคลเอนต์จะส่ง Query และรับผลลัพธ์ของ Query นั้นจากซุปเปอร์เพียร์ที่ตนอาศัยอยู่เท่านั้น นอกจากนั้นซุปเปอร์เพียร์ยังทำหน้าที่ติดต่อสื่อสารและส่งต่อ Query ไปยังซุปเปอร์เพียร์ตัวอื่นๆ ในระบบกรณีที่ตนไม่สามารถตอบคำถาม Query นั้นได้ เพื่อให้ซุปเปอร์เพียร์สามารถให้บริการไคลเอนต์ได้ ซุปเปอร์เพียร์แต่ละตัวจึงจำเป็นต้องมีฐานข้อมูลที่เก็บข้อมูลและรายละเอียดที่จำเป็นของไคลเอนต์ที่ตนดูแลอยู่ไว้ด้วย เช่นข้อมูลด้านฮาร์ดแวร์และซอฟแวร์, ข้อมูลด้านเครือข่ายและการติดต่อสื่อสาร, และข้อมูลทรัพยากรที่บริจาคให้กับระบบ และทำการปรับปรุงฐานข้อมูลดังกล่าวทุกครั้งที่ข้อมูลมีการเปลี่ยนแปลง เช่น มีไคลเอนต์ใหม่เข้ามายังระบบ, มีไคลเอนต์ออกไปจากระบบ, และข้อมูลของไคลเอนต์มีการเปลี่ยนแปลง ซึ่ง Overhead ในการดูแลรักษาฐานข้อมูลนั้นมีค่าน้อยมากเมื่อเทียบกับประสิทธิภาพที่ได้รับ [30]

ในโมเดลแบบซุปเปอร์เพียร์นั้นหากซุปเปอร์เพียร์ตรวจพบว่าภาระที่ตนรับอยู่ เช่น ภาระในการรับส่งข้อมูล มากเกินกว่าความสามารถของตน ระบบจะทำการคัดเลือกเครื่องไคลเอนต์ที่มี

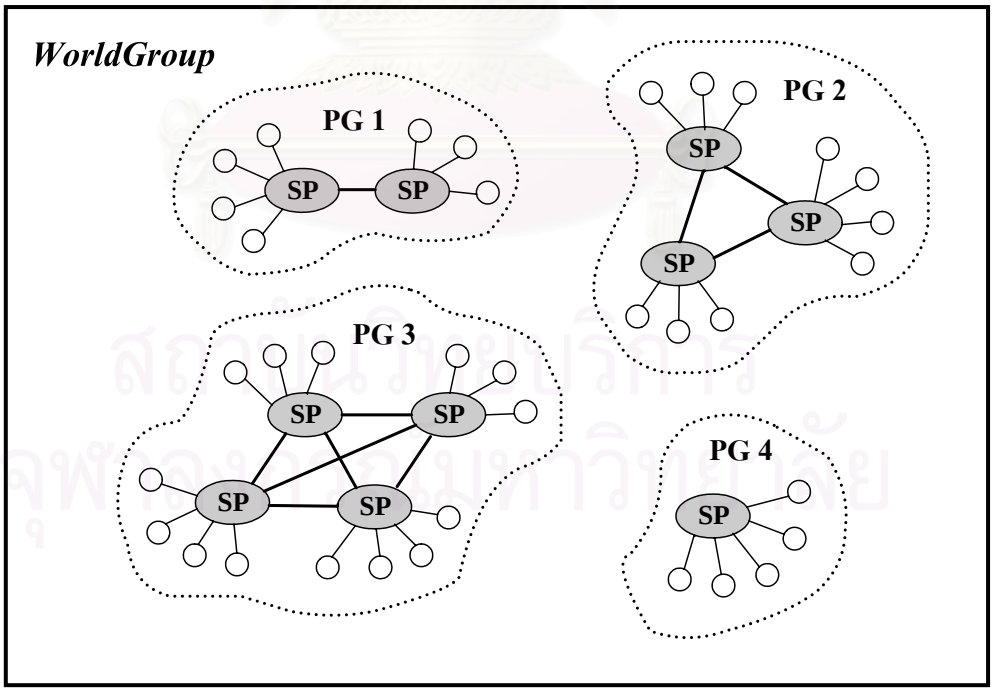
ความเหมาะสมมาเป็นซุปเปอร์เพียร์อีกตัวหนึ่งเพื่อแบ่งเบาภาระของซุปเปอร์เพียร์ตัวเดิม ซึ่งซุปเปอร์เพียร์ที่เพิ่มขึ้นนี้ช่วยให้ความสามารถในการขยายขนาดของระบบเพิ่มมากขึ้น นอกจากนั้นยังเป็นการเพิ่มความทนทานของระบบ (Robustness) อีกด้วยเนื่องจากหากซุปเปอร์เพียร์ตัวใดไม่สามารถให้บริการได้ ซุปเปอร์เพียร์อีกตัวก็จะเข้ามารับหน้าที่แทนช่วยให้ระบบยังสามารถทำงานต่อไปได้

4.2 โครงสร้างและรูปแบบการทำงานของระบบที่ใช้โมเดลแบบซุปเปอร์เพียร์

ทุกเพียร์ในระบบจะอยู่ภายใต้ Peer group ที่ชื่อว่า *WorldGroup* ภายใน WorldGroup จะประกอบด้วยหลายๆ Peer group ซึ่งแต่ละ Peer group คือกลุ่มของเพียร์ที่รวมตัวกันเพื่อร่วมกันทำงานที่อยู่ภายใน Peer group นั้น ผู้ใช้สามารถสร้าง Peer group ใหม่และกำหนดนโยบายต่างๆ ของ Peer group นั้นได้ ผู้ที่ต้องการบริจาคทรัพยากรการคำนวณของเครื่องตนก็สามารถเข้าร่วมเป็นสมาชิกของ Peer group ภายใต้นโยบายของ Peer group นั้น

ในแต่ละ Peer group จะประกอบด้วยเพียร์ 2 ประเภทคือ SuperPeer และ Volunteer โดยเพียร์หนึ่งๆ สามารถเป็นได้ทั้ง SuperPeer และ Volunteer ซึ่ง ณ เวลาหนึ่งอาจเป็นเพียง SuperPeer หรือ Volunteer อย่างใดอย่างหนึ่ง หรือเป็นทั้ง 2 ประเภทในเวลาเดียวกันก็ได้

รูปที่ 4.1 แสดงภาพรวมของระบบที่ประกอบด้วย Peer group ทั้งหมด 4 Peer groups ซึ่งแต่ละ Peer group จะมีจำนวนสมาชิกแตกต่างกัน



รูปที่ 4.1 โครงสร้างโดยรวมของระบบที่ใช้โมเดลแบบซุปเปอร์เพียร์

4.2.1 Advertisement

ในระบบที่เครื่องคอมพิวเตอร์ภายในระบบมีความหลากหลาย ทั้งในด้านของสถาปัตยกรรม, ประสิทธิภาพ, ฮาร์ดแวร์ (เช่น หน่วยประมวลผลกลาง, ขนาดหน่วยความจำหลัก, และขนาดหน่วยความจำสำรอง เป็นต้น), และซอฟต์แวร์ (เช่น ระบบปฏิบัติการ, และโปรแกรมประยุกต์ภายในเครื่อง เป็นต้น) รวมถึงความหลากหลายของงานที่ถูกส่งเข้ามายังระบบ วิธีการที่จะทำให้ระบบสามารถจัดการและรับรู้ถึงความหลากหลายเหล่านี้ได้อย่างมีประสิทธิภาพคือ เทคนิคของการทำ Advertisement [31] โดยการทำงานคือ แต่ละหน่วยในระบบซึ่งได้แก่ เครื่องคอมพิวเตอร์ และงาน จะทำการประกาศคุณสมบัติและความต้องการของตนไปยังตัวที่ทำหน้าที่จับคู่ซึ่งก็คือ Matchmaker ซึ่งอยู่ที่ตัวซุปเปอร์เพียร์ เมื่อ Matchmaker สามารถหาคู่ที่เหมาะสมกันได้แล้วก็จะแจ้งให้ทั้งคู่รับรู้ถึงการจับคู่นั้นเพื่อให้ทั้งคู่ดำเนินงานต่อไป

ระบบในงานวิจัยนี้หน่วยที่จะต้องทำ Advertisement คือ เครื่องคอมพิวเตอร์และงาน ดังนั้น Advertisement ในระบบนี้จะประกอบด้วย 2 ชนิดคือ *Resource Advertisement (Res Ad)* ซึ่งบ่งบอกถึงรายละเอียดของเครื่องและข้อกำหนดในการใช้งานเครื่องนั้น และ *Job Advertisement (Job Ad)* ซึ่งบ่งบอกถึงข้อมูลต่างๆ ที่จำเป็นในการทำงานนั้น เช่น คำสั่ง และความต้องการขั้นต่ำของเครื่องที่สามารถทำงานนี้ได้ เป็นต้น โดยรายละเอียดของ Advertisement ทั้ง 2 ชนิดจะกล่าวถึงในบทที่ 5

4.2.2 SuperPeer

ในแต่ละ Peer group จะต้องมีเพียร์ที่ทำหน้าที่เป็น SuperPeer อย่างน้อย 1 เพียร์เพื่อทำหน้าที่เป็นเซิร์ฟเวอร์คอยให้บริการต่างๆ แก่กลุ่มของ Volunteer ที่ SuperPeer นั้นดูแลอยู่ โดย SuperPeer แต่ละตัวจะเชื่อมต่อถึงกันแบบ Pure P2P

หน้าที่หลักของ SuperPeer ประกอบด้วย

1. **ตรวจสอบสถานะของ Volunteer:** ในแต่ละ SuperPeer จะมีฐานข้อมูลที่เก็บรายละเอียดและสถานะของเหล่า Volunteer ที่ SuperPeer นั้นดูแลอยู่ และทำการ Update สถานะของ Volunteer นั้นเมื่อสถานะเปลี่ยน
2. **จัดลำดับงาน:** ภายใน SuperPeer จะมีตัวจัดลำดับงาน (Scheduler) ที่ทำหน้าที่จัดลำดับงานที่ถูกส่งเข้ามาโดย Volunteer ที่ตนดูแลอยู่ หรืออาจมาจาก SuperPeer ตัวอื่นที่เหล่า Volunteer ของ SuperPeer ตัวนั้นไม่สามารถประมวลผลงานนั้นได้ ซึ่งอัลกอริทึมที่ใช้ในการจัดลำดับงานในงานวิจัยนี้ได้เลือกอัลกอริทึมแบบ First-Come-First-Serve (FCFS) เนื่องจากอิมพลีเมนต์ง่ายและมีประสิทธิภาพดีเมื่อลักษณะงานในระบบไม่สามารถทำนายได้
3. **หาคู่ที่เหมาะสม:** จะทำโดย Matchmaker ซึ่งเป็นการจับคู่ระหว่างงานและ Volunteer ที่เหมาะสม โดยจะใช้ข้อมูลที่อยู่ใน Job Ad และ Res Ad ในการพิจารณา

ในการจับคู่จะทำการหา Volunteer ทั้งหมดที่สามารถประมวลผลงานนั้นได้ เช่นมีคุณสมบัติและโปรแกรมตรงตามที่งานนั้นต้องการ และทำการเลือก Volunteer ที่มีประสิทธิภาพดีที่สุดโดยดูจาก ขนาดแบนด์วิดธ์, หน่วยความจำ, ความเร็วซีพียู ฯลฯ เพื่อประมวลผลงานนั้น

4. **เก็บผลลัพธ์ของงานชั่วคราว:** เนื่องจากความไม่เสถียรของเครือข่ายแบบเพียร์-ทู-เพียร์ที่แต่ละเพียร์สามารถเข้า-ออกระบบได้อย่างอิสระ ดังนั้นในบางครั้งที่เมื่อ Volunteer ประมวลผลงานเสร็จแล้ว แต่ไม่สามารถติดต่อกับเจ้าของงานเพื่อส่งเอาต์พุตไฟล์นั้นกลับไปได้โดยตรง SuperPeer จึงต้องทำหน้าที่รับเอาต์พุตไฟล์นั้นมาเก็บไว้ชั่วคราว เพื่อให้ Volunteer นั้นสามารถรับงานใหม่ไปทำได้ และเมื่อเจ้าของงานกลับเข้ามายังระบบอีกครั้ง SuperPeer ก็จะส่งเอาต์พุตไฟล์กลับไปให้เจ้าของงานแทน
5. **ติดต่อสื่อสารกับ SuperPeer อื่นๆ:** แต่ละ SuperPeer จะเก็บและคอย Update สถานะของ SuperPeer ทั้งหมดใน Peer group นั้นเพื่อใช้ในการส่ง Job Ad ไปให้กรณีที่ Volunteer ของตนไม่สามารถประมวลผลงานบางงานได้

4.2.3 Volunteer

Volunteer คือเพียร์ที่มีหน้าที่รับงานในระบบมาประมวลผล และส่งผลลัพธ์กลับคืนไปยังเจ้าของงาน ในบางเวลา Volunteer อาจเป็นคนส่งงานเข้าสู่ระบบได้ด้วย ดังนั้น ในแต่ละ Volunteer จะสามารถทำหน้าที่ได้ 2 อย่างคือ รับงานไปประมวลผล และส่งงานเข้าสู่ระบบโดยจะเรียกเพียร์นั้นว่าเจ้าของงาน (Submitter)

เมื่อ Volunteer เข้าสู่ระบบ Volunteer นั้นจะทำการหา SuperPeer เพื่อเชื่อมต่อ เมื่อได้ SuperPeer แล้วก็จะทำการสร้าง Res Ad ของเครื่องตนและส่งไปลงทะเบียนยัง SuperPeer ตัวนั้น และเมื่อ Volunteer ออกจากระบบ (เจ้าของเครื่องกลับมาใช้งาน) SuperPeer ก็จะทำการลบข้อมูล Res Ad ของ Volunteer นั้นออกไปเพื่อจะได้ไม่มีการส่งงานไปยัง Volunteer นั้นอีก อย่างไรก็ตามหากเครื่องนั้นมีผู้ใช้งานอยู่แต่ยังเชื่อมต่อกับระบบอยู่ Volunteer นั้นก็จะยังสามารถรับผลลัพธ์ของงานที่ส่งไปได้ตามปกติ เพียงแต่ไม่มีการรับงานมาทำเท่านั้น

Volunteer จะรับงานมาทำก็ต่อเมื่อเครื่องอยู่ในสภาพที่ไม่มีผู้ใช้งานเป็นเวลานานตามที่เจ้าของเครื่องกำหนด โดยเมื่อเครื่องพร้อมที่จะรับงานมาทำ Volunteer นั้นก็จะส่งข่าวสารไปบอกยัง SuperPeer ที่ตนอาศัยอยู่ และรับงานที่ส่งมาจาก SuperPeer นั้นมาประมวลผลต่อไป

4.2.4 รูปแบบของงาน

งานที่สามารถนำมาประมวลผลในระบบนี้ได้จะต้องมีคุณสมบัติดังต่อไปนี้

- เป็นงานที่อิสระต่อกัน (Independent job) ไม่มีการสื่อสารใดๆ ระหว่างงานสองงาน และไม่ใช้ข้อมูลใดๆ เพิ่มนอกจากข้อมูลที่อยู่ในอินพุตไฟล์เท่านั้น
- เป็นงานที่สามารถประมวลผลแบบ Batch ได้ เนื่องจากไม่สามารถทราบได้ว่างานนั้นจะถูกประมวลผลเมื่อใด
- ใช้หน่วยประมวลผลกลางเพียงตัวเดียวในการประมวลผลแต่ละงาน
- รับอินพุตและส่งเอาต์พุตเป็นไฟล์ และมีชื่อไฟล์ที่แน่นอน โดยชื่ออินพุตและเอาต์พุตปรากฏอยู่ใน Job Ad ของงานนั้น และไม่สามารถเปลี่ยนค่าได้ในระหว่างการทำงาน

นอกจากนี้ เนื่องจากระบบอาจทำงานอยู่บนสภาพแวดล้อมที่ความเร็วในการส่งข้อมูลต่ำดังนั้นงานจึงควรเป็นงานที่มีอัตราส่วนระหว่างเวลาที่ใช้ในการประมวลผลกับขนาดข้อมูลที่เหมาะสมตามที่กล่าวไว้ใน [26]

4.3 การทำงานของระบบ

4.3.1 รูปแบบการทำงาน

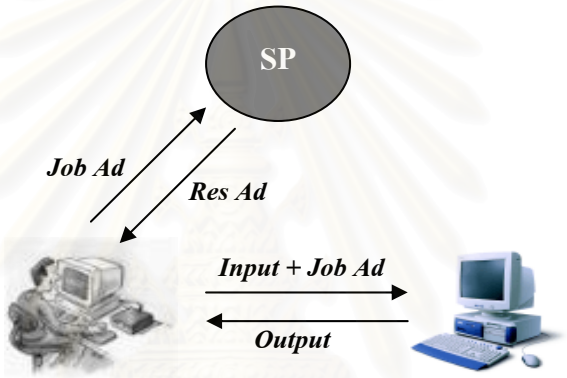
การออกแบบการทำงานของระบบได้คำนึงถึงกรณีที่ Volunteer ออกไปจากระบบก่อนที่งานที่รับไปทำจะเสร็จ ซึ่งทำให้ต้องหา Volunteer ตัวใหม่และส่งงานไปยัง Volunteer ตัวใหม่นั้น ซึ่งมีทางเลือกอยู่ 2 รูปแบบการทำงานได้แก่

รูปแบบที่ 1

ในการทำงานรูปแบบนี้เครื่องเจ้าของงานจะทำหน้าที่ ตรวจสอบสถานะงานที่ตนส่งเข้าสู่ระบบเอง โดยเครื่องเจ้าของงานจะเป็นผู้รับผิดชอบในการส่งงานเข้าสู่ระบบใหม่อีกครั้งในกรณีที่ Volunteer ที่รับงานนั้นไปทำออกจากระบบก่อนที่งานจะถูกประมวลผลเสร็จ ซึ่งขั้นตอนการทำงานเป็นดังนี้

1. เจ้าของงานส่ง Job Ad ไปยัง SuperPeer ที่ตนเชื่อมต่ออยู่
2. เมื่อ SuperPeer สามารถหา Volunteer ที่เหมาะสมกับงานได้แล้วก็จะส่ง Res Ad กลับคืนมาให้เจ้าของงาน
3. เจ้าของงานใช้ข้อมูลที่อยู่ใน Res Ad สร้างช่องสื่อสารไปยัง Volunteer เพื่อส่งงาน (อินพุต + Job Ad) ไปประมวลผล

4. หลังจากการส่งงานสำเร็จแล้ว หาก Volunteer ที่รับงานไปทำ ออกจากระบบก่อนที่งานจะเสร็จ เจ้าของงานจะต้องมีหน้าที่ส่งงานนั้นเข้าสู่ระบบใหม่อีกครั้ง หากเวลานั้นเจ้าของงานไม่อยู่ในระบบ งานก็จะไม่สามารถส่งไปประมวลผลยัง Volunteer อื่นได้ เนื่องจากไม่มีอินพุตในการส่งงาน ต้องรอจนกว่าเจ้าของงานกลับเข้ามายังระบบอีกครั้งงานนั้นถึงจะสามารถส่งเข้าสู่ระบบอีกครั้งได้
5. เมื่อ Volunteer ประมวลผลงานเสร็จแล้ว ก็จะใช้ข้อมูลที่อยู่ใน Job Ad สร้างช่องการสื่อสารเพื่อส่งผลลัพธ์กลับมายังเจ้าของงานโดยตรง หาก Volunteer ไม่สามารถติดต่อกับเจ้าของงานได้ Volunteer นั้นจะทำการส่งผลลัพธ์ไปยัง SuperPeer แทนเพื่อส่งต่อไปยังเครื่องเจ้าของงาน

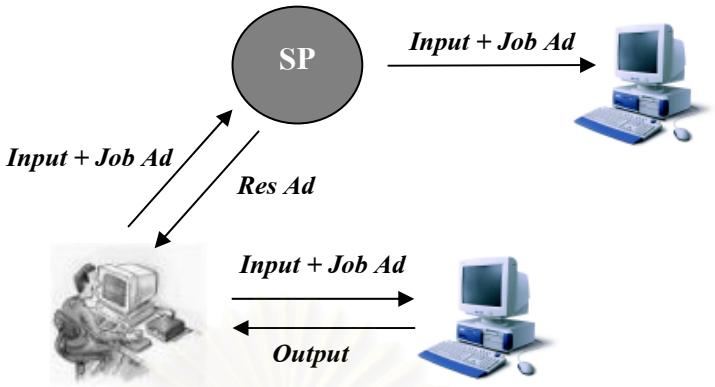


รูปที่ 4.2 การทำงานในรูปแบบการทำงานที่ 1

การทำงานรูปแบบนี้ ตัว SuperPeer จะไม่ต้องรับและส่งอินพุตไฟล์แต่อย่างใด ทำให้มีการใช้แบนด์วิดธ์ที่ตัว SuperPeer น้อย ซึ่งเหมาะกับระบบที่ตัว SuperPeer มีขนาดแบนด์วิดธ์ต่ำ อย่างไรก็ตามรูปแบบการทำงานแบบนี้หาก Volunteer ออกไปจากระบบก่อนที่งานจะถูกประมวลผลเสร็จและขณะนั้นเครื่องเจ้าของงานไม่ได้อยู่ในระบบ งานก็จะไม่สามารถถูกส่งไปประมวลผลยัง Volunteer อื่นได้ เนื่องจากอินพุตถูกเก็บไว้ที่เครื่องเจ้าของงานเท่านั้น ดังนั้นรูปแบบการทำงานนี้จึงควรนำไปใช้กับระบบที่เครือข่ายเพียร์-ทู-เพียร์มีความเสถียรสูง

รูปแบบที่ 2

รูปแบบการทำงานนี้ระบบจะรับผิดชอบงานที่ถูกส่งเข้ามายังระบบเอง โดยเครื่องเจ้าของงานไม่จำเป็นต้องตรวจดูสถานะงานเอง ในรูปแบบการทำงานนี้ SuperPeer จะทำหน้าที่ตรวจสอบสถานะงานทุกงานที่ถูกส่งเข้าสู่ระบบโดยเพียร์ที่ตนดูแลอยู่ ขั้นตอนการทำงานเป็นดังต่อไปนี้



รูปที่ 4.3 การทำงานในรูปแบบการทำงานที่ 2

1. เจ้าของงานส่ง Job Ad และอินพุตของงานมายัง SuperPeer ที่ดูแลตนอยู่
2. เมื่อ SuperPeer สามารถหา Volunteer ที่เหมาะสมให้กับงานนั้นได้แล้วก็จะส่ง Res Ad กลับมายังเจ้าของงาน
3. เจ้าของงานใช้ข้อมูลใน Res Ad สร้างช่องการสื่อสารเพื่อส่งงานไปประมวลผลยัง Volunteer
4. หลังจากการส่งงานสำเร็จแล้ว เจ้าของงานจะทำหน้าที่ตรวจสอบสถานะงานและส่งงานไปยัง Volunteer ตัวใหม่จนกว่างานจะเสร็จหรือตนออกไปจากระบบ ถ้าเจ้าของงานออกไปจากระบบก่อนที่งานจะประมวลผลเสร็จ SuperPeer จะทำหน้าที่ตรวจสอบสถานะงานต่อจากเจ้าของงาน และสามารถส่งงานไปยัง Volunteer ตัวใหม่ได้โดยใช้อินพุตและ Job Ad ที่ได้รับมาในขั้นตอนแรก
5. เมื่อ Volunteer ประมวลผลงานเสร็จแล้ว ก็จะใช้ข้อมูลที่อยู่ใน Job Ad สร้างช่องการสื่อสารเพื่อส่งผลลัพธ์กลับมายังเจ้าของงานโดยตรง หาก Volunteer ไม่สามารถติดต่อกับเจ้าของงานได้ Volunteer นั้นจะทำการส่งผลลัพธ์ไปยัง SuperPeer แทนเพื่อส่งต่อไปยังเครื่องเจ้าของงาน

การทำงานในรูปแบบนี้ แม้ว่า Volunteer จะออกไปจากระบบขณะที่งานยังประมวลผลไม่เสร็จและเครื่องเจ้าของงานไม่อยู่ในระบบ SuperPeer ก็จะสามารถส่งงานนั้นไปประมวลผลยัง Volunteer อื่นที่ว่างได้ แต่ก็ต้องเสียแบนด์วิดธ์บางส่วนเพื่อใช้ในการรับและส่งอินพุตด้วย ดังนั้นรูปแบบการทำงานนี้จึงเหมาะสมกับระบบที่ตัว SuperPeer มีขนาดแบนด์วิดธ์สูงและเจ้าของงานออกไปจากระบบบ่อยๆ อย่างไรก็ตามหากระบบมีความเสถียรของเครือข่ายเพียร์-ทู-เพียร์ต่ำปริมาณการใช้แบนด์วิดธ์ของ SuperPeer เพื่อส่งอินพุตและเอาต์พุตก็จะสูงตามไปด้วย

4.3.2 การเลือก SuperPeer

เนื่องจากเพียร์ที่ทำหน้าที่เป็น SuperPeer จะต้องรับภาระมากกว่าเพียร์อื่นๆ ทั้งการเป็นเซิร์ฟเวอร์ให้แก่กลุ่ม Volunteer และติดต่อสื่อสารกับ SuperPeer อื่นๆ ภายใน Peer group ดังนั้นเพียร์ที่จะทำหน้าที่เป็น SuperPeer จึงต้องมีคุณสมบัติดังนี้

- อยู่ในระบบเป็นเวลานาน ถ้า SuperPeer ออกจากระบบบ่อยครั้ง จะทำให้ระบบนั้นไม่เสถียร ต้องเสียเวลาในการเลือก SuperPeer ตัวใหม่ และงานในระบบอาจสูญหายได้ ซึ่งสิ่งเหล่านี้กระทบต่อประสิทธิภาพการทำงานของระบบอย่างมาก ดังนั้นเพียร์ที่จะทำหน้าที่เป็น SuperPeer จึงควรเป็นเพียร์ที่เชื่อมต่อกับระบบในแต่ละครั้งเป็นเวลานานหรือตลอดเวลา
- มีขนาดแบนด์วิดธ์ที่เหมาะสม เนื่องจาก SuperPeer จะต้องรองรับข้อมูลที่รับ-ส่งในปริมาณที่มาก ซึ่งขึ้นอยู่กับความเสถียรของเครือข่ายเพียร์-ทู-เพียร์ของระบบด้วย ดังนั้นหากระบบที่เครือข่ายมีความเสถียรต่ำ (ทำงานในรูปแบบที่ 2) เพียร์ที่จะทำหน้าที่เป็น SuperPeer จะต้องเป็นเพียร์ที่มีความเร็วในการรับ-ส่งข้อมูลที่เหมาะสม เพื่อไม่ให้เกิดปัญหาคอขวด ซึ่งจะกระทบต่อประสิทธิภาพของระบบ
- มีพลังในการประมวลผลสูง เพื่อให้สามารถรองรับปริมาณที่มากของการจับคู่, ตรวจสอบสถานะของ Volunteer, และการ Query ของ Volunteer ดังนั้น SuperPeer จึงควรจะมีพลังในการประมวลผลที่เหมาะสมเพื่อรองรับการทำงานดังกล่าวได้ทัน
- มีหน่วยความจำและเนื้อที่เก็บข้อมูลขนาดใหญ่ เนื่องจาก SuperPeer ต้องมีการเก็บฐานข้อมูลต่างๆ และงานในระบบ ซึ่งจำเป็นต้องใช้เนื้อที่ที่บริจาคให้กับระบบในการเก็บมากกว่าเพียร์ที่เป็น Volunteer

ในกรณีที่มีเพียร์ที่ทำหน้าที่เป็น SuperPeer ออกไปจากระบบ Volunteer ที่อาศัยอยู่กับ SuperPeer ที่ออกไปจะต้องทำการหา SuperPeer ตัวใหม่เพื่ออาศัย โดยเรียกตัว Discovery เพื่อหา SuperPeer ทั้งหมดที่อยู่ในระบบ จากนั้นจะส่ง Request ออกไปยัง SuperPeer แต่ละตัวเพื่อถามถึงรายละเอียดของ SuperPeer ตัวนั้น เช่น ความเร็วของหน่วยประมวลผลกลาง, ขนาดหน่วยความจำหลัก, ขนาดแบนด์วิดธ์, และจำนวน Volunteer ที่กำลังดูแลอยู่

เมื่อได้รับรายละเอียดของ SuperPeer แต่ละตัวแล้วก็จะทำการหาค่าน้ำหนัก (Weight) ของแต่ละ SuperPeer โดยค่าน้ำหนักนี้จะคิดจาก

$$\textit{ค่าน้ำหนัก} = \frac{\textit{ขนาดแบนด์วิดธ์}}{\textit{จำนวน Volunteer ที่ดูแลอยู่}} \qquad (4.1)$$

SuperPeer ที่เลือกจะเป็น SuperPeer ที่มีค่าน้ำหนักมากที่สุด หากค่าน้ำหนักเท่ากันก็จะทำการพิจารณาจากความเร็วของหน่วยประมวลผลกลางและขนาดของหน่วยความจำหลักต่อไป

4.4 การประเมินประสิทธิภาพ

จากบทที่ 3 เราได้ทราบว่าปัจจัยหนึ่งที่สำคัญต่อประสิทธิภาพการทำงานของระบบคือปริมาณแบนด์วิดธ์ของเครื่องเซิร์ฟเวอร์ ซึ่งจำกัดขนาดของไฟล์ข้อมูลในระบบดังที่ได้แสดงในบทที่ 3 สมการที่ 3.4 ดังนั้นหากต้องการให้ระบบสามารถรองรับขนาดของไฟล์ข้อมูลได้เพิ่มขึ้นจึงจำเป็นต้องเพิ่มขนาดแบนด์วิดธ์โดยรวมของเครื่องเซิร์ฟเวอร์ในระบบ

การเพิ่มขึ้นของจำนวน SuperPeer ในระบบเป็นการช่วยให้ระบบมีแบนด์วิดธ์โดยรวมของเครื่องเซิร์ฟเวอร์เพิ่มขึ้น ซึ่งช่วยให้ระบบสามารถรองรับขนาดของไฟล์ข้อมูลได้เพิ่มมากขึ้น ดังสมการที่ 4.1 และเป็นการช่วยเพิ่มความสามารถในการขยายระบบอีกด้วย [25]

$$D_{out} \leq \left(\frac{1}{1-P}\right) \bullet \left(\frac{\sum_{i=1}^{i=N_S} S_{BW_i}}{2}\right) \bullet \tau \qquad (4.2)$$

โดยที่ N_S คือ จำนวน SuperPeer ในระบบ

บทที่ 5

ระบบต้นแบบ

ในบทนี้จะกล่าวถึงโครงสร้าง ส่วนประกอบ และการทำงานของระบบต้นแบบที่ผู้วิจัยได้ทำการอิมพลีเมนต์ตามที่ได้ออกแบบไว้ในบทที่ 4 เพื่อทดสอบการทำงานของระบบที่ได้ออกแบบไว้ โดยผู้วิจัยได้อิมพลีเมนต์ระบบดังกล่าวด้วยภาษา Java และทำงานอยู่บนโพรโตคอล JXTA ซึ่งเป็นโพรโตคอลที่ใช้สำหรับแอพพลิเคชั่นที่ทำงานบนเครือข่ายแบบเพียร์-ทู-เพียร์

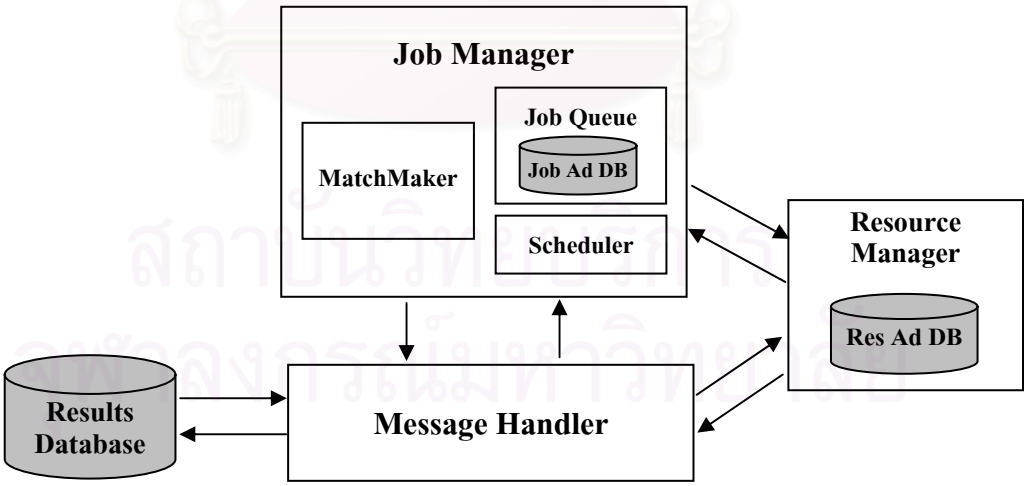
5.1 รูปแบบการทำงานของระบบต้นแบบ

เนื่องด้วยข้อจำกัดด้านเวลาในการทำวิจัย ผู้วิจัยจึงไม่สามารถสร้างระบบให้สามารถทำงานได้ทั้ง 2 รูปแบบ ดังที่ได้กล่าวไว้ในบทที่ 4 ได้ ผู้วิจัยจึงเลือกที่จะอิมพลีเมนต์ระบบโดยใช้รูปแบบการทำงานแบบที่ 1 เนื่องจากเป็นรูปแบบที่สามารถทำงานได้บนระบบที่ตัว SuperPeer มีขนาดแบนด์วิดธ์ไม่สูงมาก ซึ่งน่าจะเป็นประโยชน์แก่การนำไปใช้งานจริง

5.2 ส่วนประกอบของระบบ

5.2.1 ส่วนประกอบของ SuperPeer

SuperPeer ประกอบด้วยส่วนประกอบ 4 ส่วนหลักๆ ดังรูปที่ 5.1



รูปที่ 5.1 ส่วนประกอบของ SuperPeer

ก. ตัวจัดการงาน (Job Manager): เป็นส่วนที่ทำหน้าที่ควบคุมการไหลของงานในระบบ ประกอบด้วยส่วนต่างๆ ดังนี้

- **Matchmaker** ทำหน้าที่หาคู่ที่เหมาะสมระหว่างงานและ Volunteer ที่จะรับงานไปทำ
- **Scheduler** ทำหน้าที่จัดลำดับงานที่จะถูกส่งไปประมวลผล โดยในงานวิจัยนี้ได้เลือกวิธีการจัดลำดับงานแบบ FCFS (First-Come-First-Server) เนื่องจากเป็นวิธีที่ง่ายและเหมาะกับระบบที่มีความหลากหลายของงานมาก [29]
- **Job Queue:** ทำหน้าที่เก็บ Job Ad ของงานที่ไม่สามารถหาเครื่อง Volunteer ที่เหมาะสมได้ทันทีที่งานถูกส่งเข้าสู่ระบบ เพื่อหาเครื่อง Volunteer ที่เหมาะสมในภายหลังที่มี Volunteer ใหม่เข้ามายังระบบ

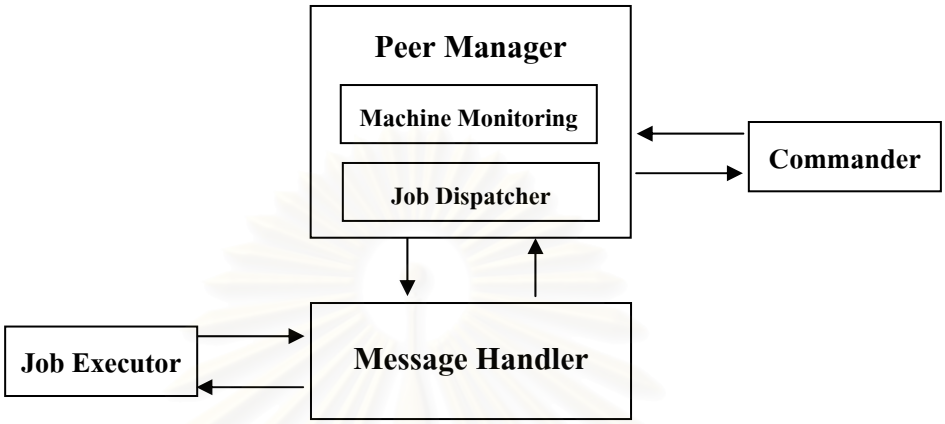
ข. ตัวจัดการทรัพยากร (Resource Management): ทำหน้าที่ดูแลและปรับปรุงฐานข้อมูลที่เก็บ Res Ad ของเครื่อง Volunteer ที่ SuperPeer ตัวนั้นดูแลอยู่ ซึ่งข้อมูล Res Ad เหล่านี้จะถูกใช้โดย Matchmaker เพื่อใช้ในการจับคู่

ค. ฐานข้อมูลเก็บผลลัพธ์ (Results Database): เก็บผลลัพธ์ของงานที่ประมวลผลเสร็จแล้วแต่เครื่อง Volunteer ที่ประมวลผลงานนั้นไม่สามารถส่งผลลัพธ์กลับไปยังเครื่องเจ้าของงานได้โดยตรง

ง. ตัวจัดการ Message (Message Handler): ทำหน้าที่รับและประมวลผล Message ที่มาจากเครื่องอื่นๆ ในระบบ เพื่อส่งต่อข้อมูลไปยังส่วนต่างๆ ต่อไป นอกจากนั้นยังทำหน้าที่ส่ง Message ไปยังเครื่องในระบบที่ต้องการอีกด้วย

5.2.2 ส่วนประกอบของ Volunteer

Volunteer ประกอบด้วย 3 ส่วนหลักๆ ดังรูปที่ 5.2



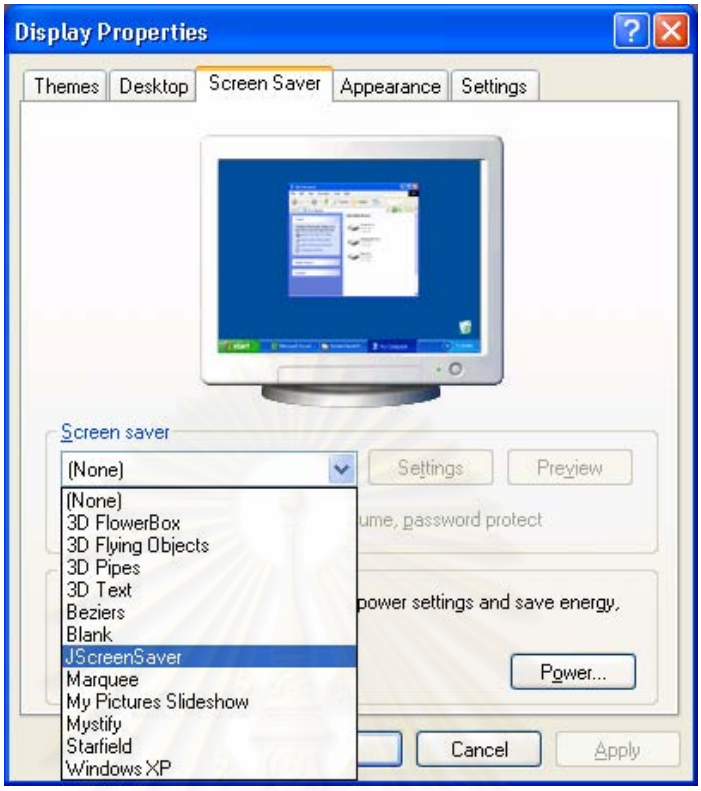
รูปที่ 5.2 ส่วนประกอบของ Volunteer

ก. ตัวจัดการเพียร์ (Peer Manager): เป็นส่วนที่ควบคุมการทำงานของเครื่อง เช่น สร้าง Res Ad, ติดต่อกับ SuperPeer, และติดต่อกับส่วนรับคำสั่งจากผู้ใช้ เป็นต้น ประกอบด้วยส่วนต่างๆ ดังนี้

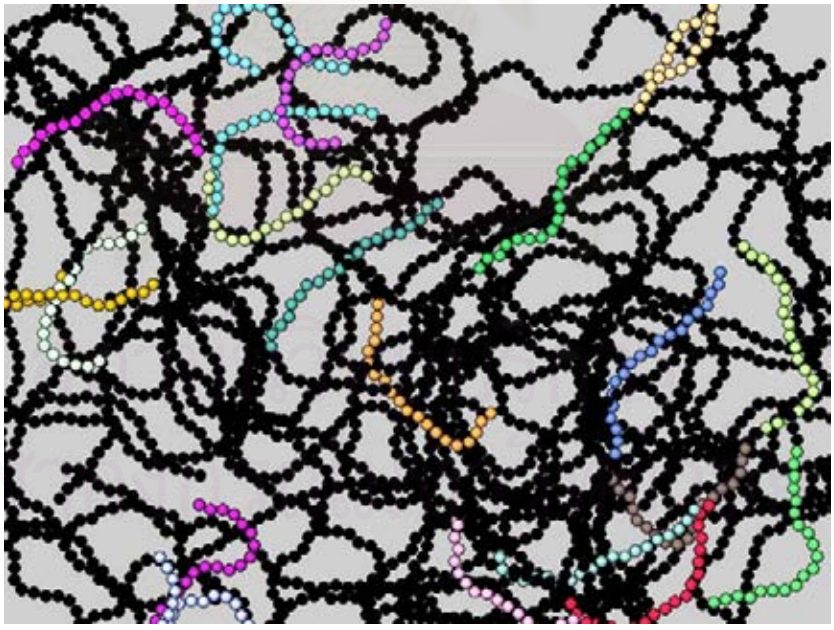
- **Machine Monitoring:** ทำหน้าที่ตรวจสอบสถานะของเครื่องว่ามีคนใช้งานเครื่องหรือไม่ โดยอยู่ในรูปของโปรแกรมถนอมหน้าจอ (Screen saver program) ที่ชื่อว่า "JScreenSaver" ดังรูปที่ 5.3 และ 5.4
- **Job Dispatcher:** ทำหน้าที่ส่งงานไปยังเครื่อง Volunteer ตามที่ SuperPeer หาให้ และรับผลลัพธ์ของงานที่ประมวลผลเสร็จแล้ว

ข. ส่วนรับคำสั่งจากผู้ใช้: ทำหน้าที่รับและประมวลผลคำสั่งของผู้ใช้ เช่นการส่งงาน หรือการตรวจสอบสถานะของงาน เป็นต้น โดยเป็นโปรแกรม Java ที่ชื่อว่า "Commander"

ค. ตัวประมวลผลงาน: ทำหน้าที่ประมวลผลงานที่รับมาทำ และส่งงานกลับไปยังเครื่องเจ้าของงาน



รูปที่ 5.3 การเลือกโปรแกรมถนอมหน้าจอ JScreenSaver

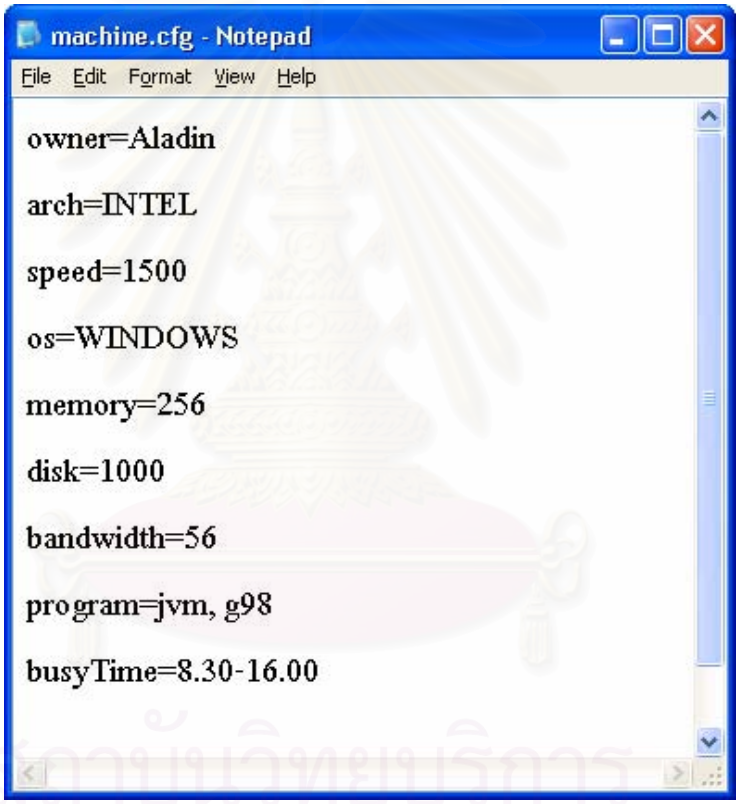


รูปที่ 5.4 โปรแกรมถนอมหน้าจอ JScreenSaver

5.2.3 Resource advertisement

ข้อมูลที่อยู่ใน Res Ad จะบอกถึงข้อมูลทางด้านฮาร์ดแวร์, ซอฟต์แวร์, และเครือข่ายของเครื่อง Volunteer นั้น รวมถึงนโยบายและข้อกำหนดในการใช้งานเครื่อง โดย Res Ad นี้จะถูกส่งไปทำการลงทะเบียนยัง SuperPeer และ SuperPeer จะเก็บ Res Ad นั้นไว้จนกว่า Volunteer ของ Res Ad นั้นออกไปจากระบบ

Res Ad ถูกสร้างขึ้นมาจากไฟล์ตั้งค่าระบบ (Config file) ที่มีชื่อว่า “machine.cfg” ที่ผู้ใช้ได้สร้างไว้ โดยข้อมูลภายในไฟล์ตั้งค่าระบบจะอยู่ในรูปของคู่ของตัวแปรและค่าที่ต้องการกำหนดดังรูปที่ 5.5



รูปที่ 5.5 ตัวอย่างไฟล์ตั้งค่าระบบ

เมื่อเครื่องเข้าสู่ระบบ โปรแกรมจะอ่านข้อมูลในไฟล์ตั้งค่าระบบนี้เพื่อสร้าง Res Ad ของ Volunteer นั้น โดยข้อมูลที่สำคัญใน Res Ad ได้แสดงในตารางที่ 5.1

ตารางที่ 5.1 ตัวอย่างข้อมูลที่อยู่ใน Resource advertisement

ชื่อหน่วยข้อมูล	รายละเอียด
ID	เลขประจำตัวของแต่ละเพียร์ในระบบ โดยใช้เลข PeerID ของโพรโตคอล JXTA
Name	ชื่อเครื่อง
Owner	ชื่อเจ้าของเครื่อง
Architecture	สถาปัตยกรรมของเครื่อง (ชนิดของหน่วยประมวลผลกลาง) “INTEL” : Intel x86 (Pentium, Xeon, etc) “ALPHA” : Digital Alpha “SGI” : Silicon Graphics MIPS “SUN” : Sun UltraSparc “HP” : Hewlett Packard PA-RISC “OTHER” : อื่นๆ
Speed	ความเร็วของหน่วยประมวลผลกลาง หน่วยเป็น MHz
Os	ระบบปฏิบัติการของเครื่อง “WINDOWS” : Microsoft Windows family “HPUX” : HPUX “LINUX” : Linux kernel systems “SOLARIS” : Sun Solaris “IRIX” : IRIX systems “OTHER” : อื่นๆ
Memory	ขนาดหน่วยความจำหลักของเครื่อง หน่วยเป็น Mbytes
Disk	ขนาดเนื้อที่เก็บข้อมูลที่บริจาคให้แก่ระบบ หน่วยเป็น Mbytes
Bandwidth	ขนาดแบนด์วิธของเครื่อง หน่วยเป็น Kbit/sec
Program	รายชื่อโปรแกรมที่สามารถให้งานมาเรียกใช้ได้ หากมีมากกว่า 1 โปรแกรมจะคั่นด้วย “,” เช่น “jvm”, “dos”
RestrictedTime	ช่วงเวลาที่จะไม่รับงานมาทำแม้ว่าเครื่องจะว่างก็ตาม เช่น “08.00-16.00” หากไม่มีกำหนดจะหมายถึงสามารถรับงานได้ตลอดเวลาที่เครื่องว่าง

5.2.4 Job advertisement

ข้อมูลที่อยู่ใน Job Ad จะบอกถึงรายละเอียดและความต้องการของงาน โดยจะถูกส่งไปยัง SuperPeer เพื่อให้ตัว Matchmaker ของ SuperPeer สามารถหา Res Ad ของ Volunteer ที่เหมาะสมกับงานนั้น

Job Ad ถูกสร้างจากไฟล์แสดงรายละเอียดของงาน (Description file) ที่ชื่อ "*ชื่องาน*.jdf" ที่ผู้ใช้สร้างขึ้นพร้อมกับงาน โดยข้อมูลที่อยู่ในไฟล์แสดงรายละเอียดของงานจะอยู่ในรูปของคู่ตัวแปรและค่าที่ต้องการกำหนดเช่นเดียวกับไฟล์ตั้งค่าระบบ ดังรูปที่ 5.6

รูปที่ 5.6 ตัวอย่างไฟล์แสดงรายละเอียดของงาน

หลังจากที่เจ้าของงานส่งงานเข้าสู่ระบบแล้ว โปรแกรมจะอ่านข้อมูลในไฟล์แสดงรายละเอียดของงานเพื่อสร้าง Job Ad ของงานนั้น โดยข้อมูลที่สำคัญใน Job Ad ได้แสดงในตารางที่ 5.2

ตารางที่ 5.2 ตัวอย่างข้อมูลที่อยู่ใน Job advertisement

ชื่อหน่วยข้อมูล	รายละเอียด
ID	เลขประจำตัวของแต่ละงาน สร้างโดย MD5 จำนวน 128 บิต
Name	ชื่องาน
Owner	ชื่อเจ้าของงาน
OwnerID	ID ของเครื่องเจ้าของงาน
Cmd	คำสั่งเพื่อเริ่มการทำงาน
Input	ชื่ออินพุตไฟล์
Output	ชื่อเอาต์พุตไฟล์
Architecture	สถาปัตยกรรมของเครื่องที่ต้องการ (ใช้ค่าเดียวกับใน Res Ad)
Speed	ความเร็วหน่วยประมวลผลกลางขั้นต่ำที่ต้องการ หน่วยเป็น MHz
Os	ระบบปฏิบัติการที่ต้องการ (ใช้ค่าเดียวกับใน Res Ad)
Memory	ขนาดหน่วยความจำหลักขั้นต่ำที่สามารถประมวลผลงานได้ หน่วยเป็น Mbytes
Disk	เนื้อที่เก็บข้อมูลที่งานใช้ หน่วยเป็น Mbytes
Program	โปรแกรมที่งานจะเรียกใช้ระหว่างประมวลผล
Length	เวลาในการประมวลผลโดยประมาณ หน่วยเป็นนาที

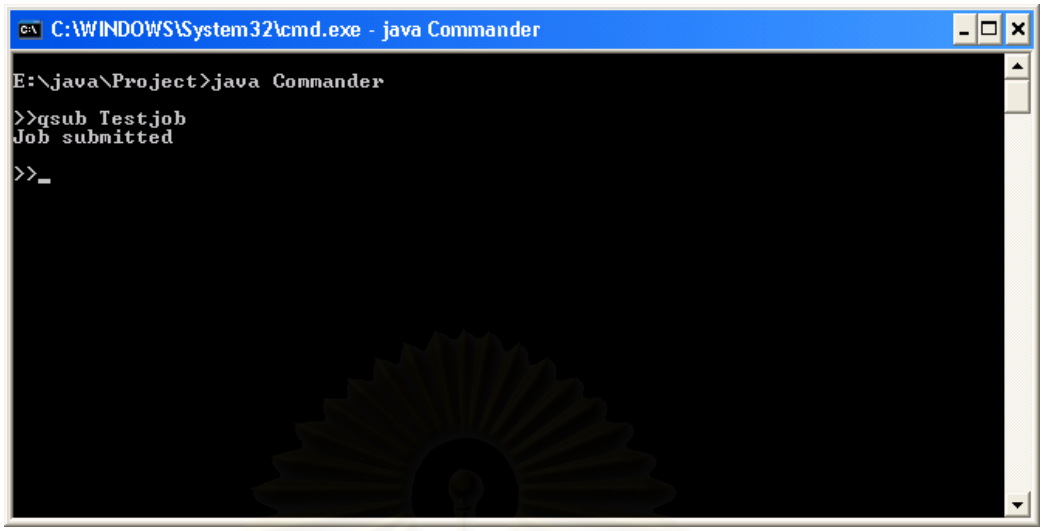
หน่วยข้อมูล Architecture, Speed, Os, Memory, Disk, และ Program หากหน่วยข้อมูลใดไม่มีการกำหนดค่า หน่วยข้อมูลนั้นก็จะไม่ถูกนำมาใช้ในการเลือก Volunteer

5.3 การส่งงานเข้าสู่ระบบ

5.3.1 วิธีการส่งงาน

ส่วนประกอบที่ใช้ในการส่งงานได้แก่ ไฟล์แสดงรายละเอียดของงาน และอินพุตไฟล์ โดยอินพุตไฟล์จะอยู่ในรูปของไฟล์บีบอัด (Zip file) ที่บรรจุอินพุตไฟล์ทั้งหมดของงานนั้นไว้ โดยมีชื่อว่า “*ชื่องาน*.zip” ทั้งไฟล์แสดงรายละเอียดของงานและอินพุตไฟล์ที่อยู่ในรูปของไฟล์บีบอัดจะต้องถูกใส่ลงในโฟลเดอร์ที่มีชื่อเดียวกับชื่องานที่อยู่ในไดเรกทอรีเดียวกับโปรแกรม

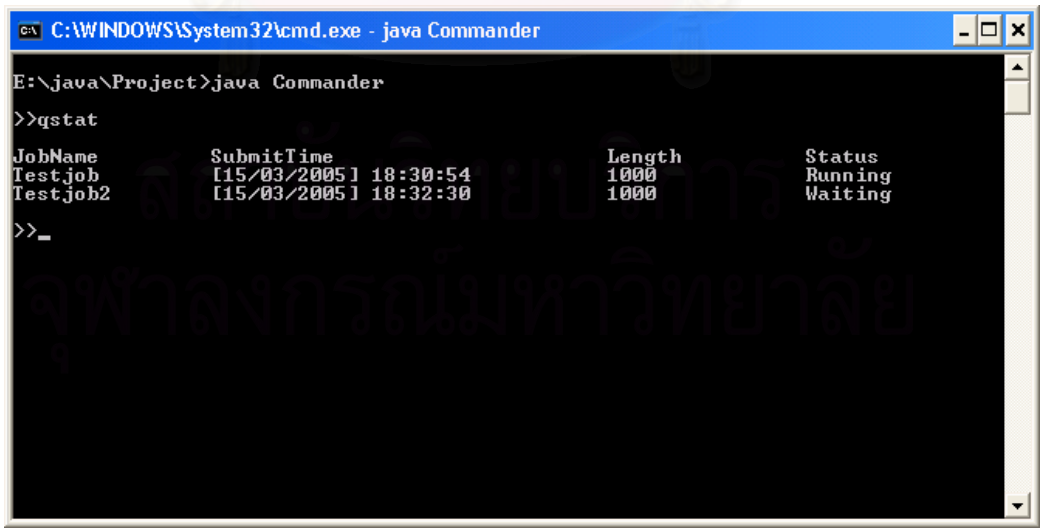
จากนั้นทำการเปิดโปรแกรมส่วนรับคำสั่งจากผู้ใช้ที่มีชื่อว่า “Commander” และพิมพ์คำสั่ง “qsub” ตามด้วยชื่องาน ดังรูปที่ 5.7



รูปที่ 5.7 โปรแกรมส่วนรับคำสั่งและคำสั่งเพื่อส่งงานเข้าสู่ระบบ

5.3.2 วิธีดูสถานะของงาน

ผู้ใช้สามารถตรวจสอบสถานะของงานที่ส่งเข้าสู่ระบบได้โดยใช้คำสั่ง “qstat” ซึ่งระบบจะทำการแสดงรายชื่อและสถานะของงานทั้งหมดที่ยังประมวลผลไม่เสร็จของ Volunteer นั้น ดังรูปที่ 5.8 โดยสถานะของงานนั้นหากเป็น “Running” นั่นหมายความว่างานนั้นได้ถูกส่งไปประมวลผลยังเครื่องใดเครื่องหนึ่งในระบบแล้ว แต่หากสถานะเป็น “Waiting” หมายความว่าระบบขณะนี้ไม่มี Volunteer ใดที่สามารถรับงานนั้นไปประมวลผลได้ งานจึงต้องรอจนกว่าจะมี Volunteer ที่สามารถรับงานนี้ไปประมวลผลได้เข้ามายังระบบ



รูปที่ 5.8 การดูสถานะของงาน

5.3.3 ผลลัพธ์ของงาน

เมื่องานถูกประมวลผลเสร็จแล้วผลลัพธ์ของงานดังกล่าวก็จะถูกส่งกลับมายังเครื่องเจ้าของงาน โดยประกอบด้วยไฟล์บีบอัดที่ชื่อว่า “ResultOf_*ชื่องาน*.zip” ที่บรรจุผลลัพธ์ทั้งหมดของงานนั้น และไฟล์แสดงรายละเอียดของการประมวลผล (log.txt) ที่มีข้อมูลได้แก่ ชื่องาน, เวลาที่ส่งเข้าสู่ระบบ, เวลาที่งานถูกส่งไปยัง Volunteer, เวลาที่ประมวลผลเสร็จ, เวลาที่เครื่องเจ้าของงานรับผลลัพธ์, และเวลาที่ใช้ในการประมวลผล

หลังจากไม่ต้องการป้อนคำสั่งใดๆ แล้วผู้ใช้สามารถปิดตัวรับคำสั่งได้โดยการพิมพ์คำสั่ง “exit” การพิมพ์คำสั่งนี้เป็นเพียงการปิดตัวรับคำสั่งเท่านั้น ไม่ใช่การออกจากระบบแต่อย่างใด เครื่องยังคงอยู่ในระบบต่อไปจนกว่าเครื่องจะปิดการเชื่อมต่อกับระบบหรือปิดเครื่อง

บทที่ 6

สรุปผลการวิจัย

6.1 สรุปผลการวิจัย

งานวิจัยนี้ได้นำเสนอโครงสร้างของระบบการคำนวณแบบกระจายแบบ Cycle Scavenger บนเครือข่ายอินเตอร์เน็ต โดยนำรูปแบบการสื่อสารแบบเพียร์-ทู-เพียร์และโมเดลแบบซุปเปอร์เพียร์มาประยุกต์ใช้เพื่อเพิ่มประสิทธิภาพการทำงานของระบบ

จากผลการทดลองในบทที่ 3 แสดงให้เห็นว่าการนำรูปแบบการสื่อสารแบบเพียร์-ทู-เพียร์มาประยุกต์ใช้กับระบบการคำนวณแบบกระจายจะช่วยให้ระบบสามารถรองรับขนาดของข้อมูลงานได้เพิ่มมากขึ้น และระบบยังสามารถรองรับจำนวนสมาชิกในระบบได้เพิ่มมากขึ้นโดยไม่ทำให้ประสิทธิภาพการทำงานของระบบลดลง เนื่องจากได้มีการนำเอาแบนด์วิดธ์ของเครื่องสมาชิกมาใช้ในการส่งข้อมูลโดยตรงโดยไม่ผ่านเครื่องเซิร์ฟเวอร์ ทำให้ความต้องการขนาดแบนด์วิดธ์ที่เครื่องเซิร์ฟเวอร์ลดลงไปด้วยซึ่งส่งผลให้ค่าใช้จ่ายของระบบลดลง ซึ่งประสิทธิภาพของระบบนั้นจะขึ้นอยู่กับค่าความเสถียรของเครือข่ายเพียร์-ทู-เพียร์ของระบบด้วย โดยยิ่งระบบมีความเสถียรมากระบบก็จะยิ่งสามารถใช้การสื่อสารแบบเพียร์-ทู-เพียร์ได้มากขึ้นและช่วยให้ระบบมีประสิทธิภาพมากขึ้นตามไปด้วย

อย่างไรก็ตาม การใช้รูปแบบการสื่อสารแบบเพียร์-ทู-เพียร์เพียงอย่างเดียวก็ยังมีข้อจำกัดในด้านของความสามารถในการขยายระบบ และขนาดข้อมูลที่ระบบสามารถรองรับได้ โดยหากระบบมีขนาดข้อมูลใหญ่เกินกว่าที่ระบบสามารถรองรับได้ และระบบมีความเสถียรของเครือข่ายต่ำ ปัญหาคอขวดที่เครื่องเซิร์ฟเวอร์ก็จะยังคงเกิดขึ้น นอกจากนั้นหากเครื่องเซิร์ฟเวอร์ที่มีอยู่เพียงเครื่องเดียวไม่สามารถให้บริการได้ ระบบก็จะไม่สามารถทำงานต่อไปได้ด้วย

เพื่อเพิ่มความสามารถในการขยายระบบ และขนาดข้อมูลที่ระบบสามารถรองรับได้ รวมถึงแก้ปัญหา Single point of failure โมเดลแบบซุปเปอร์เพียร์จึงถูกนำมาใช้เป็นโครงสร้างของระบบ โดยจำนวนซุปเปอร์เพียร์จะเพิ่มขึ้นหากระบบตรวจพบว่าภาระงานที่ซุปเปอร์เพียร์แต่ละตัวแบกรับมากเกินกว่าความสามารถของเครื่อง ซึ่งการเพิ่มขึ้นของซุปเปอร์เพียร์เป็นการช่วยให้ระบบมีปริมาณแบนด์วิดธ์โดยรวมของเครื่องเซิร์ฟเวอร์เพิ่มมากขึ้น

ผู้วิจัยได้สร้างระบบต้นแบบของโครงสร้างและรูปแบบการทำงานที่ได้ออกแบบไว้ด้วยภาษา Java โดยระบบจะทำงานอยู่บนโพรโตคอล JXTA ซึ่งเป็นโพรโตคอลสำหรับแอพพลิเคชั่นบนเครือข่ายเพียร์-ทู-เพียร์ เพื่อเป็นโครงสร้างต้นแบบสำหรับการนำไปสร้างเป็นระบบที่สามารถทำงานได้จริงต่อไป

6.2 ข้อเสนอแนะและแนวทางในการวิจัยต่อ

เนื่องจากข้อจำกัดของโปรแกรมแบบจำลองที่ไม่สามารถรองรับการจำลองระบบที่มีจำนวนเครื่องคอมพิวเตอร์มากๆ ได้ และไม่สามารถจำลองสภาพการทำงานที่มีจำนวนซุปเปอร์เพียร์มากกว่าหนึ่งตัวได้ ด้วยเหตุนี้ทำให้ไม่สามารถจำลองการทำงานของระบบที่มีขนาดใหญ่มากๆ และมีจำนวนซุปเปอร์เพียร์มากกว่าหนึ่งตัวได้ จึงควรหาโปรแกรมแบบจำลองตัวใหม่ที่สามารถจำลองระบบที่ต้องการได้เพื่อศึกษาถึงประสิทธิภาพการทำงานของระบบขนาดใหญ่

เนื่องด้วยข้อจำกัดของเวลาในการทำวิจัย ผู้วิจัยจึงไม่สามารถสร้างระบบที่สามารถทำงานได้อย่างเต็มรูปแบบและสามารถรองรับการทำงานได้ทุกกรณี รวมถึงขาดการศึกษาประสิทธิภาพของระบบโดยใช้ระบบจริง นอกจากนั้นอัลกอริทึมในการจัดลำดับงานและหาคู่เป็นอัลกอริทึมที่ง่ายแก่การอิมพลีเมนต์ แนวทางที่จะวิจัยต่อไปจึงควรเป็นการศึกษาถึงอัลกอริทึมอื่นๆ ที่ใช้ในการจัดลำดับงานและหาคู่ ว่ามีผลกระทบอย่างไรต่อประสิทธิภาพการทำงานของระบบ และหาอัลกอริทึมที่เหมาะสมที่สุดสำหรับระบบที่ได้ออกแบบไว้ในงานวิจัยนี้

รายการอ้างอิง

[1] Buyya, R. High performance cluster computing. Volume1. Prentice Hall PTR, 1999.

[2] Sterling, T., Savarese, D., Becker, D. J., Dorband, J. E., Ranawake, U. A., and Packer, C. V. Beowulf: A Parallel Workstation for Scientific Computation. Proceedings of the 24th International Conference on Parallel Processing (August 1995): 11-14.

[3] Anderson, T. E., Culler, D. E., Petterson D. A., and the NOW team. A Case for NOW (Networks of Workstations). IEEE Micro (February 1995): 54-64.

[4] Litzkow, M. J., Livny, M., and Mutka, M. W. Condor – A Hunter of Idle Workstations. Proceedings of the 8th International Conference on Distributed Computing Systems (1988): 104-111.

[5] Basney, J., Livny, M., and Tannenbaum, T. High Throughput Computing with Condor. HPCU news Volume 1(2) (June 1997).

[6] Anderson, D. P., Cobb, J., Korpela, E., Lebofsky, M., and Werthimer, D. SETI@Home: An Experiment in Public-Resource Computing. Proceedings of Communication of the ACM Volume 45 (November 2002): 56-61.

[7] SETI@home. Search for Extraterrestrial Intelligence at home [Online].Available from: http://setiathome.ssl.berkeley.edu/ [2004, June 15]

[8] FightAIDS@Home. Available from: http://fightaidsathome.scripps.edu/ [2004, June 15]

[9] Larson, S. M., Snow, C. D., Shirts, M., and Pande, V. S. Folding@Home and Genome@Home: Using distributed computing to tackle previously intractable problems in computational biology. Computational Genomics (2002).

[10] Distributed.net. Available from: http://www.distributed.net/ [2004, June 15]

[11] Oram, A. Peer-to-Peer: Harnessing the power of disruptive technologies. O'Reilly and Associates, 2001.

[12] Milojicic, D. S., Kalogeraki, V., Lukose, R., Nagaraja, K., Pruyne, J., Richard, B., et al. Peer-to-Peer computing. HP Internal Report, March 2002.

[13] Parabon Computation. Available from: http://www.parabon.com/ [2004, June 20]

[14] Christiansen, B. O., Cappello, P., Ionescu, M. F., Neary, M. O., Schauser, K. E., and Wu, D. Javelin: Internet-Based Parallel Computing Using Java, Proceedings of Concurrency: Practice and Experience Volume9(11) (June 1997):1139-1160

[15] Verbeke, J., Nadgir, N., Ruetsch, G., and Shrapov, I. Framework for Peer-to-Peer Distributed Computing in a Heterogeneous, Decentralized Environment, Proceedings of Grid Computing (GRID 2002) (November 2002):1-12.

[16] SETI@home: Search for Extraterrestrial Intelligence at home. Available from: http://setiathome.ssl.berkeley.edu/bw.html [2004, June 5]

[17] Java applets. Available from: http://java.sun.com/applets/ [2004, July 25]

[18] Gong, L., Mueller, M., Prafullchandra, H., and Schemers, R. Going Beyond the Sandbox: An Overview of the New Security Architecture in the Java Development Kit 1.2, Proceedings of the USENIX Symposium on Internet Technologies and Systems (December 1997):102-112.

[19] Adar, E., and Huberman, B. Free riding on Gnutella. Xerox PARC Technical Report, 2000.

[20] Napster. Available from: http://www.napster.com/ [2004, September 1]

[21] Good, N. S., and Krekelberg, A. Usability and privacy: a study of Kazaa P2P file-sharing. Proceedings of the conference on Human factors in computing systems (2003): 137-144.

[22] Clarke, I., Sandberg, O., Wiley, B., and Tong, T. W. Freenet: A Distributed Anonymous Information Storage and Retrieval System. Lecture Notes in Computer Science, 2002

[23] Gradecki, J.D. Mastering JXTA: Building java peer-to-peer applications. Wiley Publishing, 2002.

[24] JXTA.org. Available from: http://www.jxta.org/ [2003, September 12]

[25] Page, A., Keane, T., Allen, R., Naughton, T. J., and Waldron, J. Multi-tiered distributed computing platform, 2nd International Conference on the Principles and Practice of Programming in Java (2003): 191-194.

[26] Gray, J. Distributed Computing Economics. Microsoft Research Lab Technical Report, 2003.

[27] Buyya, R. and Murshed, M. GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. The Journal of Concurrency and Computation: Practice and Experience (CCPE) Volume14(13-15) (2002): 1-32.

[28] SPEC Banchmark. Available from: http://www.specbench.org/cpu2000/ [2004, May 15]

[29] James, H.A., Hawick, K.A., and Coddington, P.D. Scheduling Independent Tasks on Metacomputing Systems. Proceedings of Parallel and Distributed Computing Systems (PDCS'99) (August 1999): 156-162.

[30] Yang, B., Molina, H. G. Designing a Super-Peer Network. Proceedings of International Conference on Data Engineering (ICDE'03) (March 2003):49-61.

[31] Raman, R., Livny, M., and Solomon, M. Matchmaking: Distributed Resource Management for High Throughput Computing. Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing (July 1998):28-31

ภาคผนวก

ภาคผนวก ก.

โพรโตคอล JXTA

โพรโตคอล JXTA ประกอบด้วยโพรโตคอลย่อย 6 โพรโตคอล คือ

1. **Peer Resolver Protocol (PRP)**: ใช้ในการรับ-ส่ง Query ระหว่างเพียร์ โดย Query สามารถถูกส่งโดยตรงไปยังทุกๆ เพียร์ใน Peer group หรือเจาะจงส่งให้เพียร์ใดเพียร์หนึ่งก็ได้
2. **Peer Discovery Protocol (PDP)**: ใช้ในการประกาศ (Publish) ทรัพยากรของตน (เช่น เพียร์, Peer group, ไปป์, และบริการ เป็นต้น) และค้นหาทรัพยากรที่ต้องการจากเพียร์อื่น ซึ่งทรัพยากรดังกล่าวจะอยู่ในรูปแบบของ Advertisement
3. **Peer Information Protocol (PIP)**: ใช้ตรวจสอบสถานะ และข้อมูลต่างๆ ของเพียร์ที่ต้องการ เช่น Uptime, สถานะ, ปริมาณข้อมูลที่เข้า-ออก เป็นต้น
4. **Pipe Binding Protocol (PBP)**: ใช้สร้างช่องทางสื่อสารเสมือนระหว่างเพียร์ โดยไปป์จะเชื่อมระหว่าง End point ของทั้งสองเพียร์เข้าด้วยกัน
5. **Endpoint Routing Protocol (ERP)**: ใช้ในการหาเส้นทางจากเพียร์หนึ่งไปยังอีกเพียร์หนึ่ง
6. **Rendezvous Protocol (RVP)**: ใช้กระจาย Messageไปยังเพียร์ต่างๆ ใน Peer group

โครงสร้างของ Project JXTA

โครงสร้างของโพรโตคอล JXTA แบ่งเป็น 3 ระดับชั้นด้วยกัน ดังรูปที่ 2.11 คือ

- **Platform Layer (JXTA Core)**

  เป็นส่วนที่ประกอบด้วยฟังก์ชันต่างๆ ของ เพียร์, Peer group, ความปลอดภัย, และตรวจสอบติดตาม (Monitoring) รวมถึงการรับ-ส่ง Message ต่างๆ ด้วย ตัวอย่างฟังก์ชันที่สำคัญ เช่น การสร้างเพียร์ และ Peer group, การกำหนดนโยบาย (policy), การจัดการกับ Firewall เป็นต้น

- **Service Layer**

  เป็นบริการพื้นฐานต่างๆที่จำเป็นสำหรับแอปพลิเคชั่นแบบเพียร์-ทู-เพียร์ เช่น การค้นหาและทำดัชนี (Searching and indexing) ข้อมูล, ระบบเก็บข้อมูล (Storage system), การแชร์ไฟล์ (File sharing), ระบบไฟล์แบบกระจาย (Distributed file system), การรวบรวมทรัพยากรในระบบ (Resource aggregation), การยืนยันตัว (Authentication), และ บริการ PKI (Public Key Infrastructure)

- **Application Layer**

  เป็นส่วนของแอปพลิเคชั่นชันที่ทำงานอยู่บนโพรโตคอล JXTA เช่น P2P instant messaging, การแบ่งปันข้อมูลและทรัพยากร, การจัดการและขนส่งข้อมูลความบันเทิงต่างๆ, ระบบเมล์แบบเพียร์-ทู-เพียร์, ระบบการประมูลแบบกระจาย, และอื่นๆ เป็นต้น

JXTA terminology

ในส่วนนี้จะกล่าวถึงศัพท์เฉพาะ และองค์ประกอบต่างๆ ใน โพรโตคอล JXTA

- **IDs**

  ทรัพยากรต่างๆ ใน โพรโตคอล JXTA จะอ้างอิงด้วยรหัสประจำตัว ซึ่งทำหน้าที่เหมือนกับรหัสประจำตัวประชาชนนั่นเอง ซึ่งทรัพยากรที่จะมีเลขประจำตัว คือ เพียร์, Peer group, ไปป์ (Pipes), คอนเท็นต์ (Content), โมดูลคลาส (Module classes), และรายละเอียดของโมดูล (Module specifications) รหัสประจำตัวใน JXTA ที่อิมพลีเมนต์ด้วย Java จะประกอบด้วยเลขฐาน 16 จำนวน 128 บิต ตัวอย่างของ ID เช่น

  urn:jxta:uuid-59616261646162614A787461503250 33F3BC76FF13C2414CBC0AB663666DA53903

- **Peers**

  เพียร์ คืออุปกรณ์ต่างๆที่เชื่อมต่อกับเครือข่าย ซึ่งอาจจะเป็น เครื่อง PC, เครื่องเซิร์ฟเวอร์, อุปกรณ์เซ็นเซอร์, โทรศัพท์มือถือ หรือ PDA เป็นต้น โดยแต่ละเพียร์จะทำงานเป็นอิสระต่อกัน และถูกอ้างอิงด้วย Peer ID แต่ละเพียร์จะมีอินเตอร์เฟสที่เรียกว่า “Peer Endpoint” สำหรับใช้ในการสร้างการเชื่อมต่อแบบจุดต่อจุด (Point-to-point) กับเพียร์อื่นๆ

  ในโพรโตคอล JXTA มีเพียร์ทั้งหมด 4 ชนิดด้วยกัน คือ

  1. *Minimal edge peer*: มักจะเป็นเพียร์ที่มีทรัพยากรน้อย เช่น โทรศัพท์มือถือ หรือ PDA เป็นเพียร์ที่จะไม่มีการเก็บแคชของ Advertisement ไว้ และไม่ได้ทำหน้าที่หาเส้นทางของ Message ให้แก่เพียร์อื่น ทำได้แค่รับและส่ง Message เท่านั้น
  2. *Full-featured edge peer*: จะมีการเก็บแคชของ Advertisement ไว้ด้วย ทำให้สามารถตอบคำถามกับเพียร์อื่นๆ ได้ แต่จะตอบได้เฉพาะข้อมูลที่อยู่ใน Advertisement ที่ตนแคชไว้เท่านั้นและจะไม่ส่งต่อคำถามต่อไปยังเพียร์อื่น โดยทั่วไปเพียร์ใน JXTA ส่วนใหญ่ จะเป็นเพียร์ประเภทนี้

3. *Rendezvous peer*: จะเหมือนกับเพียร์อื่นๆ แต่จะมีการส่งต่อคำถามไปยังเพียร์อื่นๆ เพื่อช่วยในการหาคำตอบด้วย เมื่อเพียร์ใดเข้ามายัง Peer group เพียร์นั้นจะทำการหา Rendezvous peer แต่ถ้าไม่พบเพียร์นั้นจะกลายเป็น Rendezvous peer ให้กับ Peer group นั้นทันที Rendezvous peer แต่ละตัวจะเก็บรายชื่อของเพียร์ที่ใช้ Rendezvous peer นั้นไว้ รวมถึง Rendezvous peer ที่รู้จักอื่นๆ ด้วย
4. *Relay peer*: เป็นเพียร์ที่ทำหน้าที่เก็บข้อมูลเกี่ยวกับเส้นทาง และหาเส้นทางให้แก่ Message เพื่อให้ Message สามารถเดินทางไปถึงที่หมายได้

- **Peer groups**

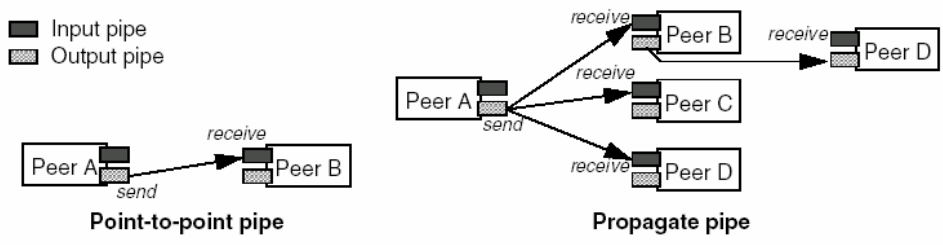
Peer group ประกอบด้วยหลายๆ เพียร์ที่รวมกันเพื่อให้บริการหรือทำงานต่างๆ ร่วมกัน แต่ละ Peer group จะอ้างอิงด้วย Peer Group ID โดย Peer group สามารถที่จะสร้างนโยบายต่างๆ เพื่อกำหนดข้อบังคับและความปลอดภัยแก่ Peer group ของตน เพียร์หนึ่งๆ สามารถเป็นสมาชิกได้หลาย Peer group พร้อมๆ กัน โดยทุกเพียร์ใน โพรโตคอล JXTA จะอยู่ภายใต้ NetPeerGroup

- **Pipes**

แต่ละเพียร์จะใช้ไปป์ในการส่ง Message ไปยังอีกเพียร์หนึ่ง ไปป์ที่อยู่ฝั่งผู้รับจะเรียกว่า อินพุตไปป์ (Input pipe) และฝั่งผู้ส่งจะเรียกว่า เอาต์พุตไปป์ (Output pipe) ซึ่งจะใช้ Peer Endpoint Interface ในการสร้างช่องสื่อสารถึงแม้ว่าทั้งสองเพียร์นั้นจะไม่มีการเชื่อมต่อถึงกันทางกายภาพก็ตาม ในโพรโตคอล JXTA มีไปป์ 3 แบบด้วยกัน

1. *Point-to-point pipes*: เป็นไปป์ที่เชื่อมต่อระหว่างเอาต์พุตไปป์ของเพียร์หนึ่งไปยังอินพุตไปป์ของอีกเพียร์หนึ่งเท่านั้น
2. *Propagate pipes*: เป็นไปป์ที่เชื่อมต่อจากเอาต์พุตไปป์ของเพียร์หนึ่งไปยังหลายๆ อินพุตไปป์ การกระจาย Message จะทำภายใต้กลุ่มเพียร์ใดกลุ่มหนึ่งเท่านั้น นั่นหมายความว่า Message ที่กระจายจากกลุ่มเพียร์หนึ่งจะไม่กระจาออกไปยังกลุ่มเพียร์อื่นๆ
3. *Secure unicast pipes*: จะเหมือนกับไปป์แบบ Point-to-point แต่จะมีการกำหนดความปลอดภัยให้แก่ช่องสื่อสารนั้น

ใน JXTA ที่อิมพลีเมนต์ด้วย Java จะมีไปป์เพิ่มขึ้นมาอีก 2 แบบ คือ ไปป์แบบสองทิศทาง (Bidirectional pipes) และไปป์แบบสองทิศทางที่ไว้ใจได้ (Bidirectional/reliable pipes) แต่ทั้ง 2 แบบนี้ก็สร้างอยู่บนพื้นฐานของไปป์ ทั้ง 3 แบบที่กล่าวมาข้างต้น



รูปที่ ก.1 Point-to-point and propagate pipes

- **Messages**

  Message เป็นออบเจ็กต์ที่ถูกส่งระหว่าง 2 เพียร์ ซึ่งภายในออบเจ็กต์นั้นจะบรรจุข้อมูลต่างๆ ที่ต้องการส่ง ในโพรโตคอล JXTA ได้ระบุ Message ไว้ 2 ชนิดคือ แบบ XML และ แบบ Binary แต่ใน JXTA ที่อิมพลีเมนต์ด้วย Java จะมีเพียง แบบ Binary เท่านั้น

- **Advertisement**

  ทรัพยากรทั้งหมดของโพรโตคอล JXTA (เช่น เพียร์, Peer group, ไปป์, และ บริการ) จะถูกนำเสนอด้วย Advertisement ซึ่งเป็นเอกสาร XML ที่บรรจุข้อมูลต่างๆ ของทรัพยากรนั้นไว้และประกาศออกไปให้เพียร์อื่นๆ รับรู้ถึงทรัพยากรนั้น และสามารถค้นหาทรัพยากรเหล่านั้นได้ ดังนั้น Advertisement นี้จึงถือเป็นหัวใจหลักของโพรโตคอล JXTA

  ในโพรโตคอล JXTA ได้กำหนดชนิดของ Advertisement ไว้ดังนี้

  1. *Peer advertisement*: ใช้บอกรายละเอียดต่างๆ ที่เกี่ยวกับเพียร์นั้น เช่น ID, ชื่อ, endpoint interface, กลุ่มเพียร์ เป็นต้น
  2. *Peer info advertisement*: ใช้บอกข้อมูลเกี่ยวกับทรัพยากรของเพียร์นั้น เช่น สถานะปัจจุบัน, Uptime, จำนวน Message ที่รับและส่ง, และเวลาที่รับหรือส่ง message ครั้งสุดท้าย เป็นต้น
  3. *Peer group advertisement*: ใช้บอกรายละเอียดของ Peer group นั้น เช่น ชื่อ, ID, ข้อมูลเฉพาะ, และบริการ เป็นต้น
  4. *Pipe advertisement*: ใช้บอกถึงข้อมูลที่จำเป็นในการสร้างช่องทางสื่อสารหรือไปป์
  5. *Module class advertisement*: ใช้บอกรายละเอียดของ Module class ซึ่งประกอบด้วย ชื่อ, ข้อมูลเฉพาะ, และ ID (Module Class ID) จุดประสงค์หลักเพื่อบอกถึงการมีอยู่ของโมดูลหรือบริการนั้น

6. *Module spec advertisement*: ใช้เพื่อบอกถึงวิธีในการเข้าถึงหรือใช้งานโมดูลนั้น และเป็นการยืนยันว่าบริการนั้นสามารถใช้งานได้ (ในบางกรณีผู้ใช้สามารถประกาศโมดูลได้ก่อนที่จะอิมพลีเมนต์ขึ้นมาจริงๆ ทำให้โมดูลนั้นมีแต่ Module class แต่ไม่มี Module specification)
7. *Module impl advertisement*: ระบุถึงรายละเอียดของส่วนที่อิมพลีเมนต์ เช่นตัวแปรที่ต้องใช้ในการใช้บริการนี้ และบริการนี้จะส่งค่ากลับเป็นชนิดอะไร เป็นต้น
8. *Rendezvous advertisement*: บอกถึง Rendezvous peer ของ Peer group นั้น

```
<?xml version="1.0"?>

<!DOCTYPE jxta:PA>

<jxta:PA xmlns:jxta="http://jxta.org">
     <PID>
         urn:jxta:uuid-
59616261646162614A78746150325033F3BC76FF13C2414CBC0AB663666DA53903
     </PID>
     <GID>
         urn:jxta:jxta-NetGroup
     </GID>
     <Name>
         suz
     </Name>
     <Svc>
         <MCID>
             urn:jxta:uuid-DEADBEEFDEAFBABAFEEDBABE0000000805
         </MCID>
         <Parm>
             <Addr>
                 tcp://192.168.1.100:9701/
             </Addr>
             <Addr>
                 jxtatls://uuid-
59616261646162614A78746150325033F3BC76FF13C2414CBC0AB663666DA53903/
TlsTransport/jxta-WorldGroup
             </Addr>
             <Addr>
                 jxta://uuid-
59616261646162614A78746150325033F3BC76FF13C2414CBC0AB663666DA53903/
             </Addr>
             <Addr>
                 http://JxtaHttpClientuuid-
59616261646162614A78746150325033F3BC76FF13C2414CBC0AB663666DA53903/
             </Addr>
         </Parm>
     </Svc>
</jxta:PA>
```

รูปที่ ก.2 ตัวอย่าง Peer advertisement

- **Modules**

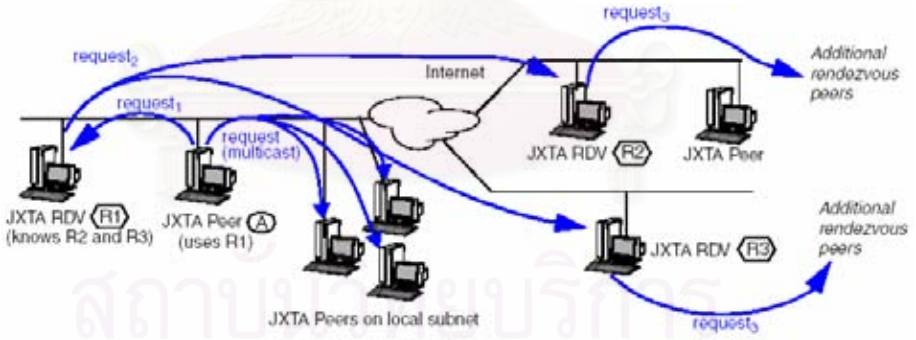
  โมดูล คือส่วนของฟังก์ชันการทำงานหรือบริการที่ถูกออกแบบมานอกเหนือจากที่ JXTA ได้สร้างไว้ให้ ซึ่งขึ้นอยู่กับความต้องการของผู้ใช้แต่ละคน เช่น ระบบส่งข้อมูลที่มีความรวดเร็วและปลอดภัยกว่าไปป์ เป็นต้น ซึ่งแต่ละกลุ่มเพียร์จะมีโมดูลเป็นของตนเองและทุกๆ เพียร์ในกลุ่มจะรับรู้ถึงโมดูลนี้ ใน JXTA ประกอบด้วยโมดูล 3 โมดูลด้วยกัน คือ

  1. **Module class**: ใช้สำหรับประกาศถึงการมีอยู่ของโมดูลหรือบริการนั้น
  2. **Module specification**: บอกถึงรายละเอียดของบริการว่าใช้ทำอะไร รวมถึงวิธีในการเข้าถึงหรือใช้งานบริการนั้น
  3. **Module implementation**: คือตัวของโมดูลหรือบริการจริงๆ ที่ถูกเขียนขึ้นโดยผู้ใช้

การทำงานของโพรโตคอล JXTA ที่อิมพลีเมนต์ด้วยภาษา Java

ในส่วนนี้จะกล่าวถึงการทำงานหลักๆ ของโพรโตคอล JXTA โดยจะกล่าวถึงเฉพาะโพรโตคอลที่อิมพลีเมนต์ด้วยภาษา Java เท่านั้น

ก. การส่ง Request ไปยังเพียร์ต่างๆ



รูปที่ ก.3 การกระจาย Request ใน JXTA

ในรูปที่ ก.3 เพียร์ A เป็น Edge peer และถูกตั้งค่าให้ใช้เพียร์ R1 เป็น Rendezvous peer เมื่อเพียร์ A เริ่มต้นหาข้อมูลที่ต้องการ ตัว Request จะถูกส่งไปที่ Rendezvous peer ซึ่งในที่นี้ก็คือเพียร์ R1 และ Request นั้นก็จะถูกส่งไปยังเพียร์อื่นๆ ที่อยู่ใน Subnet เดียวกันด้วย ในการกระจายครั้งแรกนี้ถ้ามีเพียร์ใดมีข้อมูลที่ Request นั้นต้องการ เพียร์นั้นก็จะส่งข้อมูลกลับไปยังเพียร์ A โดยตรงและเพียร์ R1 ก็จะไม่ส่ง Request ต่อไป แต่ถ้าไม่มีเพียร์ใดมีข้อมูลที่ต้องการเลย เพียร์ R1 ก็จะกระจาย Request ไปยัง Rendezvous peer อื่นๆ ที่ตนรู้จัก (เพียร์ R2 และเพียร์ R3) ต่อไปเรื่อยๆ จนกว่า Request นั้นจะหมด TTL

หรือพบเพียร์ที่มีข้อมูลที่ต้องการ Rendezvous peer นั้นก็จะแจ้งเพียร์นั้นให้ส่งข้อมูลกลับไปยังเพียร์ A โดยตรง

ข. การสร้างและประกาศ Advertisement

ใน JXTA นั้นมีหลายวิธีที่เราจะได้ Advertisement ที่ต้องการมา วิธีหนึ่งคือการรับ Advertisement มาจากเพียร์อื่นที่ส่งมาให้ อีกวิธีหนึ่งในกรณีที่เรามี Advertisement ที่ถูกเก็บอยู่ในรูปของไฟล์ภายในเครื่อง เราสามารถที่จะดึงข้อมูลในไฟล์นั้นมาสร้างเป็น Advertisement ที่ต้องการได้ รูปที่ ก.4 เป็นตัวอย่างโค้ดที่ใช้สำหรับดึงข้อมูลในไฟล์ “service1.adv” มาเพื่อสร้าง Pipe advertisement

```
PipeAdvertisement myPipeAdvertisement = null;
try {
   FileInputStream is = new FileInputStream(“service1.adv”);
   myPipeAdvertisement = (PipeAdvertisement)AdvertisementFactory.
                              newAdvertisement(new MimeMediaType(“text/xml”)
                              , is);
} catch (Exception e) { /*Read/parse advertisement failed*/}
```

รูปที่ ก.4 ตัวอย่างโค้ดสำหรับสร้าง Pipe advertisement จากไฟล์

วิธีสุดท้ายคือการสร้าง Advertisement ขึ้นมาใหม่ทั้งหมดซึ่งจะได้ ID ของ Advertisement นั้นใหม่ด้วย ตัวอย่างโค้ดแสดงในรูปที่ ก.5

```
PipeAdvertisement myPipeAdvertisement = (PipeAdvertisement)
        AdvertisementFactory.newAdvertisement(PipeAdvertisement.
            getAdvertisementType());
myPipeAdvertisement.setName(“PipeAdvertisementName”)
myPipeAdvertisement.setType(“JxtaUnicast”);
myPipeAdvertisement.setPipeID((ID)net.jxta.id.IDFactory.
        newPipeID (peerGroupID));
```

รูปที่ ก.5 ตัวอย่างโค้ดสำหรับการสร้าง Pipe advertisement ใหม่

การประกาศ Advertisement ใน JXTA มี 2 แบบด้วยกันคือ LocalPublish กับ RemotePublish โดย LcalPublish จะทำการเก็บ Advertisement นั้นลงในแคชของตนเท่านั้นแต่ไม่กระจาย Advertisement ไปยังเพียร์อื่นๆ ส่วน RemotePublish จะทำการส่ง Advertisement นั้นไปให้เพียร์อื่นๆ ได้รับรู้ด้วย รูปที่ ก.6 แสดงตัวอย่างโค้ดในการประกาศ Pipe advertisement ทั้งแบบ Local และ Remote

```
try {
   myDiscoveryService.publish(myPipeAdvertisement);
   myDiscoveryService.remotePublish(myPipeAdvertisement);
} catch (Exception e) { /*Publish Advertisement failed*/ }
```

รูปที่ ก.6 ตัวอย่างโค้ดสำหรับประกาศ Pipe advertisement ทั้งแบบ local และ remote

ค. การค้นหาทรัพยากรต่างๆ ใน Peer group

ดังที่ได้กล่าวมาข้างต้นแล้วว่าทรัพยากรของ JXTA จะถูกนำเสนออยู่ในรูปของ Advertisement ดังนั้นการค้นหาทรัพยากรในระบบก็คือการหา Advertisement ของทรัพยากรที่ต้องการนั่นเอง โดยการค้นหาจะใช้บริการของ DiscoveryService เพื่อทำการหา Advertisement ที่มีคุณสมบัติตามที่กำหนดทั้งในแคชของตนเอง (getLocalAdvertisements( )) และหาจากแคชของเพียร์อื่นๆ ที่อยู่ใน Peer group เดียวกัน (getRemoteAdvertisements( )) ส่วนการรับผลการค้นหา (Discovery response messages) ที่มาจากเพียร์อื่นมีอยู่ 2 วิธี วิธีแรกคือรอจนมีเพียร์ใดตอบกลับมา แล้วจึงเรียก getLocalAdvertisements( ) เพื่อหา Advertisement นั้นในแคชของตน (Advertisement ที่ตอบกลับมาจากเพียร์อื่นจะถูกเก็บในแคชเครื่องที่รับโดยอัตโนมัติ) ซึ่งวิธีนี้ไม่ค่อยเป็นที่นิยม เนื่องจากเราไม่สามารถรู้ได้ว่าเมื่อไรจะมีเพียร์อื่นตอบกลับมา อีกวิธีหนึ่งคือการใช้ DiscoveryListener ที่จะเรียก Method discoveryEvent( ) โดยอัตโนมัติเมื่อมีข้อมูลจากเพียร์อื่นตอบกลับมา รูปที่ ก.7 แสดงตัวอย่างโค้ดสำหรับการหาเพียร์ที่มีชื่อขึ้นต้นด้วยคำว่า “Peer” จากเพียร์อื่นและกำหนดให้แต่ละเพียร์ตอบกลับมาเพียง 5 Advertisement เท่านั้น และสร้าง DiscoveryListener เพื่อรอรับผลการค้นหาจากเพียร์อื่น

```
try {
   DiscoveryListener myDiscoveryListener = new DiscoveryListener() {
                public void discoveryEvent(DiscoveryEvent e) {
                   DiscoveryResponseMsg myMessage = e.getResponse();
                   Enumeration enum = myMeassage.getResponses();
                   // ……
   }
};
myDiscoveryService.getRemoteAdvertisement(null,DiscoveryService.PEER,
      “Name”,  “Peer*”, 5, myDiscoveryListener);
} catch (Exception e) { }
```

รูปที่ ก.7 ตัวอย่างโค้ดสำหรับหาเพียร์ที่ชื่อขึ้นต้นด้วยคำว่า “Peer”

ง. การส่ง Message ระหว่างเพียร์

การส่ง Message ใน JXTA จะใช้บริการ PipeService เพื่อใช้ในการสร้างและเข้าถึงไปป์ที่อยู่ใน Peer group “ไปป์” เป็นตัวหลักในการส่งข้อมูลระหว่างเพียร์ โดยข้อมูลที่ส่งจะต้องอยู่ในรูปแบบของ Message ขั้นตอนพื้นฐานในการส่ง Message ระหว่างเพียร์เป็นดังนี้

เพียร์ที่ต้องการรับ Message จากเพียร์อื่น

1. สร้าง Input pipe advertisement
2. สร้าง Input pipe จาก Input pipe advertisement ที่สร้างไว้
3. ประกาศ Input pipe advertisement
4. รอรับ Message โดยใช้การ Polling หรือใช้ PipeMsgListener

เพียร์ที่ต้องการส่ง Message

1. ค้นหา Pipe advertisement ของเพียร์ที่ต้องการส่ง Message ไปถึง
2. สร้าง Output pipe จาก Pipe advertisement ที่พบ (Pipe advertisement ของเพียร์ปลายทาง)
3. ส่ง Message ผ่านไปป์ที่สร้างไว้

รูปที่ ก.8 แสดงตัวอย่างการสร้าง Input pipe จาก Input pipe advertisement ที่ได้สร้างมาแล้วข้างต้น และสร้าง PipeMsgListener เพื่อรับและจัดการกับ Message ที่ได้รับ

```
PipeMsgListener myServiceListener = new PipeMsgListener() {
   Public void pipeMsgEvent(PipeMsgEvent event) {
                Message myMessage = null;
                try {
                   myMessage = event.getMessage();

                   //…code for handle with message

                } catch (Exception e) { }
   }
};
InputPipe myInputPipe = null;
try {
      myInputPipe = myPipeService.createInputPipe(myPipeAdvertisement,
                        myServiceListener);
} catch (exception e) { /*Create Input Pipe Error*/ }
```

รูปที่ ก.8 ตัวอย่างโค้ดสำหรับสร้าง Input pipe และตัว PipeMsgListener

ส่วนการสร้าง Output pipe นั้น เมื่อได้ Pipe advertisement ของเพียร์ที่ต้องการส่ง Message ไปถึงก็ทำการเรียก *PipeService.createOutputPipe(myPipeAdvertisement, 10000)* ตัวเลข 10000 บอกถึงให้พยายามสร้างเป็นเวลา 10 วินาที ถ้าสร้างไม่สำเร็จภายใน 10 วินาที ก็ถือว่าการสร้างไม่สมบูรณ์ ส่วนการส่ง Message ก็จะใช้ Method *OutputPipe.send(msg)*

ภาคผนวก ข.

ผลงานที่ได้รับการตีพิมพ์จากงานวิจัย

ส่วนหนึ่งของงานวิทยานิพนธ์นี้ได้รับการตีพิมพ์เป็นบทความทางวิชาการจำนวน 2 ฉบับ ดังนี้

1. หัวข้อ “การใช้การสื่อสารแบบเพียร์-ทู-เพียร์บนระบบการคำนวณแบบกระจายเพื่อลดปัญหาคอขวดบนเครื่องเซิร์ฟเวอร์ (Using Peer-to-Peer Communication for Distributed Computing System to Reduce Network Bottleneck on Server)” โดย นายเกษม ตริตระการ และ อ.ดร.วีระเหมืองสิน ในงานประชุมวิชาการ “The 8th National Computer Science and Engineering Conference (NCSEC 2004)” ซึ่งจัดโดย คณะวิศวกรรมศาสตร์ มหาวิทยาลัยสงขลานครินทร์ ณ โรงแรม J.B. หาดใหญ่ ในวันที่ 21-22 ตุลาคม 2547

2. หัวข้อ “Using Peer-to-Peer Communication to Improve the Performance of Distributed Computing on the Internet” โดยนายเกษม ตริตระการ และ อ.ดร.วีระเหมืองสิน ในงานประชุมวิชาการ “The IEEE 19th International Conference on Advanced Information Networking and Applications (AINA 2005)” ซึ่งจัดโดย IEEE Computer Society TCDP ณ Tamkang University ประเทศไต้หวัน ในวันที่ 28-30 มีนาคม 2548

การใช้การสื่อสารแบบเพียร์-ทู-เพียร์บนระบบการคำนวณแบบกระจาย เพื่อลดปัญหาคอขวดบนเครื่องเซิร์ฟเวอร์

Using Peer-to-Peer Communication for Distributed Computing System to Reduce Network Bottleneck on Server

Kasame Tritrakan[1] and Veera Muangsin[2]

Department of Computer Engineering, Chulalongkorn University

Patumwan, Bangkok 10330, Thailand

e-mail: kasame.t@student.chula.ac.th[1], veera.m@chula.ac.th[2]

บทคัดย่อ

ในปัจจุบันมีระบบการคำนวณแบบกระจายจำนวนมากที่ใช้พลังการประมวลผลจากเครื่องคอมพิวเตอร์ที่มีอยู่มากมายบนเครือข่ายอินเตอร์เน็ต ระบบเหล่านี้ส่วนใหญ่มีโครงสร้างแบบไคลเอนต์/เซิร์ฟเวอร์ซึ่งมีจุดด้อยที่สำคัญคือ เมื่อข้อมูลของงานมีขนาดใหญ่ และจำนวนเครื่องที่รับงานไปประมวลผลในระบบเพิ่มมากขึ้น ความต้องการในการใช้แบนด์วิดธ์ของเซิร์ฟเวอร์เพื่อส่งข้อมูลก็จะเพิ่มขึ้นด้วยจนอาจทำให้เกิดปัญหาคอขวดขึ้นที่เซิร์ฟเวอร์ ซึ่งส่งผลกระทบให้ประสิทธิภาพการทำงานของระบบลดลงอย่างมาก

งานวิจัยนี้นำเสนอการนำเอารูปแบบการสื่อสารแบบเพียร์-ทู-เพียร์มาประยุกต์ใช้กับระบบการคำนวณแบบกระจายเพื่อลดปัญหาคอขวดที่เซิร์ฟเวอร์ และช่วยให้ระบบสามารถรองรับงานที่มีข้อมูลขนาดใหญ่ได้ อีกทั้งสามารถรองรับจำนวนเครื่องที่รับงานไปประมวลผลได้เพิ่มมากขึ้นกว่าโครงสร้างแบบไคลเอนต์/เซิร์ฟเวอร์ และศึกษาถึงผลกระทบของปัจจัยต่างๆ ที่มีผลต่อประสิทธิภาพของระบบ

Abstract

There are many distributed computing systems that harvest the computing power of computers on the Internet. Most of these systems use the client/server model. However, when the size of data and the number of resources increase, the demand for server's bandwidth also increases. This causes the network bottleneck problem, which dramatically reduces the performance of the system.

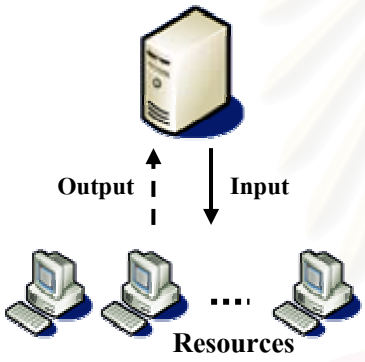
In this paper, we apply peer-to-peer communication to such a distributed computing system in order to reduce the network bottleneck on the server. We evaluate the performance of our model and study the impact of various parameters on the performance of the system.

Key-Words: peer-to-peer communication, distributed computing, network bottleneck

1. บทนำ

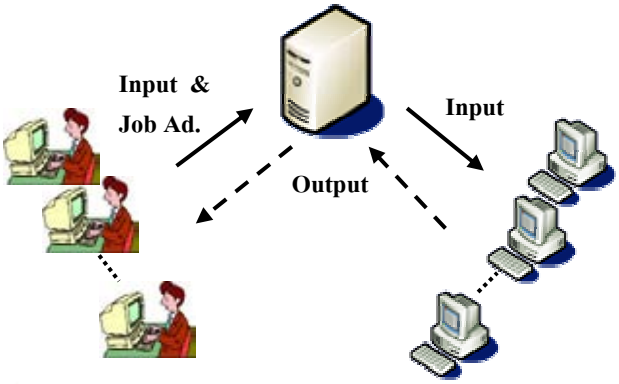
จากความสำเร็จของโครงการ SETI@Home [1] ทำให้มีระบบการคำนวณแบบกระจาย (Distributed Computing System) ที่ใช้ทรัพยากรการคำนวณจากเครื่องคอมพิวเตอร์ที่มี

อยู่มากมายบนเครือข่ายอินเตอร์เน็ตเกิดขึ้นตามมาเป็นจำนวนมาก เช่น FightAIDS@Home [2], Folding@Home [3], และ Genome@Home [4] ที่เครื่องเซิร์ฟเวอร์ทำหน้าที่สร้างและแจกจ่ายงานให้แก่ระบบดังรูปที่ 1 ต่อมาก็ได้มีระบบที่ผู้ใช้สามารถส่งงานของตนเองมาประมวลผลในระบบได้ เช่น Entropia [5], Javelin [6], และ Frontier [7] โดยระบบเหล่านี้ส่วนใหญ่จะมีโครงสร้างแบบไคลเอนต์/เซิร์ฟเวอร์ (Client/Server) (รูปที่ 2) ซึ่งประกอบด้วยเซิร์ฟเวอร์ที่ทำหน้าที่ควบคุมและจัดลำดับงานทั้งหมดในระบบ รวมถึงเป็นศูนย์กลางในการสื่อสารและรับ-ส่งข้อมูลต่างๆ ในระบบ และไคลเอนต์ซึ่งแบ่งเป็น 2 ส่วนคือ เครื่องเจ้าของงาน (submitter) (ในกรณีที่ผู้ใช้สามารถส่งงานมาได้) และเครื่องที่รับงานไปประมวลผล (resource) อย่างไรก็ตามโครงสร้างแบบไคลเอนต์/เซิร์ฟเวอร์มี



รูปที่ 1. โครงสร้างระบบแบบ SETI@home, FightAIDS@Home, และ Genome@Home

จุดด้อยที่สำคัญคือ หากจำนวนไคลเอนต์ (submitter + resource) ในระบบมีเพิ่มมากขึ้น ความต้องการในการใช้ทรัพยากรเครือข่ายหรือแบนด์วิดธ์ของเซิร์ฟเวอร์ เพื่อส่งข้อมูลต่างๆ ก็จะเพิ่มขึ้นด้วย ดังนั้นเมื่อเซิร์ฟเวอร์ต้องแบกรับการส่งข้อมูลที่มากเกินกว่าขนาดแบนด์วิดธ์ของเซิร์ฟเวอร์ที่มีอยู่อย่างจำกัดจะก่อให้เกิดปัญหาคอขวดขึ้นที่เซิร์ฟเวอร์ซึ่งจะทำให้อัตราเร็วในการส่งข้อมูลไปยังเครื่องไคลเอนต์ในระบบช้าลง ซึ่งส่งผลกระทบต่อประสิทธิภาพโดยรวมของระบบอย่างมาก วิธีการแก้ปัญหาดังกล่าวอาจทำได้โดยการเพิ่มแบนด์วิดธ์ให้แก่เครื่องเซิร์ฟเวอร์ และการใช้เซิร์ฟเวอร์หลายๆ เครื่อง (Multiple servers) [8] แต่ทั้งสองวิธีเป็นการเพิ่มค่าใช้จ่ายให้แก่ระบบ



รูปที่ 2. รูปแบบการสื่อสารแบบไคลเอนต์/เซิร์ฟเวอร์

ข้อมูลส่วนใหญ่ที่เซิร์ฟเวอร์ต้องแบกรับคือ

1. อินพุต เป็นข้อมูลที่ใช้สำหรับการประมวลผล
2. Job Advertisement (Job Ad.) เป็นข้อมูลเกี่ยวกับชิ้นงาน เช่น ชื่อเจ้าของงาน, หมายเลขไอพี, ความต้องการขั้นต่ำของเครื่องคอมพิวเตอร์ที่สามารถทำงานนี้ได้ เป็นต้น ซึ่งส่วนใหญ่จะมีขนาดเล็ก โดย Job Ad. นี้มีความจำเป็นต่อเซิร์ฟเวอร์เพื่อใช้ในการจัดลำดับและมอบหมายงานให้แก่เครื่องที่รับงานไปประมวลผล ซึ่งอาจอยู่ในรูปของไฟล์ข้อความหรือไฟล์ script ก็ได้
3. Resource Advertisement (Resource Ad.) เป็นข้อมูลที่จำเป็นสำหรับการสร้างการเชื่อมต่อไปยังเครื่องนั้นๆ เช่น หมายเลขไอพี เป็นต้น
4. เอาต์พุต เป็นผลลัพธ์ที่ได้จากการประมวลผล

ถ้าข้อมูลเหล่านี้มีขนาดใหญ่ก็จะส่งผลให้แบนด์วิดธ์ของเครื่องเซิร์ฟเวอร์ถูกใช้มากขึ้น ตัวอย่างเช่นระบบ SETI@home ที่จำเป็นต้องใช้เซิร์ฟเวอร์ที่มีแบนด์วิดธ์สูงถึง 70 Mbps และเสียค่าแบนด์วิดธ์ประมาณ 21,000 ดอลลาร์ต่อเดือน [9] ซึ่งเป็นไปไม่ได้ที่จะทำได้อย่างแพร่หลายในปัจจุบัน ดังนั้นแอพพลิเคชันที่เหมาะสมกับระบบแบบนี้จึงมักเป็นงานที่เน้นการคำนวณ (computing intensive applications) ที่มีขนาดของข้อมูลน้อยแต่ใช้เวลาในการประมวลผลมาก เพื่อความเหมาะสมในการนำไปใช้กับระบบที่ทำงานอยู่บนเครือข่ายอินเตอร์เน็ต [10]

เพื่อให้รูปแบบการคำนวณแบบกระจายบนอินเตอร์เน็ตเป็นที่แพร่หลายและทำได้ง่ายตั้งแต่ระบบขนาดเล็กถึงใหญ่ ที่มีการลงทุนเริ่มต้นน้อย เซิร์ฟเวอร์ไม่จำเป็นต้องมีแบนด์วิดธ์ที่สูง จึง

จำเป็นที่จะต้องแก้ปัญหาเรื่องแบนด์วิดธ์ของเซิร์ฟเวอร์โดยประหยัดค่าใช้จ่ายให้มากที่สุด

ในเครือข่ายแบบเพียร์-ทู-เพียร์ [11] มีการส่งข้อมูลโดยตรงจากเครื่องหนึ่งไปยังอีกเครื่องหนึ่งโดยไม่ผ่านเซิร์ฟเวอร์ ซึ่งเป็นการใช้แบนด์วิดธ์ของไคลเอนต์เอง ซึ่งแบนด์วิดธ์โดยรวมของระบบจะขยายขึ้นตามจำนวนไคลเอนต์ ส่งผลให้ระบบสามารถรองรับจำนวนเครื่องได้เพิ่มขึ้นโดยไม่ต้องเพิ่มขนาดแบนด์วิดธ์ของเซิร์ฟเวอร์แต่อย่างใด

งานวิจัยนี้เสนอการนำเอารูปแบบการสื่อสารและส่งข้อมูลของเครือข่ายเพียร์-ทู-เพียร์มาประยุกต์ใช้กับระบบการคำนวณแบบกระจายที่ผู้ใช้สามารถส่งงานของตนเข้ามาประมวลผลยังระบบได้ซึ่งเป็นรูปแบบที่องค์กรส่วนใหญ่ต้องการ เพื่อลดปัญหาคอขวดที่เซิร์ฟเวอร์ เพื่อให้ระบบมีประสิทธิภาพเพิ่มมากขึ้น สามารถรองรับแอพพลิเคชันที่มีข้อมูลขนาดใหญ่ได้ และมีความสามารถในการขยายระบบ (scalability) เพิ่มมากขึ้น ในการทดลองจะใช้โปรแกรมจำลอง (simulator) เพื่อจำลองการทำงานของระบบและวัดประสิทธิภาพของระบบเปรียบเทียบกับโครงสร้างแบบไคลเอนต์/เซิร์ฟเวอร์ และศึกษาถึงผลกระทบของขนาดของข้อมูลและแบนด์วิดธ์ของเซิร์ฟเวอร์ที่ส่งผลต่อประสิทธิภาพการทำงานของระบบ โดยจะมุ่งเน้นที่ระบบที่เซิร์ฟเวอร์มีแบนด์วิดธ์น้อย เช่นเท่ากับเว็บเซิร์ฟเวอร์ขนาดกลาง เพื่อศึกษาถึงความเป็นไปได้ที่วิธีที่เสนอนี้จะสามารถนำไปใช้กับระบบขนาดเล็กหรือขนาดกลาง

2. โครงสร้างของระบบ

2.1 ระบบการคำนวณแบบกระจายแบบไคลเอนต์/เซิร์ฟเวอร์

โครงสร้างแบบไคลเอนต์/เซิร์ฟเวอร์ถูกนำมาใช้อย่างแพร่หลายเนื่องจากมีความสามารถในการควบคุมการทำงาน (Manageability) และการรักษาความปลอดภัยที่ดี [12] โดยในโครงสร้างแบบไคลเอนต์/เซิร์ฟเวอร์นั้น เซิร์ฟเวอร์ต้องทำหน้าที่จับคู่ระหว่างงานกับเครื่อง (matchmaker) และเป็นตัวกลางในการส่งข้อมูลอินพุตจากเจ้าของงานไปยังเครื่องที่รับงานไปประมวลผล และเอาต์พุตจากเครื่องที่รับงานไปประมวลผลไปยังเจ้าของงาน ดังนั้นเวลาที่ใช้ในการส่งข้อมูลไปยังแต่ละเครื่องในระบบจึงมีผลต่อเวลาที่ใช้ในการทำงานแต่ละชิ้น (response time) (Eq1)

$$\textit{Response time} = T_{in} + T_{queue} + T_{comp} + T_{out} \qquad \textbf{(Eq1.)}$$

เมื่อจำนวนเครื่องที่รับงานไปประมวลผลมีมากขึ้น ถึงแม้ว่าจะเป็นการลดระยะเวลาในการคำนวณ (T_{comp}) แต่ถ้าแบนด์วิดธ์ของเซิร์ฟเวอร์ไม่สัมพันธ์กับจำนวนเครื่องที่รับงานไปประมวลผลจะส่งผลให้เกิดคอขวดขึ้นที่เซิร์ฟเวอร์ ซึ่งทำให้เวลาที่อินพุตถูกส่งไปยังเครื่องที่รับงานไปประมวลผล (T_{in}) เพิ่มขึ้น และเวลาที่เครื่องที่รับงานไปประมวลผลส่งคืนเอาพุตกลับไปยังเซิร์ฟเวอร์ (T_{out}) เพิ่มขึ้นด้วย ดังนั้น *Response time* ก็จะเพิ่มขึ้นอีกทั้งเครื่องนั้นจะรับงานใหม่มาประมวลผลได้ช้าลงงานจึงค้างอยู่ในคิวมากและเวลาที่งานรออยู่ในคิว (T_{queue}) ก็จะเพิ่มขึ้น ดังนั้นโครงสร้างแบบไคลเอนต์/เซิร์ฟเวอร์จึงสามารถรองรับจำนวนเครื่องที่รับงานไปประมวลผลได้อย่างจำกัด

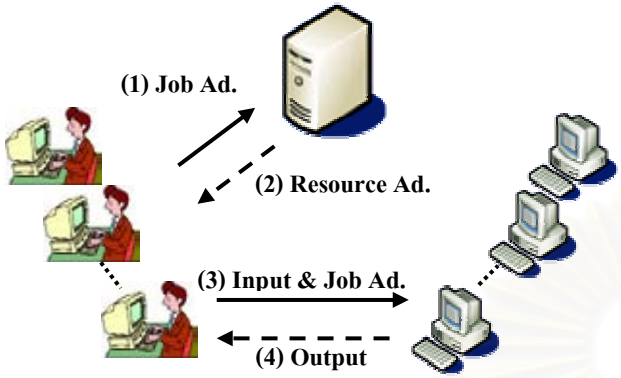
ถ้าเซิร์ฟเวอร์สามารถส่งข้อมูลดังกล่าวได้เร็วก็จะช่วยให้เวลาในการสื่อสาร ($T_{in} + T_{out}$) ของงานนั้นลดลง และ T_{queue} ของงานอื่นที่ตามมาก็จะลดลงไปด้วย ซึ่งส่งผลให้ *Response time* ของงานโดยรวมลดลง

2.2 ระบบการคำนวณแบบกระจายแบบเพียร์-ทู-เพียร์

ในระบบแบบเพียร์-ทู-เพียร์นั้นจะพยายามส่งข้อมูลระหว่างเจ้าของงานและเครื่องที่รับงานไปประมวลผลโดยตรงให้มากที่สุดเท่าที่เป็นไปได้ โดยมีขั้นตอนการทำงานดังนี้ (รูปที่3)

(1) เจ้าของงานส่ง Job Ad. ไปยังเซิร์ฟเวอร์เพื่อหาเครื่องที่เหมาะสมที่จะรับงานนั้นไปประมวลผล
(2) ถ้าในขณะนั้นมีเครื่องที่สามารถรับงานไปประมวลผลได้ทันที เซิร์ฟเวอร์ก็จะส่ง Resource Ad. (Resource Advertisement) ที่ประกอบด้วยข้อมูลที่จำเป็นในการเชื่อมต่อไปยังเครื่องที่จะรับงานไปประมวลผล) กลับมายังเจ้าของงาน
(3) เมื่อทั้งสองเครื่องสามารถเชื่อมต่อกันได้แล้ว อินพุต และ Job Ad. ก็จะถูกส่งไปยังเครื่องที่รับงานไปประมวลผลโดยตรงโดยไม่ต้องผ่านเซิร์ฟเวอร์

(4) เมื่องานถูกประมวลผลเสร็จแล้ว เครื่องที่รับงานไปประมวลผลจะใช้ข้อมูลที่อยู่ใน Job Ad. ทำการเชื่อมต่อไปยังเจ้าของงาน เพื่อส่งเอาต์พุตกลับไป



รูปที่ 3. รูปแบบที่มีการประยุกต์ใช้กับสื่อสารแบบเพียร์-ทู-เพียร์

ด้วยเทคนิคนี้ทำให้ภาระในการส่งข้อมูลและการใช้แบนด์วิดธ์ของเซิร์ฟเวอร์ลดลง โดยข้อมูลสามารถส่งไปยังที่หมาย (เครื่องเจ้าของงาน หรือเครื่องที่รับงานไปประมวลผล) ได้โดยตรงโดยไม่ต้องใช้เซิร์ฟเวอร์เป็นตัวช่วยส่งต่อข้อมูล

อย่างไรก็ตามกรณีที่ไม่มีเครื่องว่างที่จะรับงานไปประมวลผลงานนั้นก็จะถูกเก็บไว้ในคิว ในกรณีนี้เครื่องเจ้าของงานจะได้รับแจ้งให้ส่งอินพุตไปเก็บไว้ที่เซิร์ฟเวอร์เพื่อป้องกันกรณีที่เจ้าของงานออกไปจากระบบทำให้ไม่มีอินพุตสำหรับการประมวลผล และเมื่อมีเครื่องที่สามารถรับงานนั้นได้แล้ว เซิร์ฟเวอร์ก็จะทำหน้าที่ส่งอินพุต และ Job Ad. ไปยังเครื่องที่รับงานไปประมวลผลนั้นแทน และในกรณีที่เมื่องานถูกประมวลผลเสร็จแล้วแต่ขณะนั้นเจ้าของงานไม่อยู่ในระบบทำให้เครื่องที่รับงานไปประมวลผลไม่สามารถส่งเอาต์พุตกลับไปยังเจ้าของงานโดยตรงได้ เอาต์พุตจึงจำเป็นต้องถูกส่งไปเก็บไว้ที่เซิร์ฟเวอร์เพื่อให้งานใหม่สามารถถูกส่งไปประมวลผลได้ทันที เมื่อเจ้าของงานกลับเข้ามายังระบบอีกครั้งก็จะมารับเอาต์พุตที่เครื่องเซิร์ฟเวอร์

3. การศึกษาโดยใช้แบบจำลอง (Simulation)

งานวิจัยนี้ผู้วิจัยได้ทำการจำลองระบบที่มีการสื่อสารทั้ง 2 แบบเพื่อศึกษาถึงประสิทธิภาพในการทำงานและผลกระทบของค่าพารามิเตอร์ต่างๆ ที่มีผลต่อประสิทธิภาพของระบบ ในการทดลองเราใช้โปรแกรมจำลอง GridSim Toolkit [13] จำลองระบบทั้ง 2 แบบขึ้น แต่เนื่องจาก GridSim ใช้รูปแบบของเครือข่ายการส่งข้อมูลแบบง่ายๆ (ไม่มี network sharing) ซึ่งไม่ตรงกับลักษณะการทำงานของระบบที่ต้องการ ผู้วิจัยจึงได้สร้างแบบจำลองของเครือข่ายขึ้นมาใหม่ให้สามารถจำลองการทำงานได้ถูกต้องยิ่งขึ้น

เนื่องจากผู้วิจัยต้องการมุ่งเน้นที่ระบบขนาดเล็กที่สร้างขึ้นได้จากทรัพยากรที่มีอยู่อย่างแพร่หลายในปัจจุบัน ค่าตัวแปรต่างๆ ในการทดลองนี้จึงอ้างอิงตามคุณสมบัติของเซิร์ฟเวอร์และคอมพิวเตอร์ที่มีอยู่ทั่วไป

Resource Modeling

เครื่องที่รับงานมาประมวลผลหมายถึงเครื่องคอมพิวเตอร์ของสมาชิกในระบบที่ยินดีรับงานมาประมวลผล โดยจำนวนเครื่องที่รับงานไปประมวลผลในการทดลองนี้จะหมายถึงจำนวนเครื่องในระบบที่สามารถรับงานมาประมวลผลได้ในขณะนั้น (ไม่รวมถึงเครื่องที่ไม่สามารถรับงานได้เนื่องจากเจ้าของเครื่องกำลังใช้งานอยู่) เครื่องคอมพิวเตอร์ในระบบจะมีความเร็วในการประมวลผลระหว่าง 250-350 MIPS (Million Instructions Per Second) ซึ่งอ้างอิงตามความเร็วของคอมพิวเตอร์ส่วนบุคคล (PC) ที่มีความเร็วประมาณ 1 GHz หน่วยความจำ 256 MB [14] แต่ละเครื่องจะรับงานมาประมวลผลครั้งละ 1 งานเท่านั้น และเชื่อมต่ออินเตอร์เน็ตด้วยโมเด็มที่ส่งข้อมูลที่อัตราเร็ว 56 Kb/s

Job Modeling

งานที่ใช้ในการทดลองนี้แต่ละงานจะเป็นอิสระต่อกัน (ไม่มีการสื่อสารระหว่างแต่ละงาน) แต่ละงานจะใช้ซีพียูเดียวในการประมวลผลและประมวลผลจนเสร็จโดยไม่มีการย้ายเครื่อง โดยงานจะเป็นงานที่มีขนาดอินพุตเล็กแต่เอาต์พุตมีขนาดใหญ่ซึ่งตรงกับแอพพลิเคชันหลายๆ ประเภท แต่ละงานจะประกอบด้วยอินพุตขนาด 0.5 MBytes, Job Ad. ขนาด 5 KBytes และเอาต์พุตขนาด 1 ถึง 3 MBytes ขนาดของงาน (จำนวนคำสั่ง) จะมีการกระจายแบบเลขชี้กำลัง (exponential distribution) ด้วยค่าเฉลี่ยเท่ากับ 1,620,000 MI (Million Instructions) ซึ่งจะใช้เวลาในการประมวลผลประมาณ 75-100 นาที งานจะถูกส่งเข้ามายังระบบด้วยอัตราเฉลี่ยทุกๆ 60 วินาที โดยมีการกระจายแบบเลขชี้กำลังเช่นเดียวกัน

Server Modeling

เครื่องเซิร์ฟเวอร์มีหน้าที่หลักในการจัดลำดับของงานและมอบหมายงานให้แก่เครื่องที่รับงานไปประมวลผลในระบบ อัลกอริทึมที่ใช้ในการจัดลำดับงานคือ FCFS (First-Come-First-Serve) เนื่องจากอัลกอริทึมนี้ง่ายแก่การอิมพลีเมนต์และให้ประสิทธิภาพสูงในกรณีที่ไม่สามารถคาดเดาขนาดและการมาของงานได้ [15] และกำหนดให้เครื่องเซิร์ฟเวอร์มีแบนด์วิดธ์เท่ากับ 512 Kb/s

Network Modeling

รูปแบบการสื่อสารนี้จะสามารถสามารถรองรับ network sharing ได้ โดยมีสมมติฐานว่าเครื่องที่อยู่ในระบบจะมีการกระจายกันอยู่บนผู้ให้บริการอินเตอร์เน็ต (ISP) จำนวนมาก ซึ่งเป็นการกระจายภาระในการส่งข้อมูลออกไปเพื่อไม่ให้เกิดคอขวดขึ้นที่ตัวผู้ให้บริการอินเตอร์เน็ตซึ่งจะมีผลต่อแบนด์วิดธ์จริงที่ผู้รับบริการจะได้รับ ดังนั้นจึงมีสมมติฐานว่าแต่ละเครื่องสามารถส่งข้อมูลด้วยอัตราเร็วคงที่ตามที่กำหนดข้างต้น

ตารางที่ 1 แสดงค่าตัวแปรต่างๆ ที่ใช้ในการทดลองนี้ ซึ่งค่าตัวแปรที่เลือกมานี้เป็นเพียงรูปแบบหนึ่งที่เป็นไปได้ในเครือข่ายอินเตอร์เน็ตปัจจุบันเท่านั้น

ตารางที่ 1. ค่าพารามิเตอร์ต่างๆ ที่ใช้ในการทดลองนี้

Number of Jobs	5000
Input File Size	0.5 MBytes
Job and Resource Ad. File Size	5 KBytes
Output File Size	1 - 3 MBytes
Number of Instructions	Exp(1,620,000) MI
Resources' Speed	250 – 350 MIPS
Server's Bandwidth	512 Kb/s
Submitters' and Resources' Bandwidth	56 Kb/s
Job Inter-arrival time	Exp(60) seconds

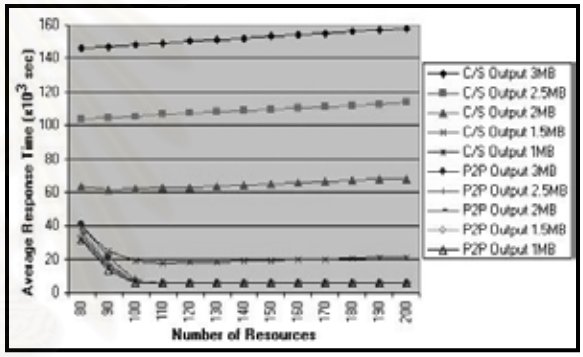
ในแต่ละการทดลองได้ทำการทดลองซ้ำกัน 10 ครั้งด้วยค่า Random seed ที่ต่างกัน ในแต่ละการทดลองจะทำการวัดค่าต่างๆ ดังนี้

- Average Response Time: เวลาเฉลี่ยตั้งแต่เจ้าของงานส่งงานเข้าสู่ระบบจนถึงเวลาที่เจ้าของงานได้เอาต์พุตกลับไป
- Average Server's Data Load: ปริมาณข้อมูลเฉลี่ยที่เครื่องเซิร์ฟเวอร์ต้องส่ง/รับ ณ เวลาใดๆ
- Average Queue Length: ความยาวเฉลี่ยของคิวงาน

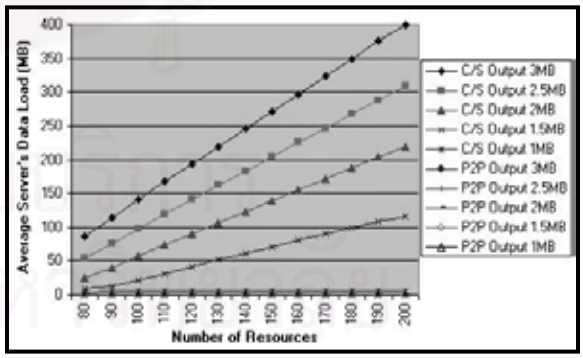
4. ผลการทดลองและวิเคราะห์ผล

4.1 ความสามารถในการขยายระบบ (Scalability)

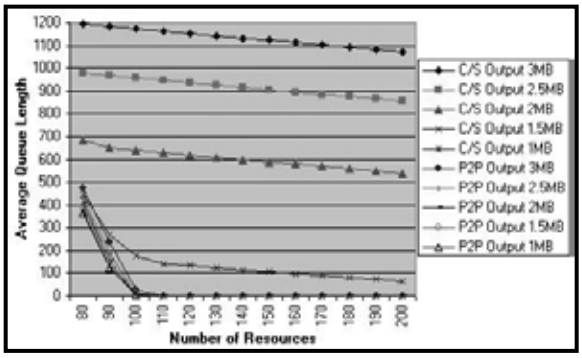
ในเบื้องต้นได้ทำการทดลองโดยเพิ่มจำนวนเครื่องที่รับงานไปประมวลผลในระบบขึ้นเรื่อยๆ เพื่อดูผลกระทบที่มีต่อประสิทธิภาพการทำงานของระบบ



รูปที่ 4. ค่า *Response time* เฉลี่ยของแต่ละงานเมื่อจำนวนเครื่องที่รับงานไปประมวลผลเพิ่มขึ้น



รูปที่ 5. ปริมาณข้อมูลเฉลี่ยที่เซิร์ฟเวอร์ต้องแบกรับเมื่อจำนวนเครื่องที่รับงานไปประมวลผลเพิ่มขึ้น



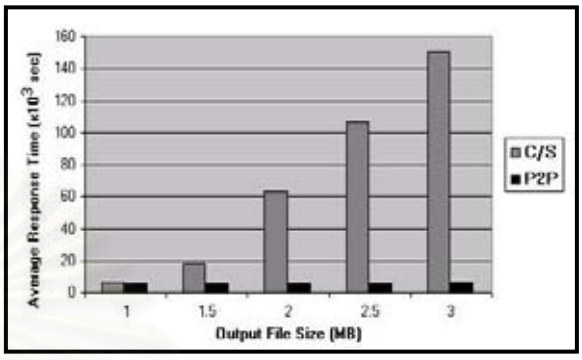
รูปที่ 6. ความยาวเฉลี่ยของคิวงานเมื่อจำนวนเครื่องที่รับงานไปประมวลผลเพิ่มมากขึ้น

จากการทดลองพบว่าในระบบแบบไคลเอนต์/เซิร์ฟเวอร์ที่มีขนาดของเอาต์พุตมากกว่า 1MB เมื่อจำนวนเครื่องที่รับงานในระบบเพิ่มมากขึ้นจะส่งผลให้ค่า *Response time* ของแต่ละงานกลับเพิ่มมากขึ้นดังรูปที่ 4 เนื่องจากการเพิ่มขึ้นของเครื่องที่รับงานทำให้ปริมาณข้อมูลที่เซิร์ฟเวอร์ต้องแบกรับ (รูปที่ 5) เพิ่มมากขึ้นตามไปด้วยทำให้ปัญหาคอขวดที่มีอยู่เดิมยิ่งทวีความรุนแรงมากขึ้น ทำให้ประสิทธิภาพการทำงานของระบบลดลงแทนที่จะเพิ่มขึ้น

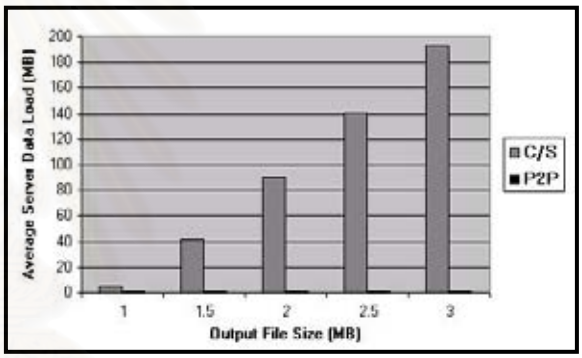
ในระบบแบบเพียร์-ทู-เพียร์ เมื่อจำนวนเครื่องที่รับงานไปประมวลผลตั้งแต่ 80 ถึง 100 เครื่อง พบว่าค่า *Response time* ของแต่ละงานมากกว่าค่า *Response time*$_{ideal}$ เนื่องจากปริมาณเครื่องที่รับงานไปประมวลผลในระบบไม่สามารถทำงานที่เข้ามาได้ทันทำให้มีบางงานจะต้องไปอยู่ในคิวทำให้อินพุตของงานนั้นก็จะต้องถูกส่งไปยังเซิร์ฟเวอร์ด้วย แต่อย่างไรก็ตามผลจากการทดลองแสดงว่า ปริมาณข้อมูลดังกล่าวโดยเฉลี่ยแล้วมีเพียงเล็กน้อยเท่านั้น (รูปที่ 5) ซึ่งไม่ก่อให้เกิดปัญหาคอขวดแต่อย่างใด แต่ที่ค่า *Response time* เพิ่มขึ้นนั้นเนื่องมาจากเวลาที่งานต้องรออยู่ในคิวเป็นเวลานาน (T_{queue}) (รูปที่ 6) ไม่ใช่เป็นเพราะปัญหาคอขวด ระบบแบบเพียร์-ทู-เพียร์จึงสามารถรองรับจำนวนเครื่องที่รับงานไปประมวลผลในระบบได้เพิ่มขึ้นโดยไม่เกิดปัญหาคอขวด

4.2 ผลกระทบของขนาดข้อมูล

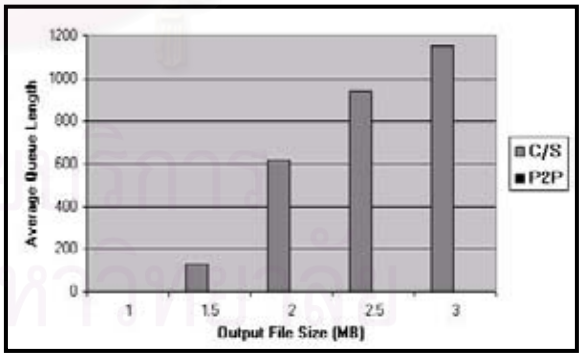
ต่อมาได้ทำการจำลองระบบโดยให้ระบบมีจำนวนเครื่องที่รับงานไปประมวลผลทั้งหมด 120 เครื่องซึ่งเป็นช่วงที่ระบบมีได้การเกิดปัญหาคอขวดขึ้น



รูปที่ 7. ค่า *Response time* เฉลี่ยของแต่ละงาน



รูปที่ 8. ปริมาณข้อมูลเฉลี่ยที่เซิร์ฟเวอร์ต้องแบกรับ



รูปที่ 9. ความยาวเฉลี่ยของคิวงาน

จากผลการทดลองซึ่งแสดงในรูปที่ 7, 8 และ 9 พบว่าโครงสร้างแบบไคลเอนต์/เซิร์ฟเวอร์ที่มีขนาดเอาต์พุตไฟล์ 1 MB ค่า *Response time* ของแต่ละงานจะใกล้เคียงกับค่า

$Response\ time_{ideal}$ (Eq2) ซึ่งคือค่า *Response Time* ในกรณีที่ไม่มีการเกิดปัญหาคอขวดขึ้น

$$Response\ time_{ideal} = T_{comp} + \frac{S_i + S_o}{BW} \qquad \textbf{(Eq2.)}$$

โดยที่ S_i และ S_o เป็นขนาดของอินพุตและเอาต์พุตของงาน และ BW เป็นขนาดแบนด์วิดธ์ของเครื่องเจ้าของงานและเครื่องที่รับงานไปประมวลผล ซึ่งเท่ากับ 56 Kb/s

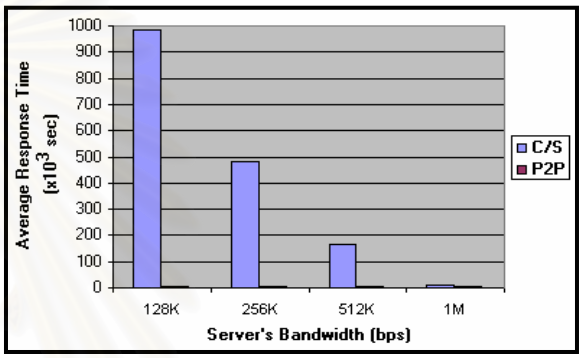
แต่เมื่อขนาดของเอาต์พุตเพิ่มขึ้นค่า *Response time* ของแต่ละงานจะเพิ่มขึ้นอย่างรวดเร็ว (รูปที่ 7) ซึ่งเป็นผลมาจากปัญหาคอขวดที่เกิดขึ้นอันเนื่องมาจากปริมาณข้อมูล (รูปที่ 8) ที่มากเกินกว่าที่เซิร์ฟเวอร์จะรับไหว และเมื่อเกิดปัญหาคอขวดขึ้นจะทำให้งานต่อๆ ไปจะต้องรออยู่ในคิวเป็นเวลานานเนื่องจากเครื่องที่รับงานไปประมวลผลนั้นส่งเอาต์พุตของงานกลับมายังเซิร์ฟเวอร์และรับงานใหม่ไปประมวลผลได้ช้า ส่งผลให้คิวงานของระบบยาวขึ้น (รูปที่ 9)

เมื่อใช้โครงสร้างแบบเพียร์-ทู-เพียร์ดังที่ได้กล่าวมาข้างต้น จะพบว่าค่า *Response Time* ของแต่ละงานจะเท่ากับค่า $Response\ Time_{ideal}$ (รูปที่ 7) ไม่ว่าเอาต์พุตจะมีขนาดเท่าใดก็ตาม เนื่องจากข้อมูลที่เซิร์ฟเวอร์ต้องรองรับจะมีเพียง Job Ad., Resource Ad., และ อินพุตบางไฟล์ในกรณีที่งานนั้นต้องถูกใส่ในคิวเท่านั้น และด้วยค่าตัวแปรต่างๆ ดังตารางที่ 1 และจำนวนเครื่องที่รับงานไปประมวลผลที่มีอยู่ในระบบนั้น ระบบจะสามารถรองรับการทำงานได้ทันทุกงานโดยไม่มีงานใดที่ต้องอยู่ในคิวเลยดังรูปที่ 5 ดังนั้นจึงไม่มีอินพุตถูกส่งไปยังเซิร์ฟเวอร์แต่อย่างใด ด้วยเหตุนี้โครงสร้างแบบเพียร์-ทู-เพียร์จึงสามารถช่วยลดปัญหาคอขวดที่เซิร์ฟเวอร์ลงได้ และช่วยให้ระบบสามารถรองรับแอพพลิเคชันที่มีขนาดไฟล์อินพุตและเอาต์พุตใหญ่ๆ ได้มากกว่าโครงสร้างแบบไคลเอนต์/เซิร์ฟเวอร์

4.3 ผลกระทบของขนาดแบนด์วิดธ์ของเซิร์ฟเวอร์

ในส่วนนี้เราได้ทำการทดลองเพื่อศึกษาถึงผลกระทบของขนาดแบนด์วิดธ์ของเซิร์ฟเวอร์ โดยกำหนดให้ระบบมีเครื่องที่รับงานไปประมวลผลจำนวน 120 เครื่อง และแต่ละงานมีขนาดเอาต์พุตเท่ากับ 3 MB ซึ่งเป็นขนาดใหญ่ที่สุดที่ได้ทำการทดลอง แต่ละการทดลองจะกำหนดให้ขนาดแบนด์วิดธ์ของเซิร์ฟเวอร์มีค่าต่างๆ ที่มักใช้กันอยู่จริงในปัจจุบัน

ผลที่ได้ดังรูปที่ 10 พบว่าในสภาวะการทำงานที่มีค่าตัวแปรต่างๆ ตามตารางที่ 1 ถ้าจะทำให้โครงสร้างแบบไคลเอนต์/เซิร์ฟเวอร์สามารถทำงานได้โดยไม่มีปัญหาคอขวดเกิดขึ้นจะต้องใช้เซิร์ฟเวอร์ที่มีแบนด์วิดธ์ 1 Mbps ขึ้นไป แต่เมื่อใช้ระบบแบบเพียร์-ทู-เพียร์ เซิร์ฟเวอร์ไม่จำเป็นต้องมีขนาดของแบนด์วิดธ์ที่สูงก็สามารถทำงานได้โดยไม่มีปัญหาคอขวดเกิดขึ้น



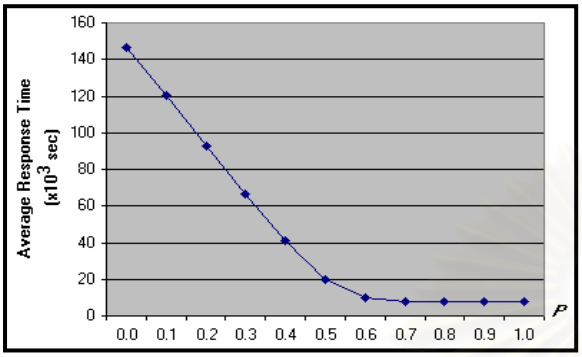
รูปที่ 10. ค่า *Response time* เฉลี่ยเมื่อขนาดแบนด์วิดธ์ของเซิร์ฟเวอร์มีค่าต่างๆ

4.4 ผลกระทบของความพลวัตในระบบเพียร์-ทู-เพียร์

ในระบบเพียร์-ทู-เพียร์นั้นแต่ละเครื่องในระบบสามารถเข้าและออกจากระบบได้ตลอดเวลา ดังนั้นบางครั้งเจ้าของงานอาจไม่ได้อยู่ในระบบขณะที่งานของตนประมวลผลเสร็จส่งผลให้เอาต์พุตไฟล์ไม่สามารถส่งคืนไปยังเจ้าของงานได้โดยตรง จำเป็นต้องส่งไปเก็บไว้ยังเครื่องเซิร์ฟเวอร์ก่อนเพื่อให้เครื่องที่รับงานไปประมวลผลนั้นสามารถรับงานใหม่ไปประมวลผลได้ทันที และเมื่อเจ้าของงานกลับเข้ามาในระบบก็จะไปเอาเอาต์พุตจากเครื่องเซิร์ฟเวอร์กลับไป

ในที่นี้จะกำหนดค่าความน่าจะเป็น P ซึ่งเป็นความน่าจะเป็นที่เจ้าของงานอยู่ในระบบขณะที่งานของตนเสร็จและสามารถรับเอาต์พุตไฟล์จากเครื่องที่รับงานไปประมวลผลได้โดยตรง ดังนั้นยิ่งค่า P มากก็จะทำให้ระบบมีการสื่อสารแบบเพียร์-ทู-เพียร์มากขึ้น และประสิทธิภาพของระบบก็จะดีขึ้นตามไปด้วย รูปที่ 11 แสดงค่า *Response time* เมื่องานมีขนาดเอาต์พุตไฟล์ 3 MB และระบบมีจำนวนเครื่องที่รับงานไปประมวลผล 120

เครื่อง จากรูปจะเห็นได้ว่าเมื่อค่า P ลดลงเพียงเล็กน้อยกลับทำให้ประสิทธิภาพของระบบดีขึ้นอย่างมาก และค่า P เพียง 0.6 ก็เพียงพอที่จะทำให้ระบบมีค่า *Response time* ใกล้เคียงกับ $Response\ time_{ideal}$



รูปที่ 11. ค่า *Response time* เฉลี่ยที่ค่า *P* ต่างๆ

5. สรุป

การนำการสื่อสารแบบเพียร์-ทู-เพียร์มาประยุกต์ใช้กับระบบการคำนวณแบบกระจาย โดยให้เครื่องเจ้าของงานและเครื่องที่รับงานไปประมวลผลมีการติดต่อสื่อสารและส่งข้อมูลถึงกันโดยตรงโดยไม่ผ่านเซิร์ฟเวอร์เป็นการใช้แบนด์วิดธ์ของเครื่องไคลเอนต์ซึ่งเพิ่มขึ้นตามจำนวนเครื่องในระบบ เป็นการลดปริมาณข้อมูลที่เซิร์ฟเวอร์ซึ่งก่อให้เกิดปัญหาคอขวดที่เซิร์ฟเวอร์ดังที่จะพบได้บ่อยในโครงสร้างแบบไคลเอนต์/เซิร์ฟเวอร์ ซึ่งช่วยให้ระบบสามารถรองรับแอพพลิเคชันที่มีอินพุตและเอาต์พุตขนาดใหญ่ได้ ไม่จำกัดอยู่เพียงแค่แอพพลิเคชันที่เน้นการคำนวณ (computing intensive) เท่านั้น อีกทั้งระบบยังสามารถรองรับจำนวนเครื่องที่รับงานไปประมวลผลได้เพิ่มมากขึ้นกว่าระบบแบบไคลเอนต์/เซิร์ฟเวอร์ โดยเซิร์ฟเวอร์ไม่จำเป็นต้องเพิ่มขนาดของแบนด์วิดธ์ ดังนั้นรูปแบบการคำนวณแบบกระจายเพียร์บนอินเตอร์เน็ตที่ใช้รูปแบบการสื่อสารแบบเพียร์-ทู-เพียร์จึงสามารถทำงานได้แม้กับระบบที่ใช้เซิร์ฟเวอร์ทั่วๆ ไปที่มีอยู่อย่างแพร่หลายในปัจจุบัน

6. เอกสารอ้างอิง

[1] SETI@Home, http://setiathome.ssl.berkely.edu/. (June 2004)

[2] FightAIDS@Home, http://fightaidsathome.org/. (June 2004)

[3] Folding@Home, http://www.stanford.edu/group/pandegroup/folding/ (June 2004)

[4] Genome@Home, http://www.stanford.edu/group/pandegroup/genome/ (June 2004)

[5] A. Chien, B. Calder, S. Elbert, K. Bhatia, “Entropia: architecture and performance of an enterprise desktop grid system,” The Journal of Parallel and Distributed Computing, Academic Press, Volume 63, Issue 5, USA, May 2003, pp. 597–610.

[6] B. O. Christiansen, P. Cappello, M. F. Ionescu, M. O. Neary, K. E. Schauser, and D. Wu, “Javelin: Internet-Based Parallel Computing Using Java”, In ACM Workshop on Java for Science and Engineering Computing, June 1997, pp. 1139-1160

[7] Frontier, http://www.parabon.com/. (June 2004)

[8] A. Page, T. Keane, R. Allen, T. J. Naughton, and J. Waldron, “Multi-tiered distributed computing platform,” Principles and Practice of Programming in Java, Ireland, June 2003, pp. 191-194.

[9] SETI@Home Bandwidth Problems, http://setiathome.ssl.berkeley.edu/bw.html (June 2004)

[10] J. Gray, “Distributed Computing Economics,” Microsoft Research Lab Technical Report, March 2003.

[11] A. Oram, Peer-to-Peer: Harnessing the Power of Disruptive Technologies, O’Reilly&Associates, 2001.

[12] P. Alto, D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu, “PEER-TO-PEER COMPUTING”, Hp Laboratories Technical Report, March 2002.

[13] R. Buyya and M. Murshed, “GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing,” The Journal of Concurrency and Computation: Practice and Experience (CCPE), Wiley Press, Volume 14, Issue 13-15, Nov.-Dec. 2002, pp. 1-32.

[14] SPEC Benchmark, http://www.specbench.org/cpu2000/ (March 2004)

[15] H.A. James, K.A. Hawick and P.D. Coddington, “Scheduling Independent Tasks on Metacomputing Systems,” Published in Proc. of Parallel and Distributed Computing Systems (PDCS'99), Fort Lauderdale, August 1999, pp. 156-162.

Using Peer-to-Peer Communication to Improve the Performance of Distributed Computing on the Internet

Kasame Tritrakan and Veera Muangsin
SPACE Research Unit, Department of Computer Engineering,
Chulalongkorn University, Bangkok, Thailand
kasame.t@student.chula.ac.th, veera.m@chula.ac.th

Abstract

This paper presents the design and evaluation of a distributed computing platform on the Internet that employs peer-to-peer communication for transferring data directly among participants without consuming scheduler's bandwidth. The aim is to reduce network congestion at the scheduler by utilizing available peer-to-peer bandwidth. The peer-to-peer model is evaluated and compared to the client/server model in different system settings. The impacts of various factors, including the dynamic nature of a peer-to-peer system, on the performance and scalability are studied. The experimental results show that when peer-to-peer communication is applied, even partially, the bandwidth demands on the scheduler significantly reduced. This results in improved system's performance and scalability, and better support for data-intensive applications.

1. Introduction

Generally, distributed computing systems on the Internet such as SETI@home [1], Entropia [2], and Parabon [3] use a client/server model having a centralized scheduler responsible for gathering and dispatching jobs and also transferring input and output data to and from participating computers. This computing model is still not widely used and there are few notable successful projects, if compared to the dedicated cluster model. One of the main problems is the bandwidth demands on the scheduler that increase as the system expands and eventually cause network congestion. The congestion prevents the scheduler from handling additional processing nodes and the performance significantly drops. Therefore, this approach for Internet computing obviously does not scale well.

Increasing the scheduler's bandwidth in order to solve the problem can be very expensive. Although the cost of computers is borne by the participants, the cost of network bandwidth is still on the scheduler side. This problem also occurs with the SETI@home project [4]. Therefore, this computing model is generally accepted as economically viable only for computational intensive applications [5].

Another solution is to use multiple schedulers [6], each of which is responsible for scheduling tasks for a fraction of processing nodes in a client/server fashion. Although this model is scalable by exploiting aggregated bandwidth of schedulers, it does not reduce the bandwidth demands being put on a scheduler for every added processing node, while it still has neglected the aggregated bandwidth of the processing nodes.

In this paper, we present the design and evaluation of a distributed computing platform on the Internet with peer-to-peer (P2P) communication. P2P communication [7] is applied to exploiting network bandwidth of participating computers, which increases along with the number of participants, for transferring input and output data. A scheduler may be used for indexing and handling queries. This technique can address the network congestion and scalability problems. Simulation study on the effects of various factors such as the utilization of P2P communication, number of processing nodes, data size, and scheduler's bandwidth are presented.

2. System models with P2P communication

In the P2P model, input and output files are transferred between job-submitters and processing nodes in a peer-to-peer fashion. The process is depicted in Figure 1. First, a submitter submits a *job advertisement (Job Ad)* to the scheduler. The Job Ad holds necessary information required by the scheduler such as the submitter's IP address, resource requirements, and the command to run the job. The scheduler also maintains a database of *resource advertisements (Res Ad)* that have been registered by processing nodes. A Res Ad holds the necessary information about the processing nodes including machine specifications and network information such as the IP address and bandwidth. If the scheduler can immediately find a machine that fits the job, the Res Ad will be returned to the submitter (2). The submitter uses

the information in the Res Ad to transfer the Job Ad and input file directly to the processing node (3). When the job is finished, the processing node returns the output to the submitter (4).

This model utilizes the processing nodes' bandwidth, instead of the scheduler's bandwidth, to transfer data to the submitters. Therefore, if the number of processing nodes increases, the available bandwidth also increases.

Figure 1. Peer-to-Peer model

However, the dynamic nature of peer-to-peer systems will play an important role in the utilization of the available peer-to-peer bandwidth.

Consider a situation when a job is submitted and there is no available processing node at the time, the input file must be transferred to the scheduler, so that the job owner can be offline and the input data can still be supplied by the scheduler once the job is assigned to a processing node. Also, when a processing node finishes the computation and finds that the job owner is offline. Then, the output data must be transferred to the scheduler until the submitter is online again and retrieves the output.

In order to study the effect of this dynamism, we introduce a probability factor, *P*, indicating the chance that a submitter is online at the time the processing node finishes the job and the processing node is able to send the output file to the submitter directly. If *P*=0.0, all output files must be sent to the scheduler and subsequently retrieved by the submitters, as in the client/server model. The higher value of *P* means the system utilizes more of the available peer-to-peer bandwidth and becomes a hybrid between peer-to-peer and client/server models. The input file can also be transferred in a peer-to-peer or client/server fashion, depending on the availability of processing nodes. To focus on the effects of *P*, we assume applications with small input and large output so that the transfer modes of input files have only slight impact on the scheduler's bandwidth.

3. Simulation model

For performance evaluation of both models we used a discrete event simulator called GridSim Toolkit [8]. We have modified the simulator to model network link sharing more accurately. We do not model a network topology but assume that each entity is connected to the system with a constant bandwidth.

Table 1 shows the values of parameters used in this study. As we aim to model a moderate distributed computing system, parameter values are set according to typical servers and PCs. The processing nodes have varied processing capacity based on typical home PCs (Pentium III 1 GHz, 256 MB RAM, according to SPEC CPU (INT) 2000 benchmark rating in [9]). It connects to the system with a 56 Kbps modem. A submitter has the same configuration as a processing node.

The job model is guided by Gray [5], who has concluded that on-demand computing is only economical for very cpu-intensive applications with at least 100,000 instructions per byte of network traffic. Therefore, we use POV-RAY [10], a ray-tracer program, as the reference application. For simplicity, we assume that a processing node executes only one job at a time and, once started, the job is executed on that machine until it is finished. Also, the jobs are independent (no communication between jobs). Memory and disk requirements are not considered in this study.

The scheduling algorithm is FCFS (first-come-first-serve). When a processing node joins the system, a 5 KB Res Ad is sent to the scheduler. A job Ad is also 5 KB in size. The network bandwidth of the scheduler, S_{BW}, is 64 KByte/sec (512 Kbps). We also evaluate the effect of scheduler's bandwidth in a following section.

Table1: Simulation parameters used in the study

Number of jobs (J)	5000
Job inter-arrival time (λ)	exp(60) sec.
Number of instructions per job	exp(1.62M)MI
Input file size (D_{in})	0.1 MB
Output file size (D_{out})	3 MB
Number of processing nodes	80-200
Processing nodes performance	250 – 350 MIPS
Bandwidth of scheduler (S_{BW})	512 Kbps
Bandwidth of processing nodes and submitters (U_{BW})	56 Kbps

4. Performance modeling and analysis

In this section, we evaluate the effects of varying different factors that affect the system performance, compared to the client/server model.

4.1. Effect of data size

Data being transferred include Job Ad, Res Ad, input, and output file. However, Job Ads, Res Ads, and input files in this study are very small compared to the output

files. Therefore, in this study, we will evaluate only the effect of the size of output files, D_{out}.

The limit on the size of output files is considered to prevent congestion. In the client/server model, the size of output file is limited by the scheduler's bandwidth. In the peer-to-peer model, it is also limited by the aggregated bandwidth of processing nodes and the value of P.

P = 1.0

If P=1.0, all output files are sent directly to the submitter. The scheduler will handle only Job Ad, Res Ad, and input, which are small. Thus, congestion will occur due to long transfer of output files by the processing nodes. A processing node will receive a new job at the rate of $\lambda_R = R \bullet \lambda$; where R is the number of processing nodes. Hence, congestion will not occur if a job is completed before λ_R:

$$T_{comp} + T_{comm} \leq \lambda_R \qquad (1)$$

Since $T_{comm} = D_{out} / U_{BW}$, congestion will not occur if

$$D_{out} \leq (\lambda_R - T_{comp}) \bullet U_{BW} \qquad (2)$$

In Figure 2, the limitation of output file size is fairly close the value given in equation (2). The greater number of processing nodes means the increase in the overall system bandwidth and larger size of output files that the system can handle.

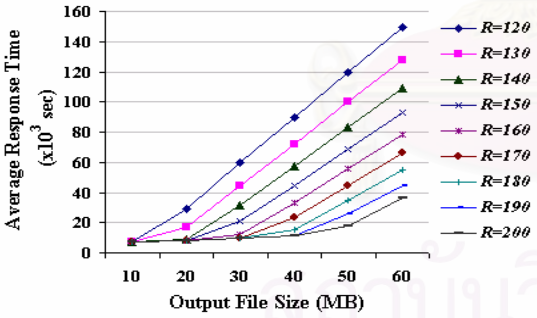


Figure 2. ART with *P*=1.0 in variation of *R* and D_{out}

0.0 ≤P <1.0

When P is less than 1.0, the scheduler becomes the bottleneck and its bandwidth, S_{BW}, is used for transferring output data, D_{out}, twice for each of $(1\text{-}P)/\lambda$ jobs. Therefore, congestion will not occur if

$$S_{BW} \geq 2\left(\frac{1\text{-}P}{\lambda}\right) D_{out} \qquad (3)$$

Or

$$D_{out} \leq \frac{\lambda \bullet S_{BW}}{2(1\text{-}P)} \qquad (4)$$

Figure 3 shows the results from simulating the system with 150 processing nodes and various values of P and D_{out} to see how large the output file the system can handle. The greater value of P means more processing nodes' bandwidth the system can utilize, and larger output file the system can handle.

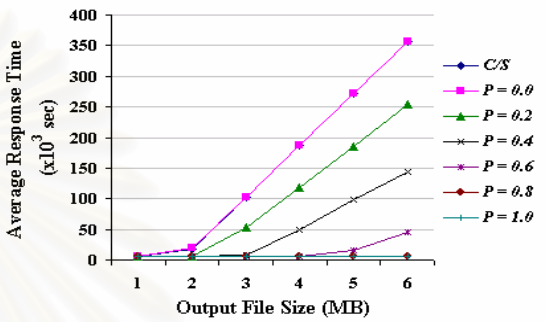


Figure 3. ART in variation of *P* and D_{out}

4.2. Effect of number of processing nodes

The number of processing nodes that the system can handle indicates the scalability of the system. In the client/server model, although the increasing number of processing nodes reduces computing time in the system, the increasing amount of data the scheduler has to handle may also cause network congestion. This can dramatically increase communication time and leads to longer response times (Figure 4). Consequently, the number of processing nodes the system can handle is limited by network bandwidth of the scheduler. This also occurs in the peer-to-peer model with small values of P.

When P=0.4, some interesting results can be observed. The scheduler's data load can be considered into three ranges (Figure 5). In the first range (80–120 processing nodes), the data load goes up because the computation bottleneck (not enough processing nodes) forces a lot of input files to be transferred to the scheduler like the client/server model. Moreover, the large amount of output files causes the network congestion problem. In the second range (120–180 processing nodes), the number of available processing nodes is quite enough, therefore more input files can be sent to the processing nodes directly and network congestion problem is alleviated. This leads to the rapid decrease in the response time (Figure 4) and queue length (Figure 6). In the third range (180-200 processing nodes), the data load slightly increases due to the increasing number of output files that are sent back to the scheduler.

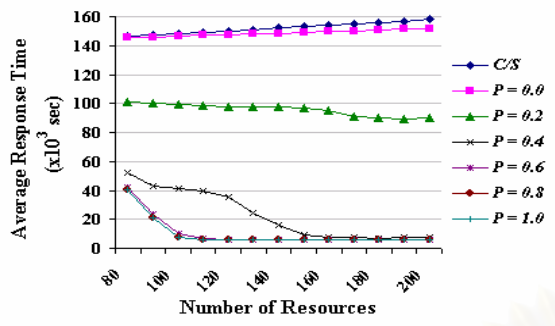


Figure 4. ART in variation of *R*

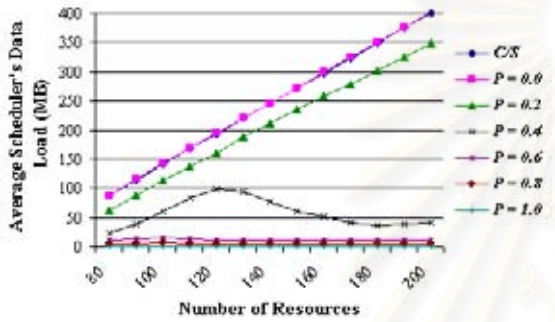


Figure 5. Average data load at the scheduler at an instant of time in variation of *R*

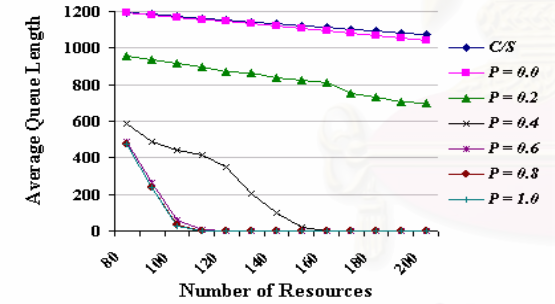


Figure 6. Average queue length in variation of *R*

When $P \geq 0.6$, network congestion is eliminated. This shows that when the peer-to-peer communication is applied, the system has more scalability than the client/server model.

4.3. Scheduler's connection bandwidth

Figure 7 show the average response times for various scheduler's bandwidths. Not surprisingly, when the scheduler's bandwidth decreases, the performance of client/server model degraded sharply. On the other hand, in the peer-to-peer model, this problem reduced accordingly to the value of *P*. This demonstrates that the scheduler in the peer-to-peer model requires less connection bandwidth than in the client/server model.

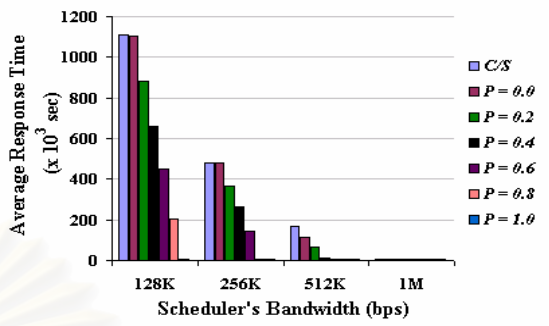


Figure 7. ART in variation of scheduler's connection bandwidth with 150 processing nodes

5. Conclusion

This paper presents the design and evaluation of a distributed computing platform that exploits peer-to-peer communication. The probability that the submitter and processing node is concurrently online to transfer data is an important factor to the performance. The higher probability indicates that the system has more peer-to-peer characteristics. The experimental results show that when peer-to-peer communication is applied, even partially, the performance is significantly improved. The system becomes more scalable and is able to handle more processing nodes and larger data size. In addition, less bandwidth is required from the scheduler.

6. References

[1] D. P. Anderson, et al., "SETI@home: an experiment in public-resource computing", *Communications of the ACM*, New York, NY, 2002, pp. 56-61.
[2] http://www.entropia.com/, September 2004.
[3] http://www.parabon.com/, September 2004.
[4] http://setiathome.ssl.berkeley.edu/bw.html, June 2004.
[5] Jim Gray, "Distributed Computing Economics", *Microsoft Research Lab Technical Report*, 2003.
[6] A. Page, T. Keane, R. Allen, T. J. Naughton, and J. Waldron, "Multi-tiered distributed computing platform", *2nd International Conference on the Principles and Practice of Programming in Java*, Kilkenny City, Ireland, 2003, 191-194.
[7] Andy Oram, *Peer-to-Peer: harnessing the power of disruptive technologies*, O'Reilly&Associates, 2001.
[8] R. Buyya and M. Murshed, "GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing", *The Journal of Concurrency and Computation: Practice and Experience (CCPE), 14*(13-15), 2002, 1-32.
[9] http://www.specbench.org/cpu2000/, March 2004.
[10] http://www.povray.org/, September 2004.

ประวัติผู้เขียนวิทยานิพนธ์

นายเกษม ตริตระการ เกิดวันที่ 18 กันยายน พ.ศ. 2524 ในจังหวัดสุราษฎร์ธานี เข้ารับการศึกษาในระดับปริญญาวิศวกรรมศาสตรบัณฑิต สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ในปี พ.ศ. 2541 และสำเร็จการศึกษาในปี พ.ศ. 2545

จากนั้นผู้เขียนได้เข้าศึกษาต่อในระดับปริญญาวิศวกรรมศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปี พ.ศ. 2545